Plotting Tools

Statistical Computing, STA3005

Tuesday Feb 26, 2024

Last chapter: Text manipulation

- Strings are sequences of characters bound together
- Text data occurs frequently “in the wild”, so you should learn how to deal with it!
- `nchar()`, `substr()` : functions for substring extractions and replacements
- `strsplit()`, `paste()` : functions for splitting and combining strings
- `grep()`, `gsub()` : functions for matching and substituting for a given regular expression
- A regular expression is a string that describe a certain text pattern. The pattern is expressed by the combination of normal and special characters
- Reconstitution: take lines of text, combine into one long string, then split to get the words
- `table()` : function to get word counts, useful way of summarizing text data
- Zipf’s law: word frequency tends to be inversely proportional to (a power of) rank

Part I: *Plot basics*

Plotting in R

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

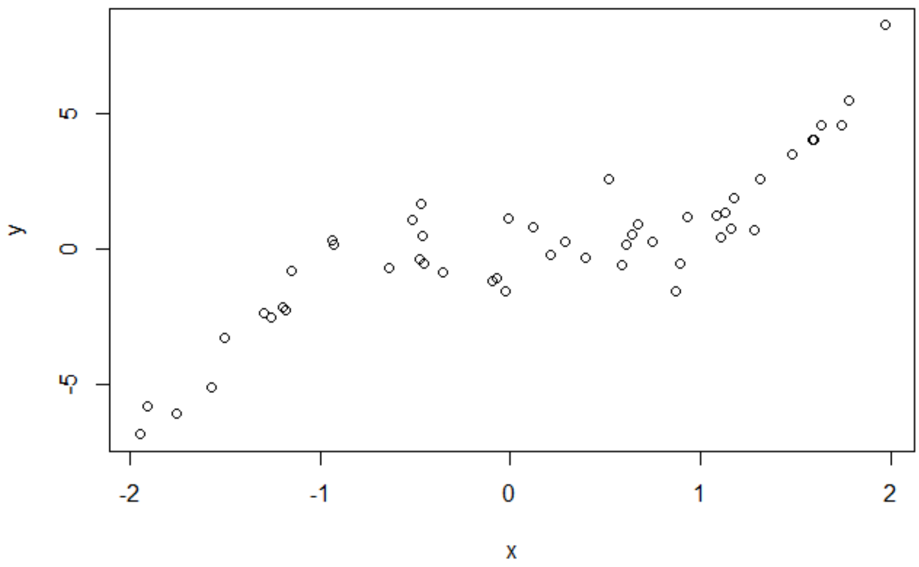
- `plot()` : generic plotting function
- `points()` : add points to an existing plot
- `lines()`, `abline()` : add lines to an existing plot
- `text()`, `legend()` : add text to an existing plot
- `rect()`, `polygon()` : add shapes to an existing plot
- `hist()`, `image()` : histogram and heatmap
- `heat.colors()`, `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

As shown in the last chapter, the `ggplot2` package also provides very nice plotting tools

Scatter plot

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

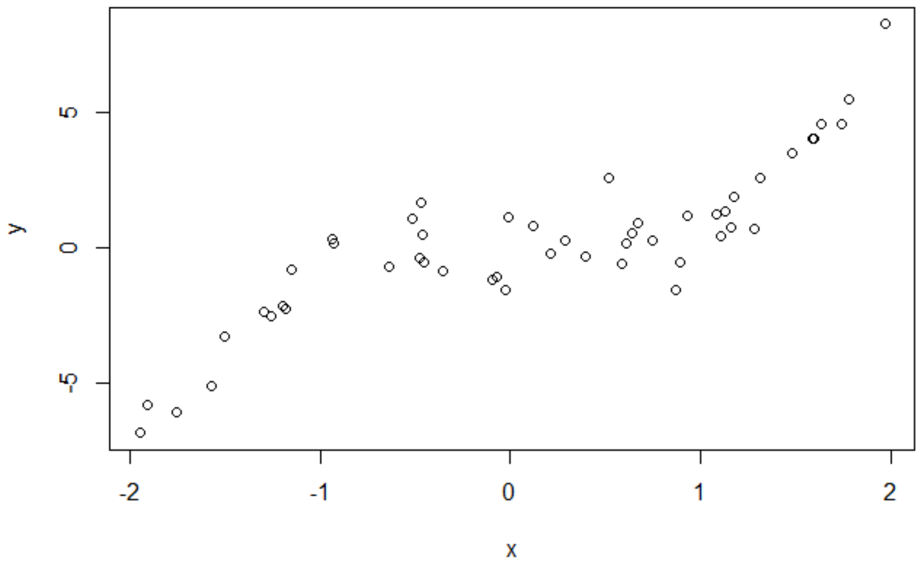
```
n = 50
set.seed(0)
x = sort(runif(n, min=-2, max=2))
y = x^3 + rnorm(n)
plot(x, y)
```



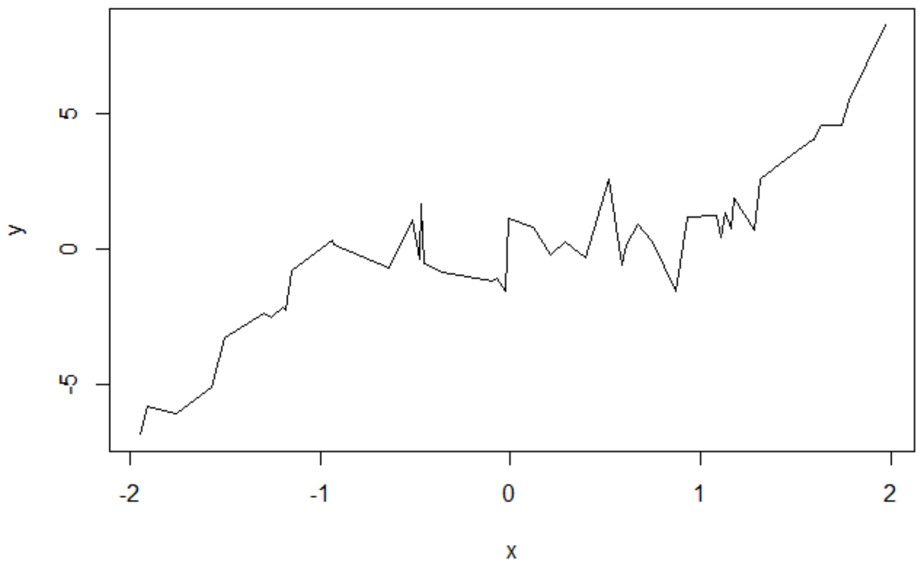
Plot type

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")
```

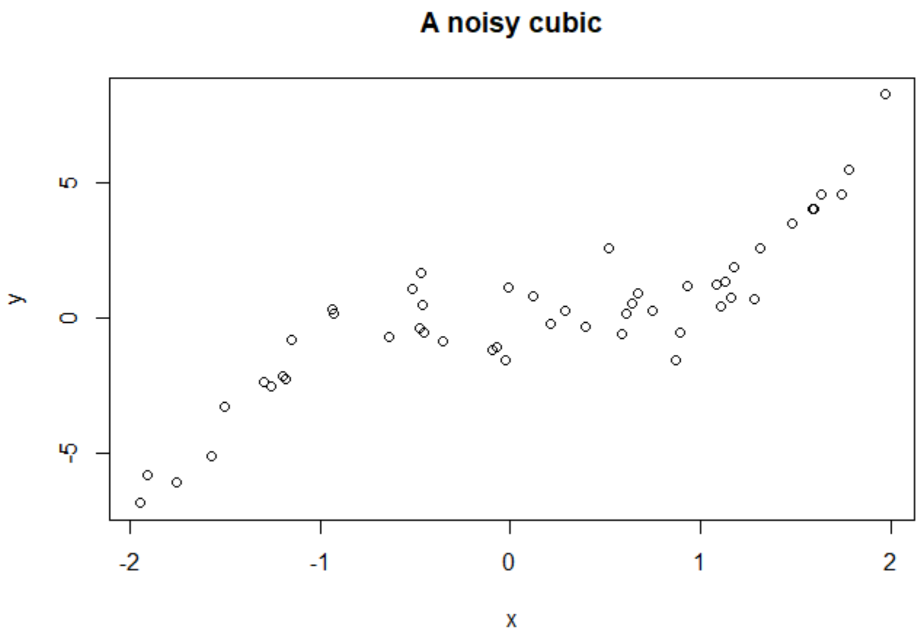


Try also `b` or `o` for both points and lines, and `n` for nothing

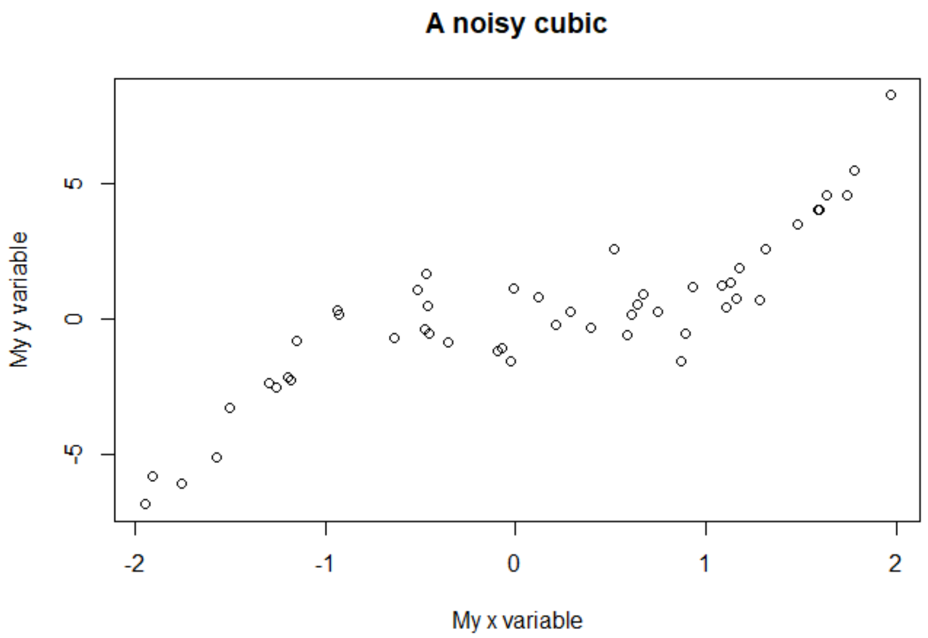
Labels

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
```



