Lecture 12. Basic Plotting

R and Data Visualization

BIG2006, Hanyang University, Fall 2022

R Graphics

R has incredibly flexible plotting tools for data and model visualization.

Mastering R's graphical functionality does require practice, but the fundamental concepts are straightforward.

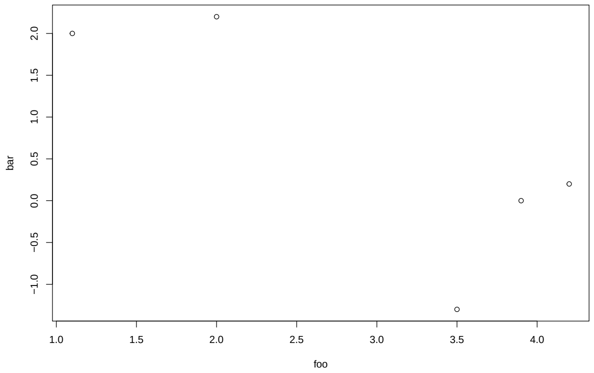
In this lecture, we will explore an overview of the `plot` function and some useful options for controlling the appearance of the final graph.

Using Plot with Coordinate Vectors

- Treat your screen as a blank, two-dimensional canvas
- Plot points and lines using x-and y-coordinates with points written as pair: (x value, y value).
- `plot`: takes in two vectors and opens a graphic device where it displays the result.

Note: The `plot` function is a one of R's versatile generic functions. It works differently for different objects and allows users to define their own methods for handling objects.

```r
foo <- c(1.1, 2, 3.5, 3.9, 4.2)
bar <- c(2, 2.2, -1.3, 0, 0.2)
plot(foo, bar)
```



Graphical Parameters

There are a wide range of graphical parameters that can be supplied as argument to the `plot` function.

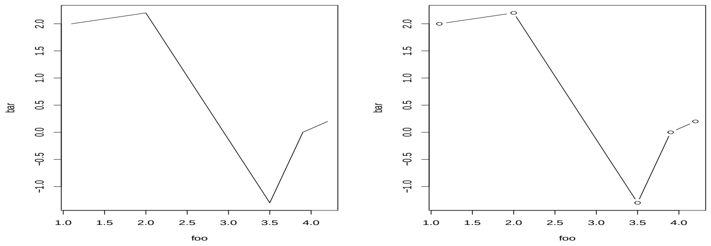
- `type`: tells R how to plot the supplied coordinates, e.g., points or joined by lines or both dots and lines
- `main, xlab, ylab`: options to include plot title, the horizontal axis label, and the vertical axis label
- `col`: colors to use for plotting points and lines

- `pch` (point character): selects which character to use for plotting individual points
- `cex` (character expansion): controls the size of plotted point characters
- `lty` (line type): specifies the type of line to use to connect the points, e.g., solid, dotted, or dashed
- `lwd` (line width): controls the thickness of plotted line
- `xlim, ylim`: provides limits for the horizontal range and vertical range of the plotting region

Automatic Plot Types (`type`)

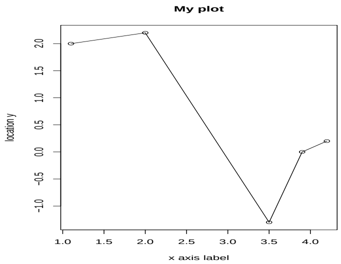
- `type="p"` (points only), `"l"` (lines only), `"b"` (both points and lines), `"o"` (overplotting the points with lines), `"n"` (no points or lines plotted)

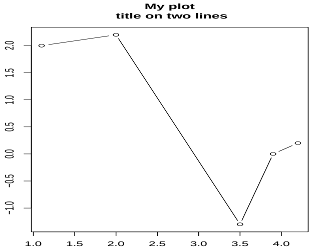
```r
par(mfrow=c(1,2))
plot(foo,bar,type="l");plot(foo,bar,type="b")
```



Title and Axis Labels (`main`, `xlab`, `ylab`)

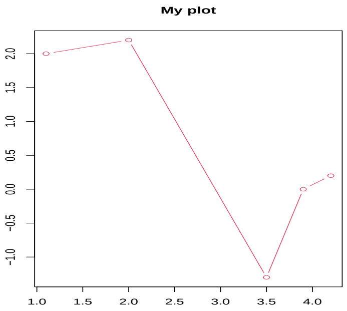
```
par(mfrow=c(1,2))
plot(foo,bar,type="o",main="My plot",
     xlab="x axis label", ylab="location y")
plot(foo,bar,type="b",main="My plot\n title on two lines",
     xlab="", ylab="")
```

