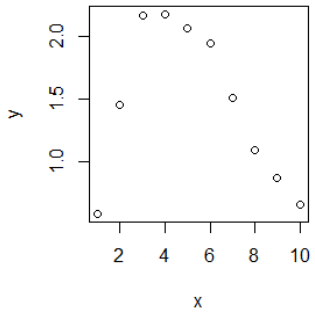
Graphing

Scatter plots

Perhaps the most common type of plot encountered in the scientific and engineering literature is the scatter plot of a dependent variable y measured or evaluated at a set of discrete points of an independent variable x. As an example, we simulate measurement of the function $y=x^2e^{-x/2}$ over the x-range 0–10 with 5% normally distributed random error.

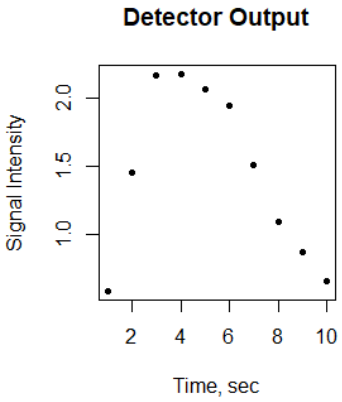
```
>  set.seed(123) # Enables reproducible random number generation
>  x = 1:10
>  y = x^2*exp(-x/2)*(1+rnorm(n=length(x), mean=0, sd=0.05))
>  par(mfrow=c(1,2)) # Plots in 1 row, 2 columns
>  plot(x,y)
```



This gives a plot suitable for initial inspection of the data, but not for formal presentation or even recording in a lab notebook. At a minimum, one will want to label the axes more informatively with xlab and ylab, and provide a title with main. In addition, one may want to change the point symbol with pch and make the points smaller with cex.

```
>  plot(x,y, pch=19, cex=0.7, xlab="Time, sec",
+  ylab = "Signal Intensity", main = "Detector Output")
```

This gives the much better



By default, R plots (x;y) pairs as points, but lines (type="l") and overlays (type="o") are also useful.

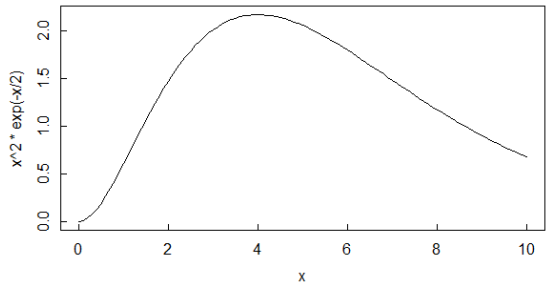
```
set.seed(123) # Enables reproducible random number generation
>  x = seq(from=0, to=10, by=0.1) # More closely spaced points
>  y = x^2*exp(-x/2)*(1+rnorm(n=length(x), mean=0, sd=0.05))
>
>  par(mfrow=c(1,2))
>   par(mar=c(4,4,1.5,1.5),mex=.8,mgp=c(2,.5,0),tcl=0.3)
>   plot(x,y, type = "l")
>   plot(x,y, type = "o")
>   par(mfrow=c(1,1))
```

Function plots

R enables plotting of functions of the variable x with the command curve. For example, to plot the function used to generate the data above, we write

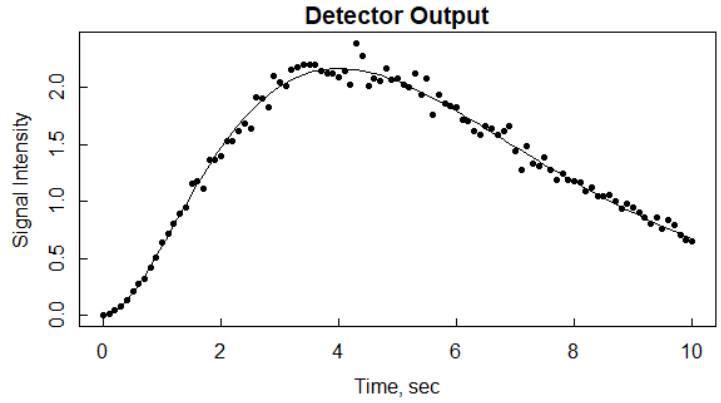
```
> curve(x^2*exp(-x/2),0,10)
```

with the result shown in Figure



To superimpose the curve on the data points, we plot the data first, then generate the curve with the condition add = TRUE:

