Laboratory Exercises

W. Venables, D. Smith & J. Pearce
Department of Statistics
The University of Adelaide

Contents

1 S-PLUS: An Introductory Session

The following session is intended to introduce to you some features of the S-PLUS environment by using them. Many features of the system will be unfamiliar and puzzling at first, but this will soon disappear.

`login:`
`...`
`> ls -a`
`> ls -a .Data`
	Login, start your windowing system (ask a demonstrator if you need help), and check that your working directory has a subdirectory `.Data`, which in turn contains the files `.First`, `.Last` and possibly `.Audit`.

	You should also have the file `morley.data` in your working directory. If not, see a demonstrator. If you have, proceed.
`> Splus -e`	Start `Splus` with the inbuilt command line editor enabled.
	The S-PLUS program begins, with a banner.
	(Within S-PLUS the prompt on the left hand side will not be shown to avoid confusion.)
`help.findsum(".Data")`	Set things up for the `help` facility. (Need only be done once for this directory.)
`help.start()`	Start the help facility. You should briefly explore the features of this facility with the mouse. Standard $X-$window conventions apply.
	Iconify the help window and move on to the next part.
`motif()`	Turn on the graphics window. You may need to re-position and re-size to make it convenient to work with both windows.

`x <- rnorm(50)`
`y <- rnorm(x)`
	Generate two standard normal vectors of $x-$ and $y-$coordinates.

`hull <- chull(x, y)`	Find their convex hull in the plane.

`plot(x, y)`
`polygon(x[hull], y[hull],`
`  dens=15)`
	Plot the points in the plane, and mark in their convex hull.

`objects()`	See which S-PLUS objects are now in the `.Data` directory.
`rm(x,y,hull)`	Remove objects no longer needed. (cleanup).
`x <- 1:20`	Make $x = (1, 2, \ldots, 20)$

```
w <- 1 + sqrt(x)/2
```

A ‘weight’ vector of standard deviations.

```
dummy <- data.frame(x=x,
  y= x + rnorm(x)*w)
dummy
```

Make a *data frame* of two columns, x and y, and look at it.

```
fm <- lm(y~x, data=dummy)
summary(fm)
```

Fit a simple linear regression of y on x and look at the analysis.

```
fm1 <- lm(y~x, data=dummy,
  weight=1/w^2)
summary(fm1)
```

Since we know the standard deviations, we can do a weighted regression.

```
lrf <- loess(y~x, dummy)
```

Make a nonparametric local regression function.

```
attach(dummy)
```

Make the columns in the data frame visible as variables.

```
plot(x, y)
```

Standard point plot.

```
lines(x, fitted(lrf))
```

Add in the local regression.

```
abline(0, 1, lty=3)
```

The true regression line: (intercept 0, slope 1).

```
abline(coef(fm))
```

Unweighted regression line.

```
abline(coef(fm1),lty=4)
```

Weighted regression line.

At any time you can make a hardcopy of the graphics window by clicking on the `Graph` section of the window and selecting the `Print` option.

```
detach()
```

Remove data frame from the search list.

```
plot(fitted(fm),
  resid(fm),
  xlab="Fitted values",
  ylab="Residuals", main=
  "Residuals vs Fitted")
```

A standard regression diagnostic plot to check for heteroscedasticity. Can you see it?

```
qqnorm(resid(fm), main=
  "Residuals Rankit Plot")
```

A normal scores plot to check for skewness, kurtosis and outliers. (Not very useful here.)

```
rm(fm,fm1,lrf,x,dummy)
```

Clean up again.

The next section will look at data from the classical experiment of Michaelson and Morley to measure the speed of light.

```
!more morley.data
```

Optional. Temporarily interrupt S-PLUS and look at the file. This is a standard way to escape to the operating system.

```
mm <-
read.table("morley.data")
mm
```

Read in the MM data as a data frame, and look at it. There are five experiments (col. `Expt`) and each has 20 runs (col. `Run`) and `sl` is the recorded speed of light, suitably coded.

`attach(mm, 1)`
`objects()`
	Place `mm` on the top of the search list, (position 1).

`Expt <- factor(Expt)`
`Run <- factor(Run)`
	Change `Expt` and `Run` into factors.

`detach(1, save="mm")`
`attach(mm)`
	Save the changes and make the data frame visible at position 2 (the default).

`plot(Expt,Speed, main=`
`"Michaelson Morley Data",`
`xlab="Experiment No.")`
	Compare the five experiments with simple boxplots.

`fm <- aov(Speed~Run+Expt,`
`  data=mm)`
`summary(fm)`
	Analyse as a randomized block, with 'runs' and 'experiments' as factors.

`fm0 <- update(fm,`
`  .~.-Run)`
`anova(fm0,fm)`
	Fit the sub-model omitting 'runs', and compare using a formal analysis of variance.

`detach()`
`rm(fm, fm0)`
	Cleanup before moving on.

	We now look at some more graphical features: contour and 3−dimensional perspective plots.

`x <- seq(-pi, pi, len=50)`
`y <- x`
	x is a vector of 50 equally spaced values in $-\pi \leq x \leq \pi$. y is the same.

`f <- outer(x, y,`
`  function(x,y)`
`    cos(y)/(1+x^2))`
	f is a square matrix, with rows and columns indexed by x and y respectively, of values of the function $\cos(y)/(1+x^2)$.

`oldpar <- par()`
`par(pty="s")`
	Save the plotting parameters and set the plotting region to "square".

`contour(x, y, f)`
`contour(x, y, f,`
`  nint=15, add=T)`
	Make a contour map of f; add in more lines for more detail.

`fa <- (f-t(f))/2`	`fa` is the "asymmetric part" of f. (`t()` is transpose).
`contour(x, y, fa, nint=15)`	Make a contour, and
`par(oldpar)`	restore the old graphics parameters.

`persp(x, y, f)`
`persp(x, y, fa)`
`image(x, y, f)`
`image(x, y, fa)`
	Make some pretty perspective and high density image plots, (of which you can get hardcopies if you wish)

`objects(); rm(x,y,f,fa)`	and clean up before moving on.

`th <- seq(-pi, pi,`
`  len=100)`
`z <- exp(1i*th)`
	S-PLUS can do complex arithmetic, also. `1i` is used for the complex number i

`par(pty="s")`
`plot(z, type="l")`
	Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

`w <- rnorm(100) +`
`    rnorm(100)*1i`
	Suppose we want to sample points within the unit circle. One method would be to take complex numbers with standard normal real and imaginary parts –

`w <- ifelse(Mod(w) > 1,`
`  1/w, w)`
	and to map any outside the circle onto their reciprocal.

`plot(w, xlim=c(-1,1),`
`  ylim=c(-1,1), pch="+",`
`  xlab="x", ylab="y")`
`lines(z)`
	All points are inside the unit circle, but the distribution is not uniform.

`w <- sqrt(runif(100))*`
`  exp(2*pi*runif(100)*1i)`
`plot(w, xlim=c(-1,1),`
`ylim=c(-1,1), pch="+",`
`xlab="x", ylab="y")`
`lines(z)`
	The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

`rm(th,w,z)`	Cleanup again.

`par(oldpar)`	Restore standard graphics parameters.
`butterfly()`	An old favourite. Take a hardcopy if you wish.
`rm(oldpar)`	Cleanup again.
`objects()`	See which objects are left in the `.Data` directory.
`q()`	Quit the Splus program.
`>`	and return to `Unix`.

2 Vectors, sorting, calculations and simple plots.

In this exercise you will be expected to do some of the calculations for yourself. Use the help facilities as much as you need, but ask a tutor if you are really stuck.

After the pyrotechnics of the introductory session, the pace may now seem a little slow. This is quite normal.

2.1 Exegesis on the Michaelson Morley data

1. Read the help file on the Michaelson Morley data `?morley` and attach the data at position 1 of the search list

   ```
   > attach(morley, 1); search()
   ```

2. Convert `Expt` and `Run` to *factors* and look at their `levels`

   ```
   > Expt <- factor(Expt); levels(Expt) ...
   ```

3. Re-do the one-way analysis of variance of `Speed` on `Expt` and check (once again) for differences between experiments. Let the fitted model object be `fm`, say.

4. Open a graphics window

   ```
   > motif()
   ```

   and produce a normal *qq*plot of the residuals and superimpose a "robust" line on it to check visually for linearity. This is done as follows (explanations will follow later)

   ```
   > r <- resid(fm); a <- qqnorm(r); abline(rreg(a$x, a$y))
   ```

   Do there seem to be any outlying residuals?

   [Optional] Find out where the original observations corresponding to the two most negative residuals are. A graphical way of doing this is as follows:

   ```
   > plot(resid(fm))                          # plots resids vs 1,2,...,100
   > identify(resid(fm))                  # interactive graphics with mice.
   ```

   At this point move the cursor into the graphics window with the cursor and click on the two most negative points with the left button. Their numbers should appear on the plot. To finish, click with the middle button.

5. As an alternative to a *qq*plot, we can look at the empirical distribution function itself and superimpose a normal distribution with the same mean and variance function.

   ```
   > rs <- sort(r); edf <- (1:length(r))/length(r)
   > sd <- sqrt(var(r)); cdf <- pnorm(rs,0,sd)
   ```

```
> plot(stepfun(rs,edf), type="l"); lines(rs,cdf)
```

6. Calculate the Kolmogoroff-Smirnoff statistic, which is defined as the maximum absolute difference between `edf` and `cdf` defined above.
7. Look to see what new objects have been created (`objects()`) and detach the data frame, either saving or discarding the changes to the data frame, as you wish:

```
> detach(1)                        # deletes all changes and new objects.
> detach(1, save="morley")          # saves changes and some new objects.
```

2.2 The Beaujolais wine quality data

Attach the Beaujolais data frame and make dotcharts of each of the sorted variables. Which wine was actually vinegar?

```
> s <- sort.list(TSO); dotchart(TSO[s], row.names(beaujolais)[s])
```

2.3 The Jellyfish data

Read the help file for the jellyfish data and do the following

1. Attach the data frame at position 2, and look at the levels of the factor `Sample`.
2. Plot length against width, marking in the two samples with a different character.

   One way to do this is as follows

```
> plot(Width, Length, type="n")                    # no plot; axes only
> points(Width[Sample=="D"], Length[Sample=="D"], pch="*")
> points(Width[Sample=="S"], Length[Sample=="S"], pch="+")
```

   another would use `text()` in place of the last two commands:

```
> text(Width, Length, Sample)
```

   Try both.

3. [Optional] Piece together from the introductory session and the above how you would put polygons on the diagram showing the convex hull of each sample.
4. Find the means of each sample, and print them.

```
> Wmean <- c(mean(Width[Sample=="D"]), Width[Sample=="S"])); Wmean ...
```

5. Look up the `symbols()` command and use it to add $\frac{1}{4}$inch circles to your plot at the two sample mean points.

```
> symbols(Wmean, Lmean, circles=c(1,1), inches=0.125, add=T)
```

3 Arrays, matrices and frequency calculations

This is largely a session aimed at making you familiar with array and matrix calculations. Statistically it is rather dull and graphically void, but it is the sort of manipulation that comes up very frequently and something with which you need to be comfortable and familiar if you are to use S-PLUS with ease.

