12 Graphical procedures

R 의 그래픽 기능은 R 구성 환경 중에서도 굉장히 중요하고 많은 역할을 담당하고 있습니다. 이는 이러한 기능들을 통해 여러 종류의 통계적인 그래프를 그릴 수 있을 뿐 아니라 완전히 새로운 종류의 그래프를 만들어내는 것도 가능합니다.

그래픽 기능들은 interactive 나 batch 모드에서 모두 사용될 수 있지만, 대부분의 경우, interactive 모드에서 사용하는 것이 더욱 효과적입니다. 특히, R 에서는 프로그램이 시작될 때 interactive 그래픽을 보여주는 특정 graphic window 를 열어주는 device driver 가 초기화되므로 이를 이용하기가 더 쉽습니다. 이러한 초기화는 자동적으로 이루어지지만, Unix 환경에서는 X11(), Windows 환경에서는 windows() 그리고 Mac OS X 에서는 quartz() 가 device driver 를 초기화시키기 위해 사용될 수 있음을 알아두는 편이 유용할 것 입니다.

역주: interactive mode – 사용자가 새로운 명령을 입력할 때마다 거기에 대응되는 graphic 상의 변화를 확인할 수 있는 방법

batch mode – 일련의 그래픽 함수들이 별도의 지정 없이도 자동 실행되어 graphic 을 실행하는 방법

일단 device driver 가 작동되기 시작하면, R plot 명령문들은 다양한 종류의 그래픽을 표현하거나 또는 완전히 새로운 형태의 표현을 위해 사용 될 수 있습니다.

Plot 를 그리는 명령문은 다음과 같이 세가지로 구분됩니다:

- **High-level** 함수는 정해진 그래픽 device 내 에서 축, 레이블, 제목 등을 변경해서 새로운 plot 을 생성 합니다.

- **Low-level** 함수는 이미 존재하는 plot 위해 추가로 점, 선, 레이블 등을 첨가하여 좀 더 많은 정보를 표현할 수 있도록 합니다.

- **Interactive** 함수들은 이미 존재하는 plot 에서 마우스와 같은 pointing device 를 이용하여 interactive 하게 정보를 추가하거나 제거하는 기능을 합니다.

또, R 에서는 일련의 graphical parameter 들을 조작하여 당신이 원하는 대로 plot 을 쉽게 변경할 수 있는 기능을 제공합니다.

이 매뉴얼에서는 “base” 그래픽에 해당하는 내용만을 대상으로 합니다. base 와는 별개로 팩키지 grid 안에 별개의 그래픽 sub 시스템이 따로 존재하는데, 이것은 좀 더 유용하긴 하지만 사용하기가 더 어렵습니다. Grid 를 바탕으로 만들어진 팩키지 lattice 도 추천할 만한데, 이 팩키지는 S 의 *Trellis* 시스템에서 제공하는 것과 비슷한 multi-panel 플롯을 그릴 수 있는 기능들을 제공합니다.

12.1 High-level plotting commands

High-level 플롯 함수들은 데이터를 인수(argument) 형태로 만들어 이 함수를 이용하여 복잡한 형태의 plot 을 그리기 위해 고안되었습니다. (특별히 따로 지정하지 않으면) 적절한 위치에 축, 레이블 그리고 제목 등이 자동적으로 생성됩니다. High-level 플롯 명령문은 항상 새로운 플롯을 시작하기 때문에 필요한 경우에는 현재 플롯을 자동적으로 지워버리기도 하빈다.

12.1.1 The plot() function

R 의 plot 함수들 중에서는 plot()이 가장 많이 사용되는 것 중의 하나일 것입니다. 이것은 일종의 일반(generic) 함수로, plot 의 형태는 입력된 첫번째 인수(argument)의 종류(type)나 class 에 의해 결정됩니다.

plot(*x*, *y*)
plot(*xy*)
x 와 y 가 둘 다 벡터이면, 이 문장은 x 에 대한 y 의 산점도(scatterplot)을 그린다. 두번째 문장 역시 이것과 같이 작동하지만, xy 는 x 와 y 를 둘 다 포함하는 하나의 열(list)이나 2 열 행렬형태의 인수여야 한다.
plot(*x*)
x 가 하나의 시계열인 경우, 이 문장으로 시계열 플롯을 그린다. x 가 하나의 숫자 벡터인 경우, 이 문장으로 각 값들의 인덱스에 대한 해당 벡터값의 플롯을 그린다. x 가 복소수(complex) 벡터인 경우라면, 이 문장으로 각 벡터의 실수 부분에 대한 허수(imaginary) 부분의 플롯이 나온다.
plot(*f*)
plot(*f*, *y*)

f 가 하나의 요인(factor)이고, y 는 하나의 숫자 벡터여야 한다. 첫번째 문장은 f 에 대한 막대 그래프를, 두번째 문장은 각 f 의 수준에서의 y 의 boxplot 들을 생성한다.

plot(*df*)

plot(~ *expr*)

plot(*y* ~ *expr*)

위 문장을 사용하기 위해서는 df 는 하나의 데이터 프레임이어야 하고, y 는 어떤 형식이든 상관 없지만, *expr* 는 여러 개의 객체(object)들이 "+"에 의해 연결된 형태로 표현되어야 한다. (e.g., a + b + c) 처음 두 문장은 일종의 변수들에 대한 분포 플롯을 그린다. (역주 – 각 변수에 대한 다른 변수의 분포를 보여주는 플롯, 일종의 pairwise scatterplots 이 출력됨) 이때 plot 함수는 데이터 프레임 안에 포함된 모든 변수들을 대상으로 하거나 (첫번째 형태), expr 안에서 사용된 몇 개의 변수들만을 대상으로 한다(두번째 형태). 세번째 문장의 경우, expr 에서 사용된 모든 변수들에 대한 y 의 값의 플롯들을 각 변수 별로 출력한다.

