

Chapter 1

A First Tour through R by Example

1.1 Getting started

In this chapter we illustrate how R can be used to explore and analyze psychophysical data. We examine a data set from the classic article by Hecht, Shlaer and Pirenne, *Energy, quanta and vision* [79] in order to introduce basic data structures and functions that permit us to examine and model data in R.

The best way to read this book is sitting at a computer with R running, typing in the examples as they occur. If you do not already have R installed on your computer, see Appendix A for instructions on how to install it. If you have properly installed R and launched it by double-clicking on an icon or by running the command “R” from a terminal command line, you should see some initial information printed out about the version followed by some initial suggestions of things to explore, `demo()`, `help()`, etc. This is followed by “>” on a line by itself. This is the *command* line prompt and indicates where you type in commands. If you enter a new line (carriage return) before the end of a command, the prompt will become a “+”, indicating a continuation of the previous line. You will continue to receive continuation lines until the command is syntactically complete, at which point R will attempt to interpret it.¹ Comments in R are preceded by a “#”. Everything on the line to the right of a comment character is ignored by R. Let’s proceed with the example.

1.2 The experiment of Hecht, Shlaer & Pirenne

In a classic experiment, Hecht, Shlaer and Pirenne [79] estimated the minimum quantity of light that can be reliably detected by a human observer.

¹ Syntactically complete commands may still be incorrect, in which case an error message will be returned.

In their experiment, the observer, placed in a dark room, was exposed at regular intervals to a dim flash of light of variable intensity. After each flash presentation, the observer reported ‘seen’ or ‘unseen’. The data were summarized as the percentage of reports ‘seen’ for each intensity level used in the experiment, for each observer.

Before collecting these data, Hecht, Shlaer and Pirenne performed extensive preliminary analyses in order to select viewing conditions for which human observers are most sensitive (requiring the fewest number of quanta to detect) that are described in their article, and a detailed analysis of their choices is presented in the opening chapter of Cornsweet’s text [41].

1.2.1 Accessing the data set

The data from their main experiment (Table V in [79]) can be found in the package **MPDiR** that contains data sets used in this book and that are not available in other packages [98]. The package is available from the CRAN website (<http://cran.r-project.org/>). If you are connected to the internet, then it can be installed from within R by executing the function

```
> install.packages("MPDiR")
```

Thereafter, to have access to the data sets, the package has to be loaded into memory and attached to the *search path* using the function:

```
> library(MPDiR)
```

The search path is a series of environments (see Sect. A.2.5) in memory in which sets of functions and data objects are stored, for example, those from a package. When a function is called from the command line, R will look for it in the succeeding environments along the search path. The `library` function attaches the package to the second position on the search path, just behind the work space, called `.GlobalEnv`. We verify this using the `search` function.

```
> search()

[1] ".GlobalEnv"          "package:MPDiR"
[3] "tools:RGUI"          "package:stats"
[5] "package:graphics"   "package:grDevices"
[7] "package:utils"       "package:datasets"
[9] "package:methods"    "AutoLoads"
[11] "package:base"
```

which, in this case, returns an 11-element vector of character strings. The specific packages you see will depend on your version of the language R and may not be the same as those shown above.

The data set is called **HSP** and can now be loaded into the work space by:

```
> data(HSP)
```

and we can check that it is there by typing

```
> ls()
```

```
[1] "HSP"
```

which returns a vector of the names of the objects in your work space. The data set corresponds to a single object, `HSP`. The function `data` is used to load data sets that are part of the default R distribution or that are part of an installed and, as here, loaded package. A second argument indicating a package name can also be used to load data from unloaded but installed packages (see `?data`). In Appendix A we discuss how to load data from different sources into the workspace.

R has a rich set of tools for exploring objects. Simply typing the name of an object will print out its contents. This is not practical, however, when the data set corresponding to the object is large. We can view the internal structure of the object `HSP` using the function `str`. This extremely useful function allows us to examine any object in R (see `?str` for more details).

```
> str(HSP)
```

```
'data.frame':      30 obs. of  5 variables:
 $ Q  : num  46.9 73.1 113.8 177.4 276.1 ...
 $ p  : num   0 9.4 33.3 73.5 100 100 0 7.5 40 80 ...
 $ N  : int   35 35 35 35 35 35 40 40 40 ...
 $ Obs: Factor w/ 3 levels "SH","SS","MHP": 1 1 1 1 1 1 1 ..
 $ Run: Factor w/ 2 levels "R1","R2": 1 1 1 1 1 1 2 2 2 ..
```

This shows that `HSP` is an object of class “data.frame”. A data frame is a type of list (Appendix A), a data structure that can contain nearly arbitrary components, in this case five of them. Their names are indicated after each of the ‘\$’ signs. What distinguishes a data frame from a list is that each component is a vector, and all the vectors must have the same length. The entries in each vector have a `class` (‘numeric’, ‘integer’, ‘logical’, ‘Factor’, etc.) indicated next to each component.

