Q.1 In a given application, we have information about the ages of a set of 12 people. Their values are 12, 30, 24, 10, 10, 23, 43, 67, 79, 34, 56, 51.

In [11]:
```
x<-c(12, 30, 24, 10, 10, 23, 43, 67, 79, 34, 56, 51)
```

In [3]:
```
print(x)
```

```
 [1] 12 30 24 10 10 23 43 67 79 34 56 51
```

In [5]:
```
x_mean=mean(x)
```

In [7]:
```
print(x_mean)
```

```
[1] 36.58333
```

What is the median of these ages? Explain.

In [8]:
```
xmedian=median(x)
```

In [9]:
```
print(xmedian)
```

```
[1] 32
```

In [12]:
```
mode<-function(x)
    ux<-unique(x)
    ux[which.max(tabulate((match(x, ux))))]
```

```
Error in eval(expr, envir, enclos): object 'ux' not found
Traceback:
```

In [22]:
```
normalization<-function(x,nmin,nmax)
    x=((x-min(x))/(max(x)-min(x))*(nmax-nmin))+nmin
    return(x)
```

12 30 24 10 10 23 43 67 79 34 56 51

In [23]:
```
normalization(x,10,12)
```

In [24]:
```
print(normalization(x,10,12))
```

```
 [1] 10.05797 10.57971 10.40580 10.00000 10.00000 10.37681 10.95652 11.65217
 [9] 12.00000 10.69565 11.33333 11.18841
```

In [27]:
```
standardization<-function(x)
    x=((x-min(x))/(max(x)-min(x)))
    return(x)
```

12 30 24 10 10 23 43 67 79 34 56 51

In [28]:
```
print(standardization(x))
```

```
[1] 0.02898551 0.28985507 0.20289855 0.00000000 0.00000000 0.18840580
[7] 0.47826087 0.82608696 1.00000000 0.34782609 0.66666667 0.59420290
```

In [ ]:
```

```

Consider the following data table:

Inst/Var	V1	V2
I1	1.5	1.7
I2	2	1.9
I3	1.6	1.8
I4	1.2	1.5

Given a new observation (1.4,1.6), which two observations in the table are nearer the new data point, using the Euclidean distance?

In [2]:
```
df=data.frame(instv=c("i1","i2","i3","i4"),
v1=c(1.5,2.0,1.6,1.2),
v2=c(1.7,1.9,1.8,1.5))
```

In [5]:
```
l=c(1,2,3,4)
```

In [9]:
```
Fun_euclid_distance<-function(x,y){
p=100
index=100
for(i in l){
y=sqrt((df[i,2]-x)^2+(df[i,3]-y)^2)
if(p>y){p=y
       index=i}
return (i)}
}
```

In [10]:
```
x<-Fun_euclid_distance(1.4,1.6)
```

In [11]:
```
print(x)
```

```
[1] 1
```

In [ ]:
```

```

Q3

Start R using the shortcut created earlier. Download the file

http://stats.ma.ic.ac.uk/das01/public_html/RCourse/hills.txt (http://stats.ma.ic.ac.uk/das01/public_html/RCourse/hills.txt)

Read the file, assigning the result to the object hills. Examine the object. Note column and row names. Construct a scatterplot matrix for dist vs. time. Make the columns of the hills object available by name. Compute a linear regression of time against distance.

```
hills <- read.table("hills.txt")
```

```
hills
```

```
attach(hills)
```

```
plot(dist,time)
```

```
lm(time~dist)
```

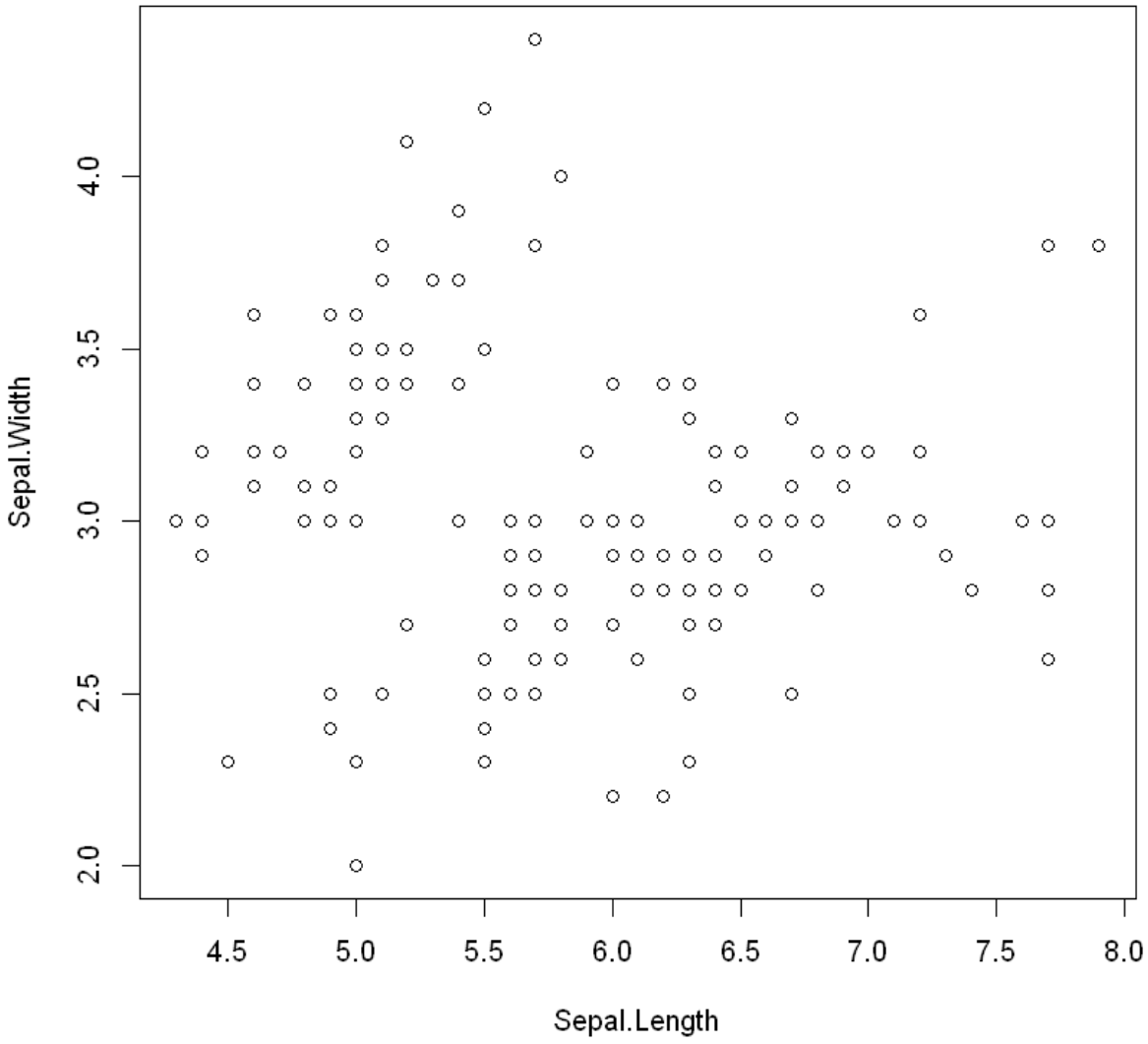
In [3]:
```
data <- iris[,]

attach(data)

plot(Sepal.Length,Sepal.Width)


mod <- lm(Sepal.Width~Sepal.Length)

detach(data)
```