12.1.2 Displaying multivariate data

R 은 다변량(multivariate) 데이터를 표현할 수 있는 매우 유용한 두 가지 함수를 제공합니다. X 가 숫자 행렬이거나 데이터 프레임인 경우, 그 명령문은

```
> pairs(X)
```

이 문장은 X 에 포함되어 있는 열들에 해당하는 변수들을 대상으로 한 행렬의 pairwise 산점도(scatterplot)을 출력합니다. 이것은 X 의 한 열(변수)에 대해 X 의 다른 모든 열들에 대한 플롯을 그리므로 그 결과 총 n*(n-1)개의 플롯이 각 열이나 각 행별로는 일정한 plot scale(단위, 축)을 가진 행렬의 형태로 출력됩니다.

세 네개의 변수들로 작업하는 경우, coplot 이 좀 더 효과적인 방법이 될 수 있을 것입니다. a 와 b 가 숫자 벡터이고 동시에 c 가 숫자 벡터이거나 요인(factor)인 경우 (물론 셋 다 같은 길이를 가져야 합니다.), 다음과 같은 명령문이 사용됩니다.

```
> coplot(a ~ b | c)
```

이 문장은 c 의 각 값 별로 b 에 대한 a 의 산점도들을 그린다. 만약 c 가 하나의 요인이라면, 이것은 단순히 c 의 수준에 대해 b 에 대해 a 가 그려진 것을 의미한다. c 가 숫자 변수이면, c 의 값들은 몇 개의 조건부 구간(conditioning interval)로 나눠지고 c 의 구간 별로 대응되는 b 에 대한 a 의 각 구간을 그린다. 구간의 수와 위치는 coplot() 에서 "given.values="선언에 의해 조절할 수 있으며 함수 co.intervals()로 구간 선택을 조

절할 수 있다. 또한 두 개의 변수를 조건부로 하여 다음과 같은 명령문을 만드는 것도 가능하다.

```
> coplot(a ~ b | c + d)
```

c 와 d 에 대한 모든 joint conditioning interval 에서 b 에 대한 a 의 산점도를 그린다.

coplot()과 pairs()함수는 모두 “panel=”선언을 사용할 수 있는데, 이는 각 패널을 표현하는 플롯의 종류를 지정하는데 사용됩니다. 디폴트로는 산점도를 그리는 points()가 지정되어 있는데 벡터 x 와 y 를 사용하는 다른 종류의 low-level 그래픽 함수를 “panel=”에 지정하면 어떤 종류의 플롯이라도 표현 가능합니다. 예를 들면 coplot 에서 사용할 수 있는 panel 함수로는 panel.smooth()가 있습니다.

12.1.3 Display graphics

다른 high-level 그래픽 함수들로 다른 여러 종류의 플롯들을 그릴 수 있습니다. 몇 가지 예로 다음과 같은 것들이 있습니다:

```
qqnorm(x)
qqline(x)
qqplot(x, y)
```

분포-비교에 사용되는 플롯들이다. 처음 명령문은 expected Normal order score 에 대한 숫자 벡터인 x 를 그린다 (normal score 플롯이라고도 한다.) 그리고 두번째 명령문에서 분포와 data 에 대한 quartile 들을 통과하는 직선을 그린다. 세번째 문장에서는 두 변수의 각각의 분포를 비교하여 y 의 quantile 에 대한 x 의 quantile 을 그린다.

```
hist(x)
hist(x, nclass=n)
hist(x, breaks=b, ...)
```

숫자 벡터인 x 에 대한 히스토그램을 그린다. 범주(class)의 숫자는 대체로 적절하게 선택되지만, “nclass=”선언을 통해 범주의 숫자를 선택할 수도 있다. 또 다른 방법으로는, “breaks=”라는 선언을 통해서 정확하게 breakpoint 를 결정할 수도 있다. probability=TRUE 선언이 포함된 경우에는, 막대들은 전체 개수에 대한 상대 도수가 아니라 막대 넓이에 대한 상대 도수를 의미하게 된다.

```
dotchart(x, ...)
```

x 에 포함된 데이터의 dotchart 를 그린다. dotchart 에는 y-축은 x 에 포함된 데이터에 의해 이름이 붙고, x-축은 x 의 값을 그대로 사용한다. 예를 들어, 이 방법으로는 정해진 특정 범위 내에만 포함된 데이터 값을 이용해서 모든 데이터 값들을 쉽게 표현할 수 있다.

image(x, y, z, ...)
contour(x, y, z, ...)
persp(x, y, z, ...)
세개 변수에 대한 플롯을 그린다. 첫번째 명령문에서는 서로 다른 z 의 값을 나타내는 여러 가지 색으로 x 와 y 에 값들을 표현한 직사각형의 틀(grid)를 그린다. 두번째 명령문은 같은 z 값을 연결한 등고선(contour line)을 이용한 contour plot 을 그리며, 세번째 명령문은 3D 표면으로 나타내는 조감도(persp plot)를 그린다.