In the data set `HSP`, the first few elements of each component are indicated to the right of the class designation. This lets us quickly verify that data are stored in the correct format. The data set includes two components of class ‘numeric’. `Q` is flash intensity level measured in average number of quanta. `p` is the percentage of responses ‘seen’ for that level. `N` is of class “integer” and indicates the number of flash presentations. `Obs` and `Run` are both factor variables encoding categorical information as integer levels. For human consumption, the levels are given labels, which are indicated in the output. The levels of `Obs` are the initials of the three observers, who were the three authors of the article. For two observers, data are available for a replication of the experiment, indicated by the factor `Run`.

Even with `str`, R objects with many components can be impractical to examine. In that case, it may be more convenient simply to list the names of the components, as in:

```
> names(HSP)
[1] "Q"  "p"  "N"  "Obs" "Run"
```

We can use this function, also, on the left side of an assignment operator, `<-`, to modify the names of the data frame, which we will take the opportunity to do here to render the meaning of some of the components more obvious.

```
> names(HSP) <- c("Quanta", "PerCent", "N", "Obs", "Run")
```

Here, we have used the function `c()` to create a five element vector of class ‘character’ and stored it as the names attribute of the columns of the data frame. The change in names occurs only on the copy of `HSP` in our work space, *not* on the original version stored in the package. Note that the assignment operator, “`<-`”, is entered with 2 keystrokes.²

The first 10 lines of the data frame can be displayed by:

```
> head(HSP, n = 10)
      Quanta PerCent  N Obs Run
1      46.9      0.0 35 SH  R1
2      73.1      9.4 35 SH  R1
3     113.8     33.3 35 SH  R1
4     177.4     73.5 35 SH  R1
5     276.1    100.0 35 SH  R1
6     421.7    100.0 35 SH  R1
7      37.1      0.0 40 SH  R2
8      58.5      7.5 40 SH  R2
9      92.9     40.0 40 SH  R2
10    148.6     80.0 40 SH  R2
```

Notice the modified column names. Note, also, that inside a function argument, we use “`=`” to bind the value to the formal argument `n`, that specifies the number of lines of output to print. Use of “`<-`” here is not advised as it would produce the side-effect of creating a variable `n` in the work space (See Problem 1.4). There is also a function `tail` that does what you might expect.

Another useful view of the data is obtained with the `summary` function.

```
> summary(HSP)
```

² The more intuitive “`=`” symbol is also valid for assignment at the command line, but the two symbols are not interchangeable in all situations, so we recommend using “`<-`” on the command line.

```

      Quanta      PerCent      N      Obs
Min.   : 24  Min.   : 0.0  Min.   :35  SH :12
1st Qu.: 58  1st Qu.: 6.4  1st Qu.:40  SS :12
Median : 93  Median : 42.0  Median :50  MHP: 6
Mean   :138  Mean   : 48.4  Mean   :45
3rd Qu.:210  3rd Qu.: 94.0  3rd Qu.:50
Max.   :422  Max.   :100.0  Max.   :50
Run
R1:18
R2:12

```

Summary statistics are displayed for numerical and integer components. The levels of factor components are tabulated. Thus, the imbalance in the data set is clearly indicated by the unequal numbers of observations per `Obs` and per `Run`. Such tabulations can also be generated with the `table` function.

```
> with(HSP, table(Run, Obs))
```

```

      Obs
Run  SH SS MHP
R1   6  6  6
R2   6  6  0

```

Here, we have also introduced the handy function `with` that creates a temporary environment within which the component names of its first argument are visible.

1.2.2 Viewing the data set

A graphical representation is often effective for gaining a sense of the data. When a data frame is used as the argument of the function `plot`, R creates a square array of scatterplots, one for each pair of columns in the data frame.

```
> plot(HSP)
```

The resulting plot for the `HSP` data set, shown in Fig. 1.1, is called a *pairs plot*. The column names are conveniently displayed along the diagonal. The scatterplots above the diagonal are based on the same pair of variables as those below but with the axes exchanged. The scatterplot at position (2,1) in the square array displays the variables of greatest experimental interest, the proportion of seen flashes as a function of average number of quanta in the flash, the frequency of seeing data. Although the values from all observers and runs are confounded in this plot, the basic trend in the data is evident: the frequency of seeing is at 0% for the lowest levels of the flash and increases to asymptote at 100% as the flash intensity increases.

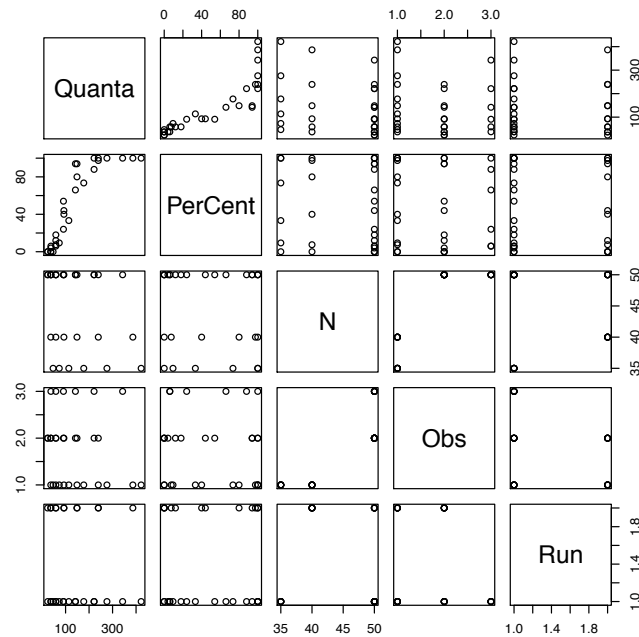


Fig. 1.1 Example of a pairs plot obtained by using a data frame as the argument to the `plot` function.