In [ ]:
```

```

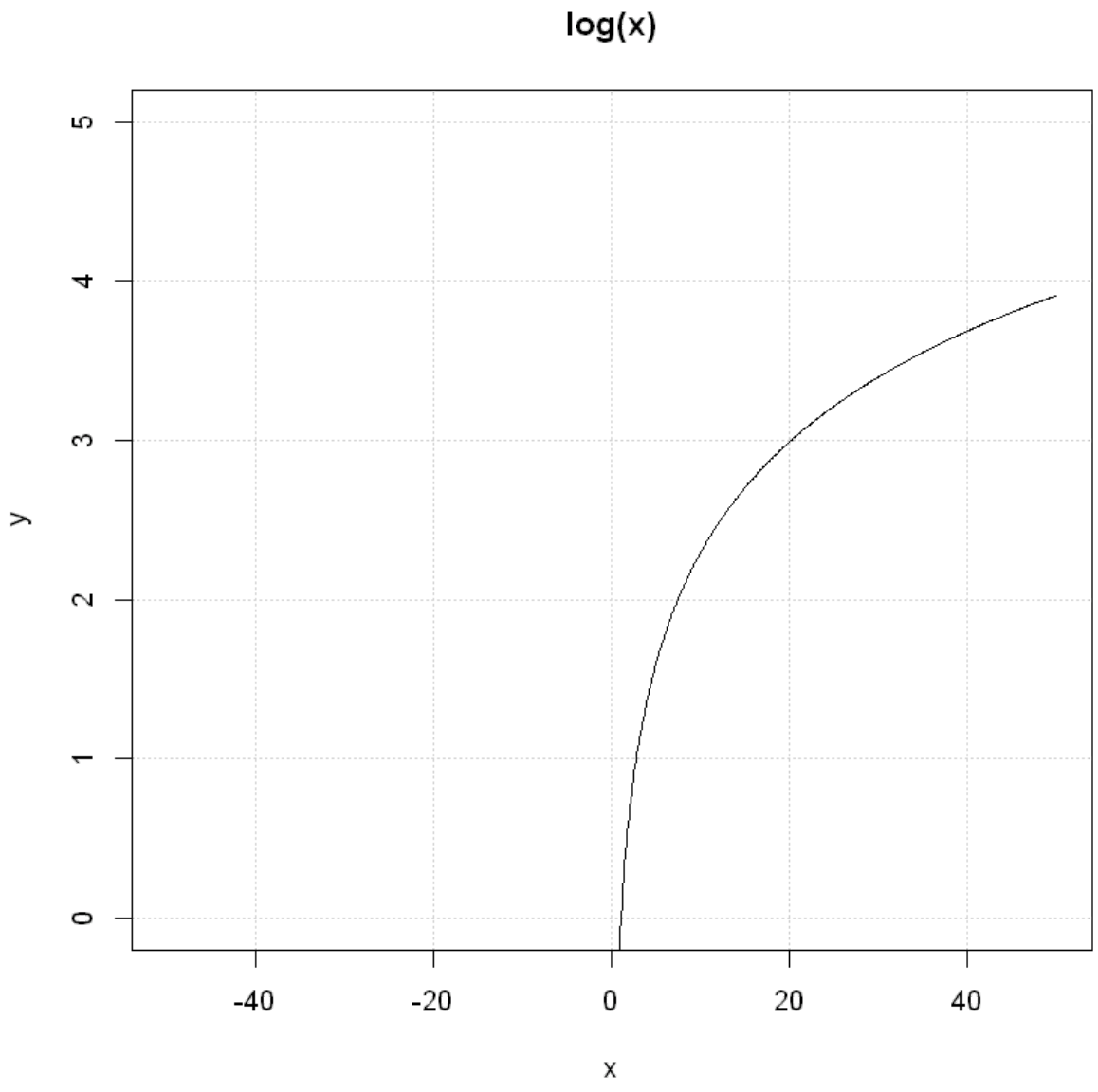
Q. 4

Generate a regular grid between -50 and 50. Construct separate plots of log(x), exp(x), sin(x), sin(2x), sin(x)/cos(x). Examine the cumulative sum of the final function. Experiment with the argument type olf the plot function.