12.1.4 Arguments to high-level plotting functions

high-level 그래픽 함수에서 사용할 수 있는 선언들은 다음과 같은 것들이 있습니다:

add=TRUE
해당 함수가 low-level 그래픽 함수처럼 사용되도록 해서, 현재의 플롯 위에 해당 플롯이 덧 그려지도록 한다. (제한된 함수만 사용 가능)

axes=FALSE
축의 생성을 제한한다-axis() 함수를 사용해서 사용자 정의로 축을 표현하고 싶을 때 사용할 수 있다. 디폴트로 axes=TRUE 되어 있고, 축을 포함할 것을 의미한다.

log="x"
log="y"
log="xy"
x, y 또는 두 축 모두를 로그화 한다. 이 함수는 많은 종류의 플롯에서 작동하지만, 작동되지 않는 경우도 있다.

type=
플롯의 종류를 지정한다:
type="p"
각각 포인트를 점으로 표현한다. (디폴트)

type="l"

선으로 표현한다.

type="b"

점으로 표현하고 선으로 연결한다. (두가지 모두)

type="o"

선 위로 포인트들을 겹쳐지게 그린다.

type="h"

0 라인부터 각 점까지의 거리를 세로선으로 그린다. (high-density)

type="s"

type="S"

Step-function 을 그린다. 첫번째 형태에서는 함수의 끊어진 부분의 윗부분이 점으로 나타나고, 두번째 형태에서는 아랫 부분이 점으로 나타난다.

type="n"

플롯을 그리지 않는다. 그렇지만 (디폴트에 의해) 축은 여전히 나타나며 데이터에 대한 좌표(coordinate system)는 세워져 있는 상태이다. Subsequnce 에 대해 low-level 그래픽 함수를 적용한 플롯을 만들어 내기위해 사용할 수 있다.

xlab=*string*

ylab=*string*

x 와 y 축에 대한 축의 이름을 지정한다. 디폴트로 지정된 축이름을 다른 것으로 바꾸기 위해 사용되는데, 디폴트 축이름으로는 주로 해당 high-level 그래픽 함수를 불러내기 위해 사용되었던 변수의 이름들이 나타납니다.

main=*string*

그림에 대한 제목, 플롯 윗 부분에 큰 폰트를 사용해서 표시된다.

sub=*string*

작은(sub)-제목, x 축 바로 아래에 좀 더 작은 폰트로 나타난다.

12.2 Low-level plotting commands

high-level 그래픽 함수로는 정학히 원하는 형태의 플롯을 그리는 것이 불가능한 경우가 있다. 이런 경우, low-level 명령문으로 현재의 플롯에 (포인트, 선 또는 문자와 같은) 추가적인 정보를 표현하는 것이 가능한다.

유용하게 사용될 수 있는 low-level 그래픽 함수로는 다음과 같은 것들이 있습니다:

points(x, y)
lines(x, y)

현재의 함수에 포인트나 이를 연결한 선을 첨가한다. plot()와 이에 포함된 “type=”선언이 이러한 함수들과 비슷하게 사용될 수 있다. (디폴트인 “p”는 points()과 "l"은 lines()과 같이 사용될 수 있다.)

text(x, y, labels, ...)

x 와 y 값으로 표현된 점들에 문자를 첨가한다. 보통 정수나 문자의 벡터 형태로 레이블이 주어지며, 이 경우 특정 i 에 대해 (x[i], y[i])위치에 labels[i] 값이 표시된다. 디폴트로 1:length(x)이 지정되어 있다.

주의 : 이 함수는 다음과 같은 순서로 사용되는 경우가 많다.

> plot(x, y, type="n"); text(x, y, names)

그래픽 parameter 인 type="n"이 포인트가 표현되는 것을 제한하고 있지만, 축은 이미 세워져 있고, text() 함수는 각 포인트들에 대한 문자 vector 형태의 이름이 표현되도록 한다.

abline(a, b)
abline(h=*y*)
abline(v=*x*)
abline(*lm.obj*)

기울기 b 이고 y 절편 a 인 선을 현재의 플롯에 첨가한다. h=*y* 는 y 좌표값을 높이로 하는 가로선이 이 그래프를 가로질러 그리며, 마찬가지로 v=*x* 은 x 좌표값을 시작점으로 하는 세로선을 그린다. 또, *lm.obj* 는 (모형적합 함수의 결과로 얻어진) 길이 2 의 coefficient 성분들을 표시하는데, 이 경우에는 순서대로 y 절편과 기울기 값에 해당한다.

polygon(x, y, ...)

