COMPUTATIONAL FINANCE & RISK MANAGEMENT
UNIVERSITY of WASHINGTON
Department of Applied Mathematics

Plotting in Base R

CFRM 425 (005)

R Programming for Quantitative Finance

References/Reading/Topics

- Zuur, Ieno, and Masters [ZIM], Ch 5, §§5.1 & 5.2
  - Remark: Ignore the subsections on "Use of a Vector for {pch, col, cex}"

- Additional topics provided by the instructor
  - Remark: Some of the examples may be found in §7.5 in [ZIM]
  - **<u>Need only be concerned about what we cover in class</u>**

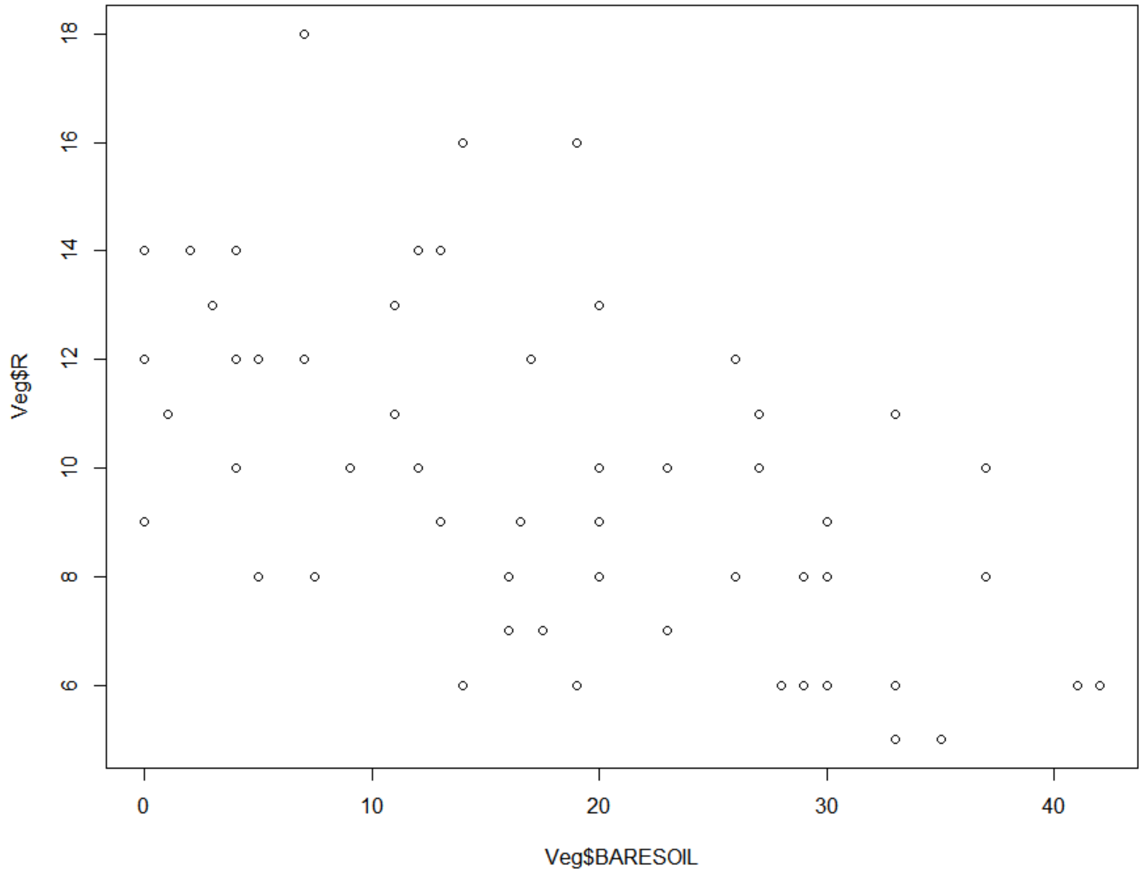
- Topics:
  - Basics of the plot(.) function, additional features, and individual x-y plots
  - Line plots and overlaying plots

Basics of the plot(.) Function
Additional Features
x-y Plots

- Let's use the Vegetation2 data set provided by the ZIM text
- The .csv format is provided on Canvas for you
- Set your working directory as you wish (setwd(.))

```
Veg <- read.csv("Vegetation_005.csv")
plot(Veg$BARESOIL, Veg$R)
```

- Not very impressive…



- The data columns we are using are:
  - BARESOIL: measure of exposed soil
  - R: is a calculated measure of species richness (*not* the R language!)
- Note that the command was a shortcut for

```
plot(x = Veg$BARESOIL, y = Veg$R)
```

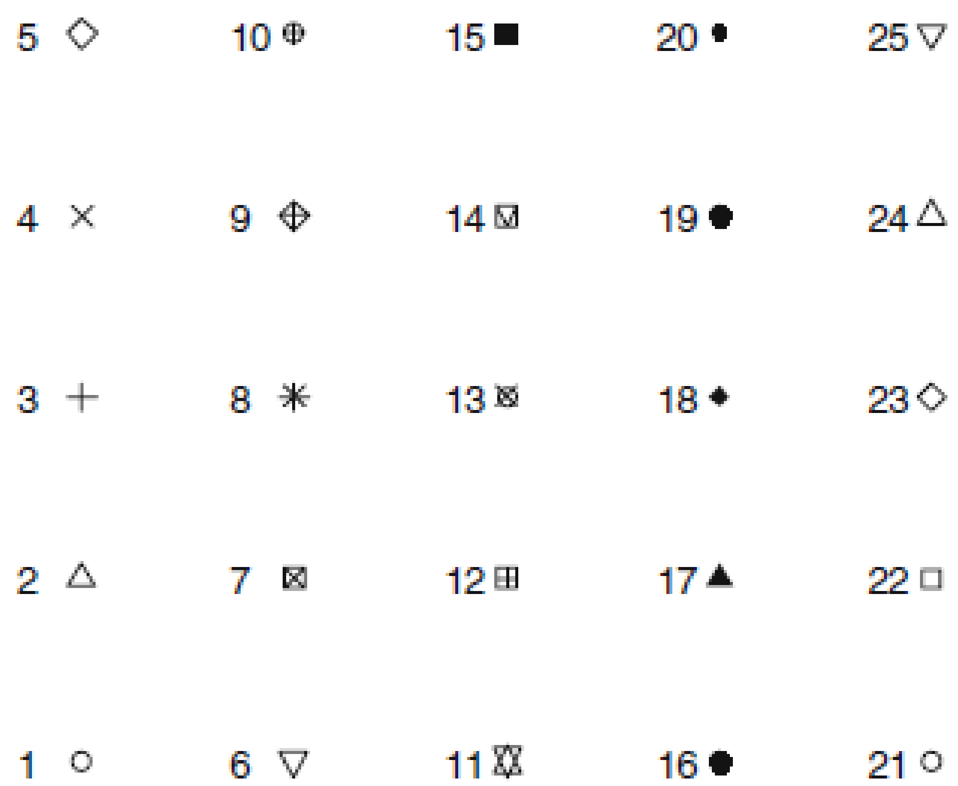
- Actually better to use the full arguments as shown above, especially if your code is for public consumption
- We can, however, remove the dataframe prefix by attaching the data set:

```
attach(Veg)
plot(x = BARESOIL, y = R)
. . .
```