Three other scatterplots display useful features of the data. The graph at position (3,4) shows that the number of presentations of each flash, N , is the same for two observers, but differs for the third one. Examining the scatterplot just to its right, at position (3,5), shows that this observer completed different numbers of trials in two runs. Finally, the graph just below, position (4,5), reveals that two of the observers completed two runs and a third only one. Pairs plots are also useful for detecting errors in recording data or coding factors.

This example demonstrates the object oriented nature of R. The `plot` function is an example of a *generic function*. Associated with a generic function is a set of method functions that are tailored to operate on specific classes of objects. In the case of a data frame, the generic `plot` function dispatches the execution to a plot method for data frames, called `plot.data.frame`, that produces the pairs plot. Other classes of objects will result in dispatch to plot methods specific to their class and result in plots that are meaningful for that class of object. We will see numerous examples of using the `plot` function on different kinds of R objects in subsequent chapters, including the case of a simple scatterplot. To see the methods that are available for `plot`, try

```
> methods(plot)
```

The `str` and `summary` functions used above are other examples of generics, and we will exploit many more throughout the book.

1.2.3 One more plot before modeling

As indicated above, the frequency of seeing data in the pairs plot confound the variables `Obs` and `Run`. It could be useful to examine the individual plots for irregularities across these conditions. The `lattice` [156] package has functions that quickly generate a series of plots of variables across the levels of a factor (or factors). It is among the nearly 30 packages that are part of the base distribution of R; you do not normally have to install it. To load it, we use the `library` function. The following code fragment creates Fig. 1.2.

```
> library(lattice)
> xyplot(PerCent/100 ~ Quanta | Obs + Run, data = HSP,
+   xlab = "Quanta/flash", ylab = "Proportion \"seen\"",
+   scales = list(x = list(log = TRUE, limits = c(10, 500)),
+     at = c(10, 20, 50, 100, 200),
+     labels = c(10, 20, 50, 100, 200))),
+   skip = c(rep(F, 5), T), layout = c(3, 2, 1),
+   as.table = TRUE)
```

The `lattice` function `xyplot` enables the efficient generation of scatterplots across the factor levels. The key feature in the `lattice` plotting commands is the specification of the variables as a formula. In this case, we specify that `PerCent` divided by 100 will be plotted as a function of `Quanta` for each combination of the levels of the factors `Obs` and `Run`. The first argument to `xyplot` is an example of a formula object. We will look at more examples of formula objects in the next chapter. For now, the tilde, ‘~’, can be interpreted here as ‘is a function of’ and the vertical bar, ‘|’, separates the functional relation from the factors on which it is conditioned. The `data` argument specifies the data frame within which the terms in the formula can be found, i.e., each of the terms in the formula corresponds to the name of a column of the data frame. Lattice plotting functions produce a default format that has been chosen to yield generally pleasing results in terms of information display. It can be easily customized. The rest of the arguments just control details of the formatting of the graphs and their layout. The `scales` argument is used to set the tick positions and labels in the figure, the `skip` argument prevents an empty graph from being displayed for observer MHP who only performed one run, and the `layout` and `as.table` arguments control the arrangement of the graphs. These additional arguments are not absolutely necessary. We include them to demonstrate that one can control fine details with the command but you should try to redo the graph without them to see what happens.

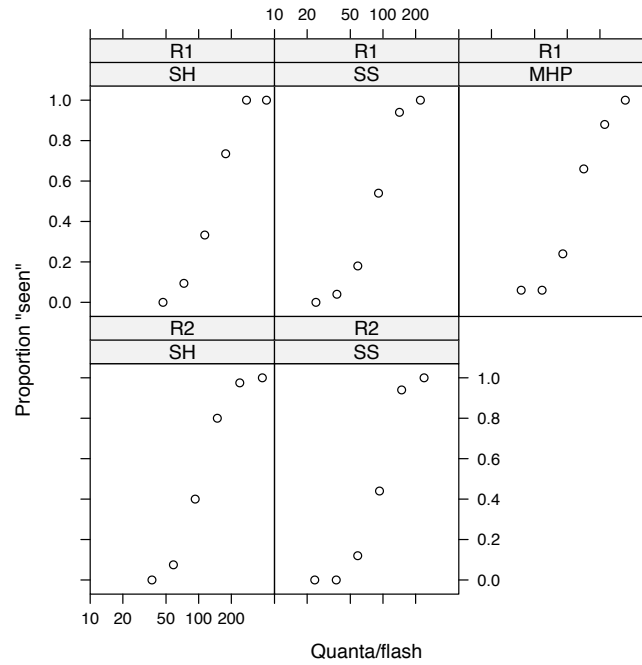


Fig. 1.2 Frequency of seeing data for each observer and run as a function of the average number of quanta/flash. The data are displayed as a lattice.

