Line R-rt

Paul Johnson[1]

[1] Center for Research Methods and Data Analysis

2018

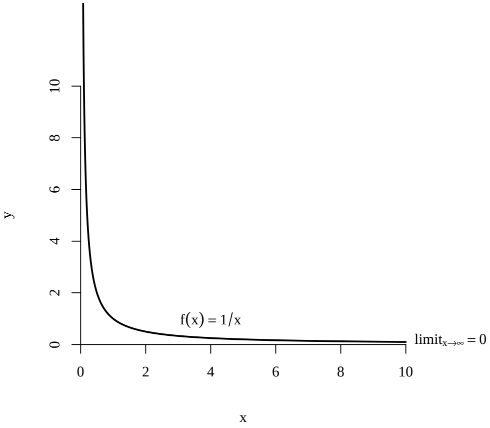
Outline

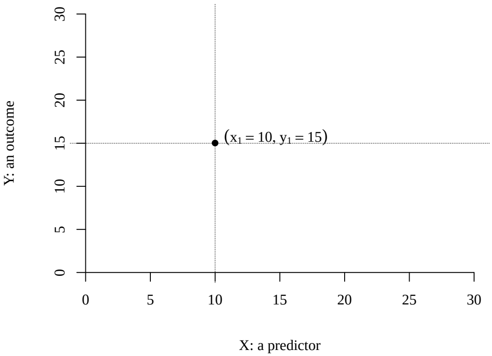
Outline

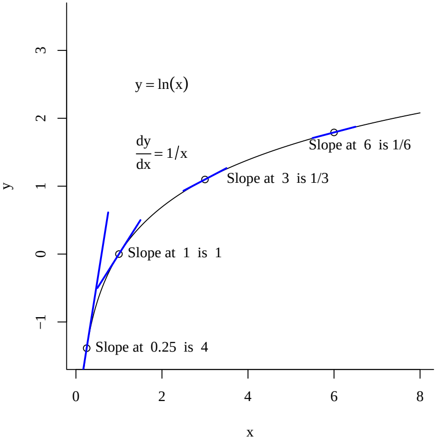
graphics!

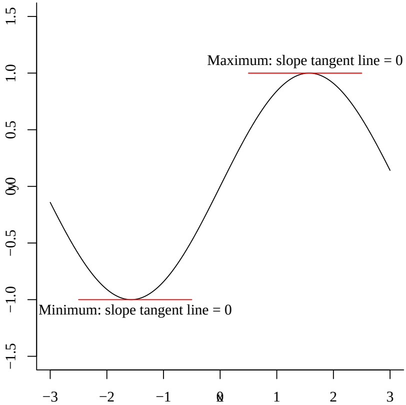
- In papers and reports, we often need technical illustrations
- Publishers refer to illustrations of this sort as “line art”, it must be supplied by authors in high-resolution graphics files (pdf, svg, tiff, etc)
- One can draw sketches by hand, of course, but almost nobody can make a publishable drawing by hand anymore
- R(R Core Team, 2017) offers a suite of functions that can be used to create artwork.

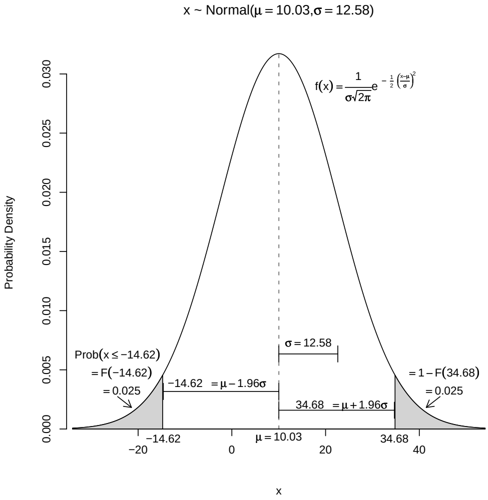
Outline

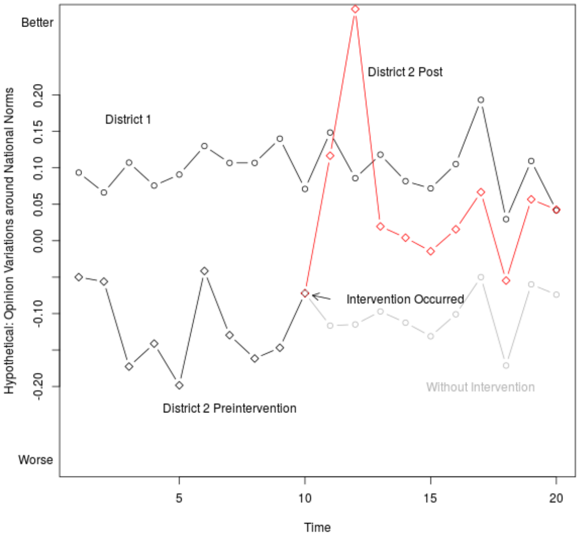

f(x) = 1/x
limit$_{x \to \infty}$ = 0
y
x
0
2
4
6
8
10

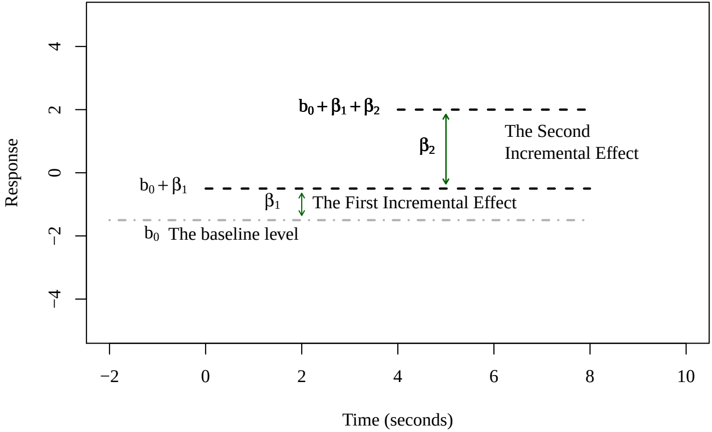

Y: an outcome
0
5
10
15
20
25
30
(x1 = 10, y1 = 15)
X: a predictor


y = ln(x)
dy/dx = 1/x
Slope at 6 is 1/6
Slope at 3 is 1/3
Slope at 1 is 1
Slope at 0.25 is 4
y
x
3
2
1
0
−1
0
2
4
6
8


Maximum: slope tangent line = 0
Minimum: slope tangent line = 0
y
x
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2
3

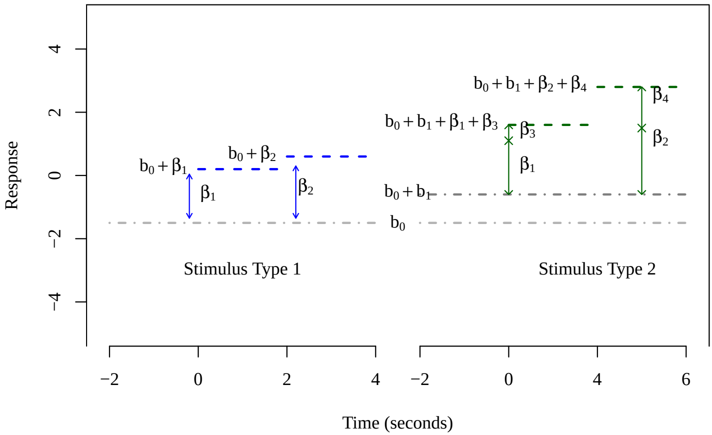

x ~ Normal(μ = 10.03,σ = 12.58)
f(x) = 1/(σ√2π) e^(−½((x−μ)/σ)²)
Probability Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
Prob(x ≤ −14.62)
= F(−14.62)
= 0.025
−14.62 = μ − 1.96σ
σ = 12.58
34.68 = μ + 1.96σ
= 1 − F(34.68)
= 0.025
−14.62
μ = 10.03
34.68
−20
0
20
40
x