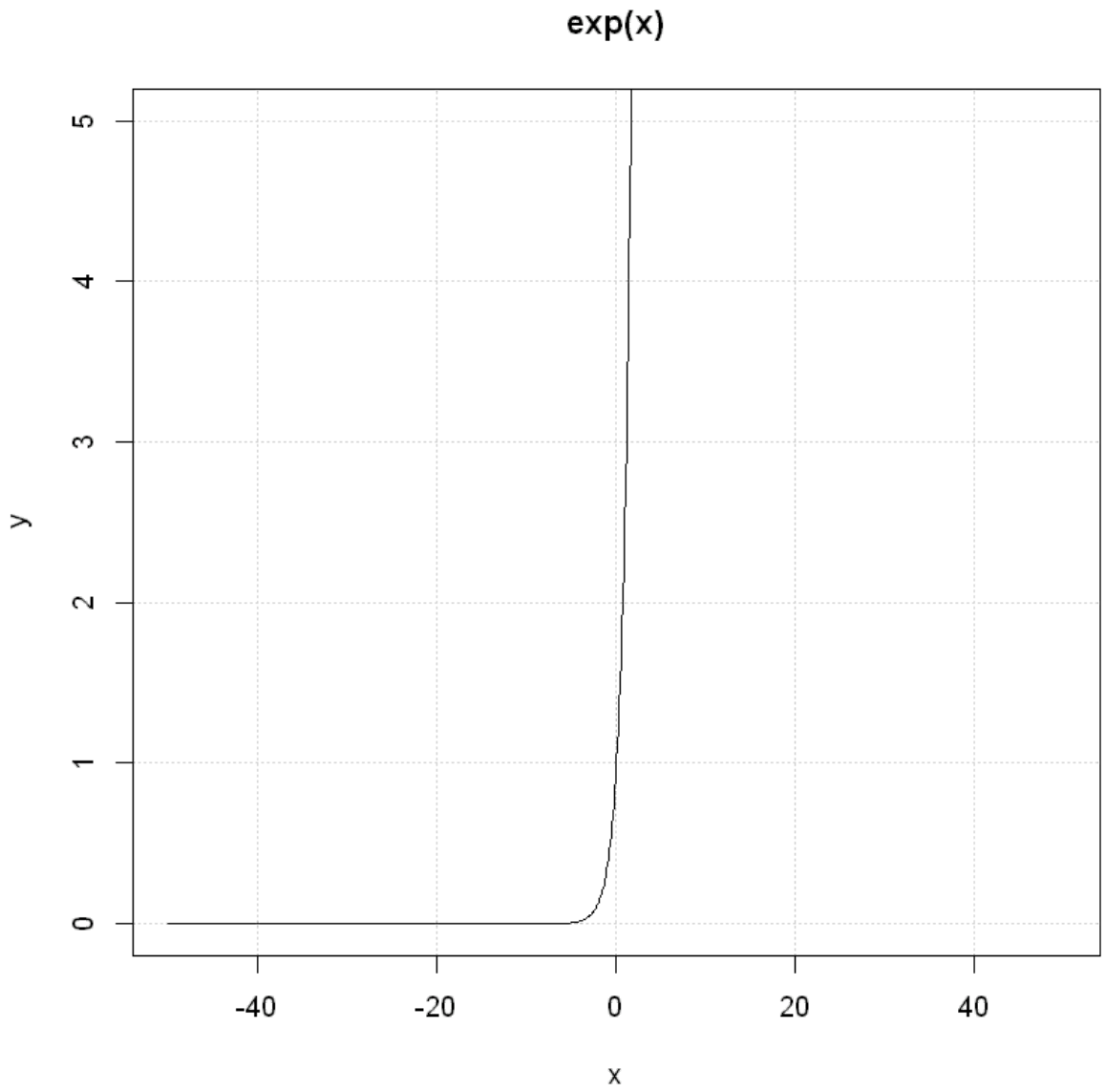
In [3]:
```
x <- seq(-50,50,by=0.1)
```

In [4]:
```
y=log(x)
x11()
plot(x,y,"l",main="log(x)",ylim=c(0,5),panel.first=grid())
```