(x,y)에 의해 정의된 꼭지점들을 순서대로 연결해서 만든 다각형을 그린다. 해당 그래픽 장치가 기능을 지원하면, (옵션을 사용해서) 그려진 다각형의 내부에 음영을 주거나 색을 채우는 것이 가능하다.

legend(x, y, legend, ...)

특정한 위치에 현재 플롯의 범례를 첨가한다. 플롯에 사용된 문자나 선의 종류 혹은 색 등이 범례에 포함된 문자 레이블을 통해 구분 가능해진다. 다음과 같이 플롯 구성 단위가 가질 수 있는 값들 중 최소한 한 개 이상의 *v* (범례와 같은 길이를 가진 벡터를 사용하여) 선언이 필요하다:

legend(, fill=*v*)

상자에 칠해진 색상들
legend(, col=*v*)

포인트나 선에 사용될 색상들
legend(, lty=*v*)

선의 종류 Line styles
legend(, lwd=*v*)

선의 굵기
legend(, pch=*v*)

문자표시를 포함할 것 (문자 벡터 형태)

title(main, sub)

현재 플롯의 윗부분에 큰 글시로 주요(main) 제목을 첨가하고 (선택에 의해) 좀 더 작은 크기의 클씨로 부(sub)제목을 아랫 부분에 표시한다.

axis(side, ...)

현재 플롯에 첫번째 인수(1 부터 4 까지의 숫자로 나타나며, 바닥에서부터 시계 방향으로 해당 사면을 의미함)에 의해 주어진 면에 새로운 축을 첨가한다. 다른 인수들로 플롯 안에 포함되거나 플롯 옆에 나타나는 축을 조절할 수도 있고, 위치나 레이블에 대한 *v*선언을 하기도한다. 사용자 정의로 축을 사용하기 위해서는 plot() 함수 안에 axes=FALSE 인수를 지정하는 편이 좋을 것이다.

Low-level 그래픽 함수들은 대체로 새로운 플롯 구성 요소를 어디에 배치할지 결정하기 휘애 위치에 관한 (x, y 좌표 같은) 정보를 요구합니다. 이런 좌표들은 이전의 high-level 그래픽 함수 명령문에 의해 정의된 사용자 좌표에 의해 표현되며, 이것은 주어진 데이터에 따라 결정됩니다.

X 와 y 인수값이 필요한 경우, x 와 y 로 명명된 성분들을 연결해서 표현한 하나의 인수로 표현하는 것 역시 가능합니다. 마찬가지로 두개의 열을 가진 행렬을 사용하는 것도 역시 유효한 방법입니다. 이 방법을 사용할 때는 locator() (아래 참조) 같은 함수를 사용해서 플롯을 표현하기 위한 여러 위치들을 interactive 하게 조정하는 것이 가능합니다.

12.2.1 Mathematical annotation

어떤 경우에는, 수학적 기호나 공식을 플롯에 첨가하는 것이 유용합니다. R 에서는 이런 작업을 text, mtext, axis 또는 title 같은 함수를 사용해서 하나의 문자열을 입력하는 것

보다는 하나의 표현(expression)을 사용해서 해결합니다. 예를 들면, 다음의 코드로는 이항분포 함수(binomial probability function)의 공식을 표현 됩니다:

```
> text(x, y, expression(paste(bgroup("(", atop(n, x), ")"), p^x, q^{n-x})))
```

이 함수에서 사용할 수 있는 모든 기능들을 포함해서 좀 더 많은 정보를 얻을 수 있는 R 명령문들은 다음과 같습니다:

```
> help(plotmath)
    > example(plotmath)
    > demo(plotmath)
```

12.2.2 Hershey vector fonts

Text 와 contour 함수를 사용할 때, text 를 조절하기 위해 Hershey 폰트를 사용하는 것이 가능합니다. Hershey 폰트를 사용하는 것은 다음의 세가지 이유에서 때문입니다:

- Hershey 폰트는 컴퓨터 스크린에서 text 를 회전시키거나 작은 text 를 사용할 때 특히 효과적입니다.

- Hershey 폰트는 일판 폰트에서는 제공하지 않는 몇몇 심볼들을 제공합니다. 특히, 조디악 기호 (zodiac sign)나 지도 제작용 (cartographic) 심볼 그리고 천문학 (astronomical) 심볼 등을 제공합니다.

- Hershey 폰트는 키릴(Cyrillic)과 일본어(Kana 와 Kanji 포함) 문자들을 제공합니다.

Hershey 문자표를 포함한 좀 더 많은 정보를 찾기 위해 R 에서 사용할 수 있는 명령문은 다음과 같습니다:

```
 > help(Hershey)
> demo(Hershey)
> help(Japanese)
> demo(Japanese)
```

12.3 Interacting with graphics

R 에서는 마우스를 사용하여 플롯에 관련된 정보를 빼거나 더하는 작업을 하는 것도 가능합니다. 이런 방식으로 작동하는 함수 중 중 가장 간단한 것은 locator()함수 입니다:

locator(n, type)

마우스 왼쪽 버튼을 사용하여 현재 플롯에 관련된 위치에 관련된 값들을 선택할 수 있도록 기다리는 함수. 이 함수는 n (디폴트는 512)개의 점들이 모두 선택될 까지 또는 다음 번 마우스 버튼을 누를 때까지 계속 작동한다. Type 인수는 선택된 포인들을 어떤 플롯으로 그릴지 결정하며 high-level 그래픽 함수에서와 같은 기능으로 사용된다; 디폴트로는 플롯을 표시하지 않는 것이 지정되어 있다. locator()는 선택된 포인트드의 위치들이 두 개의 x 와 y 의 성분으로 이루어진 하나의 리스트 값을 출력한다.