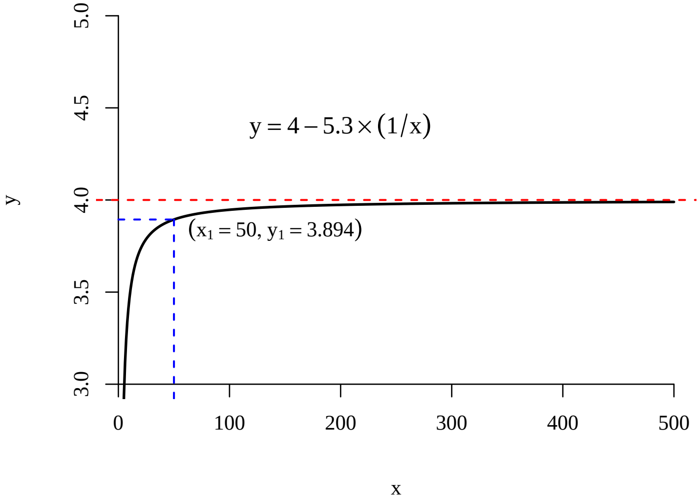

Better
Worse
Hypothetical: Opinion Variations around National Norms
-0.20
-0.10
0.00
0.05
0.10
0.15
0.20
District 1
District 2 Post
Intervention Occurred
Without Intervention
District 2 Preintervention
5
10
15
20
Time



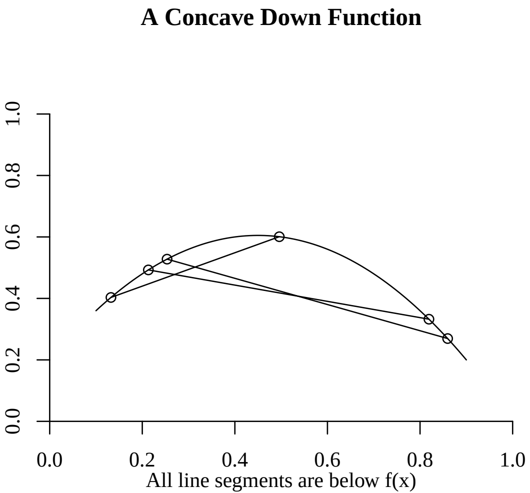
$b_0 + \beta_1 + \beta_2$
β_2
The Second
Incremental Effect
$b_0 + \beta_1$
β_1
The First Incremental Effect
b_0 The baseline level
Response
Time (seconds)
4
2
0
−2
−4
−2
0
2
4
6
8
10

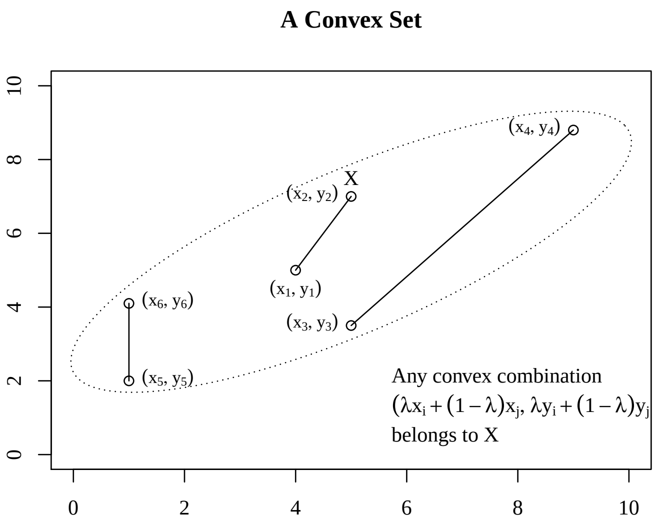
Comparing Stimulus Types: Shared Baseline Approach





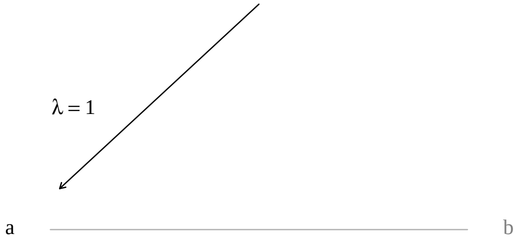
y = 4 – 5.3 × (1/x)
(x1 = 50, y1 = 3.894)
5.0
4.5
4.0
3.5
3.0
y
0
100
200
300
400
500
x


A Concave Down Function
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
All line segments are below f(x)


A Convex Set
X
(x2, y2)
(x1, y1)
(x4, y4)
(x3, y3)
(x6, y6)
(x5, y5)
Any convex combination
(λxi + (1 − λ)xj, λyi + (1 − λ)yj
belongs to X
0
2
4
6
8
10
0
2
4
6
8
10

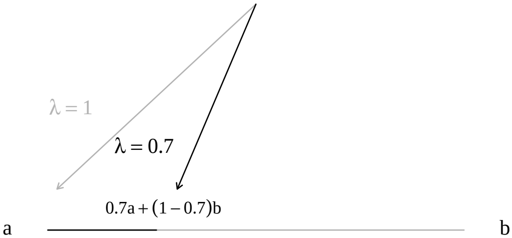
The convex combination

$$\lambda a + (1 - \lambda) b$$



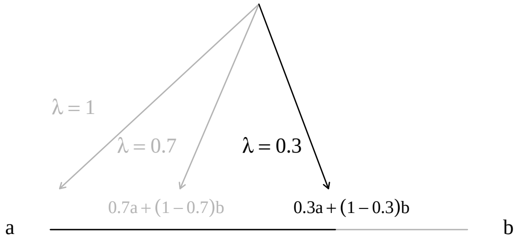
The convex combination

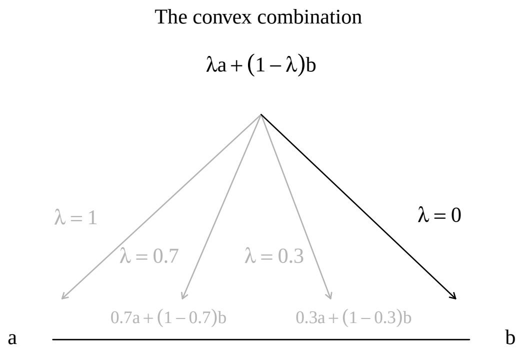
$$\lambda a + (1 - \lambda)b$$

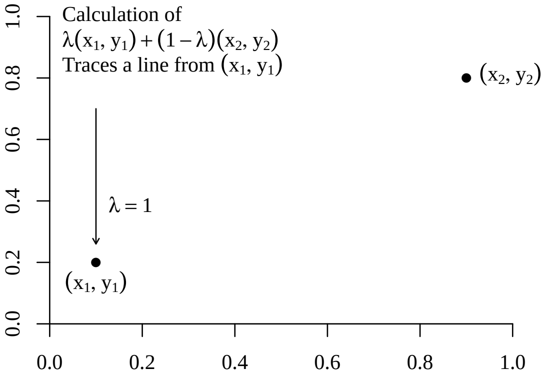


The convex combination

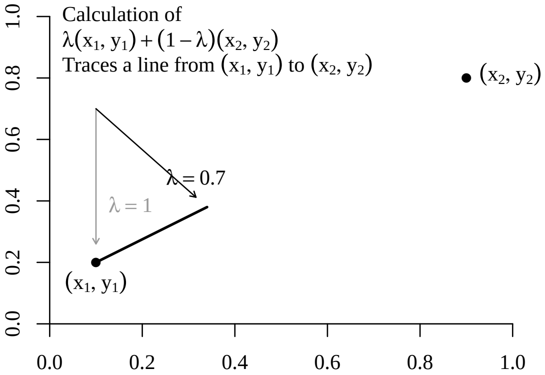
$$\lambda a + (1 - \lambda) b$$

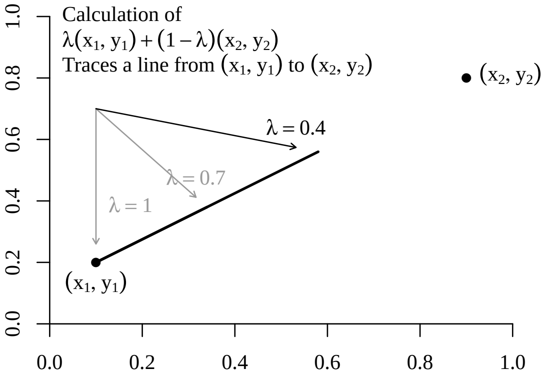


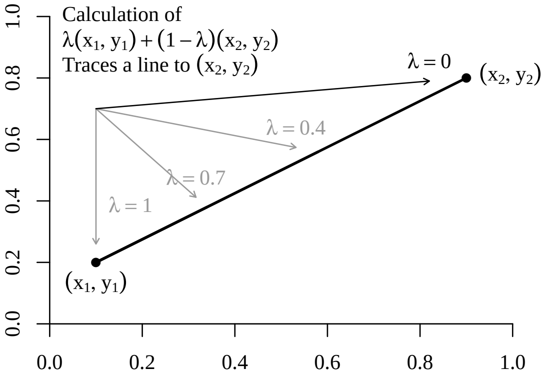

The convex combination
λa + (1 − λ)b
λ = 1
λ = 0.7
λ = 0.3
λ = 0
0.7a + (1 − 0.7)b
0.3a + (1 − 0.3)b
a
b