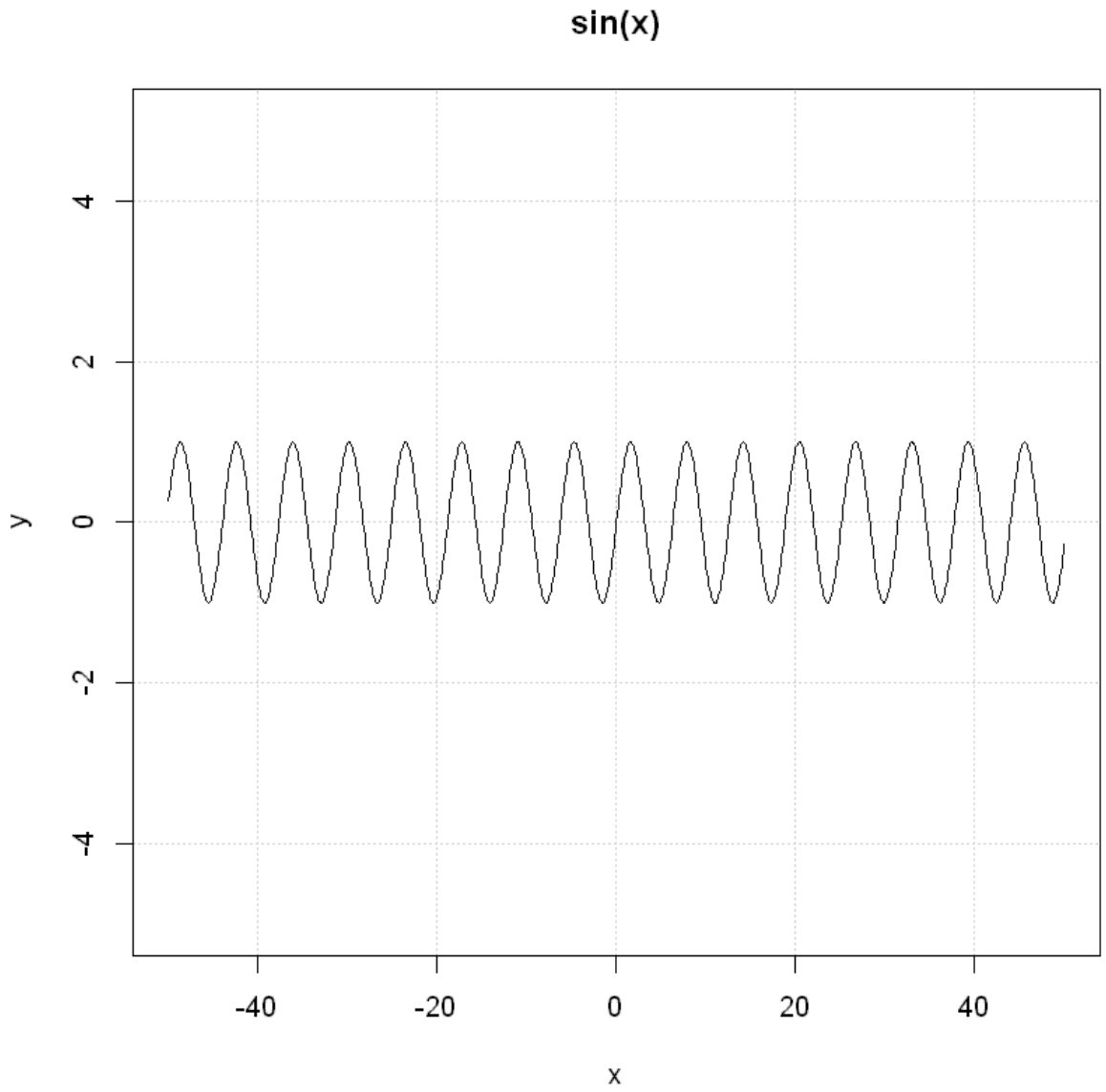
```
Warning message in log(x):
"NaNs produced"
```



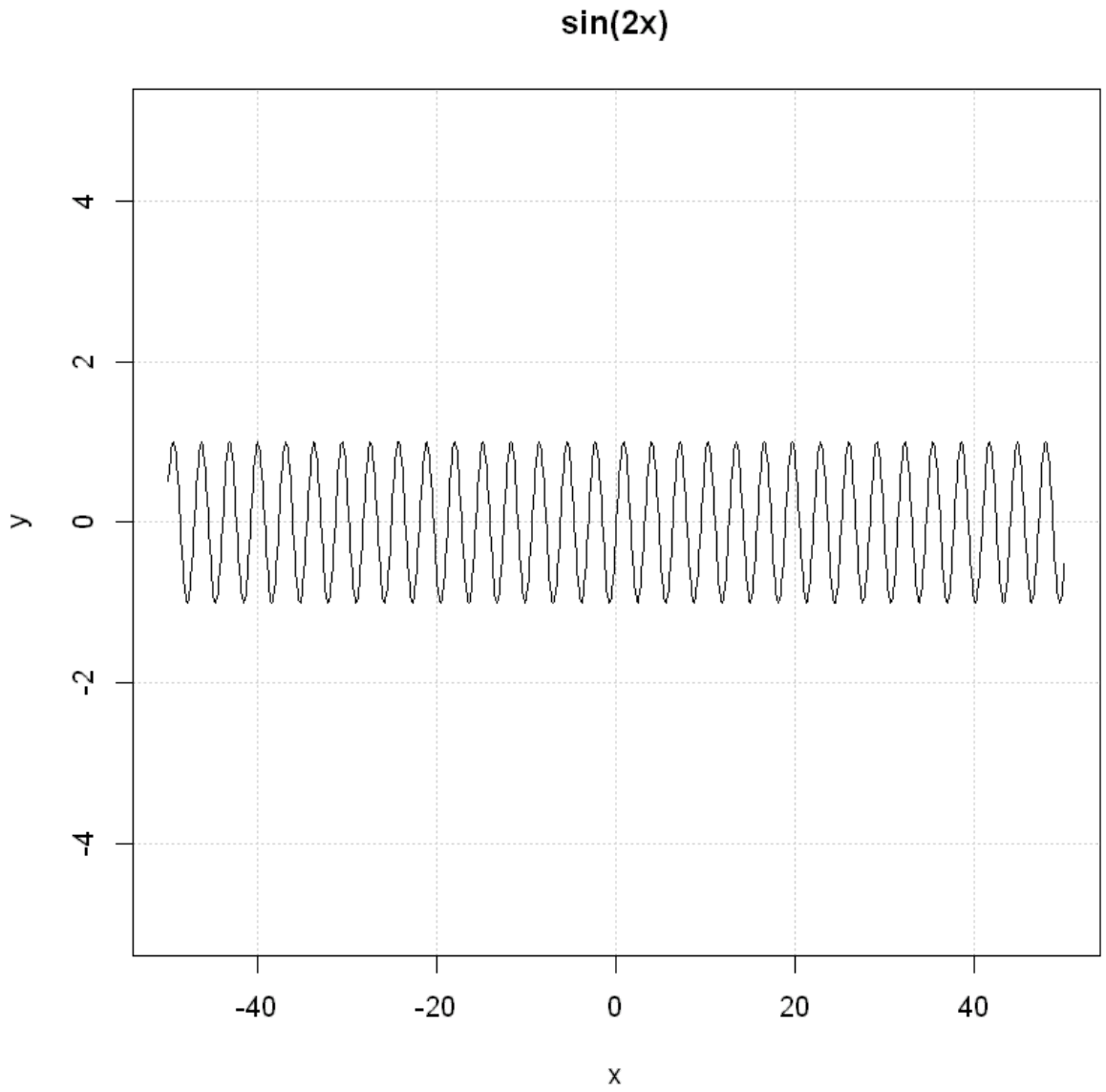
In [5]:
```
y=exp(x)
x11()
plot(x,y,"l",main="exp(x)",ylim=c(0,5),panel.first=grid())
```