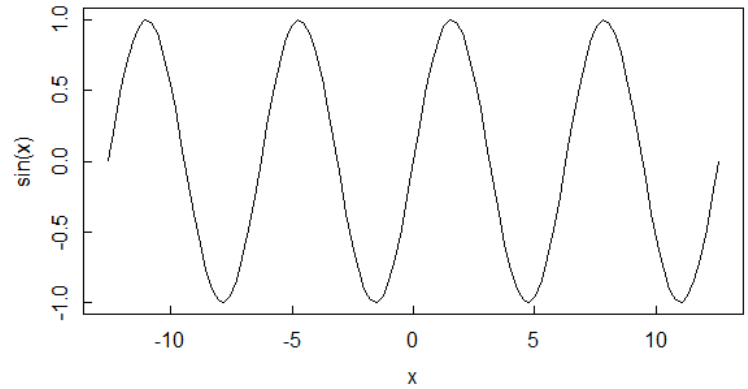
```
>     plot(x,y, pch=19, cex=0.7, xlab="Time, sec",
+     ylab = "Signal Intensity", main = "Detector Output")
>     curve(x^2*exp(-x/2),0,10, add=T)
```



The command curve() works only with the variable x, and if given a function with no argument will assume that the argument is x, as in

```
>      curve(sin,-4*pi,4*pi)
```

where we note that R provides x in the axis labels even though we did not specify it



On the other hand, the code

> curve(x)

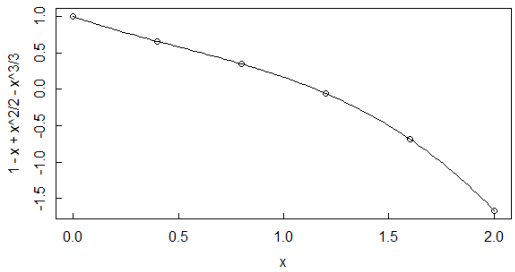
generates the error message

Error in eval(expr, envir, enclos) : could not find function "x"

because R is unsure whether x is a function or a variable.

Just as we could superimpose a function curve on a scatter plot with the curve(..., add=T) command, we can superimpose points on a function plot with the points() command. In Figure, for example, are evenly spaced points imposed on a polynomial curve.

```
>     curve(1-x+x^2/2-x^3/3,0,2)
>     x = seq(0,2,.4)
>     y = 1-x+x^2/2-x^3/3
>     points(x,y)
```



In similar fashion, connected line segments could be added to a plot of points with the lines() function. See ?lines for an example using data from the cars dataset in the R base package.

Other common plots

Bar charts

Bar charts are known in R as barplots. For example, suppose a nutritionist is doing a feeding study using feeds A and B. She measures the average weight of two groups of mice, one group fed A and the other B, each week for three weeks, with results as indicated in the following code and shown in Figures.

```
>   A = c(3,4,4)
>    B = c(6,8,10)
>     feed = matrix(c(A,B), nrow=2, byrow=TRUE,
+                         dimnames= list(c("A","B"), c("1","2","3")))
>    feed # Check that we've set up the matrix correctly
  1 2  3
A 3 4  4
B 6 8 10
```

