

Introduction to plotting in R

Michael Lundholm*

May 18, 2009

1 `plot()`

R has a generic function for the plotting of R objects.¹ Basically it takes the arguments `plot(x, y, ...)`, where `x` is the `x`-coordinates of the plot and `y` is the `y`-coordinates of the plot; alternatively `x` could be a single plotting structure containing both `x`- and `y`-coordinates. `...` are arguments that are passed on to other methods. Many methods accept the following arguments:

type What type of plot should be drawn. Here are some examples

- `p` for **p**oints,
- `l` for **l**ines,
- `h` for **h**istogram like

The argument have the syntax e.g., `type="l"`.

main The overall title the plot. Syntax: `main="Main title text"`.

sub The subtitle the plot. Syntax: `sub="Sub title text"`.

xlab Title for `x`-axis. Syntax: `xlab="X-axis text"`.

ylab Title for `y`-axis. Syntax: `ylab="Y-axis text"`,

In the following we use a data set which is available in all R installations to illustrate the plotting commands. It is called `cars` and contains speed (in miles per hour) and stopping distances (in feet) recorded on cars in the 1920's.

```
> data(cars)
> str(cars)
```

*Department of Economics, Stockholm University, michael.lundholm@ne.su.se.

¹R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.

```
'data.frame':      50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

```
> head(cars)
```

```
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

We note that `cars` is a data frame with two variables `speed` and `dist`. We first plot using the syntax `plot(x,y)` (graph in Figure 1):

```
> plot(cars$speed, cars$dist)
```

If data is attached we can employ (graph in Figure 2)²

```
> attach(cars)
> plot(speed, dist)
```

But since `cars` just contains the two variables that we want to plot we can employ the syntax `plot(x)` to get the same result (graph in Figure 3):

```
> plot(cars)
```

The latter two method have the advantage that the labels on the axis become nicer.

We can also try other plot type. However, `type="l"` does not make sense with this data; try yourself! So we try `type="h"` for a histogram like plot (graph in Figure 4):

```
> plot(cars, type = "h")
```

We close this section with a plot which exploits most of the different arguments mentioned above. There are small but noticeable difference compared to the plain `plot(cars)` (graph in Figure 5):

```
> plot(cars, main = "Speed and stopping distances for cars",
+      sub = "Source: M. Ezekiel, Methods of Correlation Analysis. Wiley, 1930",
+      xlab = "Speed (miles per hour)", ylab = "Stopping distance (feet)")
```

²Note however, that this procedure in this case makes the variable `dist` mask the function `dist()`.