3.1 The Cloud data "by hand"

Our first exercise will be to do some polynomial regression calculations "from scratch" using raw matrix manipulation, and to verify the results using S-PLUS tools. (A side effect is to reinforce the value of learning to use the tools effectively).

1. Look up the Cloud data either in the notes or using the help facility

   ```
   > help(cloud)
   ```

2. Attach the data and for ease of typing make two new variables, `x` and `y` equal to the $I_s\%$ and cloud point vectors respectively. Let `n` be the number of observations, for future reference.

   ```
   > attach(cloud); x <- Ispc; y <- Cloudpt; n <- length(x)
   ```

3. Mean correct the `x` vector and construct a model matrix for a cubic regression of y on x, (where the powers are of the mean corrected $x-$vector). Include the constant term, of course.

   ```
   > x <- x-mean(x); X <- cbind(1, x, x^2, x^3); X
   ```

4. The regression coefficients are $\boldsymbol{b} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}$. First calculate the matrices

   ```
   > XX <- t(X) %*% X                              # the literalist way
   > XX <- crossprod(X)                    # more efficient alternative
   > Xy <- crossprod(X, y)                               # 2 argument form
   ```

   Now calculate the regression coefficients in `b` using `solve()`.

5. Calculate

   - the fitted values, $\boldsymbol{f} = \boldsymbol{X}\boldsymbol{b}$,
   - the residuals, $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{f}$,
   - the residual mean square $s^2 = \boldsymbol{r}'\boldsymbol{r}/(n-4)$ and
   - the variance matrix of the regression coefficients, $s^2(\boldsymbol{X}'\boldsymbol{X})^{-1}$, which you call, say, `Vb`.

6. The square roots of the diagonal entries of `Vb` give the standard errors, and the ratio of the regression coefficients to these give the t−statistics.

   ```
   > se <- sqrt(diag(Vb)); ts <- b/se
   ```

7. Finally arrange the results in a "comprehensible" form and print them out:

   ```
   > R <- cbind(b, se, ts, 1-pt(ts,n-3))
   > rnames <- c("Constant", "Linear", "Quadratic", "Cubic")
   > cnames <- c("Coef", "Std.  Err.", "t-stat", "Tail prob.")
   > dimnames(R) <- list(rnames, cnames); R
   ```

8. Check the calculations by the standard tools

   ```
   > fm <- lm(y~1+x+I(x^2)+I(x^3)); summary(fm)        # check the table.
   > b-coef(fm)                             # check for numerical discrepancies.
   ```

9. Detach data frame and clean up.

3.2 Frequency tables from the Tuggeranong house data

1. Attach the Tuggeranong house price data and look at the variables.
2. Make a factor from the `Rooms` variable directly (3 classes) and from the `Age` variable by cutting the range into three equal parts. Rename the `Cent.Heat` factor for convenience.

   ```
   > rms <- factor(Rooms); age <- factor(cut(Age,3)); cht <- Cent.Heat
   ```

3. Find two-way frequency tables for each pair of factors. Although the table is of only a small number of frequencies, does their appear to be any two way association?
4. Use the Pearson χ^2 statistic, as discussed in detail in lectures, to check for association in any of the three tables you care to choose.
5. Check your answer with `chisq.test(tabl)` and `fisher.test(tabl)`, where the argument is your frequency table.

3.3 2×2 matrix determinant example

Finally you may care to work through the example in loop avoidance using the `outer()` function. A hardcopy of the final graph may serve as a future logo for the course.

4 Lists and data frames

4.1 The Iris data

The Iris data is a classical data set of R. A. Fisher giving 4 flower measurements on 3 samples each of 50 Iris flowers, one sample for each of 3 different species. The data is available as a system dataset in the form of a 3–way array. In lectures we saw one way to convert this to a data frame in such a way as to make maximum use of the original labels.

1. Make a data frame with appropriate variable labels from the iris data.

   The procedure given in lectures is carefully designed to cater for any number of equal sized groups. However since there are only three groups in the Iris data, most of the work can be done with

   ```
   > iris <- rbind(iris[,,1], iris[,,2], iris[,,3])
   ```

2. Plot each variable against each other, using a different symbol for each species. To what extent may the species be separated on the basis of either individual measurements, or pairs, alone?

3. [Optional] The help file for the function `discr()`, for discriminant analysis, gives a final example on how to explore the iris data using multiple discriminant analysis and other graphical techniques such as `brush()`. Work through this example.

4.2 The Longley data

This is a famous data set used in a benchmark study of least squares software accuracy. It comes as a system data set, and `?longley` will give more information. The data is not as a data frame but as an $X-$matrix, called `longley.x`, and a $y-$vector, `longley.y`.

1. Form a data frame from `longley.x` and `longley.y`

2. Find the correlation matrix between all variables. (`cor()`.)

3. Find the singular value decomposition of the mean corrected $X-$matrix and print out the singular values. (`svd()`.)

   The simplest, (if computationally rather extravagant), way of mean correcting a matrix is to use the model fitting functions:

   ```
   > Xc <- resid(lm(longley.x~1))
   ```

   A different way, that keeps the mean vector as well, is as follows

   ```
   > xb <- apply(longley.x, 2, mean); Xc <- t(t(longley.x) - xb)
   ```

Convince yourself why both work and verify that they lead to the same result.

The ratio of the largest to the smallest singular value is called the *condition number* of the matrix. Calculate it and print it out.

4. [More advanced.] The U components of the singular value decomposition are called the *principal component scores* for the $x-$variables. They form an alternative orthonormal basis for the regression model subspace. A commonly suggested technique for cautious least squares estimation when there are severe near collinearities in the data, as here, is

   - Regress y on the principal component scores, U,
   - Replace nonsignificant coefficients by 0,
   - Transform to original coordinates by $\boldsymbol{\beta} = VD^{-1}\boldsymbol{\gamma}$, where $\boldsymbol{\gamma}$ is the principal component coefficient vector, (including zero components).

   Investigate the principal component regression technique in the context of this example.

5. [For later] Build up a prediction equation for y from the $X-$variables. Either start from a full model and eliminate at each stage the least significant term, or start with the grand mean model and add at each stage the variable that most reduces the residual sum of squares. (see `add1()`, `drop1()`, `update()`)

4.3 `scan()` for reading data

1. Using an editor with which you are familiar, e.g. `vi`, remove the first (header) line from the file `morley.data`.
2. Read in the data using the `scan()` function as a list of four components, the first three of which (row names, experiment and run) are of type `character`, and the last (speed of light) is numeric.
3. Exhibit the list you have read on the terminal, and construct anew the `morley` data frame from it.

5 Statistical models in S-PLUS

Many of the features to be exemplified in this laboratory exercise have been introduced ahead of time in earlier sessions. You should review earlier work and make sure that now you have a clearer idea of what was happening in those earlier examples.

5.1 The painters data

Read the description of the `painters` data either from the notes or using `?painters` within S-PLUS.

1. To see how the schools differ on each of the characteristics, plot the data frame using the method appropriate to a design:

   ```
   > plot.design(painters)
   ```

   This will give a series of four plots that should be understandable.

2. Perform a single classification analysis of variance on each variable on school.

3. [Optional] Find the matrix of residuals for the four variables and hence the "within school" variance and correlation matrices.

5.2 The genotype data

1. Look graphically at the genotype data using the method outlined in lectures.

   ```
   > plot.design(genotype)

   > attach(genotype)

   > interaction.plot(Mother, Litter, Wt)

   > interaction.plot(Litter, Mother, Wt)
   ```

2. Verify that fitting the two-way non-additive model in `Wt~Litter*Mother` and `Wt~Mother*Litter` leads to different sequential ANOVA tables, using `anova()`.

3. Compare the sequential tables with the table got by `drop1()` on the outer model.

   ```
   > drop1(fm, Wt~.)
   ```

5.3 The oil data and partitioning sums of squares.

Read carefully the description of the oil data, either from the notes, or by using `?oil` at the terminal.

1. Read in the oil data as a data frame using `read.table()` with the file `oil.data`, and if necessary, ensure that the variable `Sample` is a factor.
2. Fit separate regressions of `Yield` on `Endpt` within each level of `Sample`. Show that the submodel with parallel regressions of `Yield` on `Endpt` may be retained.

   ```
   > fm <- lm(Yield~Sample/Endpt, oil)           # Separate regressions
   > fm0 <- lm(Yield~Sample+Endpt, oil)          # Parallel regressions
   > anova(fm0,fm)                                 # test fm0 within fm
   ```

3. Show that the common regression coefficient is significant, and hence should be retained. (`summary(fm0)`
4. Show that the differences between intercepts in the parallel regressions, and hence the differences between samples, is very significant.
5. Can the differences between samples be accounted for by a multiple regression on `API`, `Vapour` and `ASTM`?

   ```
   > update(fm0, .~.-Sample+API+Vapour+ASTM, oil) -> fm0a
   > anova(fm0a, fm0)
   ```

5.4 The sugarbeet data

This data set of R. A. Fisher presents a randomized block experiment with a 3×2^3 factorial structure in the treatments. The yield is the weight of roots for each plot, but to allow for different harvesting rates between plots, the number of roots is to be included as a covariate. The presence of the covariate technically makes the design nonorthogonal.

1. Regarding blocks as a fixed effect factor, but without interactions with the treatment factors, fit a complete factorial model for weight.

   ```
   > fm <- aov(Wt~No+Block+V*N*P*K, sugar)
   ```

2. To get a nonsequential Analysis of Variance table (like *Genstat*) use the command

   ```
   > drop1(fm, Wt~.)
   ```

   Hence or otherwise, prune sensibly to arrive at a well supported model for the data.
3. Perform the usual residual analyses to check for departures from the underlying assumptions.

6 Further Examples

This final laboratory session for the first course could be as open as you like. Use the opportunity to make sure of any feature about which you are still unclear. You may also care to work through some of the examples given in lectures to see that they do work out as stated, for example the artificial Analysis of Covariance example. The following problems are suggestions only.

6.1 The wheat yield data

The Iowa wheat yield data shows how the yield of wheat gradually increased over the period 1930 to 1962, due largely to improvements in cultivation practices and wheat varieties. The year itself is a good surrogate for these extraneous influences, and is assessing the influence of other variables, such as average temperatures and precipitation throughout the growing and harvesting season, the year itself has to be included in any contemplated model.

1. Build an interpretative regression model for yield on the other variables. Either start from the minimal model and add variables as needed, or begin with the full model and drop them.

   ```
   > fm <- lm(Yield~.  , iowheat)                    # maximum model
   > drop1(fm)                                           # backwards
   > fm0 <- lm(Yield~1, iowheat)                      # minimum model
   > add1(fm0, fm)                                        # forwards
   ```

   In the backwards direction at each stage omit that variable that least increases the residual sum of squares, and in the forwards direction include that variable that most reduces it. Stop at some appropriate stage.

   Do both methods lead essentially to the same variables as being important?

2. When you have a tentative final model, plot the residuals against the *squares* of the mean corrected determining variables. Consider additional second degree terms, and comment.

3. The South Australian wheat data, `sawheat`, comes from essentially the same period, but is radically different it its characteristics. To the extent it is possible, build a similar interpretative model for it, and comment.