Now plot stacked barplots using the default beside = FALSE option, emphasizing, on the left, time as the independent variable:

```
>        barplot(feed,xlab="Week",ylab="grams gained",
+                         main = "Weight Gain by Week\n", legend.text=c("A","B"),
+                        args.legend=list(title="Feed",x="topleft",bty="n"))
```

and, on the right, the feeds:

```
>         barplot(t(feed),xlab="Feed",ylab="grams gained",
+                                       main = "Weight Gain by Feed\n",legend.text=c("1","2","3"),
+                                        args.legend=list(title="Week",x="topleft",bty="n"))
>
```

We can display the same data in a more expanded form using the beside =TRUE option as in the following code:

```
  >         barplot(feed, beside=T,xlab="Week",ylab="grams gained",
+                            main = "Weight Gain by Week\n", legend.text=c("A","B"),
+                            args.legend=list(title="Feed",x="topleft",bty="n"))
```

and

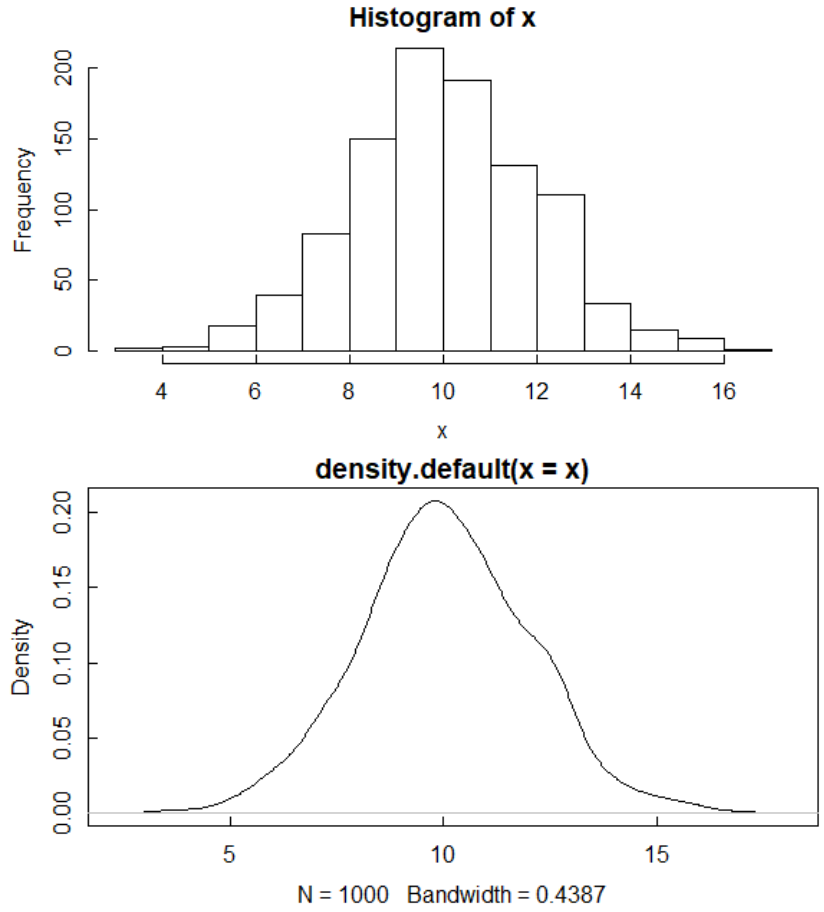
```
>           barplot(t(feed), beside=T,xlab="Feed",ylab="grams gained",
+                          main = "Weight Gain by Feed\n",
+                         legend.text=c("1","2","3"),
+                         args.legend=list(title="Week",x="topleft",bty="n"))
```

In main for both of these plots, \n is the newline command, introducing an extra line spacing after the main title. There are many options to the barplot command, some of which we have used above.

