Non-linear Modelling

Remko Duursma

WESTERN SYDNEY
UNIVERSITY

Hawkesbury Institute
for the Environment

April 13, 2017

Contents

Chapter 1

Non-linear Modelling

1.1 Introduction

This chapter introduces alternatives to linear models, with an emphasis on non-linear regression (Section 1.2). Very brief introductions to local regression smoothers (loess, Section 1.3), quantile regression (Section 1.5), and generalized additive models (Section 1.4) are also included. In each case, we avoid technical descriptions of theory and implementation, and instead illustrate each technique with examples, and many figures.

Before we delve into non-linear regression, we first briefly describe when each of these techniques is useful. In general, linear models are preferable when you can use them, since they are robust, easy to use, and the theory for inference (hypothesis testing) is well developed and not very sensitive to violation of the usual assumptions.

In general, as the name suggests, non-linear methods are to be used when the relationship between your response variable and predictor is not linear.

Transformations

In this case, one common approach is to transform the predictor and/or response so as to linearize the relationship between the two. When this is possible, it is usually the preferred approach. For example, you may find that your data can be well described with an equation of the form:

$$Y = a \cdot X^b$$

This is the so-called power function. You could go ahead and use non-linear regression to fit this curve, but it would be better to linearize this relationship by taking the logarithm of both sides, which gives:

$$log(Y) = a + b \cdot log(X)$$

And thus we can use linear regression of $log(Y)$ against $log(X)$.

In a number of cases, though, there is no simple transformation that linearizes the relationship. Typical cases are relationships with a clear asymptote (though exceptions exist), peaked relationships, or very irregular relationships with lots of peaks and curves.

Parametric or non-parametric?

In the case where no transformation can be found, or you have a specific equation that you want to fit to your data, non-linear regression may be an appropriate technique. This is an example of a parametric method, because you end up estimating specific parameters, and in doing so making assumptions about the underlying distribution. This can be useful because a fitted non-linear regression model can be applied in a new setting, for example predicting for a new dataset, or reliably extrapolating beyond your data.

In non-parametric approaches, you either cannot find or are not interested in finding an equation that describes your data. You may simply be interested in describing or visualizing a (potentially irregular) trend, maybe comparing groups, but not estimating specific parameters or predicting for new situations (let alone extrapolation). In this chapter we will briefly look at `loess` (Section 1.3), a flexible non-parametric regression method, and generalized additive models (Section 1.4), a semi-parametric approach.

Means or quantiles?

In the usual linear models, as well as non-parametric regression, we are usually forced to be interested in the mean relationship between response and predictor. Often this is what you want, but there are many cases in which you are actually interested in questions like 'What is the maximum value of my response as a function of my predictors?'. Instead of actually estimating the maximum, a particular quantile will be much more robust to outliers (for example, the 95% quantile). In this case we can use quantile regression, which will be introduced in Section 1.5. This is not strictly a non-linear method, but we present it in this chapter because it is such a constrast to the usual linear models, and because non-linear quantile regression can also be applied (Section 1.5.2).

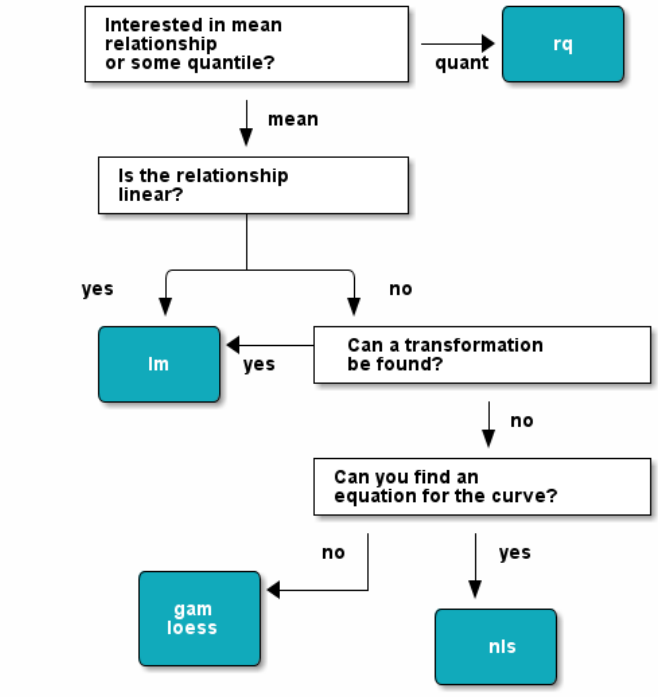

Figure 1.1: A simple flow chart to select regression methods.

Table 1.1: Packages used in this chapter. The `nls` function and associated methods are in the `stats` package, which is base R, and therefore not mentioned in this table.

Package	Key functions
`nlstools`	`overview`, `confint2`
`nlshelper`	`plot_nls`, `anova_nlslist`, `tidy.nlsList`
`nlme`	`nlsList`
`car`	`qqPlot`
`mgcv`	`gam`
`quantreg`	`rq`, `nlrq`

Further reading As mentioned, we present a practical, visual approach to non-linear models in this chapter. We recommend further reading if you are interested in technical aspects of non-linear regression :

- Ritz, C., Streibig, J.C., 2008. Nonlinear regression with R. Springer. (from the developers of the `nlstools` package)
- Fox, J. Nonlinear regression and nonlinear least squares in R (Online appendix to 'Companion to applied regression', http://socserv.mcmaster.ca/jfox/Books/Companion/appendix/Appendix-Nonlinear-Regression.pdf).

For generalized additive models, the book by Simon Wood accompanies the `mgcv` package:

- Wood, S.N., 2006. Generalized additive models: an introduction with R, Texts in Statistical Science. Chapman & Hall/CRC.

1.2 Non-linear regression

1.2.1 Introduction

A model is *linear* when you can write it like this:

$$Y = b_0 + b_1 \cdot X_1 + b_2 \cdot X_2 + b_3 \cdot X_3 \tag{1.1}$$

where Y is the dependent variable, and the X's are the predictors. The predictors can be any variable or some transformation thereof (for example, squared terms, two variables multiplied together, etc.). This model is *linear in the coefficients* b_1, b_2, etc, though it may contain terms that are non-linear in the X's (such as squared terms of X).

An example of a non-linear model is the Chapman-Richards growth equation,

$$Y = Y_{max} \cdot (1 - e^{-b \cdot x})^c \tag{1.2}$$

Here the coefficients (Ymax, b and c) occur non-linearly in the equation, and you cannot write this in a form similar to the linear model as shown above. We can look at the shape of this curve by using `curve`, like in the following example (Fig. 1.2).

```
# Define function
chapm <- function(x,Asym,b,c)Asym*(1-exp(-b*x))^c

# Plot (curve must use an x argument)
curve(chapm(x, Asym=100, b=0.15, c=3), from=0, to=50, ylab="Y")
```

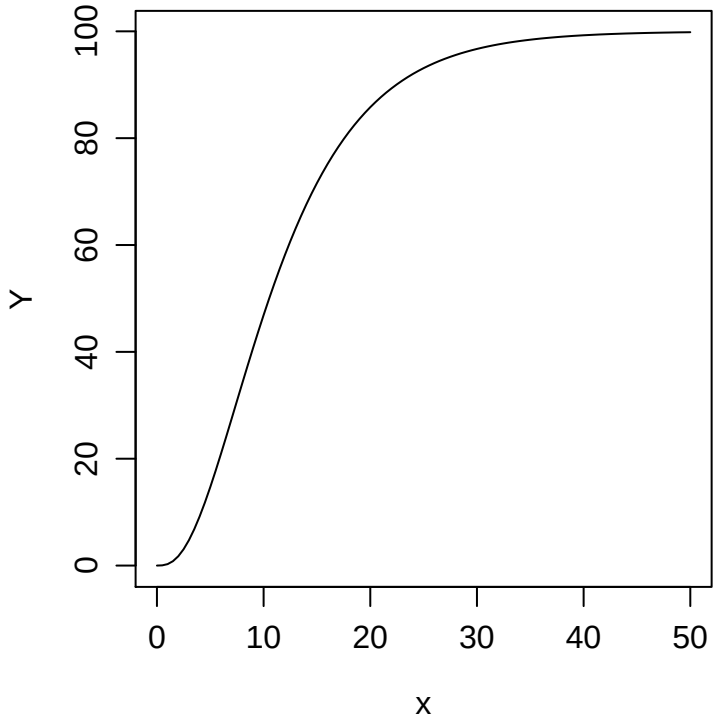

Figure 1.2: The Chapman-Richards growth equation.