- Need to be sure we *detach* it when done, however, in order to *avoid name clashes*:

```
detach(Veg)
```

- We can spice things up a bit and make our plot more appealing
- For the rest of this section, we'll build on our simple example
- First, we can use filled in circles rather than open to display the data points
  - Use the parameter setting pch = 16 in the plot(.) arguments
  - Default setting is 1 (open circles)
  - Other settings for pch are as follows (Fig 5.3 in the ZIM book):



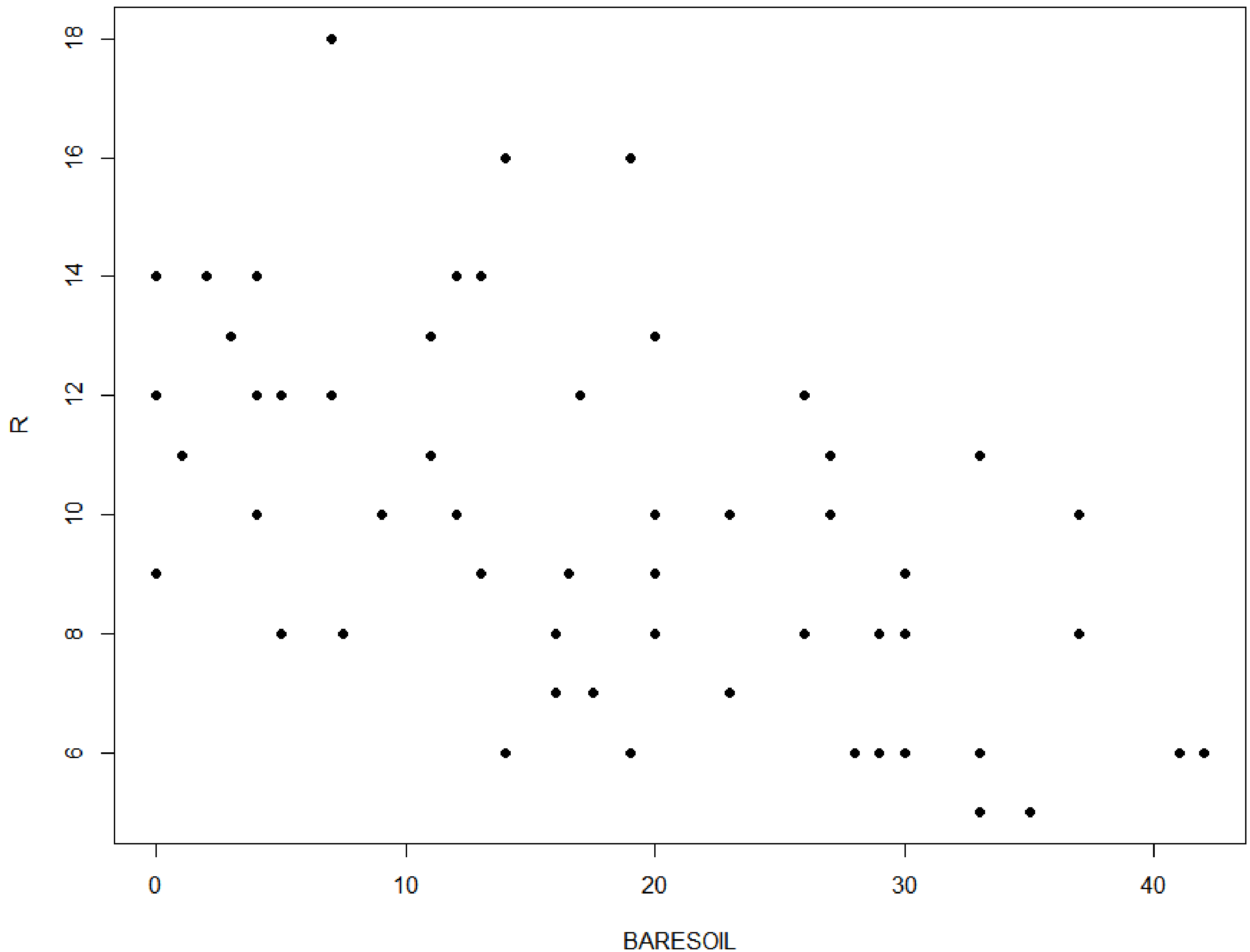
Additional Features

- In addition, it is often a good idea to specify the limits for our x and y axes
  - For plot reuse
  - For overlaying multiple plots (to be discussed in the next section)
  - Two ways we can do this:
    - Hard-coded values: Need to know a priori
    - More general approach: Get the min and max values from the data

```
# Hard-coded:
plot(x = BARESOIL, y = R, pch = 16,
     xlim = c(0, 45), ylim = c(4, 19))


# Generalized, using max/min from data set:
plot(x = BARESOIL, y = R, pch = 16,
     xlim = c(min(BARESOIL), max(BARESOIL)), ylim = c(min(R), max(R)))
```

- Result from previous slide:



Additional Features: Color

- In living **color**
  - The color of the data points is set using the col parameter
  - This may be a numerical index; eg 2 means red
  - It may also be set by a known string (these are mapped to index codes), eg "blue"
- A full list of colors may be found by entering the command **colors()** (**or colours()**) at the console prompt
- Here are the first 20; note that the vector element index is the same as the color index; one could put either **col = 8**, or **col = "aquamarine"**, and the result would be the same color

```
> head(colours(),20)
 [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
 [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
 [9] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"
[13] "azure"         "azure1"        "azure2"        "azure3"
[17] "azure4"        "beige"         "bisque"        "bisque1"
```