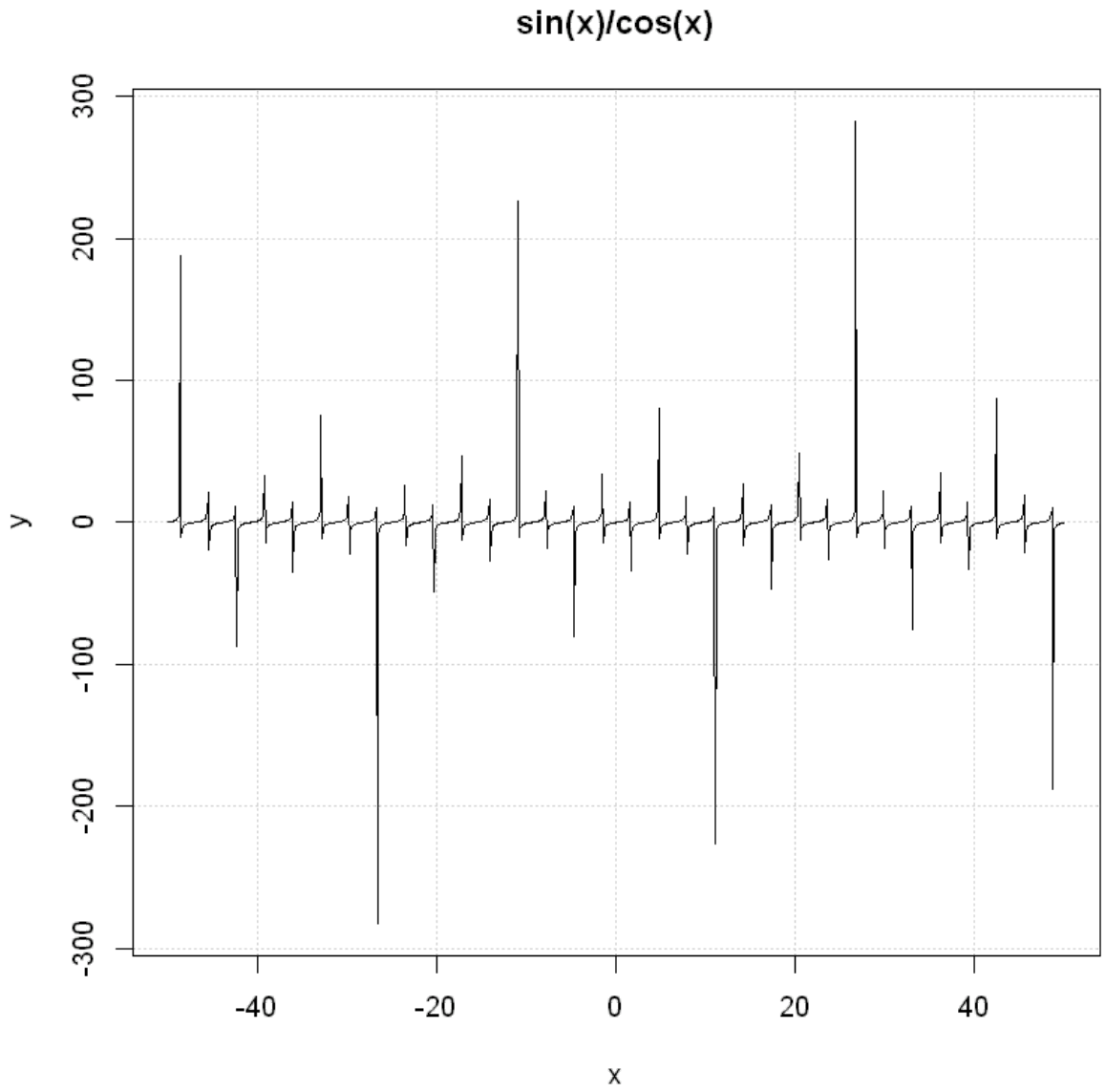
In [6]:
```
y=sin(x)
x11()
plot(x,y,"l",main="sin(x)",ylim=c(-5,5),panel.first=grid())
```



In [7]:
```
y=sin(2*x)
x11()
plot(x,y,"l",main="sin(2x)",ylim=c(-5,5),panel.first=grid())
```



In [8]:
```
y=sin(x)/cos(x)
x11()
plot(x,y,"l",main="sin(x)/cos(x)",panel.first=grid())
```



In [ ]:
```
```

Q.5 Consider the vector 1:K, where K is a positive integer. Write an R command that determines how many elements in the vector are exactly divisible by 3.x`

In [15]:
```
generateVector<-function(k)
{
    v<-c(1:k)
    return(v)
}
```

In [23]:
```
divisibleBy3<-function(v)
{
    count=0
    for(i in v)
    {
         if(i%%3==0)
         {
            count=count+1
         }
    }
    return(count)
}
```

In [24]:
```
v<-generateVector(10)
```

In [26]:
```
print(v)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

In [25]:
```
counter=divisibleBy3(v)
```

In [27]:
```
print(counter)
```