We can use non-linear regression to fit curves like that one to data, which gives estimates of the coefficients Ymax, b and c. The majority of useful applications of non-linear regression are when you have an equation already specified (arising from theory), and you wish the estimate the parameters of the curve, or use the curve for prediction (as in lots of modelling applications).

In the following examples of non-linear regression, we use the `nls` function, but point out the `nlsLM` function (in package `minpack.lm`) for a very promising alternative, and `nls2` (package of same name) for an alternative that can be used to find starting values.

1.2.2 Model selection

How do you choose the right model to fit to your data? In some applications of non-linear regression, you already know the model you want to fit, as it arises from theory. The Michaelis-Menten model, very commonly applied in enzyme kinetics, is an example (see Exercise 1.6.2).

In other cases you have to choose a model from various options, and select the best fitting one, or the one that has certain properties. One reason to apply non-linear regression, as stated above, is that the model can be expected to behave well outside the range of the data used for fitting. Models with an asymptote are a good example. Take the following equation,

$$Y = Y_{max}(1 - e^{-kX})$$

we can see that this model reaches an asymptote for very large X (because e^{-x} goes to zero for large x), given by the parameter Y_{max}. Also, this model has the, possible useful, quality that Y is zero when X is zero.

Further reading I still cannot find a better compendium of equations used in non-linear regression than this manual published by the Ministry of Forests in British Columbia, Canada (download PDF from https://www.for.gov.bc.ca/hfd/pubs/docs/bio/bio04.htm). It includes a huge number of example equations, and visually shows how the shape of the curve changes with the various parameters. Example applications in SAS may not be relevant.

1.2.3 Fitting a simple model

In this section, we show how to implement a simple non-linear regression model with the `nls` function (available in base **R**). In this first example, we define our model by hand, which is the most flexible approach but it comes at a cost of having to specify suitable starting values, or initial guesses, of the parameters. In the next section, we introduce built-in self-starting functions which do not require starting values.

When you specify your own model, you have to specify suitable *starting values* for the parameters, otherwise `nls` will in many cases not converge to a solution. Depending on the situation, these starting values need to be quite good. This will depend a lot on the type of curve you are fitting, the number of data points, and the number of parameters describing the curve.

To find suitable starting values, you can either guess them based on the meaning of the parameters (for example, the asymptote, or value of y when x is zero, and so on), or otherwise find them with trial and error. The following 'Try it yourself' box explains how to plot the curve with the data, as a method to find starting values.

Try this yourself Using the Loblolly data as in the example below, plot height versus age, and add the Chapman-Richards curve using `curve`, like so:
`with(Loblolly, plot(age, height))`
`curve(chapm(x, Asym=30, b=0.2, c=2), add=TRUE)`
Now play around with the coefficients (`Asym`,`b`,`c`) to see the change in curve shape and location, and find values so that the curve is 'somewhat close' to the data.

Here we use the built-in `Loblolly` data (see `?Loblolly`), which contains age and height of Loblolly pine (*Pinus taeda*), for several seedlots. We will fit the Chapman-Richards equation, a type of sigmoidal curve that fits these data quite well. Note that we defined the function in the previous section, but we repeat it here.

```
# Define function to fit
chapm <- function(x,Asym,b,c)Asym*(1-exp(-b*x))^c

# Note the specification of the starting values.
nls_lob <- nls(height ~ chapm(age, Asym, b,c),
               data=Loblolly,
               start=list(Asym=100, b=0.1, c=2.5))
```

If the model does not converge, it will report an error. Note that you do not have to write a function first and use it within `nls`, you can also write the equation in the formula interface of `nls` directly. However, it is usually convenient to have the function defined separately.

Next, it is useful to add the fitted curve to a plot, which is particularly easy to do with the `nlshelper` package. In Section 1.2.8, we return to predicting from a non-linear regression model, using the `predict` function. The following code makes Fig. 1.3.

```
library(nlshelper)
plot_nls(nls_lob,  ylim=c(0,80), xlim=c(0,30))
```

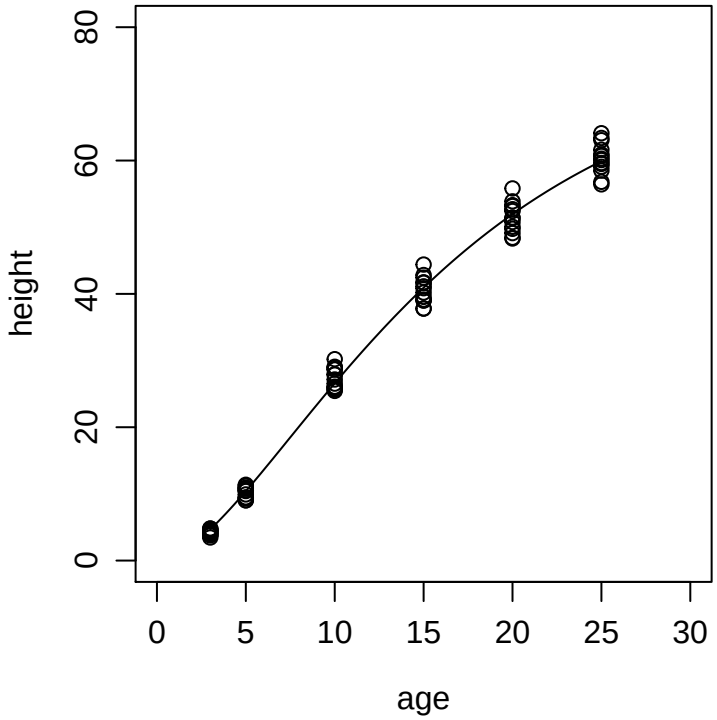

Figure 1.3: The Chapman-Richards growth equation fitted to the Loblolly data, using the nlshelper package.

1.2.4 Self-starting functions

To avoid the need for starting values, we can use one of many 'self-starting functions'. These are common non-linear models that do not require starting values, and instead estimate them from the data with clever methods. You can also develop these yourself, but this is tricky.

I won't go into much detail on these functions, but simply redo our example with the `Loblolly` data from the previous example. This time we are fitting the 'Gompertz' growth model to the data, using the `SSgompertz` function.

```
# Fit same model, using a self-starting model
nlsfitSS <- nls(height ~ SSgompertz(age, Asym, b2, b3),
                data=Loblolly)
```

Try this yourself Compare the fit of this model to the Chapman-Richard equation from above. Look at the residual standard error in the `summary` of the fitted model, and plot the curve in addition to the Chapman-Richard curve in the previous example.

1.2.5 Model summary and diagnostics

Summarizing the fit

The usual `summary` does not print a lot of useful information for `nls` models - a much better alternative is the `overview` function from the `nlstools` package:

```
library(nlstools)
overview(nls_lob)
```

```
##
## ------
## Formula: height ~ chapm(age, Asym, b, c)
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## Asym 76.933469   2.580066   29.82   <2e-16 ***
## b     0.082659   0.006067   13.62   <2e-16 ***
## c     1.846939   0.093736   19.70   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.731 on 81 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 2.935e-06
##
## ------
## Residual sum of squares: 243
##
## ------
## t-based confidence interval:
##             2.5%      97.5%
## Asym 71.79994733 82.0669900
## b     0.07058745  0.0947305
## c     1.66043435  2.0334445
##
## ------
## Correlation matrix:
##            Asym          b          c
## Asym  1.0000000 -0.9650409 -0.8491296
## b    -0.9650409  1.0000000  0.9500548
## c    -0.8491296  0.9500548  1.0000000
```

The `overview` statement shows:

- The parameter estimates, standard error (`Std. Error`), and a t-test against zero (`t value` and the p-value, `Pr(>|t|)`). The test against zero is rarely useful.
- The residual standard error, calculated as $\sqrt{\sum r^2/df}$, where r is the residual, and df the residual degrees of freedom (sample size minus number of estimated parameters). This metric is also known as the root-mean square error (RMSE).
- Some information on the convergence, normally can be ignored.
- The residual sum of squares, simply the sum of the squared residuals.
- Confidence intervals for the parameters. Note that these will be somewhat different from those calculated with `confint`.
- Correlation matrix of the parameters.