하지만 locator()함수는 인수들을 입력하지 않은 채 그대로 사용되는 경우가 더 많습니다. 이 함수는 범례나 레이블 같은 그래픽 요소들의 위치를 정할 때 interactive 하게 사용되는데, 특히 이러한 요소들을 사전에 해당 그래픽의 어느 곳에 배치해야할지 결정하기 어려운 경우에 유용하게 사용될 수 있습니다. 예를 들면, Outlier 와 같은 포인트에 해당 포인트에 대한 정보를 입력하기 위해 다음 명령문을 사용할 수 있습니다.

```
> text(locator(1), "Outlier", adj=0)
```

(locator() 함수는 현재 디바이스가 postscript 와 같이 interactive pointing 기능을 제공하지 않는 경우에는 그냥 무시되어 버립니다.)

identify(x, y, labels)

x 와 y 에 의해 정의된 어떤 포인트라도 (마우스의 왼쪽 버튼을 사용해서) 그 근처에 해당 포인트에 대응되는 레이블을 표기함으로써 그 포인트를 확인할 수 있도록 하는 함수. (레이블 정보가 없는 경우, 해당 포인트의 인덱스 번호를 사용) 버튼을 한 번 더 누르면 선택된 포인트들의 인덱스들을 출력한다.

때때로 우리는 플롯에서 특별한 몇 개의 포인트들의 위치를 보기 보다는 해당 포인트에 대한 정보를 확인하고 싶어집니다. 예를 들면, 그래픽으로 표현된 데이터 중 관심 있는 몇 개의 관측치만 선택해서 그 관측치들만 따로 처리하고 싶은 경우가 있습니다. 포인트들을 두 개의 숫자 벡터인 x 와 y 에 의해 표현된 (x, y) 좌표 쌍들로 표현하면, identify() 를 다음과 같이 사용할 수 있습니다:

```
> plot(x, y)
> identify(x, y)
```

identify() 함수는 그 자체로 플롯을 그리지는 않지만, 사용자들이 마우스 포인터를 움직이다 어떤 포인트에서는 마우스 왼쪽 버튼을 클릭할 수 있는 기능을 제공합니다. 어떤 포인트 근처에 마우스 포인터가 있는 경우, 해당 포인트는 인덱스 번호로 표시될 것 입니다. (이것은, 그 포인트의 위치가 x/y vector 로 되어 있음을 의미하는 것입니다.) 또한, identify()함수 내에 labels 인수를 사용하여 포인트들이 좀 더 informative 하게(해당 케이스의 이름 등) 표현되도록 할 수도 있으며, plot = FALSE 인수를 사용하면 포인트를 선택해도 아무런 표시가 남지 않도록 할 수도 있습니다. 해당 프로세스가 끝나고 나면(위 참조), identify()는 선택되었던 포인트들의 인덱스를 출력합니다; 이러한 인덱스 정보를 바탕으로 원래 x, y 의 벡터 형태로 되어 있는 선택된 포인트들의 정보를 따로 추출해내는 것도 가능할 것 입니다.

12.4 Using graphics parameters

그래픽을 만들어 낼 때, 특히 그것이 프리젠테이션이나 다른 공공 목적을 위한 것이라면, R 의 디폴트만으로는 항상 원하는 형태를 얻을 수 없을 것입니다. 그렇지만, 그래픽 모수(*graphics parameters)*를 이용하여 거의 모든 표현 방식을 사용자 임의대로 정의할 수 있습니다. R 에는 선의 형태, 색, 그림 배열 그리고 문자 정의 등에 이르기까지 굉장히 다양한 그래픽 모수들이 포함되어 있습니다. 모든 그래픽 모수는 이름(e.g.색을 결정하는 'col')과 값(특정색을 의미하는 숫자)으로 구성 됩니다.

각각의 디바이스를 위해 그래픽 모수의 리스트들을 따로 정의하는 것이 가능하고, 각 디바스들은 모수들의 디폴트 값을 가진 채 초기화 됩니다. 그래픽 모수들은 크게 두 가지 방법으로 사용될 수 있습니다: 하나는 현재 디바이스를 사용할 때마다 모든 그래픽 함수들에 영향을 주는 "영구적" 방법이고 다른 방법은 오직 하나의 그래픽 함수를 사용할 때에만 영향을 주는 "임시적" 방법입니다.