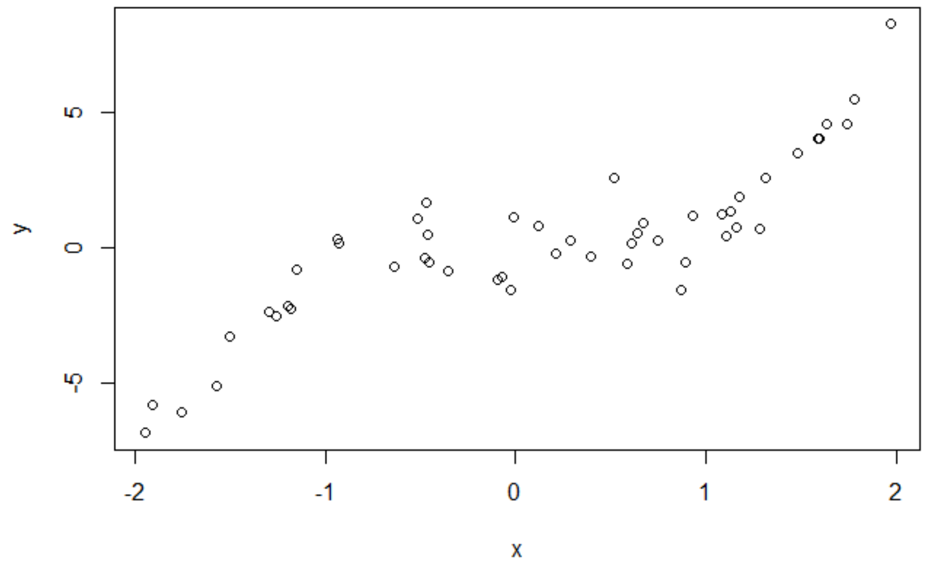

```
plot(x, y, main="A noisy cubic", xlab="My x variable", ylab="My y variable")
```