The disadvantage of `overview` is that none of the printed results can be extracted. A convenient way to print the coefficients, standard error and confidence interval is the `tidy` function from the `broom` package. The `glance` function returns things like the RMSE (`sigma`), log-likelihood, AIC, etc. Both functions

return a dataframe and are thus convenient as a basis for further calculation, or use in tables.

```
library(broom)
tidy(nls_lob, conf.int=TRUE)
```

```
##   term    estimate   std.error statistic      p.value    conf.low
## 1 Asym 76.93346868 2.580065660  29.81842 1.957501e-45 72.37773289
## 2    b  0.08265897 0.006067049  13.62425 1.168736e-22  0.07050617
## 3    c  1.84693944 0.093735923  19.70365 1.226880e-32  1.67095139
##     conf.high
## 1 82.98962856
## 2  0.09483768
## 3  2.04444265
```

```
glance(nls_lob)
```

```
##       sigma isConv       finTol    logLik      AIC      BIC deviance
## 1 1.730971   TRUE 2.934585e-06 -163.7527 335.5054 345.2287  242.697
##   df.residual
## 1          81
```

If you only want to extract the coefficients, use the base function `coef`,

```
coef(nls_lob)
```

```
##        Asym           b           c
## 76.93346868  0.08265897  1.84693944
```

Residual diagnostics

As in linear regression, an important assumption in non-linear regression is that the residuals are normally distributed. This can be inspected visually, as we strongly recommend against the use of tests for normality. The `qqPlot` function from the `car` package is very handy because it adds a confidence region for the qq-plot. The following example makes Fig. 1.4.

```
library(car)
qqPlot(residuals(nls_lob))
```

When interpreting QQ plots for non-linear regression models, don't get too upset if the residuals aren't perfectly normal. This is rarely a problem unless the deviation is very large. Most importantly, it does not affect the estimates of your parameters, but it does affect the estimated standard errors and confidence intervals. If you are mostly interested in fitting the curve and moving on, it is not important.

A more important test, again visually, is to check whether there is no pattern in the residuals with respect to the predictor(s). If there is, clearly the model does not fit very well and another model should be chosen. The following code makes a plot of the residuals against the fitted values, and a plot of the fitted values against the predictor (with a comparison to a 1:1 line) (Fig. 1.5).

```
plot(fitted(nls_lob), residuals(nls_lob), pch=16, col="dimgrey")
abline(h=0)

plot(Loblolly$height, fitted(nls_lob), pch=16, col="dimgrey")
abline(0,1)
```

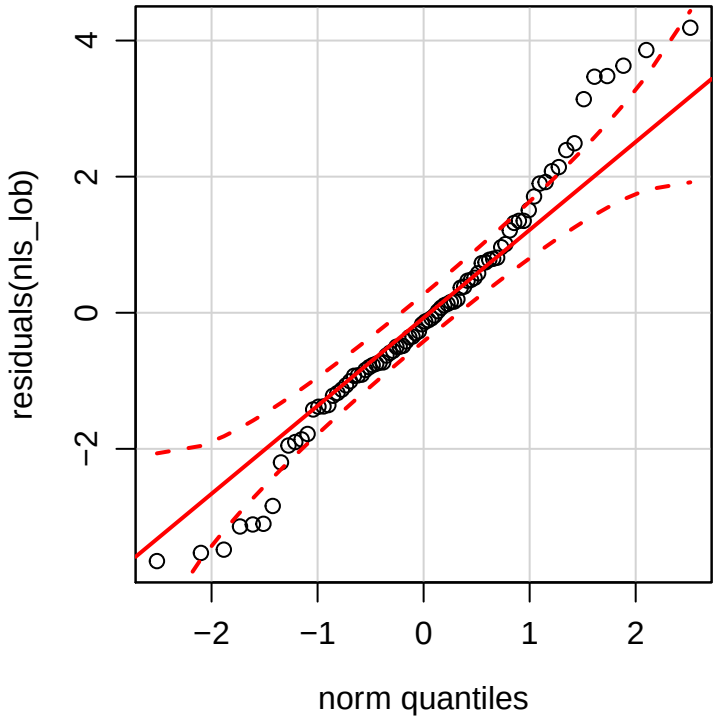

Figure 1.4: Normal QQ plot for the Chapman-Richards growth equation fitted to the Loblolly dataset.

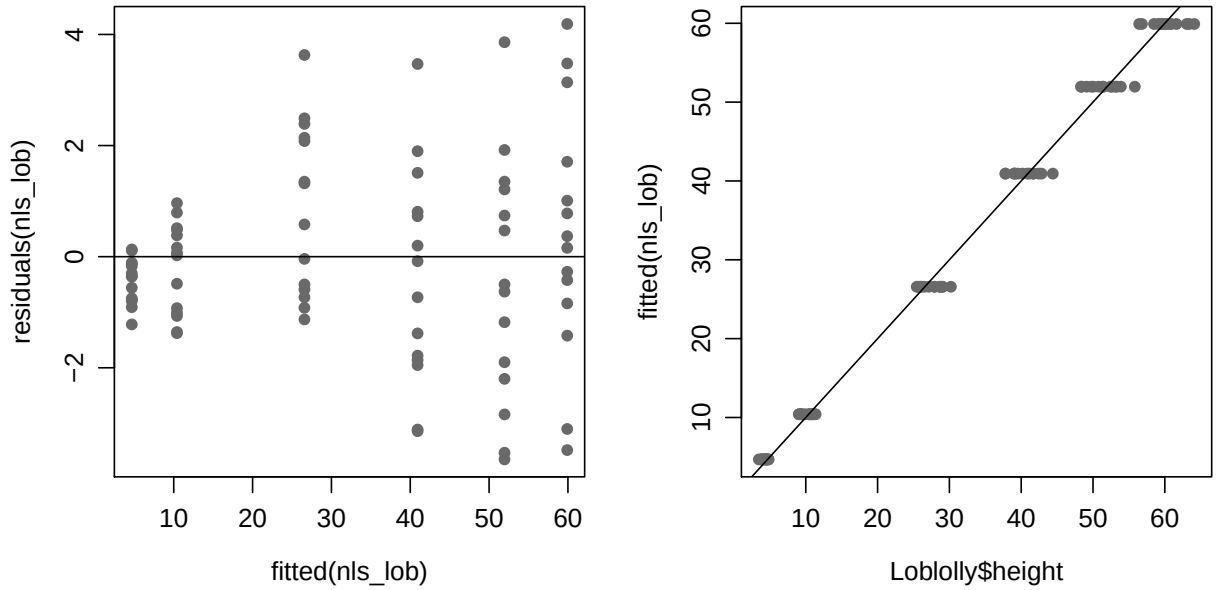

Figure 1.5: Residuals against fitted values, and fitted values against the predictor for a non-linear regression model (nls) fit to the Loblolly data.

Goodness of fit

It is well known that the R^2 is a very poor measure of goodness of fit in non-linear models[1]

Better is the RMSE (the residual standard error), which is returned by `glance`, like so. The RMSE can be roughly interpreted as the standard deviation of the residuals ('roughly' because it is adjusted slightly for the number of fitted parameters).

```
glance(nls_lob)$sigma
```

```
## [1] 1.730971
```

1.2.6 Comparing models

Thus, when comparing different models *fit to the same dataset*, we can use the RMSE as a basis for comparing their relative goodness of fit. Probably even better (though in practice usually equivalent), we can use Akaike's Information Criterion (AIC) as a basis for ranking model fit. This is quickly done with `AIC`,

```
AIC(model1, model2, model3)
```

The model with the lowest AIC can be deemed the 'best'.