Figure 1: `plot(cars$ speed,cars$ dist)`

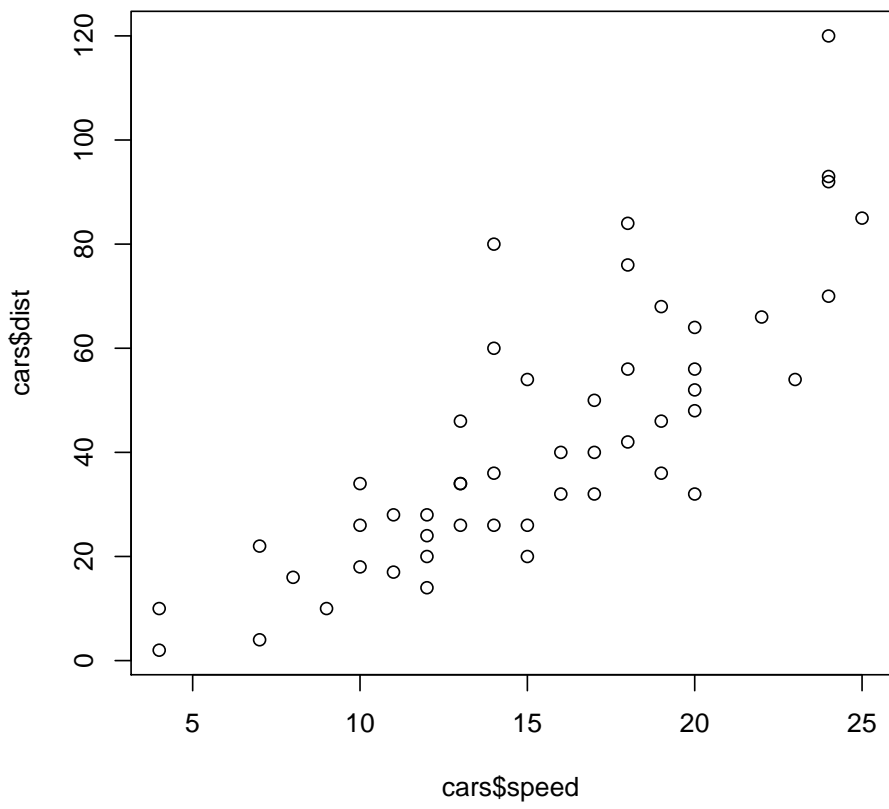


Figure 2: `plot(speed,dist)` with `attach(cars)`

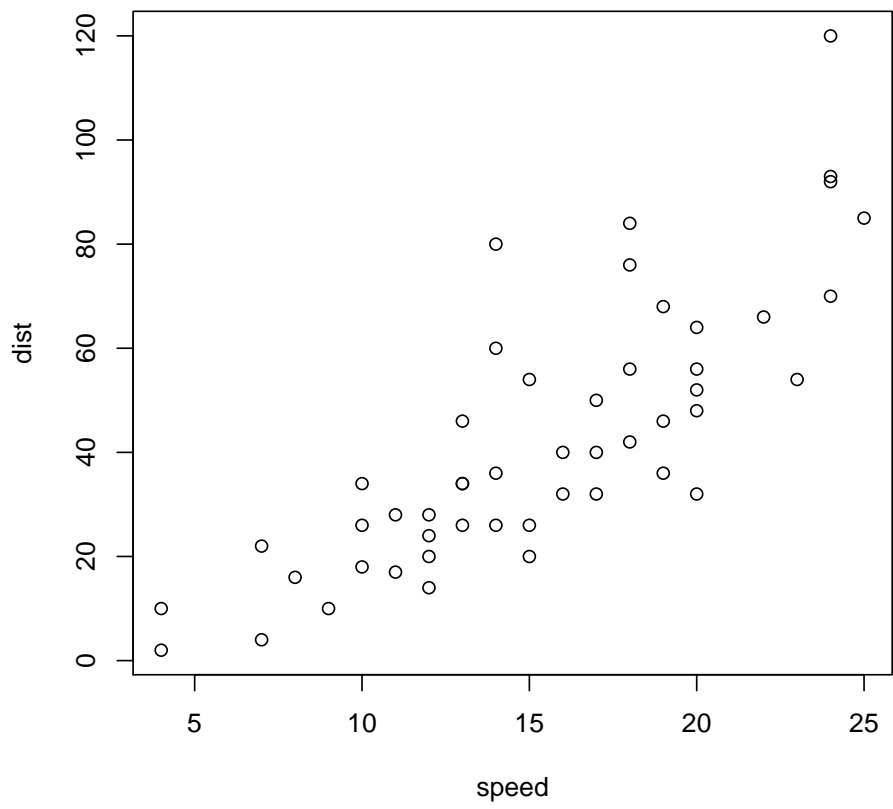


Figure 3: plot(cars)

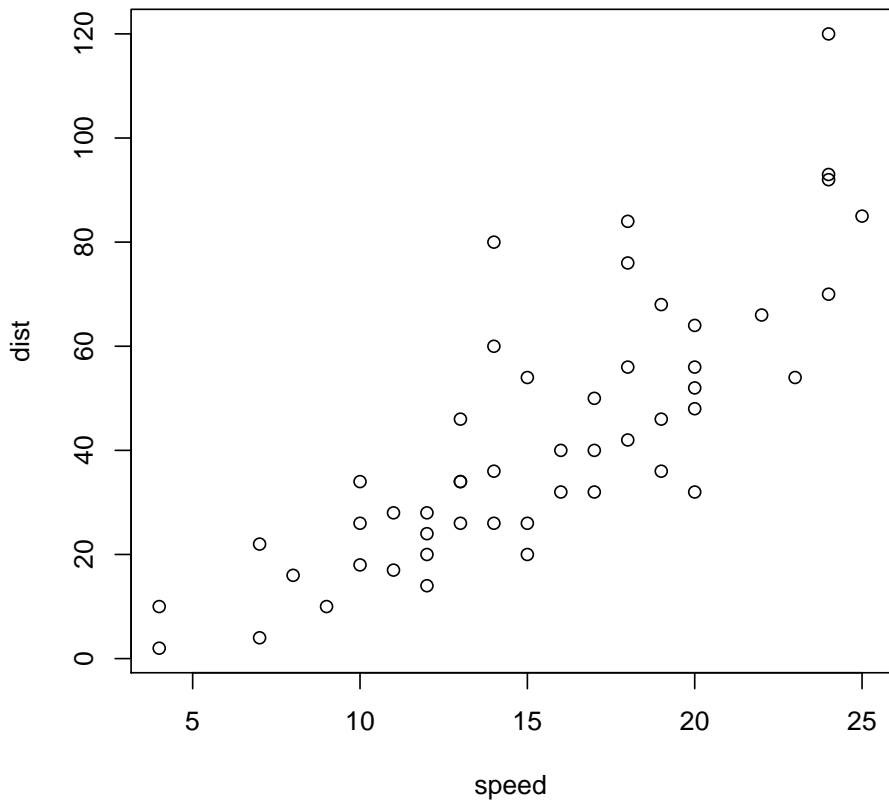


Figure 4: `plot(cars, type="h")`

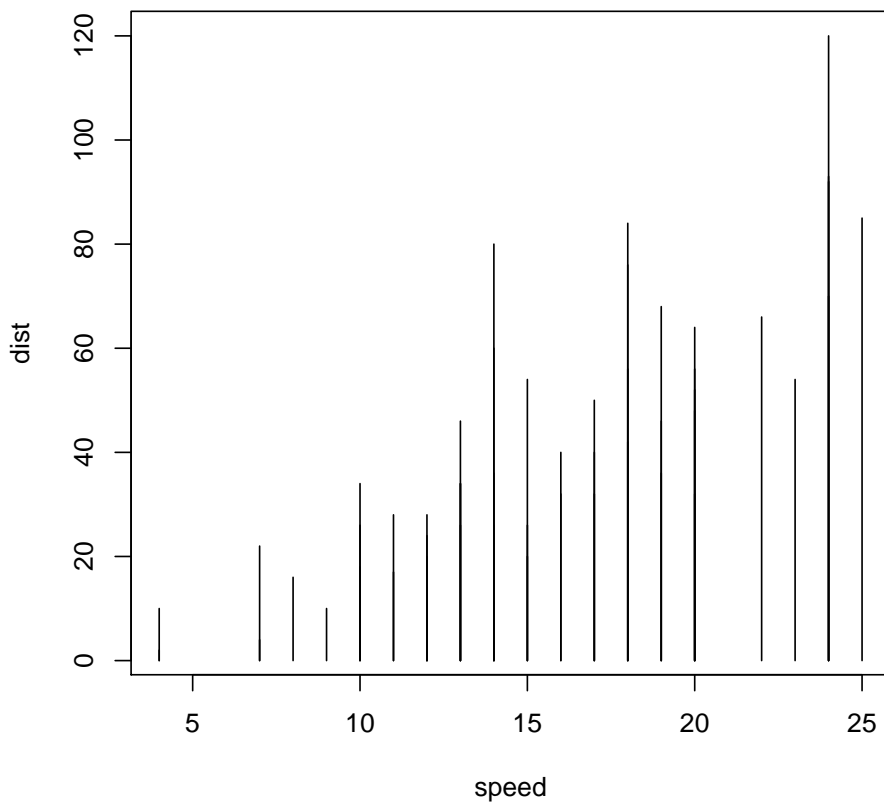
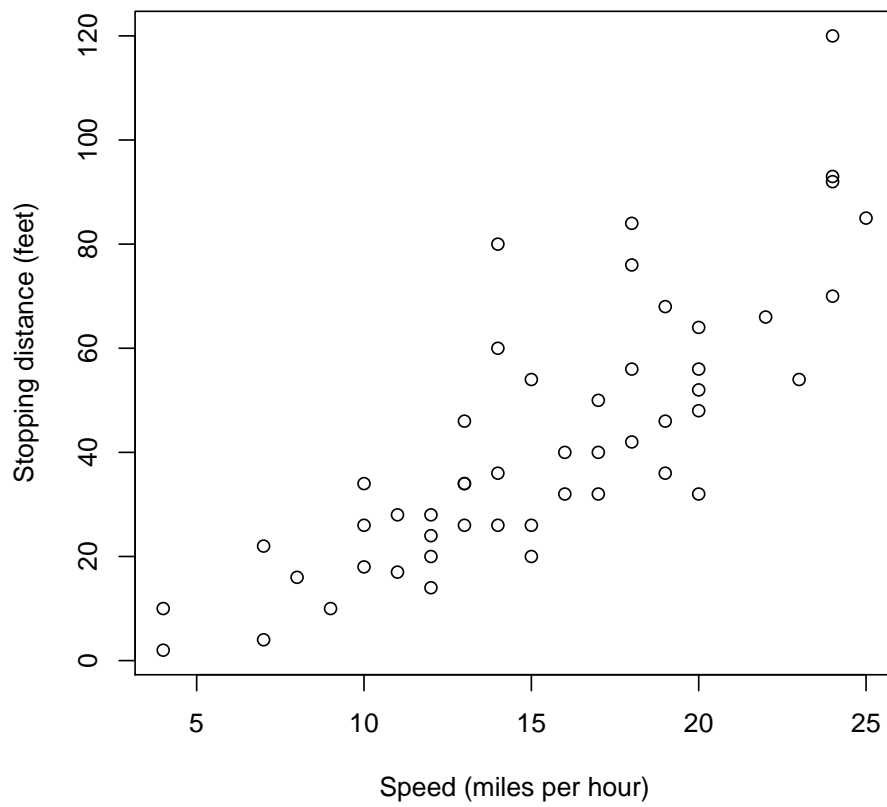