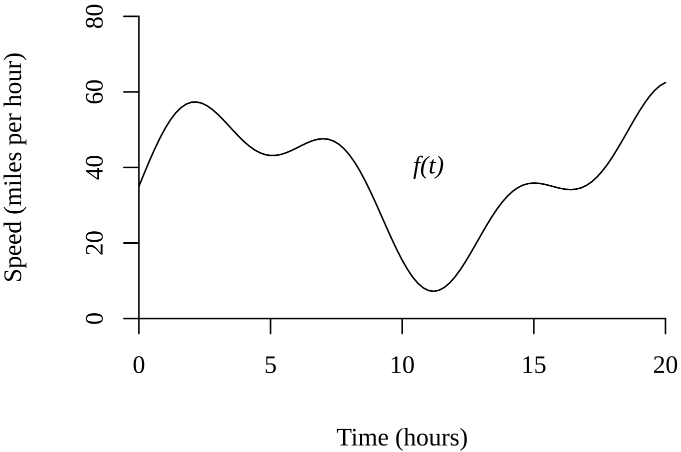


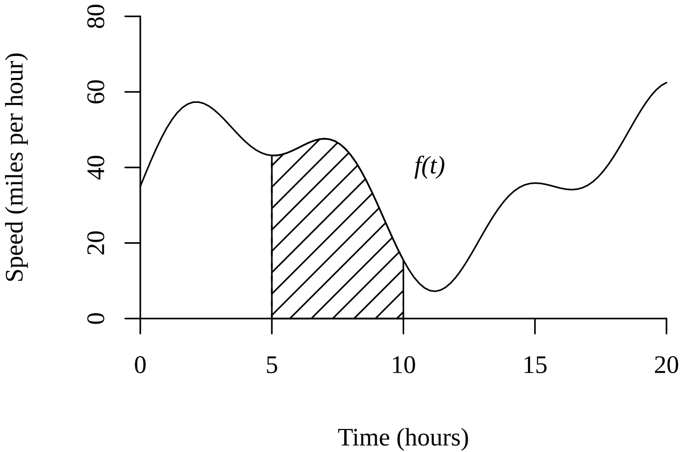
Calculation of
$\lambda(x_1, y_1) + (1-\lambda)(x_2, y_2)$
Traces a line from (x_1, y_1)
(x_2, y_2)
$\lambda = 1$
(x_1, y_1)
0.0
0.2
0.4
0.6
0.8
1.0

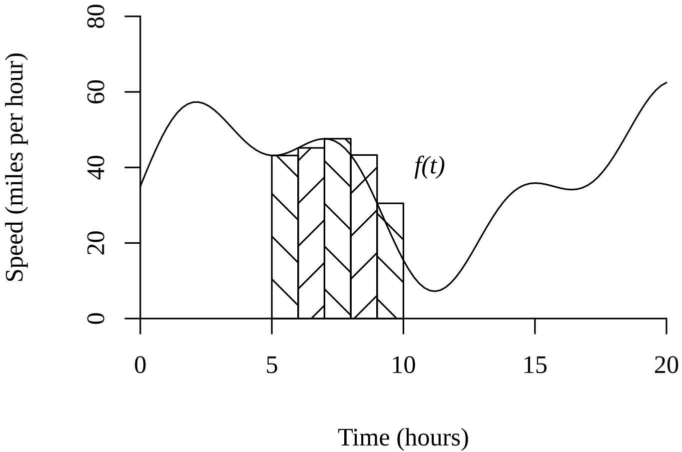

Calculation of
λ(x1, y1) + (1 − λ)(x2, y2)
Traces a line from (x1, y1) to (x2, y2)
(x2, y2)
λ = 0.7
λ = 1
(x1, y1)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

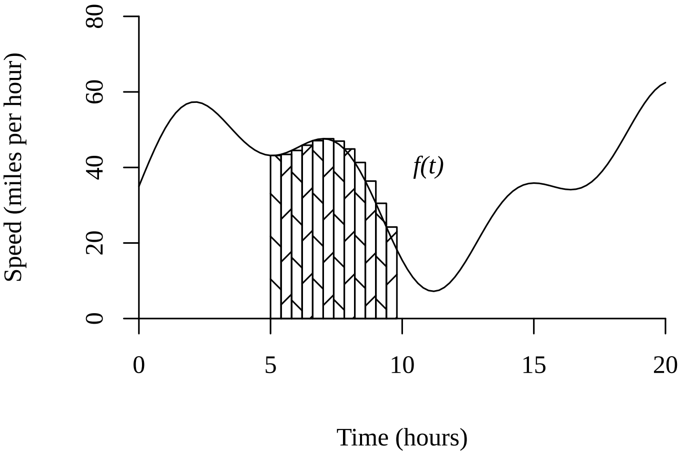

Calculation of
λ(x1, y1) + (1 − λ)(x2, y2)
Traces a line from (x1, y1) to (x2, y2)
(x2, y2)
λ = 0.4
λ = 0.7
λ = 1
(x1, y1)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

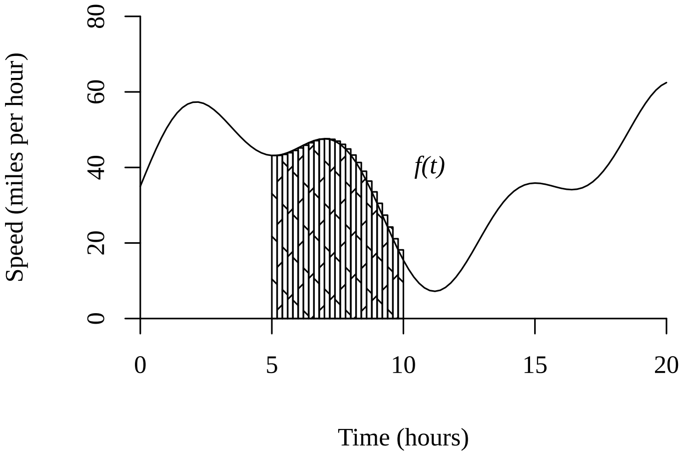

Calculation of
λ(x1, y1) + (1 − λ)(x2, y2)
Traces a line to (x2, y2)
λ = 0
(x2, y2)
λ = 0.4
λ = 0.7
λ = 1
(x1, y1)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0
20
40
60
80
Speed (miles per hour)
f(t)
0
5
10
15
20
Time (hours)

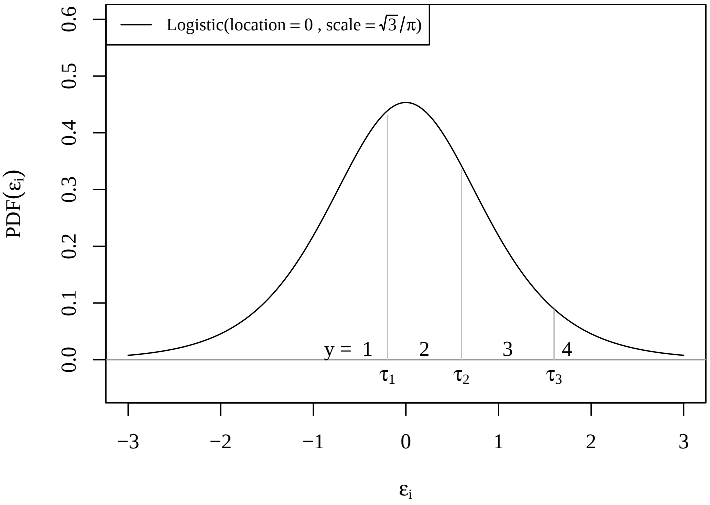

Speed (miles per hour)
0
20
40
60
80
f(t)
0
5
10
15
20
Time (hours)