Histograms

Histograms are commonly used to display the distribution of repeated measurements. R does this with the hist function. If the fraction of measurements falling into each range is desired instead, use plot(density), where the density function gives useful numerical data about the distribution. As an example, we generate 1000 normally distributed random numbers with mean 10 and standard deviation 2. Results are shown in Figure.

```
>  set.seed(333)
>  x = rnorm(1000,10,2)
>   hist(x)
>   plot(density(x))
```



Call:
density.default(x = x)

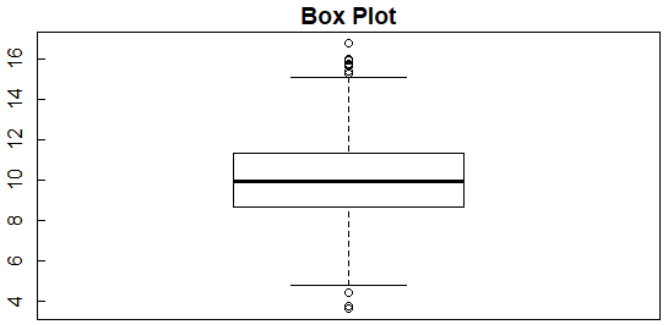
Data: x (1000 obs.); Bandwidth 'bw' = 0.4387

x	y
Min. : 2.333	Min. :1.035e-05
1st Qu.: 6.277	1st Qu.:3.456e-03
Median :10.220	Median :2.737e-02
Mean :10.220	Mean :6.333e-02
3rd Qu.:14.164	3rd Qu.:1.183e-01
Max. :18.108	Max. :2.078e-01

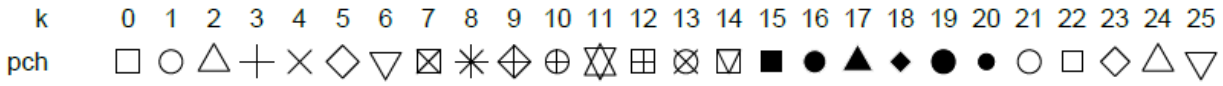
The hist function tends to have a mind of its own when setting breaks between classes. To control this, use the argument breaks = vecbreaks, where vecbreaks is a vector that explicitly gives the breakpoints between histogram cells.

Box-and-whisker plots
The boxplot function gives a very informative graphical representation of a distribution of x in this case (Figure). The box shows the values of the first and third quartiles, the heavy line in the middle of the box gives the median, and the whiskers give the values of the quartile plus approximately 1.5 times the length of the interquartile range. Values beyond the whiskers are given by points. The plot was generated simply by

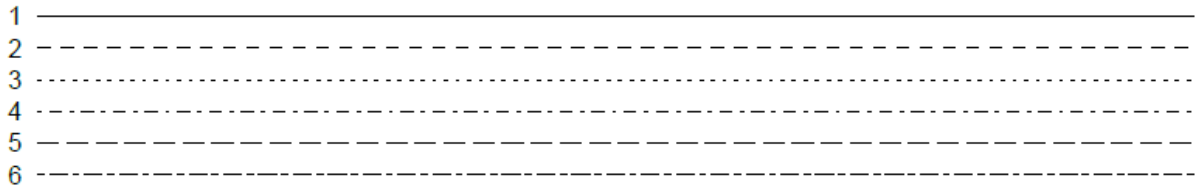
```
boxplot(x, main = "Box Plot")
```



Customizing plots
Points and lines
R has 26 point styles, numbered from 0 to 25, which can be specified by pch(n)



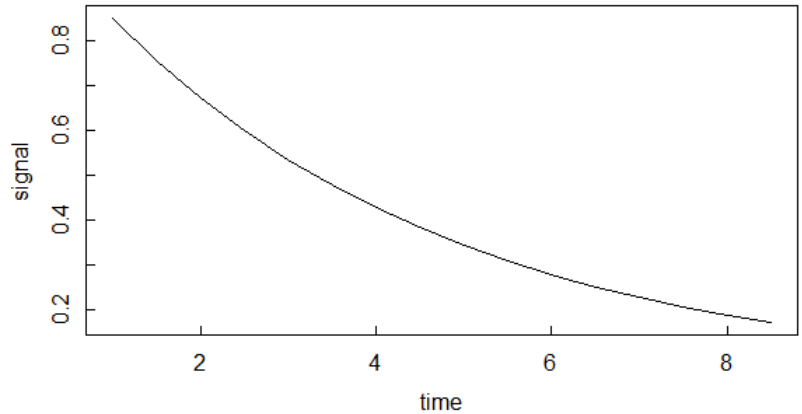
It also has six line types, numbered from 1 to 6, which can be specified by lty(n)



As noted earlier, the size of points, relative to the default size, can be set by cex.
Similarly, the relative thickness of lines can be set by lwd.