The factor levels are indicated in strips above each graph, allowing easy identification of the condition. Notice that the abscissa has been specified to be logarithmically spaced. The crude scatterplot of `PerCent` versus `Quanta` in Fig. 1.1 was plotted on a linear scale. The main tendencies noted in the pairs plot are still present in the individual graphs, however.

1.3 Modeling the data

Our aim is to fit the frequency of seeing data displayed in Fig. 1.2 with a psychometric function. The psychometric function, for our purposes, is a smooth underlying curve describing the probability of detecting a flash as a function of flash intensity. The curve is usually part of a parameterized family of functions and we seek the parameter values that specify a curve that best matches the data. As we will see in Chap. 4, there are several possibilities as to the choice of functional form to fit the data. For our example, we

will use a Gaussian cumulative distribution function (see Sect. B.2.5).³ This corresponds to a probit analysis [60] and has the advantage that it can be performed simply in R as a Generalized Linear Model (GLM) [127] with the `glm` function, specifying a binomial family. We postpone discussing the GLM until Chap. 2 and Chaps. 4 and 5 will describe how we use the GLM to fit psychometric functions in detail.

Our goal is to fit a model that best predicts the probability that the observer will respond “seen” (or from here on, “Yes” for “I saw the stimulus on that trial”) when a stimulus of intensity Q (log quanta) is presented

$$P[\text{Yes}] = \Phi\left(\frac{Q - Q_{.5}}{\sigma}\right). \quad (1.1)$$

The function Φ maps the physical quantity in parentheses to the range $(0, 1)$. It is typically a cumulative distribution function for a random variable (Sect. B.2.3), here the cumulative distribution function of a Gaussian (normal) random variable with mean 0 and standard deviation 1. We allow ourselves two free parameters to fit data. The first is $Q_{.5}$, the intensity that results in the observer responding Yes with probability 0.5. This measure is a convenient characterization of the observer’s sensitivity. The second parameter σ controls how steep the fitted psychometric function will be. The slope of the Gaussian psychometric function at $Q_{.5}$ is inversely proportional to σ . We rewrite the function above as

$$\Phi^{-1}(E[R]) = \frac{Q - Q_{.5}}{\sigma} = \beta_0 + \beta_1 Q, \quad (1.2)$$

where R denotes the observer’s response on a trial coded so that it takes on the value 1 if the observer responds Yes and 0 if the observer responds No and Q is the intensity of the stimulus.

The expected value of R is, of course, the probability that the observer responds Yes. We recode the variable in this way because the GLM is used to predict not the observer’s exact response (which is 0 or 1) but its expected value across many trials. The fixed function Φ is replaced by its inverse on the left hand side of the equation. The inverse of a cumulative distribution function is called a *quantile function* (see Sect. B.2.3). Moving Φ from one side of the equation to the other is simply cosmetic. It makes the resulting equation look like a textbook example of a GLM. We then replaced the parameters $Q_{.5}$ and σ by β_0 and β_1 where it is easy to see that $\beta_0 = -Q_{.5}/\sigma$ and $\beta_1 = \sigma^{-1}$. Again, we are simply increasing the resemblance of the resulting equation to a textbook example of a GLM. We will use the GLM to estimate β_0 and β_1 and then solve for estimates of $Q_{.5}$ and σ . In the resulting equation, the right-hand side is called the *linear predictor* and looks like a linear regression equation,

³ For theoretical reasons, Hecht et al. chose a different function, a cumulative Poisson density. We will return to consider fitting the data with their model in Sect. 4.2.1.

the linear model (LM) of which the GLM is a generalization.⁴ We discuss the GLM in detail in the following chapter. For now it is enough to recognize that Eq. 1.2 is an example of a GLM and that we can estimate parameters using the R function `glm`. Next we set up the GLM by first specifying a formula object.

In the current problem, the data are specified as the percent correct for N trials at each intensity. It would be equivalent for our analyses to recode them as N 1's and 0's, indicating the trial-by-trial responses, with the number of 1's equal to the number of successes. We recode the percent correct as a two-column matrix indicating the numbers of successes and failures. The resulting counts of successes are binomial random variables (see Sect. B.3.2) with means equal to $N_i E[R_i]$ for each stimulus intensity, i . We calculate these values from the data and add two new columns to the data frame:

```
> HSP$NumYes <- round(HSP$N * HSP$PerCent/100)
> HSP$NumNo <- HSP$N - HSP$NumYes
```

To add the new columns, we just specify them with the list operator `$`.

We could reduce the number of times that we write `HSP` by using `within` which works like `with` but makes changes within the context of a specified data frame and returns the entire, modified data frame. Then the above code fragment becomes,

```
> HSP <- within(HSP, {NumYes <- round(N * PerCent/100)
+   NumNo <- N - NumYes })
```

Notice that we still need to assign the result back to the variable in memory to make the change.