Figure 5: plot(cars) with whistle and bells

Speed and stopping distances for cars



Source: M. Ezekiel, Methods of Correlation Analysis. Wiley, 1930

2 Saving your plot

We now turn to how to save our plots. We do not just plot for fun, we want to use our plots in term papers, theses and reports. Maybe we want put the plots on a web page? Depending on our purpose with the plot we should choose different file formats. R is capable of saving plots in a variety of file formats.

First, which file format should one choose? The following answer, which can be regarded as a rule of thumb, not absolute, lists file formats by different softwares/usages:

Word for Windows Use the `bmp` file format. This is a bitmapped file format created via `bmp()`.

On the web Use the `jpg` file format. This is a compressed bitmapped file format created via `jpeg()`.

L^AT_EX Use Encapsulated Postscript. Created via `postscript()`.

PDF Use the `pdf` file format. This is file format created via `pdf()`.

2.1 `bmp()` and `jpeg()`

`bmp()` and `jpeg()` share several arguments and are therefore described together. Their syntax with the most important arguments set to default values are:

- `bmp(filename, width = 480, height = 480, units = "px", pointsize = 12)`
- `jpeg(filename, width = 480, height = 480, units = "px", pointsize = 12, quality = 75)`

`width` and `height` specify width and height of the figure given in the units as given by `units`. `pointsize` specifies that size of plotted text. In addition to these arguments `jpeg()` takes an argument `quality` which specify the compression rate in percent; the lower the rate, the smaller the resulting file.

We start by creating a `bmp` file with defaulting settings:

```
> bmp(file = "cars.bmp")
> plot(cars)
> dev.off()
```

Note that the first line of code opens the file, the second line puts some stuff into the file and the third line of code with the command `dev.off()` closes the file. Although we here are content with `plot(cars)`, the second line does not need to be restricted to this command only but should rather contain

all commands that are related to the design of the graph. This holds for the rest of the examples in section 2. The resulting file has size 226KB.

Then we create a `jpeg` file and we choose a low compression rate suitable for the web:

```
> jpeg(file = "cars.jpg", quality = 20)
> plot(cars)
> dev.off()
```

Note the extension `jpg`. The resulting file has size 11KB.

2.2 `postscript()`

For \LaTeX users it is common to use `\includegraphics{}` from the \LaTeX -package `graphics` to include the graphics in the document. This method basically require an Encapsulated Postscript file. The difference between Encapsulated Postscript and Postscript is that the former is a Postscript

“document with additional restrictions intended to make EPS files usable as a graphics file format. In other words, EPS files are more-or-less self-contained, reasonably predictable Postscript documents that describe an image or drawing, that can be placed within another Postscript document.”³

Basically this additional restriction take the form of a bounding box describing the rectangle which contains the actual graph.

The R-command `postscript()` produces Encapsulated Postscript files:

```
> postscript(file = "cars.ps")
> plot(cars)
> dev.off()
```

The graph produced has landscape orientation but can in most cases be used in \LaTeX if one also make appropriate use of the commands `\rotatebox{}` and `\scalebox{}`. The alternative would be to set options to the `postscript()` function to other than default values so that orientation becomes portrait etc. If the resulting graph shall be used in more than one way and one is comfortable with \LaTeX , using the \LaTeX -commands are probably a better method. Else, setting the options in R is probably better.

2.3 `pdf()`

R graphics can also be produced as PDF files. These files can then be merged into PDF documents using other software such as (e.g.) Adobe Acrobat, `pdf \LaTeX` `ghostscript` etc. Here is one brief example that produces the graph in an A4 paper size:

³<http://en.wikipedia.org>

```
> pdf(file = "cars.pdf", paper = "A4")
> plot(cars)
> dev.off()
```

3 Time series plots

Plot can also handle time series data (in the sense of R time series object created by `ts()`). We start by loading a multivariate time series data set:

```
> url <- "http://people.su.se/~ma/R_intro/DataWageMacro.rda"
> download.file(url, "DataWageMacro.rda")
> load("DataWageMacro.rda")
```

Note that the time index serves as x-coordinates when a time series object is plotted. Now the time series object `macro` is plotted using `plot()` (graph in Figure 6):

```
> plot(macro)
```

However, sometimes one want to have different time series in the same graph. This is handled by the function `ts.plot()` (graph in Figure 7):

```
> ts.plot(macro)
```

If the variables in the same time series object are to be plotted individually they have to be accessed via their column label or their column number (graph in Figure 8):

```
> ts.plot(macro[, "cci"])
```

produces the same result as (graph in Figure 9):

```
> ts.plot(macro[, 1])
```

All the arguments `main`, `sub`, `xlab`, `ylab` etc that we invoked using `plot()` can also be invoked using `ts.plot()`.

