Intro to Base Graphics in *R*

MTH 3220

Contents

1 Creating Plots **2**

2 Enhancing the Appearance of Plots by Setting Graphical Parameters **7**

3 Enhancing the Appearance of Plots From Within the Plotting Function **8**

4 Adding to an Existing Plot **10**

5 Customizing the Axes in a Plot **12**

6 Three-Dimensional Graphics and Surface Plots **13**

6.0.1 Surface Plots 13

6.0.2 Three-Dimensional Scatterplots 14

Graphics functions can be used to create plots, add to existing plots, and enable users to interact with a graphics window by way of mouse or keyboard actions.

1 Creating Plots

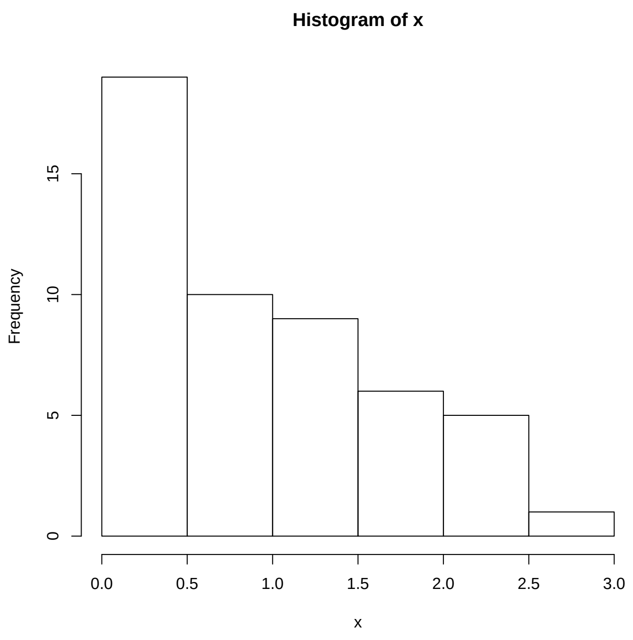
Here are some of the most commonly used functions to create plots:

```
hist()                  # Histogram
boxplot()               # Boxplot(s)
plot()                  # Scatterplot
stripchart()            # Dot plot, individual value plot
pairs()                 # Scatterplot matrix
curve()                 # Graph of a function
stem()                  # Stem and leaf plot
qqnorm()                # Normal probability plot (plot of sample vs
# theoretical quantiles)
barplot()               # Bar chart of given bar heights
pie()                   # Pie chart of given pie areas
```

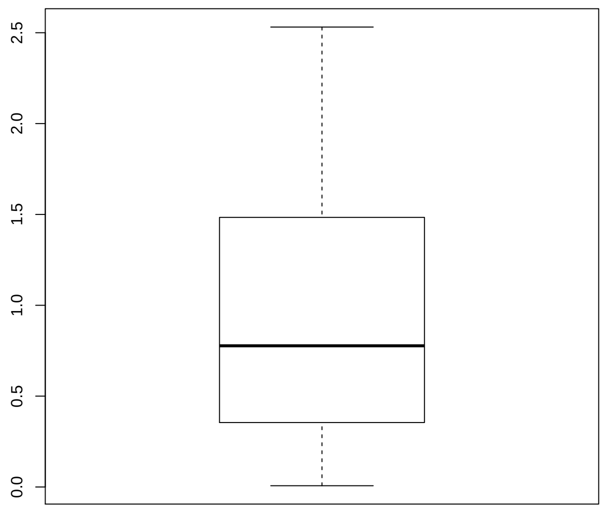
Several examples follow.

