

Plotting Tools

Statistical Computing

Last week: Data frames and apply

- Data frames are a representation of the “classic” data table in R: rows are observations/cases, columns are variables/features
- Each column can be a different data type (but must be the same length)
- `subset()`: function for extracting rows of a data frame meeting a condition
- `split()`: function for splitting up rows of a data frame, according to a factor variable
- `apply()`: function for applying a given routine to rows or columns of a matrix or data frame
- `lapply()`: similar, but used for applying a routine to elements of a vector or list
- `sapply()`: similar, but will try to simplify the return type, in comparison to `lapply()`
- `tapply()`: function for applying a given routine to groups of elements in a vector or list, according to a factor variable

Part I

Plot basics

Plotting in R

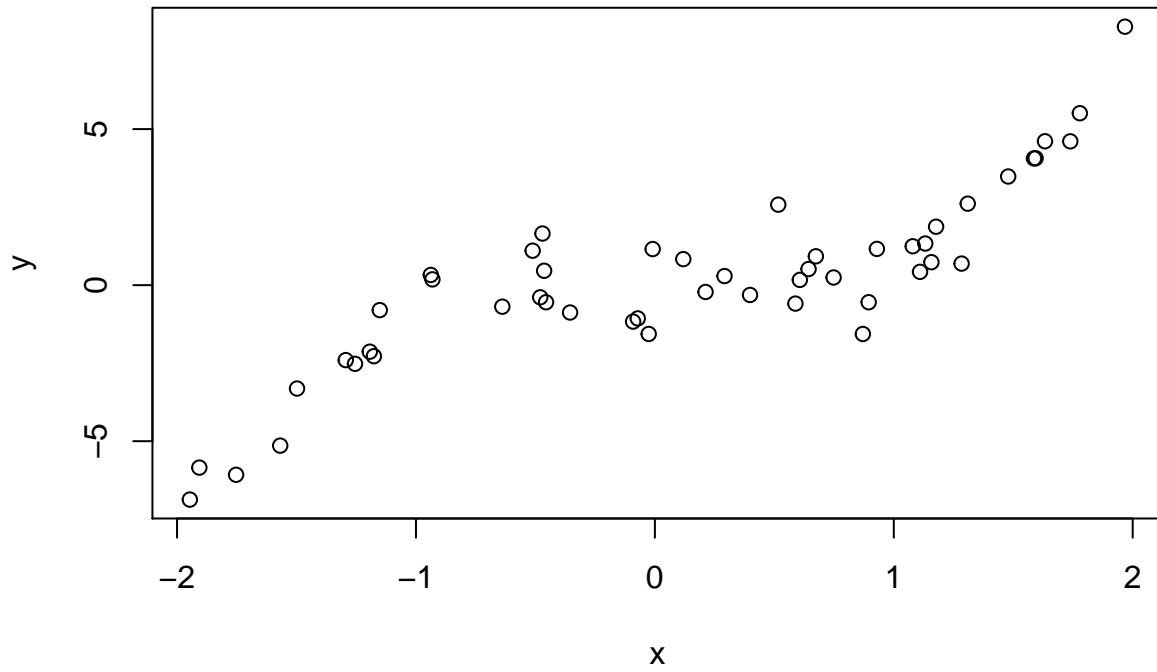
Base R has a set of powerful plotting tools. An overview:

- `plot()`: generic plotting function
- `points()`: add points to an existing plot
- `lines()`, `abline()`: add lines to an existing plot
- `text()`, `legend()`: add text to an existing plot
- `rect()`, `polygon()`: add shapes to an existing plot
- `hist()`, `image()`: histogram and heatmap
- `heat.colors()`, `topo.colors()`, etc: create a color vector
- `density()`: estimate density, which can be plotted
- `contour()`: draw contours, or add to existing plot
- `curve()`: draw a curve, or add to existing plot

Scatter plot

To make a scatter plot of one variable versus another, use `plot()`

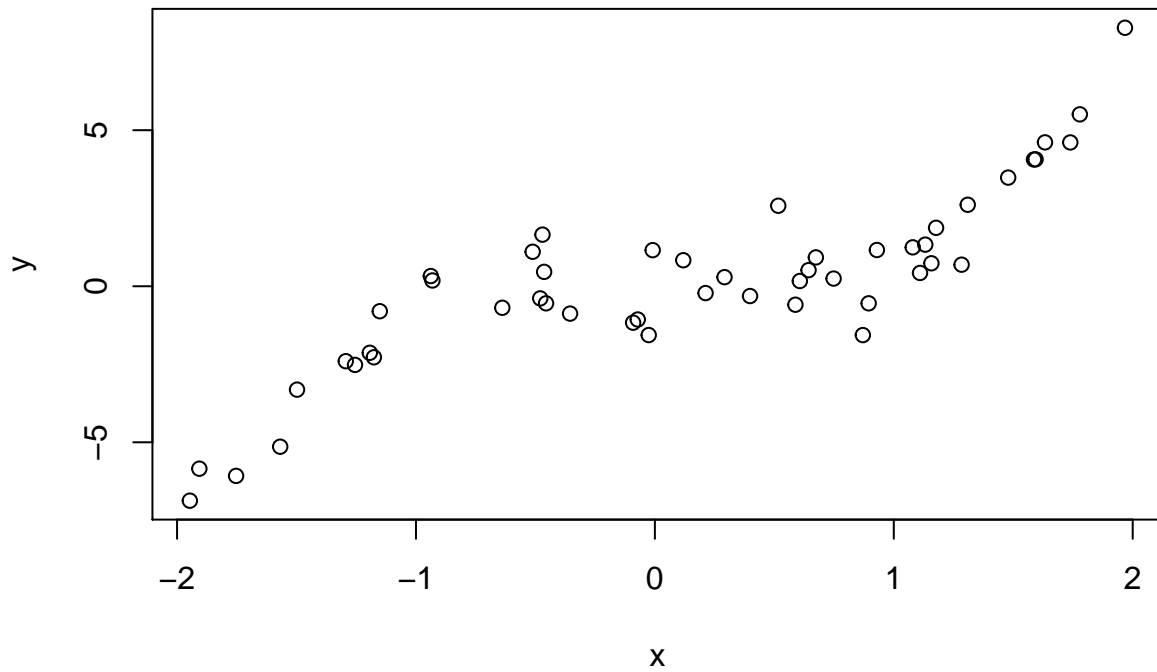
```
n = 50
set.seed(0)
x = sort(runif(n, min=-2, max=2))
y = x^3 + rnorm(n)
plot(x, y)
```



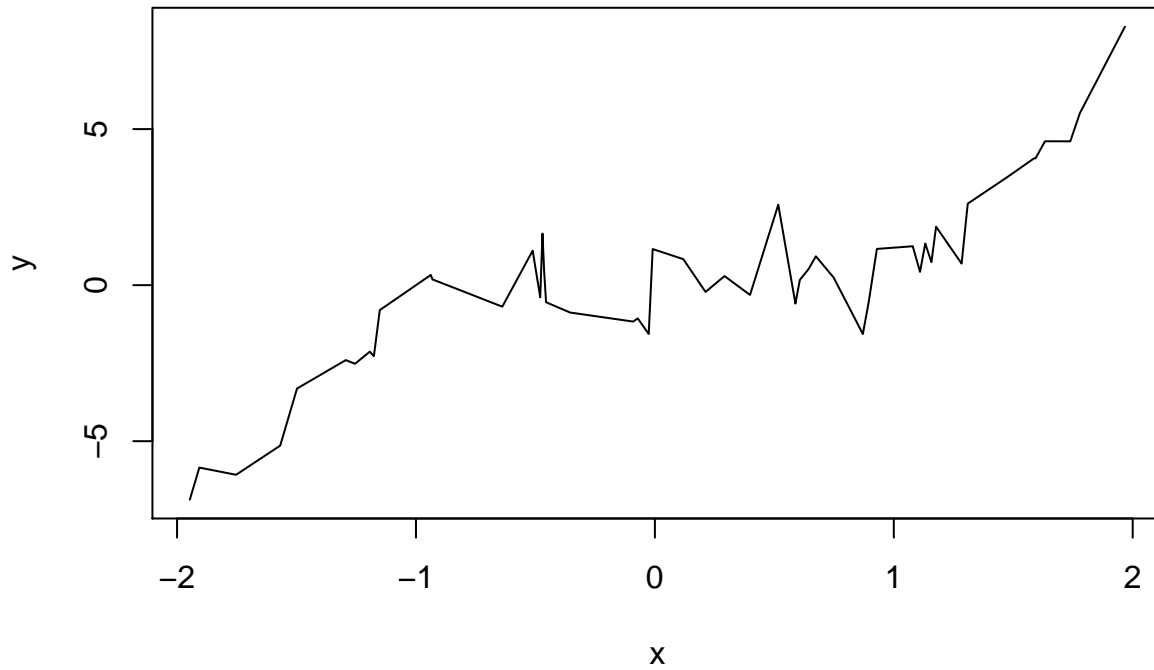
Plot type

The `type` argument controls the plot type. Default is `p` for points; set it to `l` for lines

```
plot(x, y, type="p")
```



```
plot(x, y, type="l")
```



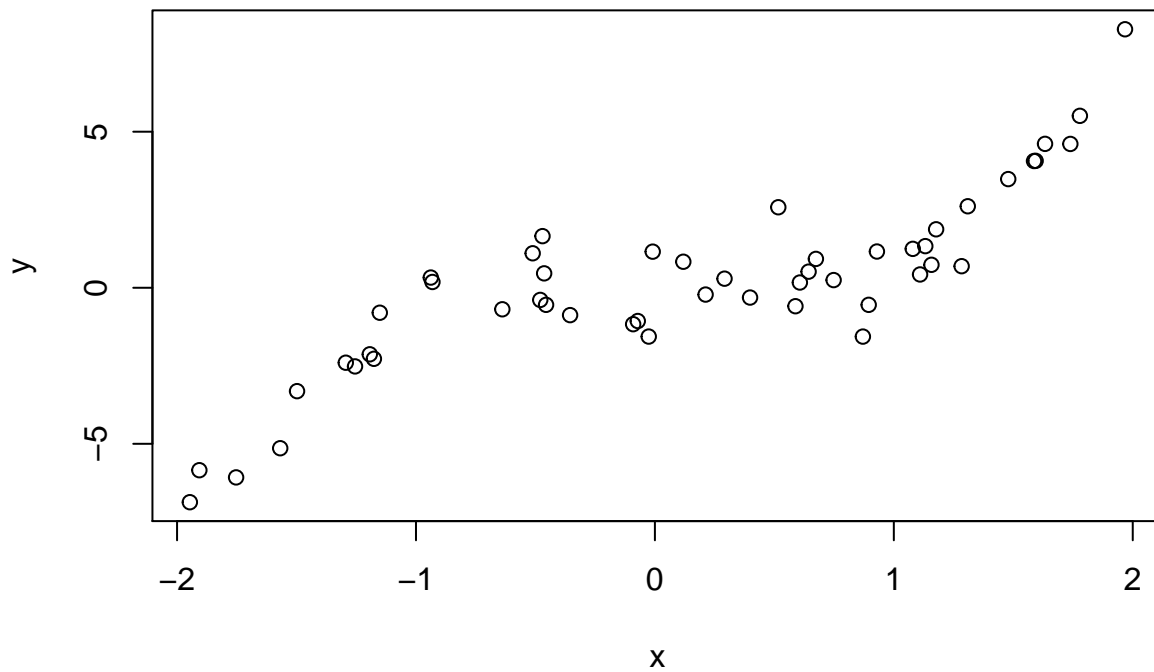
Try also `b` or `o`, for both points and lines

Labels

The `main` argument controls the title; `xlab` and `ylab` are the x and y labels

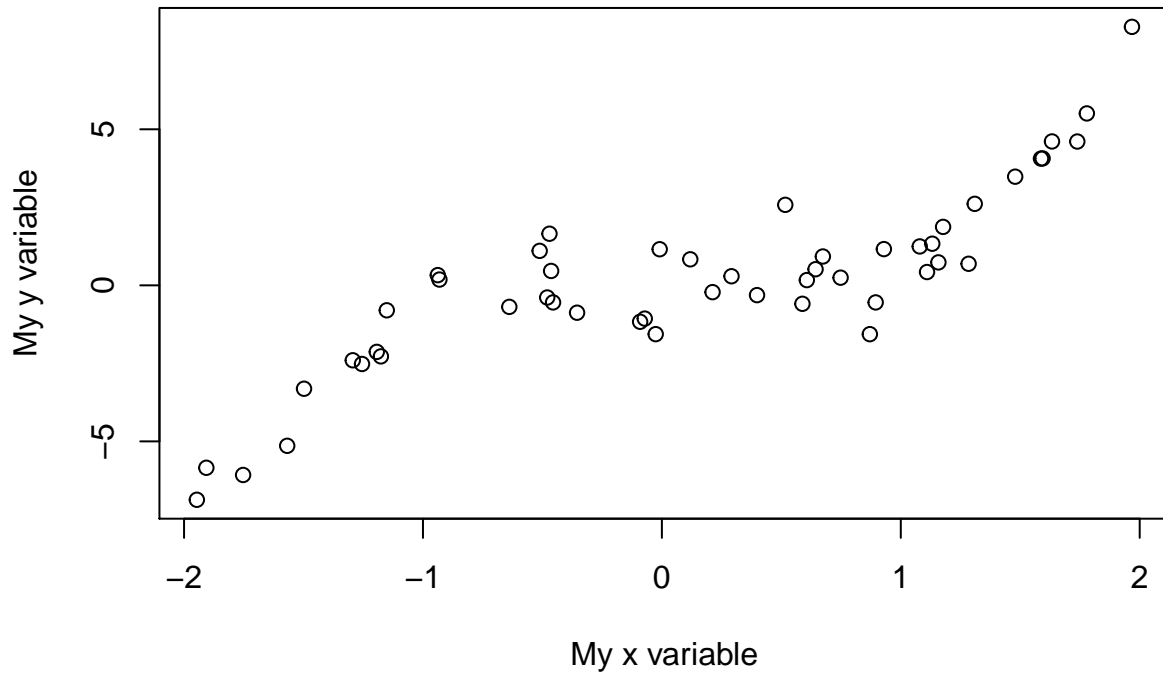
```
plot(x, y, main="A noisy cubic") # Note the default x and y labels
```