When comparing two (or more) models fit to the same dataset, but with different number of parameters, an F-test can be applied providing a direct test of model superiority. For example,

```
# Chapman-Richard model (equivalent to nls_lob used above)
nls_fit1 <- nls(height ~ Asym*(1-exp(-b*age))^c,
                data=Loblolly,
                start=list(Asym=100, b=0.1, c=2.5))

# Drop the 'c' parameter
nls_fit2 <- nls(height ~ Asym*(1-exp(-b*age)),
                data=Loblolly,
                start=list(Asym=100, b=0.1))

anova(nls_fit2, nls_fit1)
```

```
## Analysis of Variance Table
##
## Model 1: height ~ Asym * (1 - exp(-b * age))
## Model 2: height ~ Asym * (1 - exp(-b * age))^c
##   Res.Df Res.Sum Sq Df Sum Sq F value    Pr(>F)
## 1     82     693.12
## 2     81     242.70  1 450.43  150.33 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F-test in the above `anova` statement shows overwhelming support for a better fit for the model that includes the `c` parameter.

> **Try this yourself** In the example above, confirm that use of `AIC` would give the same conclusions.

[1] e.g. Spiess and Neumeyer 2010, 'An evaluation of R^2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach', BMC Pharmacol.

Correlation between parameters

This is not a problem in model fitting, but it is important that you are aware that the coefficients in a non-linear regression models are often highly correlated. A correlation matrix between the parameters is shown with the `overview` function from the `nlstools` package (see example above).

Try this yourself A nice way to visualize the correlation between parameters is the `ellipse` function from the `ellipse` package. For a fitted nls model, try the following code after setting the `which` argument to two parameters of interest:

```
library(ellipse)
plot(ellipse(nls_lob, which=c("Asym","b")), type='l')

# Optionally, add the estimated coefficients as a point
p <- coef(nls_lob)
points(p["Asym"], p["b"], pch=19)
```

It is also possible to plot all confidence ellipses with the `plotcorr` function, for a quick visualization of correlation and confidence regions of the parameters, like so:

```
plotcorr(summary(nls_lob, correlation=TRUE)$correlation)
```

1.2.7 Adding factor variables

A very common question in non-linear modelling is whether estimated coefficients are different between groups, for example species, treatment, location. It is not straightforward to compare coefficients by group, unfortunately, but we can produce estimates and confidence intervals for each group, and base our conclusions on those.

The following example fits the Gompertz growth model to foot length as a function of age, to ask the question whether foot length growth is different between boys and girls. The divergent curves, and the very different estimated parameters show a clear difference, as the following example demonstrates:

```
# Read data.
foot <- read.csv("anthropometry.csv")

# nlsList does not like missing values. We can set an option there,
# or just remove missing values now
foot <- foot[complete.cases(foot),]

# nlsList is from the nlme package
library(nlme)

# Note the grouping operator '|' to specify to fit the curve by gender
fit_foot_gender <- nlsList(foot_length ~ SSgompertz(age, Asym, b2, b3) | gender,
            data=foot)

# Load nlshelper to allow a tidy statement of the fit
library(nlshelper)
tidy(fit_foot_gender)
```

```
##    group term    estimate   std.error statistic p.value
## 1 female Asym 253.3809781 1.443232852 175.56486       0
## 2 female   b2   0.9444377 0.015901843  59.39171       0
## 3 female   b3   0.8391462 0.004492360 186.79407       0
## 4   male Asym 339.4641987 6.445086818  52.67023       0
```

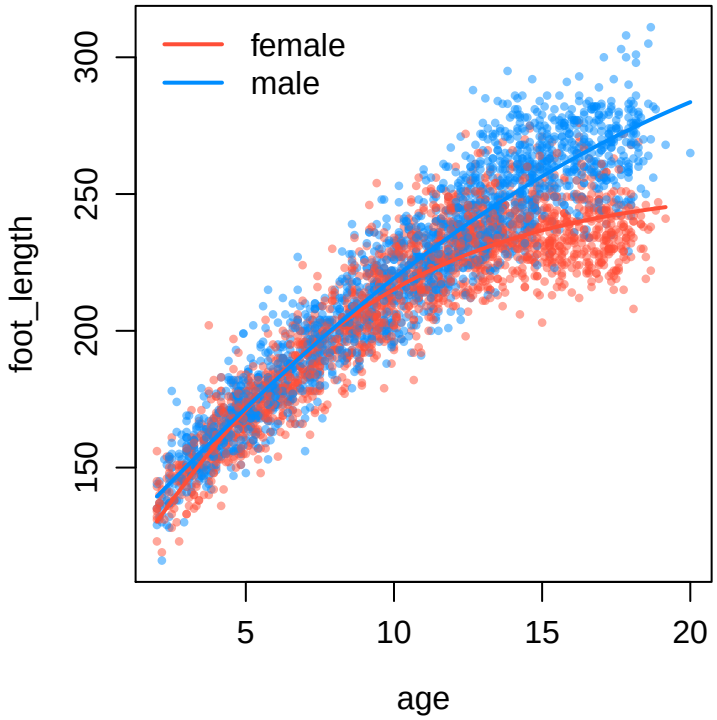

Figure 1.6: The Gompertz growth model fitted to the anthropometry dataset, for males and females separately.

```
## 5   male   b2   1.0629669 0.011426645  93.02528       0
## 6   male   b3   0.9149823 0.003572622 256.10950       0

# Define two colours for the plot
cols <- c("#FF4E37FF","#008DFFFF")
library(scales) # for alpha()

# (from nlshelper package)
plot_nls(fit_foot_gender, pch=16, cex=0.6,
         points.col=alpha(cols, 0.5),
         lines.col=cols,
         lwd=2)
legend("topleft", levels(foot$gender), col=cols, lty=1, bty='n',lwd=2)
```

Using the `nlshelper` package, we can test directly whether adding the grouping variable (`gender` in the above example) improves the fit, that is, whether the relationship between foot length and age differs significantly between genders.

To do so, we need the full model (fit with `nlsList`), and a simple `nls` model without the grouping variable, and then use `anova_nlslist`, like so:

```
fit_foot_0 <- nls(foot_length ~ SSgompertz(age, Asym, b2, b3),
            data=foot)

library(nlshelper)
anova_nlslist(fit_foot_gender, fit_foot_0)

## Analysis of Variance Table
```

```
##
## Model 1: foot_length ~ SSgompertz(age, Asym, b2, b3)
## Model 2: foot_length ~ SSgompertz(age, Asym, b2, b3) | gender
##   Res.Df Res.Sum Sq Df Sum Sq F value    Pr(>F)
## 1   3828     763924
## 2   3825     587806  3 176119  382.02 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Clearly there is plenty of support for gender affecting the relationship, but that should have been abundantly clear from Fig. 1.6!

1.2.8 Predicting

It is easy to predict a fitted value for a new value of the predictor variable with the `predict` function. Note that `predict` is a generic function, so if you want to read the help file when applied to `nls`, read `?predict.nls`.

```
# Make dataframe with X variable that we wish to predict Y values for.
# Make sure it has the same name as in the dataframe we used to fit the model!
newdat <- data.frame(age=c(18, 30))

# Predict from the fitted model for the new dataframe:
predict(nls_lob, newdata=newdat)
```

```
## [1] 47.94869 65.45555
```

Note: Unlike the predict method for linear models (`predict.lm` in particular), we cannot easily estimate standard errors or confidence intervals for predictions (although `predict.nls` does have an argument `se.fit` which is actually ignored!).

> **Try this yourself** For the `nls_lob` model, verify that the estimated asymptote ('`Asym`') is equal to the value of the curve when age is infinity (`Inf`). Extra points if you can write a statement that gives `TRUE`, if this is true.

