Introductory Programming in R

By Asef Nazari

asef.nazari@monash.edu

Faculty of IT

Monash university

7. Plotting

7.1 Building graphics from data

Dataframes are a powerful tool to organizing and visualizing data. However, it is hard to interpret large data sets, no matter how organized they are. Sometimes it is much easier to interpret graphs than numbers.

Some of the key base plotting functions

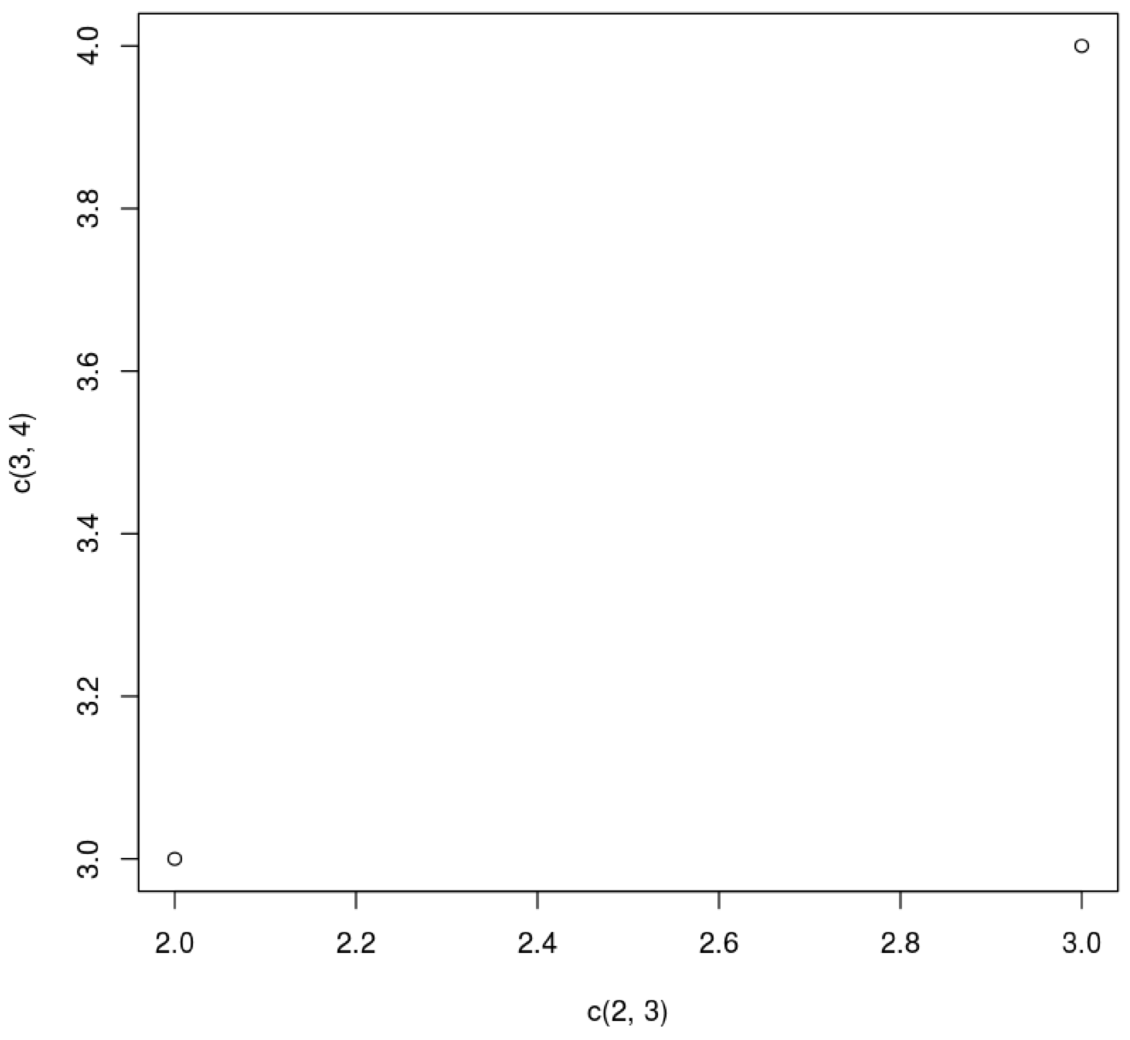
- **plot()**: plots based on the object type of the imput
- **lines()**: add lines to the plot (just connect dots)
- **points()**: add points
- **text()**: add text labels to a plot using x,y coordinates
- **title()**: add titles
- **mtext()**:add arbitrary text to the margin
- **axis()**: adding axis ticks/labels

some important parameters

- **pch**: the plotting symbol (plotting character)
- **lty**: the line type; solid, dashed, ...
- **lwd**: the line width; lwd=2
- **col**: color; col="red"
- **xlab**: x-axis label; xlab="units"
- **ylab**: y-axix label; ylab="price"

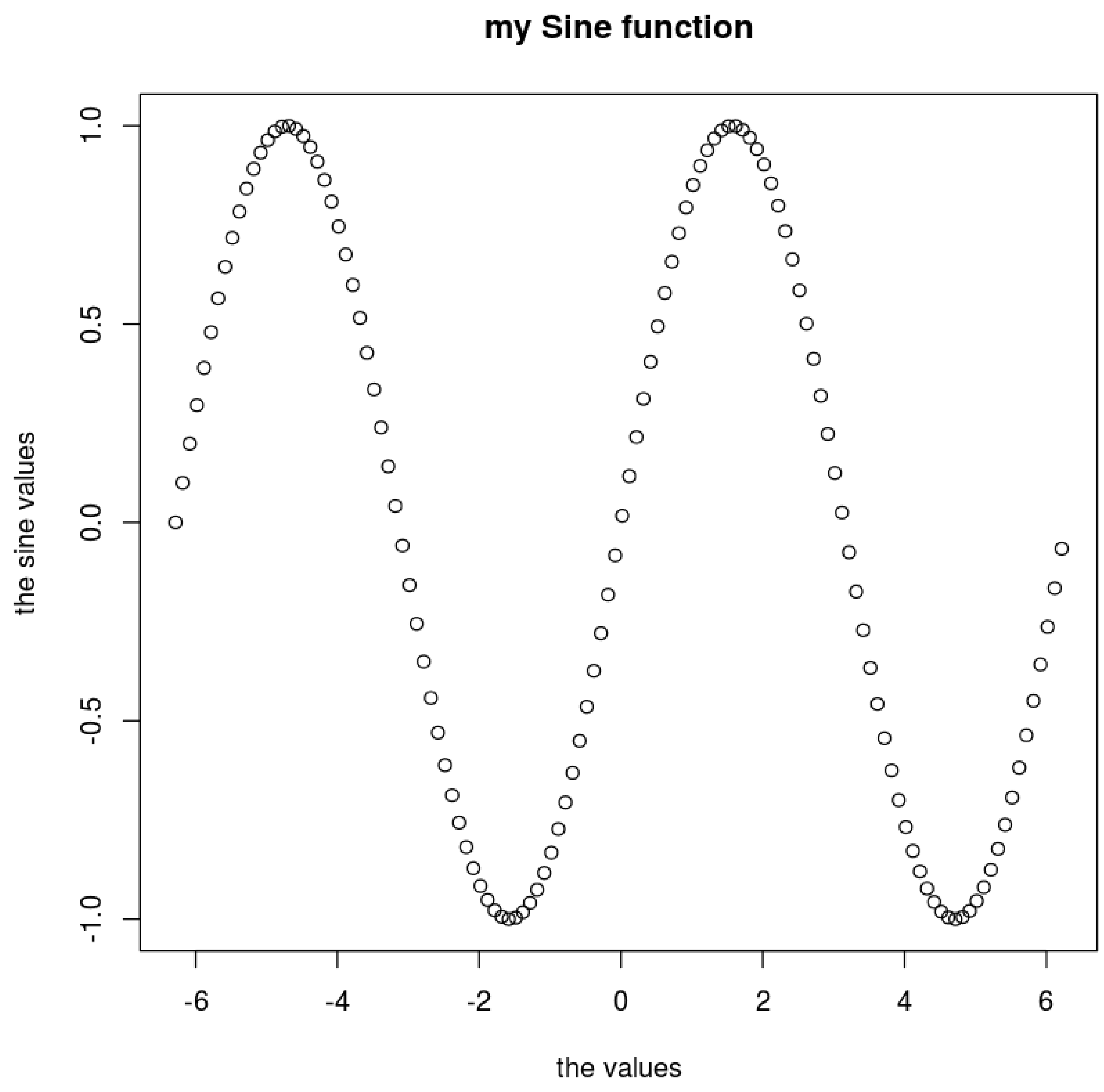
In [85]:

```
plot(c(2,3), c(3,4))
```



In [86]:

```
x <- seq(-2*pi,2*pi,0.1)
plot(x, sin(x),
    main="my Sine function",
    xlab="the values",
    ylab="the sine values")
```

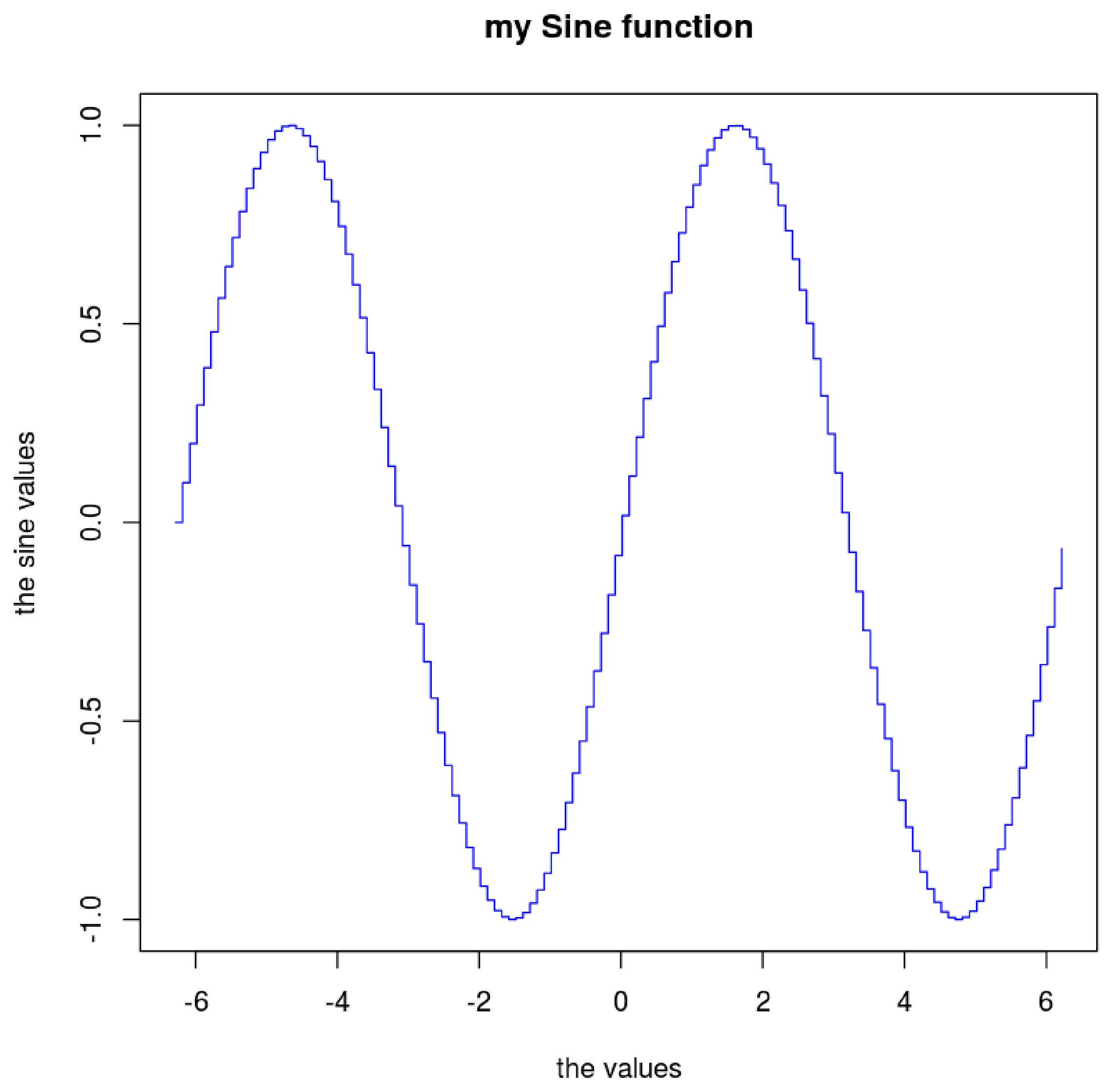


Different values for type

- "p" - points (defult)
- "l" - lines
- "b" - both points and lines
- "c" - empty points joined by lines
- "o" - overplotted points and lines
- "s" and "S" - stair steps
- "h" - histogram-like vertical lines
- "n" - does not produce any points or lines