A noisy cubic



```
plot(x, y, main="A noisy cubic", xlab="My x variable", ylab="My y variable")
```

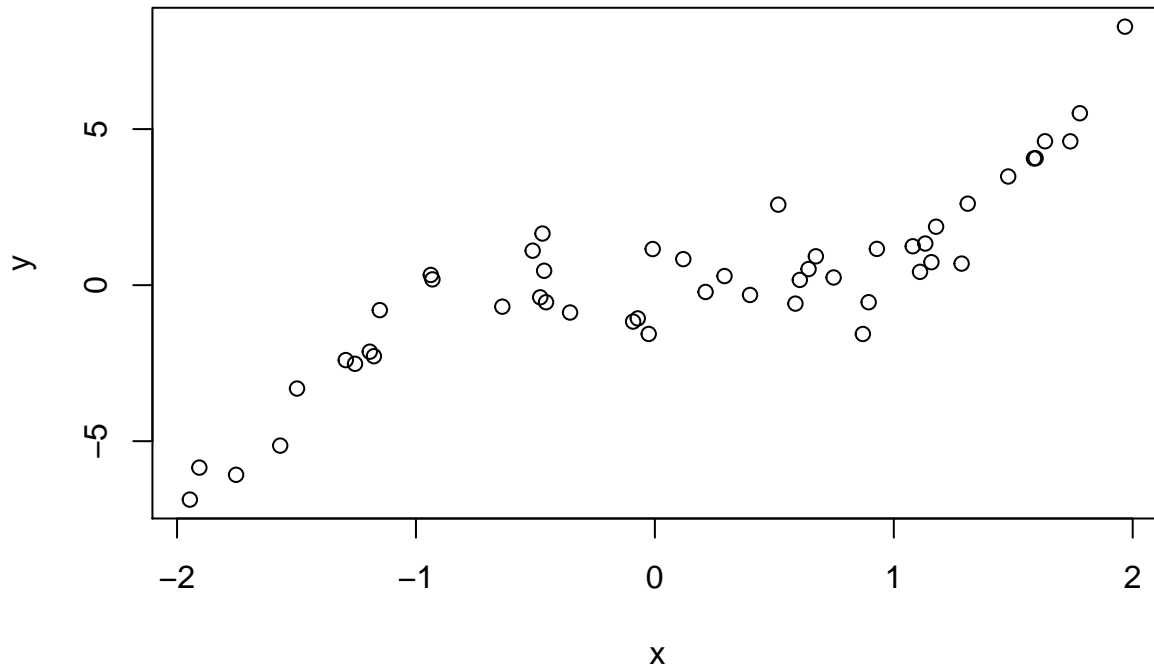
A noisy cubic



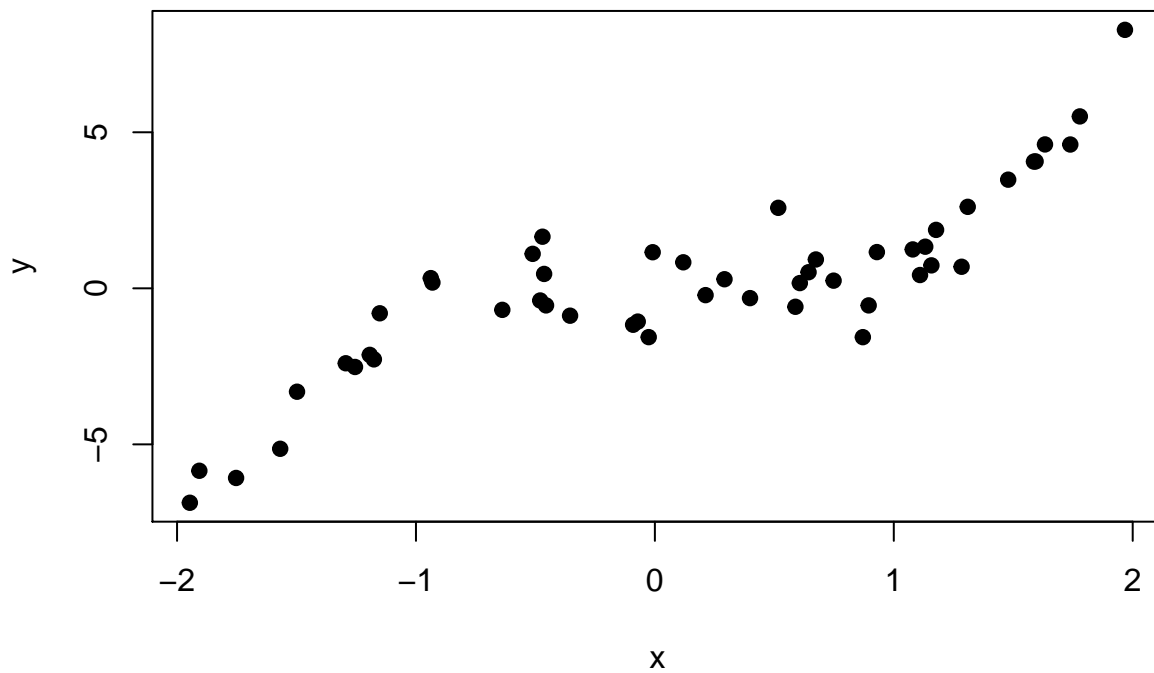
Point type

Use the `pch` argument to control point type

```
plot(x, y, pch=21) # Empty circles, default
```



```
plot(x, y, pch=19) # Filled circles
```

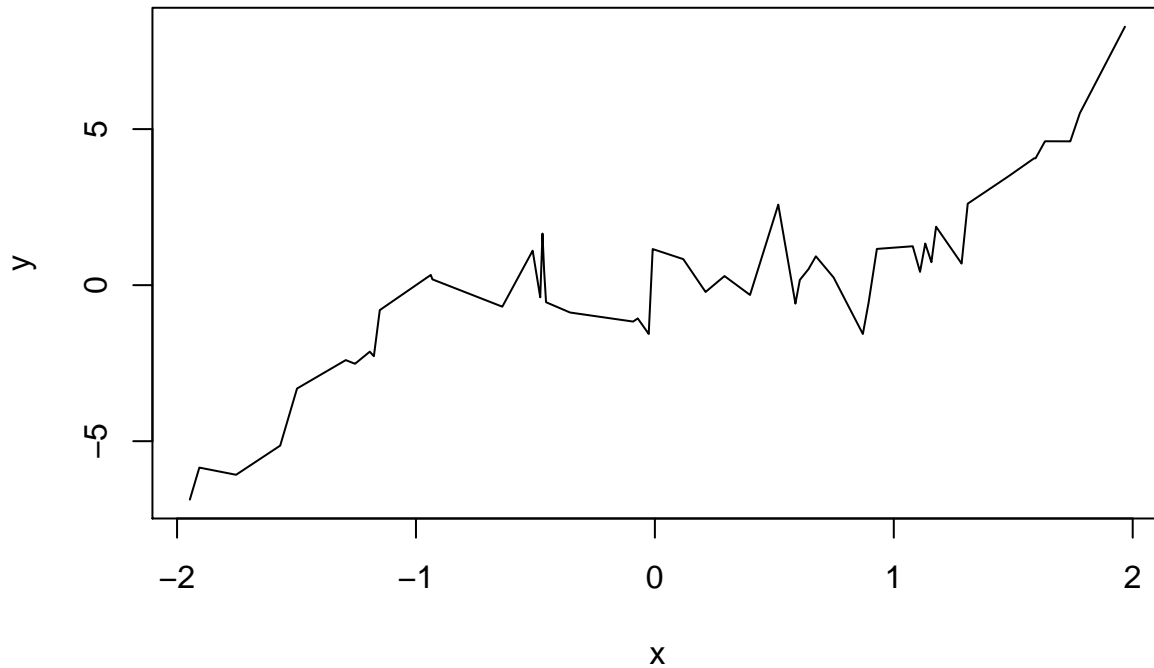


Try also 20 for small filled circles, or "." for single pixels

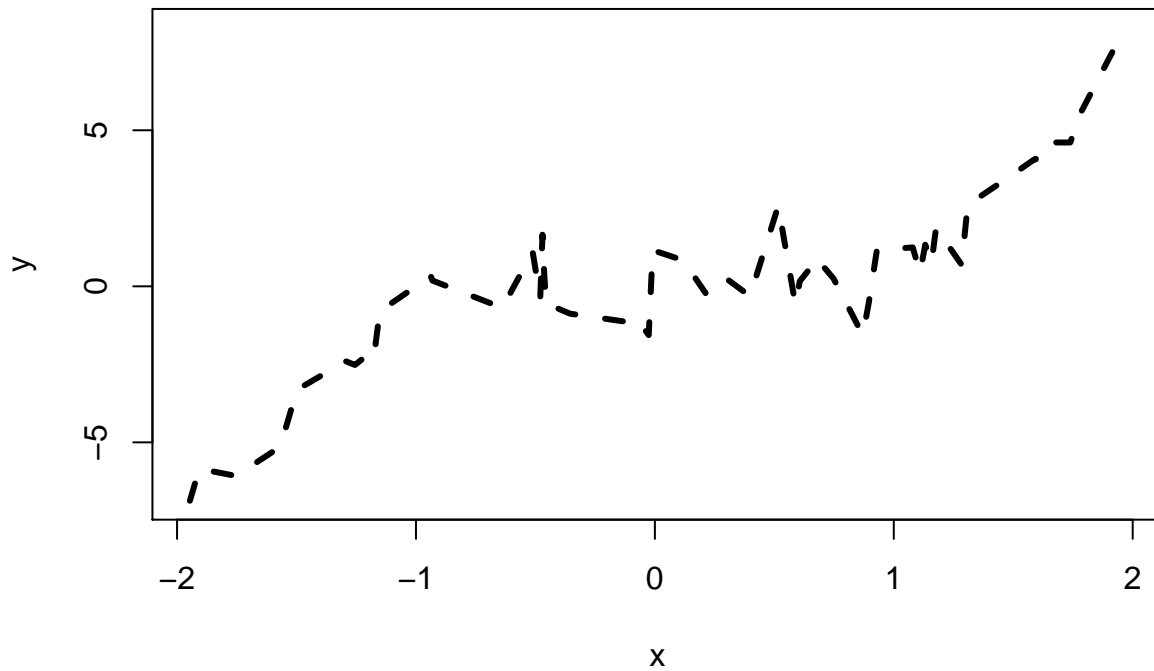
Line type

Use the `lty` argument to control the line type, and `lwd` to control the line width

```
plot(x, y, type="l", lty=1, lwd=1) # Solid line, default width
```



```
plot(x, y, type="l", lty=2, lwd=3) # Dashed line, 3 times as thick
```



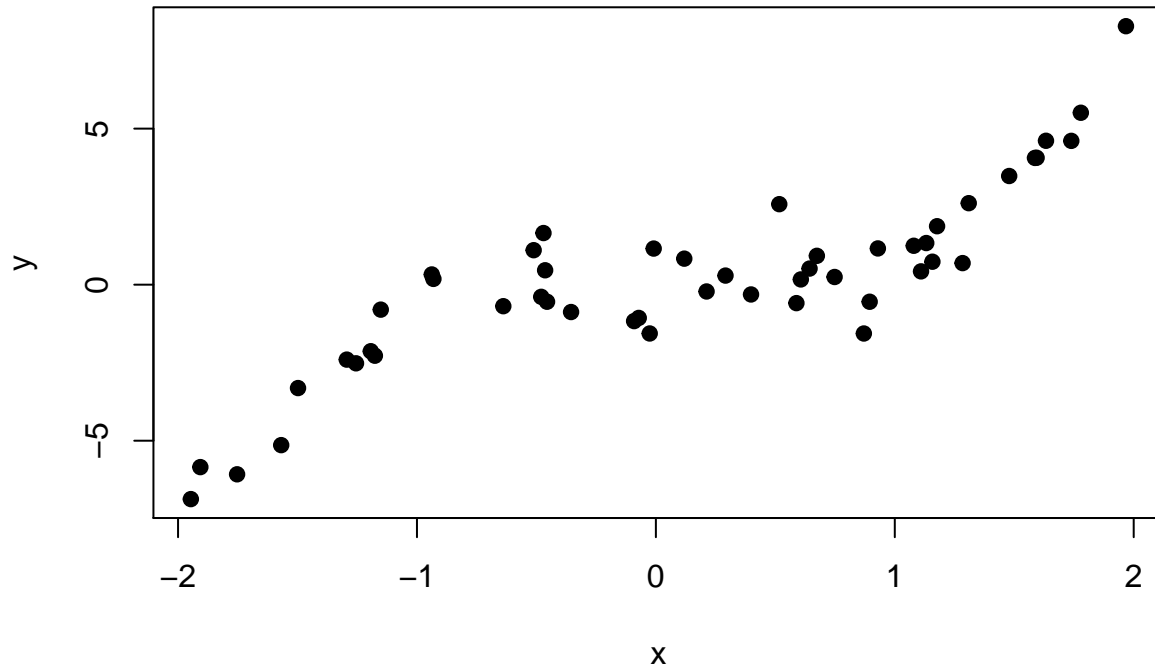
Color

Use the `col` argument to control the color. Can be:

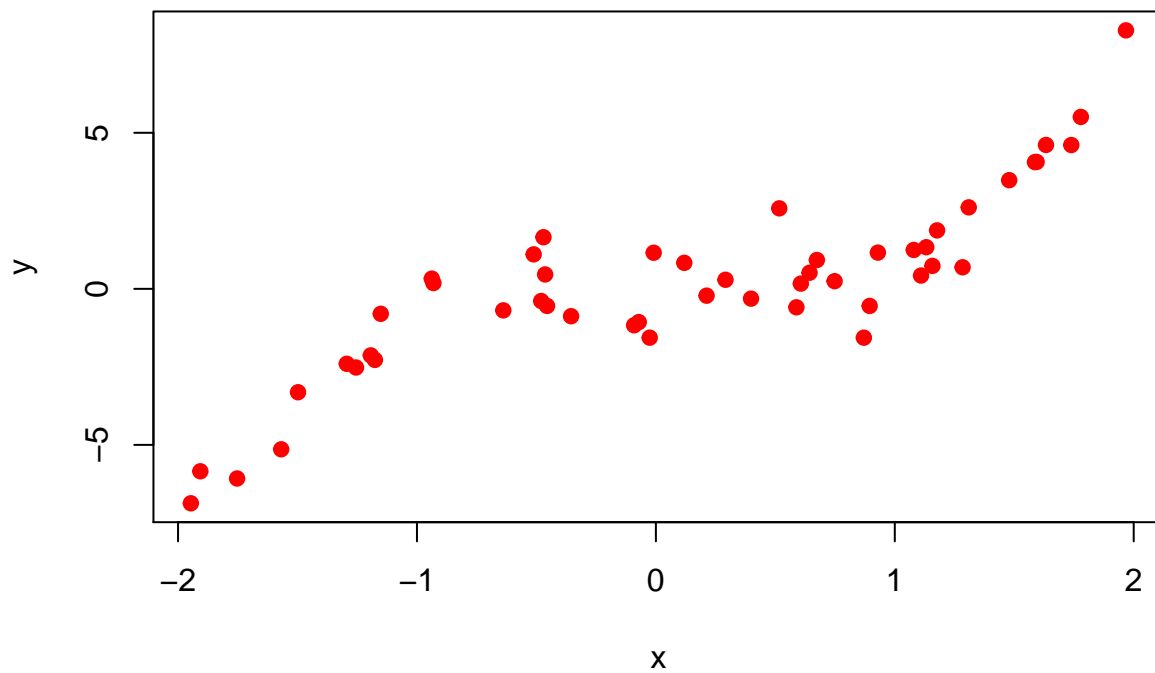
- An integer between 1 and 8 for basic colors
- A string for any of the 657 available named colors

The function `colors()` returns a string vector of the available colors

```
plot(x, y, pch=19, col=1) # Black, default
```



```
plot(x, y, pch=19, col=2) # Red
```



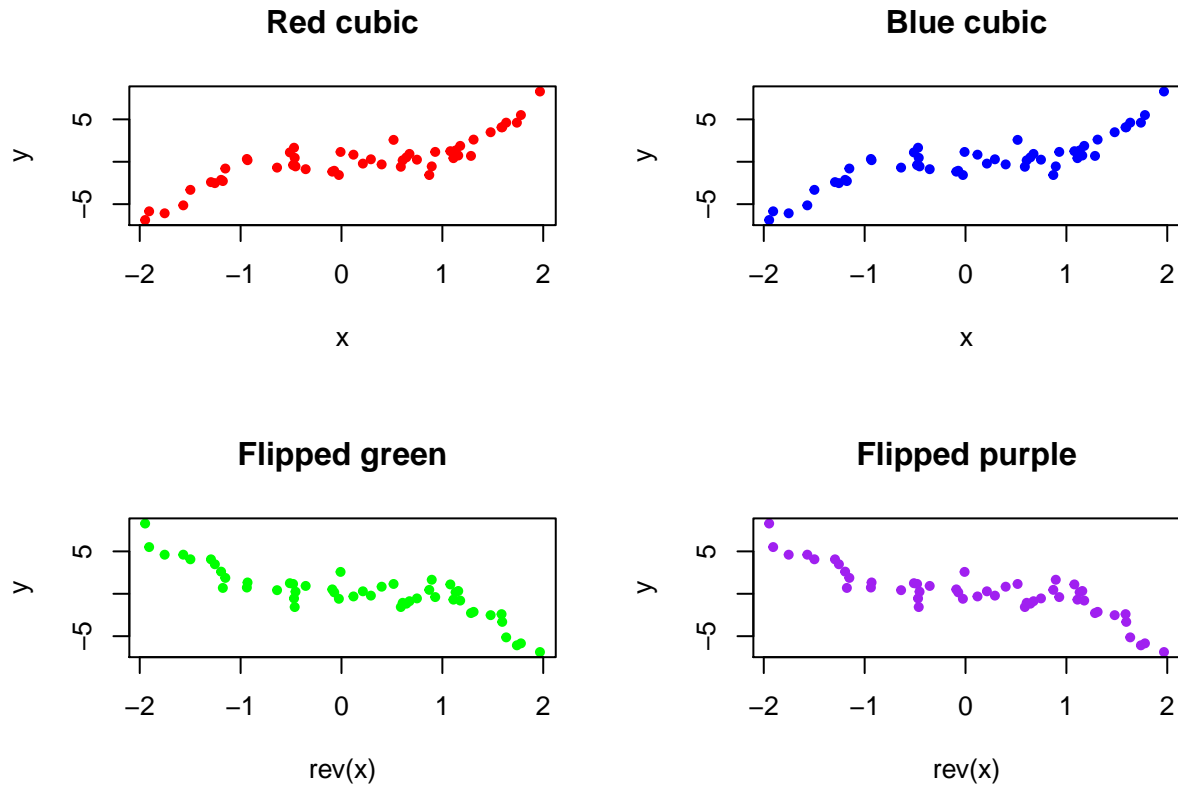
Multiple plots

To set up a plotting grid of arbitrary dimension, use the `par()` function, with the argument `mfrow`. Note: in general this will affect all following plots! (Except in separate R Markdown code chunks ...)

```

par(mfrow=c(2,2)) # Grid elements are filled by row
plot(x, y, main="Red cubic", pch=20, col="red")
plot(x, y, main="Blue cubic", pch=20, col="blue")
plot(rev(x), y, main="Flipped green", pch=20, col="green")
plot(rev(x), y, main="Flipped purple", pch=20, col="purple")

```



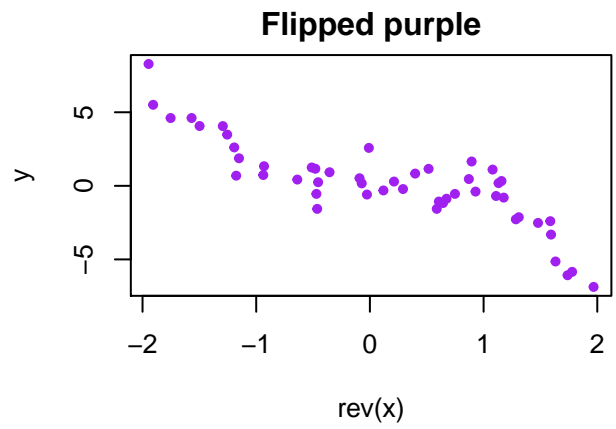
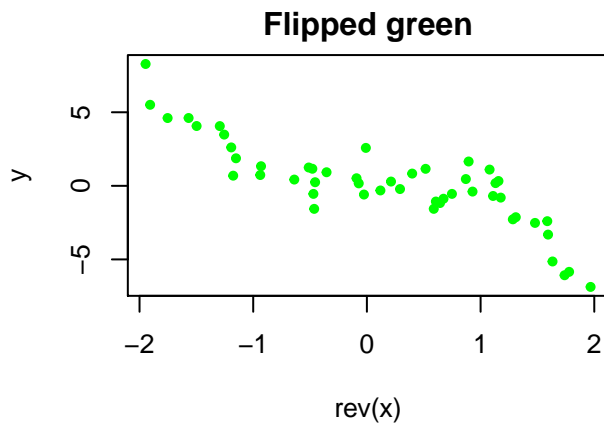
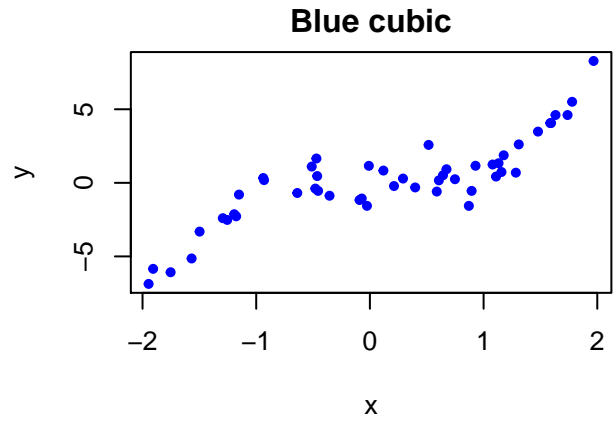
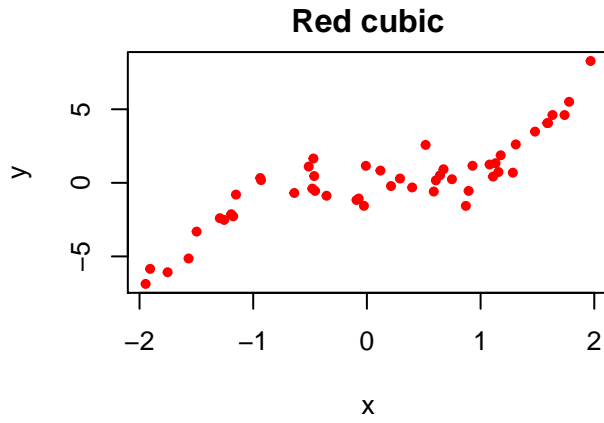
Margin

Default margins in R are large (and ugly); to change them, use the `par()` function, with the argument `mar`. Note: in general this will affect all following plots! (Except in separate R Markdown code chunks ...)

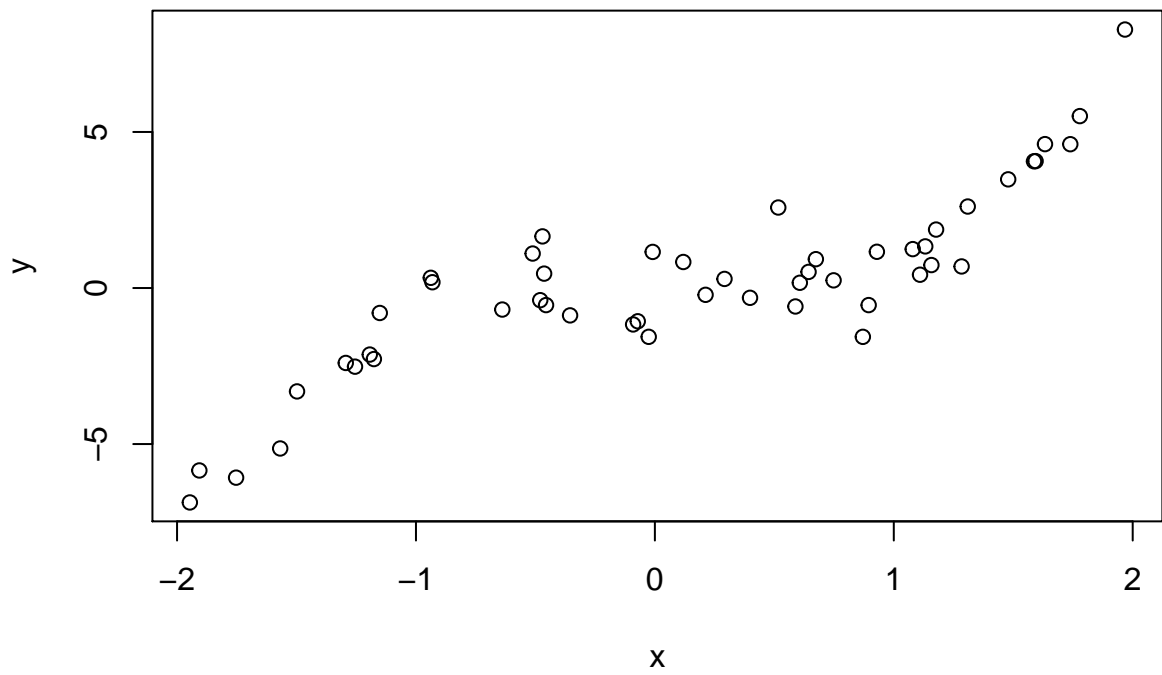
```

par(mfrow=c(2,2), mar=c(4,4,2,0.5))
plot(x, y, main="Red cubic", pch=20, col="red")
plot(x, y, main="Blue cubic", pch=20, col="blue")
plot(rev(x), y, main="Flipped green", pch=20, col="green")
plot(rev(x), y, main="Flipped purple", pch=20, col="purple")