6.2 The Tuggeranong house price data

1. Using `coplot()` look at the relationship between `Price` and `Age` with and without central heating.

   ```
   > coplot(Price~Age|Cent.Heat, house)
   ```

Look further by conditioning on both central heating and the number of rooms:

```
> coplot(Price~Age|Cent.Heat*Rooms, house)
```

2. Build a prediction equation for house price from the variables given.

6.3 The Janka data

One very clear feature of the Janka data is that in the original scale there is a marked dependence of the variance on the mean. This can be shown by fitting, say, a quadratic polynomial regression and plotting residuals against fitted values.

Consider a series of fractional power transforms of the hardness variable and their regression on density.

The simplest way to do this is to *write a function* to do most of the work, including a residuals display, and to call it with varying values of the exponent (power) as its argument.

```
> disp.resid <- function(power)
  {
      if(power==0)
      {
          ...
      } else
      {
          ...
      }
      ...
      invisible(NULL)
  }
```

Use the convention that if the exponent is given as 0, the log transform is used.

Since the function is mainly to produce a plot after doing a calculation, it may be given an empty value NULL and making it invisible(NULL) will suppress printing if the function is called as an expression.

7 Elementry Graphics

This is an introductory exercise intended to allow you to experiment with the graphical capabilities of S. To get started, log in to your workstation and start up the X windowing system as usual. Enter the S environment with the `Splus` command. Once S is ready for input, type the command

```
> motif()
```

to create a graphics window. You may also wish to create a help window with the command

```
> help.start()
```

7.1 Labelling and axes: The Janka data

The object `janka` is a data frame containing measurements of Janka hardness (`Hard`) for various densities (`Dens`). Look at the help file for this data frame with the command

```
> ?janka
```

Attach the dataset for later use with the command

```
> attach(janka)
```

We can produce a scatterplot of the two quantities with

```
> plot(Dens, Hard)
```

but the axis labels aren't particularly meaningful. Draw the plot again, but this time using the `xlab=` and `ylab=` arguments to `plot()` to generate more informative axis labels.

The relationship between density and Janka hardness seems reasonably linear ... or does it? We will investigate fitting a straight-line model using regression. Fit the least-squares line to the data with

```
> janka.lm <- lm(hard ~ dens, data=janka)
```

and add the fitted line to the scatterplot using

```
> abline(janka.lm)
```

Give the plot a title using the `title()` function.

We can plot the residuals against the fitted values with the command

```
> plot(fitted(janka.lm),resid(janka.lm))
```

What do you see? What do you think about this regression? Would you accept the straight-line model as adequately explaining the relationship?

7.2 The map of Australia function

Produce a map of of Australia with

```
> oz()
```

Now add the borders of your home state in a different colour, using one of `wa()`, `sa()`, `nt()`, `tas()`, `vic()`, `nsw()` or `qld()` (if you are from overseas or the ACT, choose any one,) for example

```
> sa(add=T,col=2)
```

The object `oz.cities` is a data frame containing the latitudes and longitudes of a number of Australian cities. Mark the position of the capital city of your state with an 'x', (`points(...,pch=...)`) and label it with the name of the city (`text()`). Can you think of a simple way of doing this for *all* cities?

7.3 Student survey data

The data frame `survey` contains the results of a survey of 237 first-year Statistics students at Adelaide University. Attach the data frame for later use, and have a look at the help file for this data frame. For a graphical summary of all the variables, type

```
> plot(survey)
```

Note that this produces a dotchart for factor variables, and a normal scores plot for the numeric variables.

(a) One component of this data frame, `Exer`, is a factor object containing the responses to a question asking how often the students exercised. Produce a barplot of these responses using `plot()`.

The `table()` function when applied to a factor object returns the frequencies of each level of the factor. Use this to create a pie chart of the responses with the `pie()` function. Do you like this better than the bar plot? Which is more informative? Which gives a better picture of exercise habits of students? The `pie()` function takes an argument `names=` which can be used to put labels on each pie slice. Redraw the pie chart with labels (**Hint:** use the `levels()` function to generate the labels.) You could also add a legend to identify the slices; the `locator()` command will help you decide where to put the legend. The command to do this will be something like

```
> legend(locator(1),labels,fill=1:3)
```

— click with the left mouse mutton where you would like the legend to appear. Note that you will have to assign `labels` yourself.

You might like to try the same things with the `Smoke` variable, which records responses to the question "How often do you smoke?" Note that `table()` and `levels()` ignore missing values; if you wish to include non-respondents in your chart use `summary()` to generate the values, and `names()` on the summary object to generate the labels.

(b) Is there a relationship between pulse rate and exercising? Create boxplots of `Pulse` for each level of `Exer` with the command

```
> plot(Exer,Pulse)
```

Does there appear to be a relationship? You may wish to test this formally with the `aov()` command. What other relationships can you find in these data?

7.4 The Swiss banknote data

In an effort to detect counterfeits, the dimensions of 100 known fake Swiss banknotes and 100 supposedly legal notes were measured. The length of the note and its diagonal were measured, along with the length of each side. The data are in the `swiss` data frame.

```
> attach(swiss)
```

Plot the variables against each other in a scatterplot matrix:

```
> pairs(swiss)
```

Which two variables seem to best discriminate the counterfeit and legal notes? Generate a coplot of these two variables, given `Fake`. It is possible that some of the "legal" notes are in fact undetected counterfeits — does your coplot suggest this?

7.5 Rainfall in Adelaide and New York

The `rain` object is a time series matrix containing rainfall data for Adelaide, SA and New York, USA. Plot the three time series on the same graph using

```
> tsplot(rain)
```

What happens if you try to create this plot "by hand" using `plot()` and `lines()`?

Produce a histogram of the Adelaide rainfall data with the command

```
> hist(rain[,3])
```

With only six classes in the histogram it's rather difficult to get an idea of the distribution of rainfall. Use the argument `nclass=10` in the above `hist()` command to produce a histogram with more (but note, not exactly 10) classes. Has your picture of the (empirical) distribution changed?

8 Advanced graphics

In this exercise we will often be changing the values of graphics parameters. After you have started the window driver, save the initial values of the grahics parameters with

```
> oldpar <- par()
```

At the end of each section, restore the graphics parameters to their default values with

```
> par(oldpar)
```

This way the graphics system will be in the same, known state for each question.

8.1 Passing graphics parameters

We are going to produce a plot for short-sighted people using characters twice as large as usual. Because of the larger characters, more space is needed for the axis labels, and we reserve it using a call to `par()`:

```
> par(mar=rep(5,4)) #5 lines on each side
```

Using the `chlorine` data, plot `Chlor` against `Time`. Pass graphics parameters to `plot()` so that text appears in twice its usual size (`cex=2`) and in a different colour (`col=`) and using thicker lines for the axes (`lwd=`.)

8.2 Multiple Figures

The jellyfish data (dataframe `jellyfish`) are measurements of width and length for jellyfish taken from two different areas.

Create side-by-side (`par(mfrow=...)`) plots of `Length` against `Width` for the two groups, suitably labelled and titled.

Make sure that axes are the same on both graphs for easy comparison (`xlim= / ylim=` and/or `par(xaxs="d", yaxs="d")`).

Place a suitable title on the page (`par(oma=...)`, `mtext(...,outer=T)`).

8.3 Axes through the origin

Plot x against $\sin(x)$, using 200 values of x between $-\pi$ and π, but do not plot any axes yet (use `axes=F` in the call to `plot()`.)

Add a y-axis passing through the origin using the "extended" style:

```
> axis(2, pos=0, yaxs="e")
```

Add an x-axis with tick-marks from $-\pi$ to π in increments of $\frac{\pi}{4}$ (`axis(..., at= )`) and with labels like `"-1/2 Pi"` (`labels=c("-Pi",...)`.)

8.4 Multiple axes

The Iowa Wheat data (dataframe `iowheat`) gives `Yield` values for a number of years along with temperature and rainfall data.

We wish to look at how yield varies with rainfall, but plotting these series together takes some thought because Rain and Yield are measured in very different units. We will be using a right-hand axis, so reserve some space (4 lines should do) on the right-hand side with `par(mai=...)`.

Plot the variables `Rain0, Rain1, Rain2 and Rain3` against year on the same graph

(use `plot(...,ylim=...,type="l")` and three `lines()` commands, or convert the `iowheat` data frame into a `tsmatrix` and use `tsplot()`).

Use different colours or line styles to distinguish the four series.

We now wish to superimpose the yield data on this plot. Use `par(new=T)` to make S superimpose the next plot on the current one. Now plot `Yield` against `Year`.

Use a thicker line type and a different colour so it stands out.

Make sure when plotting that you prevent new axes from being plotted (`axes=F`) and that new axis labels are not drawn (`x/ylab=""`).

Finally, add an axis on the right hand side with `axis(4)` and label it with

```
> mtext(..., side=4, line=3)
```

8.5 Unusual plot arrangements

The `mfg` graphics parameter is a vector of four elements: the last two are the number of rows and columns in the current multiple figure arrangement, and the first two are the row and column of the current figure.

Changing `mfg` also changes `mfrow` and `mfcol` but does *not* clear the display. Thus by changing this parameter at the appropriate time you may combine different arrangements on the same page.

Using the `genotype` data, we will plot a histogram of `Wt` on the top half of the page, and boxplots of `Weight` against the genotypes of `Mother` and `Litter` in the bottom quarters.

For the first part, use a 2×1 figure arrangement, and plot the histogram in the first panel:

```
> par(mfrow=c(2,1))
> hist(Wt)
```

For the boxplots, we change to a 2×2 figure arrangement, using only the bottom half of the display, i.e. starting at the figure in row 2, column 1:

```
> par(mfg=c(2,1,2,2))
```

Now simply plot `Wt` against `Mother`, and then `Wt` against `Litter`.

9 Interactive and Dynamic Graphics

The following session introduces the interactive and dynamic graphics features of `Splus`.

`motif()`	Start up the graphics window.

`auto <-`
`  as.data.frame(auto.stats)`
`?auto`
	Convert the `auto.stats` data set into a data frame, and read about the data using the S help facility.

`attach(auto)`	Attach the `auto` data frame to the search list.

`plot(Miles.per.gallon,`
`  Weight)`
	A standard point plot of miles per gallon versus weight for various (American) cars.

`identify(Miles.per.gallon,`
`  Weight, row.names(auto))`
	The `identify()` function lets you interactively identify points on a plot.
	
	Click on a few points with the left mouse button to see what happens. When you are finished click the middle mouse button (both mouse buttons on a two-button mouse).
	
	Notice that the returned values are the indices of the points you selected.

`detach()`	Detach the `auto` data set.
`attach(janka)`	Attach the `janka` data.
`plot(Dens, Hard)`	A scatter plot of hardness versus density.

`fm <- lm(Hard ~ Dens +`
`  I(Dens^2), data=janka)`
	Fit a quadratic regression to the data.

`lines(smooth.spline(Dens,`
`  fitted(fm)))`
	Add the fitted line to the plot.

`bad.points <- identify(Dens,`
`  Hard)`
	Use `identify()` to identify potential outliers and
	high leverage points.

