Chapter 7
Data Analysis

This chapter outlines the main statistics used by psychologists. Some of the functions described here are included in the Statistics toolbox, a toolbox specifically devoted to statistical analysis. Moreover, the chapter presents how to get some of the signal detection theory indexes.

The purpose of this chapter is to help the reader to build upon existing statistical knowledge in order to support its application in MATLAB. Most of the functions detailed here are included in the Statistics toolbox, a devoted MATLAB toolbox for statistics. In addition, a number of files and functions running statistical analysis can be found in file exchange section of the Mathworks website at the following link (see the brick for an example): http://www.mathworks.com/matlabcentral/fileexchange/

The first part of the chapter outlines statistics, first descriptive statistics and then the inferential statistics. The last part outlines the signal detection theory indexes.

Descriptive Statistics

Measures of Central Tendency

The most-used measures of central tendency, such as the mean, the mode, and the median, are provided by built-in MATLAB functions. Specifically, `mean()`, `mode()`, and `median()` return the measures of the central tendency of a data vector. When the argument is a matrix, the functions return the measure of central tendency for each *column*. Table 7.1 shows these functions.

Additional descriptive measures can be acquired through the Statistics toolbox. For example, the toolbox includes functions for calculating the geometric mean, the harmonic mean, and the trimmed mean. Table 7.2 lists these functions. Note that with these functions, too, when the argument is a matrix, the function outputs the measure of central tendency for each *column.*

M. Borgo et al., *MATLAB for Psychologists*,
DOI 10.1007/978-1-4614-2197-9_7, © Springer Science+Business Media, LLC 2012

Table 7.1 Principal measures of central tendency

MATLAB function	Description
`mean()`	For vectors, `mean(v)` returns the arithmetic mean of the elements in `v`. For matrices, `mean(X)` returns the arithmetic mean of each *column*
`mode()`	For vectors, `mode(v)` returns the most frequent of the elements in `v`. For matrices, `mode(X)` returns the most repeated of the elements in each *column*
`median()`	For vectors, `median(v)` returns the median value of the elements in `v`. For matrices, `median(X)` returns a the median of the elements in each column

Table 7.2 Additional measures of central tendency provided by the Statistics toolbox

Statistical toolbox function	Description
`Goemean()`	Geometric mean of the input variable
`harmmean()`	Harmonic mean of the input variable
`trimmean(v,percent)`	`m=trimmean(v,percent)` calculates the mean of the variable v excluding the highest and lowest (percent/2%) of the observations

Table 7.3 Principal measures of dispersion

MATLAB function	Description	Output considering the vector `v=1:10;`
`max(v)-min(v)`	Range	`>> max(v)-min(v)` `ans = 9`
`std(v)`	Standard deviation	`>> std(v)` `ans = 3.0600`
`var(v)`	Variance	`>> var(v)` `ans = 9.3636`
`std(v)/sqrt(length(v))`	Standard error	`>> std(v)/sqrt(length(v))` `ans = 0.9226`
`median(v(find(v>median(v))))-median(v(find(v<median(v))))`	Inter-quartile range	`>> median(v(find(v>median(v))))-median(v(find(v<median(v))))` `ans = 6`

Measures of Dispersion

Range, standard deviation, variance, and standard error are the most-used measures of dispersion. Range, standard error, and interquartile difference, are built-in MATLAB functions. Additional dispersion measures are included in the Statistics toolbox [for example, `range()` and `iqr()`]. Table 7.3 shows the main dispersion measures assuming that the vector `v=1:10` is the input argument.

By default, these functions adopt the unbiased estimator of dispersion (i.e., N–1). If you need the second moment of the function, pass the optional argument 1. For example, `std(v,1)` returns the second moment of the standard deviation (i.e., uses N instead of N–1).

Bivariate and Multivariate Descriptive Statistics

Correlation and covariance are the most used-statistics to measure the strength of the relationships among variables. `corrcoef()` returns the correlation between two input vectors. If a matrix is passed instead, then `corrcoef()` calculates the correlation among all the columns of the matrix. By default, `corrcoef()` returns only a matrix of correlation coefficients. However, additional outputs can be requested such as:

1. A matrix of *p*-values indicating the correlation's significance;
2. Two matrices of the lower and the upper bounds of the 95% confidence interval for each regression coefficient.

The following MATLAB code returns the correlation matrix `R`, the correlation *p*-values `P`, matrices `RLO` and `RUP` for the lower and upper bound confidence intervals for each coefficient in `R`:

```
>>[R,P,RLO,RUP] = corrcoef(X);
```

Let us suppose that a psychologist wants to find out whether there is a correlation between listening span (e.g., the number of remembered sentences of a story) and the number of errors in an arithmetic test in 7-year-old children. Twenty children participate in the study and the data are as presented in Table 7.4.

Listing 7.1 runs the correlation analysis on the data.

Listing 7.1

```
% Runs the correlation analysis on hypothetical data
% AUTHOR:   Borgo-Soranzo-Grassi
span = [2 4 4 4 5 5 3 3 2 1 2 6 6 6 5 4 4 4 3 3];
errors = [4 2 2 4 3 4 3 2 2 6 5 1 2 1 1 2 2 1 2 3];
[R,P,RLO,RUP] = corrcoef(span, errors);
```

Listing 7.1 outputs the following arguments:

1. The correlation matrix `R`, showing that there is a negative correlation between the two variables: the higher the listening span, the lower the number of errors in the arithmetic test.

```
R =
   1.0000  -0.6214
   -0.6214  1.0000
```

Table 7.4 Hypothetical data of a correlation study. The table reports the listening span (measured as the number of remembered sentences of a story) and number of errors in an arithmetic test of 20 seven years old children

Child id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Span	2	4	4	4	5	5	3	3	2	1	2	6	6	6	5	4	4	4	3	3
Errors	4	2	2	4	3	4	3	2	2	6	5	1	2	1	1	2	2	1	2	3

2. The probability matrix `P`, showing that the probability is only 0.0035 of getting a correlation of −0.6214 by random chance when the true correlation is zero.

```
P =
   0.0035
```

3. The interval bounds matrices `RLO` and `RUP` showing that the lower and the upper bounds for a 95% confidence interval of the correlation coefficient are −0.8344 and −0.2467, respectively.

```
RLO =
   1.0000   -0.8344
   -0.8344   1.0000
RUP =
   1.0000   -0.2467
   -0.2467   1.0000
```

Covariance

The covariance matrix of a set of data `X` can be obtained by typing `cov(X)` at the MATLAB prompt. The diagonal of the output matrix contains the variance of each variable. The variances of a number of variables can be obtained using the `diag()` function in combination with the `cov()` function, in the following way: `diag(cov(X))`. With this function, too, we can pass the optional argument 1 to get the second moment of the function.

Simple and Multiple Linear Regression

There are different ways to run a linear regression analysis in MATLAB. Perhaps the easiest one is to use `regstats()`, which is included in the Statistics toolbox. The function takes the dependent variable as first argument and a matrix of predictors as second argument. It is also possible to pass an optional third argument for nonlinear regressions. If you type

```
regstats(y,X);
```

MATLAB displays a GUI listing the diagnostic statistics that can be saved into the workspace (Fig. 7.1).

The diagnostic statistics can be saved in a structure in this way:

```
s=regstats(y,X); saves the diagnostic statistics in structure s.
```

Imagine that a psychologist wants to discover a regression model for a class of 13 students by knowing their IQs and the number of hours they study per week. Table 7.5 lists the data:

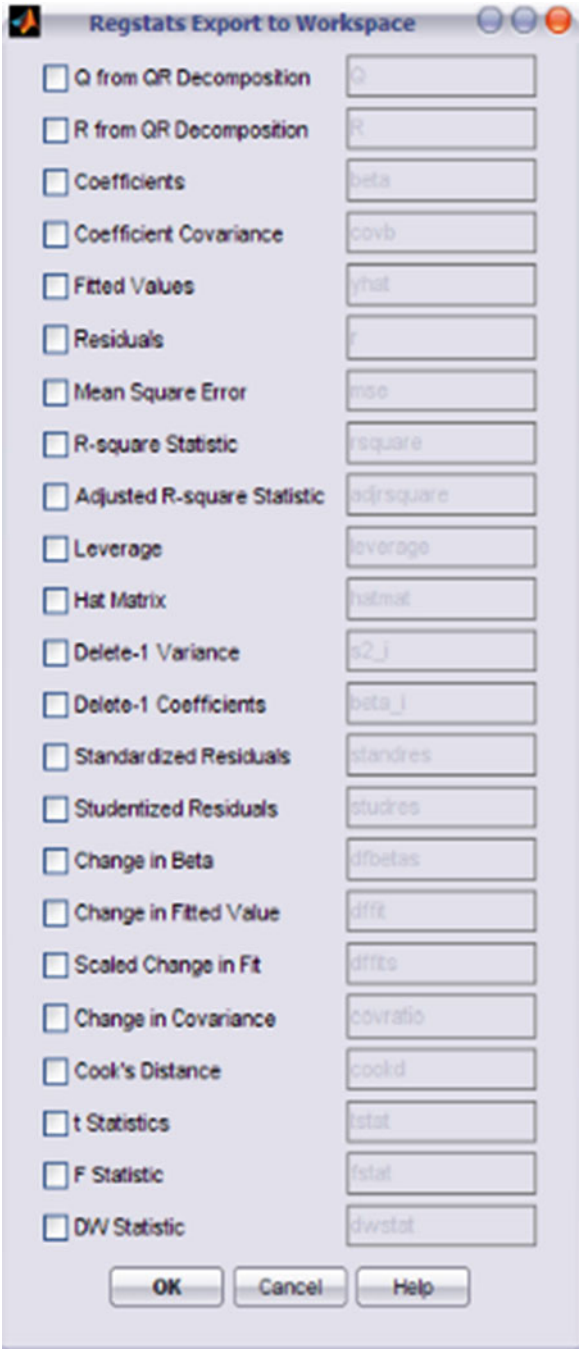


Fig. 7.1 GUI output by the `regstats()` function

Table 7.5 Hypothetical data for a regression analysis. The table lists the grade, the IQ, and the number of hours studied per week by 13 students