4 Graphical options: Adding whistles and bells

Although the above commands in section 2 also can be used to set different additional graphical arguments to produce various types of visual effects, it is often better to use other general methods to introduce these effects. One such method is `par()` via which a number of graphical parameters can be set. Some of these parameters can only be set inside `par()` and others can be set in different ways. However, here we only use `par()` to set them and only a selection of arguments are mentioned. See `?par` for details. Using `ts.plot()` these parameters can be specified using the argument `gpars=list()`.

Here only selection of possibilities and examples will be mentioned:

Figure 6: plot(macro)

macro

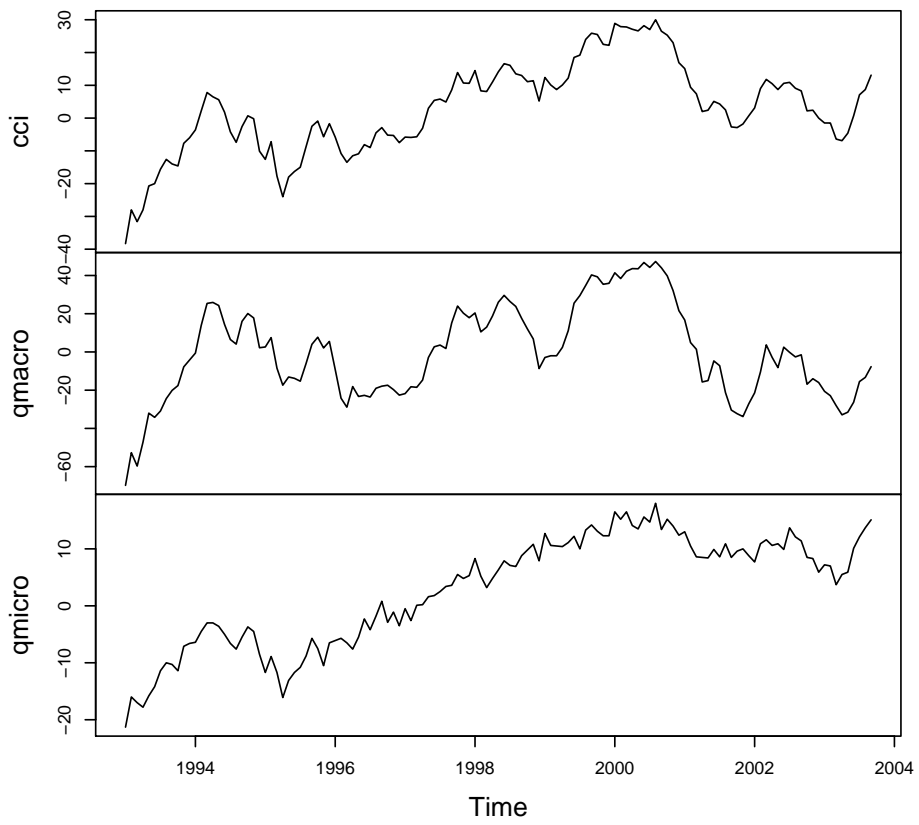


Figure 7: `ts.plot(macro)`

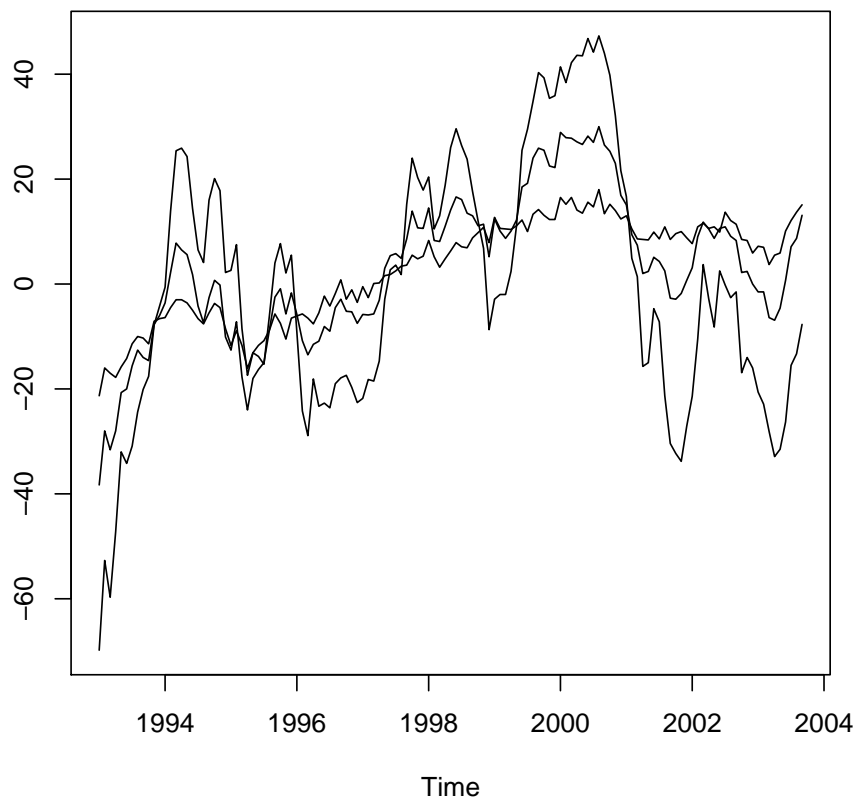


Figure 8: `ts.plot(macro[, "cci"])`

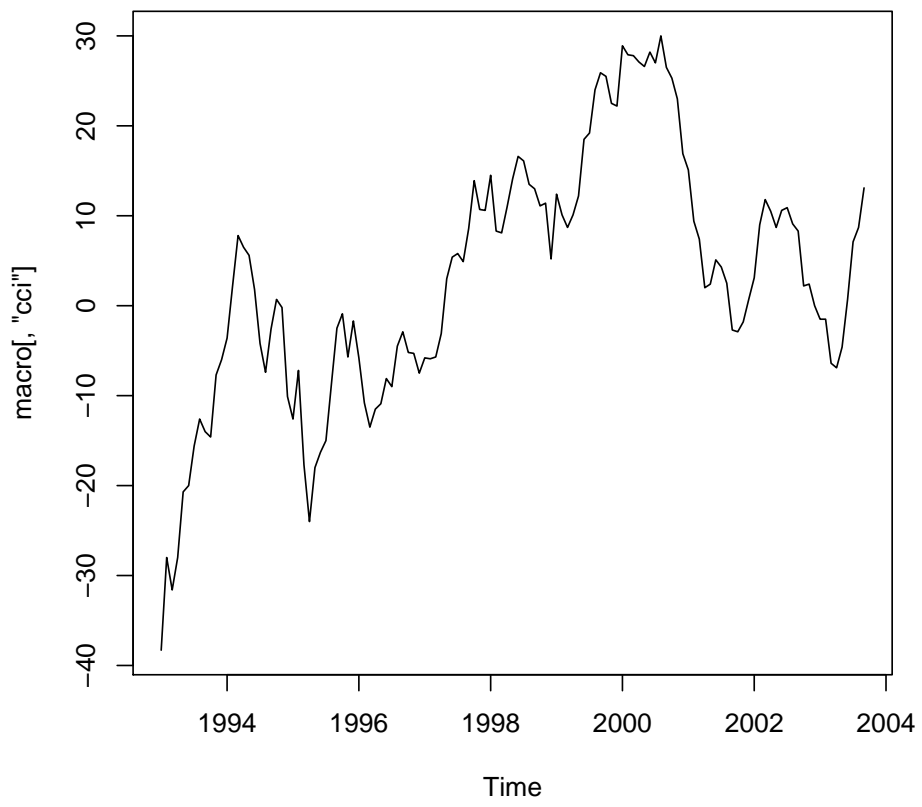
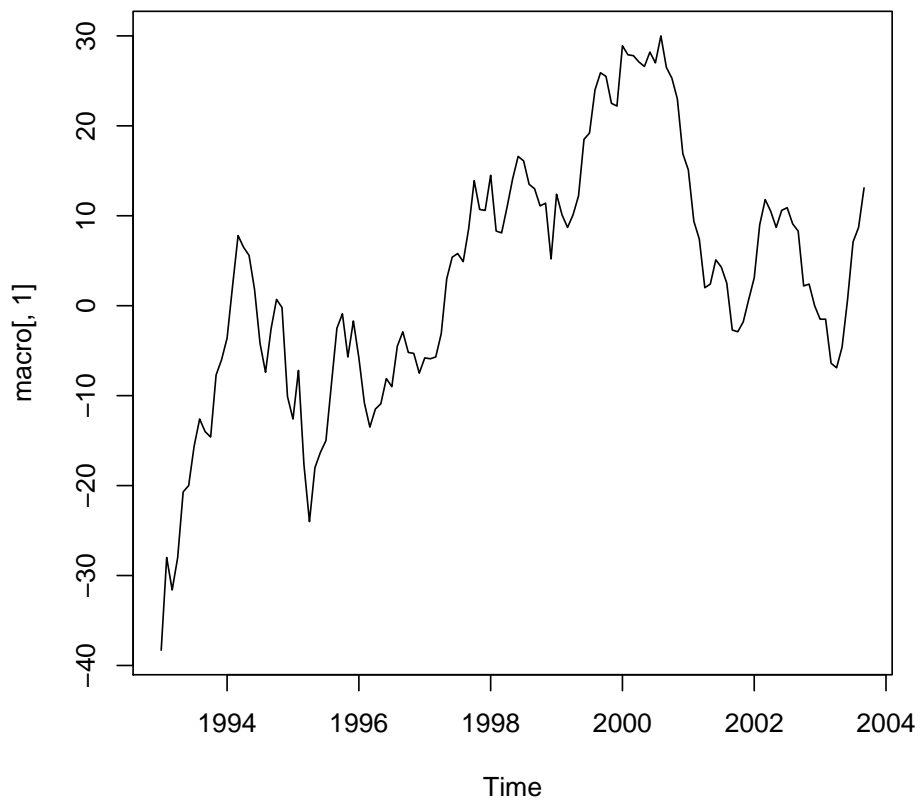


Figure 9: `ts.plot(macro[,1])`



1. line decorations, legends and colours
2. controlling the axis
3. fonts (choice of family and specific fonts)

4.1 Line decorations, legends and colours

In graphs where more than one data series are plotted is nice to be able to distinguish the the different series by giving them different looks (line type and color) as well as providing information which look corresponds to which series (a legend). Compare with Figure 7 where this information is absent. We introduce line type, line colour and legend step by step in the following by modifying Figure 7.

First we introduce different line types for the three different series. Line type is set by the argument `lty`. Best if the argument has the same number of types as there are data series. Line type can be set by either specifying integers (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or by setting character strings `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`.

Suppose we want the three series in `macro` to be solid dashed and two-dashed (graph in Figure 10):

```
> ts.plot(macro, gpars = list(lty = c(1, 2,
+   6)))
```

The line width is controlled with parameter `lwd=1` which has unity as default value; it can be set a positive number. The interpretation of the number varies between systems.

Frequently colour can increase the easiness by which a graph is read and interpreted. Line colours are added via the `col` argument. Note that the different colours can be accessed in different ways. The function `colors()` provides names of 600+ different colours that can be used with `col` (graph in Figure 11):

```
> ts.plot(macro, gpars = list(lty = c(1, 2,
+   6), col = c("black", "red", "blue")))
```

Still the person inspecting the graph do not know which line and which series that are related. We add a legend via `legend()`; see `?legend` for details (graph in Figure 12):

```
> ts.plot(macro, gpars = list(lty = c(1, 2,
+   6), col = c("black", "red", "blue")))
> legend("bottomright", c("CCI", "QMACRO", "QMICRO"),
+   lty = c(1, 2, 3), col = c("black", "red",
+   "blue"))
```