`good.janka <-`
`  janka[-bad.points,]`
	Create a new dataset without the bad points.

`fm2 <- lm(Hard ~`
`  poly(Dens, 2),`
`  data=good.janka)`
	Fit the model again, this time without the previously identified points.

`lines(smooth.spline(`
`  good.janks$Dens,`
`  fitted(fm2)), col=2)`
	Plot the fitted line in a different colour.

`points(Dens[bad.points], Hard[bad.points], pch=2, cex=2)`	Highlight the outliers.
	Now for a preliminary look at `locator()`.
`text(locator(1), "Outlier", adj=0)`	After you have typed in the instruction, click the pointer somewhere in the plot region and observe what happens. The `adj=0` part of the command simply specifies that the added text is to be left justified. Note that `locator()` will still work even if there is no point nearby, unlike `identify()`. Repeat this instruction for each outlier.
`detach()`	Remove `janka` from the search list.
`rm(fm, bad.points, Hard2, Dens2, fm2, good.janka)`	Clean up.

	Now for a bit of harmless fun.
`plot(1, type="n")`	Create an empty plot.
`locator(type="b")` `lines(locator())`	Notice that these commands are alternative ways of getting a similar result.
	Now we'll use this feature to create some data.
`plot(1, xlim=c(0,1), ylim=c(0,1), type="n")` `my.data <- locator(type="p")`	Create an empty plot. Use `locator()` to create a set of points forming a reasonably straight line, with one large outlier near one end. Press the middle mouse button to quit.
`abline(lm(my.data$x ~ my.data$y))`	Add in a Least Squares line of best fit.
`abline(rreg(my.data$x, my.data$y), lty=3)`	Add in a Robust Regression line of best fit in a different line type.
`rm(my.data)`	Clean up.

`?akima`	Read about the akima data which we will be using for 3D graphs.
`int.ak <- interp(akima.x, akima.y, akima.z)`	Interpolate the data. Note that linear interpolation is being used (compare with lectures where cubic interpolation was used).
`contour(int.ak)`	Produce a contour plot.
`oldpar <- par()` `par(mar=c(5,4,4,15)+0.1)`	Save `par` values and leave room for a legend
`image(int.ak)`	Produce an image plot. Notice the similarities to the contour plot made before.

`image.legend(no.na(int.ak),`
`  horizontal=F)`
	Produce a vertical-style legend. `no.na()` is a contributed function which removes missing values from vectors — have a look at its definition.

`par(oldpar)`	Reset the graphics state.

`r.ak <- sapply(int.ak,`
`  function(x)`
`  diff(range(x, na.rm=T)))`
	Calculate the difference in the range, $(x_{\max} - x_{\min})$, for each component of `int.ak` for use in specifying the view point later on.

`z.out <- persp(int.ak,`
`  eye=c(6,8,5)*r.ak)`
`title(main=`
`  "Waveform Distortion")`
	Produce a perspective plot and keep the details in `z.out`.

`points(perspp(akima.x,`
`  akima.y, akima.z, z.out),`
`  col=2)`
	Add the original points to the perspective plot.

`persp.setup(lty=c(1,2,1),`
`  col=c(1,3,5),`
`  lwd=c(2,1,1))`
`persp(int.ak, eye=c(-6,8,5)`
`  *r.ak)`
	Plot hidden lines in a dashed style, with lines on the top thicker, and re-plot.

`rm(int.ak, r.ak, z.out)`	Clean up.

`x <- rnorm(500)`
`y <- rnorm(x)`
`plot(x, y)`
	Generate two standard normal vectors of x- and y-coordinates, and plot them in the plane.

`persp(hist2d(x, y,`
`  nxbins=20, nybins=20,`
`  scale=T),`
`  eye=c(-6,-4,1))`
	Create a perspective plot of the data, scaled so that the area under the plot is 1.

`rm(x, y)`	Clean up.

`?aust.rainfall`
`image(aust.rainfall,`
`  xlab="longitude",`
`  ylab="latitude")`
	Make an image of the fictitious Australian rainfall data.

`image.legend(no.na(`
`  aust.rainfall$z))`
	Place a legend somewhere in the Bight.

`oz(add=T)`	Add a map.

`contour(aust.rainfall,col=8,`
`  lwd=2,add=T)`
	Superimpose contours.

`?state`	Inspect the `state.x77` data using `spin()`.
`spin(state.x77)`	

`brush(state.x77, hist=T,`
`  spin=F)`
	`brush()` can be used with or without `spin()`.
	
	Ask yourself questions such as, “Which state has the highest murder rate?” and attempt to answer these questions using the brush.
	
	Note that you can highlight all of the points represented by a column of the histograms which appear at the bottom of the plot area.

10 Design and normal linear models

10.1 Design

Construct a $3^2 2^2$ orthogonal array:

```
> dsn <- oa.design(c(3,3,2,2))
```

Generate a dummy response with main effects by coercing one or two of the factors thus generated to numeric, and add some random error:

```
> attach(dsn, 1)
```

```
> y <- as.numeric(A) + as.numeric(C) + rnorm(36)/2
```

and analyse the "data".

Try a similar experiment with a 1/4—replicate of a 2^7 factorial design.

10.2 The Tuggeranong house price data

Build up a predictive model for `Price` from the other variables by any means you consider reasonable. For example, fit a full model initially

```
> fm <- lm(Price ~ ., data=house)
```

and eliminate the least significant term until only significant terms remain.

Also consider including possible "interaction" terms with central heating.

Use `coplot()` or other tools in creative ways to suggest which interaction terms might be needed. You should look for variables on which price has a markedly different regression with and without central heating.

10.3 The Iowa wheat yield data

If you have not already done so, build an interpretative model for the historical Iowa wheat yield data.

It is known that years which are either too wet or too dry in the critical periods can both lead to reduced crops. Similarly years that are too hot or too cold. This suggests that quadratic terms in some of the predictor variables should be considered in the interpretative equation.

In including quadratic terms you should square the mean corrected version of the original variable, not the uncorrected version.

Fit the full model and plot the residuals against the squared, mean corrected variables. Add in a few that prima facie seem to be the most useful and proceed with the elimination. You should, however, respect the marginality condition and do not eliminate any term x, say, while the corresponding quadratic term, $(x - \overline{x})^2$ is still present in the equation.

10.4 The sugarbeet data

1. Convert the sugarbeet data frame `sugar` into a `design`.

   ```
   > class(sugar) <- c("design", class(sugar))
   ```

   and plot it. Ignoring the covariate, which factors appear most to influence the weight, at least marginally?

2. An old way to do analysis of covariance is as follows

   - Fit the same linear model to both response y and covariate x,
   - Regress the response residuals on the covariate residuals through the origin, giving regression coefficient, say b,
   - Fit the original linear model to $y - bx$ and give the ANOVA table, adjusting the error degrees of freedom by subtracting 1.

   Implement this procedure for the sugarbeet data, and compare it with the standard ANOVA table got by

   ```
   > fm <- aov(Wt~No + Block + V*N*P*K, sugar)
   ```

   ```
   > drop1(fm, Wt~.)
   ```

11 Multistratum experiments and multiresponse models

11.1 The Barley data

Work through the barley data example as outlined in the lecture.

- Firstly with the default factor coding, and
- Secondly with the orthogonal polynomial factor coding and partitioning.

Compare and comment.

11.2 The painters data

Attach the painters data frame and work through the example given in the lecture.

Which schools appear to be the most clearly separated from each other?

Continue identifying painters until you have found at least the following: Da Vinci, Durer, Michaelangelo, Rubens, Tintoretto and Titian,

As an alternative discrimination procedure, look up and try a tree-based model and prediction. The critical commands are

```
> fm <- tree(School~., painters)
```

```
> plot(fm); text(fm)
```

The command

```
> pr <- predict(fm)
```

gives a probability forecast for each painter of membership of each school (as if it were unknown).

11.3 The pottery data

Perform a similar analysis for the pottery data and display the results as a plot of the first two discriminant function scores, with the points plotted as the site character.

This data has the feature common to many textbook examples of having one variable essentially a perfect separator of at least one group from the others. (cf. also the iris data and the jellyfish data.) In the real world things are usually less simple, and the painters data is in this respect at least much more typical.

12 Generalized linear modelling

12.1 The snail data

Complete the analysis of the snail data begun in lectures. In particular

1. Make sure that in the data frame `Species` is a factor and the other determining variables are numeric.

2. Some prior analysis has indicated that a quadratic term in exposure is likely to be necessary. Calculate such a predictor variable

   ```
   > E2 <- (Expsoure - mean(Exposure))^2
   ```

   for use in the fitted models. Let `Y` be the required two column response variable matrix for fitting binomial models.

3. Fit the general outer model

   ```
               Y ~ Species/(Exposure + E2 + Rel.Hum + Temp)
   ```

   using a logit or probit link.

   Fit the "parallel" submodel

   ```
               Y ~ Species + Exposure + E2 + Rel.Hum + Temp
   ```

   and test this as a submodel of the outer model. (`anova(..., type="Chisq")`).

4. Use the `drop1()` command to check if any further terms should be dropped.

5. [Optional] Produce a table of estimated probabilities of death. For this purpose you could construct a `design` object making all the stimulus variables into factors and including the fitted values as a quantitative variate. You could then use `design.table()` to produce the table.

   ```
   > sn <- design(Species, factor(Exposure),
                                    factor(Rel.Hum), factor(Temp))
   > sn[,"fit"] <- fitted(fm)
   > tab <- design.table(sn)
   > print(format(round(tab, digits=3)),quote=F) # easy reading
   ```

12.2 The Janka data, revisited

Without transformation the regression of hardness on density requires a quadratic term in density, and is markedly heteroscedastic, with the variance increasing with the mean.

The square root transformation of the hardness appears to go some way towards stabilizing the variance, but not entirely, and removes the need for a quadratic term in density in the mean model.

The log transformation does appear to stabilize the variance but requires the quadratic term in the mean model.

An intermediate approach is to fit a quasi-likelihood model with a `sqrt` link and a variance function proportional to the mean.

1. Plot the raw janka data and include the `loess()` local approximating regression curve:

   ```
   > plot(Dens, Hard)
   
   > larc <- loess(Hard Dens); lines(Dens, fitted(larc))
   ```

2. Fit the quasi-likelihood referred to in the preamble above, and superimpose its fitted values as a line with a different line type.

3. Fit a similar quasi-likelihood model with a quadratic term and check its necessity with a call to `anova()`.

4. Plot the "working" residuals from the model fitted in part 2 against the fitted values and check visually for signs of excess heteroscedasticity.

12.3 The chlorine data, a preliminary

Read the description of the chlorine data. The problem here is to fit a nonlinear regression of the form

$$Y = \beta_0 + \beta_1 \exp(-\theta t)$$

We discuss later the S-PLUS facilities for fitting such nonlinear regressions, but one problem that needs to be solved beforehand is to find fairly good approximate initial values for the parameters β_0, β_1 and θ.

If we had a value for θ, say $\theta^{(0)}$ we could find corresponding values for the βs by regressing Y on $x = \exp(-\theta^{(0)}t)$ and taking the intercept and slope. So the primary problem is to find a $\theta^{(0)}$.