1.2.9 Confidence intervals

To calculate confidence intervals on the parameters, we can use the base function `confint`, which uses a so-called 'profiling' method.

```
confint(nls_lob)
```

```
## Waiting for profiling to be done...
```

```
##              2.5%       97.5%
## Asym 72.37773289 82.98962856
## b     0.07050617  0.09483768
## c     1.67095139  2.04444265
```

It is important to know that this standard (and preferred) method does not always work, in particular for more complex curves fitted to smaller datasets. In this case, one may use the `confint2` function from the `nlstools` package. A warning here is that these confidence intervals will be approximate, and particularly poor for smaller sample sizes.

```
library(nlstools)
confint2(nls_lob)
```

```
##            2.5 %     97.5 %
## Asym 71.79994733 82.0669900
## b     0.07058745  0.0947305
## c     1.66043435  2.0334445
```

In this particular example the two methods give very similar confidence intervals. Finally, to make a quick table including the fitted coefficients and their confidence intervals, use `coef` like so,

```
cbind(Estimate=coef(nls_lob), confint2(nls_lob))
```

```
##         Estimate       2.5 %     97.5 %
## Asym 76.93346868 71.79994733 82.0669900
## b     0.08265897  0.07058745  0.0947305
## c     1.84693944  1.66043435  2.0334445
```

Even more convenient is the `tidy` function from the `broom` package, but in that case the `confint2` function cannot be used.

```
library(broom)
tidy(nls_lob, conf.int=TRUE)
```

```
##   term    estimate   std.error statistic      p.value    conf.low
## 1 Asym 76.93346868 2.580065660  29.81842 1.957501e-45 72.37773289
## 2    b  0.08265897 0.006067049  13.62425 1.168736e-22  0.07050617
## 3    c  1.84693944 0.093735923  19.70365 1.226880e-32  1.67095139
##     conf.high
## 1 82.98962856
## 2  0.09483768
## 3  2.04444265
```

For models fit with `nlsList`, you can also use the `tidy` function but only after loading the `nlshelper` package:

```
library(nlshelper)

fit_foot <- nlsList(foot_length ~ SSgompertz(age, Asym, b2, b3) | gender,
            data=foot)
tidy(fit_foot, conf.int=TRUE)
```

```
##    group term    estimate   std.error statistic p.value    conf.low
## 1 female Asym 253.3809781 1.443232852 175.56486       0 250.8477773
## 2 female   b2   0.9444377 0.015901843  59.39171       0   0.9153123
## 3 female   b3   0.8391462 0.004492360 186.79407       0   0.8309367
## 4   male Asym 339.4641987 6.445086818  52.67023       0 328.2035127
## 5   male   b2   1.0629669 0.011426645  93.02528       0   1.0419459
## 6   male   b3   0.9149823 0.003572622 256.10950       0   0.9081367
##     conf.high
## 1 256.1612994
## 2   0.9755073
## 3   0.8473201
## 4 353.1134574
## 5   1.0868262
## 6   0.9218169
```

1.3 Local regression

In this section we briefly introduce `loess`, a non-parametric method to fit local regression smoothers to data. This method is useful when you want to illustrate non-linear patterns in the data, but it cannot be used for serious inference.

In a loess model, you have to set the degree of smoothness yourself, via the `span` argument. Thus you can fit a model with an arbitrary smoothness to it, which makes the method less dependable for inference. The following example demonstrates its use.

Here we also show how to extract predictions, calculate a confidence interval for the prediction, and use both in a plot. Note that this is for illustration only, since the `plot_loess` function from `nlshelper` can be used to make plots like this, as a later example demonstrates.

The following code produces Fig. 1.7.

```
# Finish times of the Sydney to Hobart race
hobart  <- read.csv("sydney_hobart_times.csv")

# Fit the local regression model. Note that we always set the span.
loes1 <- loess(Time ~ Year, data=hobart, span=0.6)

# Extract predictions from the model, including the standard errors.
# If we don't give a newdata argument, predictions for the original data
# are returned.
loes1_pred <- predict(loes1, se=TRUE)

# Use transform() to add new variables to the predictions, including
# lower confidence interval (lci), upper (uci), and Year from the original
# dataset.
loes1_pred <- transform(loes1_pred,
                        lci = fit - 2 * se.fit,
                        uci = fit + 2 * se.fit,
                        Year = hobart$Year)

# Make a plot with data, predictions, and confidence interval.
with(hobart,  plot(Year, Time, pch=16, cex=0.7, col="dimgrey", ylim=c(0, max(Time))))
with(loes1_pred, {
  lines(Year, fit)
  lines(Year, lci, lty=3)
  lines(Year, uci, lty=3)
})
```

In general, we use a loess model to visualize patterns in the data. It is also useful to visualize if there is any pattern in the residuals of a non-linear regression, as discussed in Section 1.2.5.

The following example makes Fig. 1.8, using the fitted model `nlsfitSS` (see Section 1.2.4).

```
# In order to use plot_loess, we need to make a dataframe with the variables:
df <- data.frame(resid=residuals(nlsfitSS),
                 fitted=fitted(nlsfitSS))

# Fit the loess model of residuals against fitted.
l <- loess(resid ~ fitted, data=df)

# Plot it, add a horizontal line at 0 behind the data.
```

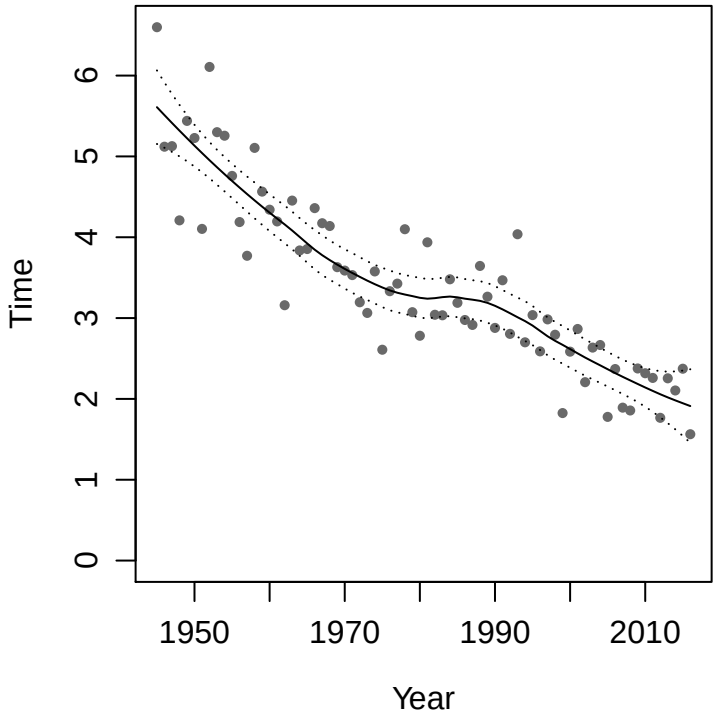

Figure 1.7: A loess fit to the Sydney to Hobart race finish time data, including the confidence interval for the fit.

```
# Assign it to an object; see example below.
rplot <- plot_loess(l,
          panel.first=abline(h=0))
```

Finally, the `plot_loess` function returns the predictions and various other variables. Take a look at the the object `rplot` we saved above:

```
head(rplot)
```

```
##    predvar        fit    se.fit residual.scale       df       lci
## 1 5.580540 -1.2852331 0.3846445       1.687124 79.73111 -2.050740
## 2 6.118038 -1.1124985 0.3632145       1.687124 79.73111 -1.835356
## 3 6.655536 -0.9449240 0.3443013       1.687124 79.73111 -1.630141
## 4 7.193034 -0.7825008 0.3280215       1.687124 79.73111 -1.435318
## 5 7.730532 -0.6252201 0.3144659       1.687124 79.73111 -1.251060
## 6 8.268030 -0.4730729 0.3036838       1.687124 79.73111 -1.077454
##             uci
## 1 -0.5197266111
## 2 -0.3896411122
## 3 -0.2597070854
## 4 -0.1296833954
## 5  0.0006194302
## 6  0.1313084631
```

Where predvar is the X variable at regular spacing, fit the fitted value, se.fit the standard error, df degrees of freedom of the residual, and lci and uci the lower and upper 95% confidence interval for the mean.