```



Evidence that par() does not carry over to separate R Markdown code chunks
`plot(x, y)`



Saving plots

Use the `pdf()` function to save a pdf file of your plot, in your R working directory. Use `getwd()` to get the working directory, and `setwd()` to set it

```
getwd() # This is where the pdf will be saved

## [1] "/Users/lili/Dropbox/Courses/Courses I teach/Statistical computing/My notes/notes"

pdf(file="noisy_cubics.pdf", height=7, width=7) # Height, width are in inches
par(mfrow=c(2,2), mar=c(4,4,2,0.5))
plot(x, y, main="Red cubic", pch=20, col="red")
plot(x, y, main="Blue cubic", pch=20, col="blue")
plot(rev(x), y, main="Flipped green", pch=20, col="green")
plot(rev(x), y, main="Flipped purple", pch=20, col="purple")
graphics.off()
```

Also, use the `jpg()` and `png()` functions to save jpg and png files

Adding to plots

The main tools for this are:

- `points()`: add points to an existing plot
- `lines()`, `abline()`: add lines to an existing plot
- `text()`, `legend()`: add text to an existing plot
- `rect()`, `polygon()`: add shapes to an existing plot

You'll get practice with this on lab. Pay attention to **layers**—they work just like they would if you were painting a picture by hand

Part II

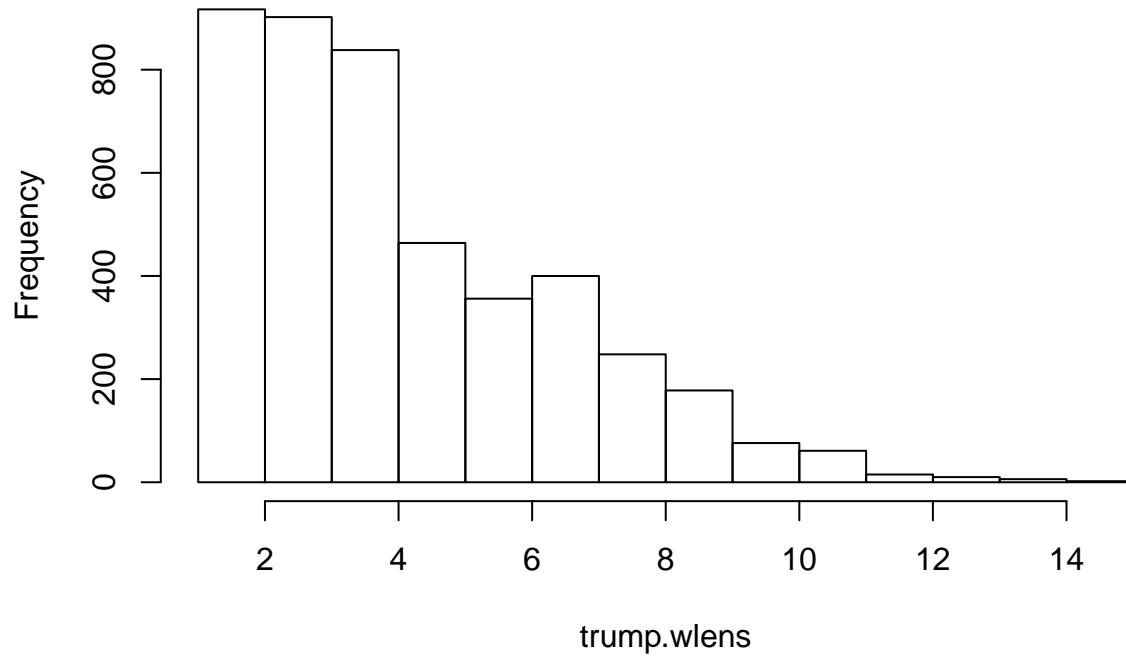
Histograms and heatmaps

Plotting a histogram

To plot a histogram of a numeric vector, use `hist()`

```
trump.lines =
  readLines("http://www.stat.cmu.edu/~ryantibs/statcomp/data/trump.txt")
trump.words = strsplit(paste(trump.lines, collapse=" "),
  split="[:,space:]|[:,punct:]",)[[1]]
#[:,space:] means 1 or more whitespace
#[:,punct:] means 1 or more punctuation
trump.words = tolower(trump.words[trump.words != ""])
#tolower(): Convert texts or tokens to lower case
#toupper(): Convert texts or tokens to upper case
trump.wlens = nchar(trump.words)
#nchar: number of characters in each word
hist(trump.wlens)
```

Histogram of trump.wlens

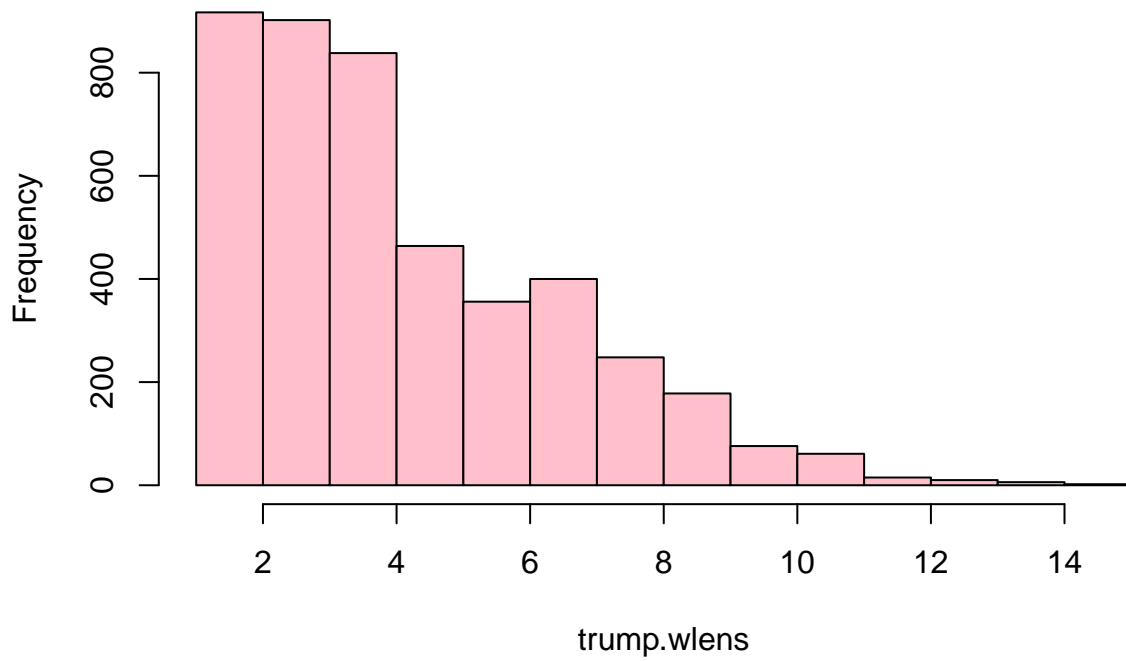


Histogram options

Several options are available as arguments to `hist()`, such as `col`, `freq`, `breaks`, `xlab`, `ylab`, `main`

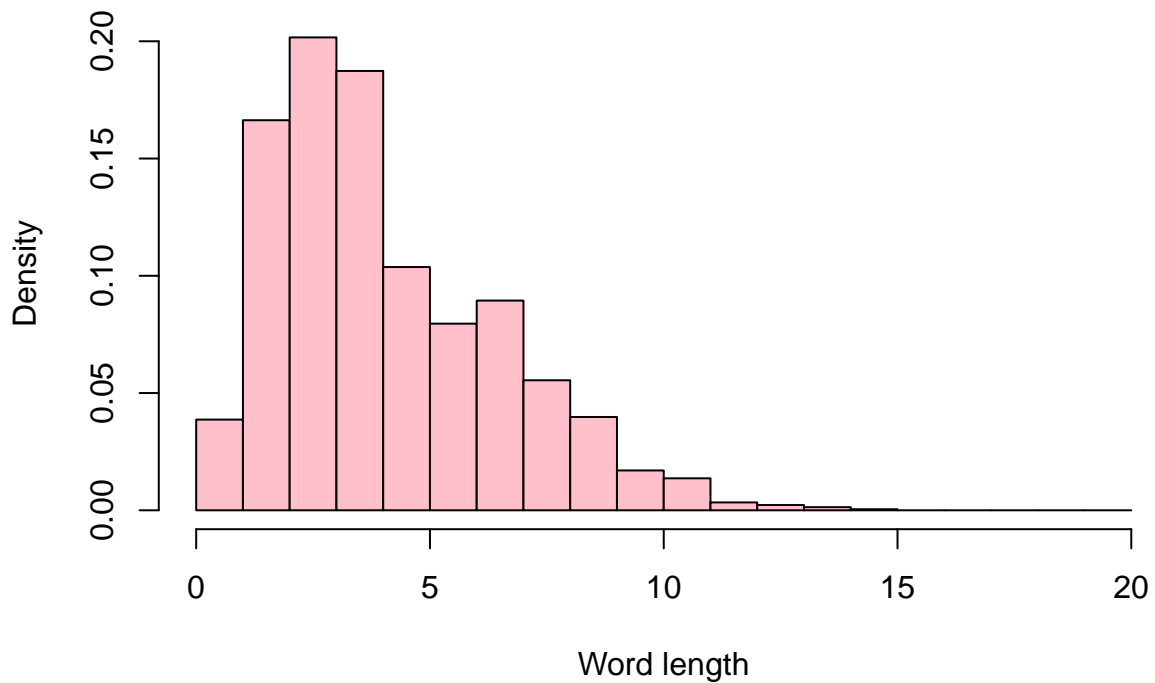
```
hist(trump.wlens, col="pink", freq=TRUE) # Frequency scale, default
```

Histogram of trump.wlens



```
hist(trump.wlens, col="pink", freq=FALSE, # Probability scale, and more options  
     breaks=0:20, xlab="Word length", main="Trump word lengths")
```

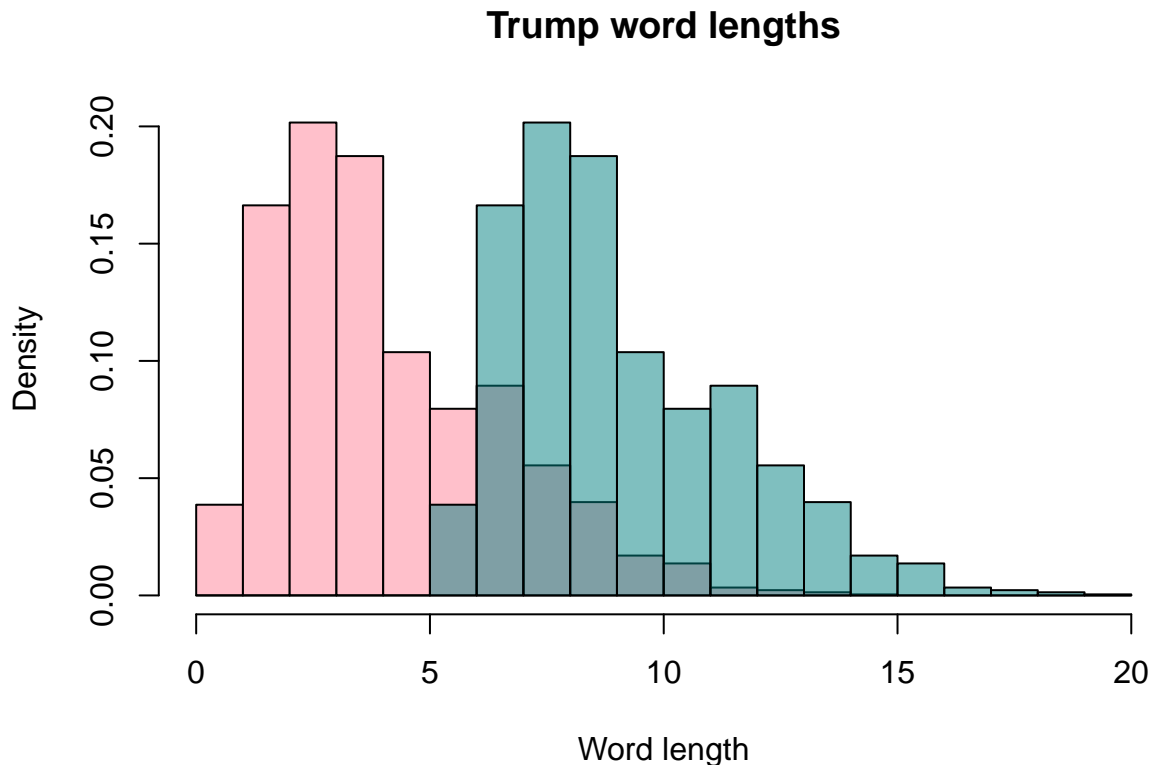
Trump word lengths



Adding a histogram to an existing plot

To add a histogram to an existing plot (say, another histogram), use `hist()` with `add=TRUE`

```
hist(trump.wlens, col="pink", freq=FALSE, breaks=0:20,  
     xlab="Word length", main="Trump word lengths")  
hist(trump.wlens + 5, col=rgb(0,0.5,0.5,0.5), # Note: using a transparent color  
     freq=FALSE, breaks=0:20, add=TRUE)
```