Student id	1	2	3	4	5	6	7	8	9	10	11	12	13
Grade	1	1.6	1.2	2.1	2.6	1.8	2.6	2	3.2	2.6	3	3.6	1.9
IQ	110	112	118	119	122	125	127	130	132	134	136	138	125
StudyTime	8	10	6	13	14	6	13	12	13	11	12	18	7

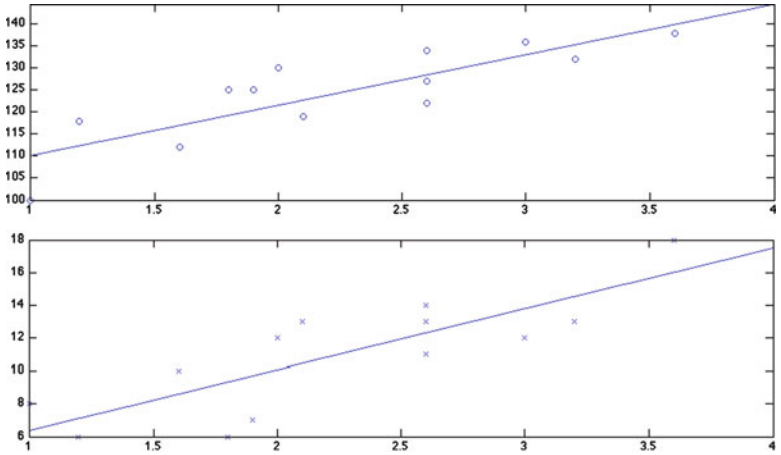


Fig. 7.2 Linear regression plot resulting from code listing 7.2

Listing 7.2 runs the regression analysis on these data.

Listing 7.2

```
grade = [1 1.6 1.2 2.1 2.6 1.8 2.6 2 3.2 2.6 3 3.6 1.9];
iq = [100 112 118 119 122 125 127 130 132 134 136 138 125];
studytime=[8 10 6 13 14 6 13 12 13 11 12 18 7];
s=regstats(grade', [iq' studytime']);
subplot(2,1,2)
plot(grade, studytime,'x')
lsline
subplot(2,1,1)
plot(grade, iq,'o')
lsline
```

By inspecting the structure `s`, we can see that the regression coefficients are `−5.3178` and `0.0505 0.1125` (`s.beta`), that they are all significant at an alpha level of `0.05` (`s.tstat`) and that the amount of variance explained by the predictors is `90.8%` (`s.rsquare`). Figure 7.2 shows the scatterplots for both the predictors with the regression lines.

The regression line can also be obtained through the built-in fitting feature: after plotting the data, select Tools > Basic Fitting from the Figure menu bar. This will show the GUI presented in Fig. 7.3.

As can be seen from Fig. 7.3, different polynomials can be fitted.

The next table shows the main functions fitting different polynomials.

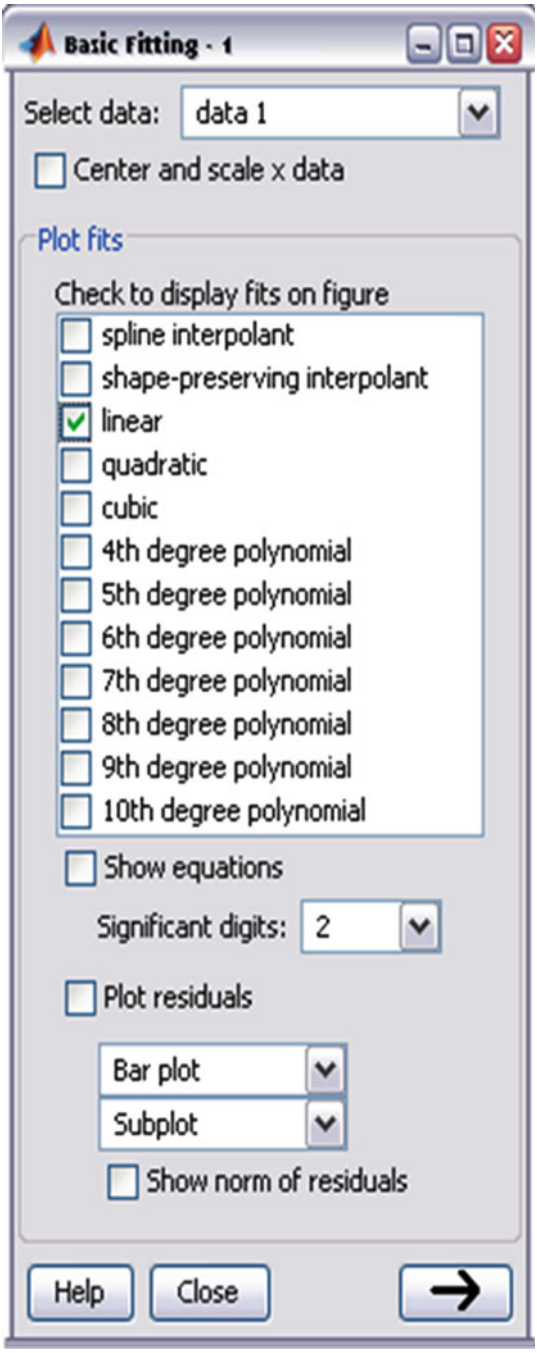


Fig. 7.3 The built-in MATLAB feature for Basic fitting. Plot the data and select Tools > Basic Fitting from the menu bar to fit different polynomials to the data

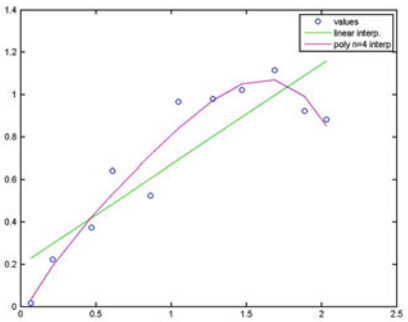
MATLAB function	Description
`p, S]=polyfit(X,y,n)`	Finds the coefficients of a polynomial *p(x)* of degree *n* that fits the data in a least squares sense. `p` is a vector of length `n`; each value is the polynomial coefficient. `S` is for use with `polyconf` to obtain error estimates or predictions
`y=polyval(p,x)`	Returns the value of a polynomial of degree `n` (having coefficient `p`) evaluated at `x`
`y,delta]=polyconf(p,X,S)`	Returns the value of a polynomial `p` evaluated at `x`. Use the optional output `S` created by `polyfit` to generate 95% prediction intervals. If the coefficients in P are least squares estimates computed by `polyfit` and the errors in the data input to `polyfit` were independent, normal, with constant variance, then there is a 95% probability that `y±delta` will contain a future observation at `x`

The next example (Listing 7.3) shows how to use each of these functions.

Listing 7.3 - M script

```
% Create the predictor variables
x1=[0:0.2:2];
x=x1+rand(size(x1))*0.1;

% create the dependent variable
y=sin(x);
y=y+randn(size(y))*0.1;

% find coeff. Of first grade
polynom.
p1=polyfit(x,y,1);

% find coeff. Of fourth grade
polynom.
p4=polyfit(x,y,4);

% evaluate the dependent variable
y1=polyval(p1,x);
y4=polyval(p4,x);

% Plot results
figure;
plot(x,y,'o');
hold on;
plot(x,y1,'g',x,y4,'m');
legend('values','linear interp.',...
'poly n=4 interp')
```

Graphical result



Generalized Linear Model

The Statistics toolbox includes the `glmfit()` function, which computes the generalized linear model regression of different distributions; that is, in addition to the default normal distribution, it is possible to use the binomial, the gamma, the inverse Gaussian, and the Poisson distributions.

Besides the regression coefficients, `glmfit()` returns the deviance of the fit at the solution vector and a structure with several pieces of information about the analysis, such as a *t*-test for each coefficient (t), the significance of each coefficient (*p*), and different types of residuals.

Table 7.6 Hypothetical data for a logistic regression analysis on a change blindness experiment. Detection is the binomial Dependent Variable (DV) indicating whether participants have detected the change within 5 s or not. Contrast is the Independent Variable (IV) of the scene contrast with three levels: low, medium, and high

Participant id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Detection	0	1	1	1	0	1	1	1	0	0	0	0	1	0	0	0	1	1	1	0
Contrast	1	2	2	3	3	1	3	2	2	1	1	1	2	1	1	2	2	1	2	3

Let's see an example of the use of `glmfit()`. In a hypothetical change blindness experiment,[1] a psychologist wants to arrive at the regression function for predicting the probability of detecting a change within 5 s of the stimulus presentation (detected = 1; not-detected = 0) as a function of the contrast of the whole scene (low-contrast = 1; medium-contrast = 2; high-contrast = 3). Twenty participants took part in the study, and the data presented in Table 7.6 have been collected.

The following code runs the logistic regression analysis on the data of Table 7.6.

```
>> contr = [1 2 2 3 3 1 3 2 2 1 1 1 2 1 1 2 2 1 2 3];%levels
of the contrast IV
>> detection = [0 1 1 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 0];%DV
>> [b dev stats]=glmfit(contr', detection', 'binomial')
b =
    -1.3421
     0.7483
dev =
    26.2642
stats =
         beta: [2x1 double]
          dfe: 18
         sfit: 1.0581
            s: 1
      estdisp: 0
         covb: [2x2 double]
           se: [2x1 double]
    coeffcorr: [2x2 double]
            t: [2x1 double]
            p: [2x1 double]
        resid: [20x1 double]
       residp: [20x1 double]
       residd: [20x1 double]
       resida: [20x1 double]
>> stats.p
ans =
    0.2763
    0.2431
```

[1] See Simons and Chabris (1999) for a funny example of change blindness.