We rounded the data to guarantee that the resulting counts of Yes and No responses are integers.⁵

1.3.1 Modeling a subset of the data

There are data for five frequency of seeing curves in HSP. To begin, we will only fit one of them. We extract a subset of the data, the first run of observer SH, using the `subset` function.

```
> ( SHR1 <- subset(HSP, Obs == "SH" & Run == "R1") )
```

⁴ The reader may be wondering whether fitting estimates of parameters in one form such as β_0, β_1 and then transforming the estimates to estimates of the parameters we want (Q_5 and σ) is legitimate. It is. See Sect. B.3.3.

⁵ Interestingly, there is an error in Hecht et al.'s table, pointed out to us by Michael Kubovy. The entries in the resulting counts of Yes and No responses for one of the conditions are not close to integers. It is possible that the number of trials for this observer was not the same at all intensity levels.

	Quanta	PerCent	N	Obs	Run	NumYes	NumNo
1	46.9	0.0	35	SH	R1	0	35
2	73.1	9.4	35	SH	R1	3	32
3	113.8	33.3	35	SH	R1	12	23
4	177.4	73.5	35	SH	R1	26	9
5	276.1	100.0	35	SH	R1	35	0
6	421.7	100.0	35	SH	R1	35	0

Enclosing the line in parentheses causes the result to be printed out.

The second argument of `subset` creates a logical vector that determines the choice of rows to be those for which the observer is SH and the run is R1. An alternative method for obtaining a subset of the data frame uses the fact that data frames can be indexed like matrices using square brackets (see Appendix A). We explicitly specify which row entries and which column entries (all of them, signified by leaving the second coordinate blank) to select using a logical vector

```
> with(HSP, HSP[Obs == "SH" & Run == "R1", ])
```

A third method of subsetting is possible, using the `subset` argument that many modeling functions, such as `glm`, include that would exploit the same logical expression. Which method to use here is largely a matter of convenience.

The call to `glm` requires three arguments: a formula object, a family object and the data frame to use in interpreting the terms of the formula. Hecht et al. used the logarithm of the covariate `Quanta` and so do we

```
> SHR1.glm <- glm(formula = cbind(NumYes, NumNo) ~ log(Quanta),
+               family = binomial(probit), data = SHR1)
```

The first argument is the formula object, a specification of how we are modeling the observer's responses in a short-hand language used by most R modeling functions. The formula object expresses the relation between the response variable on the left hand side of `~` and the linear predictor on the right hand side. We saw an example of such a formula object as the first argument to `xyplot` above. Here, the tilde can be read as "is modeled as".

The response term in the function call combines the two columns, `NumYes` and `NumNo`, into a 6×2 matrix using the function `cbind`. The term `log(Quanta)` indicates the covariate with the intercept implicit in the model formulation. With the family argument, we first specify that our responses are distributed as a binomial variable. The function `binomial` takes one argument that indicates the form of the psychometric function. The argument is called the link function. The default link function for the binomial family is the log odds ratio or logit transformation, so we must specify the probit (Gaussian) transform explicitly. The `data` argument indicates the data frame within which the column vectors corresponding to the terms in the formula are defined.

When arguments are named, as here, they may be specified in any order. Because the arguments are in the default order specified for the `glm` function

(verify by typing `help(glm)` or `?glm`), we could also leave out the names. Including them makes the code easier to read. The result returned by `glm` is a model object of class “glm” assigned to a variable named `SHR1.glm` that we will begin to examine now.

1.3.1.1 Examining the model object

Typing in the name of the model object (or, for that matter, any object in R) invokes its `print` method, equivalent to typing in `print(SHR1.glm)`:

```
> SHR1.glm
Call:  glm(formula = cbind(NumYes, NumNo) ~ log(Quanta),
          family = binomial(probit),  data = SHR1)

Coefficients:
(Intercept)  log(Quanta)
      -13.02         2.67

Degrees of Freedom: 5 Total (i.e. Null);  4 Residual
Null Deviance:      185
Residual Deviance: 2.78      AIC: 17.4
```

The result is a summary of basic information about the fit. The first line displays the function call that created the model object. The coefficients are the intercept and slope for the covariate from (1.2), i.e., β_0 and β_1 , respectively. Information on the number of degrees of freedom and measures related to the goodness of fit are also given that we will discuss further below.

R provides powerful methods for comparing different models of the same data. Deviance is a measure related to the log likelihood of a model (See Sect. B.4.2), and is used to assess goodness of fit. We use it to compare complex models to simpler versions of the same model. The printed results include the *residual deviance* that is for the fitted model with intercept and slope terms, β_0 and β_1 , respectively, and the *null deviance*, for a simpler model with just the intercept term β_0 , a special case of the first model with β_1 forced to be 0. The comparisons of the deviances of the models with and without each term lead to nested hypothesis tests (see Sect. B.5).

Finally, the AIC, or Akaike Information Criterion, is a second measure of goodness of fit useful for comparing several models that attempts to take into account model complexity (indicated by the number of parameters) as well as the closeness of the data to the estimated model [6] (See Sect. B.5.2).