```
[1] 3
```

Q.6 Construct 3x3 matrix and compute row and column sums of matrix.

In [1]:
```
v<-c(1,2,3,4,5,6,7,8,9)
```

In [2]:
```
m<-matrix(v,nrow=3,ncol=3,byrow=TRUE)
```

In [3]:
```
print(m)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

In [22]:
```
sumCol<-function(m)
{
    for(i in c(1:3))
    {
        sum=0
        for(j in c(1:3))
        {
            sum=sum+m[i,j]
        }
        print(sum)
    }
}
```

In [23]:
```
sumCol(m)
```

```
[1] 6
[1] 15
[1] 24
```

In [24]:
```
sumRow<-function(m)
{
    for(i in c(1:3))
    {
        sum=0
        for(j in c(1:3))
        {
            sum=sum+m[j,i]
        }
        print(sum)
    }
}
```

In [25]:
```
sumRow(m)
```

```
[1] 12
[1] 15
[1] 18
```

Q.7

Generate a matrix of size n × p. Use the function as.data.frame to coerce the matrix to a datab frame. Which object requires more storage space?

In [1]:
```
mat <- matrix(1:35,nrow=7,ncol=5,byrow=TRUE)
object.size(mat)
df<-as.data.frame(mat)
object.size(df)
```

```
344 bytes
```

```
1264 bytes
```

In [ ]:
```

```

Q.8 Create a for loop that, given a numeric vector, prints out one number per line, with its square and cube alongside.

(b) Show how to use a while loop to achieve the same result.

In [9]:
```
funForLoop<-function(k)
{
    l<-c(1:k)
    for(i in l)
    {
        sqar=i^2
        print(paste(i,i^2,sep=" square is "))
    }
}
```

In [10]:
```
funForLoop(10)
```

```
[1] "1 square is 1"
[1] "2 square is 4"
[1] "3 square is 9"
[1] "4 square is 16"
[1] "5 square is 25"
[1] "6 square is 36"
[1] "7 square is 49"
[1] "8 square is 64"
[1] "9 square is 81"
[1] "10 square is 100"
```

In [11]:
```
funWhileLoop<-function(k)
{
    i=1
    while(i<k)
    {
        sqar=i^2
        print(paste(i,i^2,sep=" square is "))
        i=i+1
    }
}
```

In [12]:
```
funWhileLoop(10)
```

```
[1] "1 square is 1"
[1] "2 square is 4"
[1] "3 square is 9"
[1] "4 square is 16"
[1] "5 square is 25"
[1] "6 square is 36"
[1] "7 square is 49"
[1] "8 square is 64"
[1] "9 square is 81"
```

Q.9 Write an R expression to determine if two sets, A and B, represented as integer vectors are disjoint. If they are disjoint, display elements of set A otherwise display elements of set B. (Examine the help for functions print and cat).

In [9]:
```
a<-c(1,3,5,8)
b<-c(2,4,6,8)
```

In [10]:
```
isDisjoint=TRUE
for(i in a)
{
    for(j in b)
    {
        if(i==j)
        {
            isDisjoint=FALSE
            break
        }
    }
}
if(isDisjoint)
{
    print(a)
}else
{
    print(b)
}
```

```
[1] 2 4 6 8
```

Q.10 Write R codes that takes the coefficients of a quadratic equation, and outputs an appropriate message for the cases of

(i) two distinct roots (b2 − 4ac > 0)

(ii) coincident roots (b2 = 4ac)

(iii). complex roots (b2 < 4ac).

In [1]:
```
rootCheck<-function(a,b,c)
{
    if(b^2-4*a*c>0){
        print("two distinct roots")
    }else if(b^2==4*a*c){
        print("coincident roots")
    }else if(b^2<4*a*c){
        print("complex roots")
    }
}
```

In [3]:
```
rootCheck(2,5,7)
```

```
[1] "complex roots"
```

Q.11

Let vector y be the logarithm of a random sample from a standard normal distribution, N(0, 1). Use the ifelse function to replace missing values with the value 9999.

In [16]:
```
stan = function(x){
    x = ((x-min(x))/(max(x)-min(x)))
    return (x)
}
```

In [17]:
```
d <- log(stan(rnorm(50)))
```

In [18]:
```
print(d)
```

```
 [1] -0.57393433 -0.44350358 -0.49541098 -1.21916746 -0.44480665 -1.12727646
 [7] -0.48945384 -0.73069106 -1.03159077 -0.77583500 -1.04417909 -0.90132704
[13] -0.51877544 -2.19069899 -0.67550288 -0.80863635 -2.23230980 -0.35495936
[19] -2.27515266 -1.16608142 -2.69145880 -1.13638771 -2.71289972 -1.05599010
[25] -0.68998564 -2.14728284 -1.25514864 -0.54898748 -1.12928594 -1.16894261
[31]  0.00000000 -0.75625411 -0.42474324        -Inf -0.71214323 -0.36130909
[37] -0.63175341 -0.53971186 -1.66125080 -0.52676620 -1.07486634 -0.27187386
[43] -1.42845184 -0.05412197 -1.34737758 -2.63181905 -0.55280308 -0.33754787
[49] -0.61621088 -0.71022129
```

In [19]:
```
for( i in 1: length(d)){
    if(is.infinite(d[i])){
        d[i] = 9999
    }
}
```

In [20]:
```
print(d)
```

```
 [1]   -0.57393433   -0.44350358   -0.49541098   -1.21916746   -0.44480665
 [6]   -1.12727646   -0.48945384   -0.73069106   -1.03159077   -0.77583500
[11]   -1.04417909   -0.90132704   -0.51877544   -2.19069899   -0.67550288
[16]   -0.80863635   -2.23230980   -0.35495936   -2.27515266   -1.16608142
[21]   -2.69145880   -1.13638771   -2.71289972   -1.05599010   -0.68998564
[26]   -2.14728284   -1.25514864   -0.54898748   -1.12928594   -1.16894261
[31]    0.00000000   -0.75625411   -0.42474324 9999.00000000   -0.71214323
[36]   -0.36130909   -0.63175341   -0.53971186   -1.66125080   -0.52676620
[41]   -1.07486634   -0.27187386   -1.42845184   -0.05412197   -1.34737758
[46]   -2.63181905   -0.55280308   -0.33754787   -0.61621088   -0.71022129
```