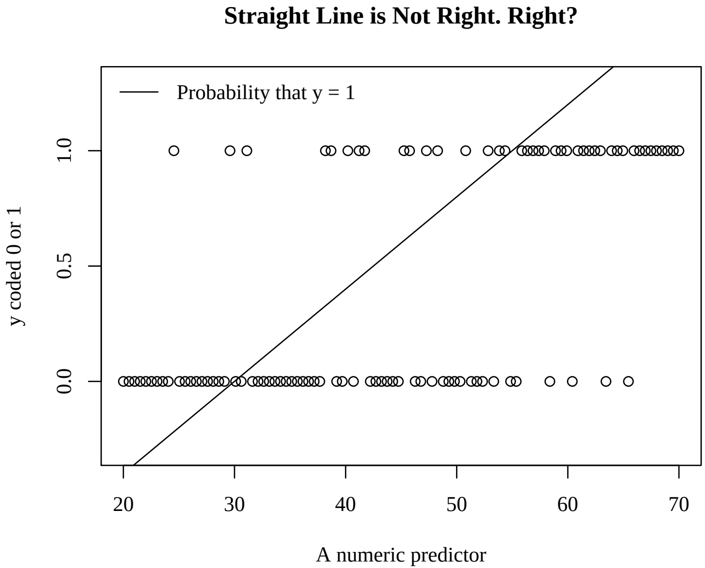

f(t)
Speed (miles per hour)
0
20
40
60
80
0
5
10
15
20
Time (hours)

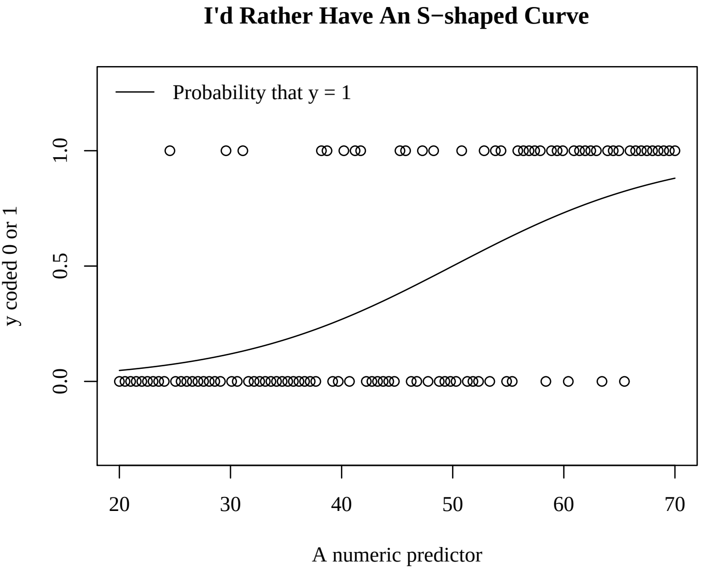

Speed (miles per hour)
0
20
40
60
80
f(t)
0
5
10
15
20
Time (hours)

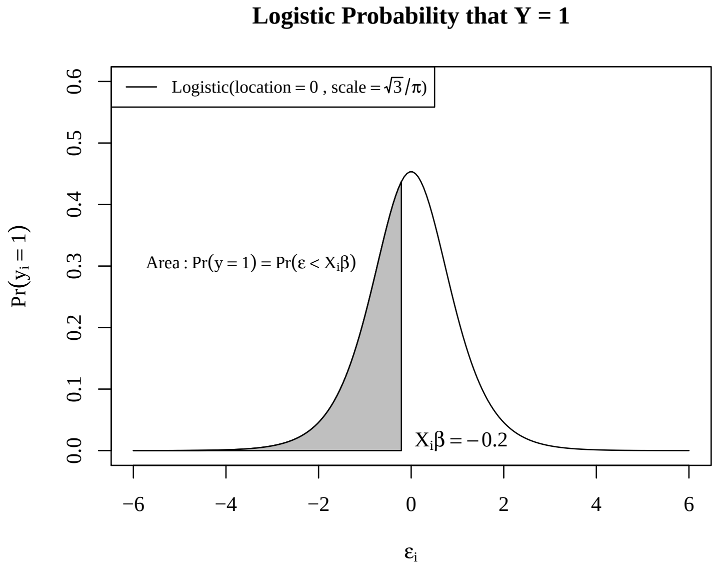

Speed (miles per hour)
0
20
40
60
80
f(t)
0
5
10
15
20
Time (hours)

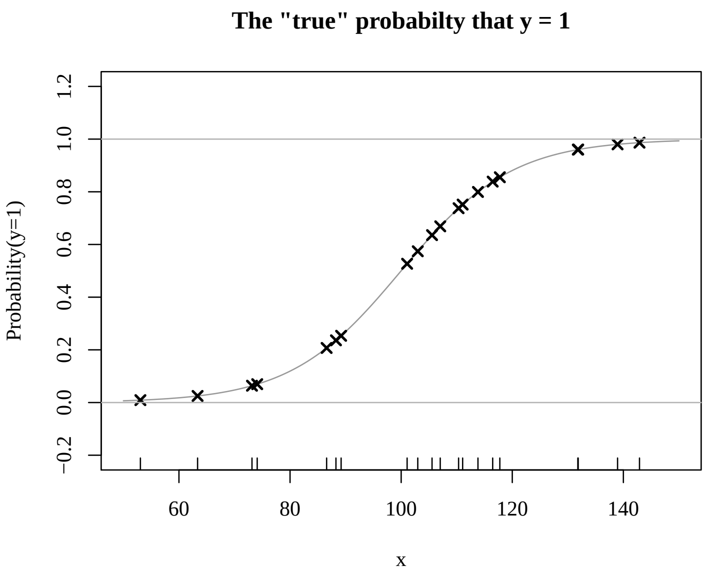
From a Logistic Regression Lecture



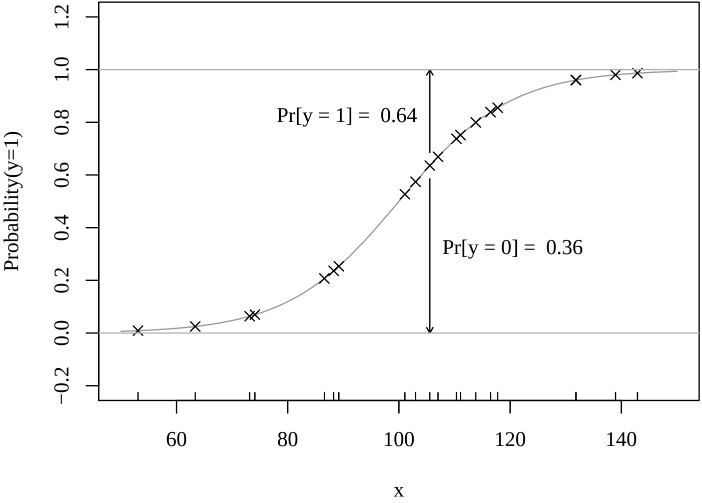

Straight Line is Not Right. Right?
Probability that y = 1
y coded 0 or 1
0.0
0.5
1.0
20
30
40
50
60
70
A numeric predictor


I'd Rather Have An S−shaped Curve
Probability that y = 1
y coded 0 or 1
0.0
0.5
1.0
20
30
40
50
60
70
A numeric predictor

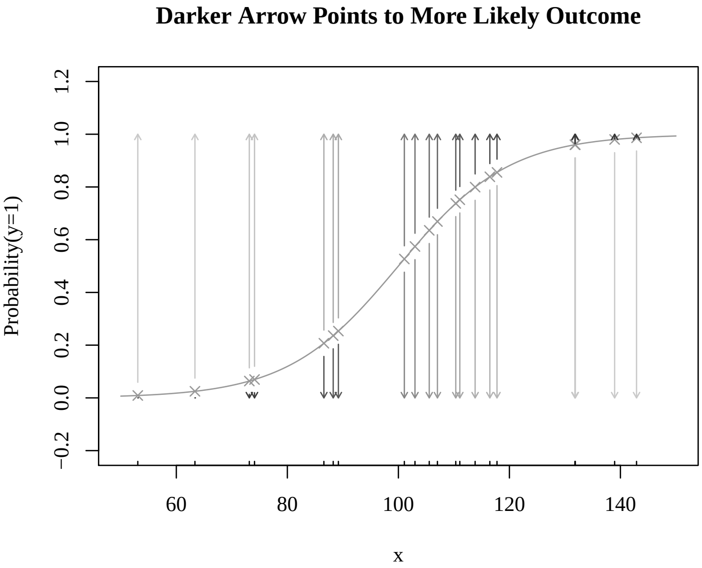

Logistic Probability that Y = 1
Logistic(location = 0 , scale = √3/π)
Area : Pr(y = 1) = Pr(ε < Xiβ)
Xiβ = −0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Pr(yi = 1)
−6
−4
−2
0
2
4
6
εi



The "true" probabilty that y = 1
Probability(y=1)
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
60
80
100
120
140
x