12.4.1 Permanent changes: The par() function

par()함수는 현재의 그래픽 디바이스의 그래픽 모수 리스트를 조정하기 위해 사용됩니다.

par()
아무런 인수가 없는 경우, 현재 디바스에 저장되어 있는 모든 그래픽 모수와 그 모수들 해당 값의 리스트를 출력합니다.
par(c("col", "lty"))

문자 벡터 인수를 사용하면, 오직 따로 지정된 그래픽 모수들에 대해서만 (이 경우도 리스트의 형태로) 출력합니다.
par(col=4, lty=2)
그래픽 모수와 해당 모수의 값들을 따로 인수(하나의 인수도 가능)로 지정하면, 지정된 값이 리스트로 출려됩니다. (?)

par()함수를 사용해서 그래픽 모수를 지정하면 해당 모수와 그 값은 “영구적”으로 바뀝니다. 즉 미래에 해당 그래픽 함수를 (현재 디바이스에서) 사용하면 새로 지정된 값이 반영되어 출력되는 것 입니다. 따라서 이 방법을 사용해서 그래픽 모수를 지정하는 것은 일종의 “디폴트” 값을 지정하는 것과 같다고 생각할 수 있을 겁니다. 즉, 다른 값이 다시 지정되지 않는 한, 모든 그래픽 함수가 이 지정값에 영향을 받게 되는 것입니다.

par()를 사용하면, 항상, 심지어 par()가 다른 함수 안에서 사용된 경우라도 전체적으로 모든 그래픽 모수들의 값이 영향을 받게 된다는 점을 주의하시기 바랍니다. 이것은 많은 경우 그다지 바람직하지 않다고 볼 수 있습니다 – 우리가 몇 개의 그래픽 모수를 따로 지정하는 것은 대체로 몇 개의 플롯을 그린 후, 현재의 R 세션이 이 값에 영향 받기 전 상태인 원래의 모수값으로 돌아가고 싶기 때문입니다. 특별히 모수를 바꾸는 작업이 필요할 때 par()의 결과를 따로 저장하고 플롯 작업이 끝난 후에는 초기 값을 다시 사용함으로써 초기값을 계속 유지하는 것이 가능합니다.

```
> oldpar <- par(col=4, lty=2)
    ... plotting commands ...
  > par(oldpar)
```

조정 가능한 모든 그래픽 모수값을 저장해서 다시 사용하기 위해서는

```
    > oldpar <- par(no.readonly=TRUE)
    ... plotting commands ...
  > par(oldpar)
```

12.4.2 Temporary changes: Arguments to graphics functions

그래픽 모수들은 (거의 모든) 그래픽 함수에서 인수처럼 사용될 수도 있습니다. 이러한 방법은 인수로 사용되었을 때의 변화가 오직 해당 함수가 사용되는 동안만 지속된다는 점을 제외하고는, par()함수에 인수들을 사용하는 것과 같은 효과를 줍니다. 예를 들면:

```
> plot(x, y, pch="+")
```

이 경우, 더하기 표시를 플롯 문자로 사용한 산점도를 그리게 되는데, 이 때 사용된 플롯 문자는 앞으로 사용하게 될 플롯의 디폴트에는 영향을 주지 않습니다.

안타깝지만, 항상 이와 같은 형태로만 실행되는 것이 아니므로 경우에 따라 par()를 사용해서 그래픽 모수들을 지정하고 다시 되돌리는 작업이 필요하기도 합니다.

12.5 Graphics parameters list

The following sections detail many of the commonly-used graphical parameters. The R help documentation for the `par()` function provides a more concise summary; this is provided as a somewhat more detailed alternative.

Graphics parameters will be presented in the following form:

name=*value*

> A description of the parameter's effect. *name* is the name of the parameter, that is, the argument name to use in calls to `par()` or a graphics function. *value* is a typical value you might use when setting the parameter.

Note that `axes` is **not** a graphics parameter but an argument to a few `plot` methods: see `xaxt` and `yaxt`.

12.5.1 Graphical elements

R plots are made up of points, lines, text and polygons (filled regions.) Graphical parameters exist which control how these *graphical elements* are drawn, as follows:

`pch="+"`

Character to be used for plotting points. The default varies with graphics drivers, but it is usually a circle. Plotted points tend to appear slightly above or below the appropriate position unless you use "." as the plotting character, which produces centered points.

pch=4

When pch is given as an integer between 0 and 25 inclusive, a specialized plotting symbol is produced. To see what the symbols are, use the command

```
> legend(locator(1), as.character(0:25), pch = 0:25)
```

Those from 21 to 25 may appear to duplicate earlier symbols, but can be coloured in different ways: see the help on points and its examples.

In addition, pch can be a character or a number in the range 32:255 representing a character in the current font.

lty=2

Line types. Alternative line styles are not supported on all graphics devices (and vary on those that do) but line type 1 is always a solid line, line type 0 is always invisible, and line types 2 and onwards are dotted or dashed lines, or some combination of both.

lwd=2

Line widths. Desired width of lines, in multiples of the “standard” line width. Affects axis lines as well as lines drawn with lines(), etc. Not all devices support this, and some have restrictions on the widths that can be used.

col=2

Colors to be used for points, lines, text, filled regions and images. A number from the current palette (see ?palette) or a named colour.

col.axis

col.lab

col.main

col.sub

The color to be used for axis annotation, x and y labels, main and sub-titles, respectively.

font=2

An integer which specifies which font to use for text. If possible, device drivers arrange so that 1 corresponds to plain text, 2 to bold face, 3 to italic, 4 to bold italic and 5 to a symbol font (which include Greek letters).

font.axis
font.lab
font.main
font.sub

The font to be used for axis annotation, x and y labels, main and sub-titles, respectively.

adj=-0.1

Justification of text relative to the plotting position. 0 means left justify, 1 means right justify and 0.5 means to center horizontally about the plotting position. The actual value is the proportion of text that appears to the left of the plotting position, so a value of -0.1 leaves a gap of 10% of the text width between the text and the plotting position.

cex=1.5

Character expansion. The value is the desired size of text characters (including plotting characters) relative to the default text size.

cex.axis
cex.lab
cex.main
cex.sub

The character expansion to be used for axis annotation, x and y labels, main and sub-titles, respectively.