This code outputs:

- Vector `b=[-1.3421; 0.7483]` indicates that the regression coefficient is 0.7483 and the value of a regression constant is −1.3421;
- Scalar `dev=[26.2642]` indicates the deviance of the fit;
- Structure `stat;` shows, among others things, that our regression coefficients are not statistically significant (`stats.p=[0.2763; 0.2431]`).

Inferential Statistics

Once we have described the data, we are interested in making inferences on these data, that is, to test our hypothesis. The Statistics toolbox includes several statistical tests for this purpose. The main parametric and nonparametric tests are outlined in the following sections.

Parametric Statistics

This section outlines the main MATLAB functions for parametric statistical analysis. However, in order to run a parametric analysis, parametric assumptions (e.g., a normality test on the data) have to be ascertained.

Assessing the Parametric Assumptions

To check for normality, we can look at the histograms and we can inspect their shape using the `hist()` function. The `kurtosis()` function returns the kurtosis of the distribution of the input vector. The function takes the optional argument zero (0), whether you want to correct for bias; by default there is no correction. The `skewness()` function returns a measure of the asymmetry. To test whether the data are normally distributed, the `kstest()` function performs a Kolmogorov–Smirnov test that compares the data to a standard normal distribution. It outputs 1 when the null hypothesis can be rejected. The probability of the test can be obtained as an additional output. For small sample sizes it might be preferable to use the `lillietest()` function, which performs a *Lillilifors* test of the default null hypothesis that the sample in the vector argument comes from a normal distribution, against the alternative hypothesis that it does not. Alternatively, the `jbtest()` function performs a Jarque–Bera test. The null hypothesis is that the sample comes from a normal distribution with unknown mean and variance.

z-Test

The Statistics toolbox includes the `ztest(v,m,sigma)` function for performing the *z*-test. The function takes as first argument a data vector v whose mean is equal to m (the second argument), and whose standard deviation is equal to sigma (the third

argument). In practice, this function is rarely used in psychology, since we normally do not know in advance the actual mean and standard deviation of the population. Much more common is, instead, the *t*-test.

t-Test

The Statistics toolbox includes two functions that perform the *t*-test: `ttest()` and `ttest2()`. The former is used to run both the one-sample *t*-test and the related *t*-test, while the latter is used to run a two-sample unrelated *t*-test. The significance level of the test and the direction of the hypotheses can be specified in both functions as optional parameters. The next section shows how to perform a one-sample *t*-test by means of the `ttest()` function; the following section details how to perform a two-sample *t*-test either through `ttest()` or through `ttest2()` depending on whether the data are related or unrelated, respectively.

One-Sample *t*-Test

The `ttest()` function tests the hypothesis that a vector's mean is different from zero. If the experimental hypothesis is against a different value, an additional argument has to be passed to the function specifying the desired value. The hypothesis that the mean is different from zero is performed against the default significance level of 0.05. A different value can be passed as third argument. In addition, the "tail" parameter, specifying the direction of the test, can be passed when you have a directional hypothesis. Pass 'left' when you are expecting that the mean is less than 0 (or the desired value); pass 'right' otherwise. By default, this optional argument is set to 'both' indicating that the test is bidirectional.

The function returns the test result in binary form: 1 means that the null hypothesis *can be* rejected; 0 means that it *cannot* be rejected. Additional outputs can be requested: (1) the probability of the test; (2) the confidence interval; and (3) a structure whose fields are the t value, the degrees of freedom, and the sample standard deviation.

To show how to use the *t*-test, we test whether the mean of 20 randomly generated numbers (ranging from 0 to 1) is actually different from 0.

```
>> [H, p, CI, stats]=ttest(rand(20, 1)) ;
H =
    1
p =
    2.6460e-06
CI =
    0.3360
    0.6491
stats =
    tstat: 6.5860
       df: 19
       sd: 0.3345
```

Table 7.7 Hypothetical data of ten participants where RTs are measured with both a red and a yellow probe

Participant id	1	2	3	4	5	6	7	8	9	10
Yellow	300	287	301	400	211	399	412	312	390	412
Red	240	259	302	311	210	402	390	298	347	380

`H=1` indicates that the null hypothesis can be rejected; `p=2.6460e-06`: the probability of finding these results by random chance is extremely low; `CI=0.3360 0.6491`: the confidence intervals of the mean are within these values. Finally, the function returns a structure with the details of the analysis.

Two-Sample *t*-Test

The Statistics toolbox includes two functions performing the two-sample *t*-test: (1) `ttest()` for the paired, or repeated measure, *t*-test, and (2) `ttest2()` for the unrelated, or independent measure, *t*-test.

The first two arguments to be passed to both functions are data vectors. Each vector represents either a condition, in the `ttest()` case, or a group, in the `ttest2()` case. Hence, the same `ttest()` function performing the one-sample *t*-test also performs the paired *t*-test. When this function receives as argument one vector only, or one vector and one scalar, it performs the one-sample *t*-test. When instead two vectors are passed, it performs the paired *t*-test. In this latter case, the participants' scores have to be in the same position in the two vectors, and the vectors should be of the same length. This constraint does not apply to the `ttest2()` function, where the two vectors might have different lengths.

`ttest2()` has the same default arguments as those of `ttest()`. Hence, the significance level is set to 0.05; again, a different value can be passed as third argument. In addition, the "tail" parameter, specifying the direction of the test, can be set. Pass `'left'` when you are expecting that the mean difference between the first and second samples is less than 0; pass `'right'` otherwise (`'both'` is the default).

`ttest2()` shares the same outputs as `ttest()`. That is, both functions return the test result in binary form: 1 to signify that the null hypothesis *can be* rejected; 0 to signify that the null hypothesis *cannot be* rejected. Additional outputs can be requested: (1) the probability of the test; (2) the confidence interval; and (3) a structure whose fields are the t value, the degree of freedom, and the sample standard deviation.

Let us suppose that a psychologist wants to run an experiment to test the effects of a probe color on Reaction Times (RTs). Specifically, the psychologist is interested in finding out whether RTs *decrease* when the color of a probe is red instead of yellow at an alpha level of 0.01. Ten participants are tested, and RTs, in milliseconds, are collected as shown in Table 7.7.

Listing 7.4 runs the related *t*-test analysis on these data.

Listing 7.4

```
% M-script to verify how ttest function works
% Authors: M.Borgo - A.Soranzo - M. Grassi
yellow = [300 287 301 400 211 399 412 312 390 412];
red = [240 259 302 311 210 402 390 298 347 380];
[h p ci stats]=ttest(yellow,red, 0.01,'right')
```

Since the alpha level is 0.01, we pass this number as a third argument. In addition, there is an expected direction of the test: we are expecting that RTs in the "red" condition should be faster than in the "yellow" condition. If we pass the vector "yellow" as a first argument, then we are expecting that the difference between the means should be larger than 0. Hence, the right argument has to be passed to the function.

Let's inspect the code's output.

```
h =
   1
p =
   0.0067
ci =
   2.3241 Inf
stats =
   tstat: 3.0719
      df: 9
      sd: 29.3381
```

h = 1 indicates that the null hypothesis can be rejected at the alpha level of 0.01; indeed, the probability of finding these results by random chance is equal to 0.0067 (*p*). ci = 2.3241 – Inf is the confidence interval (since the hypothesis is directional, one bound is Infinite); finally, stats is the structure with the *t*-test results.

If there were two different groups of participants, that is, one group was tested with the yellow probe only and the other group with the red probe only, then we would need `ttest2()` as follows:

```
>> [h p ci stats]=ttest2(yellow,red, 0.01,'right')
h =
   0
p =
   0.1787
ci =
 -48.5131 Inf
stats =
   tstat: 0.9446
      df: 18
      sd: 67.4690
```

In contrast to the within-subjects case, the null hypothesis cannot be rejected at the alpha level of 0.01. By looking at the p value, we find out that the probability of obtaining these results by chance with independent samples is 17.87%.

ANOVA

The Statistics toolbox includes three different functions for runing Analysis of Variance (ANOVA) analysis: `anova()`; `anova1()`, and `anova2()`. `anova1()` is used when there is one independent variable in between-subjects experiments; `anova2()` performs a balanced two-ways ANOVA in between-subjects experiments; `anovan()` performs both balanced and unbalanced ANOVA in between-subjects experiments with two or more factors; in addition, it performs ANOVAs in within-subjects experiments. Hence, both `anova2()` and `anovan()` can be used to run balanced two-ways ANOVAs . However, since `anovan()` has a broader application, its use is more common than `anova2()`.

One-Way ANOVA

The Statistics toolbox includes the `anova1()` function to run a one-way, independent-samples ANOVA. The function can take as argument a matrix: the matrix's cells are scores, and the matrix's columns are the groups `[anova1(X)]`. Alternatively, you can pass two arguments: the dependent variable and the group `[anova1(dv, group)]`. The latter way is useful when groups do not have the same size.

When groups have the same size, the group vector can be implemented directly in the function call to increase readability in the following way:

```
anova1([group1' group2' group3' ...])
```

Unless otherwise specified, `anova1()` displays two figures: the ANOVA table and a box plot. In addition, it returns the following arguments: (1) the *p*-value; (2) a text version of the ANOVA table; (3) a structure with values that can be used for a multicomparison analysis (see below).

To show how the function works, consider the experiment on the effects of the probe color on RTs (see the independent *t*-test section). Imagine that in that experiment, a third group is tested in which participants' RTs are measured when presented with a black probe.

The experiment's results are listed in Table 7.8.