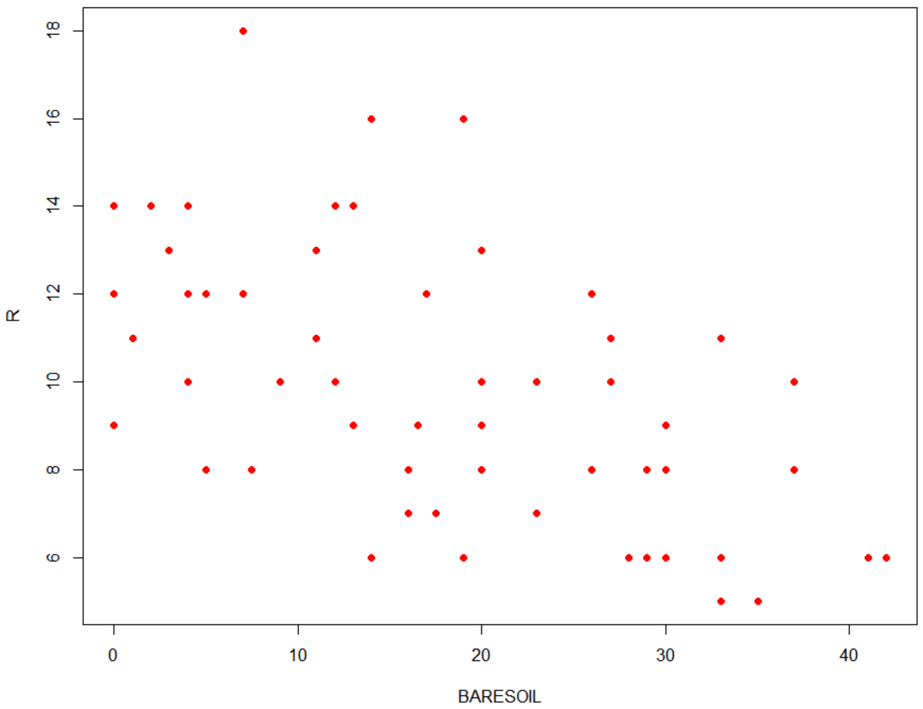
Additional Features: Color

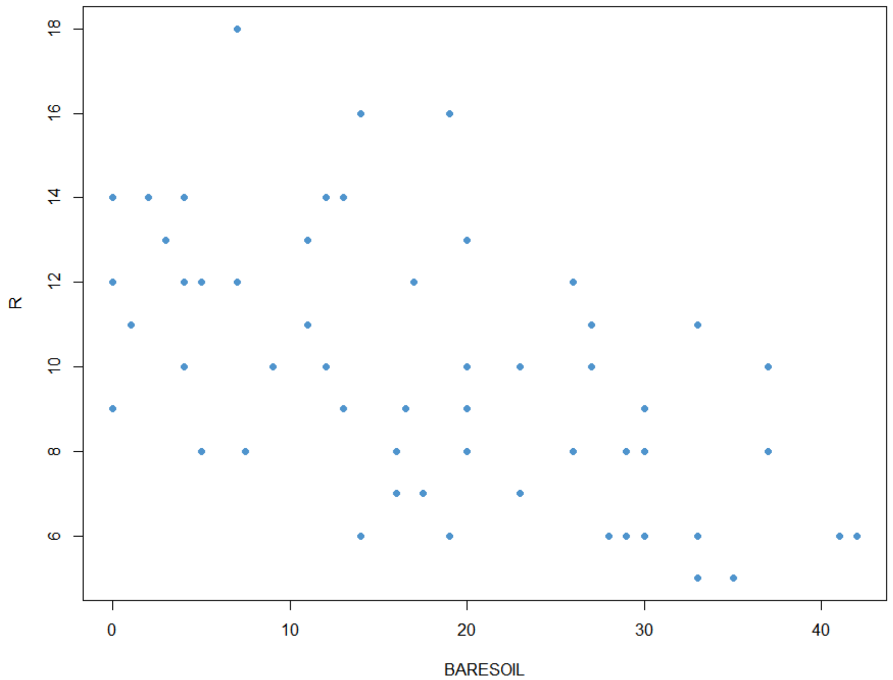
- Two examples (see next slide for results):

```
plot(x = BARESOIL, y = R, pch = 16,
     xlim = c(min(BARESOIL), max(BARESOIL)), ylim = c(min(R), max(R)),
     col = 2)

plot(x = BARESOIL, y = R, pch = 16,
     xlim = c(min(BARESOIL), max(BARESOIL)), ylim = c(min(R), max(R)),
     col = "steelblue3")
```