The specific test used by GLM is based on a χ^2 distribution that someone familiar with nested hypothesis testing might expect (see Sect. B.5 for an explanation). Conveniently, this test is provided directly by the `anova` method

that generates a table in the conventional form of an analysis of variance (or analysis of deviance, here) table.⁶

```
> anova(SHR1.glm, test = "Chisq")
```

```
Analysis of Deviance Table
```

```
Model: binomial, link: probit
```

```
Response: cbind(NumYes, NumNo)
```

```
Terms added sequentially (first to last)
```

	Df	Deviance	Resid. Df	Resid. Dev	Pr(>Chi)
NULL			5	185.1	
log(Quanta)	1	182	4	2.8	<2e-16 ***

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The summary method provides a more detailed set of results:

```
> summary(SHR1.glm)
```

```
Call:
```

```
glm(formula = cbind(NumYes, NumNo) ~ log(Quanta),
     family = binomial(probit), data = SHR1)
```

```
Deviance Residuals:
```

1	2	3	4	5	6
-0.464	0.605	-0.145	-0.694	1.279	0.250

```
Coefficients:
```

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-13.017	1.618	-8.04	8.8e-16 ***
log(Quanta)	2.671	0.331	8.07	6.8e-16 ***

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)
```

```
Null deviance: 185.053 on 5 degrees of freedom
Residual deviance: 2.781 on 4 degrees of freedom
AIC: 17.36
```

⁶ The `anova` method might be better named something like `model.compare` but, alas, `anova` it is.

Number of Fisher Scoring iterations: 6

and includes the deviance residuals as well as additional statistics for the estimated coefficients, β_0 and β_1 described above.

The coefficients for the intercept and slope can be extracted as a vector from the model object with the `coef` method. Confidence intervals for the fitted coefficients are obtained with the `confint` method.

```
> coef(SHR1.glm)

(Intercept) log(Quanta)
      -13.02      2.67

> confint(SHR1.glm)

                2.5 % 97.5 %
(Intercept) -16.55  -10.1
log(Quanta)   2.08   3.4
```

The estimated responses for the data are obtained with the `fitted` method taking the model object as an argument, but typically one wants to generate enough values to produce a smooth curve for a plot. These can be obtained with the `predict` method. To use this, we supply the model object and a data frame with the same column names used in performing the fit. Additionally, we must specify that the predicted values are on the scale of the response instead of on that of the linear predictor, which is the default. At no extra cost, we obtain standard errors for the fit by setting the argument `se.fit = TRUE`.

We illustrate this using graphic functions from the base **graphics** package [146]. Note how with the `plot` function from base graphics, we create the graphic with successive function calls that add the embellishments.

```
> plot(PerCent/100 ~ Quanta, SHR1,
+      xlab = "Quanta/Flash", ylab = "Proportion \"seen\"",
+      main = "Obs: SH, Run: 1", log = "x",
+      xlim = c(20, 440)) # set up plot
> xseq <- seq(20, 450, len = 100)
> SHR1.pred <- predict(SHR1.glm, newdata =
+   data.frame(Quanta = xseq), type = "response",
+   se.fit = TRUE) # obtain predicted values
> polygon(c(xseq, rev(xseq)), c(SHR1.pred$fit + SHR1.pred$se.fit,
+   rev(SHR1.pred$fit - SHR1.pred$se.fit)),
+   border = "white", col = "grey") # plot SE envelope
> lines(xseq, SHR1.pred$fit, lwd = 2) # add fitted curve
> points(PerCent/100 ~ Quanta, SHR1, cex = 1.5, pch = 21,
+   bg = "white") #add points
```

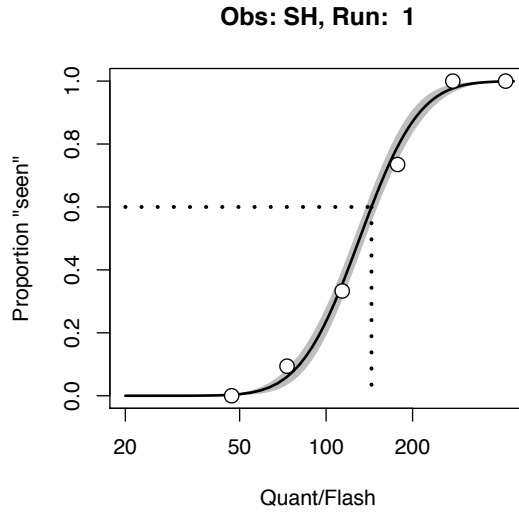


Fig. 1.3 Psychometric function fit by probit analysis to frequency of seeing data (solid curve) with standard error limits (grey envelope) for the first run of observer SH. The dotted lines indicate a threshold criterion

Notice that we did not have to worry about the fact that the covariate was transformed logarithmically in obtaining the predicted values displayed in Fig. 1.3 as the solid curve. The dotted lines obtained with the `segments` function are described below.

1.3.2 Estimating thresholds and *jnd*'s

Next we reparameterize the Gaussian function fit to the data in terms of the parameters $Q_{.5}$ and σ , that we began with.

$$Q_{.5} = -\frac{\beta_0}{\beta_1} \quad (1.3)$$

$$\sigma = \frac{1}{\beta_1}. \quad (1.4)$$

We extract the model coefficients from the `glm` object with the `coef` method and obtain the inverse Gaussian from the normal quantile function, `qnorm`.

Hecht et al. reported observer's performance level not as the $\log(\text{Quanta})$ value at which the observer would say Yes half of the time, $Q_{.5}$, but at the

$\log(\text{Quanta})$ value at which the observer would be expected to say Yes on 60% of the trials. We can use the Gaussian quantile function `qnorm` to compute this value for this observer:

```
> (thresh <- exp(qnorm(p = 0.6,
+                 mean = -coef(SHR1.glm)[1]/coef(SHR1.glm)[2],
+                 sd = 1/coef(SHR1.glm)[2])))
[1] 144
```

We use the `coef` method that returns the fitted parameters from the model object as a vector. To specify the threshold on the scale of quanta, the estimated value must be exponentiated to invert the logarithmic transformation of the covariate in the model formula. The threshold value is indicated in Fig. 1.3 by the dotted lines and was added to the figure with the `segments` function.