Table 7.8 Hypothetical data of three groups made by ten participants each. RTs are measured as a function of probe color (three levels: yellow, red, and black)

Yellow	300	287	301	400	211	399	412	312	390	412
Red	240	259	302	311	210	402	390	298	347	380
Black	210	230	213	210	220	208	290	300	201	201

The following vectors are therefore implemented:

```
>> yellow = [300 287 301 400 211 399 412 312 390 412];
>> red = [240 259 302 311 210 402 390 298 347 380];
>> black = [ 210 230 213 210 220 208 290 300 201 201];
```

Since the number of participants per group is the same, `anova1()` can be used in both its ways. Let's see both of them. The first is to pass as first argument a matrix whose columns are the results of the groups (as observed above, this matrix can be implemented directly in calling the function). We also pass to the function an optional argument 'names' to improve output readability.

```
>> names = [{'yellow'} ; {'red'}; {'black'}];
>> [p table stats]=anova1([yellow' red' black'], names);
```

Figures 7.4 and 7.5 show the ANOVA table and the box plot outputted by the above code.

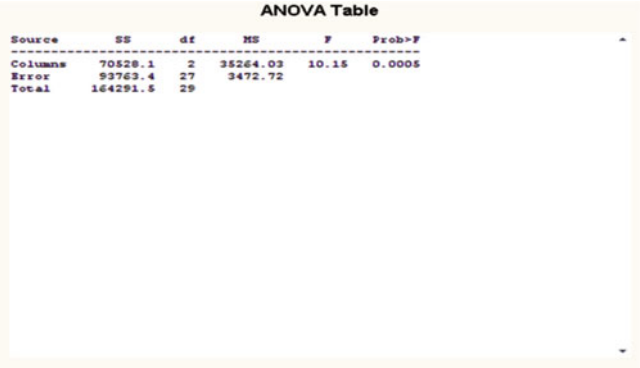


Fig. 7.4 ANOVA table resulting from the hypothetical probe color example

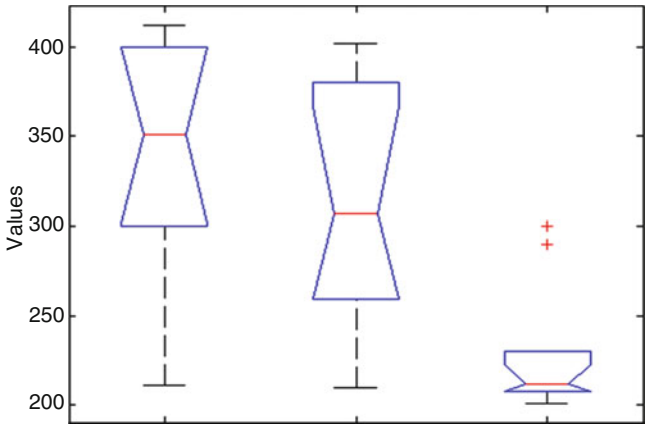


Fig. 7.5 Box plot resulting from the hypothetical probe color example

An alternative way of using `anova1()` is to pass as arguments a single vector with all RTs (i.e., of length 30) and a second vector having the same size as the first one, specifying the group type. Keep in mind that this is the only possible way of using the function when group sizes differ.

First, we merge the three groups' RTs into a single vector:

```
X=[yellow red black]';
```

Second, the vector with the groups' names has to be implemented in such a way that each group type is repeated for the number of participants belonging to the group (in this case the number of participants is the same for the three groups).

```
n_yellow=repmat({'yellow'},10,1);
n_red=repmat({'red'},10,1);
n_black=repmat({'black'},10,1);
group= [n_yellow' n_red' n_black']';
```

We added "n_" in front of the cell array's name to specify that these are names. Note the use of the apostrophe to transpose the vectors when the group vector is implemented.

Finally, we run the ANOVA test, whose results should be the same as in the previous case.

```
[p table stats]=anova1(X,group);
```

By looking at the function outputs, the null hypothesis can be rejected: there is a significant effect of the probe color on RTs.

The next step is to run a multicomparison analysis to find out the differences among groups. `multcompare()` is the function we want. It works together with the `anova1()` function and takes as argument the third argument returned by `anova1()`. The multicomparison test is returned by `multcompare(stats)` in the form of a five-column matrix. The first two columns indicate the group pair being compared, the third column is the estimated difference in means, and the last two columns are the interval for the difference.

In addition, the function returns an interactive figure. By clicking on the group symbol at the bottom, in part of the figure is displayed the group(s) from which the selected one statistically differs. In addition to the structure argument, some other arguments may be passed. The second one is the desired alpha level; the third is whether to display a plot, which can be set on or off. The fourth, which is more interesting, indicates the comparison type we need. There are several options: `'hsd'` for Tukey's honestly significant difference criterion (default); `'lsd'` for Tukey's least significant difference procedure; `'bonferroni'` for the Bonferroni adjustment to compensate for multiple comparisons; `'dunn-sidak'` for the Dunn and Sidák adjustment for multiple comparisons; and `sheffe()` for critical values from Scheffé's procedure.

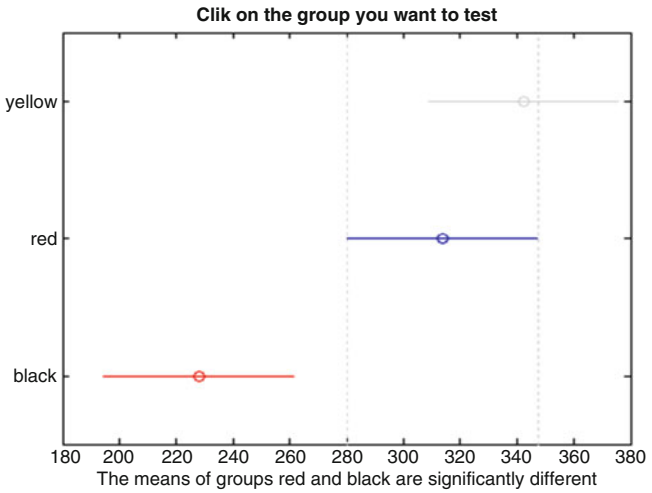


Fig. 7.6 Interactive plot resulting from multicomparison with Bonferroni correction

Let's suppose that we want to run a multicomparison test on the stats structure resulting from the previous ANOVA, using Bonferroni correction and that we are happy with the other defaults. The following code will do the job:

```
multcompare(stats,[],[],'bonferroni');
```

The interactive figure outputted by this code is shown in Fig. 7.6.

Two- and n-Way ANOVA

The `anova2()` function performs a balanced two-way ANOVA in a between-subjects experiment. It takes as first argument a matrix in which data of different columns represent changes in the first factor, while data in the rows represent changes in the second factor. If there is more than one observation for each combination of factors, an optional second argument can be passed to the function indicating the number of replicates in each position, which must be the same and a multiple of the number of rows.

Let's see how to use `anova2()` with an example. Imagine again the experiment on the effects of the probe color on RTs, but now we consider an additional factor that is probe size with two levels: small and large.

The data that have been collected for the six experimental groups are displayed in Table 7.9.

Table 7.9 Hypothetical data of six groups made by ten participants each. RTs are measured as a function of probe color (three levels: Yellow, Red, and Black); and as a function of probe size (two levels: Small and Big)

Small	Yellow	300	287	301	400	211	399	412	312	390	412
Small	Red	240	259	302	311	210	402	390	298	347	380
Small	Black	210	230	213	210	220	208	290	300	201	201
Big	Yellow	289	289	300	402	251	389	422	332	360	422
Big	Red	210	229	300	301	200	340	350	290	317	320
Big	Black	226	220	253	218	260	228	380	300	221	211

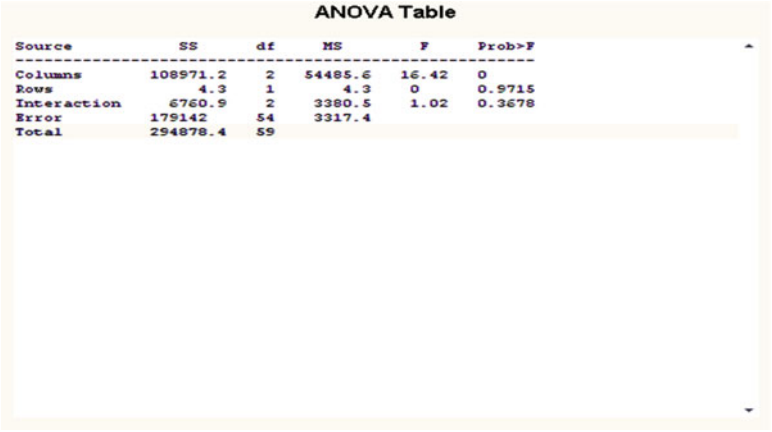


Fig. 7.7 Two-way ANOVA table output

Listing 7.5 performs a two-way ANOVA on the above data. Note that since we have ten participants per group, reps = 10. When implementing the matrix to be passed to `anova2()`, we need to be careful. Specifically, each column of the matrix has to be made in such a way that the corresponding rows indicate a score of the same level of the second variable, which is organized in rows. Since in this case the second variable has two levels, the first 10 scores belong to its first level, while the last 10 scores belong to the second one (Fig. 7.7).

