```
par(mfrow=c(2,2))
plot(x,y,pch=16)
plot(log(x),log(y),pch=16)
plot(x,y,pch=16,log="xy")
plot(x,y,pch=16,log="y")
```

The untransformed data are in the top left-hand graph, and both x and y are transformed to logs before plotting in the upper right. The bottom left-hand plot shows both axes log-transformed, while the bottom right shows the data with only the y axis log-transformed. Note that all logs in R are to the base e by default (not base 10). It is important to understand that when R is asked to plot the log of zero it simply omits any such points without comment (compare the top left-hand graph with a point at $(0, 0)$ with the other three graphs).

Axis Labels Containing Subscripts and Superscripts

The default xlab and ylab do not allow subscripts like r2 or superscripts or xi. For these you need to master the expression function. In R, the operator for superscripts is 'hat' (or, more correctly, 'caret') so 'r squared' is written r^2. Likewise subscripts in R involve square brackets [] so x_i is written x[i]. Suppose that we want r^2 to be the label on the y axis and x_i to be the label on the x axis. The expression function turns R code into text like this:

```
plot(1:10,1:10, ylab=expression(r^2), xlab=expression(x[i]),type="n")
```

Different font families for text

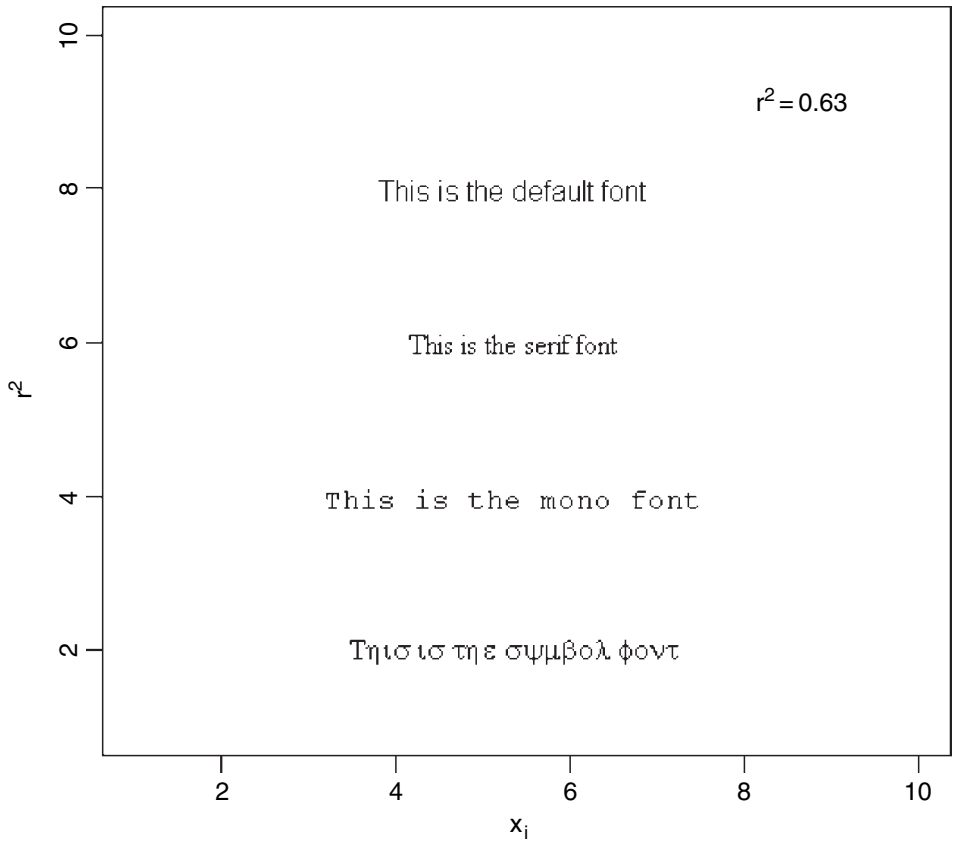
To change the typeface used for plotted text, change the name of a font family. Standard values are family = "serif", "sans" (the default font), "mono", and "symbol", and the Hershey font families are also available. Some devices will ignore this setting completely. Text drawn onto the plotting region is controlled using par like this:

```
par(family="sans")
text(5,8,"This is the default font")
par(family="serif")
text(5,6,"This is the serif font")
par(family="mono")
text(5,4,"This is the mono font")
par(family="symbol")
text(5,2,"This is the symbol font")
par(family="sans")
```

Don't forget to turn the family back to "sans", otherwise you may get some very unexpected symbols in your next text. To write the results of calculations using text, it is necessary to use substitute with expression. Here, the coefficient of determination (cd) was calculated earlier and we want to write its value on the plot, labelled with '$r^2 =$':

```
cd<- 0.63
...
text(locator(1),as.expression(substitute(r^2 == cd,list(cd=cd))))
```

Just click when the cursor is where you want the text to appear. Note the use of 'double equals'.