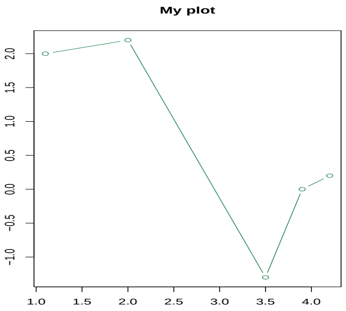




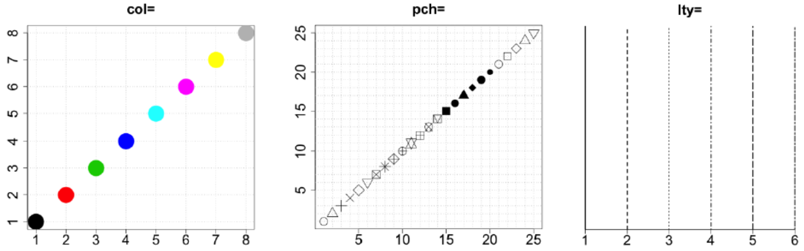
Color (col; can make data much clearer)

```
par(mfrow=c(1,2))
plot(foo,bar,type="b",main="My plot",xlab="",ylab="",col=2)
plot(foo,bar,type="b",main="My plot",xlab="",ylab="",col="seagreen4")
```



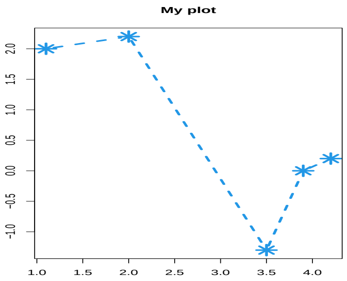


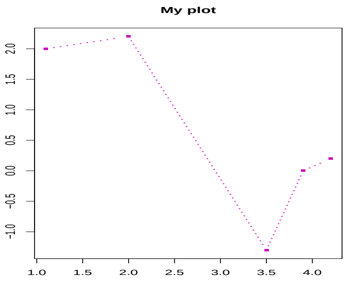
Note: You can set colors with the `col` parameter in a number of ways. There are eight possible integer values. Also, you can specify colors using RGB (red, green, and blue) levels and by creating your own palettes.



Line and Point Appearances (pch, lty, cex, lwd)

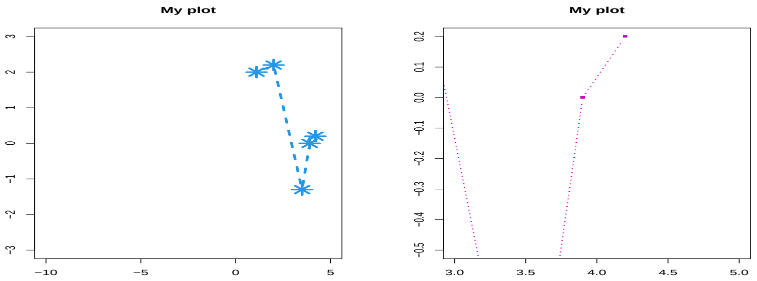
```
par(mfrow=c(1,2))
plot(foo,bar,type="b",main="My plot",xlab="",
     ylab="",col=4,pch=8,lty=2,cex=2.3,lwd=3.3)
plot(foo,bar,type="b",main="My plot",xlab="",
     ylab="",col=6,pch=15,lty=3,cex=0.7,lwd=2)
```





Plotting Region Limits (`xlim`, `ylim`)

```
par(mfrow=c(1,2))
plot(foo,bar,type="b",main="My plot",xlab="",ylab="",
  col=4,pch=8,lty=2,cex=2.3,lwd=3.3,xlim=c(-10,5),ylim=c(-3,3))
plot(foo,bar,type="b",main="My plot",xlab="",ylab="",
  col=6,pch=15,lty=3,cex=0.7,lwd=2,xlim=c(3,5),ylim=c(-0.5,0.2))
```



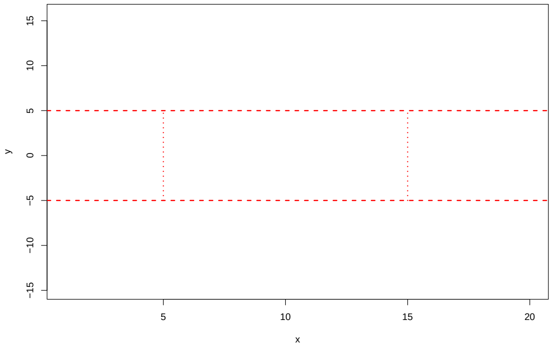
Adding Points, Lines, and Text to an Existing Plot

Each call to `plot` will refresh the active graphics device for a new plotting region.