Figure 10: `ts.plot(macro)` with line types

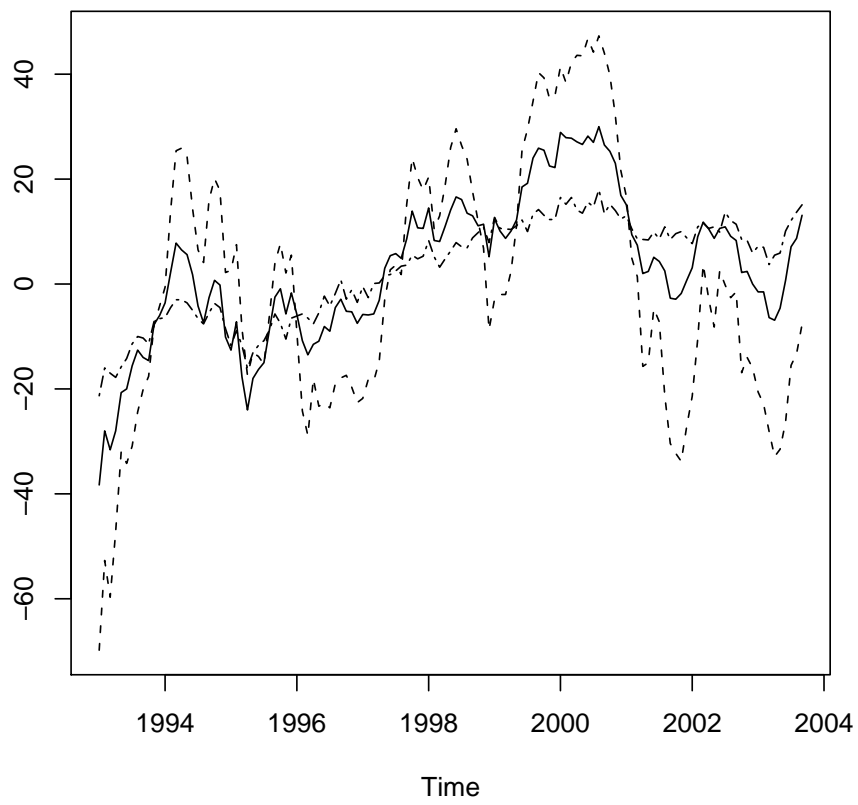


Figure 11: `ts.plot(macro)` with line types and line colours

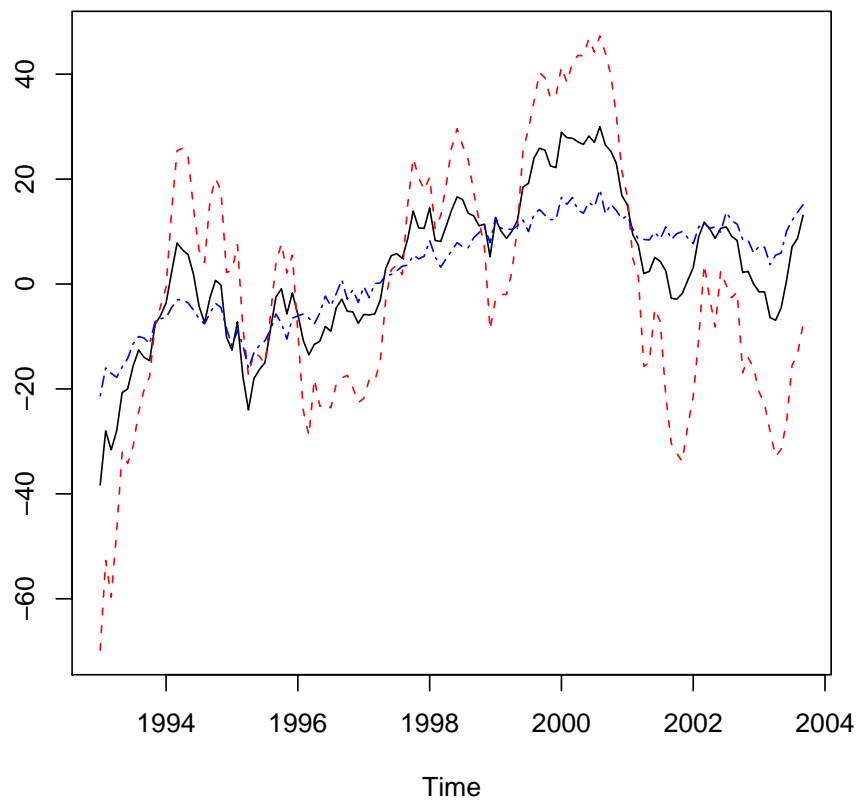
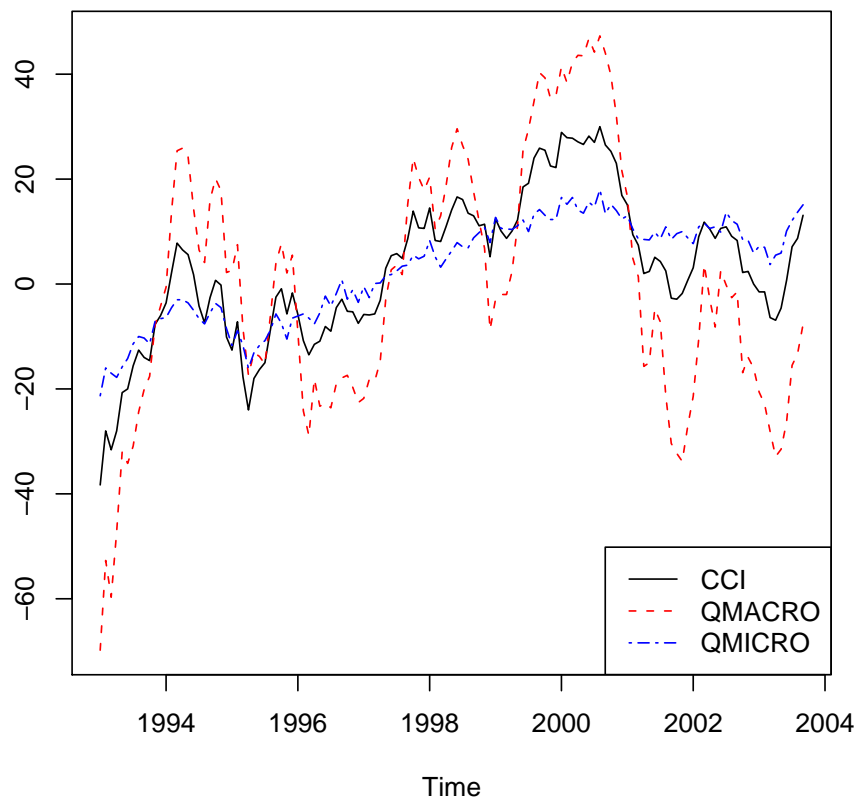