Adding a density curve to a histogram

To estimate a density from a numeric vector, use `density()`. This returns a list; it has components `x` and `y`, so we can actually call `lines()` directly on the returned object

```
density.est = density(trump.wlens, adjust=1.5) # 1.5 times the default bandwidth  
class(density.est)
```

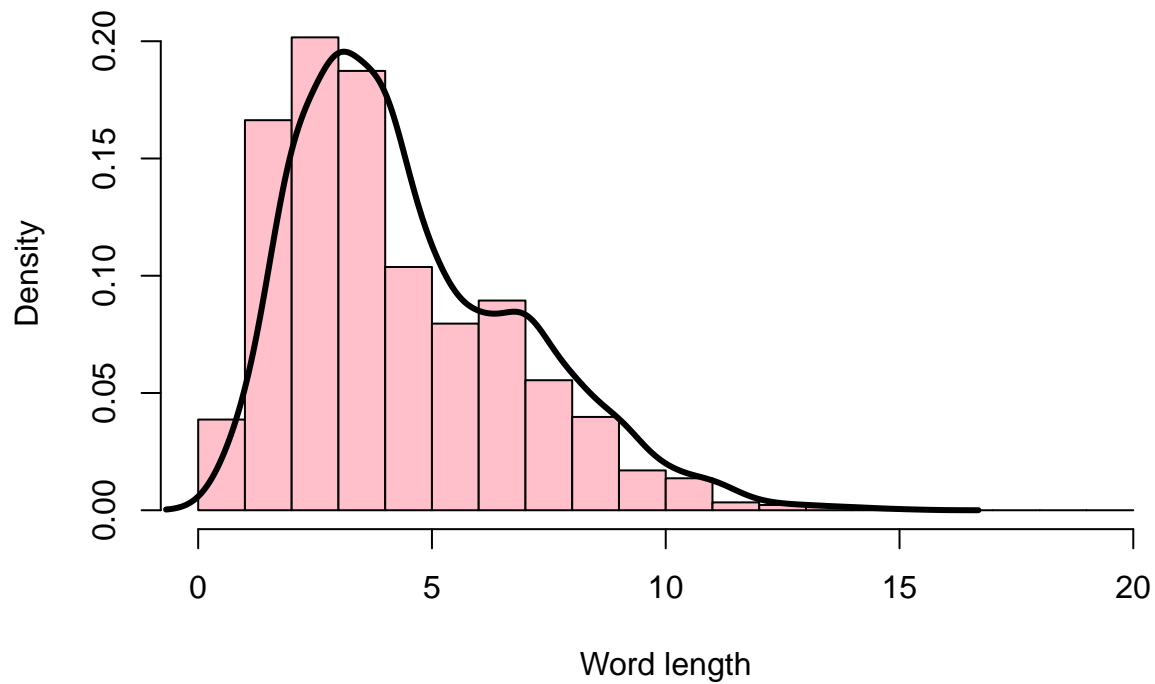
```
## [1] "density"
```

```
names(density.est)
```

```
## [1] "x"      "y"      "bw"     "n"      "call"   "data.name"  
## [7] "has.na"
```

```
hist(trump.wlens, col="pink", freq=FALSE, breaks=0:20,  
     xlab="Word length", main="Trump word lengths")  
lines(density.est, lwd=3)
```

Trump word lengths



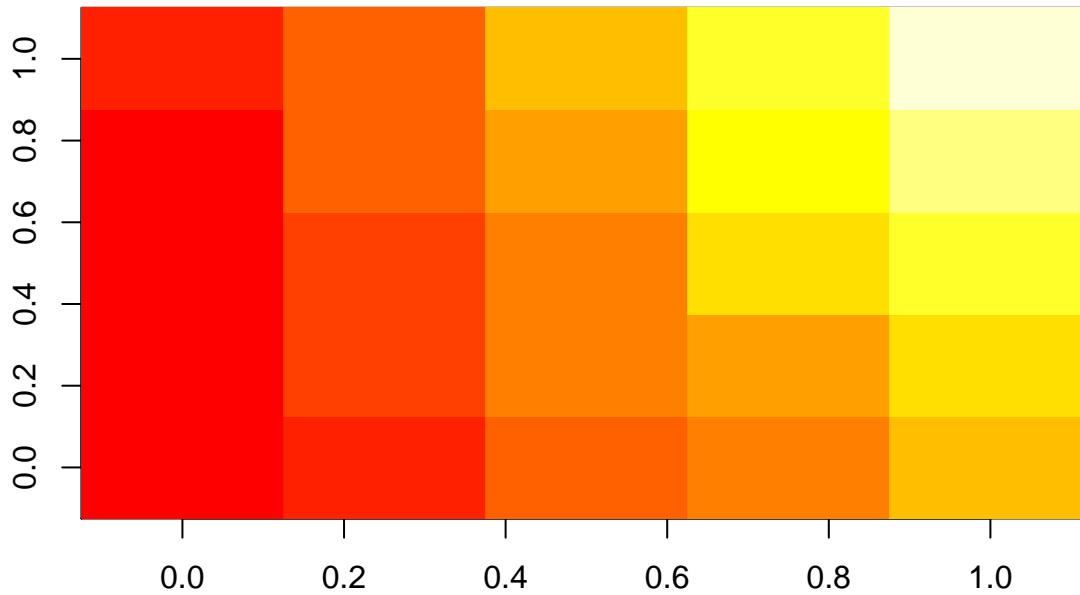
Plotting a heatmap

To plot a heatmap of a numeric matrix, use `image()`

```
(mat = 1:5 %o% 6:10) # %% gives for outer product
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   6   7   8   9  10
## [2,]  12  14  16  18  20
## [3,]  18  21  24  27  30
## [4,]  24  28  32  36  40
## [5,]  30  35  40  45  50
```

```
image(mat) # Red means low, white means high
```



Orientation of `image()`

The orientation of `image()` is to plot the heatmap according to the following order, in terms of the matrix elements:

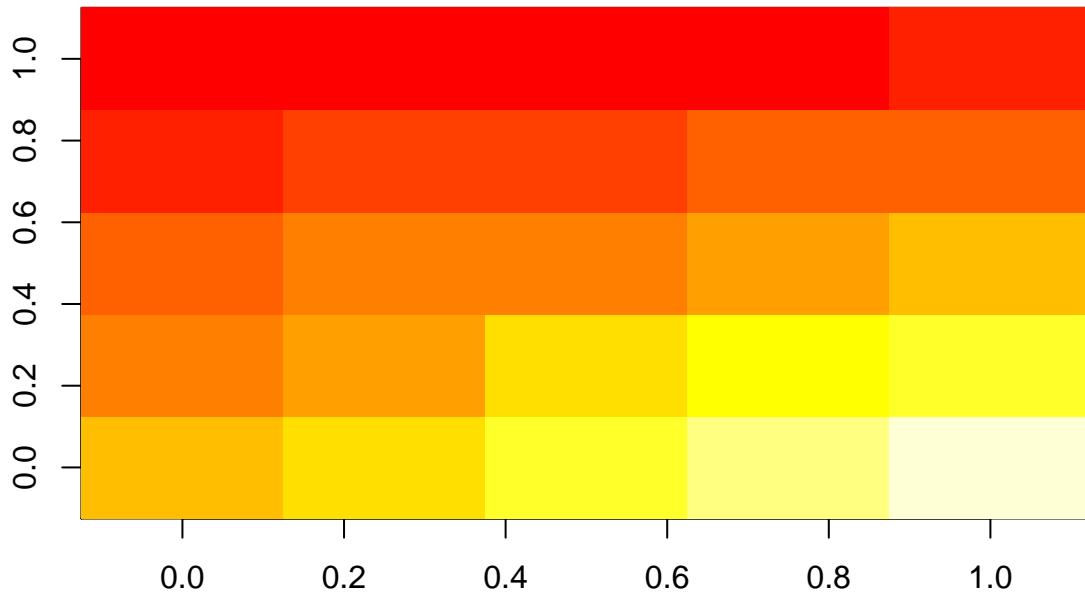
$$\begin{array}{cccc}
 (1, \text{ncol}) & (2, \text{ncol}) & \dots & (\text{nrow}, \text{ncol}) \\
 \vdots & & & \\
 (1, 2) & (2, 2) & \dots & (\text{nrow}, 2) \\
 (1, 1) & (2, 1) & \dots & (\text{nrow}, 1)
 \end{array}$$

This is a *90 degrees counterclockwise* rotation of the “usual” printed order for a matrix:

$$\begin{array}{cccc}
 (1, 1) & (1, 2) & \dots & (1, \text{ncol}) \\
 (2, 1) & (2, 2) & \dots & (2, \text{ncol}) \\
 \vdots & & & \\
 (\text{nrow}, 1) & (\text{nrow}, 2) & \dots & (\text{nrow}, \text{ncol})
 \end{array}$$

Therefore, if you want the displayed heatmap to follow the usual order, you must rotate the matrix *90 degrees clockwise* before passing it in to `image()`. (Equivalently: reverse the row order, then take the transpose.) Convenient way of doing so:

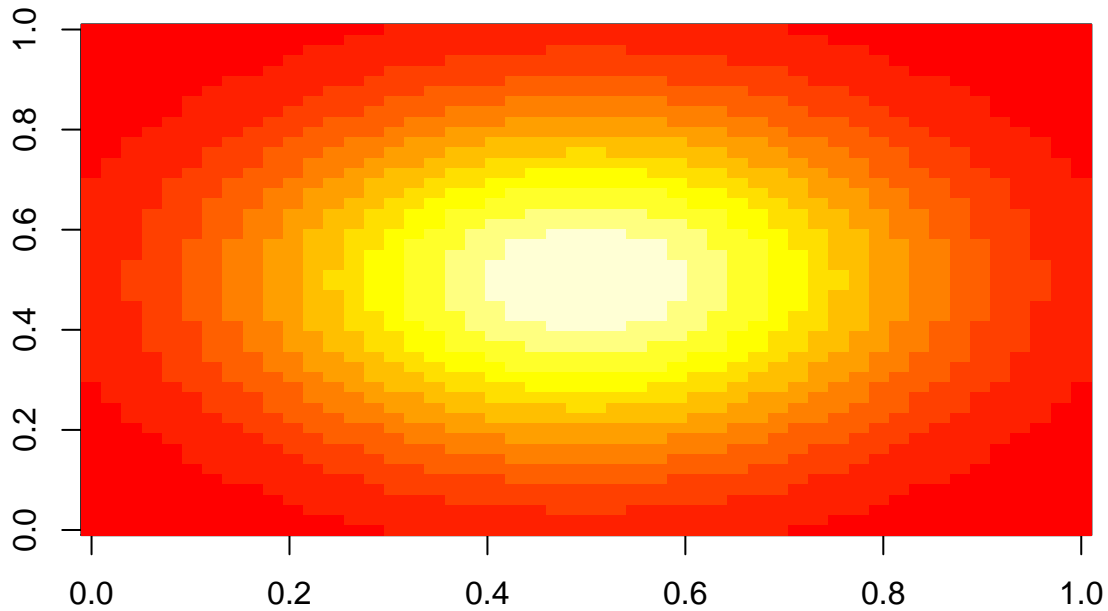
```
clockwise90 = function(a) { t(a[nrow(a):1,]) } # Handy rotate function
image(clockwise90(mat))
```



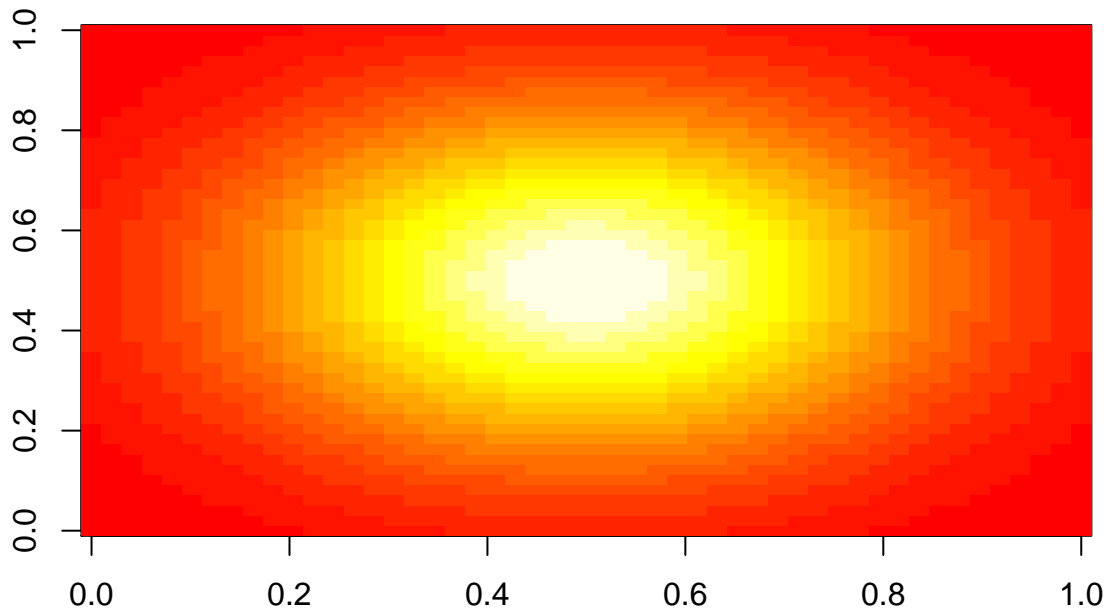
Color scale

The default is to use a red-to-white color scale in `image()`. But the `col` argument can take any vector of colors. Built-in functions `gray.colors()`, `rainbow()`, `heat.colors()`, `topo.colors()`, `terrain.colors()`, `cm.colors()` all return contiguous color vectors of given length