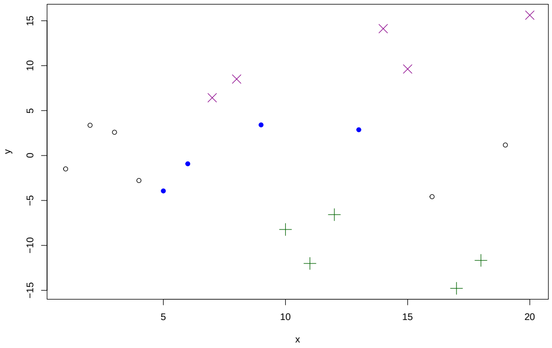
Here are some useful, ready-to-use functions that will add a plot without refreshing or clearing the window:

- `points` (add points)
- `lines`, `abline`, `segments` (add lines)
- `text` (writes text), `arrows` (add arrows)
- `legend` (adds a legend)

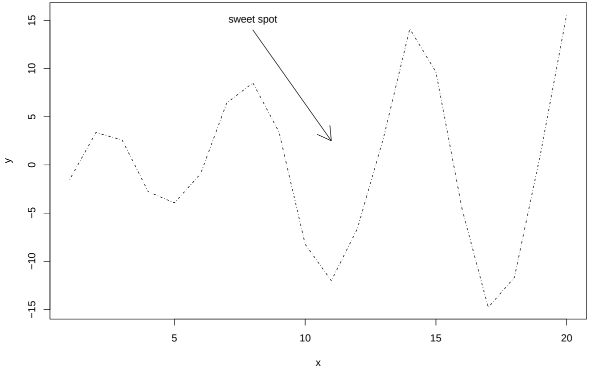
```r
x <- 1:20
y <- c(-1.49,3.37,2.59,-2.78,-3.94,-0.92,6.43,8.51,3.41,-8.23,
  -12.01,-6.58,2.87,14.12,9.63,-4.58,-14.78,-11.67,1.17,15.62)
plot(x,y,type="n",main="")
abline(h=c(-5,5),col="red",lty=2,lwd=2)
segments(x0=c(5,15),y0=c(-5,-5),x1=c(5,15),y1=c(5,5),
         col="red",lty=3,lwd=2)
```



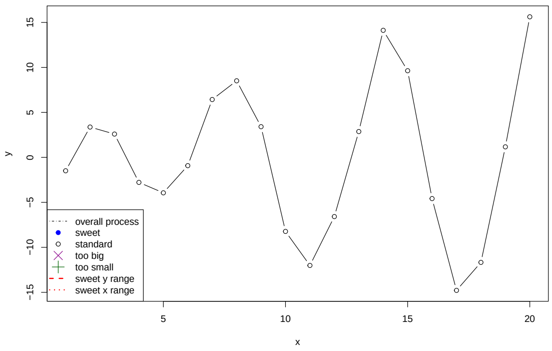
```
plot(x,y,type="n",main="")
points(x[y>=5],y[y>=5],pch=4,col="darkmagenta",cex=2)
points(x[y<=-5],y[y<=-5],pch=3,col="darkgreen",cex=2)
points(x[(x>=5&x<=15)&(y>-5&y<5)],y[(x>=5&x<=15)&(y>-5&y<5)],
       pch=19,col="blue")
points(x[(x<5|x>15)&(y>-5&y<5)],y[(x<5|x>15)&(y>-5&y<5)])
```