Figure 12: `ts.plot(macro)` with line types, line colours and legend



Finally, we add some background colour (graph in Figure 13)

```
> par(bg = "lightyellow")
> ts.plot(macro, gpars = list(lty = c(1, 2,
+   6), col = c("black", "red", "blue")))
> legend("bottomright", c("CCI", "QMACRO", "QMICRO"),
+   lty = c(1, 2, 3), col = c("black", "red",
+   "blue"))
```

4.2 Controlling the axis

Sometimes one want change the scale of the axis to something other than the default chosen by R. For instance, the variable CCI (from the `macro` data set) is a Consumer Confidence Indicator (as well as the other two indicators in the data set) that can take any value in $[-100, 100]$. This does not R know and instead chooses the max and min values on the y-axis so that all observations are appropriately plotted. By changing the scale one can affect how the data series is interpreted. For instance, by compressing the scale of the y-axis we can increase the impression of variability in the data. By setting the maximum and minimum values of the y-axis to 100 and -100 we get a clearer picture of how much the series varies within its bounds. This is done with the argument `ylim` which must be given the min and the max values (graph in Figure 14, which should be compared with Figure 8 or 9):

```
> par(bg = "lightyellow")
> ts.plot(macro, gpars = list(lty = c(1, 2,
+   6), col = c("black", "red", "blue"), ylim = c(-100,
+   100)))
```

A corresponding `xlim` argument controlling the x-axis also exists of course.

4.3 Fonts

There are many ways to control the choice of fonts, but some are specific to the system one is using. In most cases, however, `font` can be used to specify the font to be used troughout in the graph so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic. Clones are `fonts.axis` for the axis font, `fonts.lab` label font, `fonts.main` main title font and `fonts.sub` sub title font.

Here is an example which for the purpose of illustration (not beauty) shows some of these arguments in practice (see Figure 15):

```
> par(bg = "lightyellow", font = "2", font.main = "3",
+   font.lab = "4")
> ts.plot(macro, gpars = list(lty = c(1, 2,
```

Figure 13: `ts.plot(macro)` with line types, line colours , legend and back-ground colour

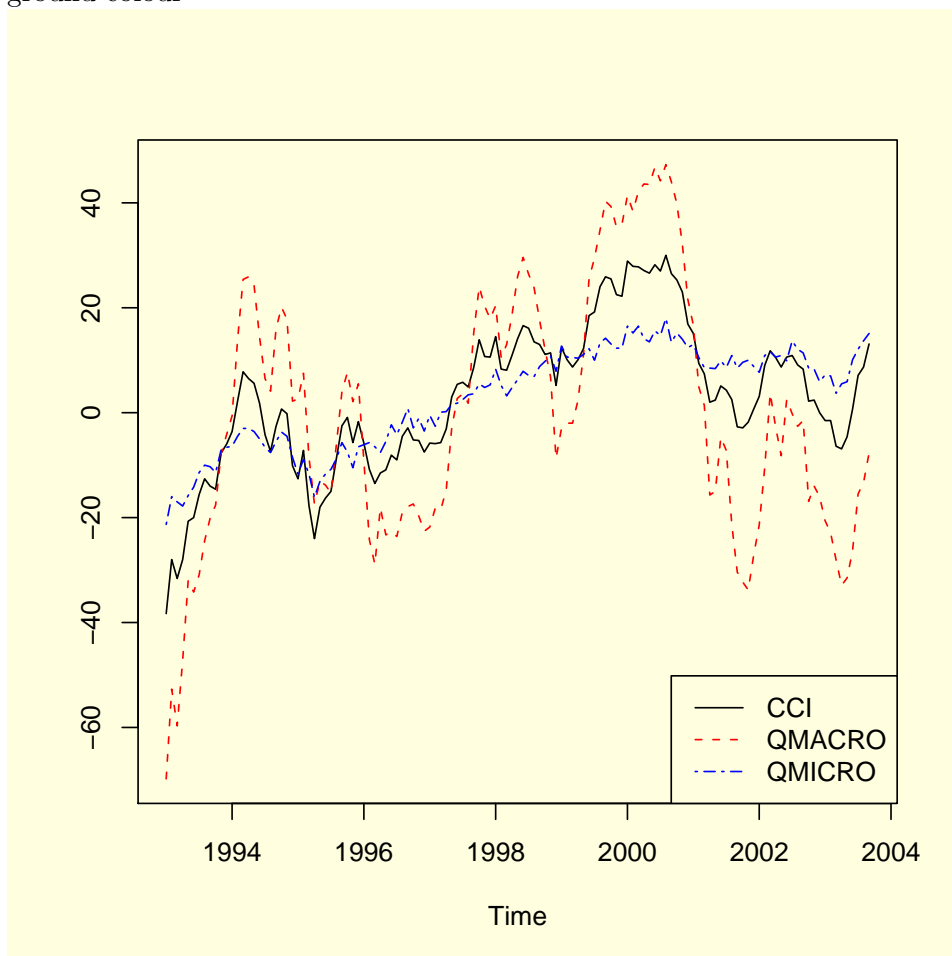
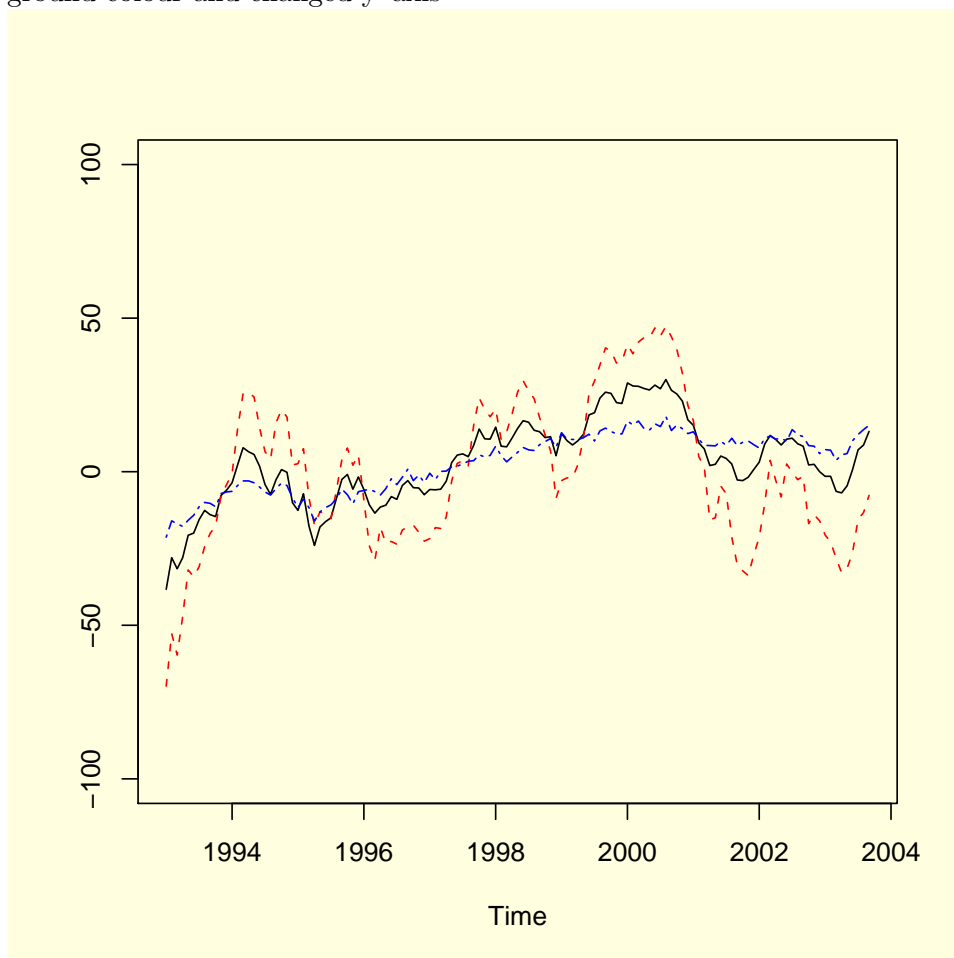