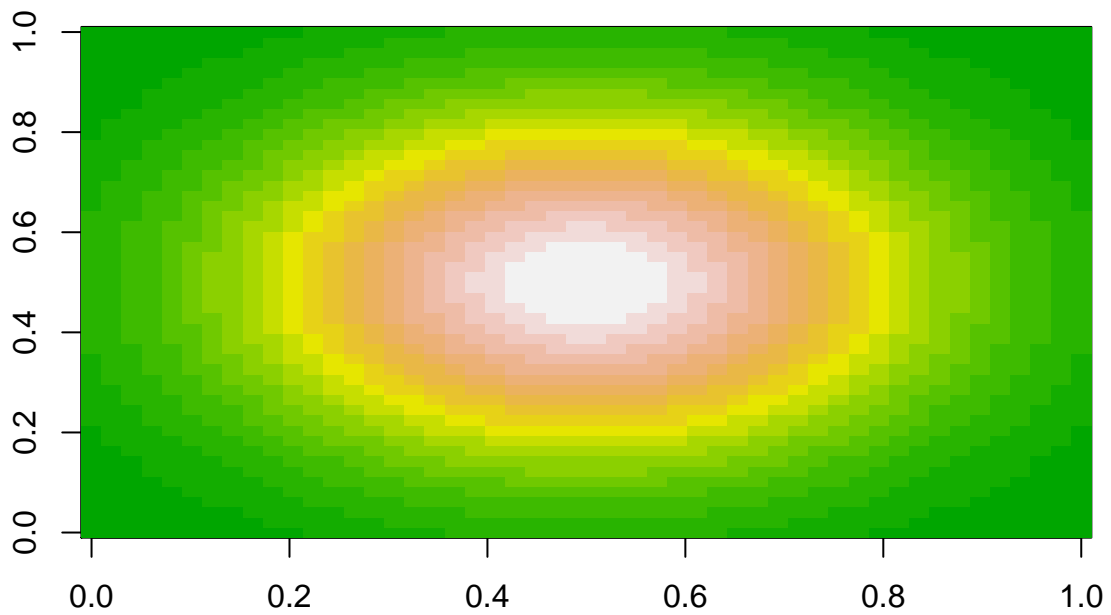
```
phi = dnorm(seq(-2,2,length=50))
normal.mat = phi %>% phi
image(normal.mat) # Default is col=heat.colors(12)
```



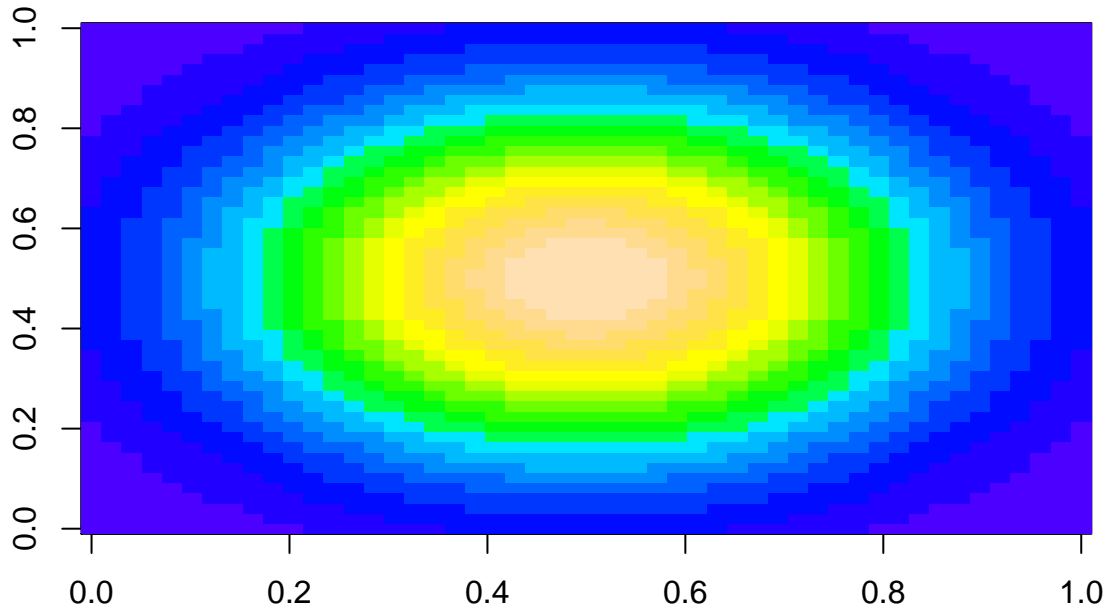
```
image(normal.mat, col=heat.colors(20)) # More colors
```



```
image(normal.mat, col=terrain.colors(20)) # Terrain colors
```



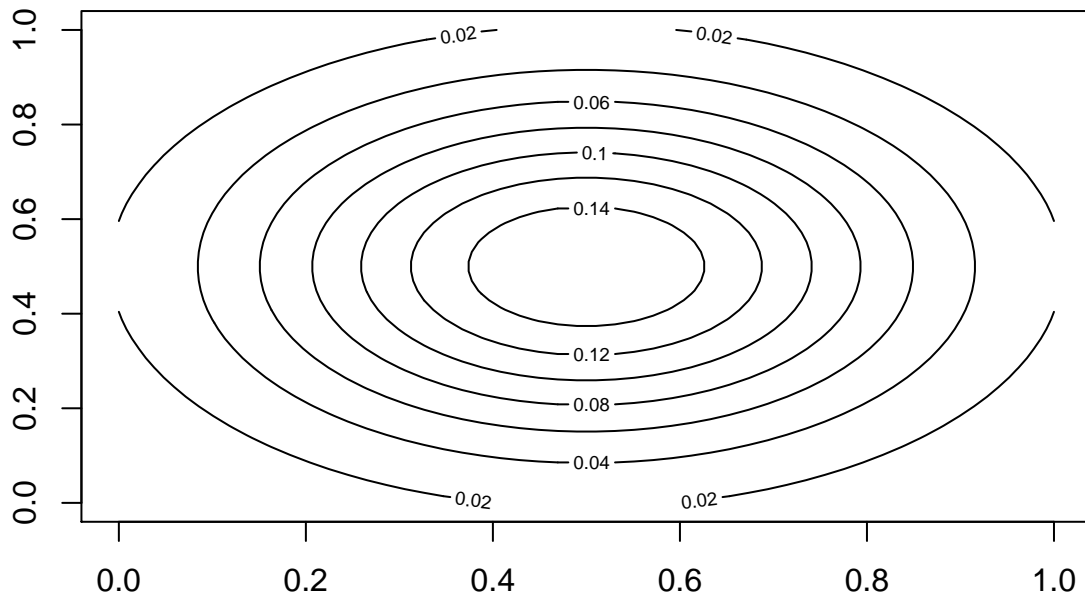
```
image(normal.mat, col=topo.colors(20)) # Topological colors
```



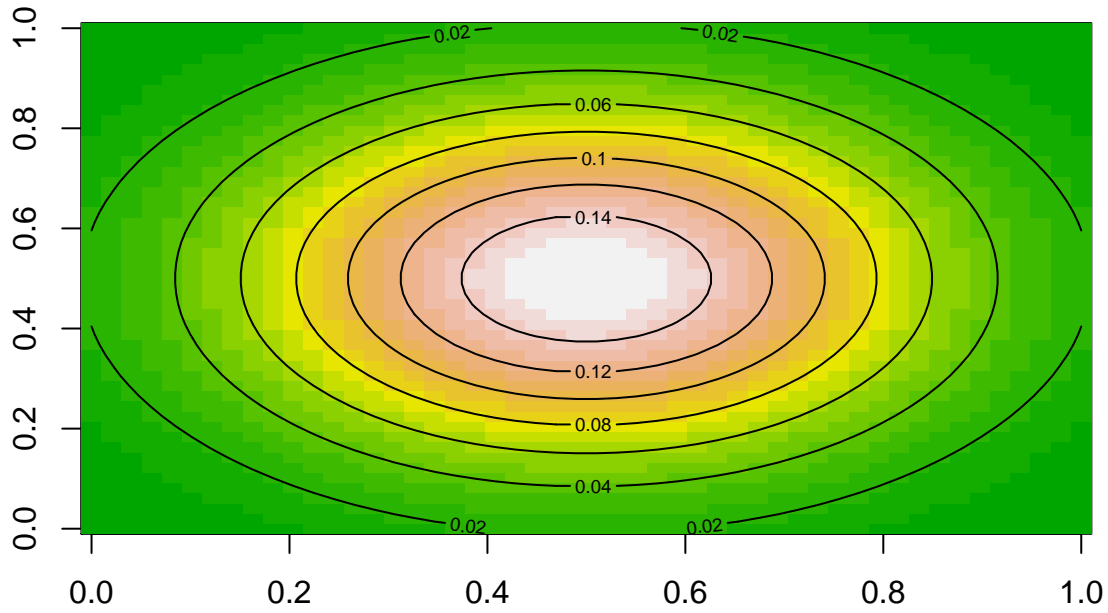
Drawing contour lines

To draw contour lines from a numeric matrix, use `contour()`; to add contours to an existing plot (like, a heatmap), use `contour()` with `add=TRUE`

```
contour(normal.mat)
```



```
image(normal.mat, col=terrain.colors(20))  
contour(normal.mat, add=TRUE)
```



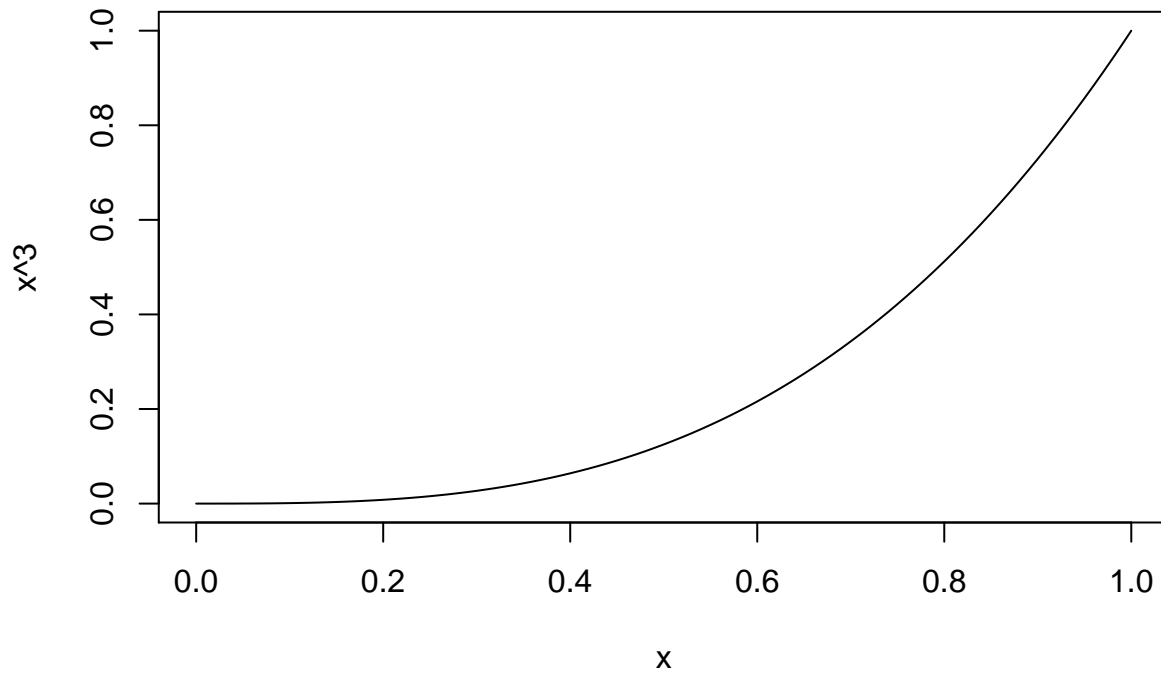
Part III

Curves, surfaces, and colors

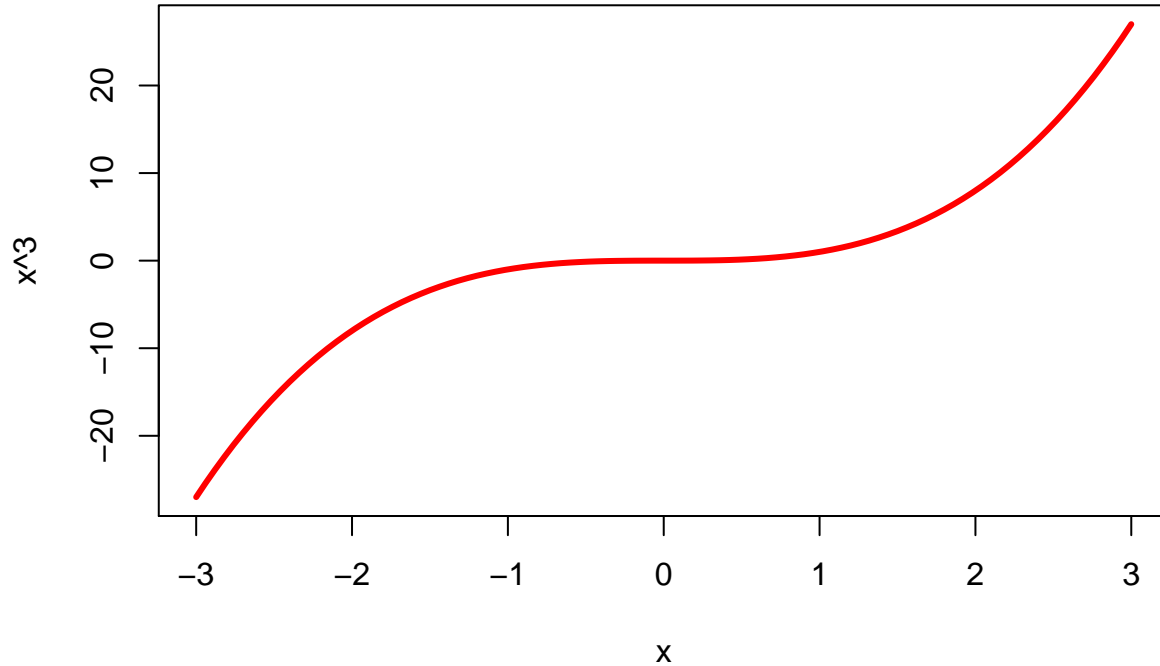
Drawing a curve

To draw a curve of a function, use `curve()`

```
curve(x^3) # Default is to plot between 0 and 1. Note: x here is a symbol
```



```
curve(x^3, from=-3, to=3, lwd=3, col="red") # More plotting options
```