The functions `hist()` and `boxplot()` take *vector* arguments and produce the plots:

```
x <- rexp(n = 50, rate = 1)
hist(x)
```

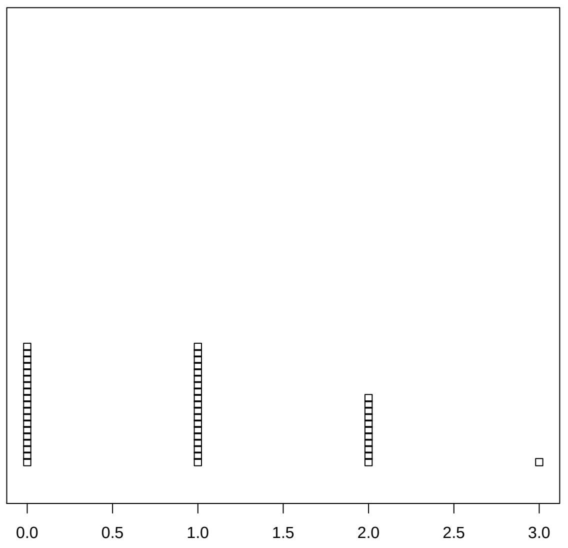


```
boxplot(x)
```



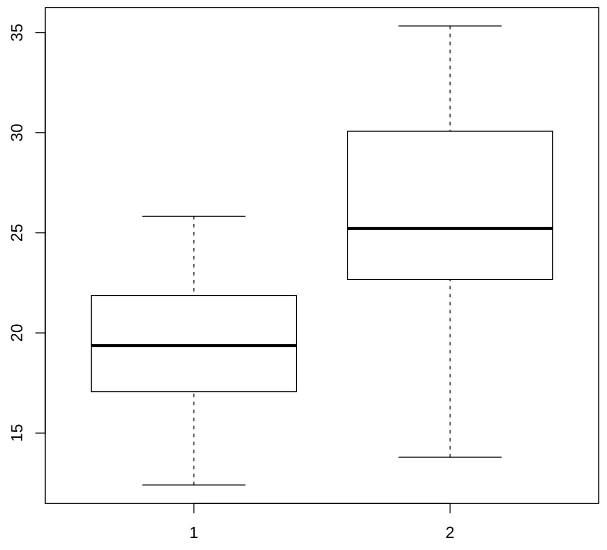
The function `stripchart()` will produce a dot plot if we specify `method = "stack"`:

```
stripchart(round(x), method = "stack", at = 0.1)
```



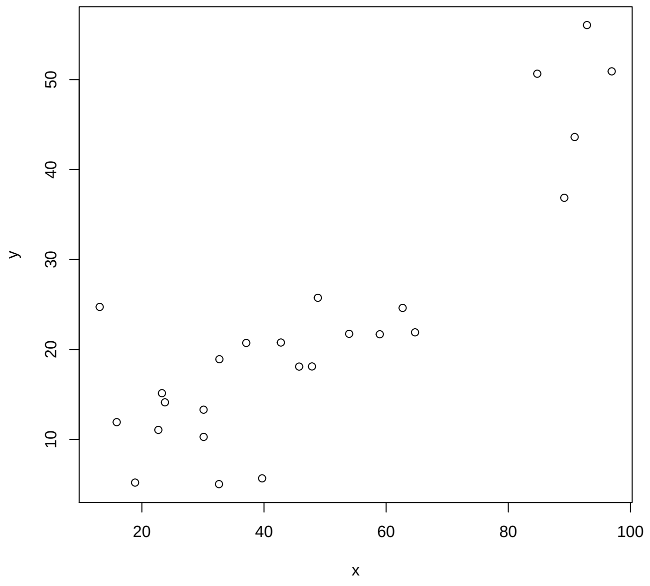
To produce boxplots side by side in the same graph, we can pass multiple *vectors* to `boxplot()`:

```r
x1 <- rnorm(n = 50, mean = 20, sd = 3)
x2 <- rnorm(n = 40, mean = 25, sd = 5)
boxplot(x1, x2)
```



To make a scatterplot, we bass two *vectors* to the `plot()` function via the arguments `x` and `y`. Here's an example:

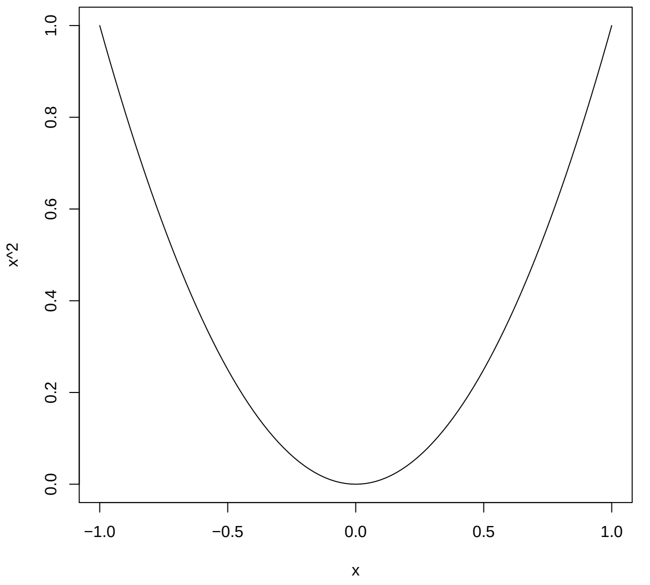
```r
x <- runif(n = 25, min = 10, max = 100)
y <- 0.5 * x + rnorm(n = 25, mean = 0, sd = 8)
plot(x, y)
```



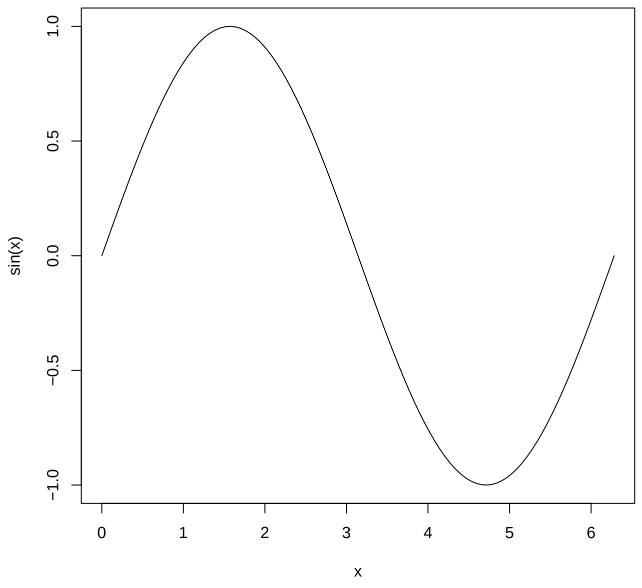
(We could also pass `plot()` a single vector `x`, in which case the values in `x` are plotted in order).

The `curve()` function takes as an argument either an expression, written as a function of `x`, or the name of an existing function, and graphs it over the range `from` to `to`:

```
curve(x^2, from = -1, to = 1)
```

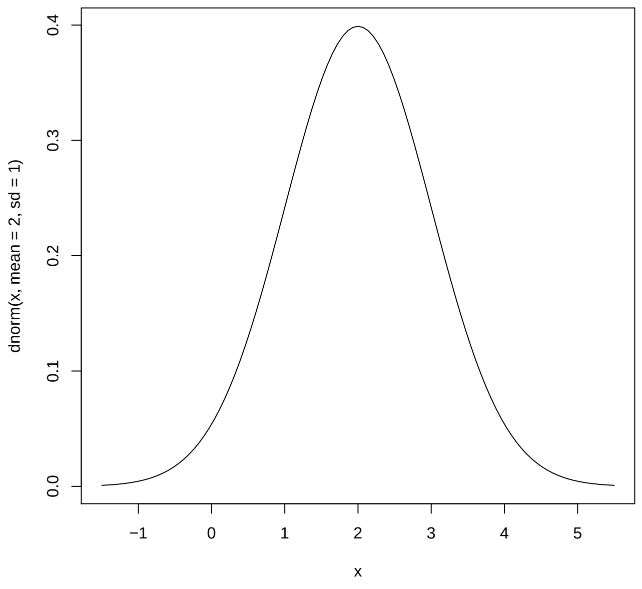


```r
curve(sin, from = 0, to = 2*pi)
```



Here's how to plot the normal density function:

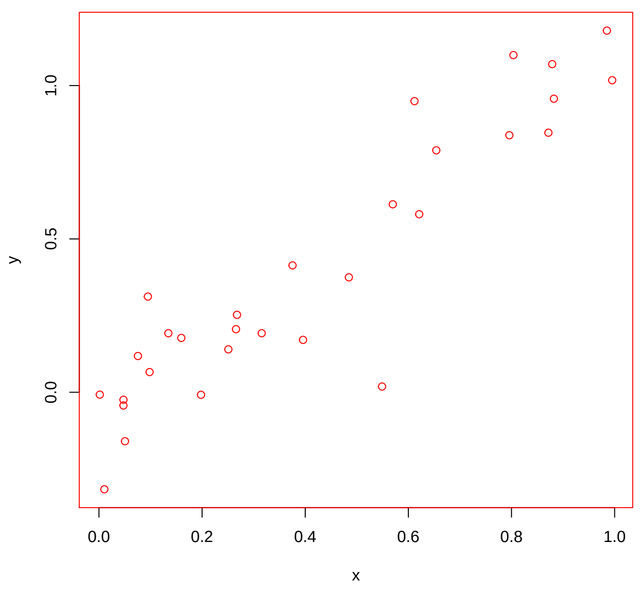
```r
curve(dnorm(x, mean = 2, sd = 1), from = -1.5, to = 5.5)
```



2 Enhancing the Appearance of Plots by Setting Graphical Parameters

A vast number of graphical parameters can be set prior to calling the plotting functions using the `par()` function. Each graphical parameter has a name (e.g. `col` for color) that can be set or reset to specific values (e.g. 'red') with `par()`. For example:

```
x <- runif(n = 30, min = 0, max = 1)
y <- x + rnorm(n = 30, mean = 0, sd = 0.2)
par(col = 'red')
plot(x, y)
```



To see a list of all the colors, run `colors()`.
Note that once a graphical parameter value has been set, that setting affects all subsequent plots until it is reset or the R session is terminated. To check the current parameter settings at any time, type `par()`.

To reset the `col` parameter back to `'black'`, just run

```
par(col = 'black')
```

A useful trick that saves an entire set of parameter values *before* setting any of them to new values is this:

```r
oldpar <- par(col = 'red')
```

The original parameter values are stored in the object `oldpar` before the `col` parameter is set to 'red'. Typing the following resets the `col` parameter back to its original value (`'black'`):

```r
par(oldpar)
```

Here are some of the graphical parameters that can be set by `par()`:

```
mfrow; mfcol    # Used to put figures in an nr by nc array (e.g.
                # par(mfrow = c(3, 2)) puts figures in an array with
                # 3 rows and 2 columns)
pch             # Plotting character (open circles, solid circles, etc.)
col             # Plotting color
lty             # Line type (solid, dashed, etc.)
lwd             # Line width
cex             # Amount by which characters (symbols, text, etc.)
                # should be expanded
bg              # Background color
fg              # Foreground color
xaxt; yaxt      # Axis type (par(xaxt = "n") and par(yaxt = "n")
                # suppress plotting of the x and y axes, respectively)
```

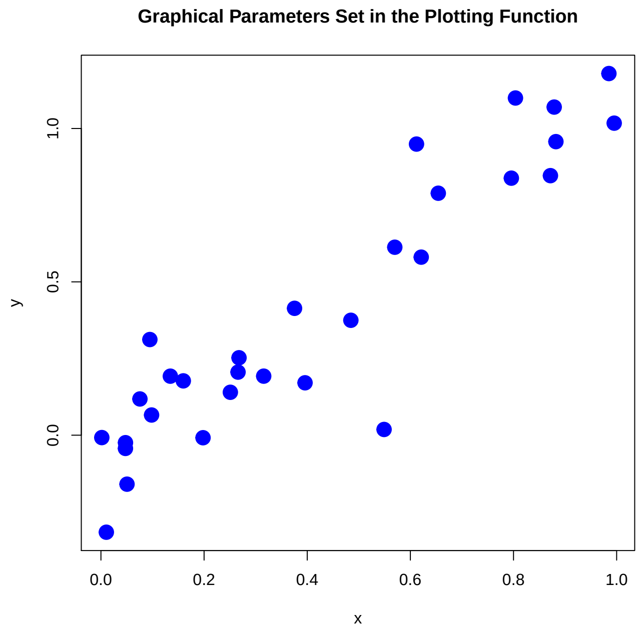
For the complete list, run `?par`.