Listing 7.5

```
% M-script to test the anova2 function
% Authors: M. Borgo - A. Soranzo - M. Grassi
small_yellow = [300 287 301 400 211 399 412 312 390 412];
small_red = [240 259 302 311 210 402 390 298 347 380];
small_black = [210 230 213 210 220 208 290 300 201 201];
big_yellow = [289 289 300 402 251 389 422 332 360 422];
big_red = [210 229 300 301 200 340 350 290 317 320];
big_black = [226 220 253 218 260 228 380 300 221 211];
yellow = [small_yellow big_yellow];
red = [small_red big_red];
black = [small_black big_black];
[p,table,stats]= anova2([yellow' red' black'],10);
multcompare (stats)
```

Table 7.10 Hypothetical data of one group of ten participants. In a repeated-measure design, RTs are measured as a function of probe color (three levels: Yellow, Red, and Black)

Participant id	1	2	3	4	5	6	7	8	9	10
Yellow	300	287	301	400	211	399	412	312	390	412
Red	240	259	302	311	210	402	390	298	347	380
Black	210	230	213	210	220	208	290	300	201	201

As can be seen from the table, the results show a significant effect of the Color variable (organized in columns) and no significant effect of the Size variable (organized in rows). The interaction between the two variables is not significant. However, the use of `anova2()` is uncommon; generally, `anovan()` is used when there is more than one Independent variable, because it can be used with both balanced and unbalanced data.

anovan()

`anovan()` performs both balanced and unbalanced multiple-way ANOVA for comparing the means of the observations in X (the first argument vector) with respect to different groups (the second argument). Hence, it works in a similar way to the `anova1()` function when group sizes are different. The first vector argument has to be carefully implemented. In detail, the position of the factors' levels has to correspond to the group name in the second vector argument. For example, if the first factor has three levels, the numbers in the vector should be organized in triplets. The second argument is a cell array containing the name of the conditions. Hence, with two factors, the code is as follows:

```
anovan(X,{IV1 IV2})
```

Note that the second argument is a cell array passed to the function by means of the curly braces.

`anovan()` receives many further optional arguments, including the 'alpha' level, the 'model' type (the interaction type is very often used in psychology because we are interested not only in the main effects of the factors but also in their interactions), and the 'names' cell array containing the names of the factors.

`anovan()` outputs the same arguments as `anova1()`, as well an additional vector with the main and interaction terms. The following example illustrates `anovan()` use.

Example

To show the use of `anovan()`, let's return to the experiment on the effects of the probe color on RTs that we have used to study `anova1()`. Listing 7.6 rearranges the data of Table 7.10 to be used with `anova1()`.

Listing 7.6

```
1  % M-script to test the anovan and multicompare functions
2  % Authors: M. Borgo - A. Soranzo - M. Grassi
3  small_yellow = [300 287 301 400 211 399 412 312 390 412];
4  small_red = [240 259 302 311 210 402 390 298 347 380];
5  small_black = [210 230 213 210 220 208 290 300 201 201];
6  big_yellow = [289 289 300 402 251 389 422 332 360 422];
7  big_red = [210 229 300 301 200 340 350 290 317 320];
8  big_black = [226 220 253 218 260 228 380 300 221 211];
9  nsubjsy=10;
10 nsubjsr=10;
11 nsubjsb=10;
12 nsubjby=10;
13 nsubjbr=10;
14 nsubjbb=10;
15 sizesmall=repmat({'small'},nsubjsy+nsubjsr+nsubjsb,1);
16 sizebig=repmat({'big'},nsubjsy+nsubjbr+nsubjbb,1);
17 Size=[sizesmall; sizebig];
18 yellowsmall=repmat({'yellow'},nsubjsy,1);
19 redsmall=repmat({'red'},nsubjsr,1);
20 blacksmall=repmat({'black'},nsubjsb,1);
21 yellowbig=repmat({'yellow'},nsubjby,1);
22 redbig=repmat({'red'},nsubjbr,1);
23 blackbig=repmat({'black'},nsubjbb,1);
24 Color=[yellowsmall; redsmall; blacksmall; yellowbig; redbig; blackbig];
25 X=[small_yellow small_red small_black big_yellow big_red big_black]';
26 [p table stats terms]=anovan(X,{Size Color}, 'model','interaction',
   'varnames',{'Size' 'Color'});
27 multcompare(stats,'dimension', [1 2])
```

Code listing 7.6 has been implemented for general purposes. Indeed, in this hypothetical experiment all groups have the same size, and there was no need to implement different variables for each group size (rows 9–14). This code could have been written in many different ways. For example, it is possible, and quicker, to assign a number to each group rather than a label. However, these changes would have reduced the output's legibility. Figure 7.8 shows the results of the ANOVA analysis.

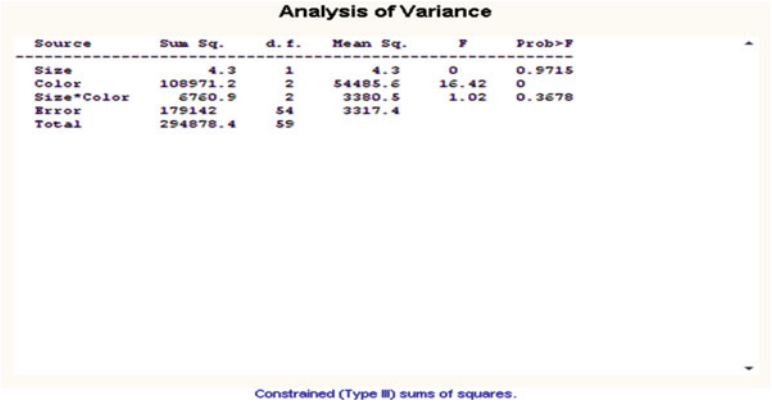


Fig. 7.8 ANOVA table output from listing 7.6

By looking at the ANOVA table we can conclude that there is a significant effect of the color variable, while there is no effect of the probe dimension on RTs (results are exactly the same as those obtained with `anova1()`).

One-Way Repeated-Measure ANOVA Analysis with `anova1()`

Let's reconsider the experiment on the effects of the probe color on RTs. However, imagine that the data come from a repeated-measure design in which the same participants have been tested with probes of different colors. Hence, the hypothetical data are the same as before, as displayed in Table 7.10.

To run a one-way repeated-measure ANOVA, we use the `anovan()` function by passing the variable Subject as a second factor having random effects, whose levels are the participants. To do this, we add the optional argument `'random'` specifying that the second variable has random effects. Since there are two variables only, it would be not necessary to specify that the variable Subject has random effects. However, for the sake of clarity, it is better to specify that these variables' effects are not fixed. In addition, we familiarize ourselves with the use of the optional argument 'random', whose use becomes necessary in ANOVAs designs with more than one factor. Hence, we have to implement two factors: the first, Color, with three levels (yellow, red, and black) and the second, Subject, with ten levels (the ten participants).

Listing 7.7 will do the job:

Listing 7.7

```
yellow = [300 287 301 400 211 399 412 312 390 412];
red = [240 259 302 311 210 402 390 298 347 380];
black = [ 210 230 213 210 220 208 290 300 201 201];
nsubj=10;
ncond=3;
s=repmat(1:nsubj,ncond,1);
col=repmat(1:ncond,nsubj,1);
X=[yellow; red; black];
X=reshape(X,nsubj*ncond,1);
Subj=reshape(s,nsubj*ncond,1);
Color=reshape(col',nsubj*ncond,1);
[p table stats]=anovan(X,{Color Subj},'random',2);
multcompare(stats)
```

Two-Way Repeated-Measure ANOVA

Listing 7.8 shows how to run a two-way repeated-measure ANOVA using `anovan()`. For this purpose, we will use again the experiment on the effects of probe size and color on RTs, but now we hypothesize that the experimental design was within subjects (i.e. that the same participants run all the conditions).

Listing 7.8

```
yellow = [300 287 301 400 211 399 412 312 390 412];
red = [240 259 302 311 210 402 390 298 347 380];
black = [ 210 230 213 210 220 208 290 300 201 201];
big_yellow = [289 289 300 402 251 389 422 332 360 422];
big_red = [210 229 300 301 200 340 350 290 317 320];
big_black = [226 220 253 218 260 228 380 300 221 211];
X=[small_yellow small_red small_black big_yellow big_red big_black]';
nsubj=10;
nlevelsSize=2;
nlevelColor=3;
Subj=repmat(1:nsubj,1,nlevelsSize*nlevelColor)';
sizesmall=repmat({'small'},1,nlevelColor*nsubj)';
sizebig=repmat({'big'}, 1,nlevelColor*nsubj)';
Size= [sizesmall' sizebig']';
Colyellow=repmat({'yellow'},nsubj,1);
Colred=repmat({'red'},nsubj,1);
Colblack=repmat({'black'},nsubj,1);
Color= [Colyellow' Colred' Colblack' Colyellow' Colred' Colblack']';
[p table stats terms]=anovan(X,{Size Color Subj}, 'random', 3, 'model', 2,
'varnames', {'Size' 'Color' 'Subj'});
multcompare(stats,'dimension', [1 2])
```

Note the use of the "'random', 3" argument in `anovan()` to specify that the third factor has random effects. Note also the use of the argument "`'dimension', [1 2]`" in `multcompare()` to specify that we want to run a multiple comparisons test between the first two variables.

Three-Way Mixed-Measures ANOVA

To conclude the `anovan()` overview, let's see how to run an ANOVA with mixed models, when some factors are within subjects, and others are between subjects. To do this, we have to pass to `anovan()` an additional optional argument, "nesting". Indeed, if one factor is between subjects in an otherwise repeated-measure design, then it can be said that the variable "subjects" is "nested" within the between-subjects variable. For example, let's consider again the experiment on the effects of color and size of a probe on RTs. The experimenter wants to test also whether there is a difference in RTs between males and females. In this case, we say that the variable 'Subjects' is "nested" within the sex variable. We have to pass to `anvoan()` an additional argument specifying which is the nested variable. To do this, we need to implement a square matrix whose dimensions are equal to the number of factors, and each row and column represents a factor in the same order in which they are passed to the function. So row 1 and column 1 represent factor 1; row 2 and column 2 represent factor 2; and so on. Each matrix cell represents the nesting relationship between each pair of variables. The cell value will be 0 when there is no nesting relationship and 1 otherwise. Specifically, it will be 1 when the factor represented by the row is nested within the IV represented by the column. If the third factor is nested within the fourth, then cell 3,4 will be 1. The implementation of this matrix may be slightly laborious; at the end of the chapter we will see a way to make it simpler.