Axes, ticks, and par()
R automatically chooses the scales of x and y axes, but sometimes you'd like a different choice. This is done with xlim and ylim, in which you explicitly set the upper and lower limits for the axes. Points and lines can also be colored, using the parameter col in plot or curve. The default, col=1, gives black. Integers 2–6 correspond to red, green, blue, cyan, and magenta, respectively. These colors can also be called by name, e.g., col ="red".
Typing ?palette gives much more information on graphic color capabilities in R.
The axes, text placement, and other aspects of a graph can readily be customized in R. For example, suppose we want a line plot of the function $0.8e^{-t/4} + 0.05$ for t values between 1 and 8.5. The result, produced by the code

```
>  time = seq(1,8.5,.5)
>   signal = 0.8*exp(-(time-1)/4) + 0.05
>   plot(time, signal, type="l")
```



We can modify this default result in several ways. For example, suppose we want the ordinate to run from 0 to 10, and the abscissa from 0 to 1.We want the ticks to be inside the axes rather than outside and to be a bit shorter. We want the plot margins to be somewhat smaller, and the axis labels to be closer to the axes, to tighten up the white space. We also want to add some lines to the graph, to emphasize that the signal starts at time = 1 and that it levels off at signal = 0.05. These modifications are accomplished by the following code, giving the result shown in Figure.

```
>   par(mar=c(4,4,1.5,1.5),mex=.8,mgp=c(2,.5,0),tcl=0.3)
>    plot(time, signal, type="l",xlim = c(0,10), ylim = c(0,1))
>    abline(h = 0.05, lty=3) # Horizontal line at y = 0.05
>    abline(v=1,lty=3) # Vertical line at x = 1
```

The axis limits are specified with xlim = c(0,10) and ylim = c(0,1). The tick direction and length are set with tcl = 0.3 (the default is -0.5, where the minus sign specifies ticks outside the plot, and the number is the fraction of a line height).
The internal lines in the plot are drawn with the abline command. Additional aspects of the plot are set with various arguments to the par function, which specifies a broad range of graphical parameters. For example, figure margins may be made tighter or looser with the mar argument to par, (the default is mar =c(5,4,4,2)+0.1), where the numbers are multiples of a line height and are in the order bottom, left, top, right. mex determines coordinates in the margins of plots; values less than 1 move the labels toward the margins, thus decreasing white space. The location of the axis labels is set by mgp (the default is mgp = c(3,1,0)). The new settings of the par parameters are retained until odified by future par() statements, or until a new R session is started.

Overlaying plots with graphic elements

```
par(mar=c(1.5,1.5,1.5,1.5)) # Reduce margins
  par(mfrow=c(1,2))
plot.new(); plot.window(c(0,100), c(0,100), asp=1); box()
   x0=c(5,10,15); y0=x0; x1 = x0 + 5; y1 = y0 + 60
   segments(x0=x0,x1=x1,y0=y0,y1=y1, lty=1:3)
   rect(80,25,95,80,border="black",lwd=3)
  polygon(50+25*cos(2*pi*0:8/8), 50+25*sin(2*pi*0:8/8),
            col=gray(0.8), border=NA)

    require(plotrix) # Assumes the package is already installed

     plot.new(); plot.window(c(0,100), c(0,100), asp=1); box()
     draw.arc(20, 20, (1:4)*5, deg2 = 1:20*15)
     draw.circle(20, 80, (1:4)*5)
     draw.ellipse(80, 20, a = 20, b = 10, angle = 30, col=gray(.5))
     draw.radial.line(start=2, end = 15, center = c(80,80), angle=0)
     draw.radial.line(start=2, end = 15, center = c(80,80), angle=pi/2)
     draw.radial.line(start=2, end = 15, center = c(80,80), angle=pi)
     draw.radial.line(start=2, end = 15, center = c(80,80), angle=3*pi/2)

     par(mfrow=c(1,1))
```