About the simplest way to do this in S-PLUS is to take any value, say $\theta = 1$ and plot Y against $x = \exp(-\theta t)$ and check visually for how "linear" it appears.

```
> attach(chlorine)

> plot(exp(-1.0*Time), Chlor)
```

Then using command recall, adjust the value of θ by either halving or doubling, say, as appropriate and re-plotting. Continue unitl the plot seems closest to a straight line.

Once you have a satisfactory value for $\theta^{(0)}$ find the corresponding $\beta_0^{(0)}$ and $\beta_1^{(0)}$ by regression.

Hold the values permanently for a future exercise.

13 Generalized additive models

13.1 Curvature in the Iowa wheat data

1. Investigate the need for curvature terms in modelling the response with the Iowa wheat data in a method similar to that outlined in lectures.

   ```
   > fms <- gam(Yield~bs(Rain0, 3) + bs(Rain2, 3) + bs(Temp4, 3) + Year,
   data=iowheat)
   ```

   ```
   > par(mfrow=c(2,2))                                  # all on one plot.
   ```

   ```
   > plot(fms, se=T)
   ```

2. As well as $B-$splines, investigate using natural splines `ns()` and/or smooth splines `s()` for the fitting process.
3. [Open ended extension.] Fit an ordinary regression model with a suitable choice of linear and quadratic terms, and consider whether other terms are now needed.

13.2 Locally weighted regression in the Janka data

1. Fit a locally weighted regression to the Janka data and compare it with the fitted curve from the quasi-likelihood model fitted when last you used the Janka data.

   ```
   > fma <- gam(Hard lo(Dens), data=janka)
   ```

2. Experiment with the `span=` and `degree=` parameters of the `lo()` function to see how (if anything much) it affects the characteristics of the fitted curve in this case.
3. Compare this fitted curve with a *local regression model* got by

   ```
   > fmb <- loess(Hard Dens, data=janka)
   ```

14 Nonlinear regression

14.1 The chlorine data

The data was the amount of active chlorine `Chlor`, Y, remaining in a product at various times, `Time`, t after manufacture. The suggested nonlinear regression curve was of the form

$$Y = \beta_0 + \beta_1 \exp(-\theta t) + E$$

1. Using the initial value for θ got in a previous laboratory session, find initial values for β_0 and β_1 as well by ordinary regression.
2. Fit the nonlinear regression model by the method outlined in lectures.
3. Any standard diagnostic plot of the residuals will reveal two points with suspiciously large residuals. Identify these two points in any simple way and refit the model omitting them. Check what effect their omission has on the parameter estimates and their standard errors.

14.2 The Stormer viscometer data

The suggested nonlinear regression model in this case has the form

$$T = \frac{\beta V}{W - \theta} + E$$

where T is the time, V the viscosity and W the actuating weight.

1. If an initial value for θ is available, a corresponding initial value for β can be got by regression of T on $x = \frac{V}{W-\theta}$ *through the origin*:

   ```
   > dummy <- lm(Time~I(Viscosity/(Wt - th)) - 1, data=janka)
   ```

   One way to get such an initial value for θ is the “plot and adjust” method, however a simpler one in this instance is to multiply through by the denominator of the regression function and re-arrange to give

   $$WT = \beta V + \theta T + E^{\star}$$

   and use ordinary regression methods in the obvious way.

   Try either.
2. Fit the nonlinear regression and check that the parameter estimates agree with those got via the quasi-likelihood model.
3. Perform the usual diagnostic checks. Do the residuals appear reasonable for a normal, homoscedastic model? If not, what would you do about it?

4. Using the `deriv()` function, find a function that `nls()` can use to estimate the parameters using first derivative information, and look carefully at it. Suppose `obj` is the previous `nls()` fitted object. Then the appropriate step is

    ```
    > storm <- deriv(formula(obj), function(Wt, Viscosity, bt, th) NULL)
    ```

    Starting at the same initial values as before, extimate the nonlinear regression again using the new model function, and check whether the first derivative information has appreciably improved the convergence performance.

15 Tree based models

15.1 The painters data

The painters data tells us more about the personal predilections and prejudices of de Piles than it does of the painters in question. Nevertheless it is a natural candidate for tree based models in an interpretative rather than a predictive utility.

As data sets go it is rather small for serious use of tree based methods, however.

1. Construct a classification tree for `School` based on the four quality scores and display it. Set `minsize=1` which will lead to a tree with perfect prediction on the training sample.
2. Display the probability forecasts for School membership and hence establish that the tree does indeed correctly classify all painters.
3. Using the `identify()` function, interactively identify the painters in some of the major internal nodes of the classification tree.
4. Using `snip.tree()` prune the classification tree to a level you consider "optimal" in some sense. Record its deviance.
5. Using the automatic pruning procedure, prune the original classification tree to an optimal tree with about the same number of nodes you obtained in the manually pruned tree in step 4.

   This requires two steps:

   ```
   > plot(prune.tree(ftr))
   ```

   From the plot then visually identify a value of the tuning constant, α, (upper axis) that will produce about the required number of nodes in the optimal tree (lower axis).

   ```
   > opt.tr <- prune.tree(ftr, k=α)
   ```

   Compare the deviance of this optimal tree with that for the tree you obtained in step 4.

15.2 The survey data

The survey data comes from a student survey done to produce "live" data for the first year statistics class in the University of Adelaide, 1992. The variables recorded included `Sex`, `Pulse`, `Exer` (an exercise level rating), `Smoke`, `Height` and `Age`.

There are missing values in the data frame, so be sure to use appropriate actions to accommodate them.

1. Fit a naive linear model for `Pulse` in terms of the other variables listed above and record its deviance.

2. Construct a regression tree for `Pulse` in terms of the other variables listed. Compare the deviance with that of the linear model, (which since this is a regression tree, are comparable). Comment.

3. Using the automatic pruning procedure to what you consider to be a sensible level (without a context for use of the tree, this is impossible) and compare the residuals with those of the linear model, for example, using histograms.

16 Object Oriented Programming

16.1 A print method for `terms`

Objects of class `terms` (as produces by the `terms()` function) currently have no print method. Using the information in the help files for `terms` and `terms.object` as a guide, write a `print.terms` method function to display the information in a nice way.

16.2 A new generic function

Design a generic function `"pop"` which returns its argument after removing the the first element. Write methods (if necessary) for objects of class `"data.frame"` (remove the first row) and `"factor"`, and of course a default method (consider the cases of lists, matrices and vectors.)

16.3 A class for documentation

Create a function `document(x,docs)` which returns the original object `x` adorned with the documentation information contained in `docs`. The returned value should have class `"documented"` and inherit from the class of `x` and all its inheritants.

Write a `print` method for your new class. It should print out `"Documentation:"` followed by a new line and the documentation for the object, and then the value of the object.

If you have not already done so, refine your `document` function so that re-documenting an already documented function replaces the original documentation, and so that calling `document` without the `docs` argument removes the documentation.

Give `print.documented` an argument `suppress` with a default value of `F` which, if `T`, will suppress printing of the documentation.

Hints: In accordance with the principle of information hiding, the implementation details are up to you. Here are some suggestions, however:

- Store the documentation information in an attribute of the object, as in

  ```
  attr(x,".docs") <- docs
  ```

  This means that other generic functions will not change their behaviour. You may wish to limit `docs` to a character vector, but this is not necessary.

- The `document` function should prepend `"documented"` to the class list of `x` (or move it to the front if already in the class list) and return the adorned object.

- `print.documented` should simply print the documentation as required, and then call `NextMethod`.

- Don't forget to pass arguments to `print` when printing the value of the object.

A Answers to §7

First we open up a graphics window and a help window.

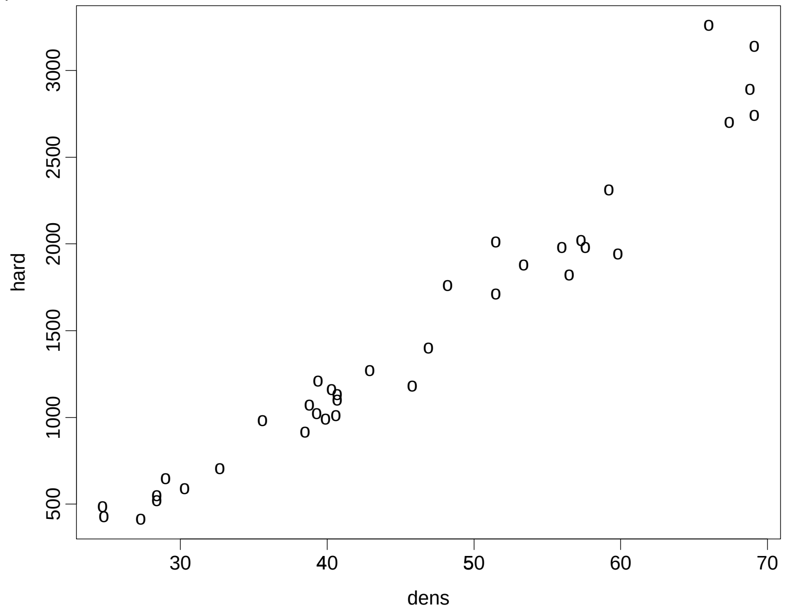
```
S-PLUS : Copyright (c) 1988, 1991 Statistical Sciences, Inc.
S : Copyright AT&T.
Version 3.0 Release 1 for Sun SPARC : 1991
Working data will be in .Data
> motif()
> help.start()
```

8.1 The janka data

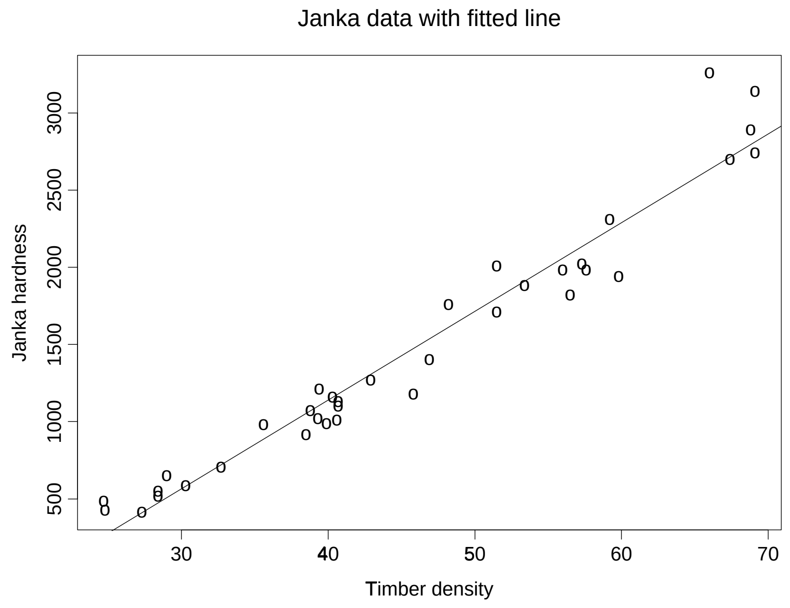
```
> ?janka
Sending request to help window
This object inherits from the following classes.
Select those for which you would like documentation.

Choices are :
   1 : data.frame
Enter number of choice (0 means no choice): 0
```

Entering 1 here instead of 0 brings up documentation for `data.frame` as well.