3 Enhancing the Appearance of Plots From Within the Plotting Function

Most of the graphical parameters that can be set by `par()` can also be set from within the call to the plotting function. In this case, the parameter setting only affects the current plot, not subsequent ones.

Below, the graphical parameters pch, `col`, `cex` are set within the call to `plot()`. Note also the use of the argument `main` for adding a title:

```r
plot(x, y,
     main = "Graphical Parameters Set in the Plotting Function",
     pch = 19, col = "blue", cex = 2)
```



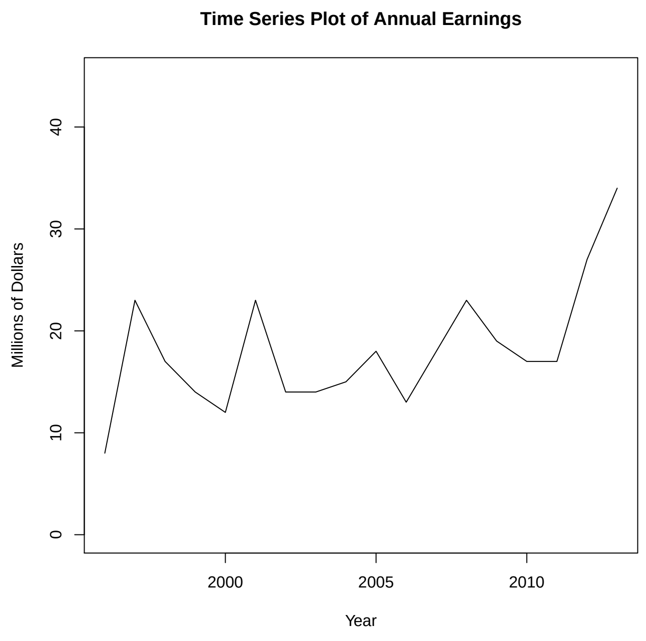
In addition to the graphical parameters that can be set by `par()`, most of the plotting functions accept the following arguments that add titles, label axes, and change the axis limits:

```
main                    # Main title for the plot
xlab; ylab              # Labels for the x and y axes
xlim; ylim              # Limits for the x and y axes in the plot
```

Specific plotting functions have their own unique arguments that enhance the appearance of the plots, and are listed in the help files for those functions.

Here's an example example in which the argument `type` is used in `plot()` to make a time-series plot:

```
earnings <- c(8, 23, 17, 14, 12, 23, 14, 14, 15,
              18, 13, 18, 23, 19, 17, 17, 27, 34)
plot(x = 1996:2013, y = earnings, type = "l", ylim = c(0, 45),
     main = "Time Series Plot of Annual Earnings", xlab = "Year",
     ylab = "Millions of Dollars")
```