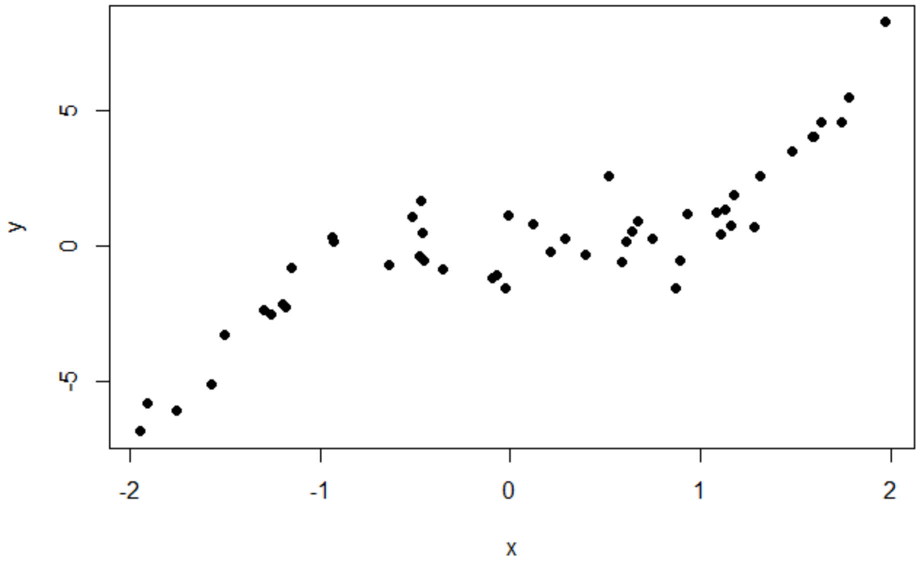
Point type

Use the `pch` argument to control point type

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




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

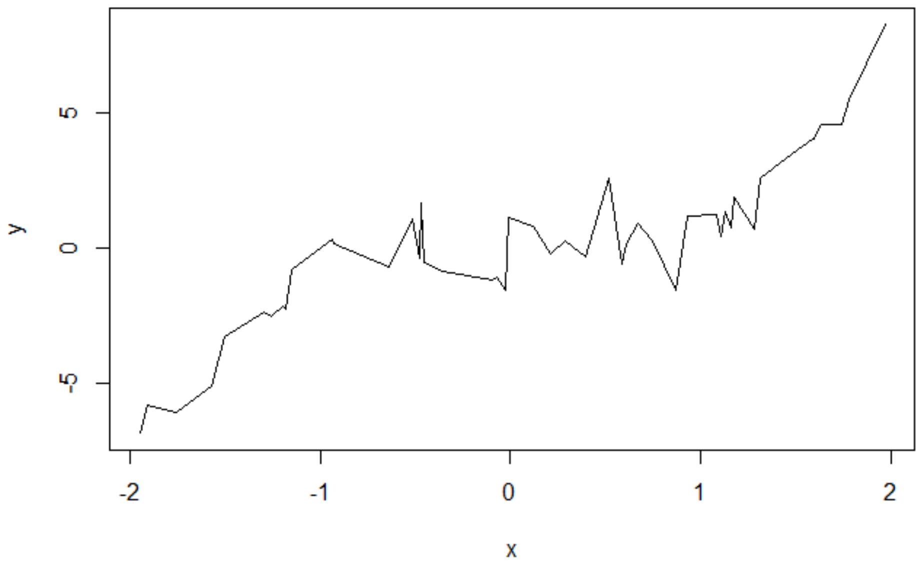


Try also `20` for small filled circles, or `"."` for single pixels

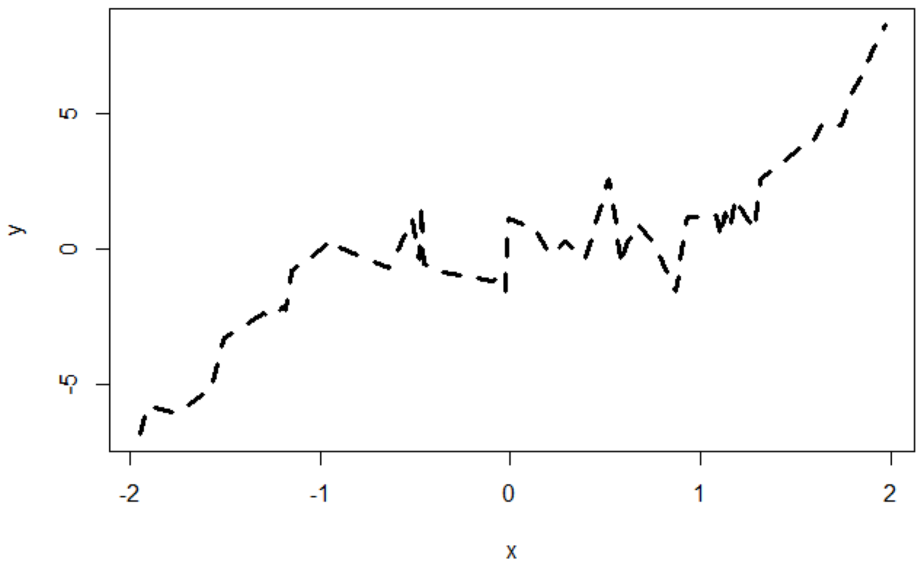
Line type

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
```



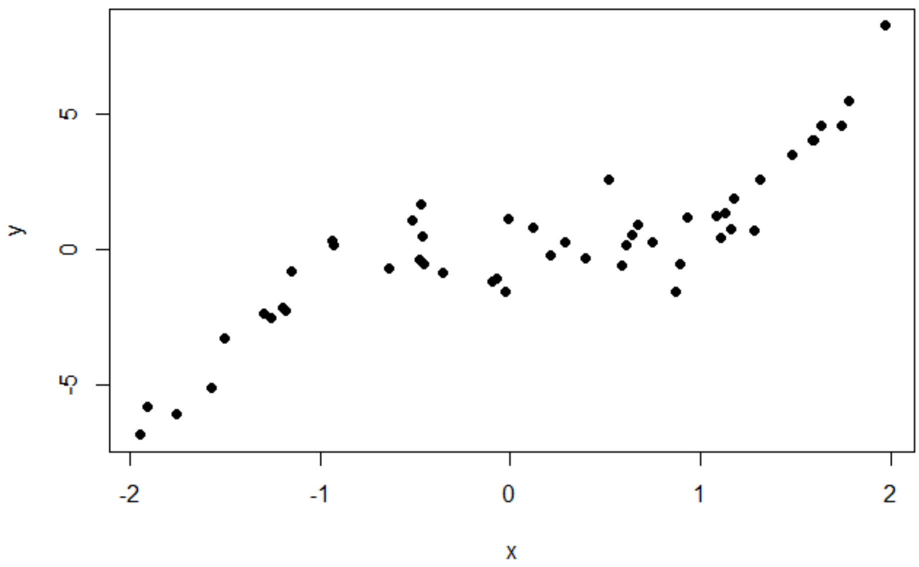
Color

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

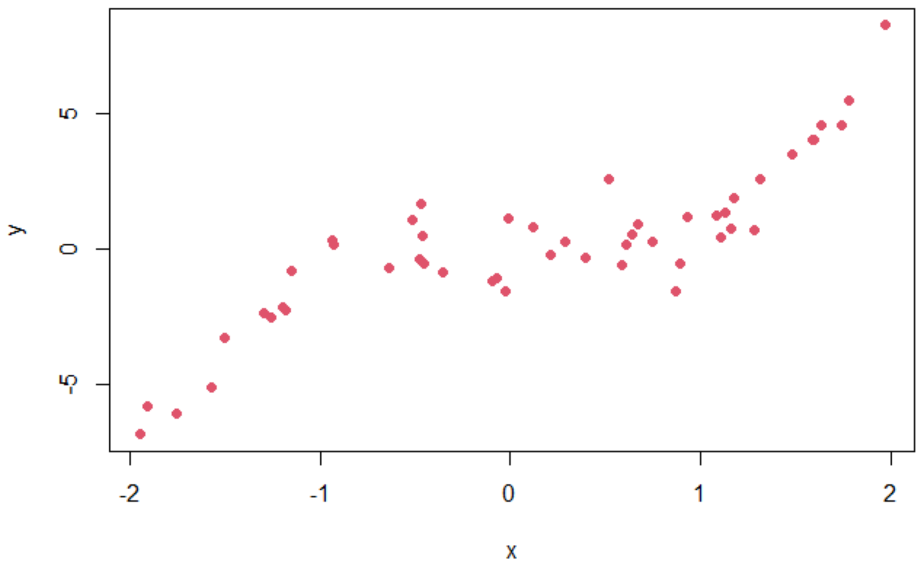
- An integer between 1 and 8 for basic colors
- A string for any of the 657 available named colors

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
```

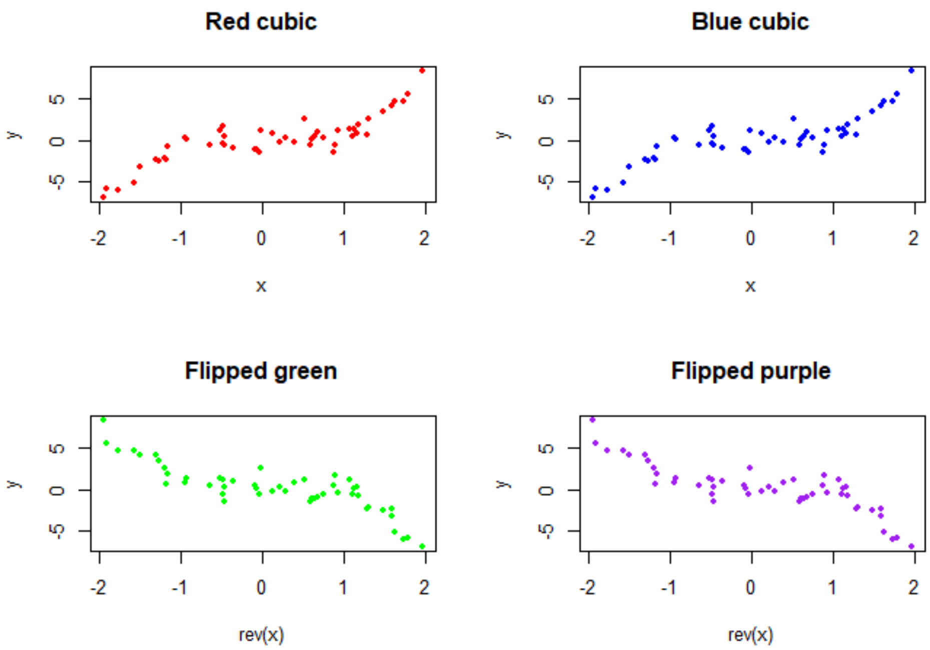


Multiple plots

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

作图顺序是 by row　mfcol: 作图顺序是 by column

```
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")
```

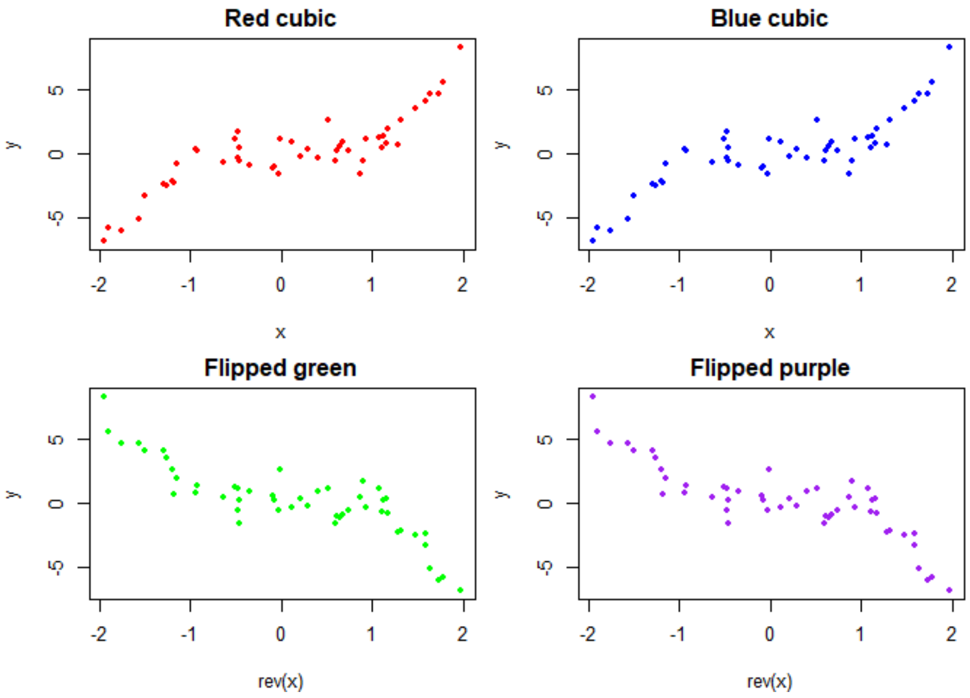


Margin

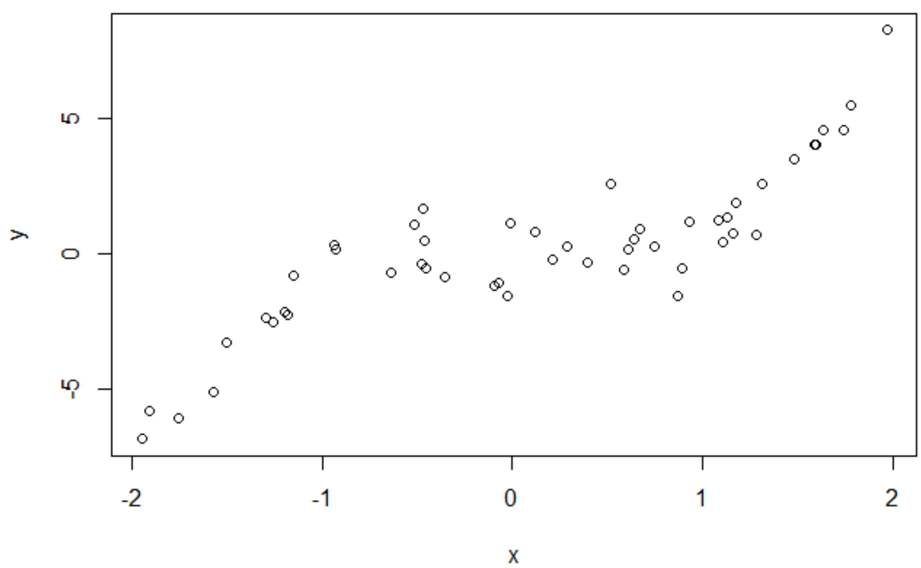
Default margins in R are large (and ugly); to change them, use the `par()` function, with the argument `mar`. Note: in general this will affect all the following plots! (Except in separate R Markdown code chunks ...)

```
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")
```

```r
plot(rev(x), y, main="Flipped green", pch=20, col="green")
plot(rev(x), y, main="Flipped purple", pch=20, col="purple")
```




```r
# 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

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

```
## [1] "D:/CUHKSZ/Teaching/STA3005/24Spring_STA3005/01LectureNotes/Chapter7"
```

```
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")
dev.off()
```

```
## png
##   2
```

```
jpeg(filename = "noisy_cubics2.jpg", width = 480, height = 480)
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")
dev.off()
```