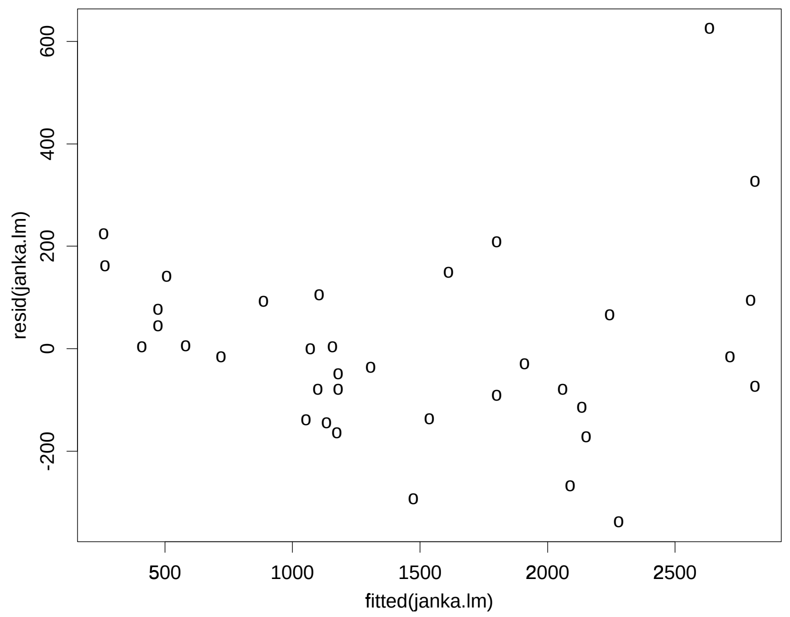
```
> attach(janka)
> plot(Dens,Hard)
```



```
> plot(Dens,Hard,xlab="Timber density",ylab="Janka hardness")
> janka.lm <- lm(Hard ~ Dens, data=janka)
> abline(janka.lm)
> title("Janka data with fitted line")
```



```
> plot(fitted(janka.lm),resid(janka.lm))
```



There is obviously curvature here ... evidently the straight-line model isn't explaining the data very well.

7.2 Maps

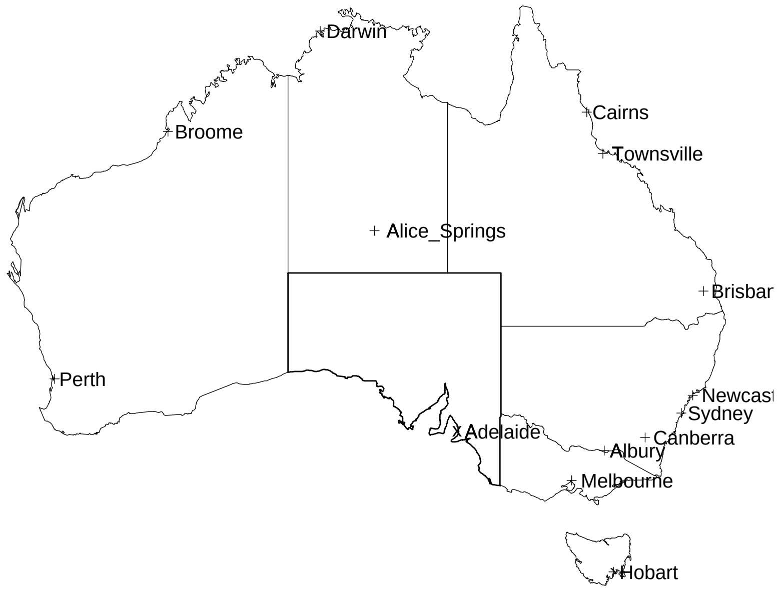
```
> oz()
> sa(add=T,col=2)
```

Since I can't use colour on these solutions, I actually used `lwd=2` instead of `col=2` to highlight South Australia.

```
> row.names(oz.cities)
 [1] "Adelaide"      "Albury"        "Alice_Springs" "Brisbane"
 [5] "Broome"        "Cairns"        "Canberra"      "Darwin"
 [9] "Hobart"        "Melbourne"     "Newcastle"     "Perth"
[13] "Sydney"        "Townsville"
> points(oz.cities["Adelaide",],pch="x")
> text(oz.cities["Adelaide",],"Adelaide",adj=-0.1)
```

To plot labels for all cities, use the fact that `text()` will accept vectors as arguments. Adelaide is excluded from the `oz.cities` dataframe with `[-1,]` so it is not plotted twice.

```
> points(oz.cities[-1,],pch=3)
> text(oz.cities[-1,],row.names(oz.cities[-1,]),adj=-0.1)
```



7.3 Survey data

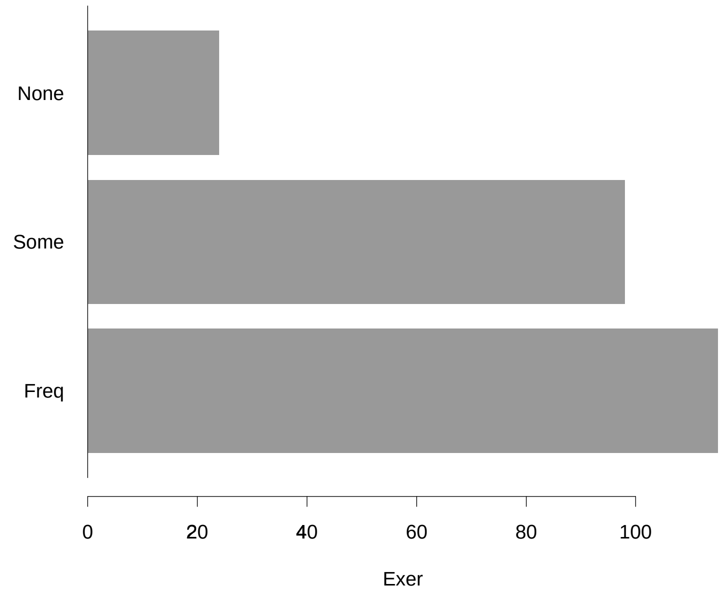
```
> plot(survey)
```

The 12 plots generated here have been omitted for brevity.

```
> attach(survey)
> ?survey
Sending request to help window
This object inherits from the following classes.
Select those for which you would like documentation.
1: data.frame
Selection: 0
```

To create a bar plot of `Exer` we just use `plot()`:

```
> plot(Exer)
```



For a pie chart, we need to first tabulate the frequencies:

```
> exer.freq <- table(Exer)
> exer.freq
 Freq Some None
  115   98   24
```

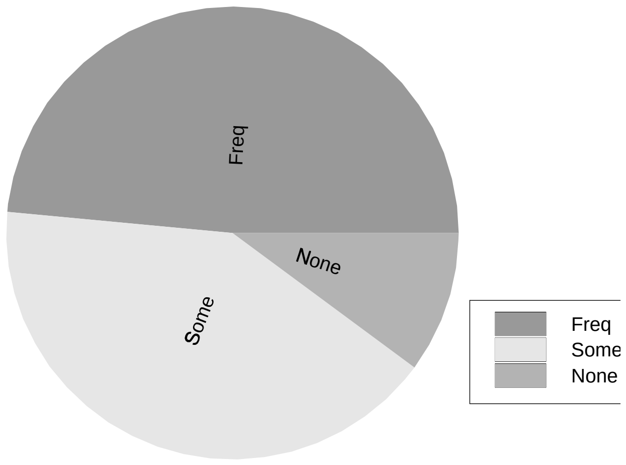
The command `pie(exer.freq)` will now create a pie chart, but to add labels to the slices we use the `names=` argument.

```
> pie(exer.freq,names=levels(Exer))
```

Adding a legend is accomplished by using `legend()` with the `fill=` argument. Note that we have used an alternative method of getting the labels: this time the `names` attribute of `exer.freq` is used.

```
> legend(locator(1),names(exer.freq),fill=1:3)
```

Clicking somewhere in the lower-right corner of the graphics window will leave you with a plot something like:

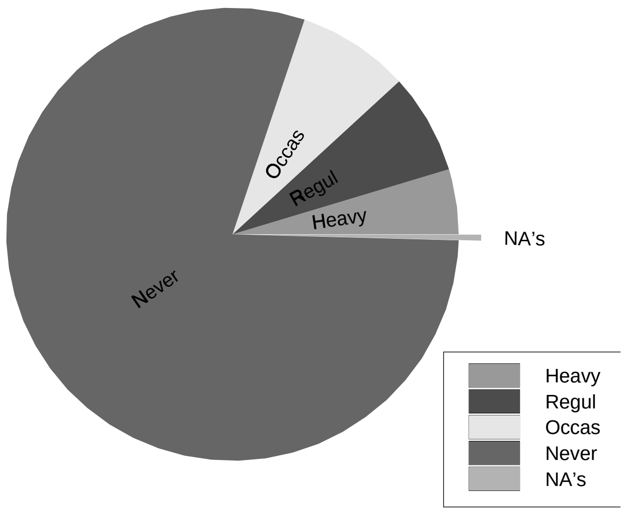


For the `Smoke` variable a slightly different approach is needed if we wish to include the missing value in the plot.

```
> smoke.freq <- summary(Smoke)
> smoke.freq
 Heavy Regul Occas Never NA's
    11    17    19   189    1
```

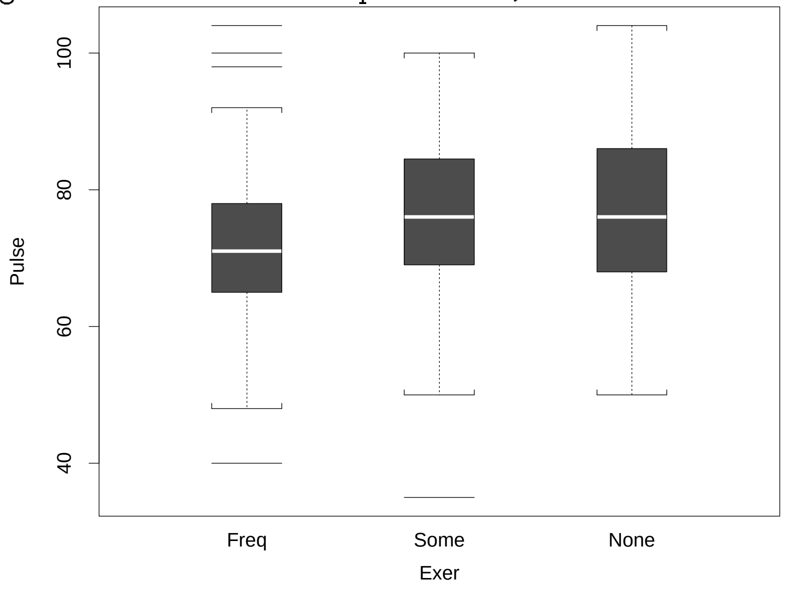
Since the missing value represents such a small proportion of the data, we highlight it with `explode=5` (because `NA's` is the fifth category) so it isn't lost in the pie. There's also a legend for good measure.

```
> pie(smoke.freq,names=names(smoke.freq),explode=5)
> legend(locator(1),names(smoke.freq),fill=1:5)
```