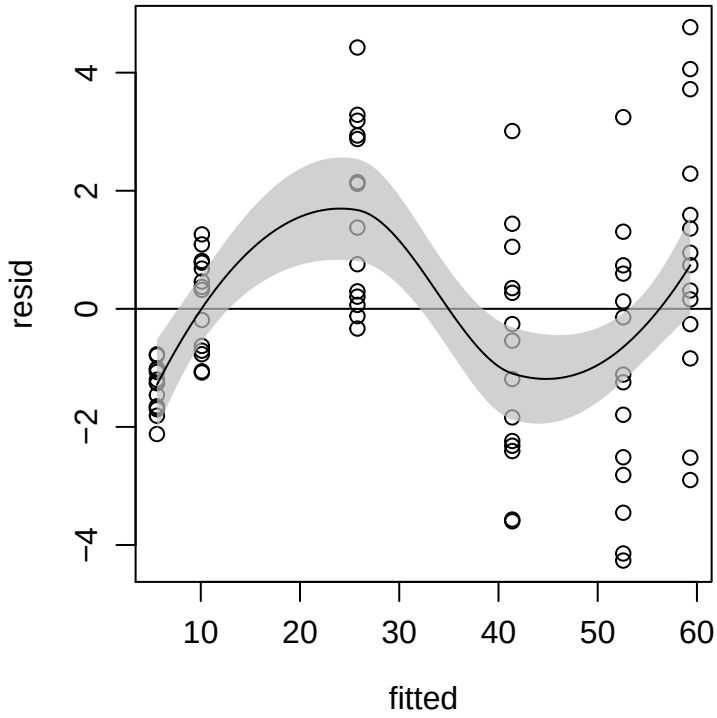

Figure 1.8: Residuals against fitted values for the nlsfitSS model.

1.4 Generalized additive models

In this section we very briefly introduce generalized additive models (GAMs), with the `mgcv` package. Like loess regression, we use GAMs to visualize highly non-linear patterns in the data. However unlike loess, we can use GAMs for inference on smoothness terms, as well as factor variables, and we can even add random effects and account for non-normal errors (hence the 'generalized'). All of these topics are however well outside the scope of this chapter.

The most defining aspect of a GAM is that the smoothness of the model is semi-automatically estimated. Basically, the algorithm finds the smoothest representation that is supported by the data. The user does have to set a 'maximum' degrees of freedom that is to be spent on the smooth terms, but this is generally less influential than the `span` parameter in loess regression. Another aspect is that a number of smoothing models can be chosen, but here we restrict ourselves to 'cubic splines' only.

The following example fits a cubic spline to the Howell height data. Note the use of `by=sex` to fit the spline separately for each sex. We use the `visreg` package as a very convenient method to plot the fitted model.

```
howell <- read.csv("howell.csv")

# It may still be necessary to change k.
library(mgcv)
g <- gam(height ~ s(age, by=sex, k=7), data=howell)

library(visreg)
visreg(g, "age", by="sex", overlay=TRUE, ylab="height (cm)", xlab="age (years)")
```

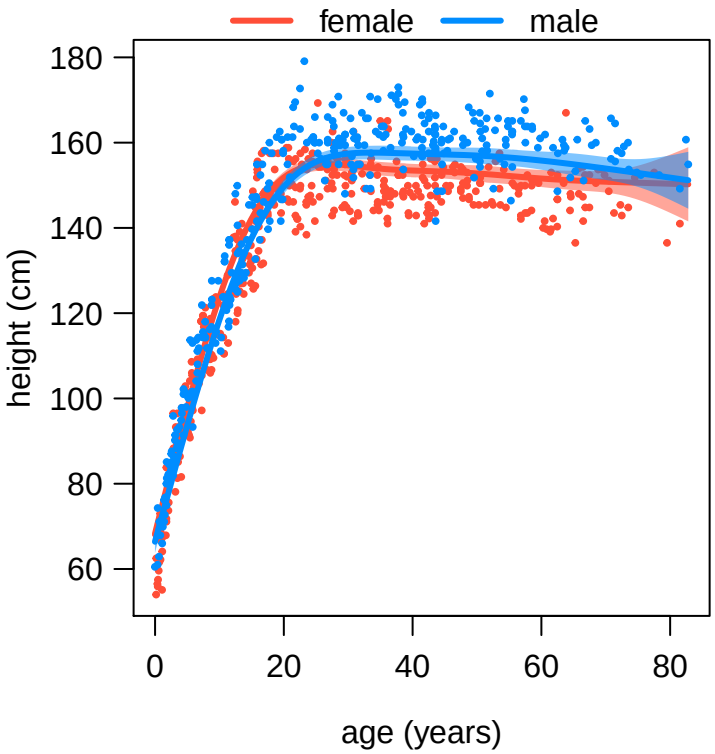

Figure 1.9: Generalized additive model (GAM) to the Howell height data, fit by sex, and plotted with visreg.

Further reading The book by Simon Wood, the developer of the `mgcv` package, is the standard text (Wood, 2006, 'Generalized Additive Models').

Try this yourself You can also fit multiple splines, one for each predictor variable. On the fitted object, you can then use `anova` to see if all or any terms are significant. Try the following example.

```
g2 <- gam(height ~ s(age, k=5) + s(weight, k=5), data=howell)
anova(g2)

# Use visreg to make a 3D plot:
library(visreg)
visreg2d(g2, "age", "weight", plot.type="rgl")
```

1.5 Quantile regression

There are surprisingly many situations where you might be interested in relationships other than the 'mean' relationship, which is what you study with the usual regression techniques.

The next example uses a dataset which includes measurements of plant drought tolerance (P50, more negative values indicate higher tolerance for drought) and mean annual precipitation for 115 plant species. Although significant, the data show no convincing relationship between the two variables in terms of the mean response. It does seem that the lower end (i.e. the minimum P50) is much more

constrained by mean annual precipitation. We can thus ask the question: does the 10% quantile of P50 increase with precipitation?

We use `rq` from the `quantreg` package to perform quantile regression. The argument `tau` sets the quantile to be estimated, for example if `tau = 0.5`, we would perform 'median regression'.

```
# Read data
choat <- read.csv("Choat_precipP50.csv")

# A basic quantile regression
library(quantreg)
choat_rq <- rq(P50 ~ annualprecip, tau=0.1, data=choat)

# Test for significance, use bootstrapped standard errors
summary(choat_rq, se="boot")
```

```
##
## Call: rq(formula = P50 ~ annualprecip, tau = 0.1, data = choat)
##
## tau: [1] 0.1
##
## Coefficients:
##              Value     Std. Error t value   Pr(>|t|)
## (Intercept)   -8.76629   0.67552  -12.97710   0.00000
## annualprecip   0.00225   0.00028    8.02827   0.00000
```

In the above code, to calculate p-values, we must specify `se="boot"` to `summary.rq`, so that standard errors are calculated with a bootstrap.

As with `lm`, `rq` also returns a slope and intercept when you run `coef(f1)`, and you can also use `abline` to add a line to a plot. We can also fit multiple quantiles at once, by passing a vector of `tau` to `rq`. The result can be added to a plot with a simple `for` loop. The next example makes Fig. **??**

```
with(choat, plot(annualprecip, P50, pch=16, col="dimgrey"))
abline(lm(P50 ~ annualprecip, data=choat))

# fit multiple quantiles
choat_rq_2 <- rq(P50 ~ annualprecip, tau=c(0.1,0.25,0.5,0.75,0.9), data=choat)

# store coefficients (inspect p, it is a matrix)
p <- coef(choat_rq_2)

# Plot
with(choat, plot(annualprecip, P50, pch=16, col="dimgrey"))

# Add the quantiles in thin red lines, with a loop
for(i in 1:ncol(p))abline(p[,i], col="red3")
legend("bottomright", c("10,25,50,72,90%"), col="red3", title="Quantiles")
```

Try this yourself The example above shows how to add multiple quantile regressions to a plot. When you fit a single quantile (as in the first example with `rq`), you can just use `abline` to add the model to a plot. Try this with the `choat_rq` model fitted in the previous example.