```
## png
##   2
```

- The default size unit of `pdf()` is `in` (inch), while that of `jpeg()` is `px` (pixel). In `jpeg()`, the unit is controlled by the `units` argument. The relationship between pixels and inches depends on the resolution of your computer
- Moreover, `pdf()` generates a vector graph, while `jpeg()` generates bit graphs. Vector graphs will not turn blurred after zooming in, but bit graphs will.
- Similar to `jpeg()`, use the `png()` functions to save png files

Adding to plots

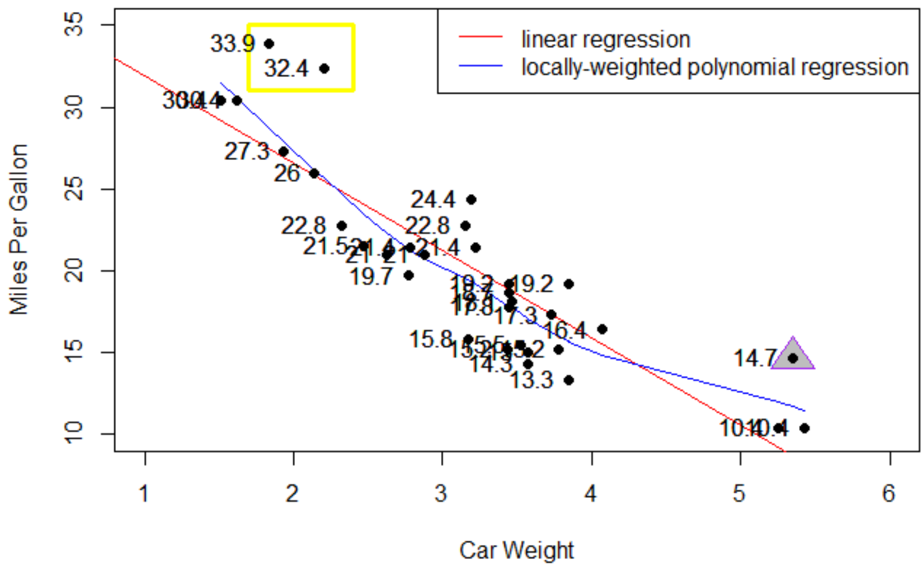
The main tools for adding extra layers on the existing plot are:

- `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

```
plot(c(1,6), c(10,35), type = "n",
   xlab="Car Weight", ylab="Miles Per Gallon")
points(mtcars$wt, mtcars$mpg, pch=19)

# Add fit lines
abline(lm(mtcars$mpg~mtcars$wt), col="red") # regression line (y~x)
lines(lowess(mtcars$wt,mtcars$mpg), col="blue") # lowess line (x,y)

# text example and legend example from assignments
text(mtcars$wt, mtcars$mpg, labels = mtcars$mpg, pos = 2)
legend("topright", legend = c("linear regression", "locally-weighted polynomial regression")
# draw a rectangle
rect(1.7, 35, 2.4, 31, border = "yellow", lwd = 3)
polygon(c(5.2,5.5,5.35), c(14,14,16), col = "grey", border = "purple")
points(mtcars$wt, mtcars$mpg, pch=19)
text(mtcars$wt, mtcars$mpg, labels = mtcars$mpg, pos = 2)
```

仅作出坐标系

也可以指定坐标



更改代码运行顺序以避免覆盖

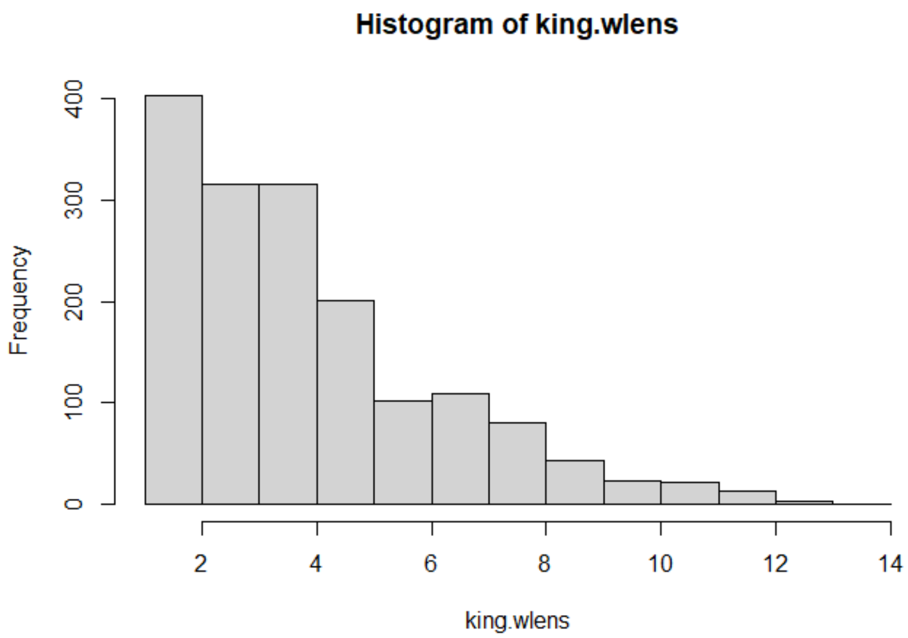
Pay attention to **layers**—they work just like they would if you were painting a picture by hand

Part II: *Histograms and heatmaps*

Plotting a histogram

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

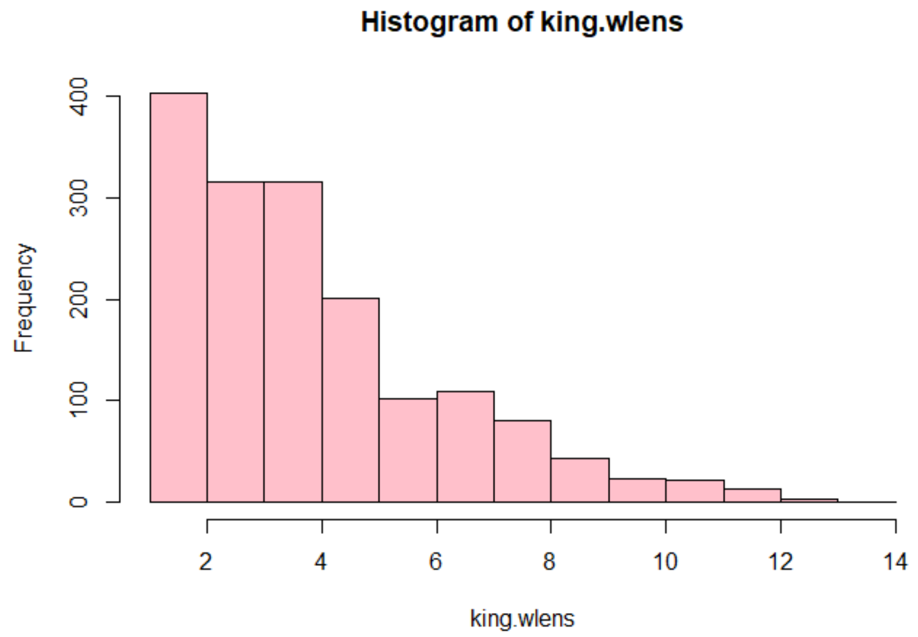
```
king.lines = readLines("king.txt")
king.words = strsplit(paste(king.lines, collapse=" "),
                      split="[[:space:]]|[[:punct:]]")[[1]]
king.words = tolower(king.words[king.words != ""])
king.wlens = nchar(king.words)
hist(king.wlens)
```



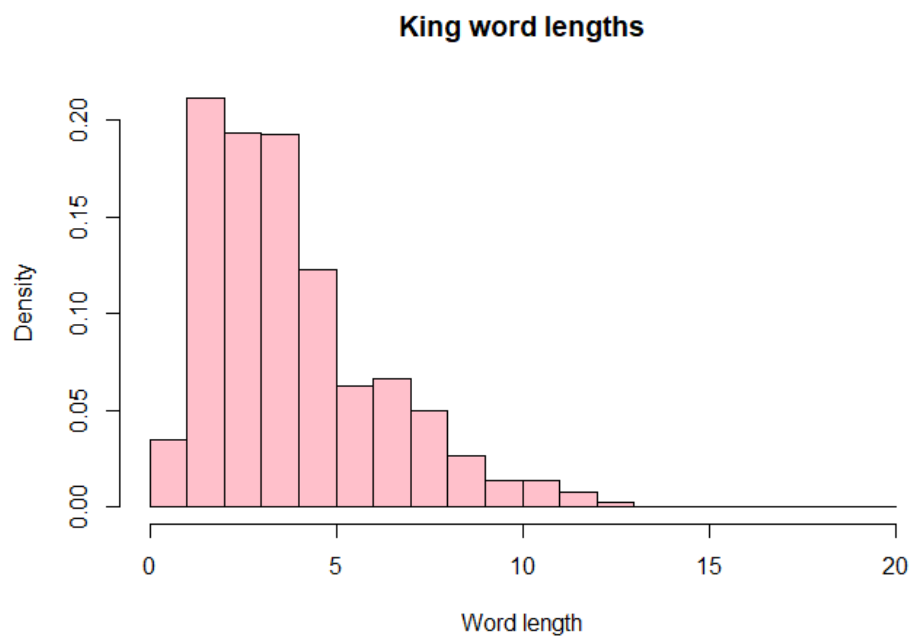
Histogram options

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

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




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



`breaks` argument allows different types of input:

- a vector: the breakpoints between histogram cells
- a single number: the number of cells/bars for the histogram

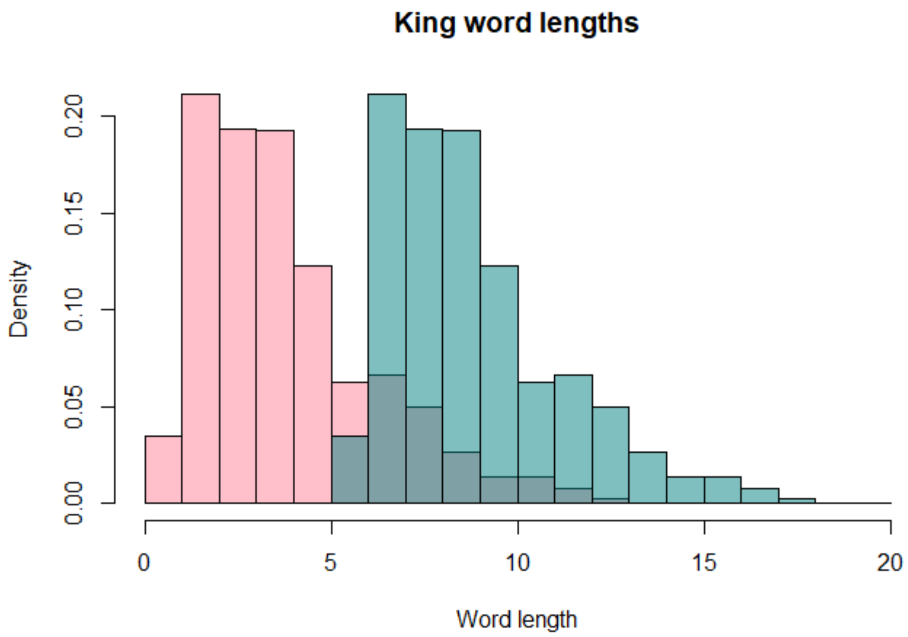
Adding a histogram to an existing plot

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

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

透明度

RGB



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

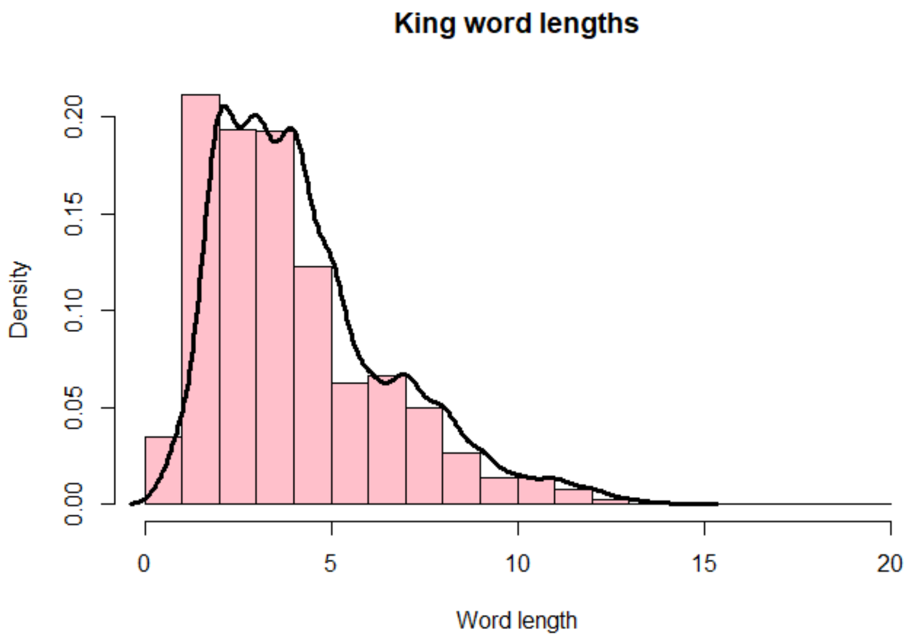
```
density.est = density(king.wlens, adjust=1.5) # 1.5 times the default bandwidth
class(density.est)
```

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

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

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

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



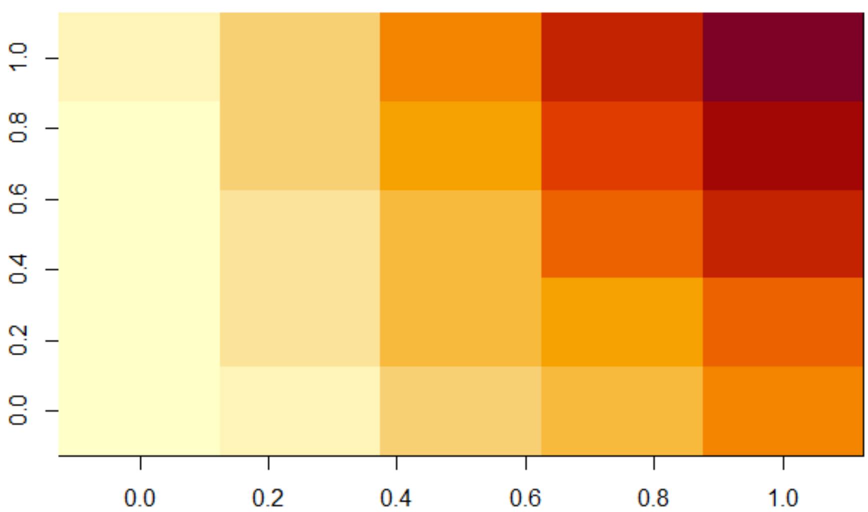
Plotting a heatmap

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