(b)

```
> plot(Exer,Pulse)
Warning messages:
  45 missing values in Pulse in: plot(Exer, Pulse)
```

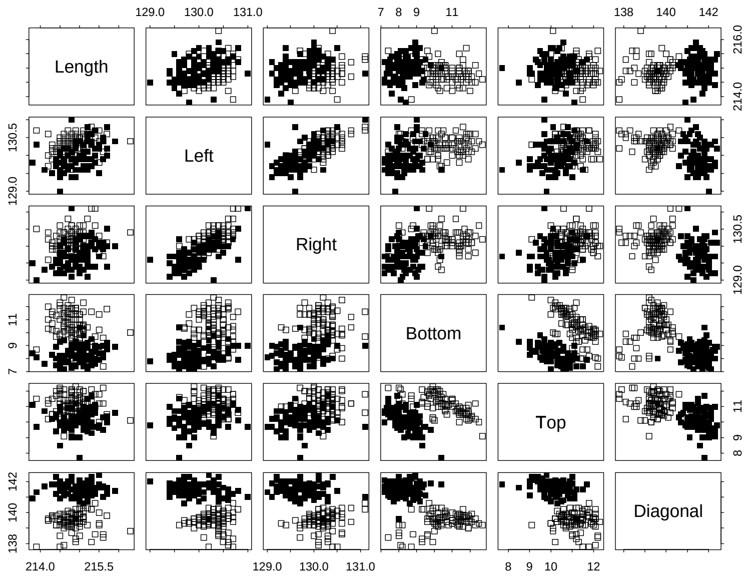


If you wish, also plot `Height` against `Sex` or `Exer`.

7.4 Swiss data

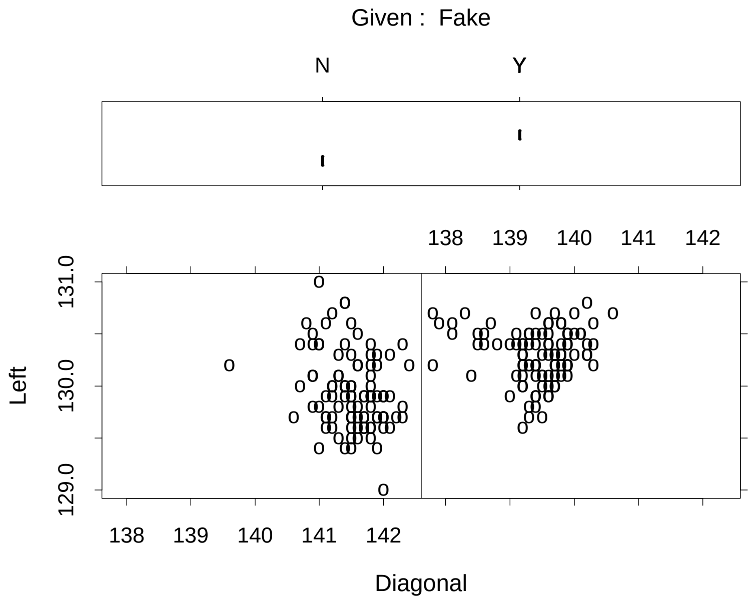
I have used the `panel=` argument to differentiate the fake and legal notes in each scatterplot, instead of simply plotting against `Fake`:

```
> pairs(swiss[,-1],panel=function(x,y) {
+ points(x[swiss$Fake=="N"],y[swiss$Fake=="N"],pch=15)
+ points(x[swiss$Fake=="Y"],y[swiss$Fake=="Y"],pch=0)})
```



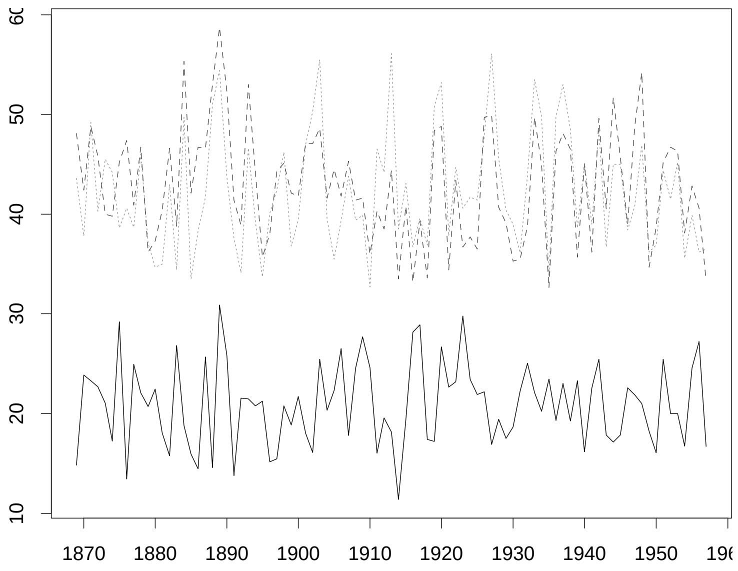
The legal notes are in solid squares, and the known fake notes are represented by the hollow squares. Note that in many plots one particular "real" note stands out amongst the fakes. `Diagonal` against `Left` seems to differentiate the fakes from the real notes fairly well — a coplot will bring the intruder note into the open:

```
> coplot(Left ~ Diagonal | Fake)
```



7.5 Rain data

```
> tsplot(rain)
```



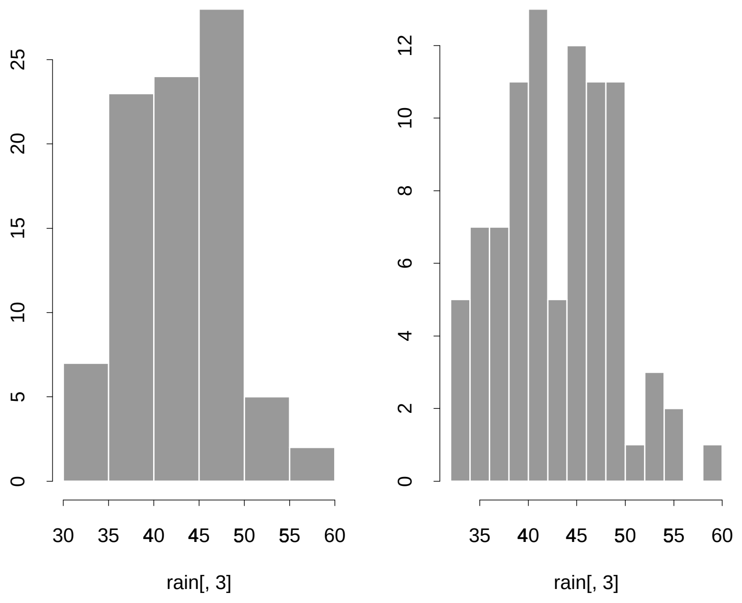
Note that the Adelaide rainfall was much lower than the New York rainfalls. Had we tried to plot each time series in turn, say with

```
> plot(rain[-3],type='l')
> lines(rain[-2])
> lines(rain[-1])
```

we would have had "Lines out of bounds" errors when plotting the New York series since they would have been well outside the Adelaide plot. The `tsplot()` function overcomes this problem, as well as producing sensible tick labels.

The two histograms can be created with the commands

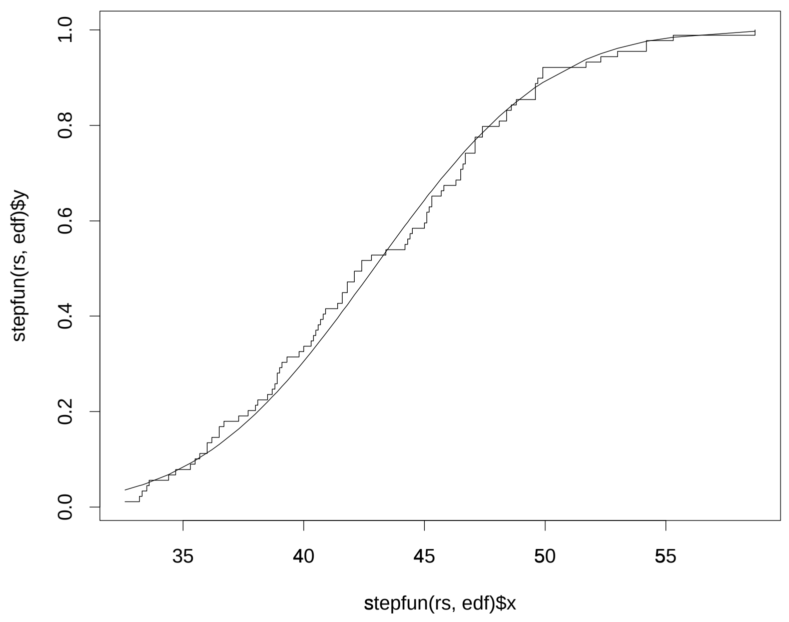
```
> hist(rain[,3])
> hist(rain[,3],nclass=10)
```



Note that the distribution appears rather skew.

On the other hand, the empirical and normal distribution functions are reasonably close:

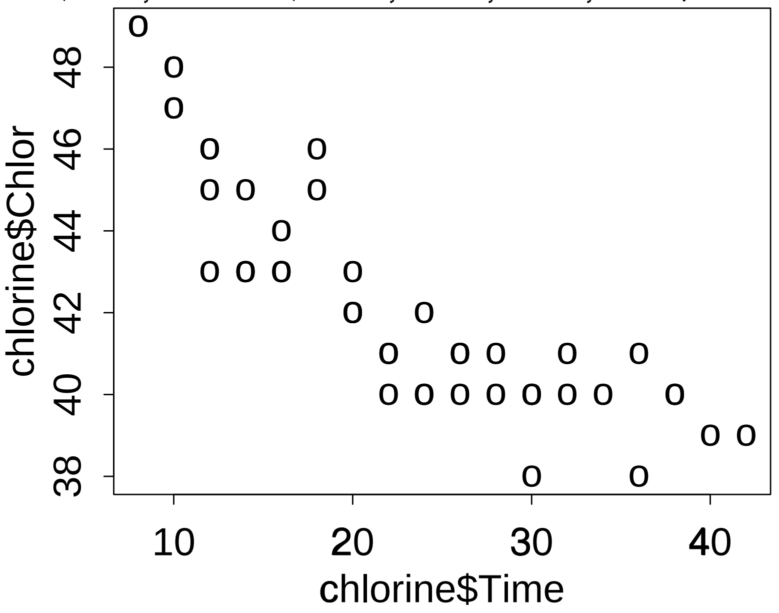
```
> rs <- sort(rain[,3])
> edf <- (1:length(rs))/length(rs)
> plot(stepfun(rs,edf),type="l")
> lines(rs,pnorm(rs,mean=mean(rain[,3]),sd=sqrt(var(rain[,3]))))
```