Pr[y = 1] = 0.64
Pr[y = 0] = 0.36
Probability(y=1)
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
60
80
100
120
140
x


Darker Arrow Points to More Likely Outcome
Probability(y=1)
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
60
80
100
120
140
x

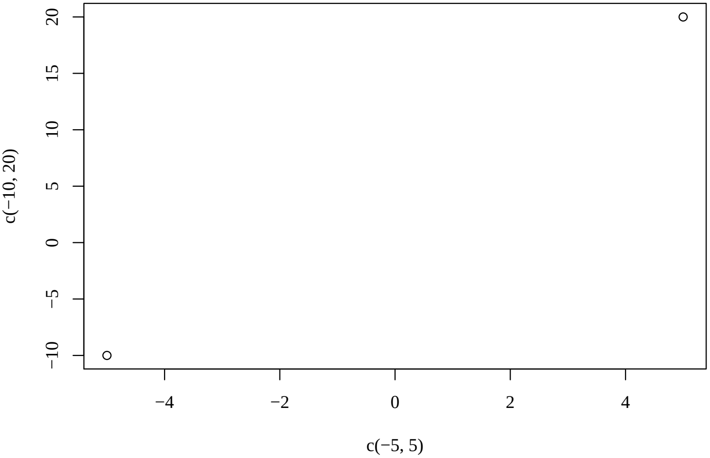
Source Code Available

- The R files I used to produce these graphs are in the R folder distributed with this project
- The output files (displayed above) are in the output folder

Outline

- Get a separate "device" display window
  - `dev.new()`
  - If in RStudio, `dev.new()` is blocked, must run
    - MS windows: windows()
    - Mac: quartz()
    - Linux: X11()
- Create a drawing region inside there.
  - I choose to have x scale go from -5 to +5 and y from -10 to 20

```
plot(x = c(-5, 5), y = c(-10, 20))
```

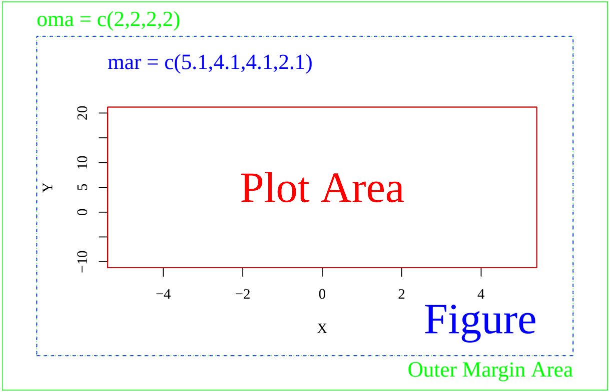

20
15
10
5
0
−5
−10
c(−10, 20)
−4
−2
0
2
4
c(−5, 5)

Oops, I forgot to hide border, axes, labels, and points

```
plot(x = c(-5, 5), y = c(-10, 20), type = "n",
    axes = FALSE, xlab = "", ylab = "")
```

Result: blank sheet of paper

Draw inside the Plot Area



Defaults

- margins asymmetric (measure: lines of text)
- most commands we use write only in the Plot Area

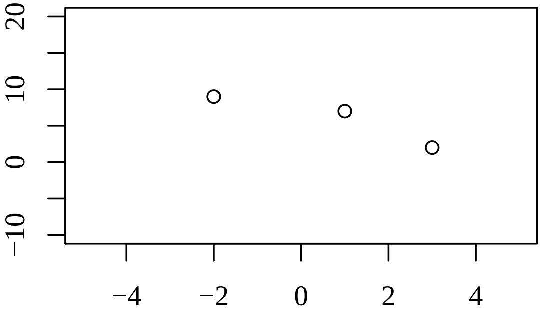
Here is the plan of attack

Demonstrate various drawing functions in R. For each we need to

1. Run blank sheet creator
2. Draw on the sheet
3. Save or Throw away that sheet.
4. Start over.
   (There is no eraser!)

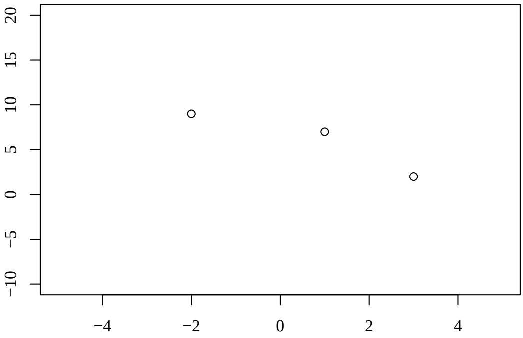
Outline

```
plot(x = c(-5, 5), y = c(-10, 20),
     type = "n", xlab = "", ylab = "")
points(x = c(-2, 1, 3), y = c(9, 7, 2))
```



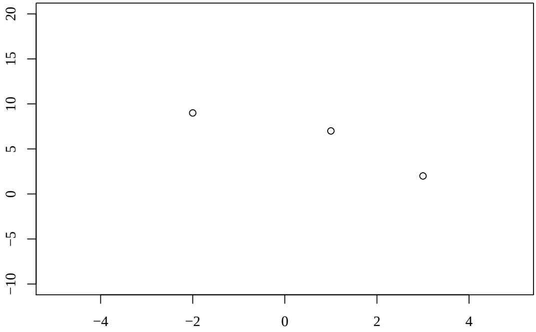
Create x and y vectors separately

```
plot(x = c(-5, 5), y = c(-10, 20), type = "n",
    xlab = "", ylab = "")
x <- c(-2, 1, 3)
y <- c(9, 7, 2)
points(x = x, y = y)
## Same result as
## points(x, y)
## because of R positional matches
```

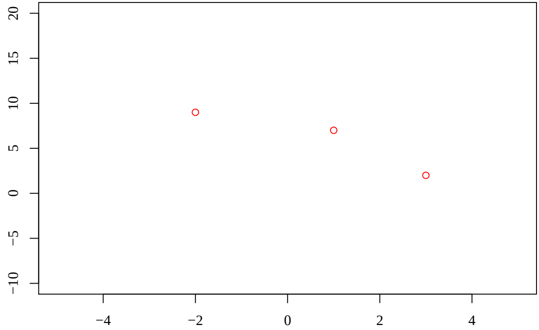

20
15
10
5
0
−5
−10
−4
−2
0
2
4

bdraw() is a little function, it re-draws the graph area for me (same as typing plot command)

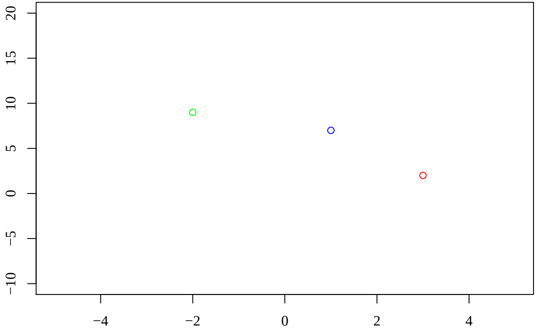
```
bdraw <- function() {
     plot(x = c(-5, 5), y = c(-10, 20),
          type = "n", xlab = "", ylab = "")
 }
bdraw()
points(x = x, y = y)
```


20
15
10
5
0
−5
−10
−4
−2
0
2
4

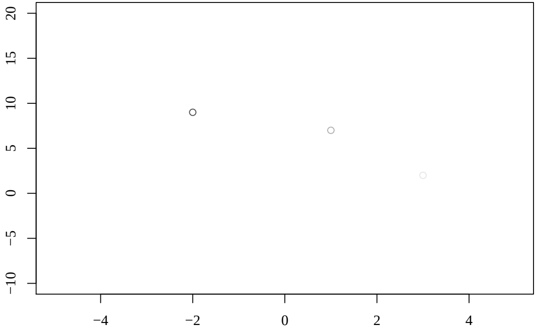
```
bdraw()
points(x = x, y = y, col = "red")
```



```
bdraw()
points(x = x, y = y, col = c("green", "blue",
    "red"))
```



```
bdraw()
points(x = x, y = y, col = gray.colors(3))
```



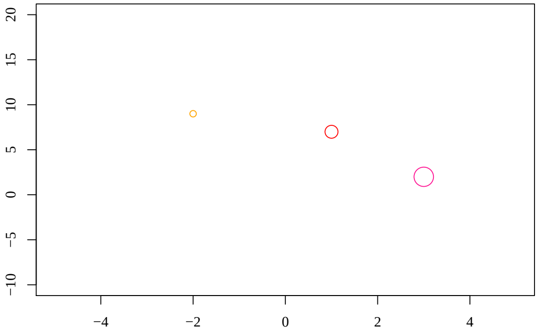
RTFM

- ?points. See arguments
  - pch: plot symbol
  - lwd: thickness of line in drawing
  - cex: character expansion: 1 is default
  - bg: background color for outline symbols
- Run example(points)
- ?points.formula
  - allows syntax like points(y ~ x, data = dat)

Try some practices

```
plot(x = c(-5, 5), y = c(-10, 20), type = "n",
   xlab = "", ylab = "")
x <- c(-2, 1, 3)
y <- c(9, 7, 2)
points(x = x, y = y, cex = c(1, 2, 3), col =
   c("orange", "red", "deeppink"))
```

Try some practices ...