In [87]:

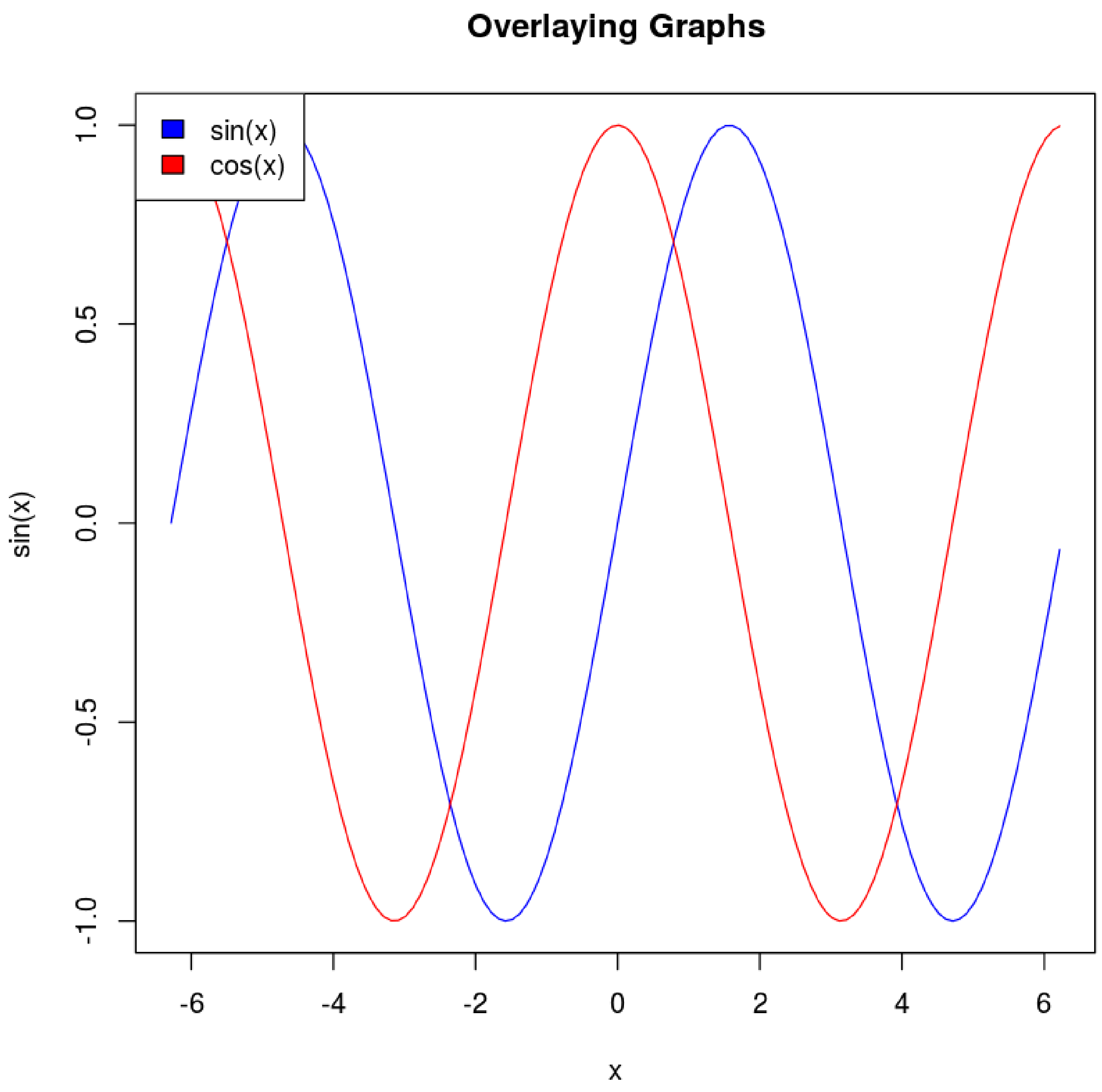
```
x <- seq(-2*pi,2*pi,0.1)
plot(x, sin(x),
    main="my Sine function",
    xlab="the values",
    ylab="the sine values",
    type="s",
    col="blue")
```



Calling plot() multiple times will replace the current graph with the previous one. However, sometimes we wish to overlay the plots in order to compare the results. This is done with the functions **lines()** and **points()** to add lines and points respectively, to the existing plot.

In [88]:

```
plot(x, sin(x),
 main="Overlaying Graphs",
 type="l",
 col="blue")

lines(x,cos(x), col="red")

legend("topleft",
      c("sin(x)","cos(x)"),
      fill=c("blue","red")
)
```



By setting some graphical parameters we can put several graphs in a single plot. The **par()** is used for global graphics parameters. R programming has a lot of graphical parameters which control the way our graphs are displayed.

- before doing any change record the standard default parameters oldpar <- par()
- **las**: the rientation of axix labels on the plot
- **bg**: the background color
- **mar**: the margin size
- **oma**: the outer margin size

- **mfrow**: number of plots per row (plots are filled row-wise)
- **mfcol**: number of plots per row (plots are filled column-wise)
- at the end, par(oldpar) and neglect the warning messages.

In [89]:

```
#par() # to see all the parameters
```

In [90]:

```
par("mar") # to see the margins, bottom, left, top, right
```

```
5.1   4.1   4.1   2.1
```

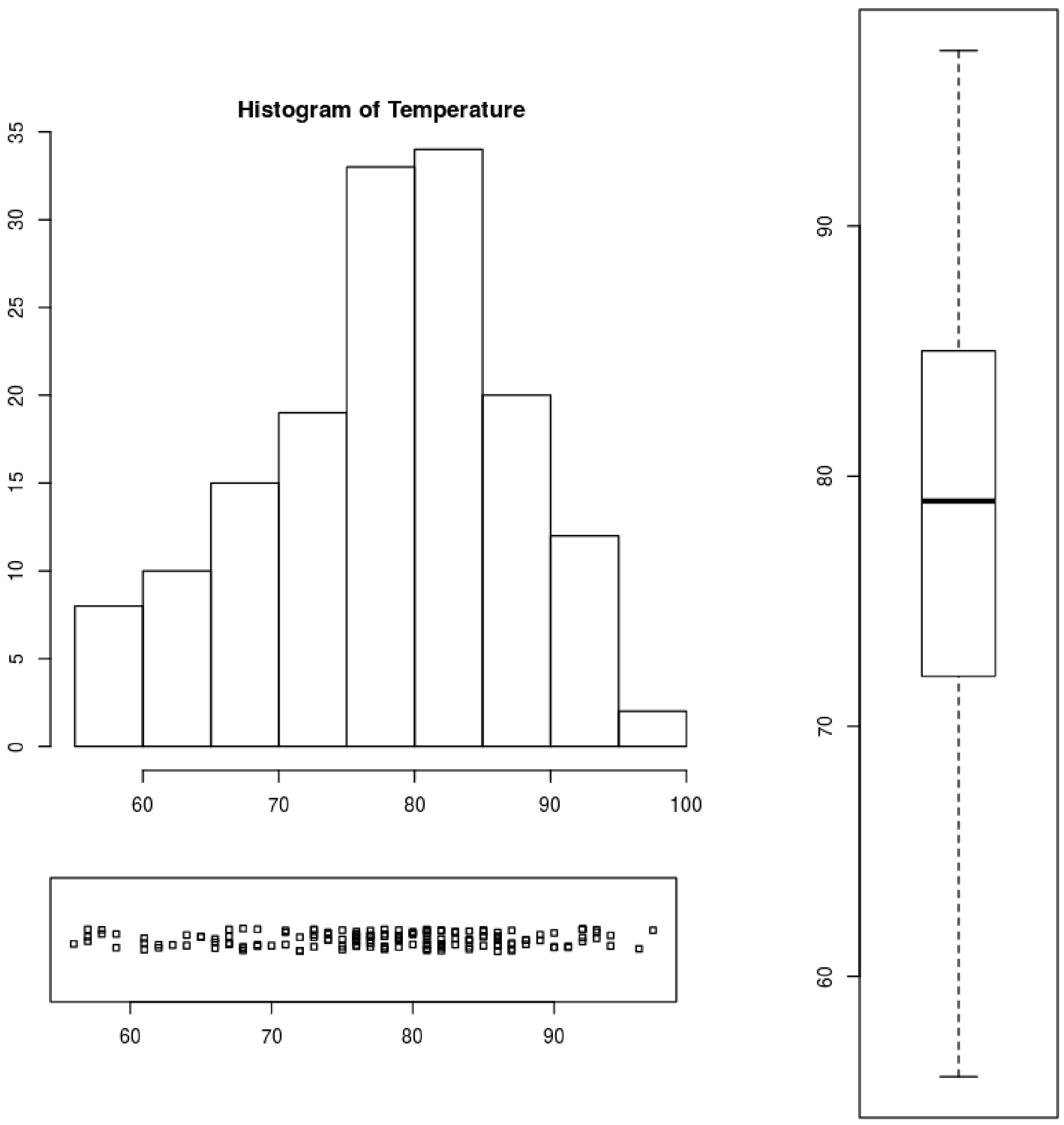
In [91]:

```
# par(mfrow=c(1,2))    # set the plotting area into a 1*2 array
```

In [92]:

```
oldpar <- par()
# make labels and margins smaller
par(cex=0.7, mai=c(0.1,0.1,0.2,0.1))

Temperature <- airquality$Temp

# define area for the histogram
par(fig=c(0.1,0.7,0.3,0.9))
hist(Temperature)

# define area for the boxplot
par(fig=c(0.8,1,0,1), new=TRUE)
boxplot(Temperature)

# define area for the stripchart
par(fig=c(0.1,0.67,0.1,0.25), new=TRUE)
stripchart(Temperature, method="jitter")
par(oldpar)
```

```
Warning message:
In par(oldpar): graphical parameter "cin" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "cra" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "csi" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "cxy" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "din" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "page" cannot be set
```