Additional Features: Color





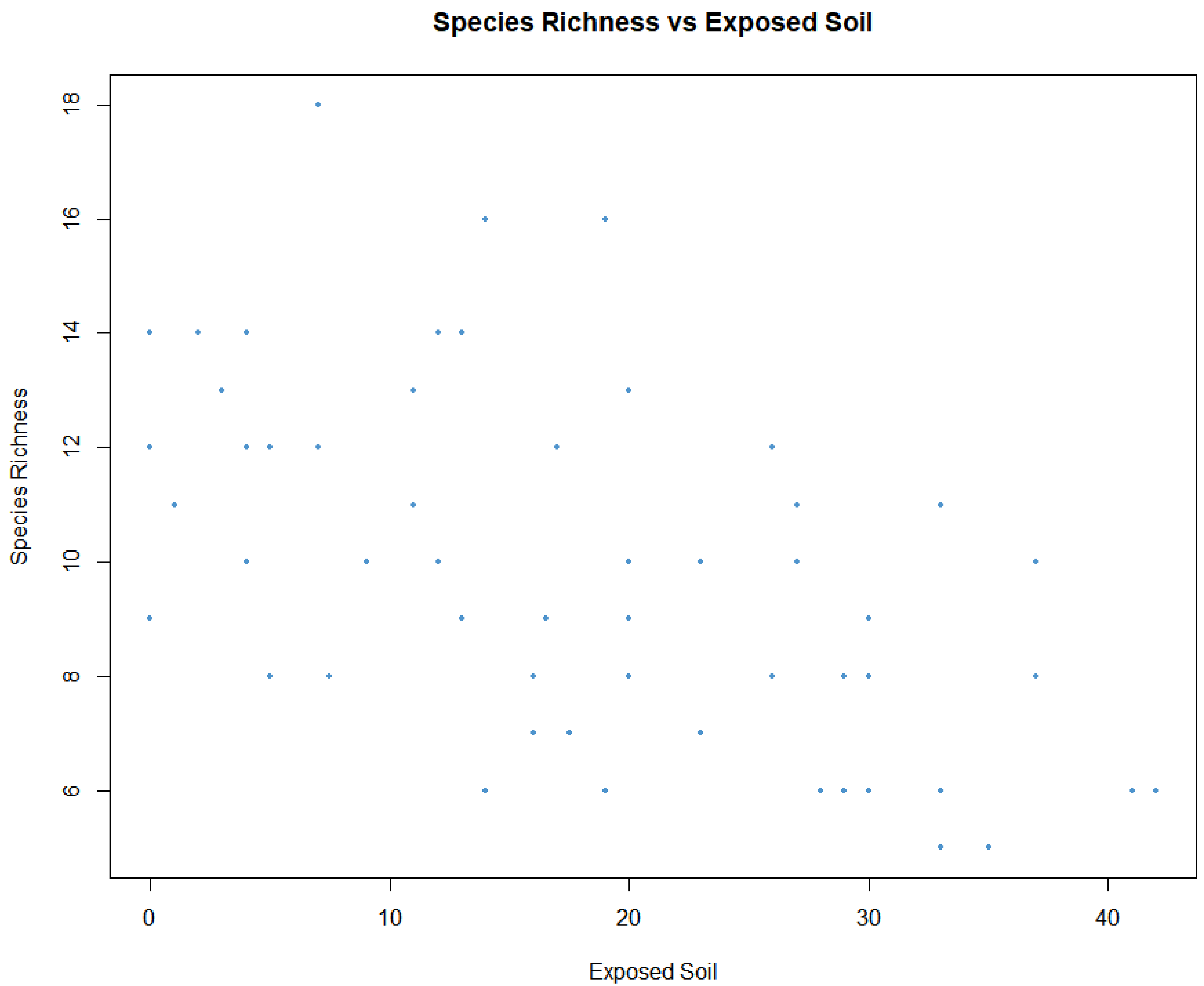
Additional Features: More `plot(.)` Parameters

- Size of data point character: **cex**
  - As a proportion, in decimal format; eg, **cex = 0.5** (50%)
  - Default = 1
- Title of plot: **main**
  - Example: **main = "Species Richness vs Exposed Soil"**
  - Text placed at top of plot image
- Horizontal and vertical axis labels
  - **xlab**, **ylab**
  - Example: **xlab = "Exposed Soil", ylab = "Species Richness"**

- Full example (result on next page):

```
plot(x = BARESOIL, y = R, pch = 16,
     xlim = c(min(BARESOIL), max(BARESOIL)), ylim = c(min(R), max(R)),
     col = 'steelblue3', cex = 0.5, main = "Species Richness vs Exposed Soil",
     xlab = "Exposed Soil", ylab = "Species Richness")
```

- Plot result:


Species Richness vs Exposed Soil
Species Richness
6
8
10
12
14
16
18
0
10
20
30
40
Exposed Soil

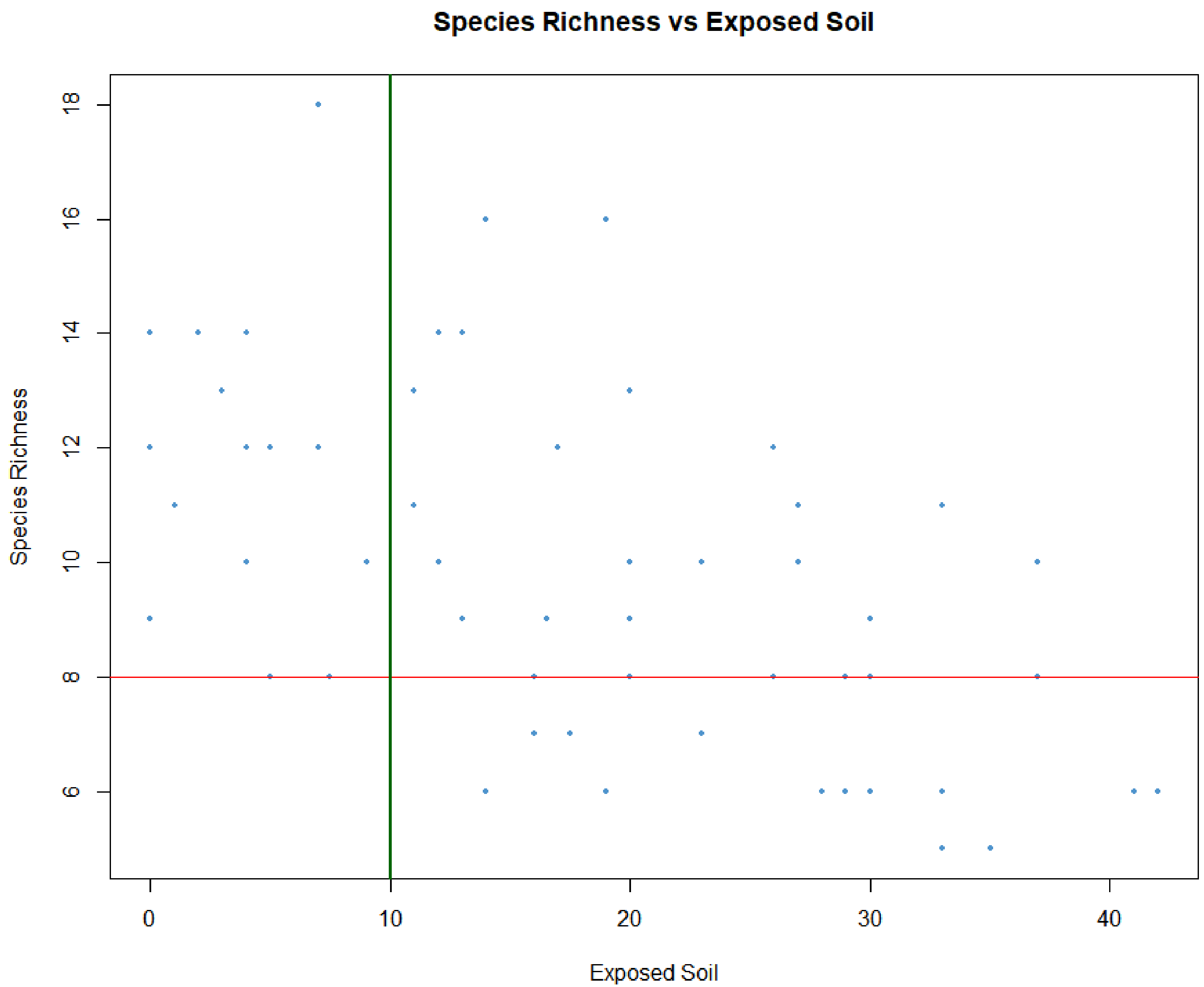
Additional Features: Horizontal and Vertical Lines