The plot function shortcut . . .

- I want you to understand you can draw points on top of any plot.
- But if you only want points, there is a shortcut

```
plot(y ~ x, axes = FALSE, xlab = "", ylab = "")
```

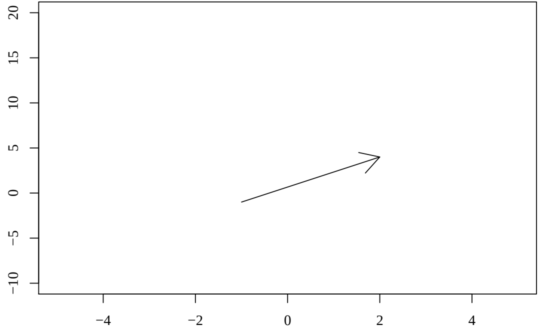
The plot function shortcut . . . ...

Points worth mentioning

- points() are drawn centered on the coordinates in x and y
- for larger symbols, adjust cex
- for darker lines in outlines of symbols, adjust lwd

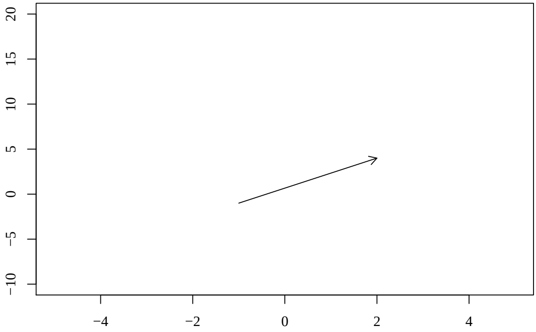
Arrows. Learn by doing!

```
bdraw()
arrows(x0 = -1, y0 = -1, x1 = 2, y1 = 4)
```



Arrows. Smaller fins

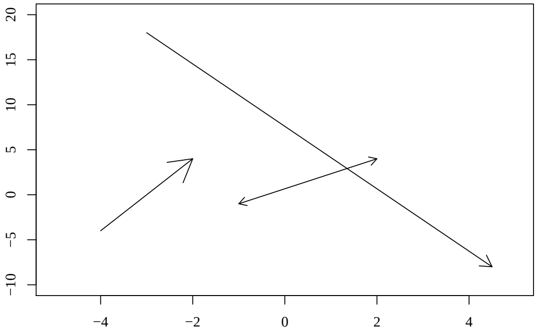
```
bdraw()
arrows(x0 = -1, y0 = -1, x1 = 2, y1 = 4, length =
    0.1)
```



Code 1 2 3

```
bdraw()
# code 3 is both ways
arrows(x0 = -1, y0 = -1, x1 = 2, y1 = 4, length =
    0.1, code = 3)
# code 2 points forwards
arrows(x0 = -4, y0 = -4, x1 = -2, y1 = 4, length
    = 0.3, code = 2)
# code 1 is backwards
arrows(x0 = 4.5, y0 = -8, x1 = -3, y1 = 18,
    length = 0.15, code = 1)
```

Code 1 2 3 ...



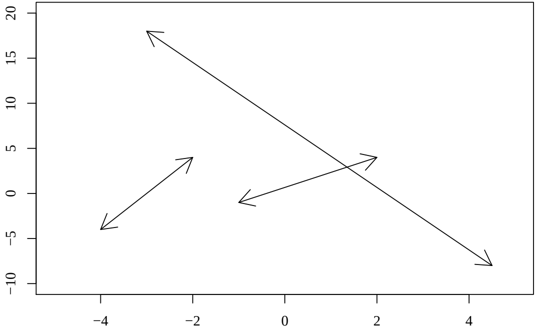
Use one arrows() command

- I tried to show off, but discovered something that looks rather like a weakness in arrows(), possibly even a bug.
- My idea was to stack together the input data

```
x0 <- c(-1, -4, 4.5)
y0 <- c(-1, -4, -8)
x1 <- c(2, -2, -3)
y1 <- c(4, 4, 18)
mylengths <- c(0.2, 0.3, 0.15)
mycodes <- c(3, 2, 1)
bdraw()
arrows(x0 = x0, y0 = y0, x1 = x1, y1 = y1, length
    = mylengths, code = mycodes)
```

Use one arrows() command ...

only the first elements in mylengths and mycodes obeyed.

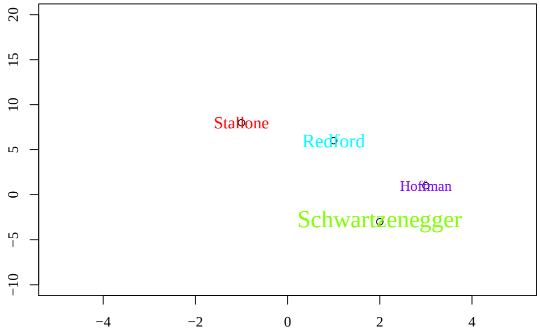


Text is like points, except . . .

- text needs labels, one for each location
- positioning text can be tricky because sometimes we want text above, below, or on side of a point.

```
x <- c(-1, 2, 1, 3); y <- c(8, -3, 6, 1)
labels <- c("Stallone", "Schwartzenegger",
   "Redford", "Hoffman")
mycolors <- rainbow(4)
bdraw()
points(x, y)
text(x = x, y = y, labels = labels, col =
   mycolors, cex = c(1.2, 1.7, 1.3, 1))
```

Text is like points, except . . . ...



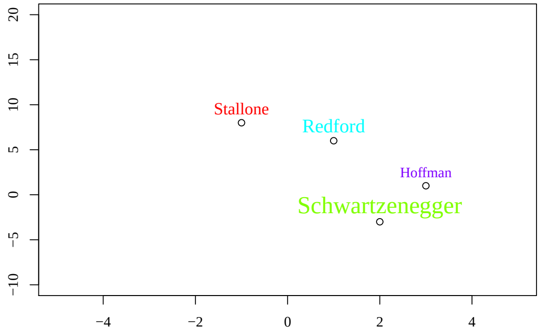
Text labels overlap points, if you are not careful

Default text() has pos = 1.

pos = 3 moves text above the point

```
bdraw()
points(x, y)
text(x = x, y = y, labels = labels, col =
    mycolors, cex = c(1.2, 1.7, 1.3, 1), pos = 3)
```

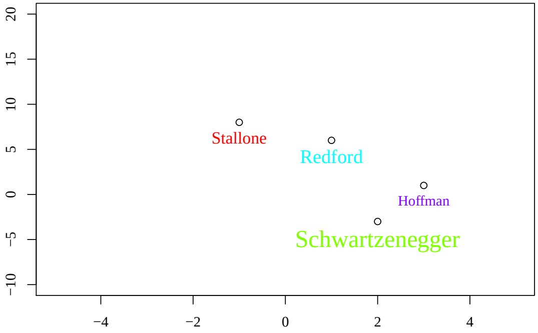
Text labels overlap points, if you are not careful ...



offset needed to write "under" the points

```
bdraw()
points(x, y)
text(x = x, y = y, labels = labels, col =
    mycolors, cex = c(1.2, 1.7, 1.3, 1), pos = 1,
    offset = 0.7)
```

offset needed to write ”under” the points ...



offset units are “character widths”

offset needed to write "under" the points

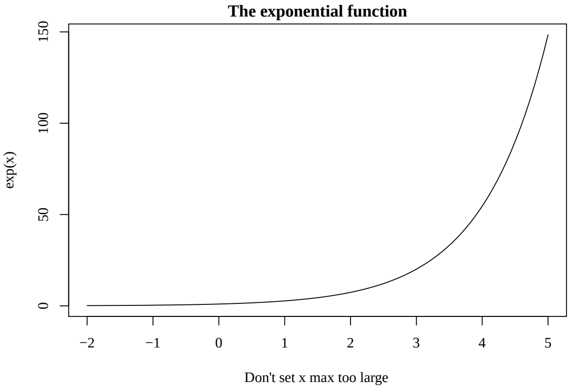
- lines() : will "connect the dots" and do so with some smoothing for pleasant curve
- segments() : straight line connect the dots, no smoothing
- abline() : a "shortcut" function to draw some commonly used straight lines
- curve() : a "shortcut" function for drawing curves for functions of x.