Figure 14: `ts.plot(macro[,1])` with line types, line colours, legend, background colour and changed y-axis



```
+     6), col = c("black", "red", "blue"), ylim = c(-100,
+     100), main = "Consumer Confidence Indicators",
+     ylab = "Indicator index"))
> legend("bottomright", c("CCI", "QMACRO", "QMICRO"),
+     lty = c(1, 2, 6), col = c("black", "red",
+     "blue"))
```

5 Histograms

Histograms can be used to plot the frequency of different observations. We use the `lnu` data set which is a part of `DataWageMacro.Rda` file (see Figure 16):

```
> attach(lnu)
```

To plot a histogram one use the function `plot()`. Some arguments are specified:

```
> par(bg = "lightyellow", font = "1", font.main = "1",
+     font.lab = "1")
> hist(wage, col = "yellow")
```

Figure 15: `ts.plot(macro)` with line types, line colours, legend, background colour, changed y-axis and changed fonts throughout

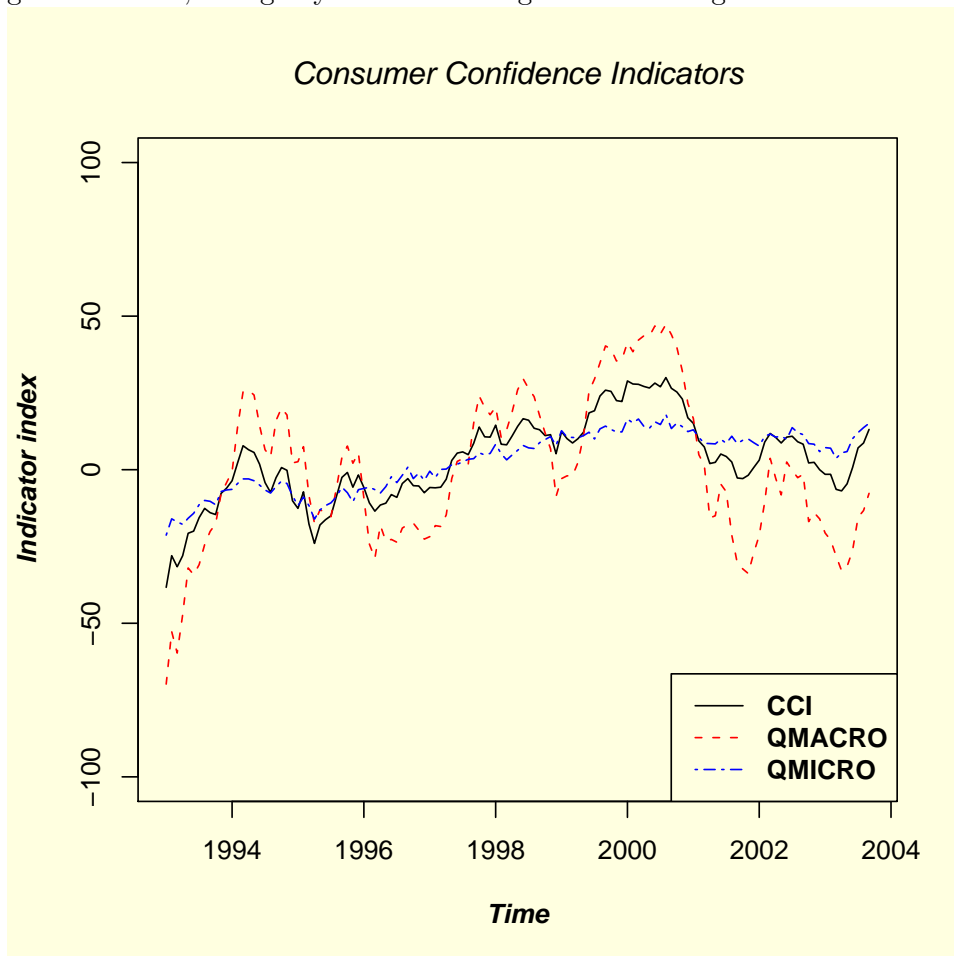


Figure 16: `hist(wage,col="yellow")`