Mathematical Symbols on Plots

To write on plots using more intricate symbols such as mathematical symbols or Greek letters we use expression or substitute. Here are some examples of their use. First, we produce a plot of sin ϕ against the phase angle ϕ over the range $-\pi$ to $+\pi$ radians:

```
x <- seq(-4, 4, len = 101)
plot(x,sin(x),type="l",xaxt="n",
        xlab=expression(paste("Phase Angle ",phi)),
        ylab=expression("sin "*phi))
axis(1, at = c(-pi, -pi/2, 0, pi/2, pi),
        lab = expression(-pi, -pi/2, 0, pi/2, pi))
```

Note the use of xaxt = "n" to suppress the default labelling of the x axis, and the use of expression in the labels for the x and y axes to obtain mathematical symbols such as phi (ϕ) and pi (π). The more intricate values for the tick marks on the x axis are obtained by the axis function, specifying 1 (the x ('bottom') axis is axis no. 1), then using the at function to say where the labels and tick marks are to appear, and lab with expression to say what the labels are to be.

Suppose you wanted to add $\chi^2 = 24.5$ to this graph at location $(-\pi/2, 0.5)$. You use text with substitute, like this:

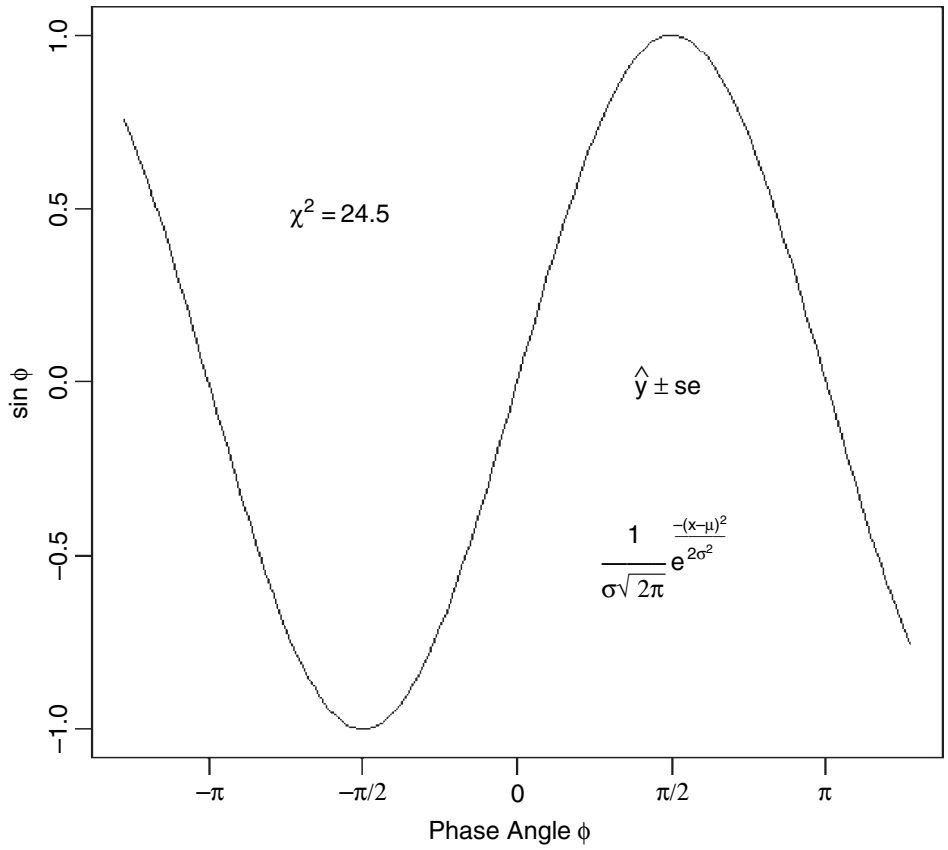
```
text(-pi/2,0.5,substitute(chi^2=="24.5"))
```

Note the use of 'double equals' to print a single equals sign. You can write quite complicated formulae on plots using paste to join together the elements of an equation. Here is the density function of the normal written on the plot at location $(\pi/2, -0.5)$:

```
text(pi/2, -0.5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
      e^{frac(-(x-mu)^2, 2*sigma^2)})))
```

Note the use of frac to obtain individual fractions: the first argument is the text for the numerator, the second the text for the denominator. Most of the arithmetic operators have obvious formats $(+, -, /, *, \hat{}$, etc.); the only non-intuitive symbol that is commonly used is 'plus or minus' $\pm$; this is written as $\%+-\%$ like this:

```
text(pi/2,0,expression(hat(y) %+-% se))
```