```
(mat = 1:5 %o% 6:10) # %o% 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) # White means low, red means high
```



- `%o%` stands for an outer product

Orientation of `image()`

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

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

orientation 和 matrix 不同

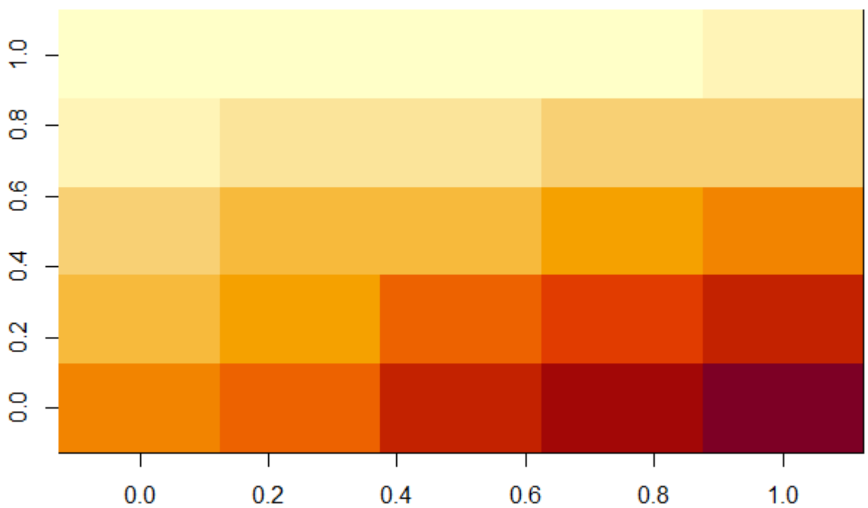
This is a *90 degrees counterclockwise* rotation of the "usual" printed order for a matrix:

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

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))
```

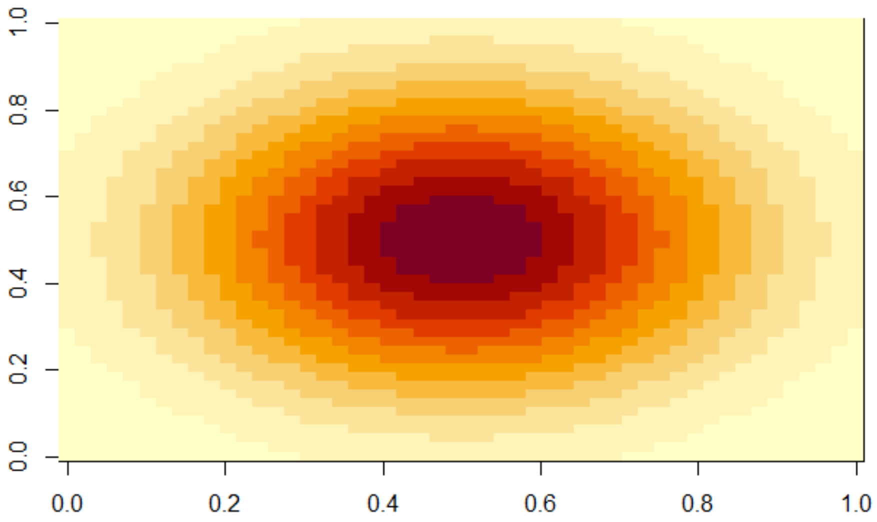
先将行倒序，再转置



Color scale

The default is to use a white-to-red (low-to-high) 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

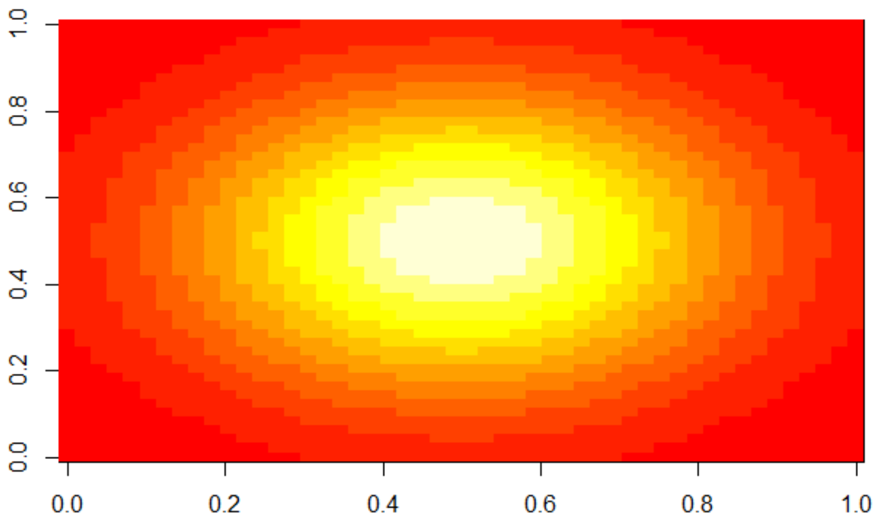
```
phi = dnorm(seq(-2,2,length=50))
normal.mat = phi %o% phi
image(normal.mat) # Default is col = hcl.colors(12, "YlOrRd", rev=TRUE)
```