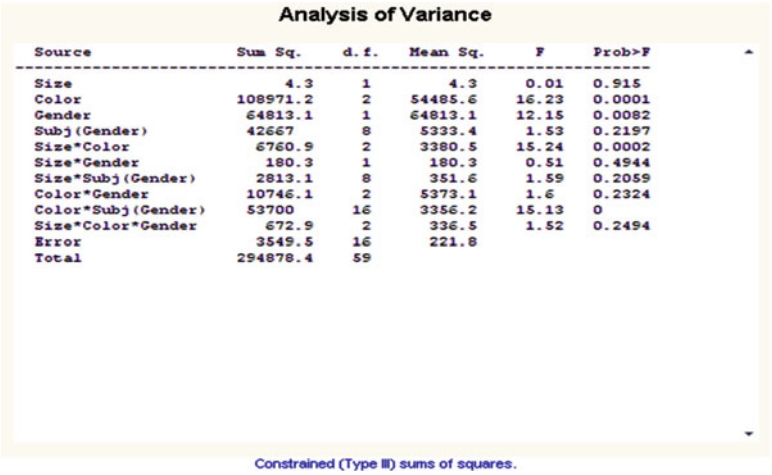
In addition, since a three-way mixed-measures design is quite a complex one, modeling the interactions might be laborious as well. We will see a shortcut for the implementation of this argument. But first, let's analyse code Listing 7.9 to see how to implement both the "nesting" and "modeling" variables and how to pass them to `anovan()`. Data are the same as in the two-way repeated-measures example above, but now we also take into account the Gender of the participants as an independent variable: the first five subjects were males, and the remaining five were females, and we are interested in testing whether there is any difference in the RTs between these two groups.

Listing 7.9

```
yellow = [300 287 301 400 211 399 412 312 390 412];
red = [240 259 302 311 210 402 390 298 347 380];
black = [ 210 230 213 210 220 208 290 300 201 201];
big_yellow = [289 289 300 402 251 389 422 332 360 422];
big_red = [210 229 300 301 200 340 350 290 317 320];
big_black = [226 220 253 218 260 228 380 300 221 211];
X=[small_yellow small_red small_black big_yellow big_red big_black]';
nsubj=10;
nlevelsSize=2;
nlevelColor=3;
nmales=5;
nfemales=5;
Subj=repmat(1:nsubj,1,nlevelsSize*nlevelColor)';
sizesmall=repmat({'small'},1,nlevelColor*nsubj)';
sizebig=repmat({'big'}, 1,nlevelColor*nsubj)';
Size= [sizesmall' sizebig']';
Colyellow=repmat({'yellow'},nsubj,1);
Colred=repmat({'red'},nsubj,1);
Colblack=repmat({'black'},nsubj,1);
Color= [Colyellow' Colred' Colblack' Colyellow' Colred' Colblack']';
male= repmat({'m'},1,nmales)';
female= repmat({'f'},1,nfemales)';
Gender=repmat([male; female]',1,nlevelsSize*nlevelsColor)';
nesting=[0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 1 0 ];
modeling=[1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1; ... %main effects
  1 1 0 0; 1 0 1 0; 1 0 0 1 ; ... %first IV interactions
  0 1 1 0; 0 1 0 1;...   %second IV interactions
  1 1 1 0];              % all interactions among IVs apart the nested one
[p table stats terms]=anovan(X,{Size Color Gender Subj }, 'random',4,
'model',modeling,'nested',nesting,'varnames',{'Size' 'Color' 'Gender'
'Subj'});
```

Figure 7.9 shows that there was a significant effect of Gender on RTs, while the interactions with both the Size and Color factors were not statistically significant.

Let's return to Listing 7.9 and see how we can implement the two matrices for modeling and nesting in a simpler way. Regarding the modeling matrix, we could have passed the "interaction" label to the "model" argument [i.e.: `anovan(.... 'model', 'interaction',... )`]. However, when used in this way, `anovan()` computes all the first-term interactions without computing the interactions among more than two variables. Hence, the remaining error is larger, since it is not explained by the interactions among more than two factors. Hence, unless you do not have good theoretical reasons to assume that some interactions have to be omitted, this is



Analysis of Variance

Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
Size	4.3	1	4.3	0.01	0.915
Color	108971.2	2	54485.6	16.23	0.0001
Gender	64813.1	1	64813.1	12.15	0.0082
Subj(Gender)	42667	8	5333.4	1.53	0.2197
Size*Color	6760.9	2	3380.5	15.24	0.0002
Size*Gender	180.3	1	180.3	0.51	0.4944
Size*Subj(Gender)	2813.1	8	351.6	1.59	0.2059
Color*Gender	10746.1	2	5373.1	1.6	0.2324
Color*Subj(Gender)	53700	16	3356.2	15.13	0
Size*Color*Gender	672.9	2	336.5	1.52	0.2494
Error	3549.5	16	221.8		
Total	294878.4	59			

Constrained (Type III) sums of squares.

Fig. 7.9 ANOVA table output from listing 7.9

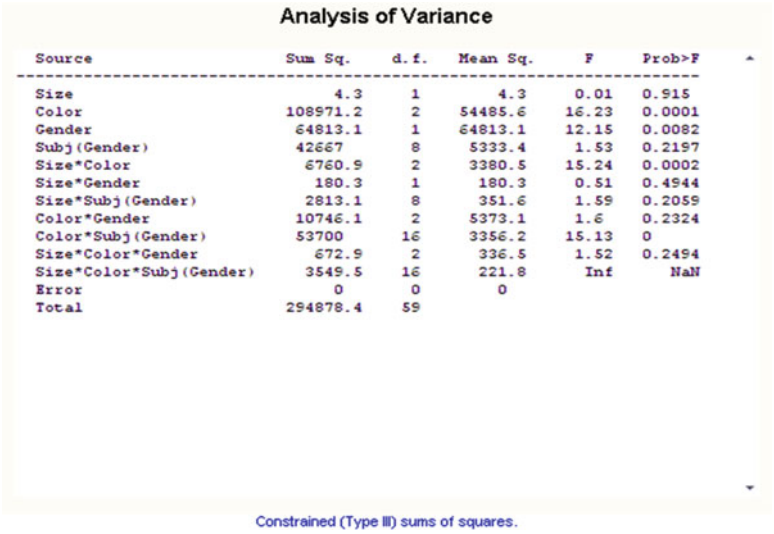
not always a good option. A handy shortcut to model the interactions, however, might be to pass to the "model" argument a scalar corresponding to the number of the not-nested factors [in this way: `anovan(....'model', 3)]`. The drawback of doing this is negligible: the error is wrongly attributed to the interaction among all the variables, and therefore the "real" error of the statistic will be set to 0. Figure 7.10 shows the output of the same listing as above, but the `anovan()` call is the following:

```
[p table stats terms]=anovan(X,{Size Color Gender Subj },'random',4,
 'model',3,'nested',nesting,'varnames',{'Size' 'Color' 'Gender'
'Subj'});
```

As can be seen from Fig. 7.10, the last interaction term is actually the error shown in Fig. 7.9.

The shortcut that we can suggest to implement the matrix specifying the nesting relationship among the variables is the following. Remember to pass to `anovan()` the variable "Subjects" and the nesting variable as the last one and the second from the last, respectively. Implement the "nesting" matrix with all zeros and then replace with 1 the cell whose coordinates are [number of variables, number of variables−1]. Hence, in Listing 7.9, line 24 could have been replaced by:

```
Ivnumber=4;
nesting=zeros(IVnumber,IVnumber);
nesting(IVnumber,IVnumber-1)=1;
```



Analysis of Variance

Source	Sum Sq.	d.f.	Mean Sq.	F	Prob>F
Size	4.3	1	4.3	0.01	0.915
Color	108971.2	2	54485.6	16.23	0.0001
Gender	64813.1	1	64813.1	12.15	0.0082
Subj(Gender)	42667	8	5333.4	1.53	0.2197
Size*Color	6760.9	2	3380.5	15.24	0.0002
Size*Gender	180.3	1	180.3	0.51	0.4944
Size*Subj(Gender)	2813.1	8	351.6	1.59	0.2059
Color*Gender	10746.1	2	5373.1	1.6	0.2324
Color*Subj(Gender)	53700	16	3356.2	15.13	0
Size*Color*Gender	672.9	2	336.5	1.52	0.2494
Size*Color*Subj(Gender)	3549.5	16	221.8	Inf	NaN
Error	0	0	0		
Total	294878.4	59			

Constrained (Type III) sums of squares.

Fig. 7.10 The last interaction term is actually the error term in Fig. 7.9 (see text for details)

Obviously, this shortcut can be applied only if you have only one between (nested) subjects variable in an otherwise repeated-measure design. If your design includes more than one nested Independent variable, a more general shortcut to the full implementation of the matrix might be the following:

```
nesting=zeros(factornumber,factornumber);
for i = 1: NestedNumber
   nesting(factornumber,factornumber-i)=1;
end
```

where factornumber and NestedNumber are the number of factors and the number of between-subjects factors, respectively. It has to be remembered that this solution can be applied only when the variable Subjects is passed to `anovan()` as the last argument and the nested variables are passed just before it.

Nonparametric Statistics

Categorical Data

Binomial Distribution

The first categorical statistic we see in this section is the binomial distribution, which is used when each independent trial results in one of two mutually exclusive outcomes (Bernoulli trial). The probability of getting x (usually called “success”)

out of n trials given the probability p of a success on any one trial is given by the number of combinations of n objects taken x at a time. We use the MATLAB built-in function `nchosek(n,x)`, which computes the combinations of n things taken x at a time. The following code returns the probability of x successes out of n trials:

```
nchoosek(n,x)*p^x*(1-p)^(n-x)
```

The same result can be obtained using the `binopdf(x,n,p)` function (which is included in the Statistics toolbox).