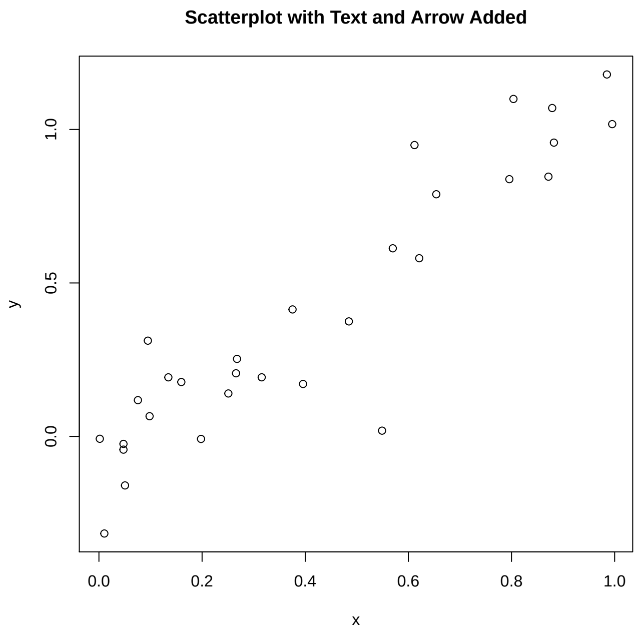
4 Adding to an Existing Plot

Several functions are available for adding to an existing plot. Here are some of them:

```
points()          # Add points to the plot at the specified locations
symbols()         # Add various symbols to the plot
abline()          # Add a line to the plot with given slope and intercept
lines()           # Add a line to the plot connecting specified coordinates
segments()        # Add line segments to the plot between pairs of points
qqline()          # Add a line to a normal probability plot
arrows()          # Draw an arrow in the plot
title()           # Add a main title to the plot
text()            # Add text to the plot at a given set of coordinates
mtext()           # Add text in a margin of the plot
legend()          # Add a legend
axis()            # Add an axis with specified labels at specified points
                  # (see below)
rect()            # Draw a rectangle in plot at a given set of coordinates
polygon()         # Draw a polygon in plot with a given set of vertices
```

Here's an example in which the `text()` and `arrows()` functions are used to add text and an arrow to an existing plot. The symbol `\n` within the quoted text tells R to go to the next line:

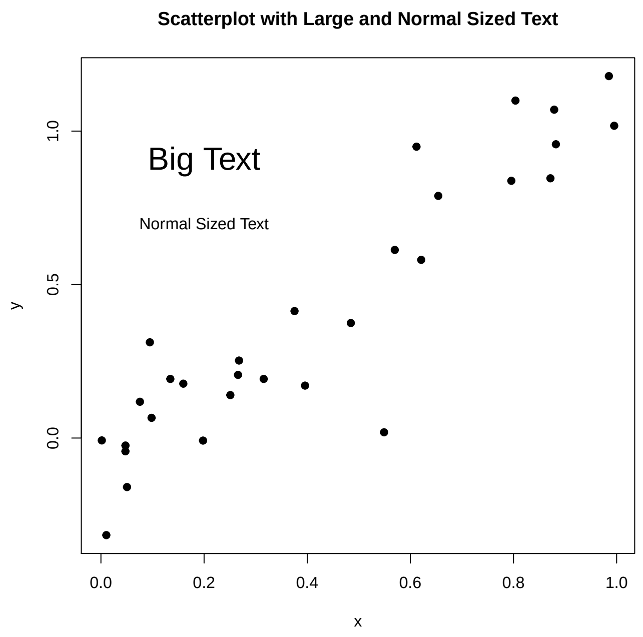
```r
plot(x, y, main = "Scatterplot with Text and Arrow Added")
text(x = 25, y = 50, labels = "Here's Some Text \n and an Arrow")
arrows(x0 = 40, y0 = 50, x1 = 65, y1 = 40)
```



Many of the graphical parameters that can be set by `par()` can also be set within a call to a function that adds to an existing plot.

Below, the character expansion graphical parameter `cex` is set within calls to `text()`. Note the use of `title()` to add a main title to an existing plot:

```r
plot(x, y, pch = 19)
title(main = "Scatterplot with Large and Normal Sized Text")
text(x = 0.2, y = 0.9, labels = "Big Text", cex = 2)
text(x = 0.2, y = 0.7, labels = "Normal Sized Text", cex = 1)
```

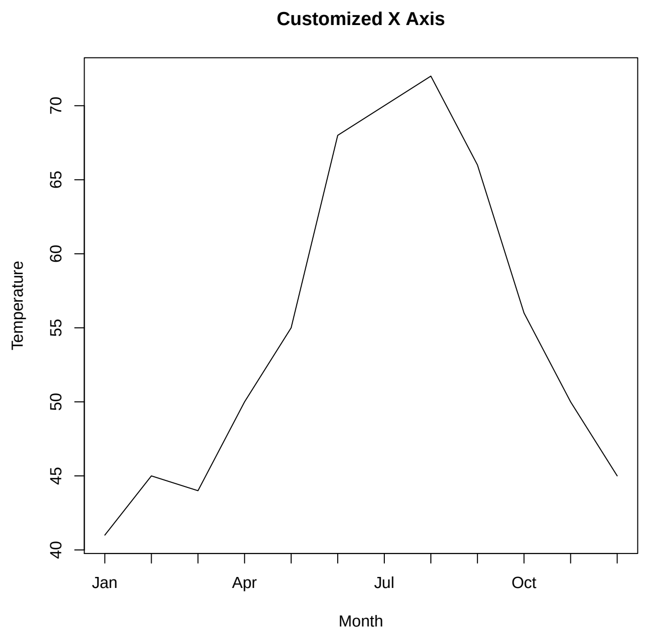


5 Customizing the Axes in a Plot

We've seen that the `xlim`, `ylim`, `xlab`, and `ylab` arguments control some aspects of the x and y axes in a plot.