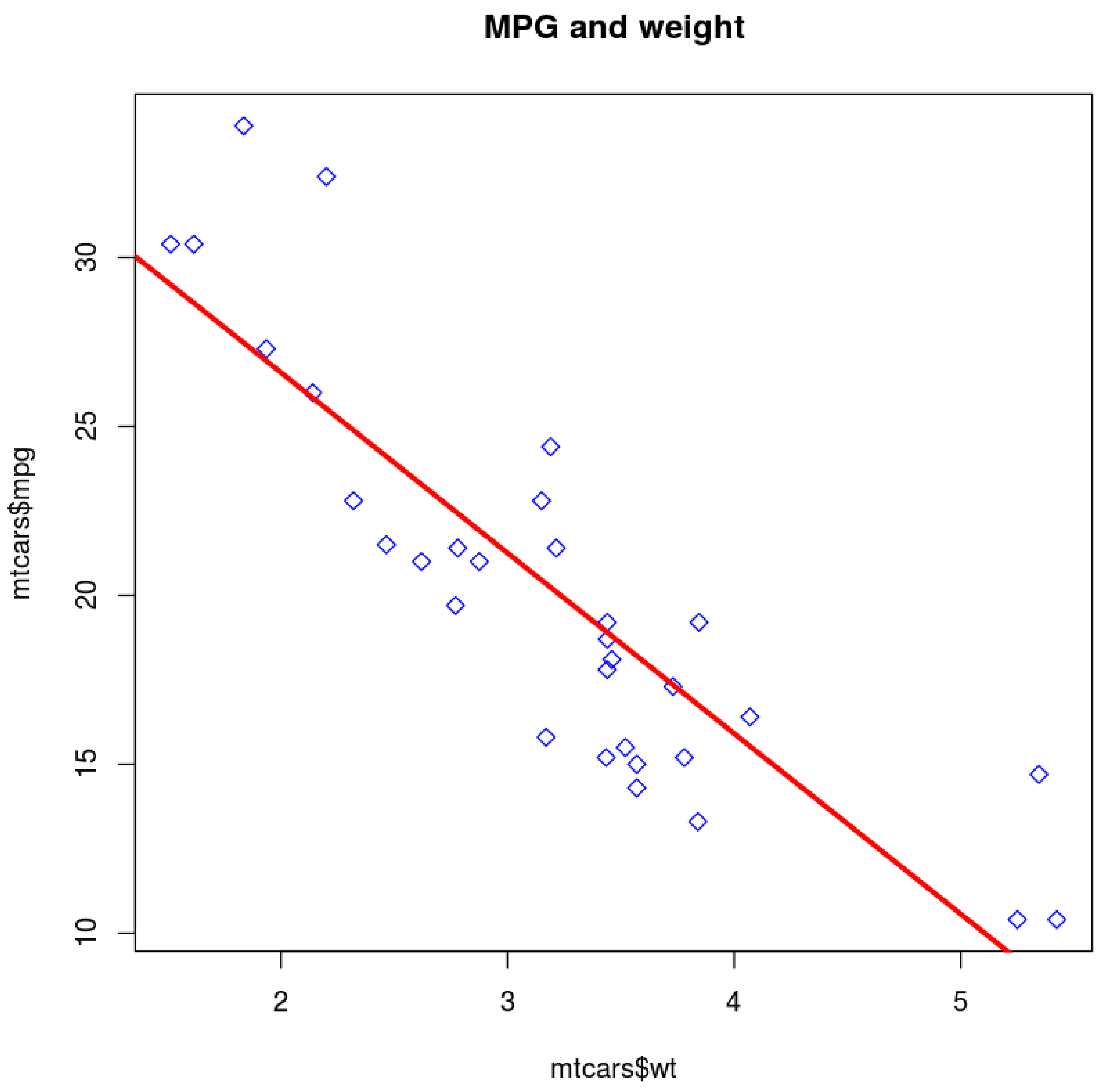
In [93]:

```
str(mtcars)
```

```
'data.frame':	32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

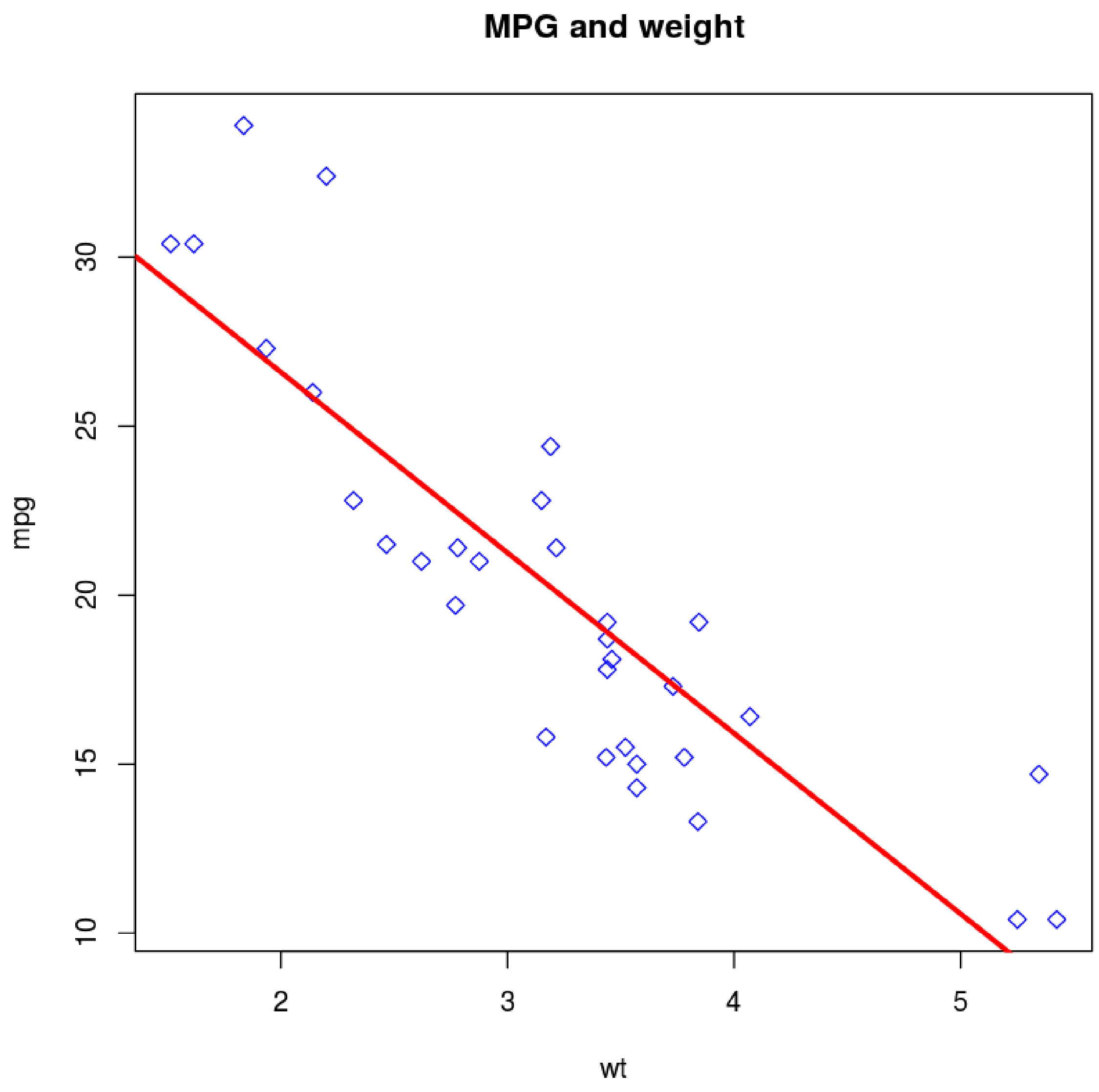
In [94]:

```
plot(mtcars$wt, mtcars$mpg, main="MPG and weight", col="blue", pch=5)
abline(lm(mtcars$mpg~mtcars$wt), col="red", lwd=3)
```



In [95]:

```
plot(mtcars$wt, mtcars$mpg,
     main="MPG and weight",
     col="blue",
     pch=5,
    xlab="wt",
    ylab="mpg")
abline(lm(mtcars$mpg~mtcars$wt), col="red", lwd=3)
```

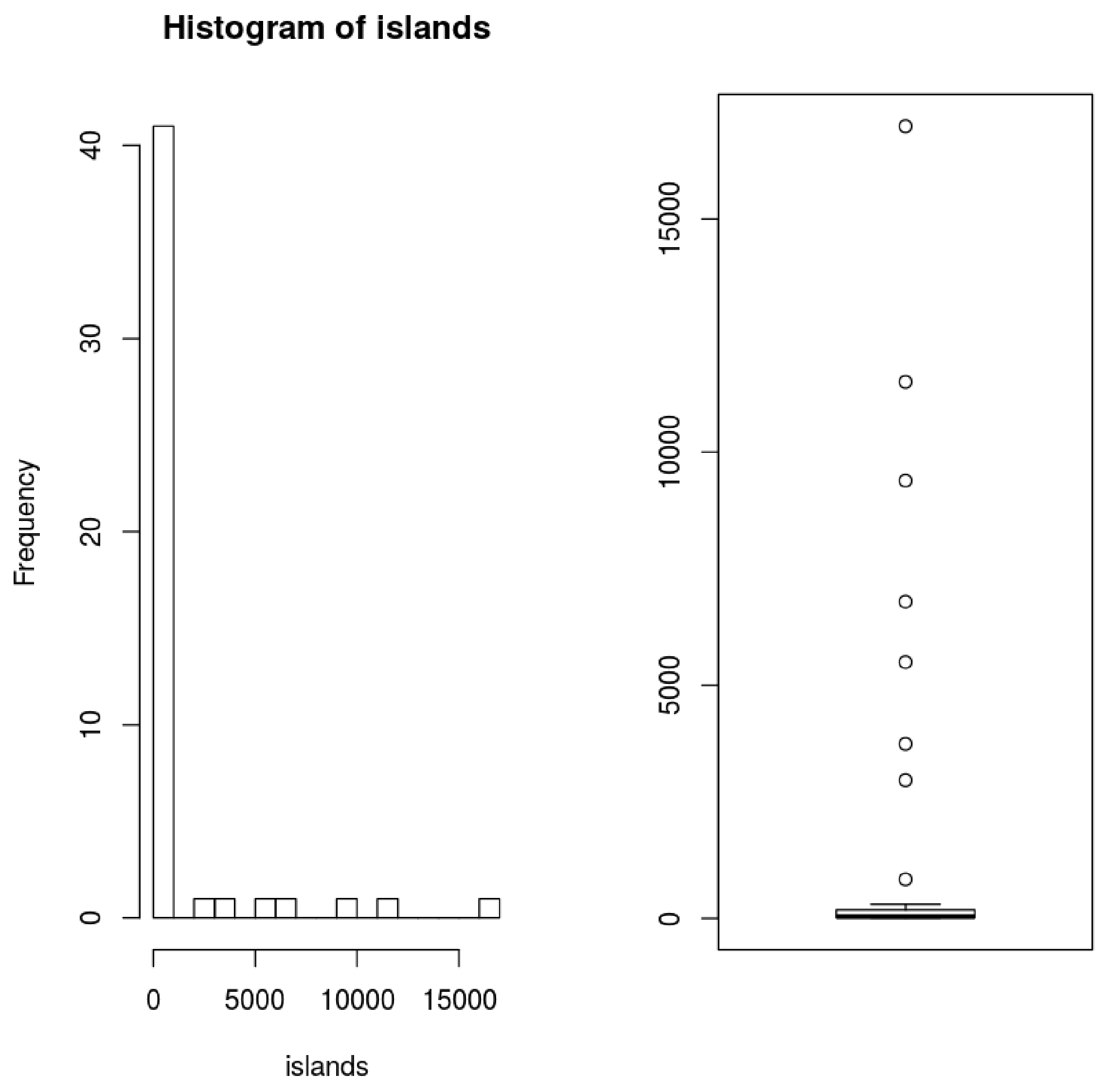


Type *Markdown* and LaTeX: α^2

In [96]:

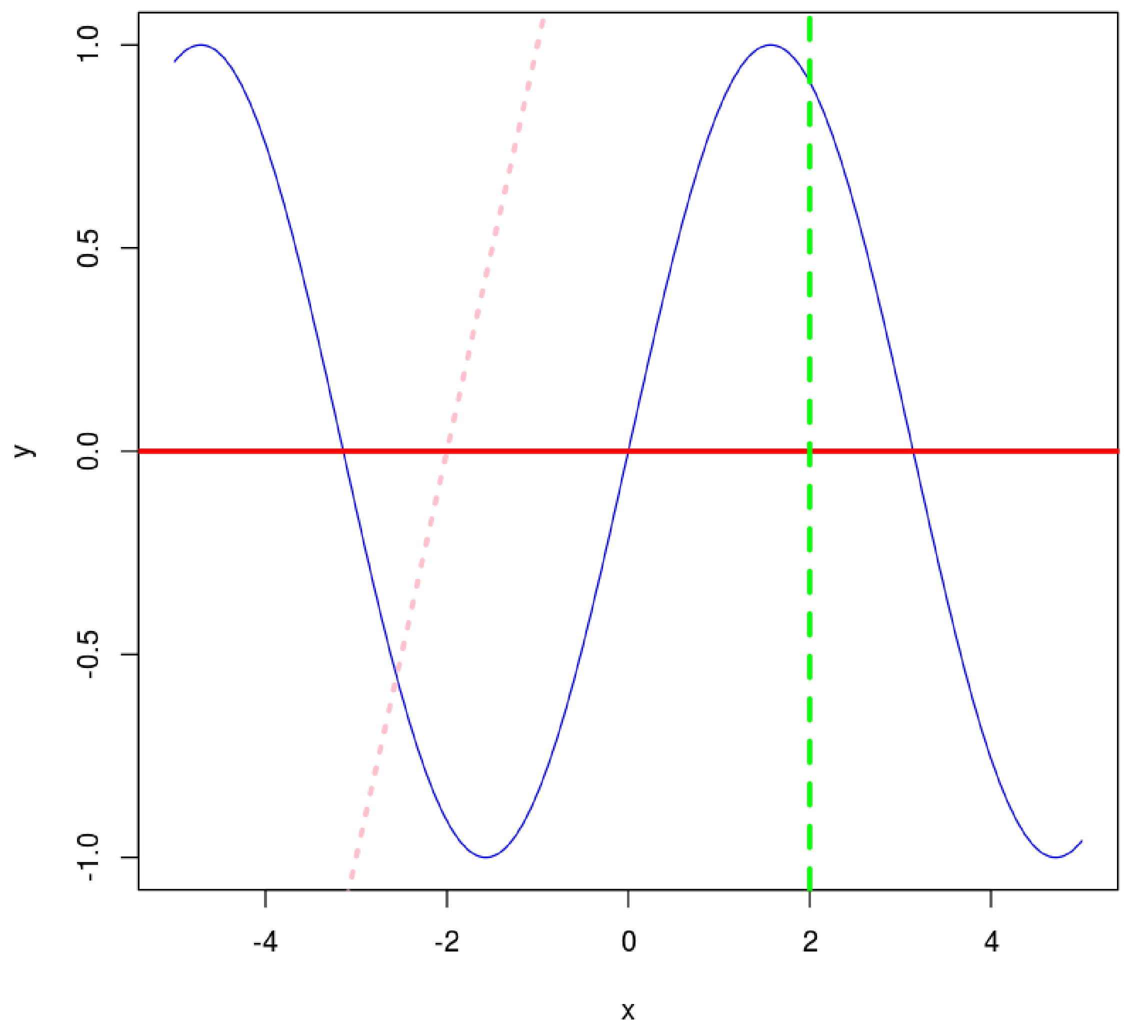
```
oldpar <- par()
par(mfrow = c(1,2))
hist(islands, breaks = 16)
boxplot(islands)
par(oldpar)
```