12.5.2 Axes and tick marks

Many of R's high-level plots have axes, and you can construct axes yourself with the low-level axis() graphics function. Axes have three main components: the *axis line* (line style controlled by the lty graphics parameter), the *tick marks* (which mark off unit divisions along the axis line) and the *tick labels* (which mark the units.) These components can be customized with the following graphics parameters.

lab=c(5, 7, 12)

The first two numbers are the desired number of tick intervals on the x and y axes respectively. The third number is the desired length of axis labels, in

characters (including the decimal point.) Choosing a too-small value for this parameter may result in all tick labels being rounded to the same number!

las=1

Orientation of axis labels. 0 means always parallel to axis, 1 means always horizontal, and 2 means always perpendicular to the axis.

mgp=c(3, 1, 0)

Positions of axis components. The first component is the distance from the axis label to the axis position, in text lines. The second component is the distance to the tick labels, and the final component is the distance from the axis position to the axis line (usually zero). Positive numbers measure outside the plot region, negative numbers inside.

tck=0.01

Length of tick marks, as a fraction of the size of the plotting region. When tck is small (less than 0.5) the tick marks on the x and y axes are forced to be the same size. A value of 1 gives grid lines. Negative values give tick marks outside the plotting region. Use tck=0.01 and mgp=c(1,-1.5,0) for internal tick marks.

xaxs="r"

yaxs="i"

Axis styles for the x and y axes, respectively. With styles "i" (internal) and "r" (the default) tick marks always fall within the range of the data, however style "r" leaves a small amount of space at the edges. (S has other styles not implemented in R.)

12.5.3 Figure margins

A single plot in R is known as a figure and comprises a *plot region* surrounded by margins (possibly containing axis labels, titles, etc.) and (usually) bounded by the axes themselves.

Graphics parameters controlling figure layout include:

mai=c(1, 0.5, 0.5, 0)

Widths of the bottom, left, top and right margins, respectively, measured in inches.

mar=c(4, 2, 2, 1)

Similar to mai, except the measurement unit is text lines.

mar and mai are equivalent in the sense that setting one changes the value of the other. The default values chosen for this parameter are often too large; the right-hand margin is rarely needed, and neither is the top margin if no title is being used. The bottom and left margins must be large enough to accommodate the axis and tick labels. Furthermore, the default is chosen without regard to the size of the device surface: for example, using the postscript() driver with the height=4 argument will result in a plot which is about 50% margin unless mar or mai are set explicitly. When multiple figures are in use (see below) the margins are reduced, however this may not be enough when many figures share the same page.

12.5.4 Multiple figure environment

R allows you to create an n by m array of figures on a single page. Each figure has its own margins, and the array of figures is optionally surrounded by an *outer margin*, as shown in the following figure.

The graphical parameters relating to multiple figures are as follows:

mfcol=c(3, 2)
mfrow=c(2, 4)

Set the size of a multiple figure array. The first value is the number of rows; the second is the number of columns. The only difference between these two parameters is that setting mfcol causes figures to be filled by column; mfrow fills by rows.

The layout in the Figure could have been created by setting mfrow=c(3,2); the figure shows the page after four plots have been drawn.

Setting either of these can reduce the base size of symbols and text (controlled by par("cex") and the pointsize of the device). In a layout with exactly two rows and columns the base size is reduced by a factor of 0.83: if there are three or more of either rows or columns, the reduction factor is 0.66.

mfg=c(2, 2, 3, 2)

Position of the current figure in a multiple figure environment. The first two numbers are the row and column of the current figure; the last two are the number of rows and columns in the multiple figure array. Set this parameter to jump between figures in the array. You can even use different values for the last two numbers than the *true* values for unequally-sized figures on the same page.

`fig=c(4, 9, 1, 4)/10`

Position of the current figure on the page. Values are the positions of the left, right, bottom and top edges respectively, as a percentage of the page measured from the bottom left corner. The example value would be for a figure in the bottom right of the page. Set this parameter for arbitrary positioning of figures within a page. If you want to add a figure to a current page, use `new=TRUE` as well (unlike S).

`oma=c(2, 0, 3, 0)`
`omi=c(0, 0, 0.8, 0)`

Size of outer margins. Like `mar` and `mai`, the first measures in text lines and the second in inches, starting with the bottom margin and working clockwise.

Outer margins are particularly useful for page-wise titles, etc. Text can be added to the outer margins with the `mtext()` function with argument `outer=TRUE`. There are no outer margins by default, however, so you must create them explicitly using `oma` or `omi`.