若想将 x-axis 调整为 [-2,2]，
可以调整 x argument

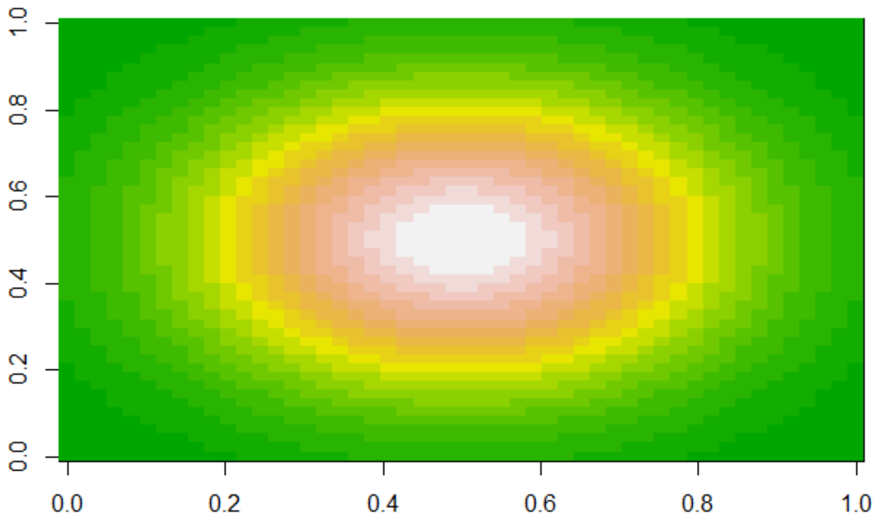
色阶的数量

```
image(normal.mat, col=heat.colors(12)) # This was the old default!
```

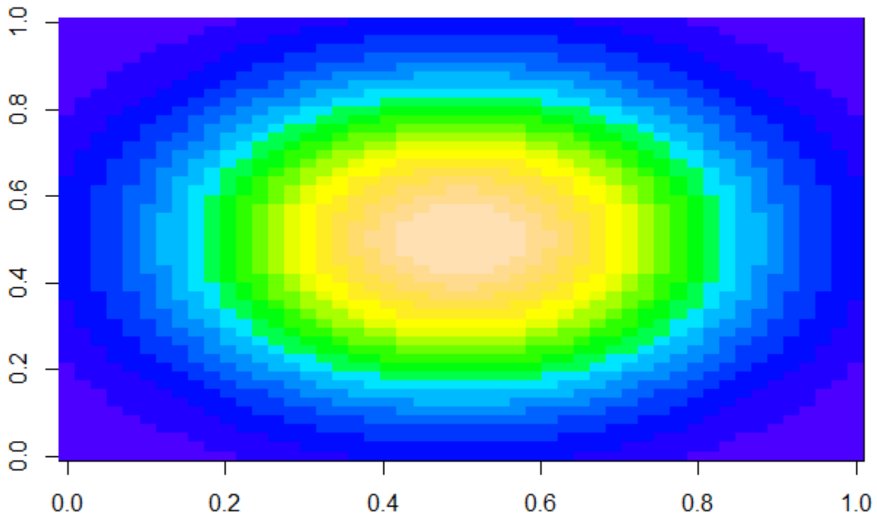



```
image(normal.mat, col=terrain.colors(20)) # Terrain colors
```

(通常用于画 map)



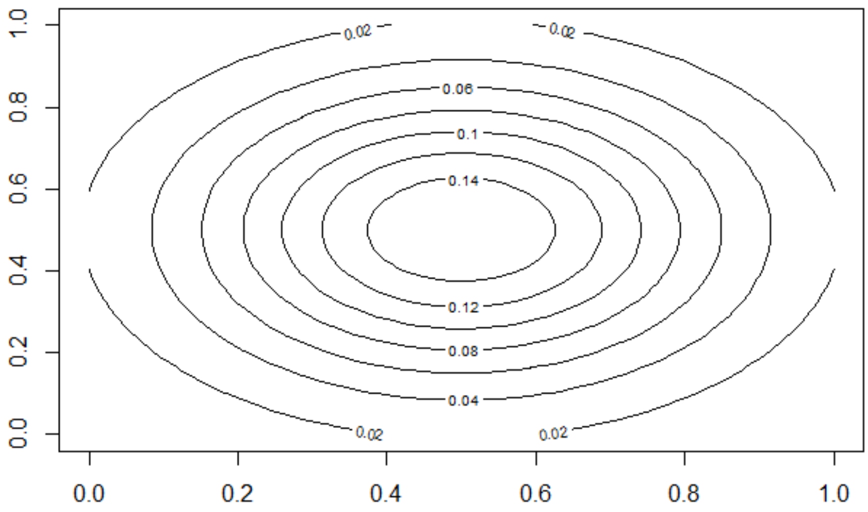

```
image(normal.mat, col=topo.colors(20)) # Topological colors
```



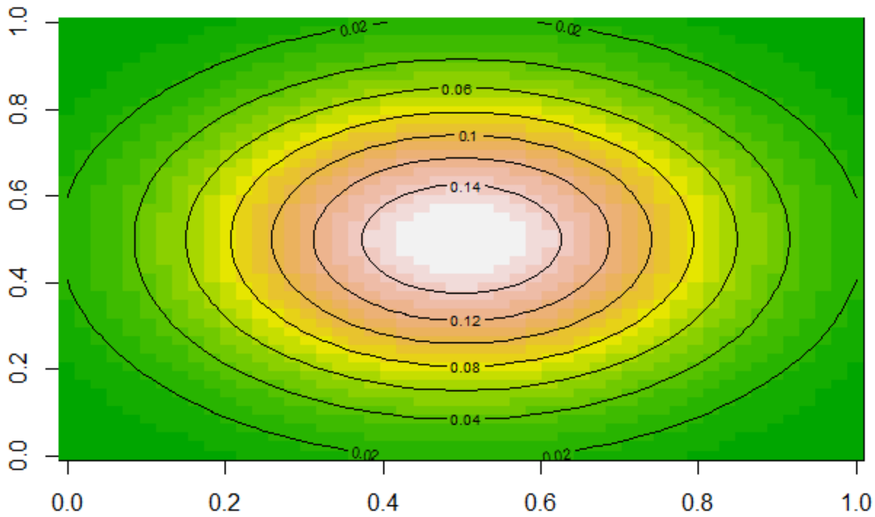
Drawing contour lines

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

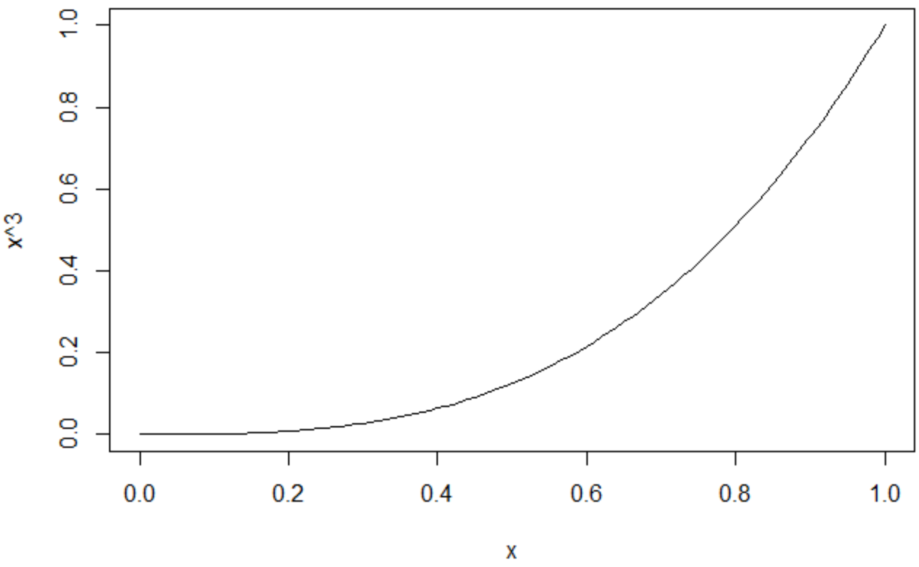
To draw a curve of a function, use `curve()`