```
> segments(c(20, thresh), c(0.6, 0.6), c(thresh, thresh),
+         c(0.6, 0), lty = 3, lwd = 3)
```

If we need to perform this calculation frequently, it would be practical to make it into a function

```
> thresh.est <- function(p, obj) {
+   cc <- coef(obj)
+   m <- -cc[1]/cc[2]
+   std <- 1/cc[2]
+   qnorm(p, m, std)
+ }
```

where `p` is the probability and `obj` is the glm object⁷. An R function returns the value of its last executed line.

Then, we simply execute:

```
> thresh.est(0.6, SHR1.glm)
[1] 4.97
```

to get the estimate in $\log(\text{Quanta})$ or

```
> exp(thresh.est(0.6, SHR1.glm))
[1] 144
```

Although not of interest for these data, this function can be used as is to calculate a `jnd`, which we will define here as the intensity difference⁸ between 50% and 75% proportion seen. We use a vector for the argument `p` and apply the function `diff` to the result.

⁷ See the function `dose.p` in the package **MASS** for a more general version [178].

⁸ Actually, here it corresponds to the log of the ratio because of the log transform of **Quanta**


```
> diff(thresh.est(c(0.5, 0.75), SHR1.glm))
[1] 0.253
```

1.3.3 Modeling all of the data

The HSP data set contains the results for five experiments, representing three observers, two of whom were tested twice. Using a model formula, it is easy to model all of the data at once. For the sake of demonstration, we will consider the five experiments as independent replications with a single source of variance. To facilitate this point of view, we define a new factor with five levels from the interaction of `Obs` and `Run`:

```
> HSP$id <- with(HSP, interaction(Obs, Run, drop = TRUE))
```

This code adds a new column to the data frame in memory that codes each combination of observer and run as a separate level of a categorical or factor variable. The argument `drop = TRUE` avoids the creation of a level for the missing run of observer MHP.

We will compare two models. The simplest model assumes that a single psychometric function fits all five runs, i.e., a single intercept and slope describe all five runs. The call is identical to the one that we used earlier except that we specify the data frame `HSP` instead.

```
> HSP0.glm <- glm(cbind(NumYes, NumNo) ~ log(Quanta),
+               binomial(probit), HSP)
```

As an alternative, we consider a model in which each run has a separate intercept and slope. The `update` function simplifies fitting a new model. It takes a model object and a modification of its specification as arguments and returns the new fit.

```
> HSP1.glm <- update(HSP0.glm, . ~ id/log(Quanta) - 1)
```

The `.'` stands for what was in the model previously. The `/'` indicates that a separate model should be considered for each level of the factor `id`. This is really a short hand for the equivalent model: `id + id:log(Quanta)`, where the `:'` indicates an interaction between the two terms that it separates. Since there will be one intercept for each curve, we remove the overall intercept with the term `'-1'`.

We compare the models by calling `anova` with more than one argument.

```
> anova(HSP0.glm, HSP1.glm, test = "Chisq")
```

Analysis of Deviance Table

Model 1: cbind(NumYes, NumNo) ~ log(Quanta)

```

Model 2: cbind(NumYes, NumNo) ~ id + id:log(Quanta) - 1
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1      28      81.3
2      20      16.2  8     65.1  4.5e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The analysis of deviance table rejects the hypothesis that a single psychometric function fits the data in favor of the alternative.

The fitted curves under the two models are compared in Fig. 1.4 using the `ggplot2` package [190] which is based on *The Grammar of Graphics* [192]. It provides an alternate approach and look to visualizing data sets than `lattice`. Note that the factor levels by default are indicated on both vertical and horizontal margins. From this graph, we readily appreciate that a single psychometric function fits the data less well.

```

> nd <- data.frame(
+   Quanta = rep(seq(20, 500, len = 100), 5),
+   id = rep(levels(HSP$id), each = 100)
+ )
> levels(nd$id) <- levels(HSP$id)
> nd$pred0 <- predict(HSP0.glm, newdata = nd, type = "response")
> nd$pred1 <- predict(HSP1.glm, newdata = nd, type = "response")
> mm <- matrix(unlist(strsplit(as.character(nd$id),
+   "\\.")), ncol = 2, byrow = TRUE)
> nd$Obs <- factor(mm[, 1], levels = c("SH", "SS", "MHP"))
> nd$Run <- factor(mm[, 2])
> library(ggplot2)
> qplot(Quanta, Percent/100, data = HSP, facets = Run ~ Obs) +
+   geom_point(size = 4) +
+   geom_line(data = nd, aes(x = Quanta, y = pred1), size = 1) +
+   geom_line(data = nd, aes(x = Quanta, y = pred0), size = 1,
+     linetype = "dashed") +
+   scale_x_log10(limits = c(10, 500),
+     breaks = c(10, 20, 50, 100, 200),
+     labels = c(10, 20, 50, 100, 200))