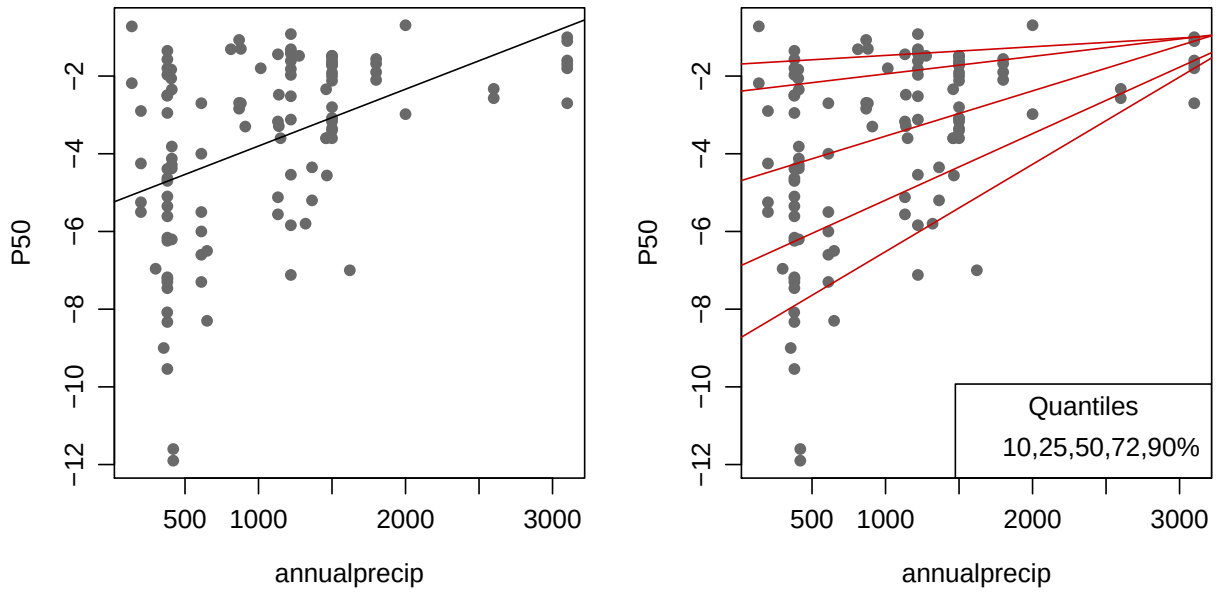

Figure 1.10: The Choat dataset with the mean response fitted (left panel), or a range of quantiles (right panel).

1.5.1 Confidence intervals

In the next example we show how to calculate confidence intervals for quantile regression fits and add it to a plot. For this example we use the Flux tower dataset, and are interested in the upper limit of relative humidity at a given air temperature.

The following code makes Fig. 1.11.

```
# Half-hourly met observations on top of a tower in a forest
flux <- read.csv("Fluxtower.csv")

# Quantile regression of relative humidity on air temperature, estimate the 90% quantile
flux_rq <- rq(RH ~ Tair, tau=0.9, data=flux)

# Set up a 'prediction' dataframe, we will obtain fitted values for these
# values of Tair.
preddfr <- data.frame(Tair=seq(0, max(flux$Tair, na.rm=TRUE), length=101))

# Predict from the fitted quantile regression model, and return a
# confidence interval. Look at the 'type' argument in ?rq for options
# used to calculate the confidence interval (here we use the default).
FH20_90 <- as.data.frame(predict(flux_rq, preddfr, interval="confidence"))
preddfr <- cbind(preddfr, FH20_90)

# Plot symbols
with(flux,plot(Tair, RH, pch=16, col="dimgrey", cex=0.8))

# Add lines
with(preddfr,{
  lines(Tair, fit)
  lines(Tair, lower, lty=3)
```

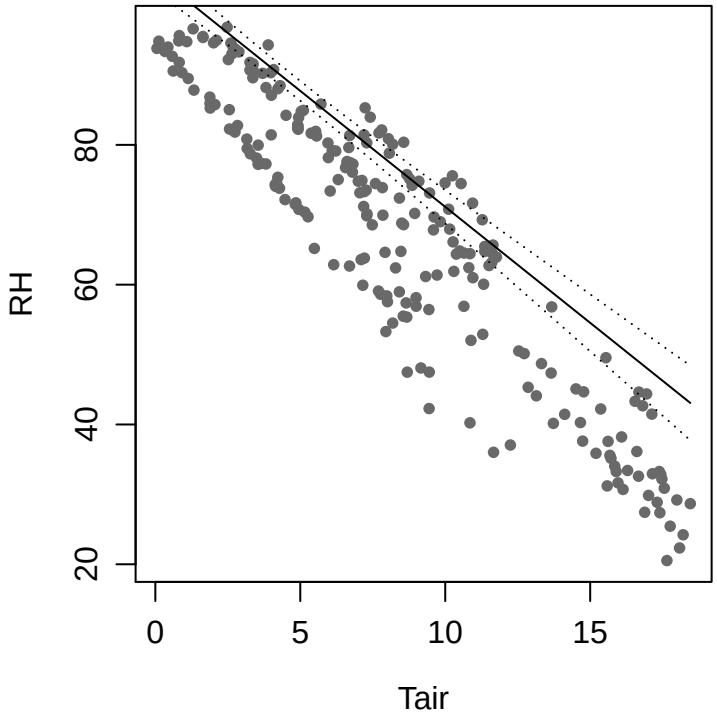

Figure 1.11: Relative humidity (RH) and air temperature (Tair)

```
  lines(Tair, higher, lty=3)
})
```

1.5.2 Non-linear quantile regression

Finally we can combine two techniques introduced in this chapter: non-linear regression and quantile regression. Using `nlrq` from `quantreg`, we can perform non-linear quantile regression.

The following example shows how to estimate quantiles of growth over time - similar to growth charts for childrens' heights - in this case for foot length against age.

First we read and prepare the data, and fit a single model for closer inspection.

```
# for nlrq
library(quantreg)

# The anthropometry dataset, with foot length, age, and sex
foot <- read.csv("anthropometry.csv")
foot <- foot[complete.cases(foot),]

# Use male subset only for this example
foot_male <- subset(foot, gender == "male")

# Fitting an upper quantile, can be interpreted as the 90% quantile of
# foot length at a given age (i.e. only 10% of children would have feet
# longer than this at some age).
foot90 <- nlrq(foot_length ~ SSgompertz(age, Asym, b2, b3),
```

```
              tau=0.9,
             data=foot_male)
```

There are very few things you can do with the fitted model, other than predicting from it (see below, but no confidence intervals!), and summarizing the fit, which tests each coefficient against zero (not very informative in this case, see for yourself: `summary(foot90)`). The coefficients can be extracted with, `coef(foot90)`.

Now we are ready to make our growth chart. To do this, we loop through a few values of `tau`, each time adding a line to a plot.