In [ ]:

Q. 12

Write a loop structure to scan through an integer vector to determine the index of the maximum value. The loop should terminate as soon as the index is obtained. (Don’t worry about ties). Of course, this is not a good way to do this! Examine the help for the rank, sort and order

In [1]:
```
v<-c(2,10,20,20,40,40,30,30,40,10)
```

In [2]:
```
m=max(v)
```

In [3]:
```
print(m)
```

```
[1] 40
```

In [4]:
```
l<-c(1:10)
for(i in l){
    if(v[i]==m){
        print(i)
        break
    }
}
```

```
[1] 5
```

Q. 13

Suppose

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$$

(a) Check that A3 = 0 where 0 is a 3x 3 matrix with every entry equal to 0.

(b) Replace the third column of A by the sum of the second and third columns.

In [11]:
```
v<-c(1,1,3,5,2,6,-2,-1,-3)
```

In [12]:
```
a=matrix(v,nrow=3,ncol=3,byrow=TRUE)
```

In [13]:
```
print(a)
```

```
     [,1] [,2] [,3]
[1,]    1    1    3
[2,]    5    2    6
[3,]   -2   -1   -3
```

In [14]:
```
a2=m*m
```

In [15]:
```
a3=a2*m
```

In [16]:
```
print(a3)
```

```
     [,1] [,2] [,3]
[1,]    1    1   64
[2,]  125    8  512
[3,]   -8   -1  -64
```

In [21]:
```
l<-c(1:3)
k<-c(2:3)
```

In [23]:
```
isZero=TRUE
for(i in 1){
    for(j in 1){
        if(a3[i,j]!=0){
            isZero=FALSE
            break
        }
    }
}
if(!isZero){
print("A3 is not zero")
}
```

```
[1] "A3 is not zero"
```

In [19]:
```
for(i in 1){
        a[i,3]=a[i,2]+a[i,3]
}
```

In [20]:
```
print(a)
```

```
     [,1] [,2] [,3]
[1,]    1    1    4
[2,]    5    2    8
[3,]   -2   -1   -4
```

In [ ]:

Q.14

Create the following matrix B with 15 rows:Calculate the 3 x 3 matrix BTB. (Look at the help for crossprod.)

$$\mathbf{B} = \begin{bmatrix} 10 & -10 & 10 \\ 10 & -10 & 10 \\ \dots & \dots & \dots \\ 10 & -10 & 10 \end{bmatrix}$$

In [1]:
```
v<-c(10,-10,10)
```

In [2]:
```
m=matrix(v,nrow=15,ncol=3,byrow=TRUE)
```

In [3]:
```
print(m)
```

```
      [,1] [,2] [,3]
 [1,]   10  -10   10
 [2,]   10  -10   10
 [3,]   10  -10   10
 [4,]   10  -10   10
 [5,]   10  -10   10
 [6,]   10  -10   10
 [7,]   10  -10   10
 [8,]   10  -10   10
 [9,]   10  -10   10
[10,]   10  -10   10
[11,]   10  -10   10
[12,]   10  -10   10
[13,]   10  -10   10
[14,]   10  -10   10
[15,]   10  -10   10
```

In [9]:
```
help(crossprod)
```

In [14]:
```
crossprod(m,m)
```

```
 1500  -1500   1500
-1500   1500  -1500
 1500  -1500   1500
```

In [ ]:
```

```

Q. 15

Create a 6 x 10 matrix of random integers chosen from 1, 2,: : : , 10 by executing the following two lines of code:

set.seed(75)

aMat <- matrix(sample(10, size=60, replace=T), nr=6)

(a) Find the number of entries in each row which are greater than 4.

(b) Which rows contain exactly two occurrences of the number seven?

(c) Find those pairs of columns whose total (over both columns) is greater than 75. The answer should be a matrix with two columns; so, for example, the row (1; 2) in the output matrix means that the sum of columns 1 and 2 in the original matrix is greater than 75. Repeating a column is permitted; so, for example, the final output matrix could contain the rows (1; 2), (2; 1) and (2;2).What if repetitions are not permitted? Then, only (1; 2) from (1; 2), (2; 1) and (2; 2) would be permitted.

In [22]:
```
l<-c(1:60)
v<-c()
```

In [35]:
```
set.seed(75)
for(i in l){
    v[i]=ceiling(runif(1,1,10))
}
```

In [36]:
```
print(v)
```

```
 [1]  3  2  8  5  2  3  6  9 10  4  9  6  7  8  8  2  2  8  7  8  5  2  9  5
2
[26]  3 10  4  9  7  4  7  6  4  8 10  4 10  5  9  2  4  7 10  8  7  3  7  2
6
[51]  5  4  7 10  4  3  4  3  7  2
```

In [41]:
```
m<-matrix(v,nrow=6,ncol=10,byrow=TRUE)
```

In [42]:
```
print(m)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    3    2    8    5    2    3    6    9   10     4
[2,]    9    6    7    8    8    2    2    8    7     8
[3,]    5    2    9    5    2    3   10    4    9     7
[4,]    4    7    6    4    8   10    4   10    5     9
[5,]    2    4    7   10    8    7    3    7    2     6
[6,]    5    4    7   10    4    3    4    3    7     2
```