Chi2

The chi-square statistic is often used to understand whether the distribution of the results is consistent with a theoretical distribution. In this case, we use the `chi2gof()` function, which tests the goodness of fit between a theoretical distribution and empirical data. By default, `chi2gof()` returns `1` when the null hypothesis can be rejected and `0` otherwise. We can ask for further outputs such as the probability of the test and for a structure, whose fields are:

- `chi2stat`, is the value of the chi square statistic;
- `dof`, degrees of freedom;
- `edges`, vector of categories' edges that have been used to calculate the frequencies;
- `O`, observed frequency for each category;
- `E`, expected frequencies for each category according to our theoretical distribution.

(Expected values can be taken from any function. The default function used by `chi2gof()` is the normal distribution, but we can ask for a different function by passing it as optional argument).

Let's see an example of the `chi2gof()` function in action. A psychologist is interested in finding out whether there is any systematic preference for a specific display position to pick an object at will (for example, for an object that is at eye level). Each stimulus display consists of five identical objects placed in five different positions. Let us assume that both participants' choices and frequencies of position are uniformly distributed (i.e., we do not expect there is any specific preference for any of the positions of the display). Listing 7.10 runs the test.

Listing 7.10

```
x = round(rand(100,1)*4)+1;  % generate 100 random numbers
                             % within the 1-5 range
ctrs = [1, 2, 3, 4, 5];      % an array that keeps the coding
                             % of the values we expect
expectedCounts = [20, 20, 20, 20, 20]; % the expected freq. for each value
[h,p,st] = chi2gof(x,'ctrs',ctrs,'expected',expectedCounts)
```

This code returns the following variables:

```
h = 1
p = 0.0197
st =
   chi2stat: 11.7000
         df: 4
      edges: [0.5000 1.5000 2.5000 3.5000 4.5000 5.5000]
          O: [14 28 25 23 10]
          E: [20 20 20 20 20]
```

`h` informs that the null hypothesis can be rejected. p shows that the probability if finding these results by chance is equal to 0.0197. Then the structure `st` shows the chi2 value, the degrees of freedom, the vector of categories' edges that have been used to calculate the frequencies, and the observed and expected frequencies.

The `chi2gof()` function has several options; each one is written in string form (e.g., `"expected"` in the previous example) that is followed by the function's optional values (in the above example, an array with the expected frequencies).

Ordinal Data

Rank data are used with ordinal data or when the parametric assumptions to run parametric tests are not met.

Wilcoxon Signed Rank Test

The `signrank()` function performs the Wilcoxon signed rank test and tests the hypothesis that the median of the vector argument is significantly different from zero. Similarly to the `ttest()` function, it is possible to pass an additional second argument specifying a different median value of the experimental hypothesis. When the second optional argument is a vector, the function tests the hypothesis that the median for the two vectors is different. Use this option with paired samples. If your samples are independent, use `ranksum()` instead.

To show an example of the Wilcoxon signed rank test, let us consider a boundary extension (BE) experiment, where BE is the tendency to remember scenes as if they included information beyond the boundaries (Intraub and Richardson 1989). A psychologist wants to test the hypothesis that alcohol consumption favors this phenomenon. In a repeated-measures design, participants were presented with ten pictures on two different days (with and without alcohol consumption). In a recognition task, participants were presented with ten pictures, and after 1 min they were asked to select from two pictures which one had been presented before. One of these two pictures was the original one, while in the other the boundary was extended. The number of BE occurrences has been recorded in the two vectors shown in Table 7.11.

Table 7.11 Hypothetical data of ten participants, where BE occurrences (out of ten pictures) are measured after alcohol consumption or without alcohol consumption

Part id	1	2	3	4	5	6	7	8	9	10
Alcohol	6/10	4/10	5/10	6/10	3/10	3/10	6/10	7/10	8/10	2/10
No alcohol	1/10	3/10	3/10	6/10	3/10	2/10	5/10	6/10	6/10	3/10

Listing 7.11 runs the nonparametric test on the hypothetical data presented in Table 7.1.

Listing 7.11

```
1 NoAlcohol =[1/10 3/10  3/10  6/10  3/10  2/10  5/10  6/10  6/10  3/10];
2 Alcohol =   [6/10      4/10  5/10  6/10  3/10  3/10  6/10  7/10  8/10
      2/10];
3 [p,h,stats]= signrank(Alcohol , NoAlcohol);
```

Listing 7.11 outputs `h=1` and `p=0.0391`. Hence, the Wilcoxon signed rank test indicates that we can reject the null hypothesis at level 0.05 of significance: alcohol consumption affects BE occurrences.

Mann–Whitney *U* Test (or Wilcoxon Rank Sum Test)

The `ranksum()` function performs the Mann–Whitney *U* test. It is the equivalent of the `ttest2()` function for parametric data. It tests the hypothesis that the median of the two independent groups is equal. It takes the same optional arguments as `ttest2()`.

Again, we use the same data we used for the Wilcoxon signed rank test (see Listing 7.11), but now let us say that the data come from two independent groups. The following line of code runs the analysis:

```
>> [p,h,stats]= ranksum(Alcohol , NoAlcohol)
```

This code outputs `h=0` and `p=0.1864`. Hence, the Mann–Whitney U indicates that we cannot reject the null hypothesis. By looking at the p value, we can see that the probability of finding these results by chance with independent samples is 18.64%.

Kruskal–Wallis Test

`kruskalwallis()` is used when there are more than two *independent* groups. It is therefore similar to the `anova1()` function. Similarly to `anova1()`, it can be passed either a single matrix in which each column represents a variable, or two vectors, the data vector and the group vector. The latter method has to be used when the groups' sizes differ.

Table 7.12 Hypothetical data of three groups of ten participants each. BE occurrences are measured as a function of alcohol consumption (three levels: alcohol, no alcohol, and placebo)

Alcohol	6/10	4/10	5/10	6/10	3/10	3/10	6/10	7/10	8/10	2/10
No alcohol	1/10	3/10	3/10	6/10	3/10	2/10	5/10	6/10	6/10	3/10
Placebo	1/10	4/10	4/10	6/10	3/10	3/10	6/10	6/10	5/10	3/10

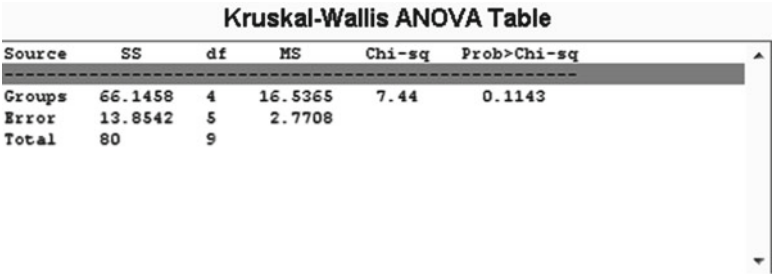


Fig. 7.11 Kruskal–Wallis ANOVA table

`kruskalwallis()` returns the same arguments as `anova1()`. Hence, we can run a multicomparison test through the `multcompare()` function in the same way we did for `anova1()`.

Let's consider again the BE experiment, but now there is a third group, which has been given a placebo instead of alcohol.

Table 7.12 shows the data for the three groups.

The following code tests the hypothesis that alcohol assumption affects BE when there are three groups of participants and data are not parametric.

```
>> Alcohol = [6/10 4/10 5/10 6/10 3/10 3/10 6/10 7/10 8/10 2/10];
>> NoAlcohol =[1/10 3/10 3/10 6/10 3/10 2/10 5/10 6/10 6/10 3/10];
>> Placebo=[1/10 4/10 4/10 6/10 3/10 3/10 6/10 6/10 5/10 3/10];
>> [p table stats]= kruskalwallis (Alcohol, NoAlcohol, Placebo)
```

Figure 7.11 shows the output of this code.

Friedman's Test

`friedman()` is used when there are two balanced independent variables. It takes as first argument a matrix in which data in different columns represent changes in the first factor, while data in the rows represent changes in the second factor. If there is more than one observation for each combination of factors, an optional second argument can be passed to the function indicating the number of replicates in each position, which must be the same and a multiple of the number of rows. Hence, this function works exactly in the same way as `anova2()`. However, although the column effects are weighted for row effects, the effect of rows is not considered by the

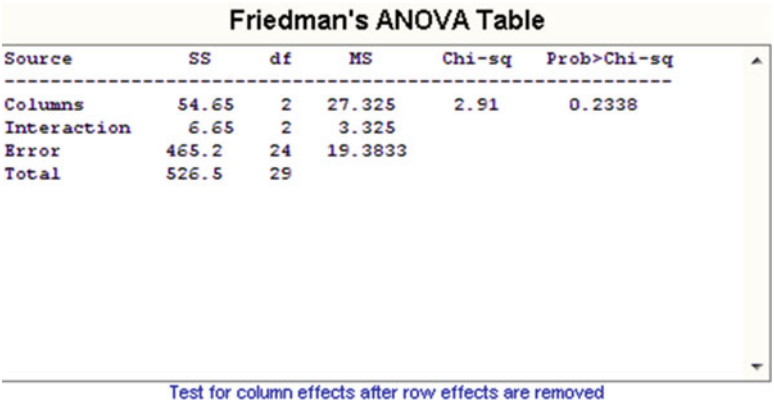


Fig. 7.12 Friedman's ANVOA table

Friedman test, and they represent nuisance effects that need to be taken into account but are not of any interest.

To show its use, let's consider again the data presented in Table 7.12 but let assume that for each group, the first five participants are females and the other five are males. The following code runs the Friedman test on two variables: Alcohol consumption and Gender.

```
>> [p table stats]= friedman ([Alcohol' NoAlcohol' Placebo'], 5)
```

Figure 7.12 shows the output of this code.