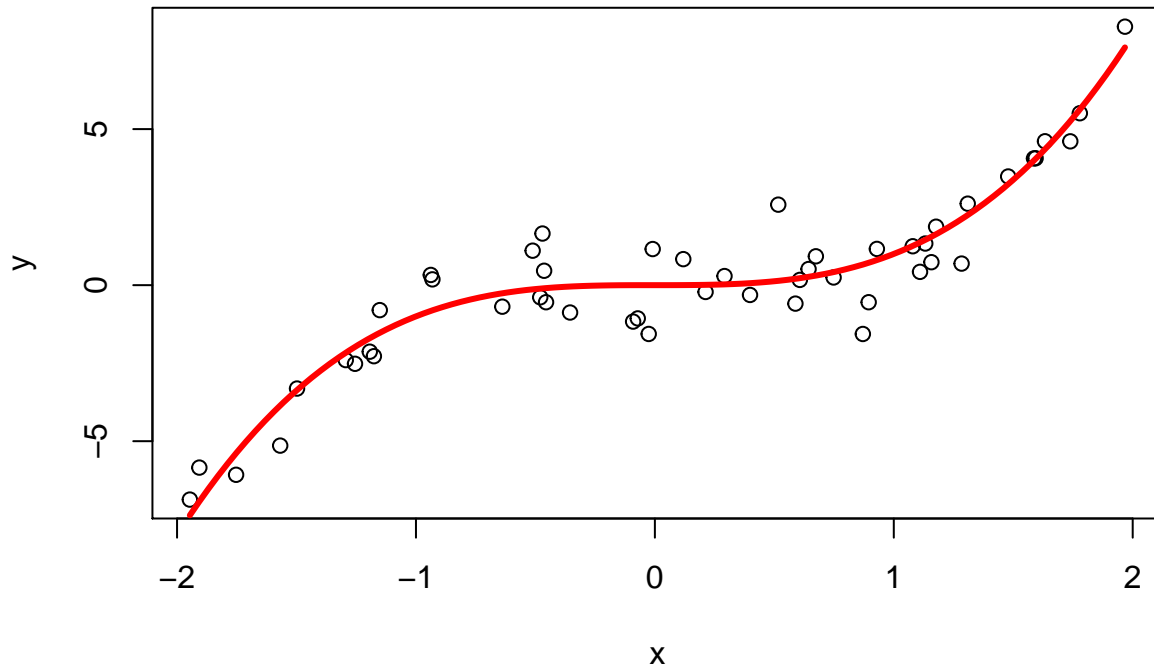


Adding a curve to an existing plot

To add a curve to an existing plot, use `curve()` with `add=TRUE`

```
n = 50
set.seed(0)
x = sort(runif(n, min=-2, max=2))
y = x^3 + rnorm(n)

plot(x, y)
curve(x^3, lwd=3, col="red", add=TRUE)
```

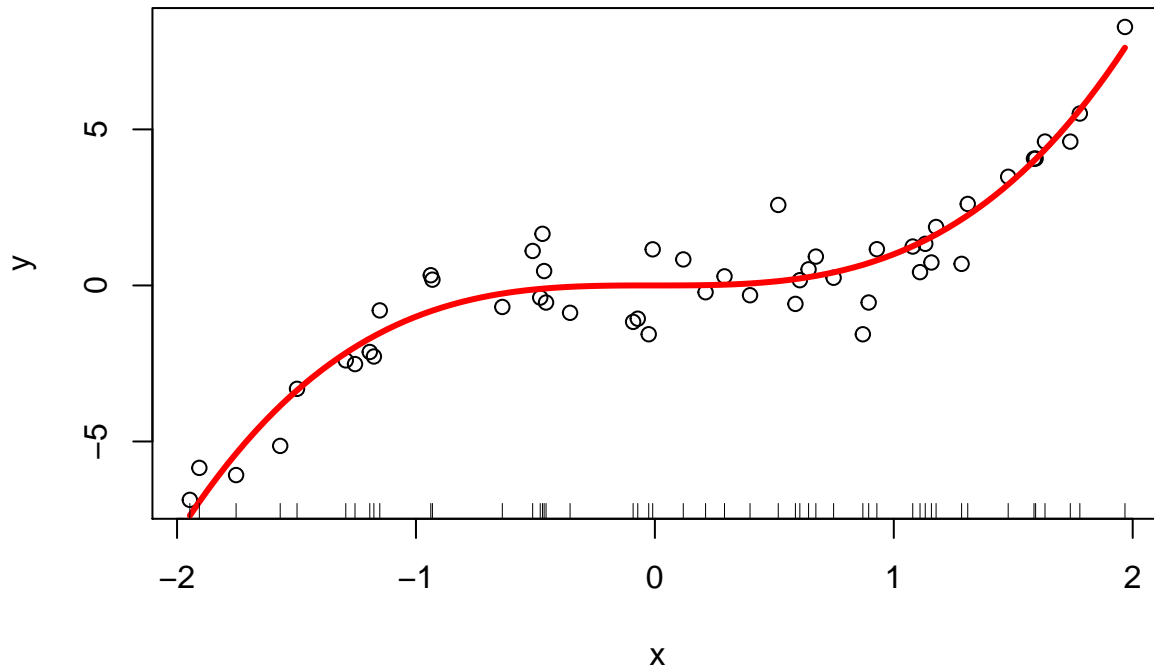


*# Note: the x argument here and the x vector we defined above are different!
 # Reminder: x here is a symbol*

Adding a rug to an existing plot

To add a rug to an existing plot (just tick marks, for where the x points occur), use `rug()`

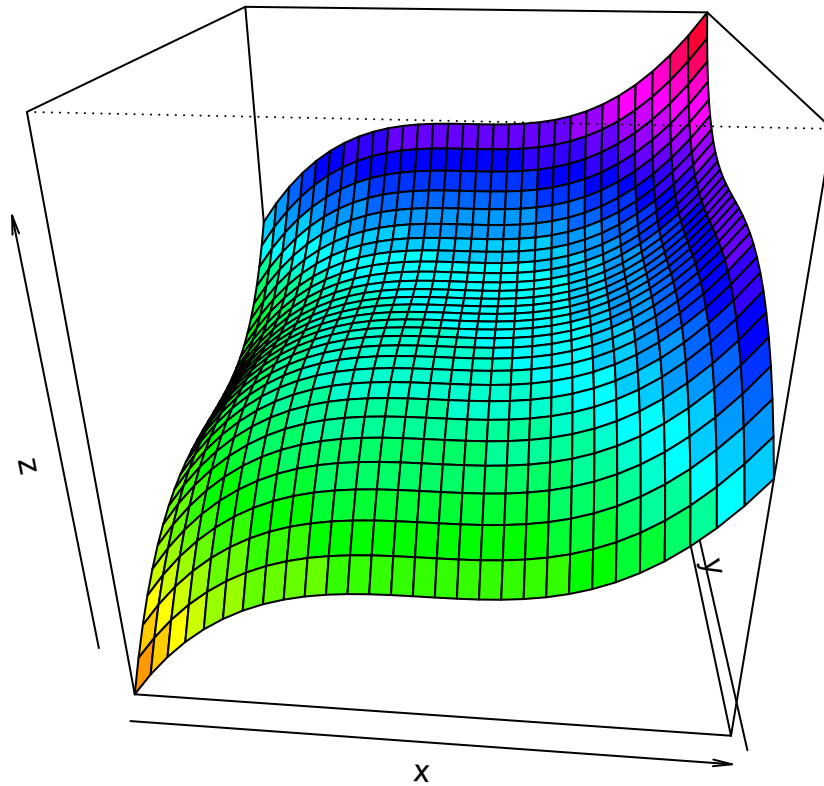
```
plot(x, y)
curve(x^3, lwd=3, col="red", add=TRUE)
rug(x)
```



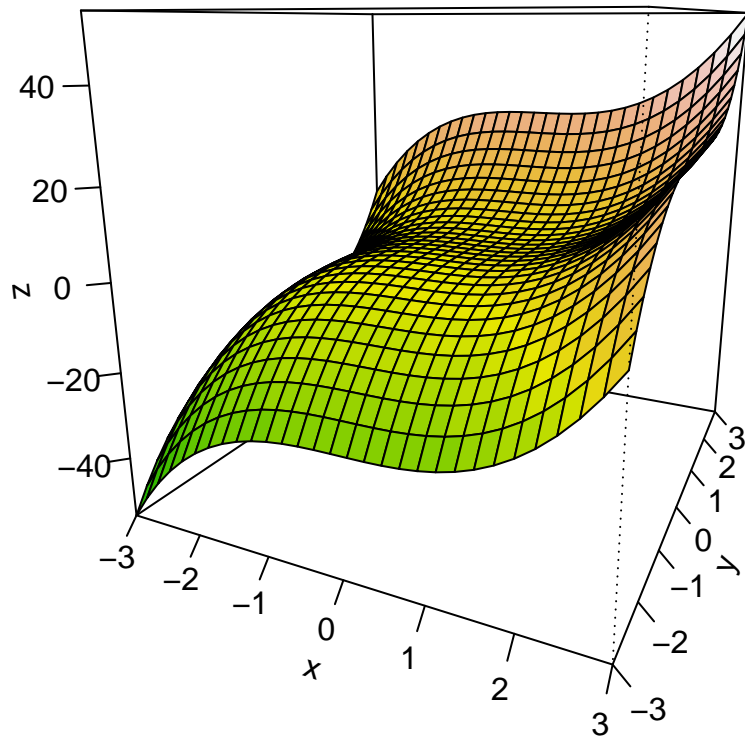
Drawing a surface

To draw a surface, use `surface()`, available at <http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/surface.R>. (This is a function written by Professor Tibs, relying on the built-in `persp()` function)

```
source("http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/surface.R")
surface(x^3 + y^3, from.x=-3, to.x=3, from.y=-3, to.y=3)
```



```
surface(x^3 + y^3, from.x=-3, to.x=3, from.y=-3, to.y=3,
        theta=25, phi=15, col=terrain.colors(30),
        ticktype="detailed", mar=c(2,2,2,2))
```



#theta and phi represent angles defining the viewing direction. theta gives the azimuthal direction and

Adding points to a surface

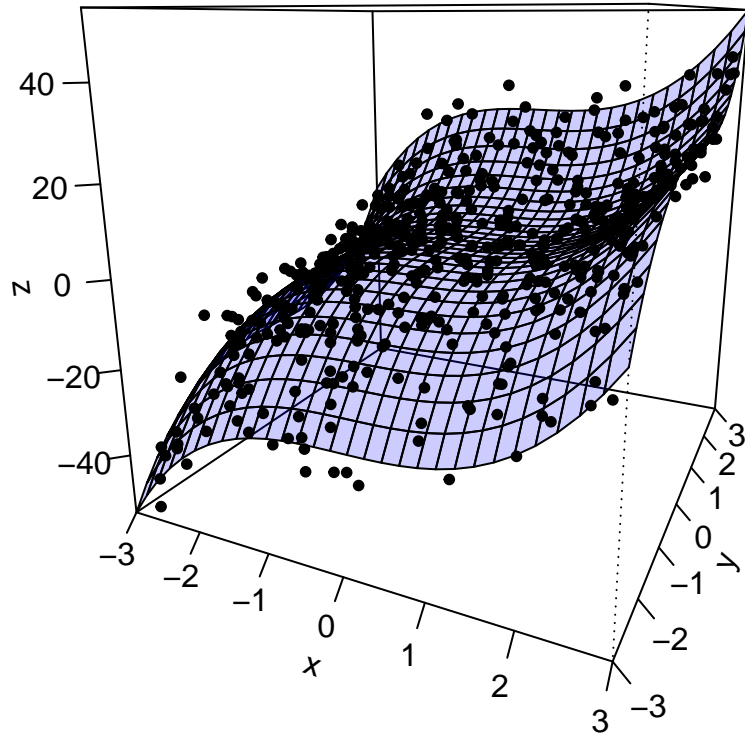
To add points to a surface, save the output of `surface()`. Then use `trans3d()`, to transform (x,y,z) coordinates to (x,y) coordinates that you can pass to `points()`

```
persp.mat = surface(x^3 + y^3, from.x=-3, to.x=3, from.y=-3, to.y=3,
                    theta=25, phi=15, col=rgb(0,0,1,alpha=0.2),
                    ticktype="detailed", mar=c(2,2,2,2))
```