B Answers to §8

1. Passing graphics parameters.

```
> par(mar=rep(5,4))
> plot(chlorine$Time,chlorine$Chlor,cex=2,col=2,lwd=2)
```



Don't forget to reset the graphics parameters with

```
> par(oldpar)
```

2. Multiple Figures.

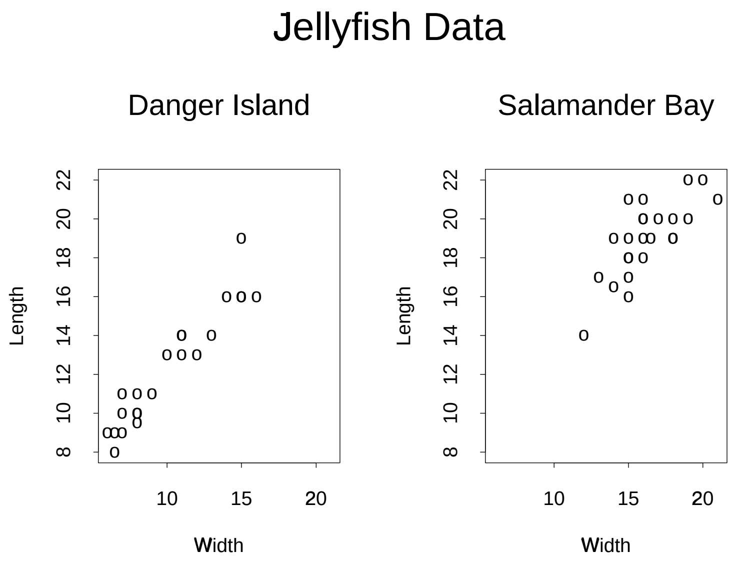
```
> attach(jellyfish)
> par(oldpar)
> par(mfrow=c(1,2),oma=c(0,0,4,0))
```

We need to put 4 lines of space at the top of the page before we start plotting.

```
> plot(Width[Sample=="D"],Length[Sample=="D"],xlim=range(Width),
+   ylim=range(Length),xlab="Width",ylab="Length")
> title("Danger Island")
> par(xaxs="d",yaxs="d")
```

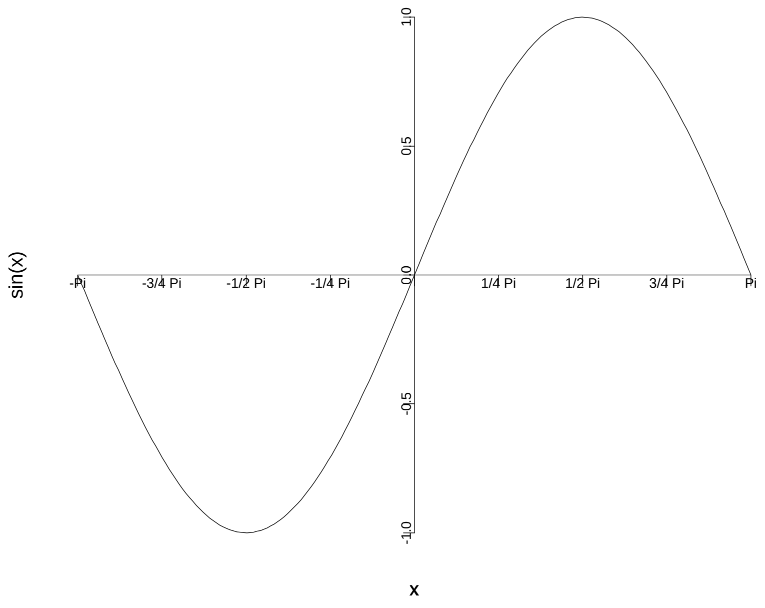
This `par()` call locks in the current axis limits for use in subsequent plots.

```
> plot(Width[Sample=="S"], Length[Sample=="S"],
+                             xlab="Width",ylab="Length")
> title("Salamander Bay")
> mtext("Jellyfish Data",side=3,line=1,outer=T,cex=2)
```



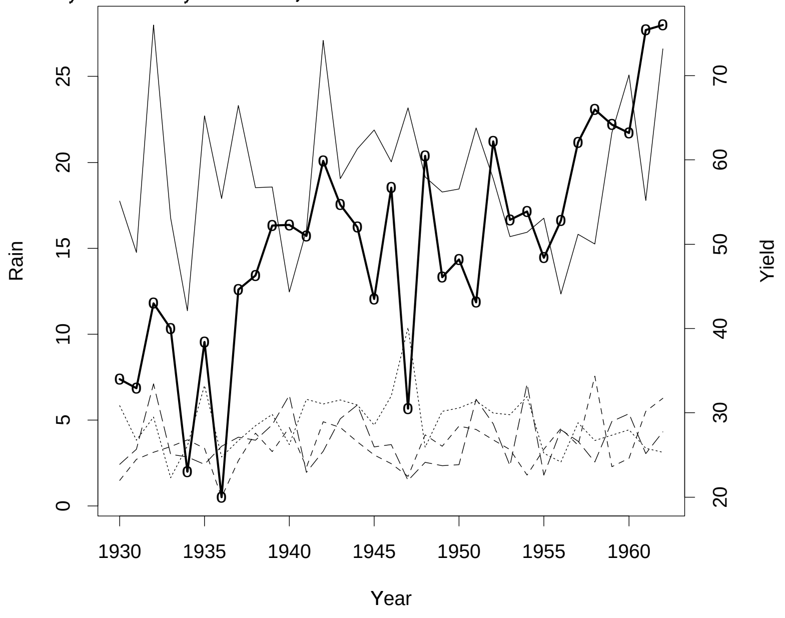
3. Axes through the origin.

```
> x <- seq(-pi,pi,length=200)
> plot(x,sin(x),type="l",axes=F)
> axis(1,pos=0,at=seq(-pi,pi,by=pi/4),labels=c("-Pi","-3/4 Pi",
+  "-1/2 Pi", "-1/4 Pi","","1/4 Pi","1/2 Pi","3/4 Pi", "Pi"))
> axis(2,pos=0,yaxs="e")
```



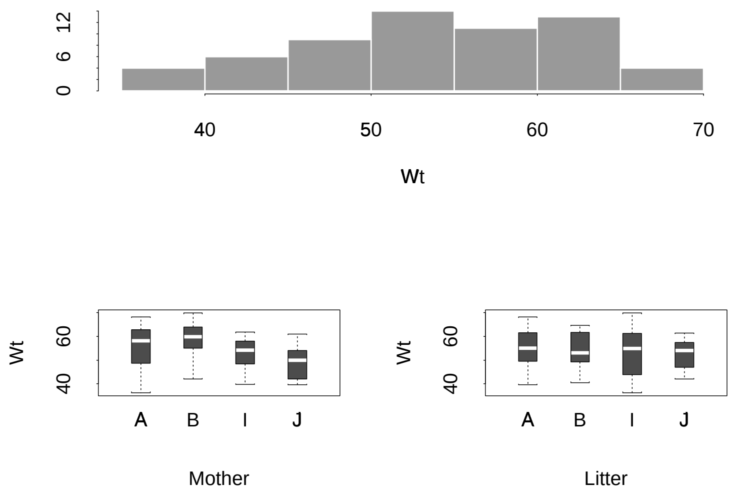
4. Multiple axes.

```
> attach(iowheat)
> par(mar=c(4,4,4,5))
> plot(Year, Rain0, ylab="Rain",
+                      ylim=range(Rain0,Rain1,Rain2,Rain3), type="l")
> lines(Year, Rain1, col=2)
> lines(Year, Rain2, col=3)
> lines(Year, Rain3, col=4)
> par(new=T)
> plot(Year, Yield, xlab="", ylab="", col=5, lwd=2, type="o", axes=F)
> axis(4)
> mtext("Yield",side=4,line=3)
```



5. Unusual plot arrangements.

```
> par(mfrow=c(2,1))
> hist(Wt)
> par(mfg=c(2,1,2,2))
> plot(Mother,Wt)
> plot(Litter,Wt)
```



C Answers to §16

C.1 A print method for `terms`

Here's one solution.

```
"print.terms"<-
function(termobj, ...)
{
        cat("Terms:\n")
        print(attr(termobj, "term.labels"), ...)
        if(attr(termobj, "intercept"))
                cat("plus intercept term.\n")
        cat("\nFormula: ")
        print(attr(termobj, "formula"), ...)
        response <- attr(termobj, "response")
        if(response == 0)
                cat("There is no response.\n")
        else {
                cat("Response is ")
                print(attr(termobj, "variables")[[attr(termobj,
                        "response")]],...)
        }
        invisible()
}

> terms( y ~ a/(b*c)+d)
Terms:
[1] "a"          "d"          "b %in% a"   "c %in% a"   "b:c %in% a"
plus intercept term.

Formula: y ~ a/(b * c) + d
Response is y
```

C.2 A new generic function

Unsurprisingly, the generic simply calls `UseMethod`:

```
"pop"<-
function(x, ...)
UseMethod("pop")
```

One implemetation of the default method is as follows:

```
"pop.default"<-
function(x, ...)
{
        if(is.matrix(x))
                x[-1, ]
        else if(is.list(x)) {
                x[[1]] <- NULL
                x
        }
        else x[-1]
}

> pop(1:10)
[1]  2  3  4  5  6  7  8  9 10
> pop(list(1,2,3))
[[1]]:
[1] 2

[[2]]:
[1] 3
```

It turns out that no specific method is needed for classes `"data.frame"` (since data frames satisfy `is.matrix()`) or `"factor"` (since vector-style subscription works.) You may well ask: "Why bother to write a generic funtion, then?" The answer is that if any new class is added at a later time which needs to use the generic `pop()`, the designer need only add a method for the new class, rather than having to modify a single, non-generic `pop()` function.

C.3 A class for documentation

Here's one implementation for `document`:

```
"document"<-
function(x, docs)
{
    attr(x, ".documentation") <- if(missing(docs)) NULL else docs
    newdoc <- if(is.null(class(x))) 0 else match(class(x), "documented",
            nomatch = 0)
    if(sum(newdoc == 0) && missing(docs))
        return(x)
    if(sum(newdoc) == 0)
        class(x) <- c("documented", class(x))
    else {
        newclass <- c(if(!missing(docs)) "documented", class(x)[ -
            newdoc])
        class(x) <- if(length(newclass) == 0) NULL else newclass
        x
    }
}
```

Note that there are four special cases to consider: adding documentation to documented and undocumented objects, and removing documentation (i.e. `docs` is missing) from [un]documented objects. Hence all the `if` statements.

The print method is somewhat simpler:

```
"print.documented"<-
function(x, suppress = F, ...)
{
    if(!suppress) {
        cat("Documentation:\n")
        print(attr(x, ".documentation"))
        cat("\n")
    }
    if((length(class(x)) == 1) && (class(x) == "documented")) {
        attr(x, ".documentation") <- NULL
        print(unclass(x))
    }
    else NextMethod("print")
}
```

The second `if` statement handles the case when the only inheritant is the default print method, and prevents the attributes of the object being printed. Here's a test of the four cases mentioned before:

```
> test <- document(1:10,"Silly example.")
> test
Documentation:
[1] "Silly example."

 [1]  1  2  3  4  5  6  7  8  9 10
> document(test)
 [1]  1  2  3  4  5  6  7  8  9 10
> document(test,"New documentation.")
Documentation:
[1] "New documentation."

 [1]  1  2  3  4  5  6  7  8  9 10
> document(1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
```