As can be seen from Fig. 7.12, the results are different from those obtained with the Kruskal–Wallis test because the gender effects have been removed.

Signal-Detection Theory (STD) Indexes

In yes/no psychophysics experiments, data can be interpreted according to signal-detection theory (Green and Swets 1966; Stanislaw and Todorov 1999). In particular, participants' responses can be coded as hits and false alarms, and the signal-detection indexes d′ (sensitivity index), β, and c (bias indexes) can be calculated. Of course, MATLAB allows this calculation. Again, the Statistics toolbox includes functions devoted to this, but we can derive some of these functions by working on the MATLAB built-in functions. In the next sections, we see the parametric indexes d′, β, and c; in the following section we show how to calculate the nonparametric indexes A′ and B″.

d′

d′ is found by subtracting the z score corresponding to the false-alarm proportion from the z score that corresponds to the hit proportion. The Statistics toolbox includes the `norminv()` function, which can be used to calculate the z scores. If the variable pHit contains the hits proportion and pFA contains the false alarms proportion, then d′ can be calculated[2] with the following code:

```
zHit = norminv(pHit) ;
zFA = norminv(pFA) ;
d = zHit-zFA;
```

However, we can obtain the same results using MATLAB built-in functions in the following way:

```
zHit = -sqrt(2)*erfcinv(2*pHit) ;
zFA = -sqrt(2)*erfcinv(2*pFA) ;
d = zHit-zFA;
```

β

β is the most-used index for estimating the subject's bias. Using the Statistics toolbox, it can be calculated as follows:

```
>> B= exp((norminv(pHit)^2 - norminv(pFA)^2)/2);
```

If you want to get the same result using the MATLAB built-in erfcinv function, then you can write the following command:

```
>> norminv (x) = -sqrt(2)*erfcinv(2*x);
```

c

Another useful index to estimate subjects' bias is c. It can be calculated as follows:

```
>> c= -(norminv(pHits) + norminv(pFA))/2;
```

A′ and B″

A′ and **B″** are the nonparametric indexes for the measure of sensitivity and bias (Pollack and Norman 1964; Grier 1971). They are very easy to calculate without using any specific function.

[2] Note that if the proportion of hits or false alarms is equal to 0 or 1, the calculation of the signal-detection indexes is undetermined. In these cases, the proportion of hits and false alarms has to be calculated in a particular way (see Hautus 1995 for further details).

The following `sensitivity_A` function takes Hits and False Alarms rates and returns the sensitivity measure A′. Here we have implemented the formula suggested by Snodgrass and Corwin (1988):

Listing 7.12

```
function A = sensitivity_A(pHit, pFA)

% A = sensitivity_A(pHit, pFA))
%
% The function returns the parametric index Ai.
%
% INPUT:  pHit is the hit rate
%       pFA is the false alarm
% OUTPUT:  Ai
% AUTHOR:  M. Borgo, A. Soranzo and M.Grassi  - 2009

if pHit>=pFA
  A = 0.5 + ((pHit-pFA)*(1+pHit-pFA))/(4*pHit*(1-pFA));
else
  A = 0.5 - ((pFA-pHit)*(1+pFA-pHit))/(4*pFA*(1-pHit));
end
```

The `bias_B` function shown in Listing 7.13 takes the Hit and False Alarm rates and returns the bias measure B″. Once again we have implemented the formula suggested by Snodgrass and Corwin (1988):

Listing 7.13

```
function B = bias_B(pHit, pFA)

% A = bias_B(pHit, pFA))
%
% The function returns the parametric index Bi.
%
% INPUT:  pHit is the hit rate
%         pFA is the false alarm
% OUTPUT:  Bi
% AUTHOR:  M. Borgo, A. Soranzo and M.Grassi  - 2011

B = (pHit*(1-pHit)-pFA*(1-pFA))/(pHit*(1-pHit)+pFA*(1-pFA));

if pHit<pFA
  B=-B;
end
```

Summary

- MATLAB is a powerful tool for statistical analysis.
- Statistics can be implemented in MATLAB using the Statistics toolbox. (Nevertheless, keep in mind that any statistic can be implemented by writing a custom function.)
- Several custom functions can be found at the MATLAB central web site.

- MATLAB can be used to calculate descriptive statistics, bivariate statistics, multivariate statistics, and inferential statistics, either parametric or nonparametric.
- MATLAB can also be used to calculate all indexes of signal-detection theory.

Exercises

1. Calculate the standard deviation of n ($n = 2^1, 2^2, \ldots 2^{10}$) random numbers taken from a normal distribution and plot the absolute value of the results. You should see how the standard deviation becomes closer to unity as n becomes larger.

   Solution:

```
for i=1:10
    StdDev(i)=std(randn(2^i, 1));
end
plot(abs(StdDev))
```

2. Suppose your subject has a hit rate of .9 and a false alarm rate of .5. Calculate the d′ and c indexes of signal-detection theory associated with proportions.

   Solution:

```
d'=1.28, c=-.64
dprime=norminv(.9)-norminv(.5);
c=-(norminv(.9)+norminv(.5))/2
```

3. Calculate a one-sample *t*-test (against zero) for 20 random numbers generated from a normal distribution. Try also to have all possible results returned by the function.

   Solution:

```
numbers = randn(20, 1);
[H, p, CI, stats] = ttest(numbers);
```

4. Suppose three groups of subjects (named A, B, and C) have produced the following results: `scoresA=rand(10, 1); scoresB=rand(20, 1)*3; scoresC=rand(15, 1);` calculate a one-way analysis of variance and test whether the three groups were different from one another. Moreover, try to get all possible outputs of the function.

   Possible solution:

```
scoresA = rand(10, 1);
scoresB = rand(20, 1)*3;
scoresC = rand(15, 1);
GroupsCoding = zeros(length(scoresA)+length(scoresB)+length(scoresC),1);
GroupsCoding(1:10) = 1;
GroupsCoding(11:30) = 2;
GroupsCoding(31:45) = 3;
[p, AnovaTab, Stats] = anova1([scoresA; scoresB; scoresC],GroupsCoding);
```

A Brick for an Experiment

In Chap. 2, we saw how to import a data file and how to calculate simple statistics on the data of our brick experiment. However, we need to perform a more complex statistical analysis to understand the real outcome of the experiment. The experiment described in the brick is a 2 by 2 within-subjects design experiment. We can therefore see whether there are differences in the number of bounce responses observed for the continuous vs. stopped motion display as well as for the silent vs. with sound display. We can do this with a two-way analysis of variance. To perform the analysis of the brick experiment we will use a function that is freely available from the MATLAB central web site (http://www.mathworks.com/matlabcentral/). This web site is a large community of MATLAB users who exchange function files as well as problems (and often the problems' solutions). The function can be found by searching "two way repeated measures ANOVA" in the search engine. The function's name is rm_anova2. This function expects five input parameters. The first four parameters are numbers, and they are all single-dimensional arrays. One array contains the dependent variable (i.e., the probability of bounce responses of each subject). The other three arrays contain the variables' coding and the number of repetitions (i.e., the subjects). All arrays must have identical lengths. Now we write a short code that stores and sorts all four arrays. Here we hypothesize that we have run ten subjects.

Listing 7.14

```
motion_cond = [1, 2];
snd_cond = [1, 2];
subjects = 1:10;

X = zeros(length(motion_cond)*length(snd_cond)*length(subjects), 1);
factor1 = X;
factor2 = X;
repetitions = X;

m=1;
for i = 1:length(motion_cond)
   for j = 1:length(snd_cond)
      for k = 1:length(subjects)
         factor1(m)=i;
         factor2(m)=j;
         repetitions(m)=k;
         X(m) = mean(data(data(:, 5)==motion_cond(i) &
data(:,6)==snd_cond(j) & data(:, 1)==repetitions(k), 7));
         m = m + 1;
      end
   end
end
```

Now data within the X array are sorted according to the variable codes contained within the f1, f2, and repetitions variables. We have now to run the analysis of variance.

```
>> stats = rm_anova2(X, repetitions, factor1, factor2, {'motion',
'sound'});
```

Note that motion and sound are two labels that later become useful for easily reading the results of the analysis returned by the function. These labels need to be passed to the function within a cell type variable. If you now type stats at the MATLAB prompt, you will see the results of the experiment.

References

Green DM, Swets JA (1966) Signal-detection theory and psychophysics. Wiley, New York

Grier JB (1971) Nonparametric indexes for sensitivity and bias: computing formulas. Psychol Bull 75:424–429

Hautus MJ (1995) Corrections for extreme proportions and their biasing effects on estimated values of d′. Behav Res Methods Instrum Comput 27:46–51

Intraub H, Richardson M (1989) Wide-angle memories of close-up scenes. J Exp Psychol Learn Mem Cogn 15:179–187

Pollack I, Norman DA (1964) A nonparametric analysis of recognition experiments. Psychon Sci 1:125–126

Simons DJ, Chabris CF (1999) Gorillas in our midst: sustained inattentional blindness for dynamic events. Perception 28:1059–1074

Snodgrass JG, Corwin J (1988) Pragmatics of measuring recognition memory: applications to dementia and amnesia. J Exp Psychol Gen 117:34–50

Stanislaw H, Todorov N (1999) Calculation of signal detection theory measures. Behav Res Methods Instrum Comput 31:137–149

Suggested Readings

The journals *Psychological Methods* and *Behavior Research Methods* often suggest statistical tools that are either implemented in that MATLAB environment or can be easily implemented in MATLAB.

Martinez WL, Martinez AR (2005) Exploratory data analysis with MATLAB. Boca Raton, FL: Chapman & Hall/CRC

Marques de Sa JP (2007) Applied statistics using SPSS, STATISTICA, MATLAB and R, 2nd edn. Springer