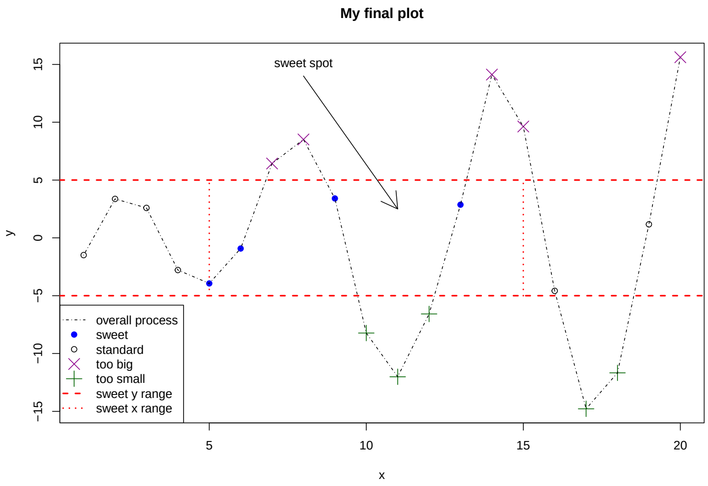


```
plot(x,y,type="n",main="")
lines(x,y,lty=4)
arrows(x0=8,y0=14,x1=11,y1=2.5)
text(x=8,y=15,labels="sweet spot")
```



```
plot(x,y,type="b",main="")
legend("bottomleft",
  legend=c("overall process","sweet","standard",
  "too big","too small","sweet y range","sweet x range"),
  pch=c(NA,19,1,4,3,NA,NA),lty=c(4,NA,NA,NA,NA,2,3),
  col=c("black","blue","black","darkmagenta","darkgreen","red","red"),
  lwd=c(1,NA,NA,NA,NA,2,2),pt.cex=c(NA,1,1,2,2,NA,NA))
```





My final plot
sweet spot
overall process
sweet
standard
too big
too small
sweet y range
sweet x range
x
y
5
10
15
20
−15
−10
−5
0
5
10
15

The `ggplot2` Package

Another important suite of graphical tools: `ggplot2`, a prominent contributed package by Hadley Wickham (2009)

`ggplot2` offers particularly powerful alternatives to the standard plotting procedures in R.

gg stands for *grammar of graphics*.

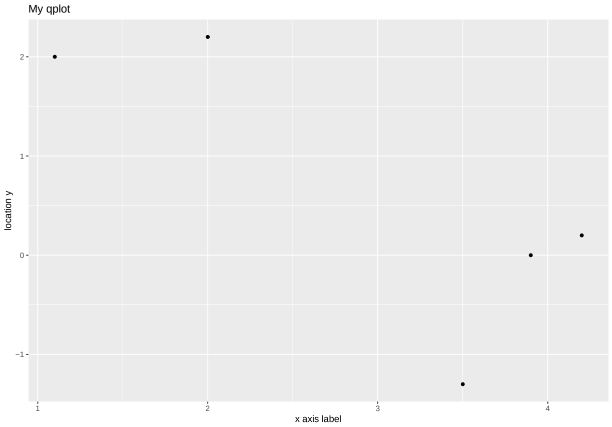
`ggplot2` standardizes the production of different plot and graph types and let you build plots by defining and manipulating *layers*.

A Quick Plot with `qplot`

- You must install `ggplot2` package.
- `ggplot2` plots are stored as objects, meaning that they have an underlying, static representation until you change the object.

Note: The object-oriented nature of `ggplot2` graphics means tweaking a plot or experimenting with different visual features no longer requires you to return every plotting command each time you change something.

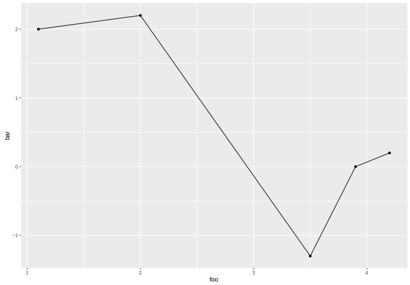
```r
foo <- c(1.1, 2, 3.5, 3.9, 4.2)
bar <- c(2, 2.2, -1.3, 0, 0.2)
qplot(foo, bar, main="My qplot",
      xlab="x axis label", ylab="location y")
```



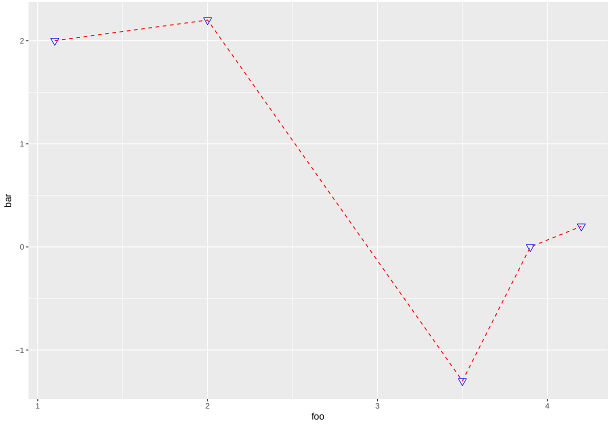
Setting Appearance Constants with Geometric Modifiers

- To customize ggplot2 graphic, you alter the object itself.
- First store the qplot object you created earlier and then use geom_point and geom_line with different style.