The following code makes Fig. 1.12.

```
# Plot symbols, very small
with(foot_male,
     plot(age, foot_length, pch=16, cex=0.2, col="grey",
          xlim=c(0,22), ylim=c(100,320)))

# Loop through desired quantiles ...
for(TAU in c(0.1, 0.5, 0.9)){

  # ... use them in a non-linear quantile regression
  fit_mod <- nlrq(foot_length ~ SSgompertz(age, Asym, b2, b3),
                  tau=TAU,
                 data=foot_male)

  # Set up x-values to predict over
  xpred <- seq(min(foot$age), max(foot$age), length=101)

  # And add a line to the plot
  lines(xpred, predict(fit_mod, data.frame(age=xpred)))

  # Label the curve on the right (y-value is predicted from the curve)
  text(21, predict(fit_mod, data.frame(age=max(foot_male$age))),
       labels=as.character(TAU))
}
```

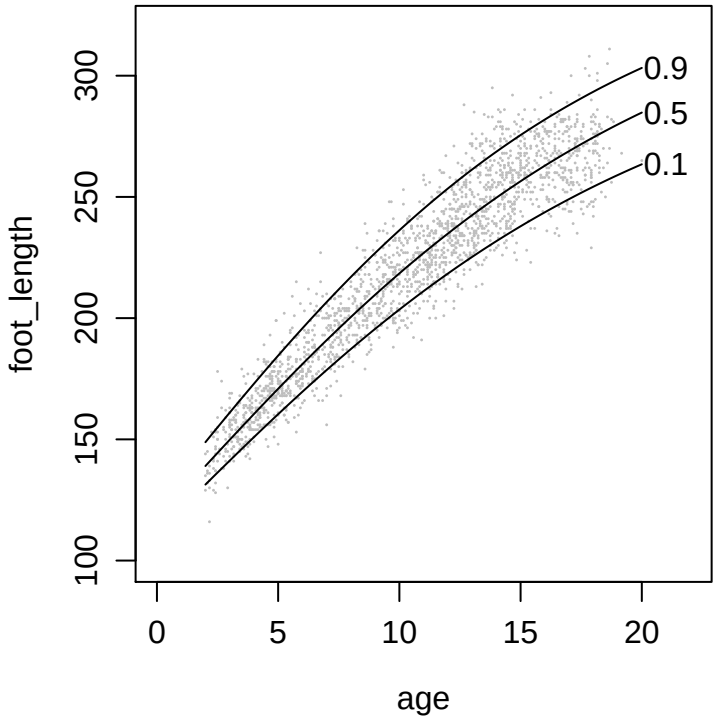

Figure 1.12: Growth chart for foot length (mm) against childrens' age (years)

1.6 Exercises

In these exercises, we use the following colour codes:

- ■ **Easy**: make sure you complete some of these before moving on. These exercises will follow examples in the text very closely.
- ♦ **Intermediate**: a bit harder. You will often have to combine functions to solve the exercise in two steps.
- ▲ **Hard**: difficult exercises! These exercises will require multiple steps, and significant departure from examples in the text.

We suggest you complete these exercises in an **R** markdown file. This will allow you to combine code chunks, graphical output, and written answers in a single, easy-to-read file.

1.6.1 Chick Weight

For this exercise, you will be using the `ChickWeight` data, which is a dataset already available and loaded in **R**. See `?ChickWeight` for a description.

Note that the data are in a special `groupedData` format, for the purpose of this exercise, first convert the data to a standard dataframe:

```
chickweight <- as.data.frame(ChickWeight)
```

1. Make a subset of ChickWeight, for Chick number 6, and exclude Time equals zero (i.e. `Time > 0`). Inspect the data, and make a simple line plot of weight over time for this chick.

2. For this subset, fit the Weibull, logistic, and Gompertz growth models to chick weight against time. Each of these models is defined as a self-starting model, thus no starting values have to be guessed.
3. Which of the three models fits the data best? Decide based on the AIC. Also make a plot of the fitted curves on the data, either as separate panels, or all added to the same plot with the argument `add=TRUE` to the `plot_nls` function.
4. For the best-fitting model, predict the weight of the chick at time 25 (see Section 1.2.8).

1.6.2 Michaelis-Menten kinetics

For this example we use the built-in dataset `Puromycin`, which can be accessed without loading a package. The data include reaction velocity (`rate`) versus subtrate concentration in an enzymatic reaction for cells treated with the antibiotic Puromycin, or an untreated control.

For enzymatic reactions that depend on the concentration of the substrate, the Michaelis-Menten model is often used, and follows from simple assumptions on the reaction rate versus the concentration of the substrate and enzyme.

1. Plot `rate` against `conc`, and colour the symbols by `state` (treated or untreated). Confirm visually that taking the log of both sides does not linearize the relationship.
2. Fit the Michaelis-Menten model to the whole dataset, using `nls` and `SSmicmen`. Inspect the summary, make a plot of the data with the fitted curve, and extract the confidence interval of the fitted parameters.
3. Now fit the curve by `state`, using `nlsList`. Perform an F-test comparing the model with and without `state`, using `anova_nlslist` (see Section 1.2.7). Is there evidence for `state` affecting the reaction velocity?
4. Make a plot of the `nlsList` model with `plot_nls`, make sure the y-axis limit is large enough to include the `Vmax` estimated for each level of `state`. Now add the estimated `Vmax` parameters to the plot as horizontal lines, with `abline(h=...)`.

1.6.3 Brunhilda the radioactive baboon

The data used in this exercise were extracted from http://www.statsci.org/data/general/brunhild.html. The observed responses are Geiger counter counts (times 10-4) used to measure the amount of radioactively tagged sulfate drug in the blood of a baboon named Brunhilda after an injection of the drug. The variables are `Hours` (time since injection), `Sulfate` (Geiger counter counts).

One researcher claims that a simple exponential decay function should describe the data well:

$$Y = A0 + A * exp(-k * X)$$

where Y is the response variable, X the predictor, and `A0`, `A` and `k` are parameters to be estimated.

A second researcher does not believe her and claims a bi-exponential decay is necessary, which is a mixture of two simple exponential decays (see `?SSbiexp` for details).

1. Follow the link above, and download the data file ('brunhild.txt'), place it in your working directory. Read it in with `read.delim` (it is a TAB delimited file).
2. Fit the simple exponential decay function defined above to the data. You may have to set reasonable starting values. When setting the starting values, consider what the `A0` parameter represents.

Also don't forget to plot the data. *Hint:* if you have trouble finding starting values that make the model converge, try `A0=4, A=10, k=0.5`.

3. Plot the data with the fitted curve. Note that an asymptote is reached, confirm that one of the parameters in the equation gives the value of the asymptote (and add it with a dashed line, using `abline`).
4. Fit the bi-exponential model (`SSbiexp`), plot the data with the curve.
5. Compare residual plots vs. fitted values, RMSE, and AIC of the two models.
6. Perform an F-test on the two models to test whether the added complexity of the bi-exponential model is supported by the data. *Hint*: use `anova`.

1.6.4 Weight loss

Consider a dataset of sequential measurements of a person's weight while on a diet (the 'weightloss' dataset).

Read the dataset ('weightloss.csv'), call it `weightdat`, and convert the 'Date' variable to the `Date` class, like this:

```
weightdat$Date <- as.Date(weightdat$Date, format="%d/%m/%y")
```

In order to use the `loess` function, you must next convert `Date` to a numeric variable. Run the following code to convert it to days since the start of measurement:

```
weightdat$Days <- with(weightdat, as.numeric(Date - min(Date)))
```

1. ■ Fit a loess regression model of Weight against Days. Try various values of `span`, and decide on a value that fits the data well without overfitting.
2. ♦ Plot the fitted model, and assign it to an object (see the bottom of Section 1.3 for details). This object contains the fitted values; inspect the object returned by `plot_loess`.
3. ▲ With the object from above, calculate the weight loss rate in pounds per day, using `diff`. Note that `diff` returns a vector that is one element shorter than the original vector (because it calculates successive differences). Make a plot of weight loss rate vs. Days. How sensitive is the calculated weight loss rate to the value of `span` chosen in the above?
4. ■ Fit a generalized additive model to the same dataset, plot the resulting object with visreg. Is the fit similar to the loess fit? What happens when you change the `k` parameter?

1.6.5 Race quantiles

For this exercise you will use the Sydney to Hobart race finish times, see Section 1.3. There is some indication from the data that the spread of finish times has decreased over time. We will test this idea using quantile regression, ignoring the observation that the finish times did not exactly linearly decrease over time

1. ■ Read the data, fit a linear regression model and a median regression model. The latter is achieved via quantile regression with `tau = 0.5`. Compare the fits by plotting the data and adding both fitted models with `abline`.
2. ■ Now fit two quantile regression models, at 5% and 95%. Plot the data and add both lines.
3. ▲ Now calculate the 90% range of the data by calculating the difference between the 95% and 5% quantiles for each `Year`. To do this, setup a dataframe with the years for which you want to make the calculations (a sequence from min to max Year in the dataset), and calculate the

predictions from both models from the previous exercise. Then plot the difference vs. Year. Inspect Section 1.5.1 for using `predict` with fitted quantile regression models (but now you can set `se=FALSE`).

Index