- There are times we will want to place a horizontal or vertical line on a plot; eg,
  - Mean of a sample distribution
  - Visual guide for where to place a stop limit trade
- The **abline(.)** function overlays either a horizontal or vertical line, as a follow-on task to a **plot(.)** function call
  - parameter **h = value** on the vertical axis at which to place a horizontal line
  - parameter **v = value** on the horizontal axis at which to place a vertical line
- We can also set
  - Color: **col** parameter
  - Thickness of the line: **lwd** parameter
    - Similar to **cex** parameter
    - Expressed as a proportion (eg 1.5 = 150%)
- Example (result on next slide):

```
plot(x = BARESOIL, y = R, pch = 16,
     xlim = c(min(BARESOIL), max(BARESOIL)), ylim = c(min(R), max(R)),
     col = 'steelblue3', cex = 0.5, main = "Species Richness vs Exposed Soil",
     xlab = "Exposed Soil", ylab = "Species Richness")
abline(h = 8.0, col = 2, lwd = 1.5)              # h => horizontal
abline(v = 10, col = "darkgreen", lwd = 2.0)  # v => vertical
```

Additional Features: Horizontal and Vertical Lines

- Graphical result of example on previous slide:



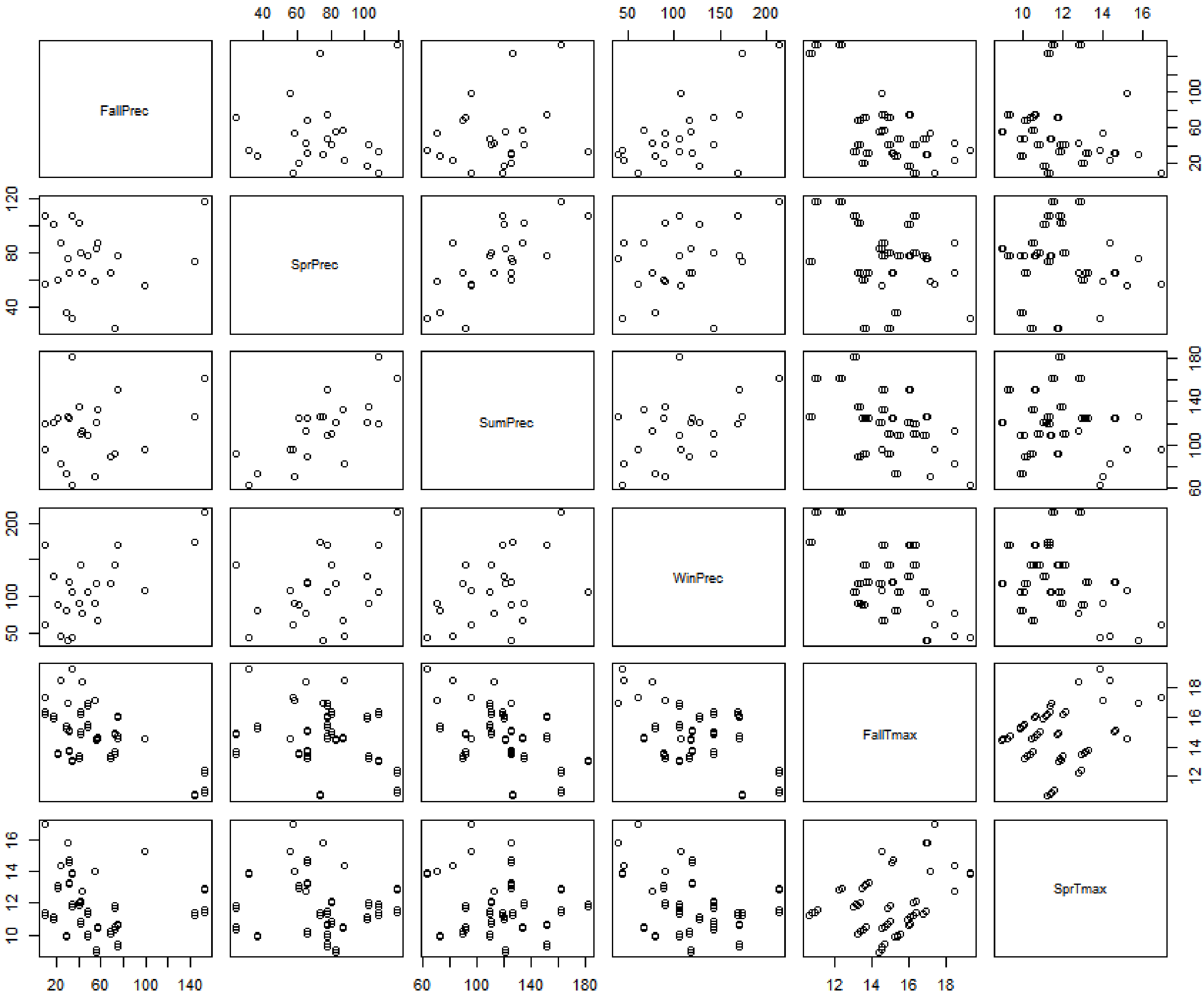
Additional Features: x-y Plots among Pairs of Data

- Suppose we wish to examine pairwise relationships among precipitation and temperature data in the Veg dataframe
- Use the **pairs(.)** function:

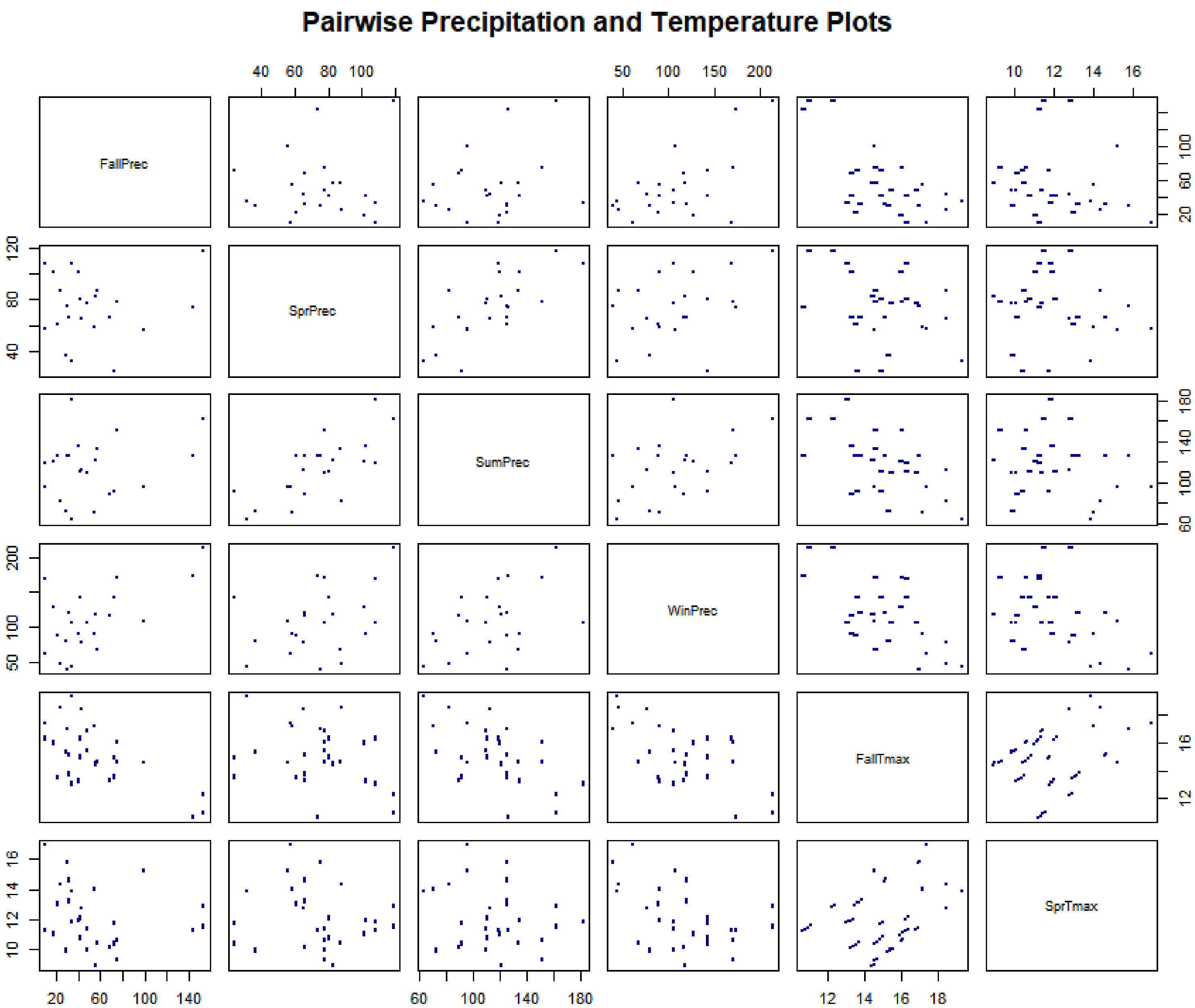
```
pairs(Veg[,c(10:15)])

pairs(Veg[,c(10:15)], pch = 16, cex = 0.25, col = "darkblue",
      main = "Pairwise Precipitation and Temperature Plots")
```

- Results on next two slides


FallPrec
SprPrec
SumPrec
WinPrec
FallTmax
SprTmax
40 60 80 100
50 100 150 200
10 12 14 16
20 60 100
120 80 40
60 100 140 180
200 100 50
12 14 16 18
16 14 12 10
20 60 100 140
60 100 140 180
12 14 16 18

Additional Features: x-y Plots among Pairs of Data (Ex 2)



Line Plots and Overlaying Plots

Line Plots

- What about cases where we are dealing with mathematical functions of a single variable, or an approximation thereof?
- Answer: Use the type parameter setting in **plot(.)**
- More specifically, put **type = "l"** (small "L", for "line plot")
- We can again set the color and thickness with the col and lwd parameters
- Example: Four functions defined on the interval [-10, 10]
  - $f(x) = e^x$
  - $f(x) = \log(x + 15)$
  - $f(x) = \sin x$
  - $f(x) = \sqrt{x^2 + x + 1}$
- Data is found in MathFunctions.csv
- First: Generate a line plot for the first (exponential) function; result on next slide