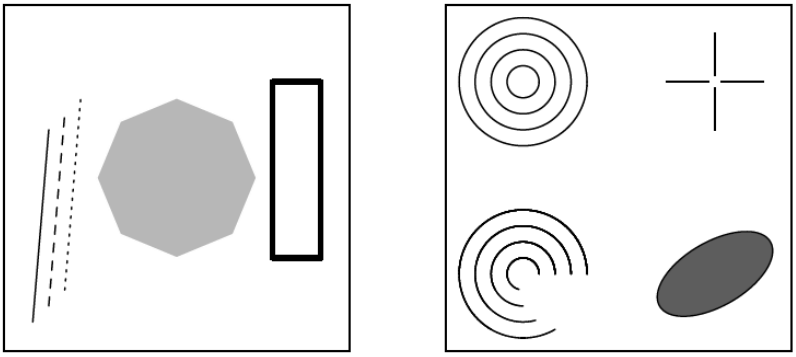


Figure : Left: Graphic elements produced with base R; Right: Graphic elements produced with plotrix package.

It is evident that these functions could be used to draw simple – or even not so simple – diagrams.
Finally, we show how to color defined areas of a plot with the polygon() function. As an example, we distinguish the positive and negative regions of a function with different shades of gray, which might be useful in a pedagogical presentation of how to integrate the function.
Consider integrating the first order Bessel function besselJ(x,1) from x = 0 to its zero-crossing point near x = 10. We first compute the crossing points with the uniroot function.

```
>          x10 = uniroot(function(x) besselJ(x,1),c(9,11))$root
>         x4 = uniroot(function(x) besselJ(x,1),c(3,5))$root
>         x7 = uniroot(function(x) besselJ(x,1),c(6,8))$root
```

We compute the value of the function over the desired range, using many steps to give a smooth polygon fill.

```
> x = seq(0,x10,len=100)
>         y = besselJ(x,1)
```

Next we construct an empty plot with the desired x and y limits, and add the polygon of the function with a medium gray fill.

```
>         plot(c(0,x10),c(-0.5,0.8), type="n", xlab="x", ylab="J(x,1)")
>         polygon(x,y,col="gray", border=NA)
```

We then “paint over” the negative region with white, and add a horizontal line at x = 0.

```
>         rect(0,-0.5,x10,0, col="white", border=NA)
>         abline(h=0, col = "gray")
```

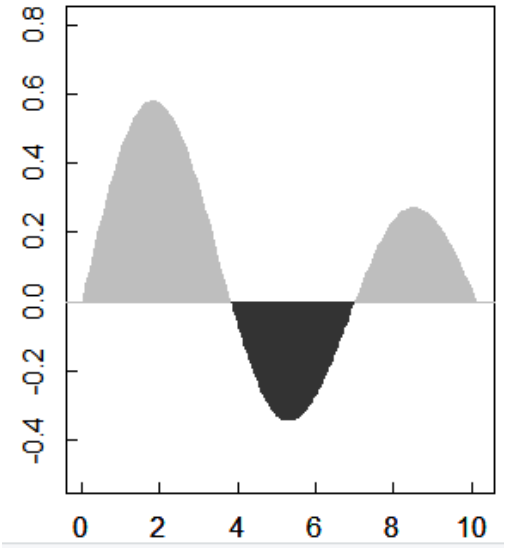
We calculate the value of the function in the negative region, again using many steps for smoothness.

```
>          xminus = seq(x4,x7,len=50)
>           yminus = besselJ(xminus,1)
```

Finally, we cover the negative region with a polygon in a darker gray, and add back the ticks that were painted over in an earlier step.