```
Warning message:
In par(oldpar): graphical parameter "cin" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "cra" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "csi" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "cxy" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "din" cannot be setWarning messag
e:
In par(oldpar): graphical parameter "page" cannot be set
```



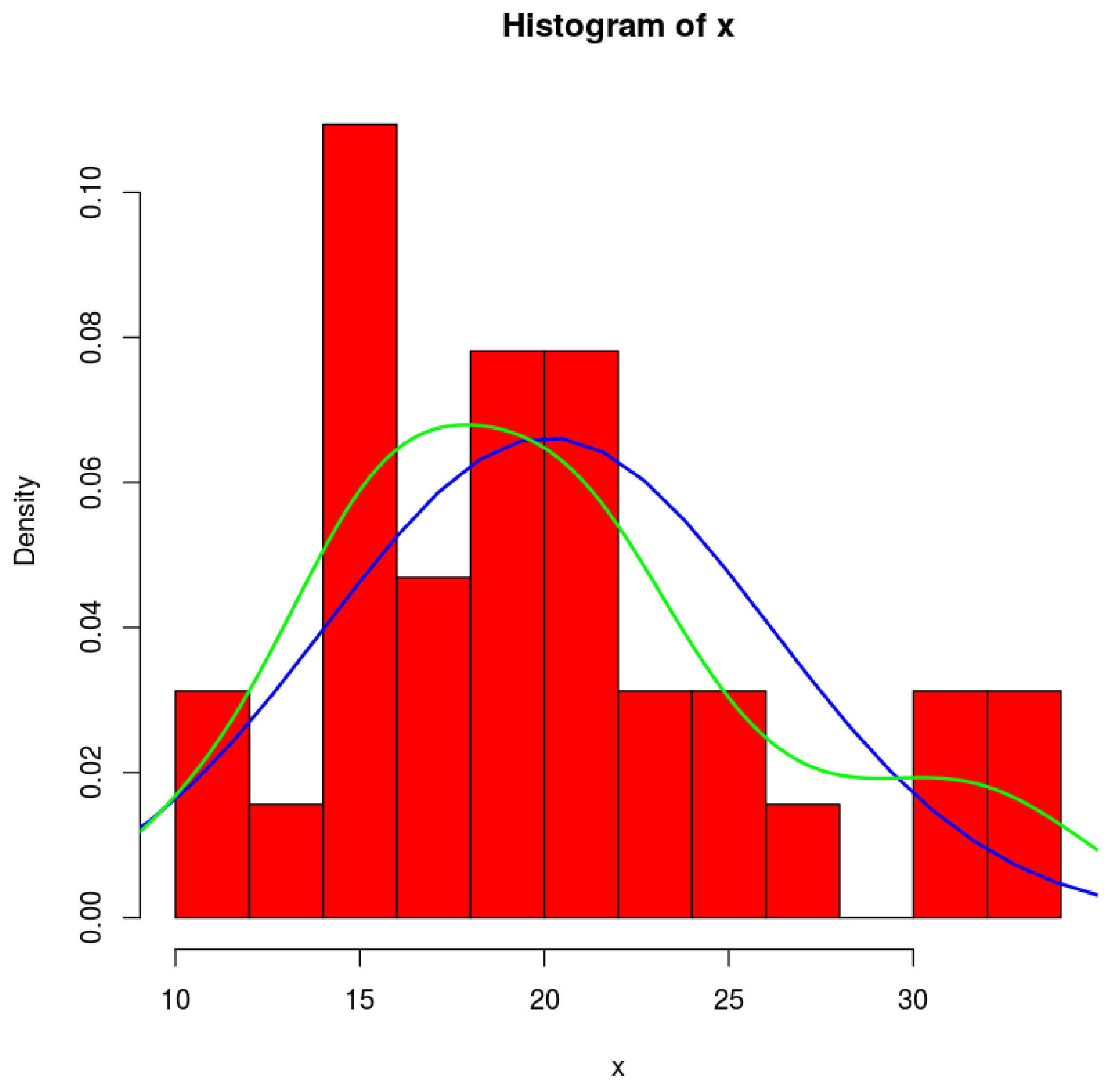
In [97]:

```
drawFun <- function(f){
    x <- seq(-5, 5, len=1000)
    y <- sapply(x, f)
    plot(x, y, type="l", col="blue")
}

drawFun(sin)
abline(h=0, col="red", lwd=3, lty=1)
abline(v=2, col="green", lwd=3, lty=2)
abline(2,1, col="pink", lwd=3, lty=3)
```



In [98]:

```
#develop a function which overlays a normal approximation density function and kern

funn <- function(x){
    h <- hist(x, col="red", breaks=10, freq=FALSE)
    xfit<-seq(min(x)-10,max(x)+10,length=40)
    yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
    #yfit <- yfit*diff(h$mids[1:2])*length(x)
    lines(xfit, yfit, col="blue", lwd=2)
    d <- density(mtcars$mpg) # returns the density data
    lines(d, col="green", lwd=2)


}

funn(mtcars$mpg)
```



7.2 Saving Garphs

In [99]:

```
Temperature <- airquality$Temp
```

In [100]:

```
#to save as a jpeg
#Saves to the currnt directory
jpeg(file="plot1.jpeg")
hist(Temperature, col="darkgreen")
dev.off()

#saving as a png
png(file="plot2.png",
   width=600, height=350)
hist(Temperature, col="gold")
dev.off()

#saving as a pdf file
pdf(file="saving_plot4.pdf")
hist(Temperature, col="violet")
dev.off()
```

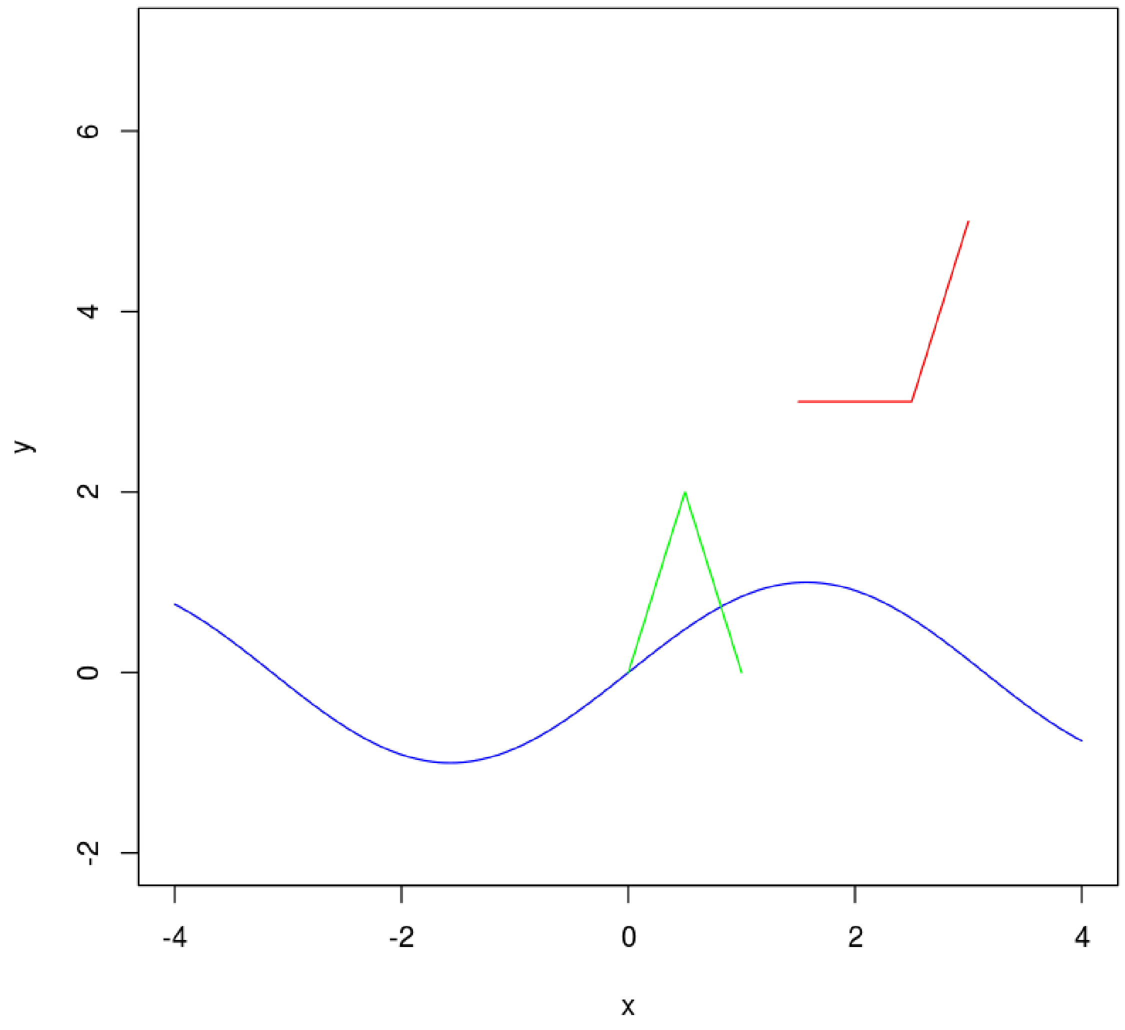
png: 2

png: 2

png: 2

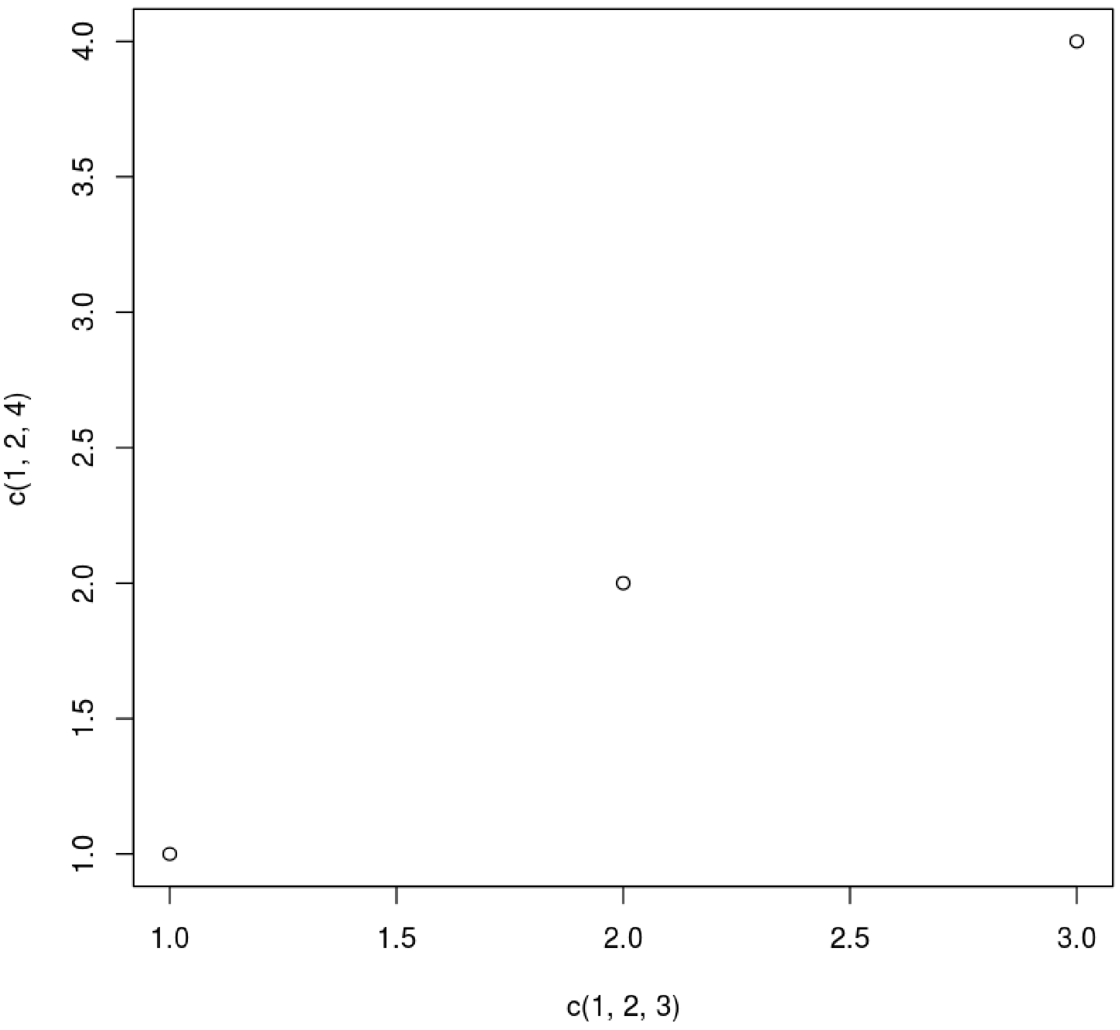
In [101]:

```
x <- seq(-4,4, 0.01)
y <- sin(x)
plot(x,y, ylim=c(-2,7), type="l", col="blue")
lines(c(1.5,2.5,3),c(3,3,5), col="red")
lines(c(0,0.5,1),c(0,2,0), col="green")
```



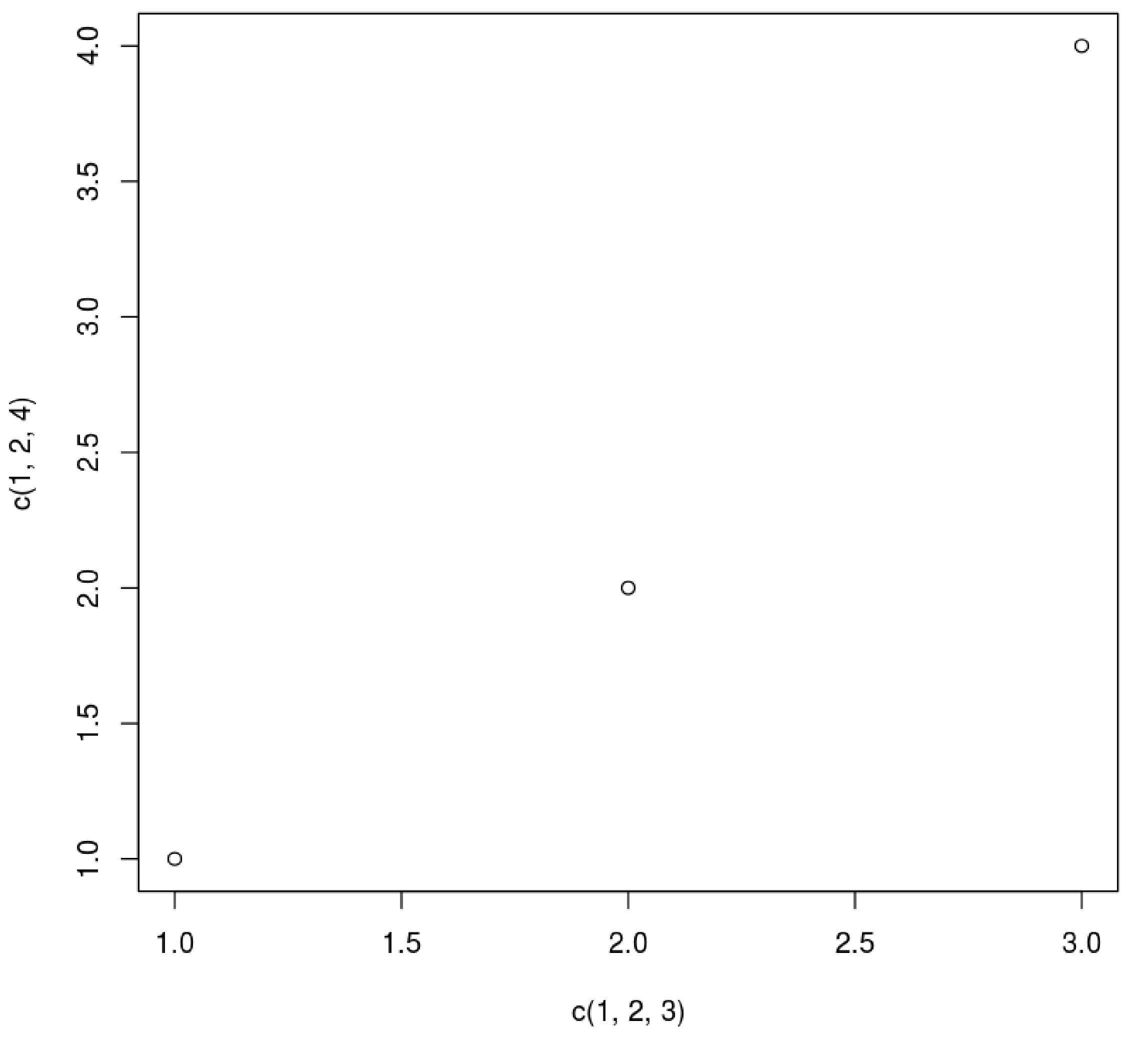
In [102]:

```
plot(c(1,2,3), c(1,2,4))
```



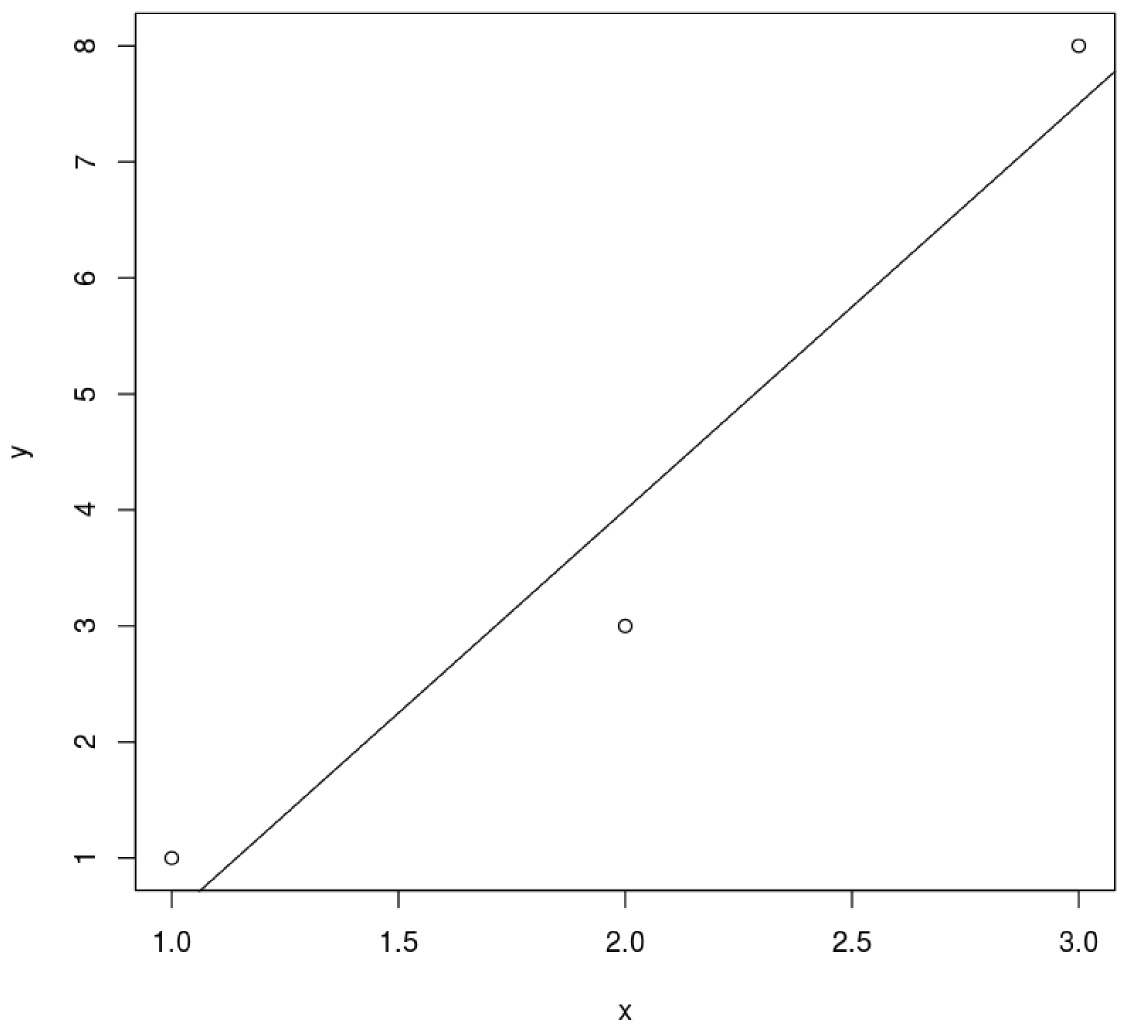
In [103]:

```
plot(c(1,2,3), c(1,2,4))
```



In [104]:

```
x <- c(1,2,3)
y <- c(1,3,8)
plot(x,y)
lmout <- lm(y ~ x)
abline(lmout)
```

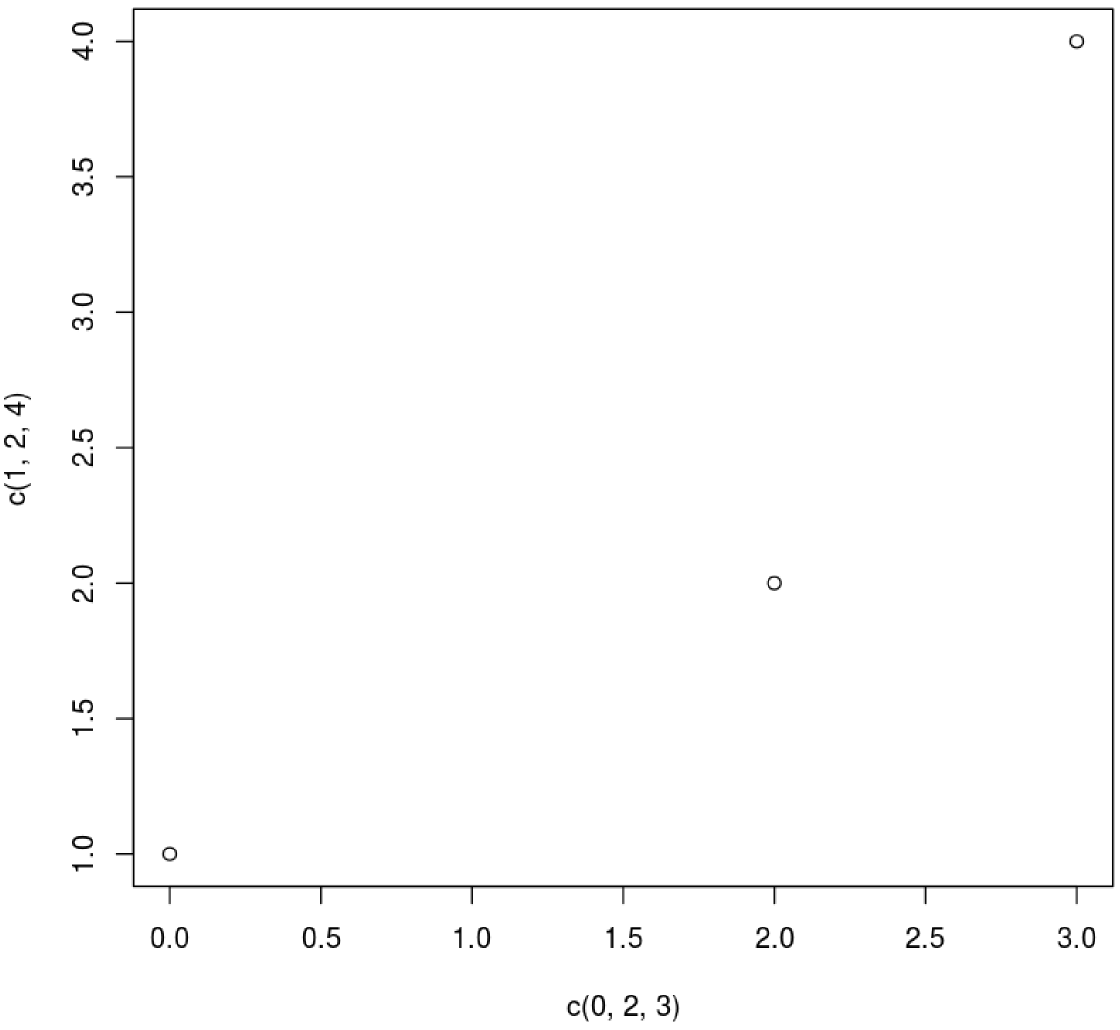


plot() is a generic function meaning that it is a placeholder for a family of functions. The function that actually gets called will depend on the class of the object on which it is called. Using plot(), you can add componenets one by one.

- **abline()** then adds a line to the current graph
- **lines()** gets a vector of x values and a vector of y values, and joins the ponits to each other
- **points()** function adds a set of (x,y)-points
- **legend()** is used to add a legend to a multicurve graph
- **text()** function places some text anywhere in the current graph
- **mtext()** adds text in the margins
- **polygon()** draws arbitrary polygonal objects

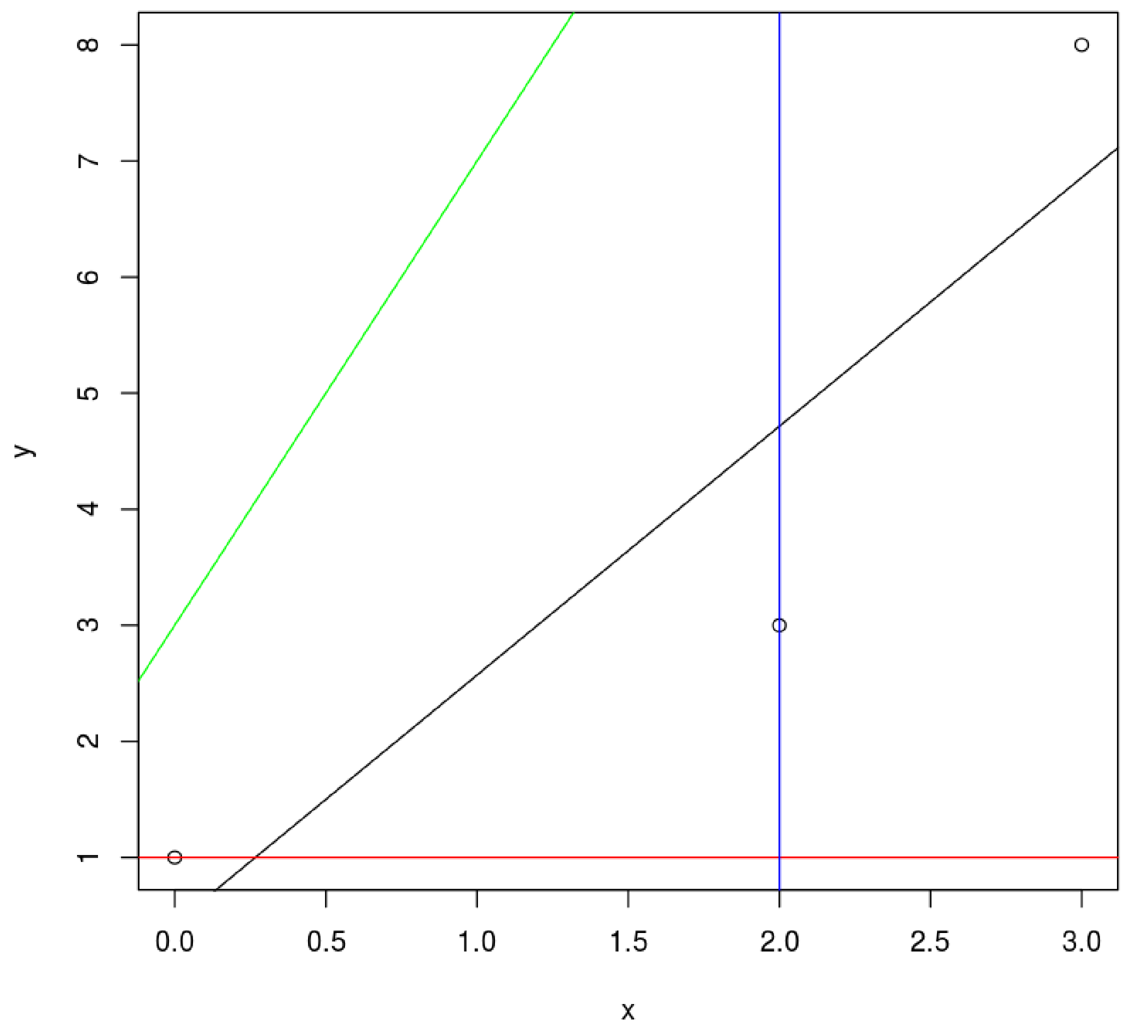
In [105]:

```
plot(c(0,2,3), c(1,2,4))
```



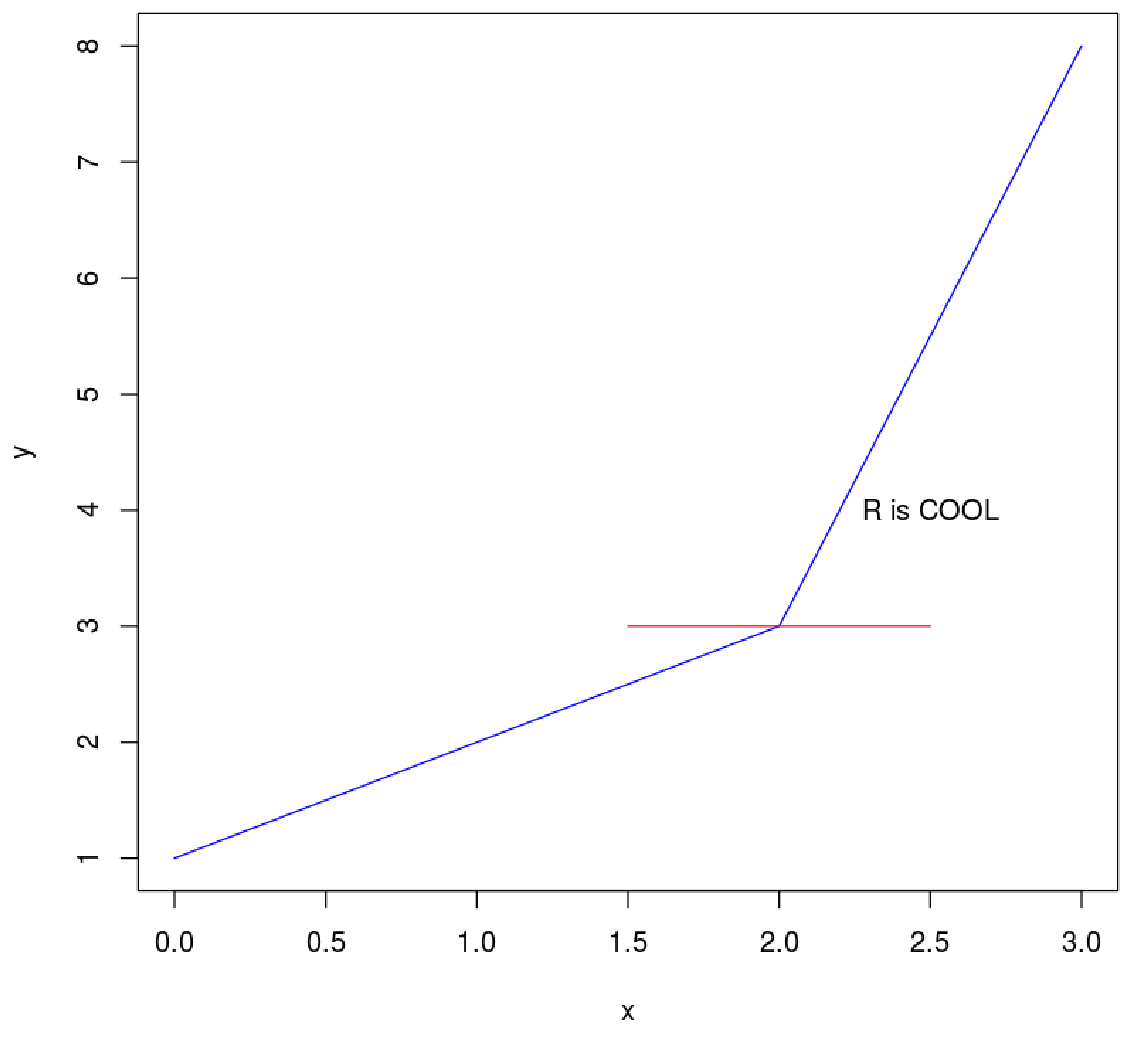
In [106]:

```
x <- c(0,2,3)
y <- c(1,3,8)
plot(x,y) # same as before
fit <- lm(y ~ x) # a regression line
#The call to abline() then adds a line to the current graph.
#abline(c(2,1)) adds y = x + 2
abline(fit) #adds a line to a plot.
abline(h=1, col="red")
abline(v=2, col="blue")
abline(3,4, col="green") # y=3x+4
```



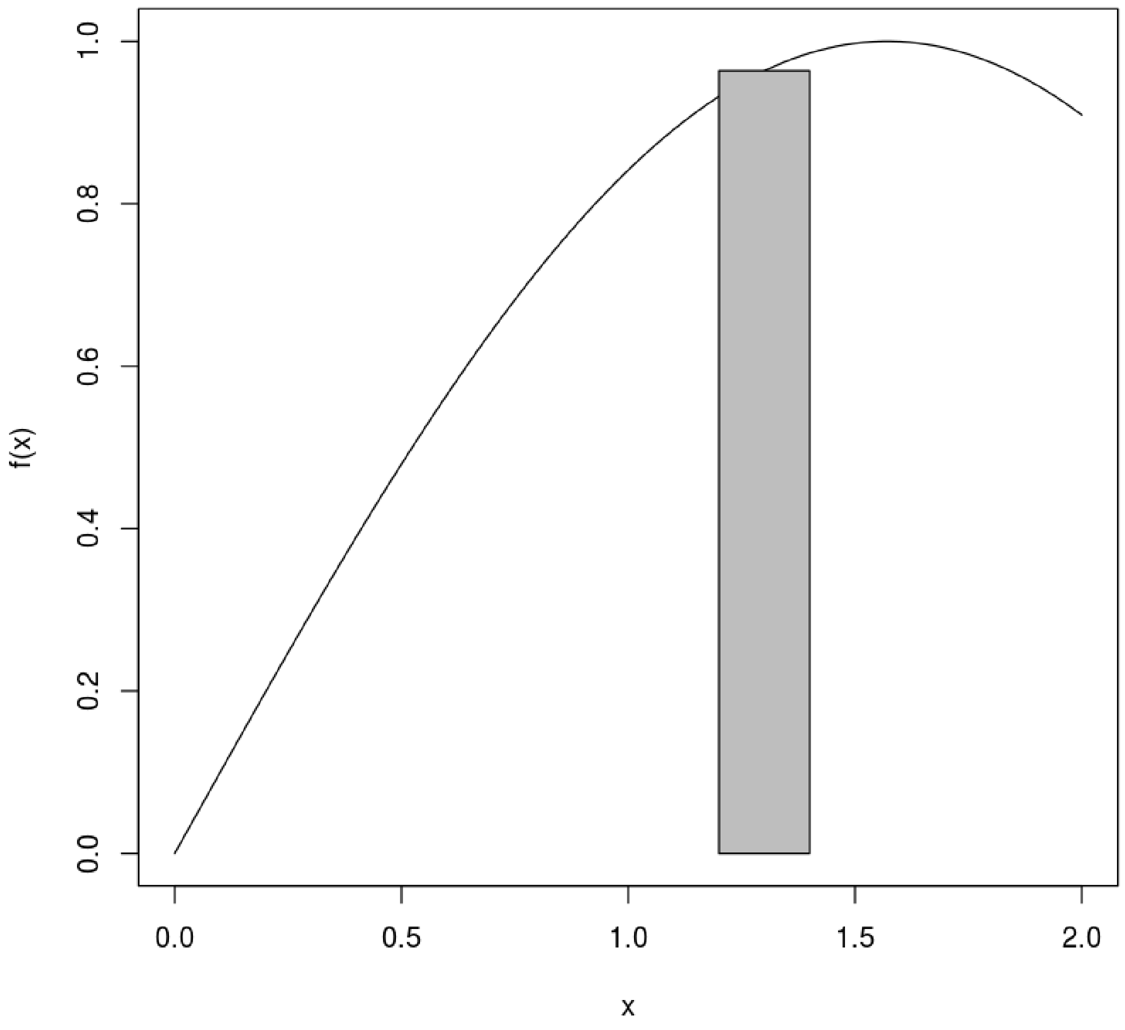
In [107]:

```
plot(x,y, type="l", col="blue")
lines(c(1.5,2.5),c(3,3), col="red")
text(2.5,4,"R is COOL")
```



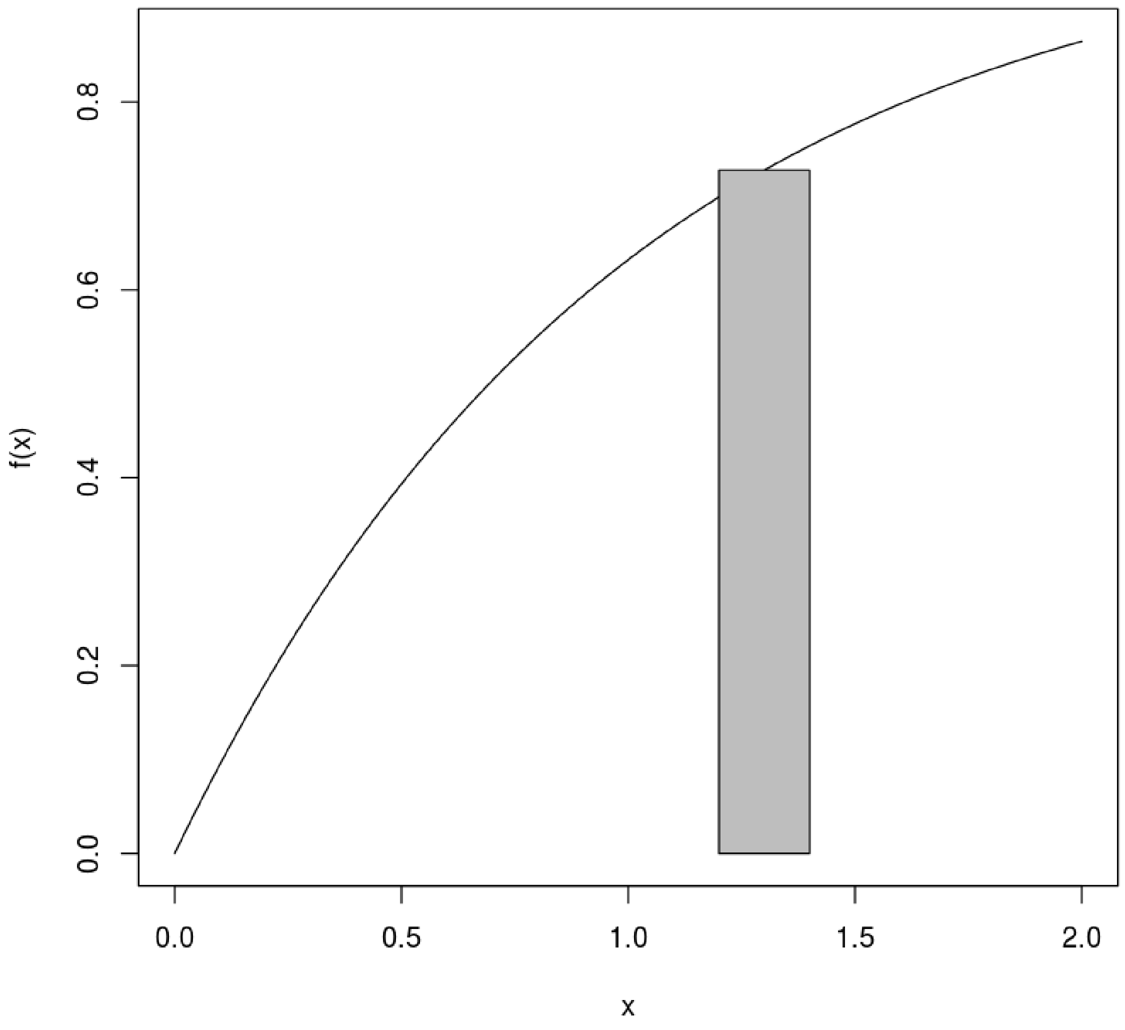
In [108]:

```
f <- function(x) return(sin(x))
curve(f,0,2)
polygon(c(1.2,1.4,1.4,1.2),c(0,0,f(1.3),f(1.3)),col="gray")
```



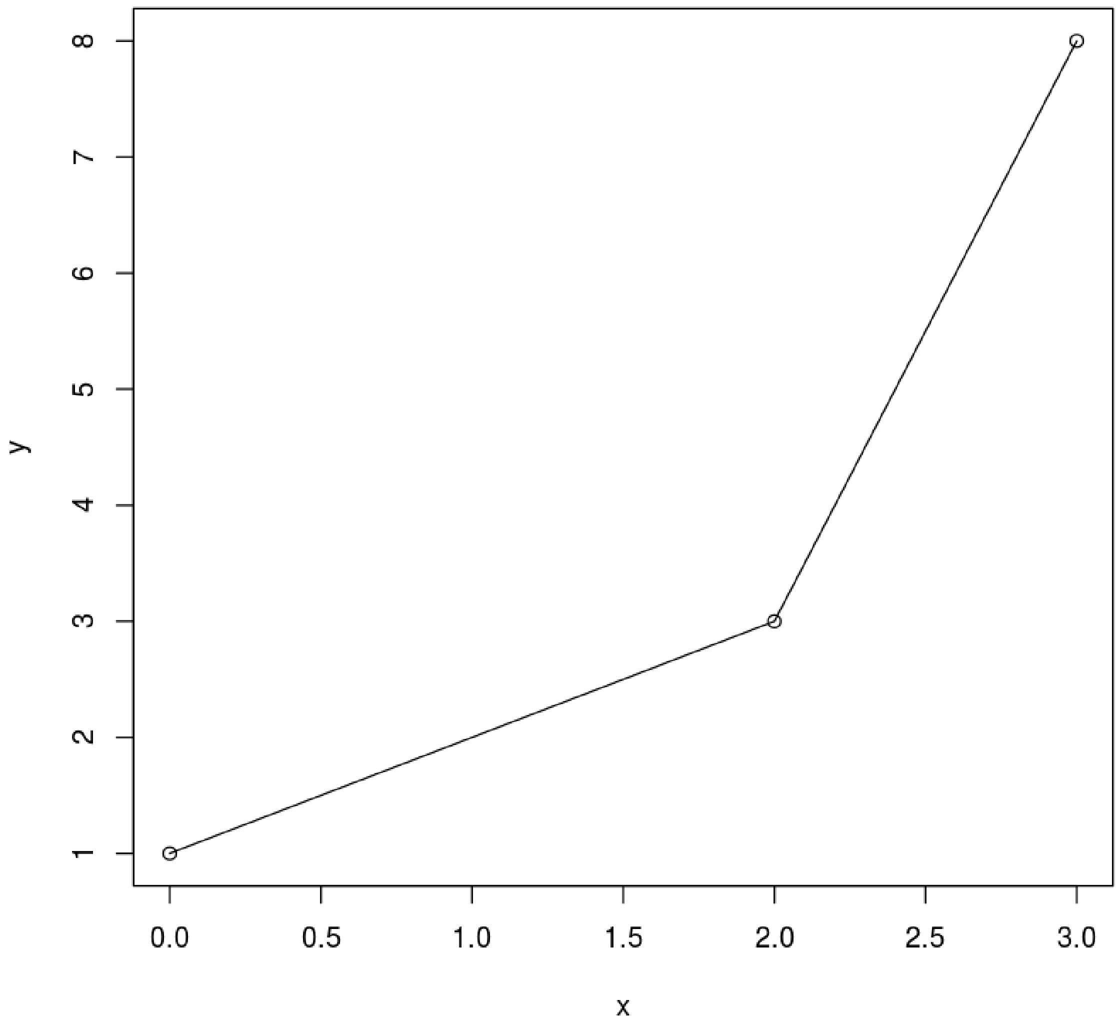
In [109]:

```
f <- function(x) return(1-exp(-x))
curve(f,0,2)
polygon(c(1.2,1.4,1.4,1.2),c(0,0,f(1.3),f(1.3)),col="gray")
```



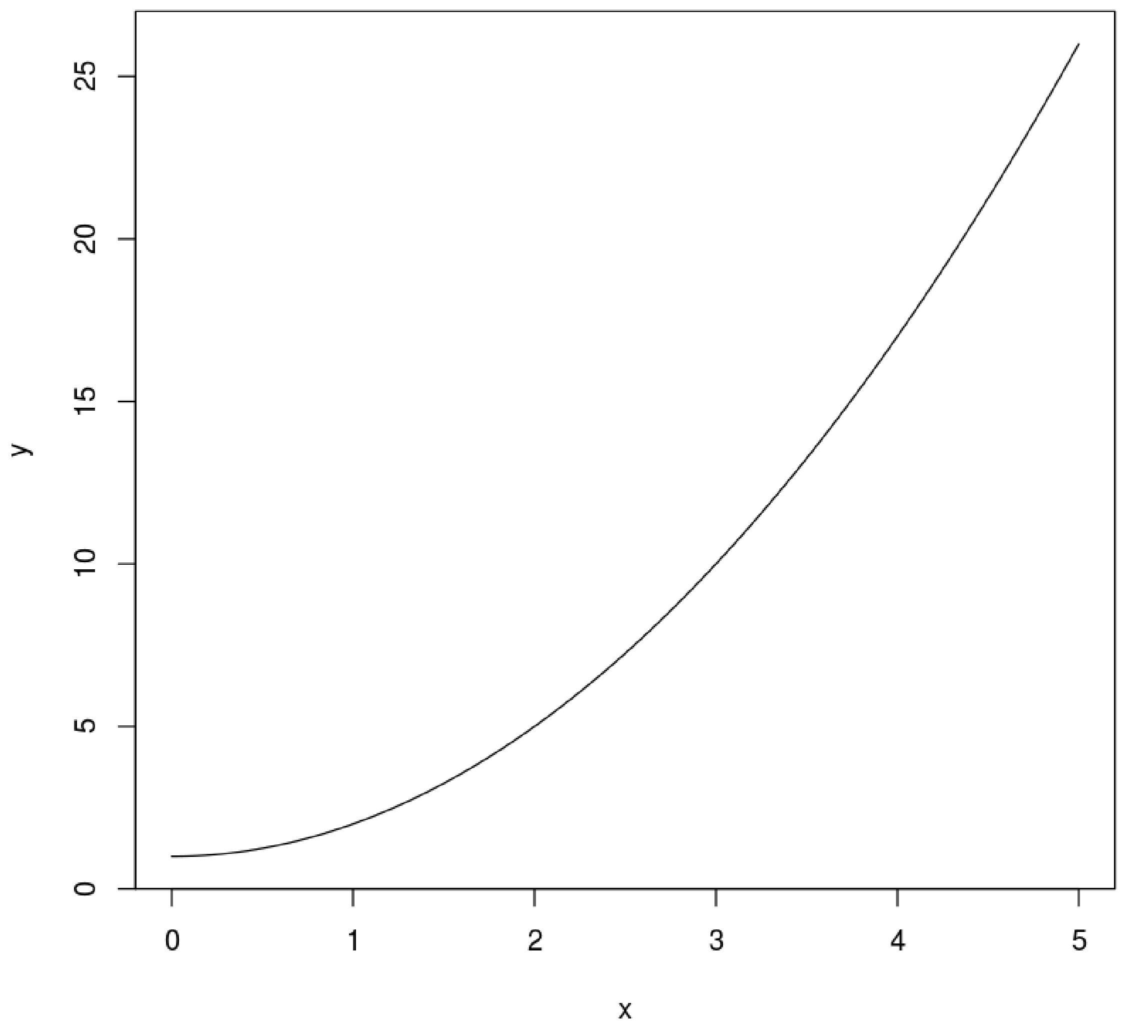
In [110]:

```
plot(x,y)
lines(lowess(x,y))
```



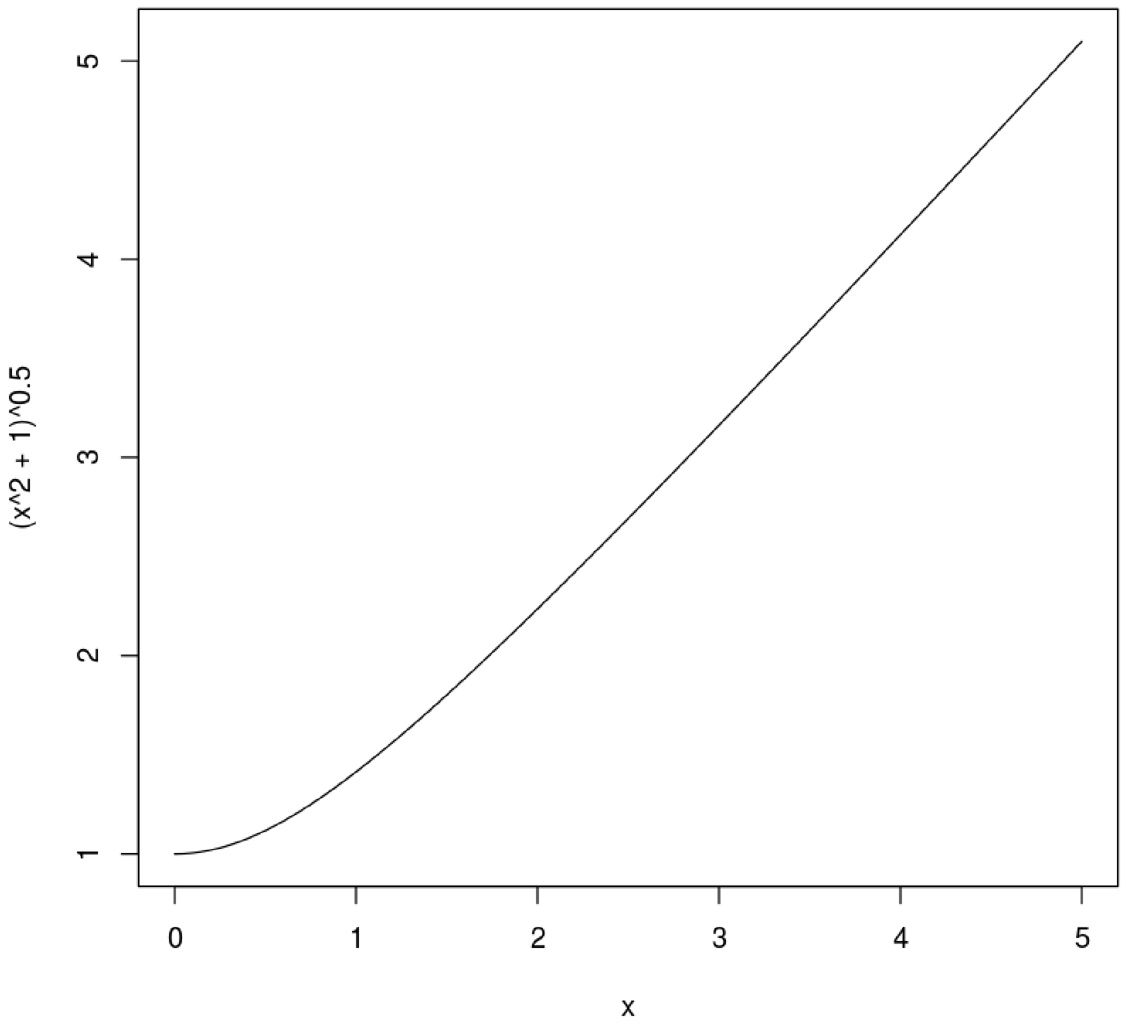
In [111]:

```
g <- function(t) { return (t^2+1)^0.5 } # define g()
x <- seq(0,5,length=10000) # x = [0.0004, 0.0008, 0.0012,..., 5]
y <- g(x) # y = [g(0.0004), g(0.0008), g(0.0012), ..., g(5)]
plot(x,y,type="l")
```



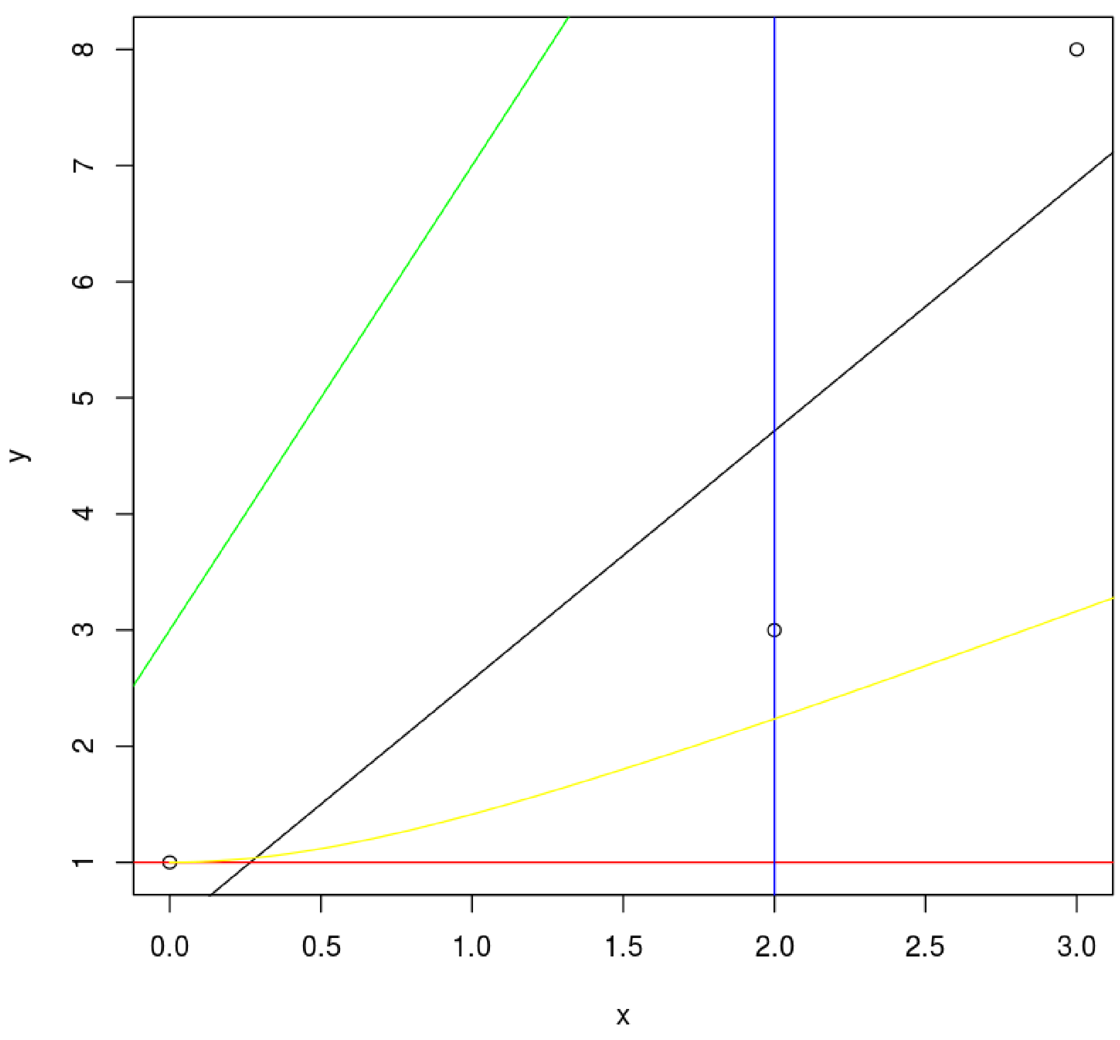
In [112]:

```
curve((x^2+1)^0.5,0,5)
```



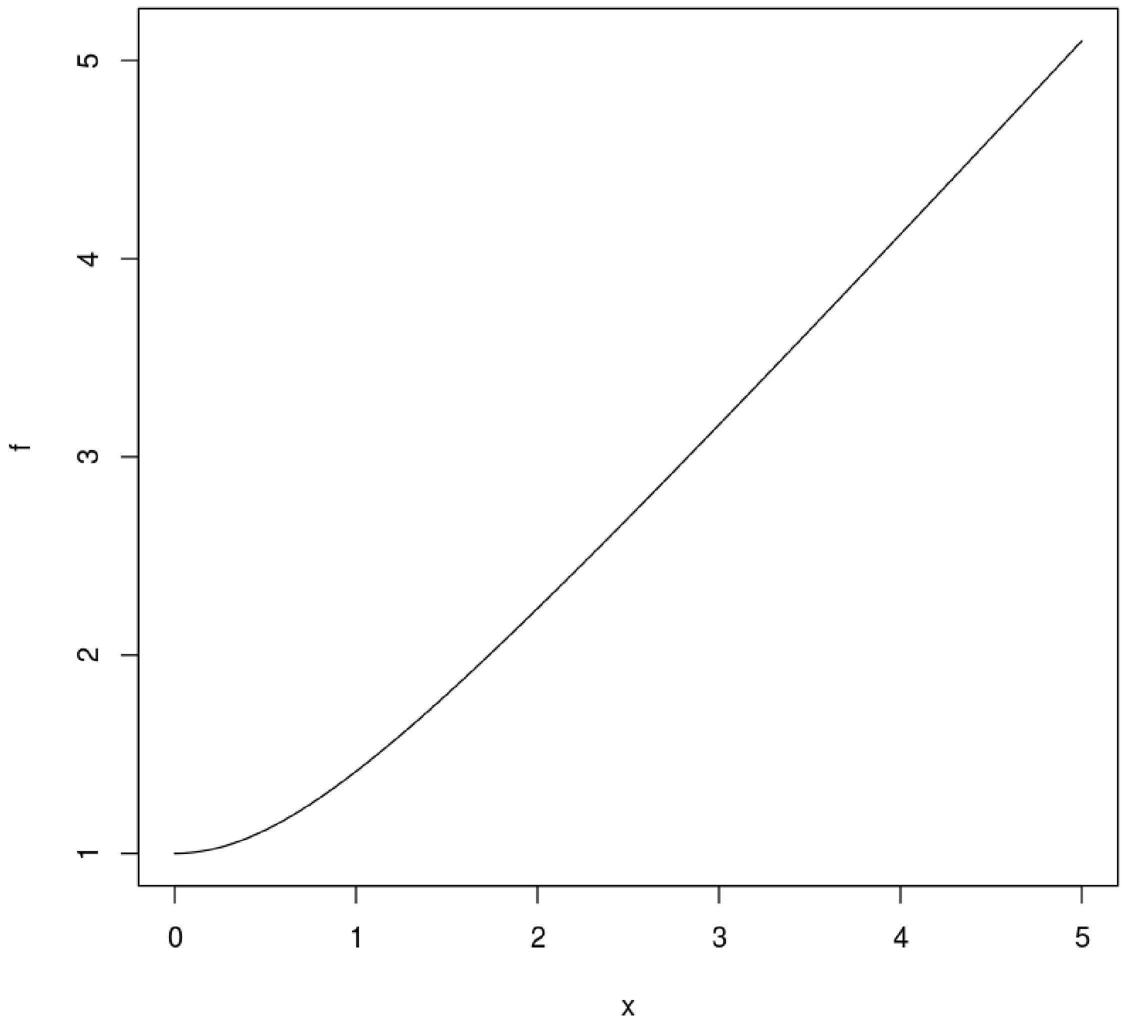
In [113]:

```
x <- c(0,2,3)
y <- c(1,3,8)
plot(x,y) # same as before
fit <- lm(y ~ x) # a regression line
#The call to abline() then adds a line to the current graph.
#abline(c(2,1)) adds y = x + 2
abline(fit) #adds a line to a plot.
abline(h=1, col="red")
abline(v=2, col="blue")
abline(3,4, col="green") # y=3x+4
curve((x^2+1)^0.5,0,5,add=T, col="yellow")
```



In [114]:

```
f <- function(x) return((x^2+1)^0.5)
plot(f,0,5) # the argument must be a function name
```



In [ ]:

```
```