*#rgb function creates colors corresponding to the given intensities
#(between 0 and max) of the red, green and blue primaries.
#alpha specifies the transparency value*

```
n = 500
x = runif(n, -3, 3)
y = runif(n, -3, 3)
z = x^3 + y^3 + 5*rnorm(n)

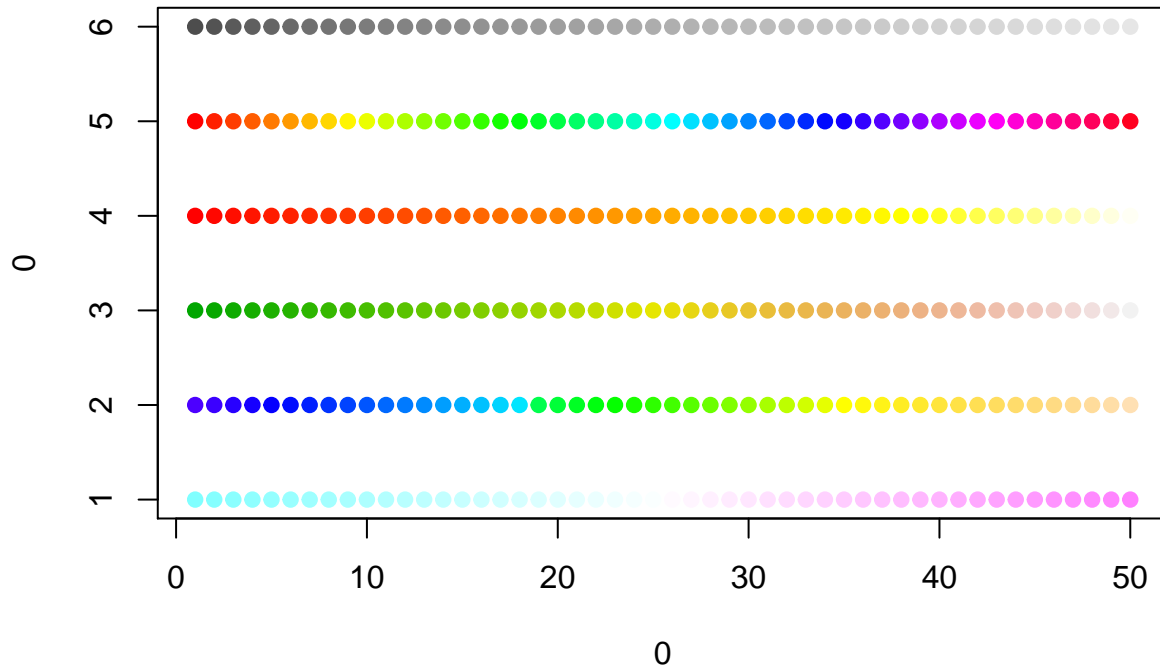
xy.list = trans3d(x, y, z, persp.mat)
points(xy.list, pch=20)
```



Color palettes

Color palettes are functions for creating vectors of contiguous colors, just like `gray.colors()`, `rainbow()`, `heat.colors()`, `terrain.colors()`, `topo.colors()`, `cm.colors()`. Given a number `n`, each of these functions just returns a vector of colors (names, stored as strings) of length `n`

```
n = 50
plot(0, 0, type="n", xlim=c(1,n), ylim=c(1,6))
points(1:n, rep(6,n), col=gray.colors(n), pch=19)
points(1:n, rep(5,n), col=rainbow(n), pch=19)
points(1:n, rep(4,n), col=heat.colors(n), pch=19)
points(1:n, rep(3,n), col=terrain.colors(n), pch=19)
points(1:n, rep(2,n), col=topo.colors(n), pch=19)
points(1:n, rep(1,n), col=cm.colors(n), pch=19)
```



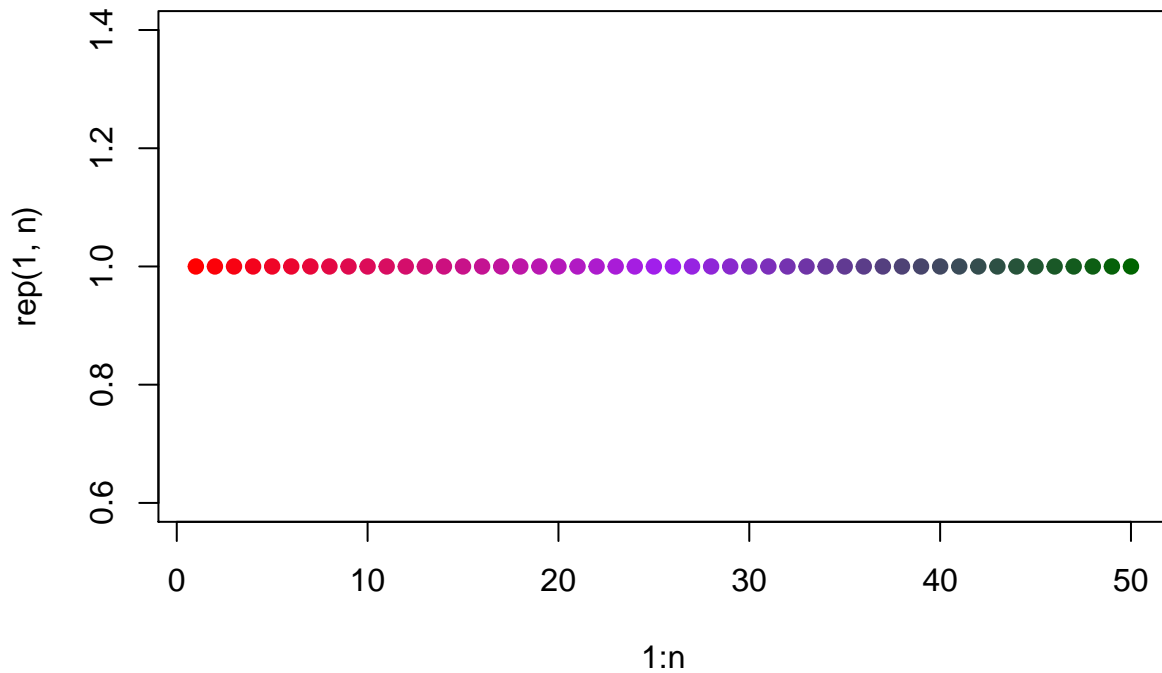
Creating a custom color palette

To create a custom palette, that interpolates between a set of base colors, `colorRampPalette()`

```
cust.colors = colorRampPalette(c("red", "purple", "darkgreen"))
class(cust.colors)
```

```
## [1] "function"
```

```
plot(1:n, rep(1,n), col=cust.colors(n), pch=19)
```



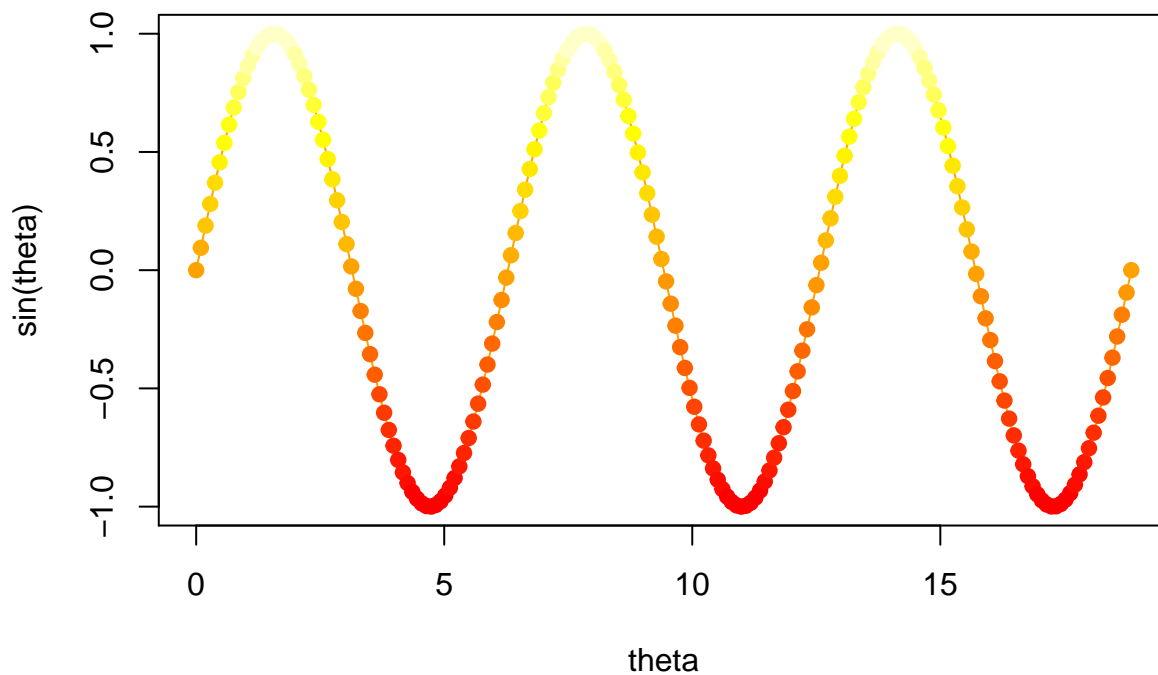
Coloring points by value

Coloring points according to the value of some variable can just be done with a bit of indexing, and the tools you already know about colors

```
# Function to retrieve a color according to a value
# - val: the value in question
# - lim: a vector of length 2, lower and upper limits for possible values
# - col.vec: the color vector to choose from
get.col.from.val = function(val, lim, col.vec) {
  col.vec[(val-lim[1])/(lim[2]-lim[1]) * (length(col.vec)-1) + 1]
}

# Let's color points according to y value
col.vec = heat.colors(30)
lim = c(-1, 1)
theta = seq(0, 6*pi, length=200)

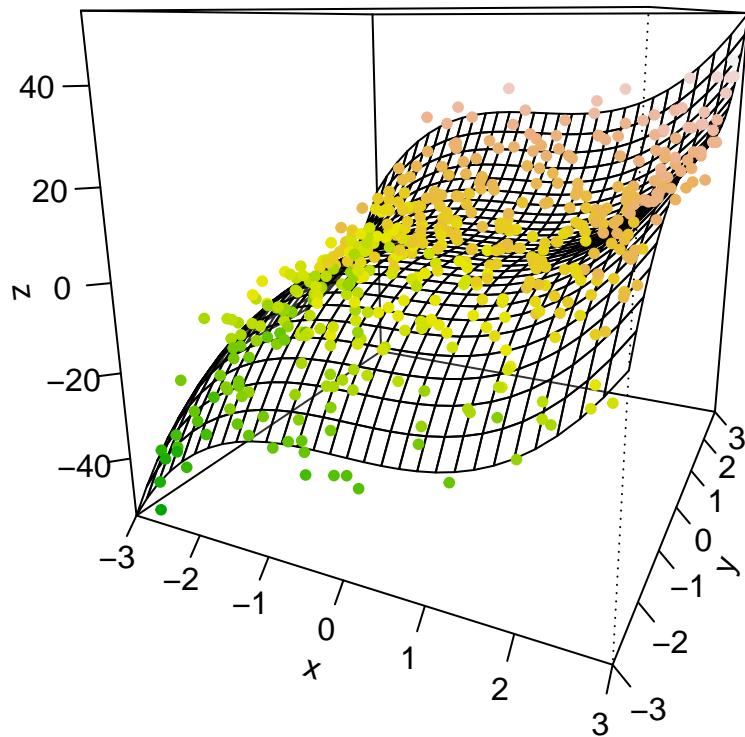
plot(theta, sin(theta), type="o", pch=19,
      col=get.col.from.val(sin(theta), lim, col.vec))
```



```
# Another example, now in 3d
persp.mat = surface(x^3 + y^3, from.x=-3, to.x=3, from.y=-3, to.y=3,
                   theta=25, phi=15, col=rgb(1,1,1,alpha=0.2),
                   ticktype="detailed", mar=c(2,2,2,2))

# Let's color points according to z value
col.vec = terrain.colors(30)
lim = c(min(z), max(z))

xy.list = trans3d(x, y, z, persp.mat)
points(xy.list, pch=20, col=get.col.from.val(z, lim, col.vec))
```



Summary

- `plot()`: generic plotting function
- `points()`: add points to an existing plot
- `lines()`, `abline()`: add lines to an existing plot
- `text()`, `legend()`: add text to an existing plot
- `rect()`, `polygon()`: add shapes to an existing plot
- `hist()`, `image()`: histogram and heatmap
- `heat.colors()`, `topo.colors()`, etc: create a color vector
- `density()`: estimate density, which can be plotted
- `contour()`: draw contours, or add to existing plot
- `curve()`: draw a curve, or add to existing plot