Sometimes we need to customize the locations and labels of tick marks on the axes. To do so, we suppress the plotting of axes in the call to the plotting function by passing the arguments `xaxt = "n"` or `yaxt = "n"`, and then add customized axes using `axis()`. Here's an example:

```r
Month <- 1:12
Temperature <- c(41, 45, 44, 50, 55, 68, 70, 72, 66, 56, 50, 45)
plot(Month, Temperature, type = "l", xaxt = "n",
     main = "Customized X Axis")
axis(side = 1, at = 1:12,
     labels = c("Jan", "", "", "Apr", "", "", "Jul", "", "", "Oct", "", ""))
```



6 Three-Dimensional Graphics and Surface Plots

6.0.1 Surface Plots

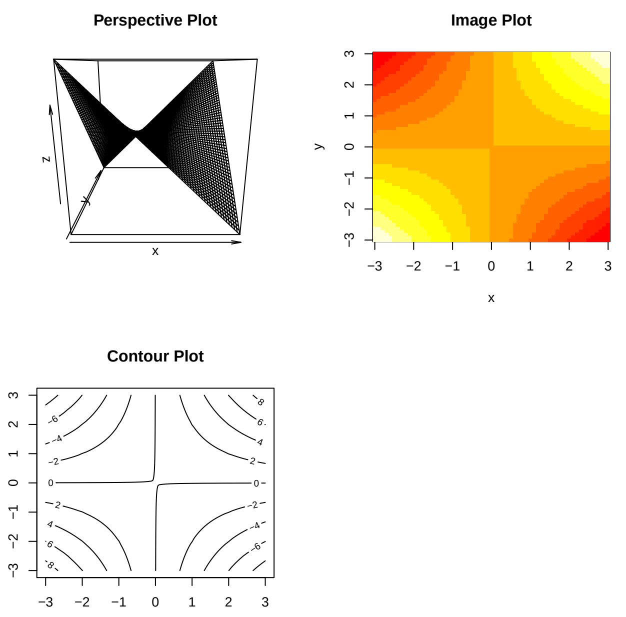
Here are some useful functions for making three-dimensional plots and images of surfaces over two-dimensional domains:

```
persp()                 # Graph a surface over the x,y plane. You can use
                        # outer() to generate values of the surface over
                        # a grid of x,y pairs.
image()                 # Plot a grid of colored rectangles corresponding to
                        # the value of a function over the x,y plane.
contour()               # Make a contour plot or add contour lines to an
                        # existing plot.
filled.contour()        # Make a contour plot and fill with colors
```

Below, we make graphs of the function $f(x, y) = xy$ over the range $-3 \leq x \leq 3$, $-3, \leq y \leq 3$ (note the use of `par(mfrow = c(2, 2))`):

```
x <- seq(from = -3, to = 3, by = 0.1)
y <- seq(from = -3, to = 3, by = 0.1)
z <- outer(X = x,Y = y, FUN = '*')
oldpar <- par(mfrow = c(2,2))
```

```r
persp(x, y, z, main = "Perspective Plot")
image(x, y, z, main = "Image Plot")
contour(x, y, z, main = "Contour Plot")
par(oldpar)
```



6.0.2 Three-Dimensional Scatterplots

The package `scatterplot3d` has a function `scatterplot3d()` that will produce three-dimensional scatterplots. To download and install the texttt scatterplot3d package, from the Packages pulldown menu, select Install Package(s), choose a CRAN mirror nearby (e.g. Iowa State), then select `scatterplot3d` from the list. Once the package is installed, it can be loaded into the current R session by typing `library(scatterplot3d)`.

Content created by Dr. Grevstad.