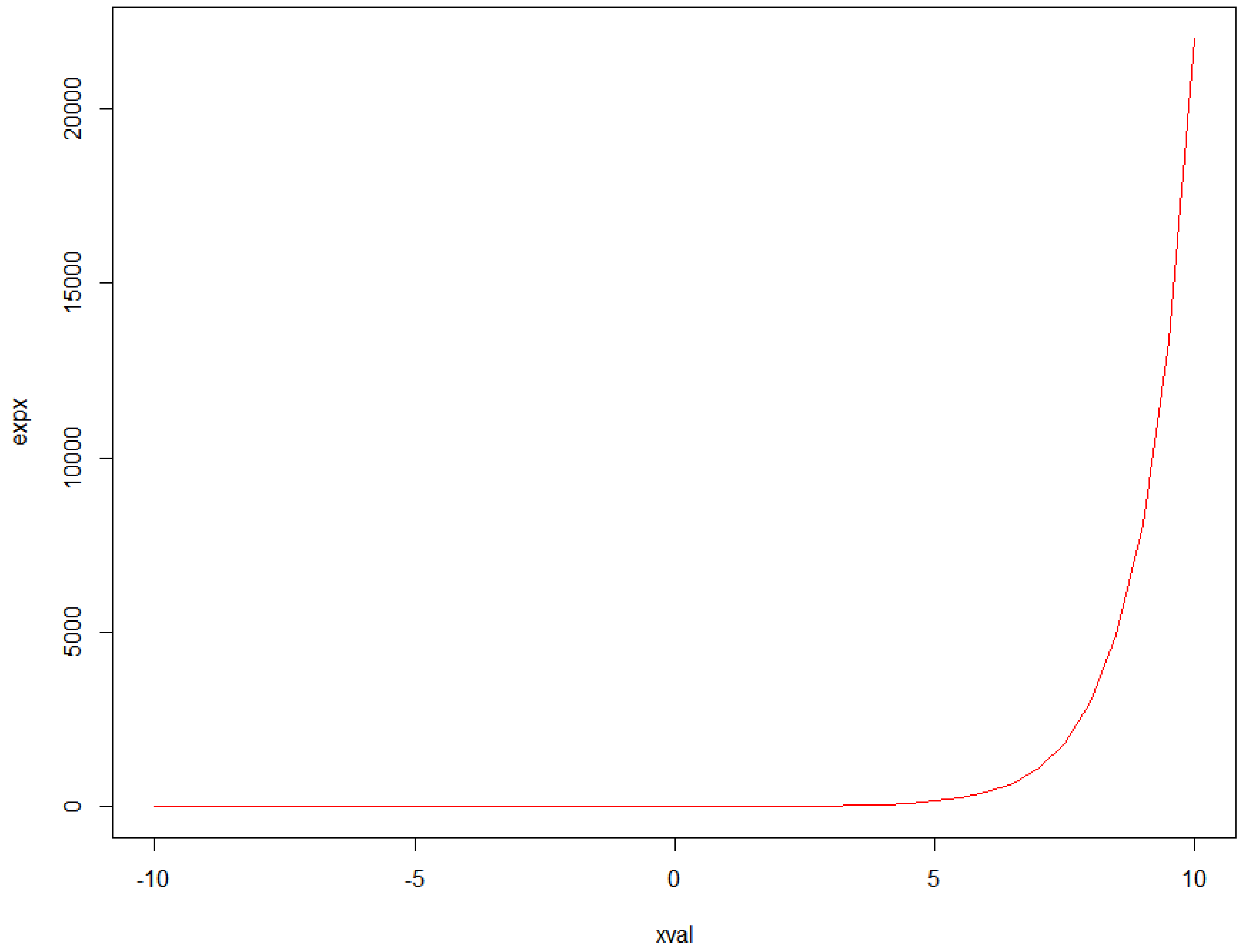
```
Fcns <- read.csv("MathFunctions.csv")

attach(Fcns)

plot(x = xval, y = expx, type = "l", col = 2)
```

Line Plots

- Exponential function example:



Overlaying Line Plots

- Recall our example: Four functions defined on the interval [-10, 10]
  - $f(x) = e^x$
  - $f(x) = \log(x + 15)$
  - $f(x) = \sin x$
  - $f(x) = \sqrt{x^2 + x + 1}$
- Generate a line plot with the last three plots overlayed
  - Before proceeding, we need to allow enough vertical space for each plot
  - Take the minimum of the minima over all three functions
  - Take the maximum of the maxima over all three functions
- In other words:

```
miny = min(min(logx15), min(sinx), min(sqrt_quad))

maxy = max(max(logx15), max(sinx), max(sqrt_quad))
```