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

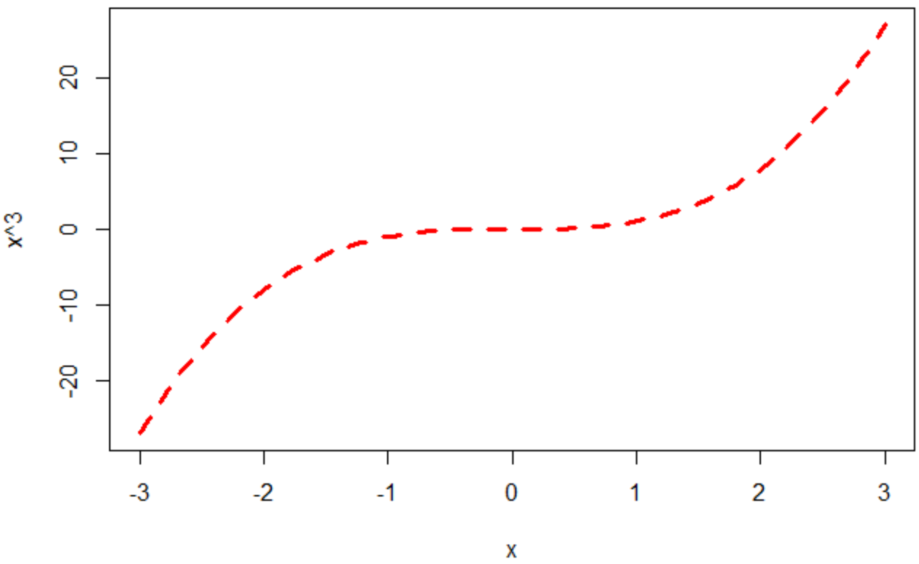
curve 的 argument 是表达式

abline 的 argument 是 a 和 b

lines 的 argument 是点集




```
curve(x^3, from=-3, to=3, lwd=3, lty = 2, col="red") # More plotting options
```

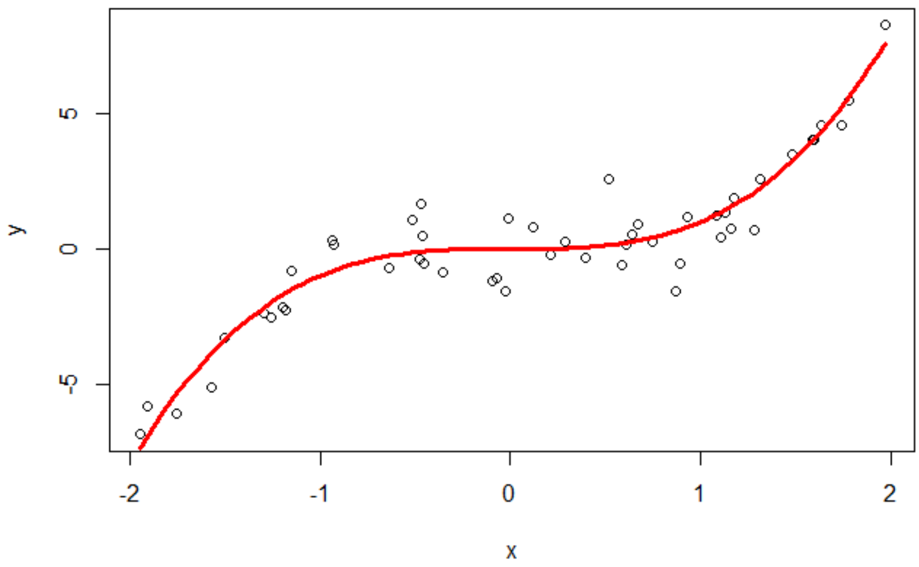


- `lwd` controls the width of lines, while `lty` controls line types

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)
```

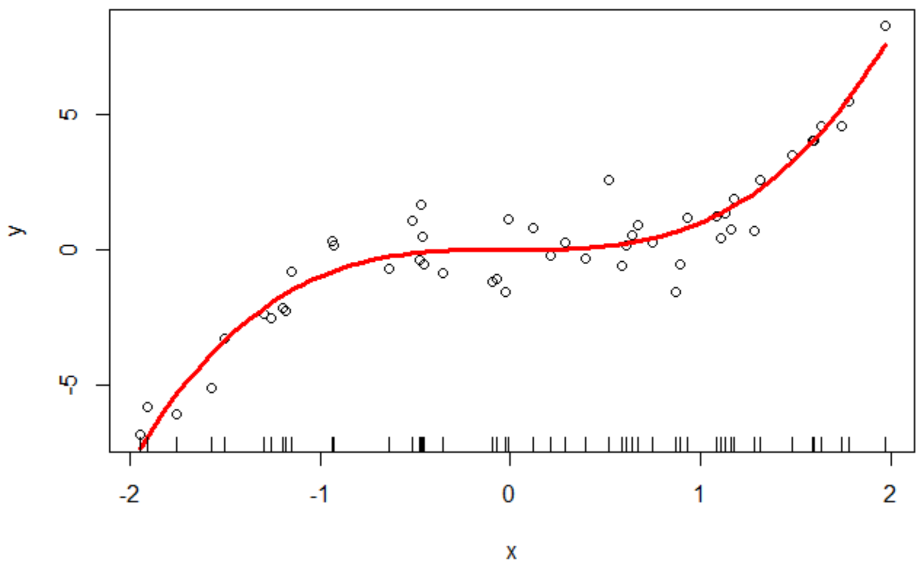


- `x` in `plot` function is the vector we have defined above, while `x` in `curve` is a symbol in `x^3`.
- Again, `x` ranges from 0 to 1 in `curve` by default

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)
```



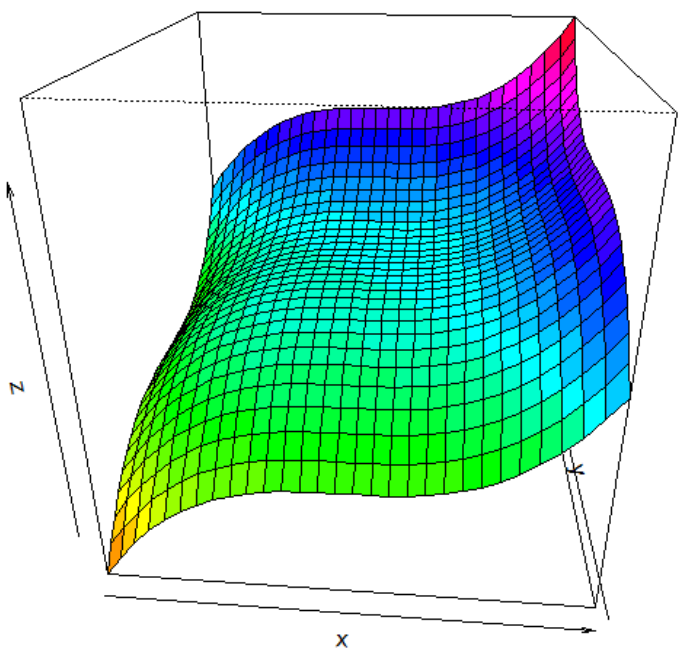
Drawing a surface

Similar to the structure of `curve`, `surface` function can be used to draw a surface in 3D space. It replies on the built-in `persp()` function

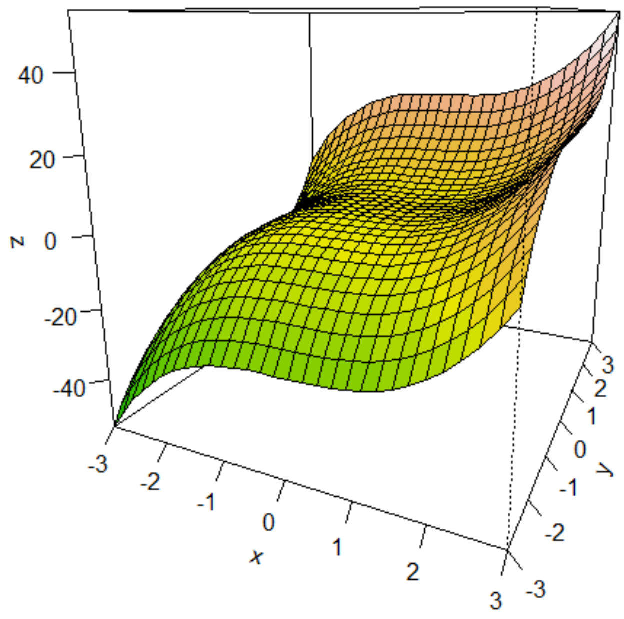
```
surface = function(expr, from.x=0, to.x=1, from.y=0, to.y=1, n.x=30,
  n.y=30, col.list=rainbow(30), theta=5, phi=25, mar=c(1,1,1,1), ...) {
```