Phase Planes

Suppose that we have two competing species (named 1 and 2) and we are interested in modelling the dynamics of the numbers of each species (N_1 and N_2). We want to draw a phase plane showing the behaviour of the system close to equilibrium. Setting the derivates to zero and dividing both sides by $r_i N_i$ we get

$$0 = 1 - \alpha_{11} N_1 - \alpha_{12} N_2,$$

which is called the isocline for species 1. It is linear in N_1 and N_2 and we want to draw it on a phase plane with N_2 on the y axis and N_1 on the x axis. The intercept on the y axis shows the abundance of N_2 when $N_1 = 0$: this is $1/\alpha_{12}$. Likewise, when $N_2 = 0$ we can see that $N_1 = 1/\alpha_{11}$ (the value of its single-species equilibrium). Similarly,

$$0 = 1 - \alpha_{21}N_1 - \alpha_{22}N_2$$

describes the isocline for species 2. The intercept on the y axis is $1/\alpha_{22}$ and the value of N_1 when $N_2 = 0$ is $1/\alpha_{21}$. Now we draw a phase plane with both isoclines, and label the ends of the lines appropriately. We might as well scale the axes from 0 to 1 but we want to suppress the default tick marks:

```
plot(c(0,1),c(0,1),ylab="",xlab="",xaxt="n",yaxt="n",type="n")
abline(0.8,-1.5)
abline(0.6,-0.8,lty=2)
```

The solid line shows the isocline for species 1 and the dotted line shows species 2.

Now for the labels. We use at to locate the tick marks – first the x axis (axis = 1),

```
axis(1, at = 0.805, lab = expression(1/alpha[21]))
axis(1, at = 0.56, lab = expression(1/alpha[11]))
```

and now the y axis (axis = 2),

```
axis(2, at = 0.86, lab = expression(1/alpha[12]),las=1)
axis(2, at = 0.63, lab = expression(1/alpha[22]),las=1)
```

Note the use of las=1 to turn the labels through 90 degrees to the horizontal. Now label the lines to show which species isocline is which. Note the use of the function fract to print fractions and square brackets (outside the quotes) for subscripts:

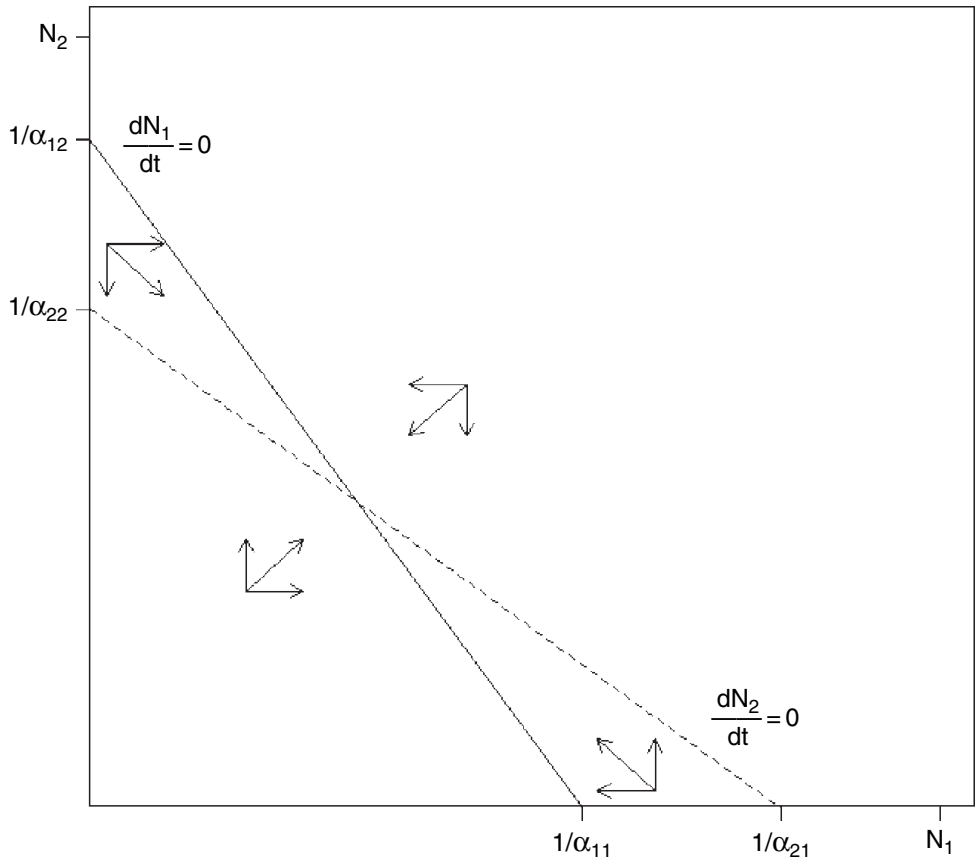
```
text(0.05,0.85, expression(paste(frac("d N"[1],"dt"), " = 0" )))
text(0.78,0.07, expression(paste(frac("d N"[2],"dt"), " = 0" )))
```