More complicated arrangements of multiple figures can be produced by the `split.screen()` and `layout()` functions, as well as by the **grid** and **lattice** packages.

12.6 Device drivers

R can generate graphics (of varying levels of quality) on almost any type of display or printing device. Before this can begin, however, R needs to be informed what type of device it is dealing with. This is done by starting a *device driver*. The purpose of a device driver is to convert graphical instructions from R (“draw a line,” for example) into a form that the particular device can understand.

Device driver 는 device driver function 에 의해

Device drivers are started by calling a device driver function. There is one such function for every device driver: type `help(Devices)` for a list of them all. For example, issuing the command

```
> postscript()
```

causes all future graphics output to be sent to the printer in PostScript format. Some commonly-used device drivers are:

`X11()`
: For use with the X11 window system on Unix-alikes

`windows()`
: For use on Windows

`quartz()`
: For use on Mac OS X

`postscript()`
: For printing on PostScript printers, or creating PostScript graphics files.

`pdf()`
: Produces a PDF file, which can also be included into PDF files.

`png()`
: Produces a bitmap PNG file. (Not always available: see its help page.)

`jpeg()`
: Produces a bitmap JPEG file, best used for `image` plots. (Not always available: see its help page.)

When you have finished with a device, be sure to terminate the device driver by issuing the command

```
> dev.off()
```

This ensures that the device finishes cleanly; for example in the case of hardcopy devices this ensures that every page is completed and has been sent to the printer. (This will happen automatically at the normal end of a session.)

12.6.1 PostScript diagrams for typeset documents

파일 명령문을 postscript()라는 device driver 함수로 , 그래픽들을 PostScript 포맷의 파일의 형태

By passing the file argument to the postscript() device driver function, you may store the graphics in PostScript format in a file of your choice. The plot will be in landscape orientation unless the horizontal=FALSE argument is given, and you can control the size of the graphic with the width and height arguments (the plot will be scaled as appropriate to fit these dimensions.) For example, the command

```
> postscript("file.ps", horizontal=FALSE, height=5, pointsize=10)
```

will produce a file containing PostScript code for a figure five inches high, perhaps for inclusion in a document. It is important to note that if the file named in the command already exists, it will be overwritten. This is the case even if the file was only created earlier in the same R session.

Many usages of PostScript output will be to incorporate the figure in another document. This works best when *encapsulated* PostScript is produced: R always produces conformant output, but only marks the output as such when the onefile=FALSE argument is supplied. This unusual notation stems from S-compatibility: it really means that the output will be a single page (which is part of the EPSF specification). Thus to produce a plot for inclusion use something like

```
> postscript("plot1.eps", horizontal=FALSE, onefile=FALSE,
        height=8, width=6, pointsize=10)
```

12.6.2 Multiple graphics devices

In advanced use of R it is often useful to have several graphics devices in use at the same time. Of course only one graphics device can accept graphics commands at any one time, and this is known as the *current device*. When multiple devices are open, they form a numbered sequence with names giving the kind of device at any position.

The main commands used for operating with multiple devices, and their meanings are as follows:

X11()

[UNIX]

windows()

win.printer()

win.metafile()

[Windows]

quartz()

[Mac OS X]

postscript()

pdf()

png()

jpeg()

tiff()

bitmap()

...

Each new call to a device driver function opens a new graphics device, thus extending by one the device list. This device becomes the current device, to which graphics output will be sent.

dev.list()

Returns the number and name of all active devices. The device at position 1 on the list is always the *null device* which does not accept graphics commands at all.

dev.next()

dev.prev()

Returns the number and name of the graphics device next to, or previous to the current device, respectively.

dev.set(which=*k*)

Can be used to change the current graphics device to the one at position *k* of the device list. Returns the number and label of the device.

dev.off(*k*)

Terminate the graphics device at point *k* of the device list. For some devices, such as postscript devices, this will either print the file immediately or correctly complete the file for later printing, depending on how the device was initiated.

dev.copy(device, ..., which=*k*)

dev.print(device, ..., which=*k*)

Make a copy of the device *k*. Here device is a device function, such as postscript, with extra arguments, if needed, specified by '...'. dev.print is similar, but the copied device is immediately closed, so that end actions, such as printing hardcopies, are immediately performed.

graphics.off()

Terminate all graphics devices on the list, except the null device.

12.7 Dynamic graphics

Dynamic 또는 interactive 한 그래픽 built-in 기능을 제공하지 않습니다. (e.g. rotating point clouds or to "brushing" (interactively highlighting) points) 그렇지만, Swayne, Cook 과 Buja 에 의해 개발된 GGobi 시스템 (http://www.ggobi.org)을 통해 충분한 dynamic 그래픽 기능들을 사용하는 것이 가능합니다. 이러한 GGobi 기능들을 이용하기 위해서는 R 의 팩키지 중의 하나인 **rggobi** 를 사용해야 합니다. (사용법: http://www.ggobi.org/rggobi)

또한, **rgl** 팩키지는 3D plot 을 사용하여 이러한 plot 의 표면(surface) 등에서 interactive 하게 작업할 수 있도록 해 줍니다.