Plotting Functions

- In statistics, we often find transformations like $\exp(x)$ or $\log(x)$
- A good way to learn about them is to plot them with R's curve function
- `curve()` creates its own graphic device, so we don't need to run plot first.

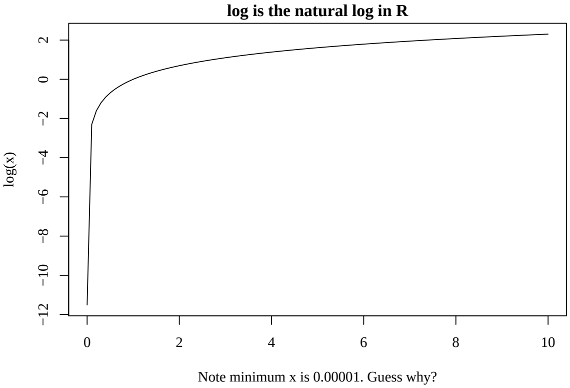
```
curve(exp(x), from = -2, to = 5, xlab = "Don't
    set x max too large", main = "The exponential
    function")
```

Plotting Functions ...



The Natural Logarithm

```
curve(log(x), from = 0.00001, to = 10, xlab =
    "Note minimum x is 0.00001. Guess why?", main
    = "log is the natural log in R")
```

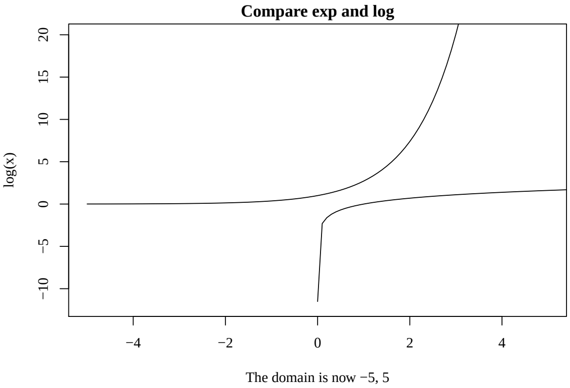


Can "Overlay" curves

- The first curve we draw sets the scale.
- xlim, ylim: arguments so that the scale is big enough to show the interesting parts of both curves.

```
curve(log(x), from = 0.00001, to = 10, xlab =
    "The domain is now -5, 5", main = "Compare
    exp and log", xlim = c(-5, 5), ylim = c(-12,
    20))
curve(exp(x), from = -5, to = 5, add = TRUE)
```

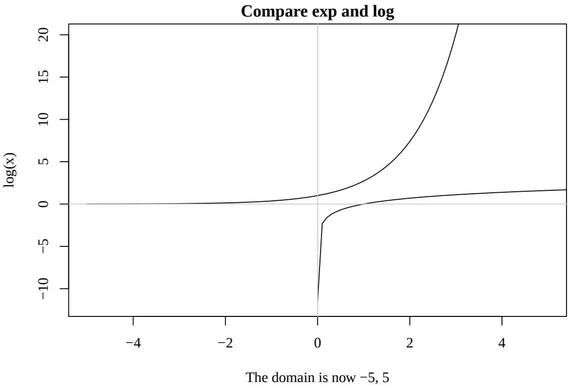
Can "Overlay" curves ...



Insert light reference lines with abline

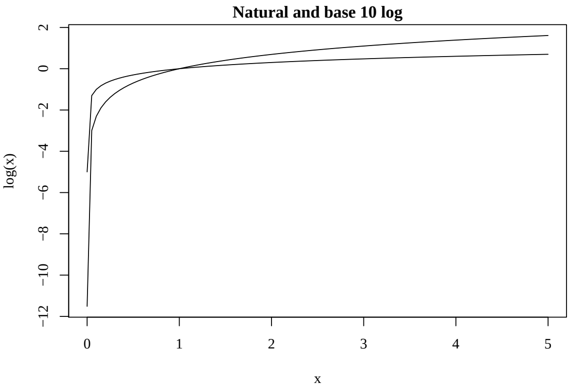
```
curve(log(x), from = 0.00001, to = 10, xlab =
    "The domain is now -5, 5", main = "Compare
    exp and log", xlim = c(-5, 5), ylim = c(-12,
    20))
curve(exp(x), from = -5, to = 5, add = TRUE)
abline(v = 0, col = "gray80")
abline(h = 0, col = "gray80")
```

Insert light reference lines with abline ...



What is the Natural Logarithm?

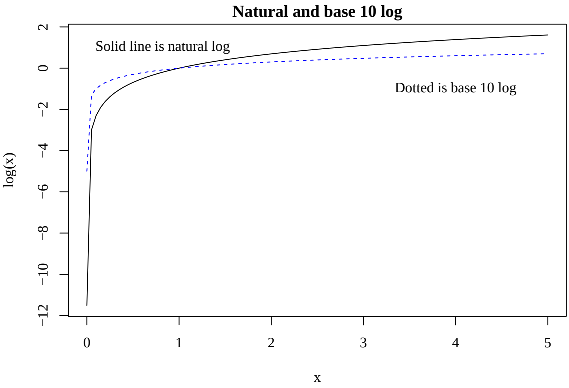
```
curve(log(x), from = 0.00001, to = 5, xlab = "x",
   main = "Natural and base 10 log")
curve(log(x, 10), from = 0.00001, to = 5, add =
   TRUE)
```



I cannot tell those apart!

```
curve(log(x), from = 0.00001, to = 5, xlab = "x",
   main = "Natural and base 10 log")
curve(log(x, 10), from = 0.00001, to = 5, add =
   TRUE, lty = 2, col = "blue")
text(4, -1, "Dotted is base 10 log")
text(0, 1, "Solid line is natural log", pos = 4)
```

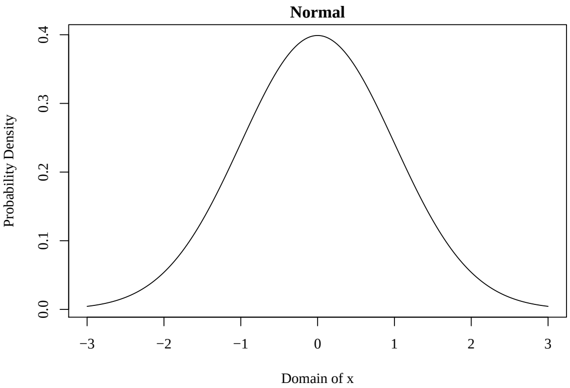
I cannot tell those apart! ...



My favorite use of lines(): draw probability density functions

```
x <- seq(-3, 3, length.out = 200)
xprob <- dnorm(x, m = 0, s = 1)
plot(xprob ~ x, type = "n", xlab = "Domain of x",
    ylab = "Probability Density", main = "Normal")
lines(xprob ~ x)
```

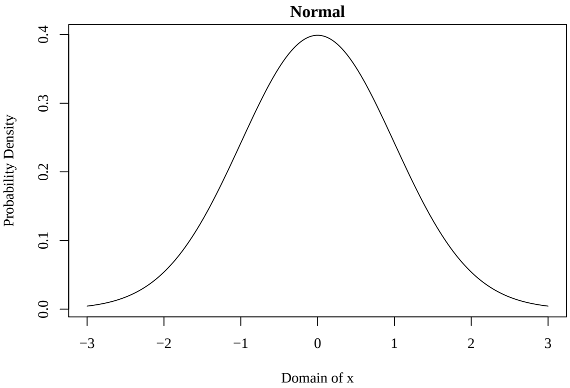
My favorite use of lines(): draw probability density functions ...