We need to draw phase plane trajectories to show the dynamics. Species will increase when they are at low densities (i.e. 'below' their isoclines) and decrease at high densities (i.e. 'above' their isoclines). Species 1 increasing is a horizontal arrow pointing to the right. Species 2 declining is a vertical arrow pointing downwards. The resultant motion shows how both species' abundances change through time from a given point on the phase plane.

```
arrows(-0.02,0.72,0.05,0.72,length=0.1)
arrows(-0.02,0.72,-0.02,0.65,length=0.1)
arrows(-0.02,0.72,0.05,0.65,length=0.1)

arrows(0.65,-0.02,0.65,0.05,length=0.1)
arrows(0.65,-0.02,0.58,-0.02,length=0.1)
arrows(0.65,-0.02,0.58,0.05,length=0.1)

arrows(0.15,0.25,0.15,0.32,length=0.1)
arrows(0.15,0.25,0.22,0.25,length=0.1)
arrows(0.15,0.25,0.22,0.32,length=0.1)

arrows(.42,.53,.42,.46,length=0.1)
arrows(.42,.53,.35,.53,length=0.1)
arrows(.42,.53,.35,.46,length=0.1)
```

All the motions converge, so the point is a stable equilibrium and the two species would coexist. All other configurations of the isoclines lead to competitive exclusion of one of the two species. Finally, label the axes with the species' identities:

```
axis(1, at = 1, lab = expression(N[1]))
axis(2, at = 1, lab = expression(N[2]),las=1)
```



Fat Arrows

You often want to add arrows to plots in order to draw attention to particular features. Here is a function called fat.arrows that uses locator(1) to identify the bottom of the point of a vertical fat arrow. You can modify the function to draw the arrow at any specified angle to the clicked point of its arrowhead. The default widths and heights of the arrow are 0.5 scaled *x* or *y* units and the default colour is red:

```
fat.arrow<-function(size.x=0.5,size.y=0.5,ar.col="red"){
size.x<-size.x*(par("usr")[2]-par("usr")[1])*0.1
size.y<-size.y*(par("usr")[4]-par("usr")[3])*0.1
pos<-locator(1)
xc<-c(0,1,0.5,0.5,-0.5,-0.5,-1,0)
yc<-c(0,1,1,6,6,1,1,0)
polygon(pos$x+size.x*xc,pos$y+size.y*yc,col=ar.col) }
```

We will use this function later in this chapter (p. 857).

Trellis Plots

You need to load the lattice package and set the background colour to white. You can read the details in ?trellis.device.

```
library(lattice)
```

The most commonly use trellis plot is xyplot, which produces conditional scatterplots where the response, y, is plotted against a continuous explanatory variable x, for different levels of a conditioning factor, or different values of the shingles of a conditioning variable. This is the standard plotting method that is used for linear mixed-effects models and in cases where there are nested random effects (i.e. with groupedData see p. 668). The structure of the plot is typically controlled by the formula; for example

```
xyplot(y ~ x | subject)
```

where a separate graph of y against x is produced for each level of subject (the vertical bar | is read as 'given'). If there are no conditioning variables, xyplot(y ~ x), the plot produced consists of a single panel. All arguments that are passed to a high-level trellis function like xyplot that are not recognized by it are passed through to the panel function. It is thus generally good practice when defining panel functions to allow a . . . argument. Such extra arguments typically control graphical parameters.

Panels are by default drawn starting from the bottom left-hand corner, going right and then up, unless as.table = TRUE, in which case panels are drawn from the top left-hand corner, going right and then down. Both of these orders can be modified using the index.cond and perm.cond arguments. There are some grid-compatible replacements for commonly used base R graphics functions: for example, lines can be replaced by llines (or equivalently, panel.lines). Note that base R graphics functions like lines will not work in a lattice panel function. The following example is concerned with root growth measured over time, as repeated measures on 12 individual plants:

```
results<-read.table("c:\\temp\\fertilizer.txt",header=T)
attach(results)
names(results)
```

```
[1] "root"  "week"  "plant"  "fertilizer"
```

Panel scatterplots

Panel plots are very easy to use. Here is a set of 12 scatterplots, showing root ~ week with one panel for each plant like this: | plant

```
xyplot(root ~ week | plant)
```

By default, the panels are shown in alphabetical order by plant name from bottom left (ID1) to top right (ID9). If you want to change things like the plotting symbol you can do this within the xyplot function,

```
xyplot(root ~ week | plant,pch=16)
```

but if you want to make more involved changes, you should use a panel function. Suppose we want to fit a separate linear regression for each individual plant. We write