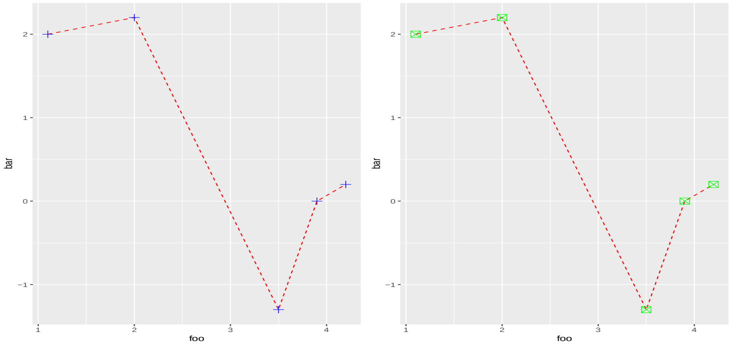
```
qplot(foo,bar,geom="blank") + geom_point() + geom_line()
```



```r
qplot(foo,bar,geom="blank") +
  geom_point(size=3,shape=6,color="blue") +
  geom_line(color="red",linetype=2)
```



```
myqplot <- qplot(foo,bar,geom="blank") +
  geom_line(color="red",linetype=2)
a <- myqplot + geom_point(size=3,shape=3,color="blue")
b <- myqplot + geom_point(size=4,shape=7,color="green")
grid.arrange(a,b, nrow=1, ncol=2) # gridExtra package
```



Aesthetic Mapping with Geoms

- Geoms and `ggplot2` provide efficient, automated ways to apply different styles to different subsets of a plot.
- `ggplot2` apply particular styles to different categories using a factor object.
- The factor that holds these categories is called a *variable*, which `ggplot2` can *map* to *aesthetic* values.

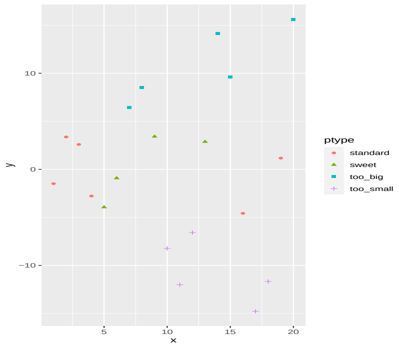
Note: The above gets rid of much of the effort that goes into isolating subsets of data and plotting them separately using base R graphics.

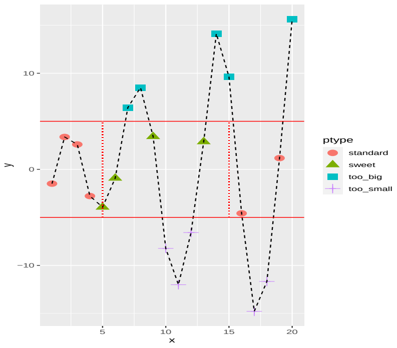
- Based on x and y values, define several categories.

```r
x <- 1:20
y <- c(-1.49,3.37,2.59,-2.78,-3.94,-0.92,6.43,8.51,3.41,-8.23,
  -12.01,-6.58,2.87,14.12,9.63,-4.58,-14.78,-11.67,1.17,15.62)
ptype <- rep(NA,length(x=x))
ptype[y>=5] <- "too_big"
ptype[y<=-5] <- "too_small"
ptype[(x>=5&x<=15)&(y>-5&y<5)] <- "sweet"
ptype[(x<5|x>15)&(y>-5&y<5)] <- "standard"
ptype <- factor(x=ptype)
ptype
```

```
##  [1] standard  standard  standard  standard  sweet     sweet     too
##  [8] too_big   sweet     too_small too_small too_small sweet     too
## [15] too_big   standard  too_small too_small standard  too_big
## Levels: standard sweet too_big too_small
```

```r
a <- qplot(x,y,color=ptype,shape=ptype)
b <- qplot(x,y,color=ptype,shape=ptype) + geom_point(size=4) +
 geom_line(mapping=aes(group=1),color="black",lty=2) +
 geom_hline(mapping=aes(yintercept=c(-5,5)),color="red") +
 geom_segment(mapping=aes(x=5,y=-5,xend=5,yend=5),color="red",lty=3) +
 geom_segment(mapping=aes(x=15,y=-5,xend=15,yend=5),color="red",lty=3)
grid.arrange(a,b, nrow=1, ncol=2) # gridExtra package
```





Reference

- Davies, T. M. The Book of R. No Starch Press. Chapter 7.