dnorm is R's function to calculate probability density of the normal

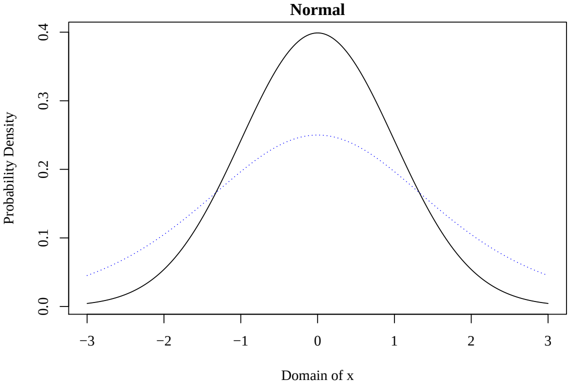
plot type = "l" is a shortcut for that

```
plot(xprob ~ x, type = "l", xlab = "Domain of x",
   ylab = "Probability Density", main = "Normal")
```



Compare densities of 2 different distributions

```
plot(xprob ∼ x, type = "l", xlab = "Domain of x",
    ylab = "Probability Density", main = "Normal")
x2prob <- dlogis(x, location = 0, scale = 1)
lines(x2prob ∼ x, lty = 3, col = "blue")
```

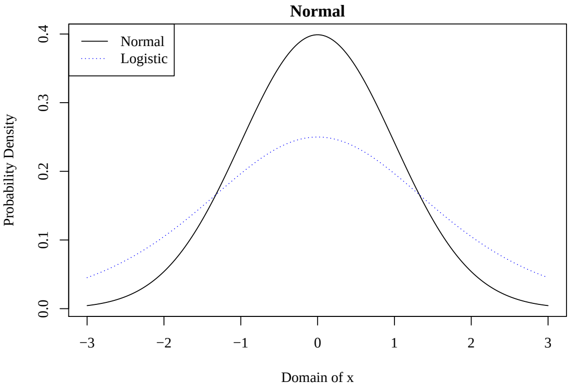


Insert a legend

I've found that getting a legend "just right" can be very frustrating.

```
plot(xprob ~ x, type = "l", xlab = "Domain of x",
    ylab = "Probability Density", main = "Normal")
x2prob <- dlogis(x, location = 0, scale = 1)
lines(x2prob ~ x, lty = 3, col = "blue")
legend("topleft", legend = c("Normal",
    "Logistic"), lty = c(1, 3), col = c("black",
    "blue"))
```

Insert a legend ...

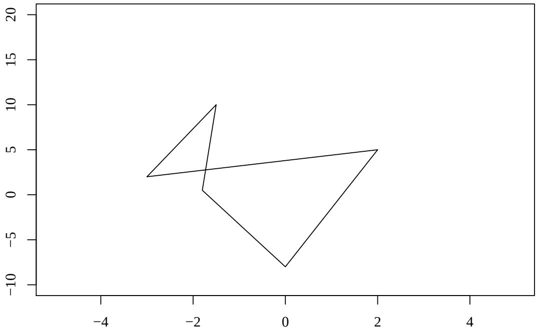


color in shapes

- If you can supply the points, R can draw a smooth, “connect-the-dots” curve, and decorate the insides.

```
bdraw()
x <- c(-3, -1.5, -1.8, 0, 2, -3)
y <- c(2,    10,     0.5, -8, 5, 2)
polygon(x, y)
```

color in shapes ...

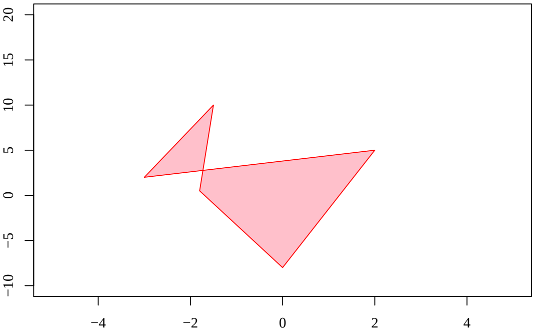


Whoops! I forgot that Splash of Color!

- If you can supply the points, R can draw a smooth, "connect-the-dots" curve, and decorate the insides.

```
bdraw()
polygon(x, y, col = "pink", border = "red")
```

Whoops! I forgot that Splash of Color! ...

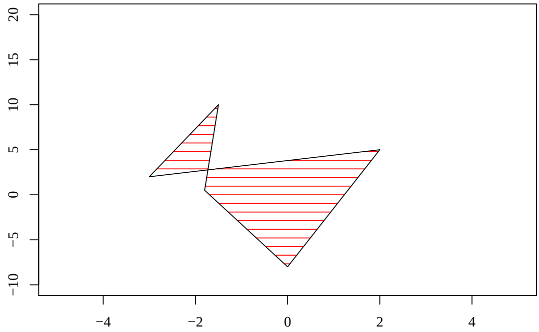


Play with polygons

- density: Instead of coloring background, can draw lines on it.
- angle: direction of lines inside polygon
- If you can supply the points, R can draw a smooth, “connect-the-dots” curve, and decorate the insides.

```
bdraw()
polygon(x, y, col = "red", border = "black",
    density = 10, angle = 0)
```

Play with polygons ...

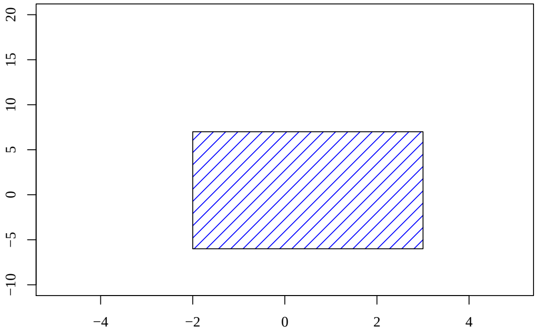


The rect() function is almost identical to polygon

- rect wants 4 arguments, the corner coordinates

```
bdraw()
rect(xleft = -2, ybottom = -6, xright = 3, ytop =
    7, col = "blue", border = "black", density =
    10, angle = 45)
```

The rect() function is almost identical to polygon ...



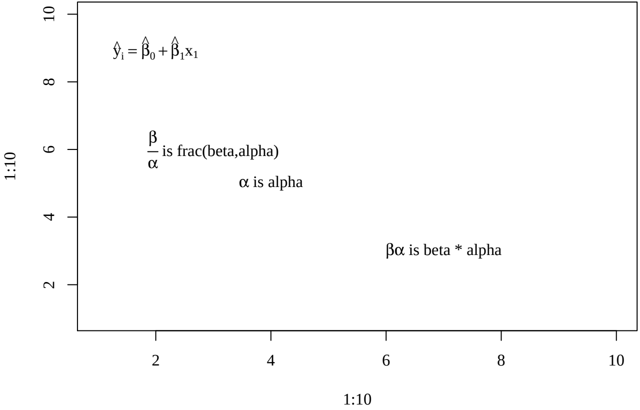
Outline

Sometimes a well placed σ or ψ pushes your plot over the top

- I don't want to spend a lot of time on this, because it is almost mind-numbingly complicated in some ways, but let's just run an example.

```
plot(1:10, 1:10, type="n")
text(4, 5, expression(paste(alpha ," is alpha")))
text(7, 3, expression(paste(beta * alpha, " is beta * alpha")))
text(3, 6, expression(paste(frac(beta, alpha), " is frac(beta,alpha)")))
text(2,9, expression(paste(hat(y)[i] == hat(beta)[0]+hat(beta)[1]*x[1])))
```

I also like β, α and Σ



A Few plotmath Tips

- Two Equal Signs ($==$ gives back $=$)
- Use hard brackets [] for subscripts, ^ for superscripts
- Want * to show? Type %*%
- Want centered $\cdot$ for multiplication? Type cdot
- Want $(x - 1, y_1)$? Type group("(", list(x[1], y[1]), ")")

Outline

What To Practice Today?

- Maybe this will get you started

```
plot(1:10, 1:10, type = "n")
abline(h = 2:9, v = c(3, 5, 7), col =
    "gray80")
arrows(x0 = 2, y0 = 3, x1 = 9, y1 = 2, length
    = 0.1)
text(3, 7, "Kansas in Summer is like Paris",
    pos = 4)
text(3.2, 6.6, "if Paris were hot and humid",
    pos = 4)
```

- Sketch a technical illustration on paper
  - Figure out how to draw it by starting with a blank device and adding lines, rectangles, etc.
- Step through the code that generates the graphs in section 1 of this presentation.
  - Leave SAVEME <- FALSE if you want on-screen graphics.
- If you have R for Windows or Macintosh, lets find the keystrokes to “step next” through one of those examples.

References

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Session

```
sessionInfo()
```

```
R version 3.4.4 (2018-03-15)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04 LTS

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
     LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8
     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                 LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8
    LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  base

loaded via a namespace (and not attached):
[1] compiler_3.4.4 tools_3.4.4
```