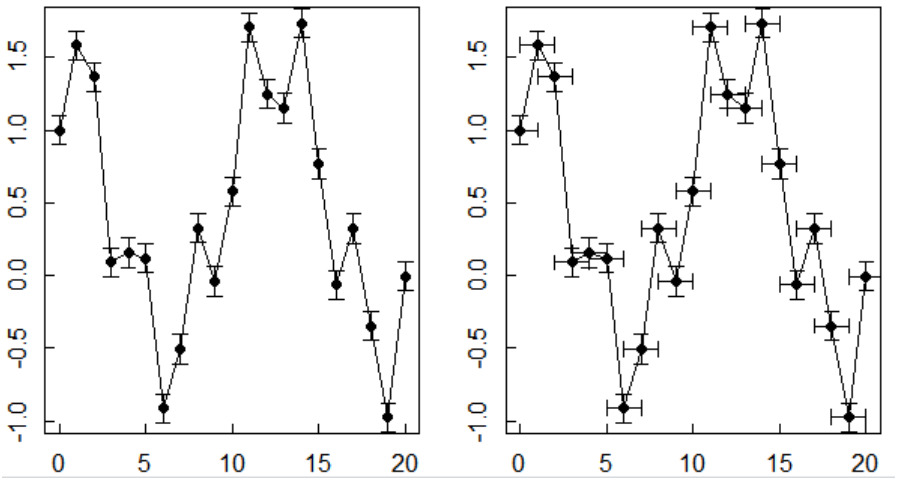
```
>           polygon(xminus,yminus, col=gray(.2), border=NA)
>           axis(1, tick=TRUE)
```

The result is seen in Figure



Plots of experimental data or simulations should generally have error bars on the points to indicate uncertainty or statistical variation. Error bars are formed in R using the arrows command, as in the following code.

```
>           x = 0:20
>           y = sin(x)^2 + cos(x/2)
>           err.y = 0.1
>          err.x = 1
>            par(mfrow=c(1,2))
>            # Plot just y error bars
>             plot(x,y,type="o", pch=19)
>           arrows(x,y,x,y+err.y,0.05,90); arrows(x,y,x,y-err.y,0.05,90)
>            # Plot both x and y error bars
>             plot(x,y,type="o", pch=19)
>           arrows(x,y,x,y+err.y,0.05,90); arrows(x,y,x,y-err.y,0.05,90)
>            arrows(x,y,x+err.x,y,0.05,90); arrows(x,y,x-err.x,y,0.05,90)
```



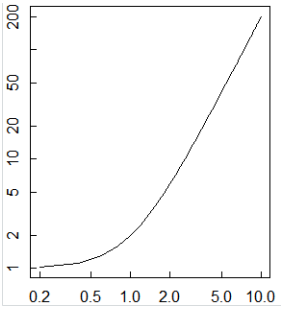
Modifying axes

Scientific and engineering graphs often need axes other than the default linear axes with ticks on the bottom and left. R has many options for customizing axes, of which we present here a few of the most commonly used.

Logarithmic axes

The following code produces a log-log plot of the function $y = 1 + x^{2.3}$ where x runs from 0.2 to 10 .