In [43]:
```
aMat <- matrix( sample(10, size=60, replace=T), nr=6)
```

In [44]:
```
print(aMat)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    6    7    9    8    7   10    8    3    7     7
[2,]    6    2    5    5   10    9    5    2    8     5
[3,]    3    7    9    6    5    2    4    8    4     7
[4,]    3    7    6    9    6    8   10    8    1     7
[5,]    4    7    5    4   10    3    7    6    1     8
[6,]    8    1    2    8    5    3   10    1    3     1
```

In [45]:
```
l<-c(1:10)
k<-c(1:6)
```

In [47]:
```
for(i in k){
    count=0
    for(j in l){
        if(aMat[i,j]>4){
            count=count+1
        }
    }
    print(paste("row",i,"have total",count,"element which is greter than 4",se
p=" "))
}
```

```
[1] "row 1 have total 9 element which is greter than 4"
[1] "row 2 have total 8 element which is greter than 4"
[1] "row 3 have total 6 element which is greter than 4"
[1] "row 4 have total 8 element which is greter than 4"
[1] "row 5 have total 6 element which is greter than 4"
[1] "row 6 have total 4 element which is greter than 4"
```

In [48]:
```
for(i in k){
    count=0
    isSeven=FALSE
    for(j in l){
        if(aMat[i,j]==7){
            count=count+1
        }
    }
    if(count==2){
        print(paste("row",i,"contain exectly two time seven",sep=" "))
    }
}
```

```
[1] "row 3 contain exectly two time seven"
[1] "row 4 contain exectly two time seven"
[1] "row 5 contain exectly two time seven"
```

In [49]:
```
v2<-c()
```

In [77]:
```
q=0
for(p in l){
    sum=0
    for(i in l){
        sum=0
        for(j in k){
            sum=sum+aMat[j,p]+aMat[j,i]
        }
        if(sum>75){
            v2[q]=p
            q=q+1
            v2[q]=i
        }
    }
}
```

In [78]:
```
print(v2)
```

```
 [1]  3  3  4  4  4  4  5  5  5  5  5  5  6  6  7  7  7  7  7  7 10 10  7  9
9
[26]  9  9 10 10 10 10 10 10 10 10  9  5  5  5  5  5  5  6  6  6  6  6  6  6
6
[51]  6  7  7  7  7  7  7  7  7  7  8  8  8  8  8  8  8  8  8  9  9  9  9  9
9
[76]  9  9  9 10 10 10 10 10 10 10 10 10 10 10
```

In [79]:
```
n<-matrix(v2,nrow=44,ncol=2,byrow=TRUE)
```

In [ ]:
```
print(n)
```

Q. 16

Write functions tmpFn1 and tmpFn2 such that if xVec is the vector (x1; x2; : : : ; xn), then tmpFn1(xVec) returns the vector and tmpFn2(xVec) returns the vector

$$\left(x_1, \frac{x_2^2}{2}, \ldots, \frac{x_n^n}{n}\right)$$

(b) Now write a function tmpFn3 which takes 2 arguments x and n where x is a single number and n is a strictly positive integer. The function should return the value of

$$1 + \frac{x}{1} + \frac{x^2}{2} + \frac{x^3}{3} + \cdots + \frac{x^n}{n}$$

In [37]:
```
v<-c(10:1)
l<-c(1:10)
v2<-c()
v3<-c()
v4<-c()
sum=0
```

In [8]:
```
print(v)
```

```
 [1] 10  9  8  7  6  5  4  3  2  1
```

In [14]:
```
tmpFn1<-function(v){
    for(i in l){
        v2[i]=v[i]^2
    }
    return(v2)
}
```

In [15]:
```
v2=tmpFn1(v)
```

In [16]:
```
print(v2)
```

```
 [1] 100  81  64  49  36  25  16   9   4   1
```

In [17]:
```
tmpFn2<-function(v){
    for(i in l){
        v3[i]=v[i]^2/i
    }
    return(v3)
}
```

In [18]:
```
v3=tmpFn2(v)
```

In [19]:
```
print(v3)
```

```
 [1] 100.0000000  40.5000000  21.3333333  12.2500000   7.2000000   4.1666667
 [7]   2.2857143   1.1250000   0.4444444   0.1000000
```

part (b)

In [47]:
```
tmpFn3<-function(x,n){
    l<-c(2:n)
    sum=1
    v4[1]=1
    for(i in l){
        v4[i]=((x^i)/i)
        sum=sum+v4[i]
    }
    print(v4)
    return (sum)
}
```

In [48]:
```
sum=tmpFn3(2,5)
```

```
[1] 1.000000 2.000000 2.666667 4.000000 6.400000
```

In [49]:
```
print(sum)
```

```
[1] 16.06667
```

In [ ]:

Q.17

Examine the built in ChickWeight data (the help gives background about the data). The function split will prove useful to do the following (as will a script)

(a) Construct a plot of weight against time for chick number 34.

(b) For chicks in diet group 4, display box plots for each time point.

(c) Compute the mean weight for chicks in group 4, for each time point. Plot this mean value

against time.

(d) Repeat the previous computation for group 2. Add the mean for group 2 to the existing plot.

(e) Add a legend and a title.

(f) Copy and paste the graph into Word.

Note that some of these steps will be easier once we have some more programming expertise.

In [2]:
```
head(ChickWeight)
```

weight	Time	Chick	Diet
42	0	1	1
51	2	1	1
59	4	1	1
64	6	1	1
76	8	1	1
93	10	1	1

In [14]:
```
df<-matrix(ChickWeight)
```

In [15]:
```
barplot(df,xlab="wight",ylab="time",main="plot")
```

```
Error in apply(height, 2L, cumsum): (list) object cannot be coerced to type
'double'
Traceback:

1. barplot(df, xlab = "wight", ylab = "time", main = "plot")
2. barplot.default(df, xlab = "wight", ylab = "time", main = "plot")
3. rbind(rectbase, apply(height, 2L, cumsum))
4. apply(height, 2L, cumsum)
```