控制3-D图的视角

```
  # Build the 2d grid
  x = seq(from=from.x,to=to.x,length.out=n.x)
  y = seq(from=from.y,to=to.y,length.out=n.y)
  plot.grid = expand.grid(x=x,y=y)   900x2

  # Evaluate the expression to get matrix of z values
  uneval.expr = substitute(expr)
  z.vals = eval(uneval.expr,envir=plot.grid)
  z = matrix(z.vals,nrow=n.x)   30x30
                                            为什么去掉第一行和第一列？因为需要填充的格子数为29×29
  # Figure out margins
  orig.mar = par()$mar # Save the original margins
  par(mar=mar)
  col.grid = col.list[round((z[-1,-1]-min(z))/(max(z)-min(z))   把z∈[-54,54] map到{1,2,...,30}上
    * (length(col.list)-1) + 1)]
  r = persp(x,y,z,theta=theta,phi=phi,col=col.grid,...)
  par(mar=orig.mar) # Restore the original margins
  invisible(r) # Return the persp object invisibly
}

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` control the viewing direction of the 3D graph. `theta` gives the azimuthal direction and `phi` provides the colatitude or elevation direction.

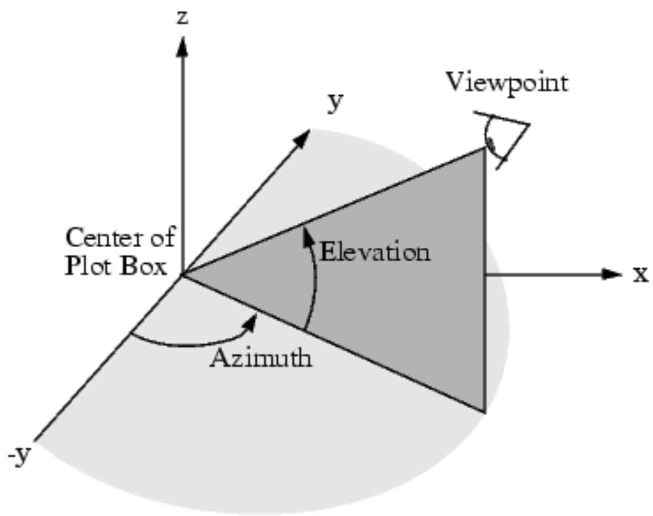


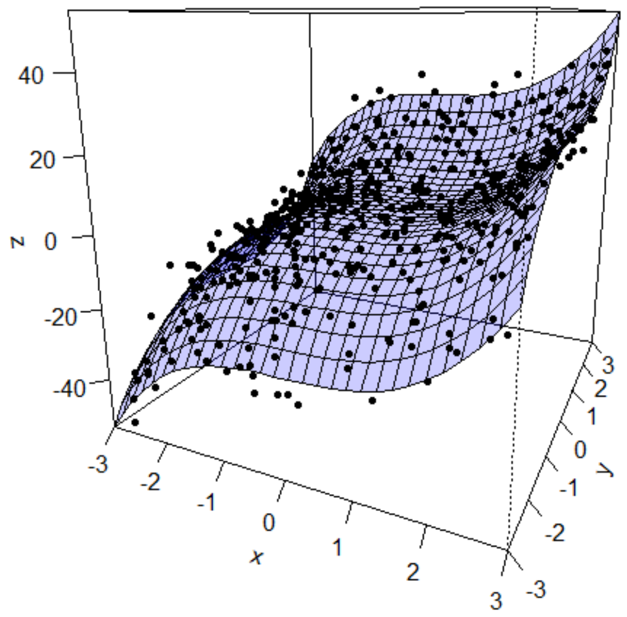
Diagram: azimuthal direction and colatitude

Adding points to a surface

只能在2D图上画出points

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))

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)
```

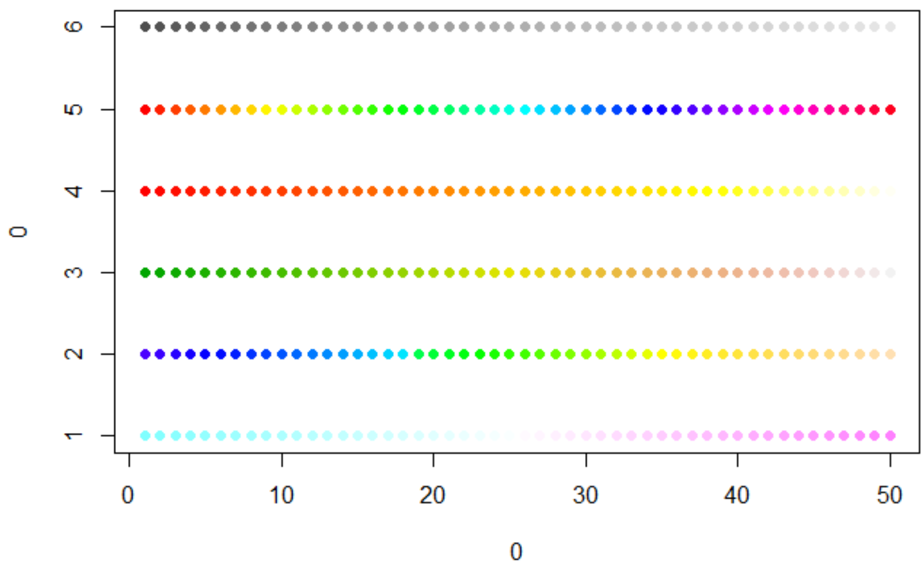


- The forth argument of `trans3d` is a 4×4 viewing transformation matrix, which can be returned by `persp`.

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

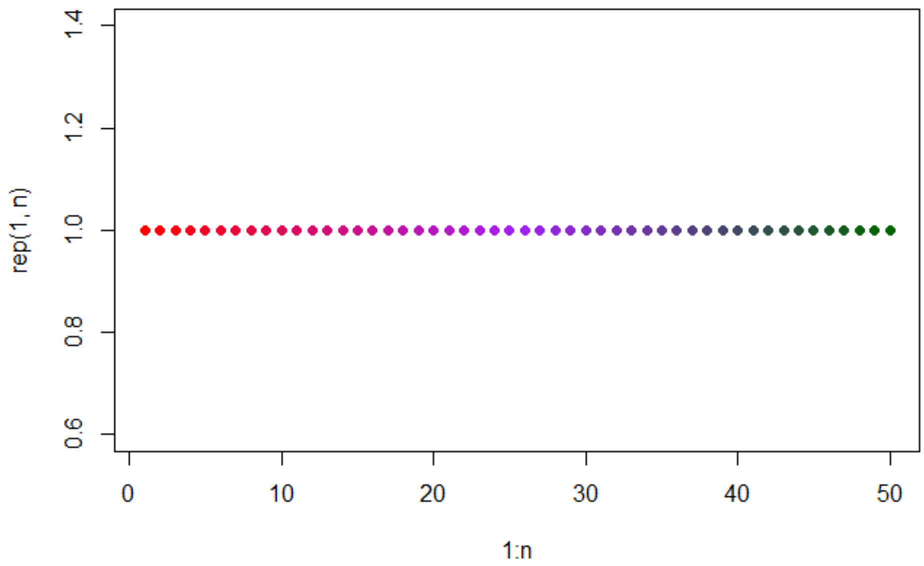
To create a custom palette, that interpolates between a set of base colors, `colorRampPalette()`

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

(是一个function)

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

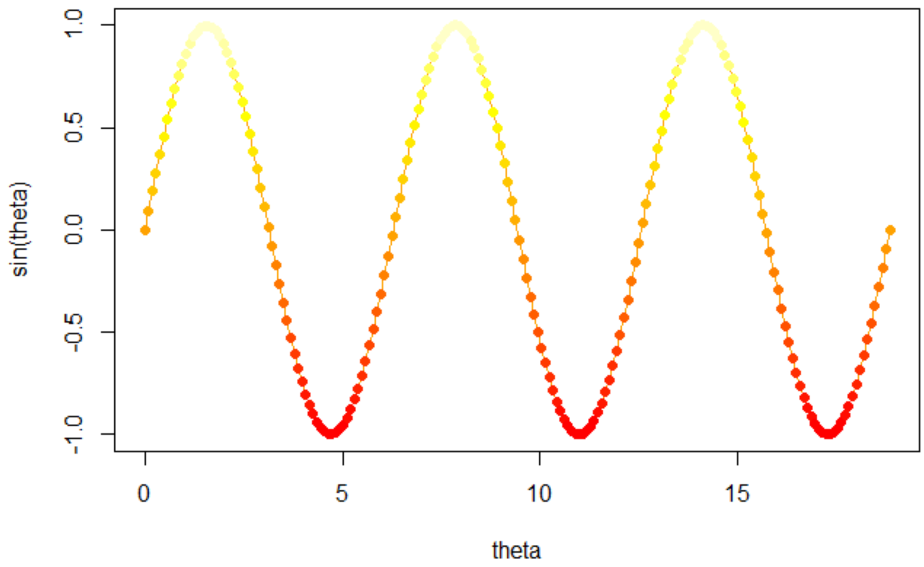
```
plot(1:n, rep(1,n), col=cust.colors(n), pch=19)
```



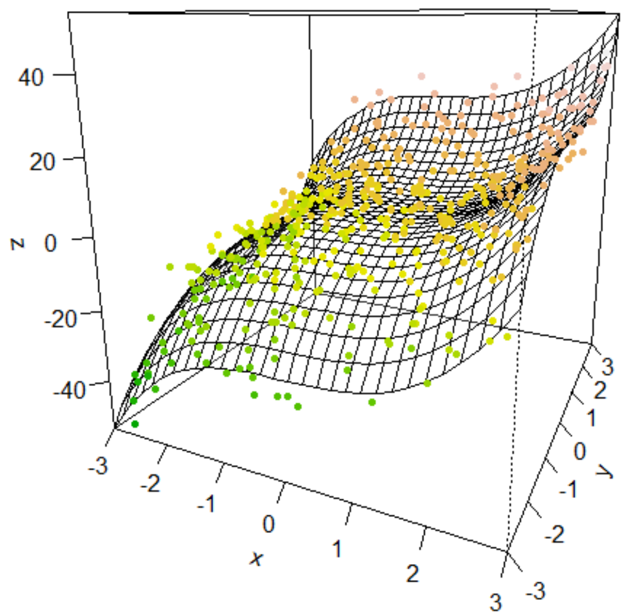
Coloring points by value

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))
```



- Double indices is coerced as integer indices implicitly by rounding down.

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