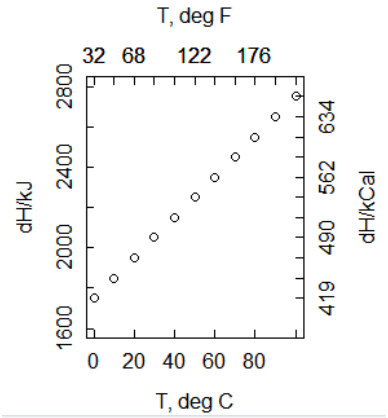
```
>           x = seq(.2,10,by=.2)
>            y = 1 + x^2.3
>            plot(x,y,log="xy", type="l")
```



Supplementary axes

Sometimes one wants the top and right axes to provide different scales than the bottom and left axes. The axes are numbered 1 (bottom), 2 (left), 3 (top), and 4 (right) . In the following code, we put temperature in celsius on the bottom and in fahrenheit on the top, and energy in kilojoules on the left and kilocalories on the right. The par commands online 3 set margins for the plot

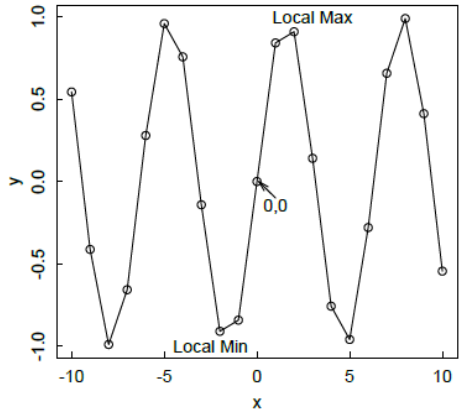
```
>               tC = seq(0,100,10)
>                dH = 2000 + 10*(tC - 25)
>                par(tcl=0.3, mar=c(3,3,4,4)+0.1, mgp = c(2,0.4,0))
>                plot(tC, dH, xlim = c(0,100), ylim = c(1600,2800),
+                      xlab="T, deg C",ylab="dH/kJ", tcl=0.3)
>                axis(3, at = tC, labels = tC*9/5+32, tcl=0.3)
>                mtext(side=3,"T, deg F", line = 2)
>                axis(3, at = tC, labels = tC*9/5+32, tcl=0.3)
>                axis(4, at = dH, labels = round(dH/4.18,0), tcl=0.3)
>                mtext(side=4,"dH/kCal", line = 2)
```



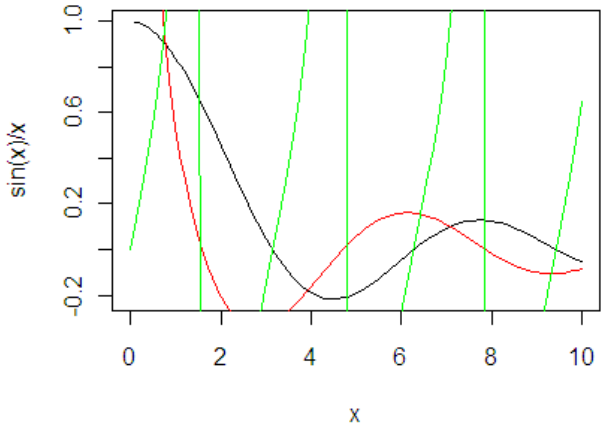
Adding text and math expressions

To annotate certain points on a graph, one can use the arrows and/or text functions.

```
> x = -10:10
> y = sin(x)
> plot(x,y, type="o")
> text(3,1, "Local Max")
> text(-2.5,-1,"Local Min")
> arrows(1,-.1,.1,0,length=0.1,angle=15)
> text(1,-.15,"0,0")
```



```
>           curve(sin(x)/x, 0, 10)
>           curve(cos(x)/x, 0, 10, col="red", add=TRUE)
>           curve(tan, 0, 10, col="green", add=TRUE)
```



TASK

Plot function

$$y = \frac{x^2 - 4x + 3}{9 - 3x}$$

$$y = x^2 - 8x + 15$$

$$y = \sqrt[3]{\frac{x^2}{x+2}}$$

$$y = \frac{x^2\sqrt{x^2 - 1}}{2x^2 - 1}$$

$$y = \frac{\sin x}{\sin(x + \frac{\pi}{4})}$$

$$y = \frac{x}{2} + arctgx$$

$$y = \sin x + \cos^2 x$$