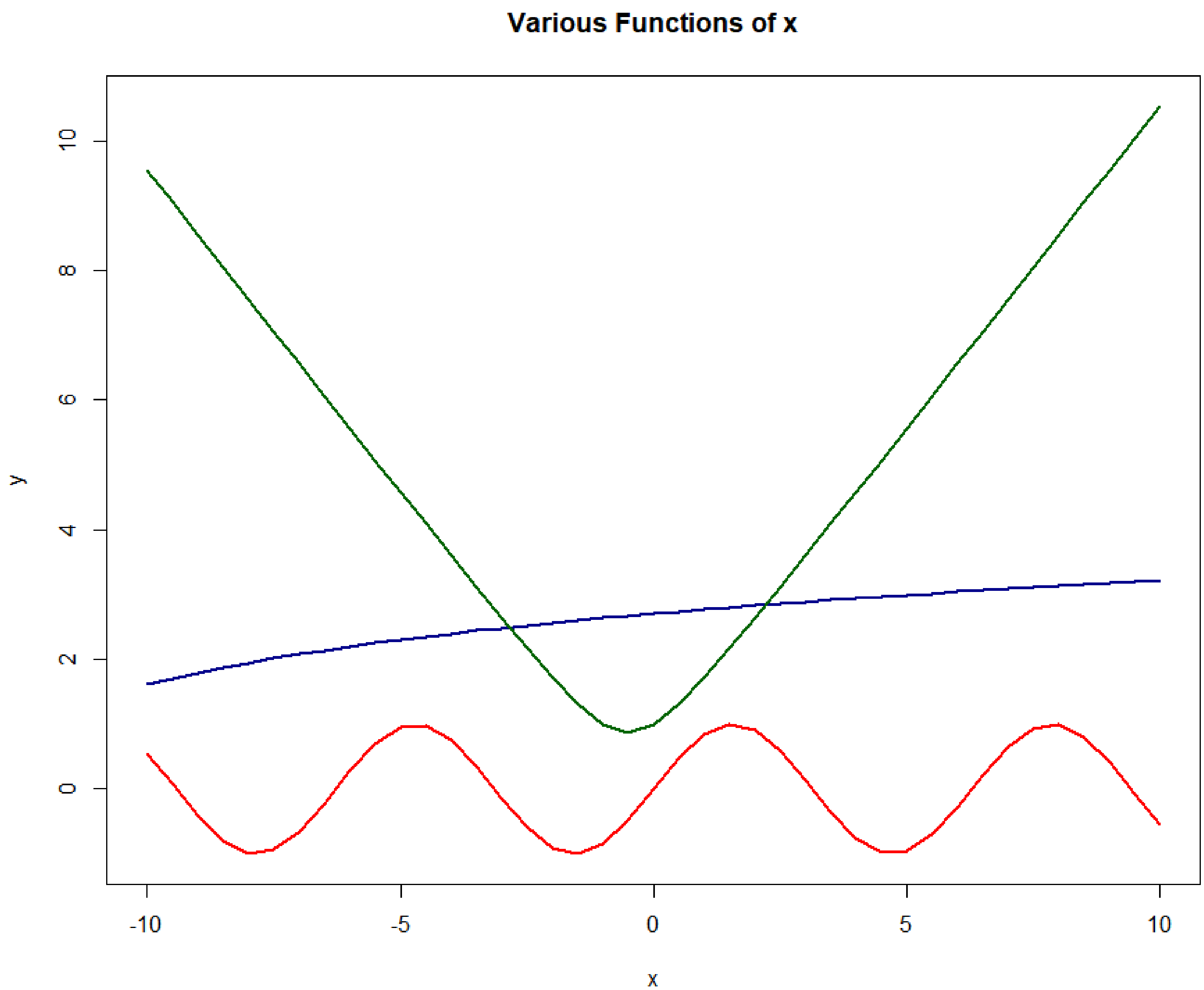
Overlaying Line Plots

- Now, combine the plot of the first function with overlays of the remaining two
    - Plot the log function using **plot(.)**
    - Plot the remaining two with the **lines(.)** function
    - Similar to the **abline(.)** function, **lines(.)** overlays additional curves to the plot:

```
plot(x = xval, y = logx15, type = "l", lwd = 2.5, col = "darkblue",
     main = "Various Functions of x", xlab = "x",
     ylab = "y", ylim = c(miny, maxy))
lines(x = xval, y = sinx, type = "l", lwd = 2.5, col = "red")
lines(x = xval, y = sqrt_quad, type = "l", lwd = 2.5, col = "darkgreen")
```

- Result on next slide

Overlaying Line Plots



Overlaying Line Plots

- One more variation: We can change the line types of each plot
  - Solid line (default)
  - Dashed line
  - Dotted line
  - Others
- See the Help file for **par**, and check the setting **lty**, which stands for “line type”
  - Like the **col** parameter, this can be represented by an integer index, or by text in quotes
  - As an integer
    - 0=blank
    - 1=solid (default)
    - 2=dashed
    - 3=dotted
    - 4=dotdash
    - 5=longdash
    - 6=twodash
  - Or, as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses ‘invisible lines’ (i.e., does not draw them).
- We can also set the thickness of each line using **lwd** (as shown in the previous example, but they need not be the same)

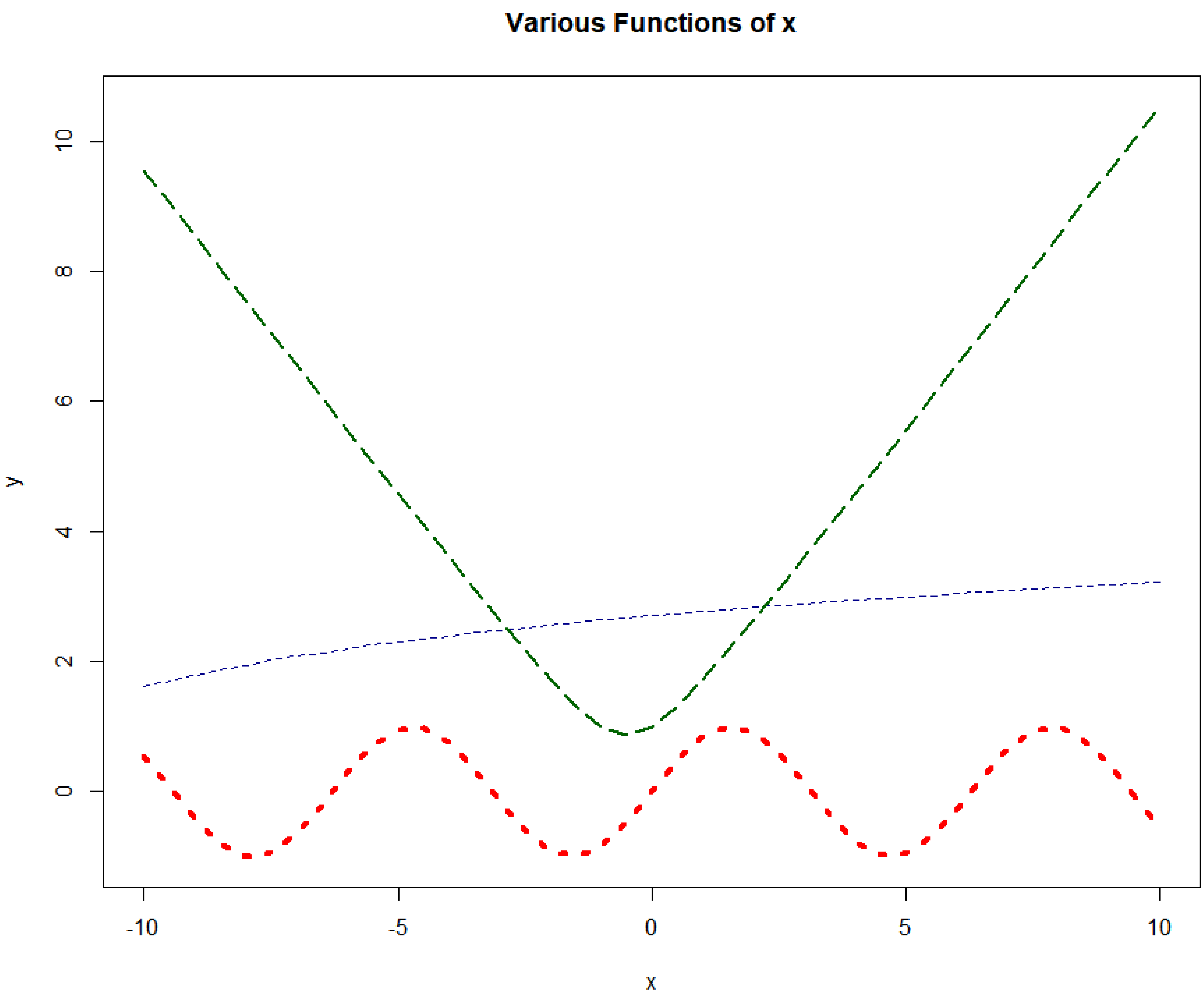
Overlaying Line Plots

- Example:

```
plot(x = xval, y = logx15, type = "l", lwd = 1.5, col = "darkblue",
     main = "Various Functions of x", xlab = "x", ylab = "y",
     ylim = c(miny, maxy), lty = "dashed")
# 3 = "dotted":
lines(x = xval, y = sinx, type = "l", lwd = 4.3, col = "red", lty = 3)
# Use text representation of line type ("longdash") :
lines(x = xval, y = sqrt_quad, type = "l", lwd = 2.7, col = "darkgreen",
      lty = "longdash")
```

- Result on next slide

Overlaying Line Plots



[END]