```

1.3.4 Odds and ends

Before ending this chapter, we include a few additional household functions that are essential. First, the function `rm` will delete objects from the work space. To remove all visible objects in the workspace use:

```
> rm(list = ls())
```

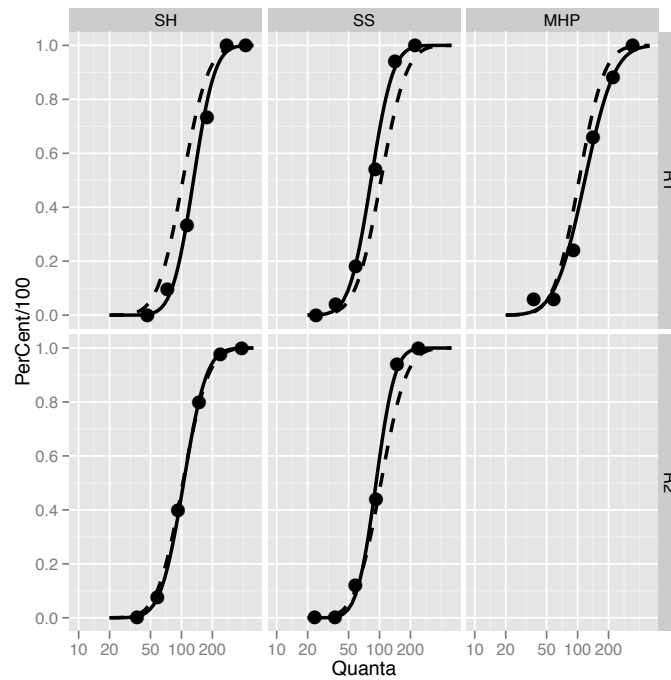


Fig. 1.4 Frequency of seeing data with a single psychometric function (dashed curves) and with individual curves per data set (solid curves), using the **ggplot2** package.

To end an R session, use the function `q9`. By default, this will open a dialog requesting whether you really want to quit or not and whether you want to save the variables from your work space. If these are saved, then the next time that you open R from that directory, they will be reloaded into memory.

In this chapter, we demonstrated an approach to modeling data in R through an extended example. Basic functions were introduced for examining objects and modeling data. In subsequent chapters, we will go deeper into the theory underlying several psychophysical paradigms. We will show that a common modeling framework links them and that R provides a remarkably rich, powerful and extensible environment for implementing these analyses.

⁹ In the GUI interface for Mac OS X, a platform dependency rears its ugly head. It is recommended to quit using the Quit button (in the upper right corner of the console window, resembling a switch for turning the room lights out) as otherwise, the history of past commands will not be saved.

1.4 Exercises

1.1. Explore the help facilities of R. At the command line, type `help(help)` and read over the documentation. Try the same with other help-related commands cited in the text, e.g., `help(apropos)`. Try the command `help.start()`

1.2. After having executed the code in this chapter, examine the search path. How has it changed? Why?

1.3. Check if the **MPDiR** package is in the search path. Examine its contents using the `pos` argument of `ls`.

1.4. We indicated that the operators “<-” and “=” are not identical. To appreciate the difference, consider the following exercise. Make sure that the **HSP** data frame is loaded in your work space and then assign a variable with the name `n` the value of 10, i.e., a 1 element integer vector. Now, examine the value of `n` after executing each of the following function calls.

```
> head(HSP, n = n)
> head(HSP, n = 15)
> head(HSP, n <- 15)
```

1.5. The function `splom` in the **lattice** package generates a matrix of scatterplots, similar to the pairs plot of the data frame in Sect 1.2.2. Plot the **HSP** data set with this function and compare it with Fig. 1.1 to appreciate the difference in presentation style.

1.6. Assign the results of `summary(SHR1.glm)` to a variable. What is its class? How many components does it have? How would you access the p -values of the fitted coefficients?

1.7. Refit the data of **SHR1** using the default logit link. Plot the predicted values on the same graph with the predicted values of the probit fit. How do they differ? How do the thresholds differ?

1.8. Examine the summary of `HSP1.glm`. What do the different terms represent in the model? Estimate the threshold at proportion seen = 0.6 for each of the 5 experimental runs.

1.9. The data set **Vernier** in the package **MPDiR** contains results from an experiment on the detection of misalignment (phase difference) between two adjacent, horizontal gratings that were drifting either upward or downward [167]. Extract the subset of the data for which the waveform is “Sine” and the temporal frequency is 2 cycles/degree and the direction is upward. Fit a psychometric function to this subset of the data using `glm`. Then, plot the data points and the fitted curve on the same graph. Using the fitted object, calculate the discrimination threshold as the phase shift difference between 0.5 and 0.75 detection upward.