Packages

The packages menu is very important, as it is the easiest way to load and install packages to the R system. Therefore the entire section following this is devoted to demonstrating how to use this menu.

Windows

The Windows menu provides options for cascading, and tiling windows. If there is more than one window open (for example, the console and a help window) you can use the open Windows list on the bottom of this menu to access the different open windows.

Help

The Help menu directs you to various sources of help and warrants some exploration. The first option, called "Console" pops up a dialog box listing a cheat sheet of "Information" listing various shortcut keystrokes to perform tasks for scrolling and editing in the main console window.

The next two options provide the FAQ (Frequently Asked Questions) HTML documents for R and R for the operating system you are using. These should work whether or not you are connected to the internet since they are part of the program installation. The FAQ documents provide answers to technical questions and are worth browsing through.

The next section on the help menu contains the options "R language (standard)", "R language (HTML)" and "Manuals". "R language (standard) pops up the help dialog box in Figure 2-5. This will popup the help screen for the specified term, provided you enter a correct term (which can be hard if you don't know ahead of time what you're looking for). This can also be accomplished using the help () command, as we will see in the next chapter.

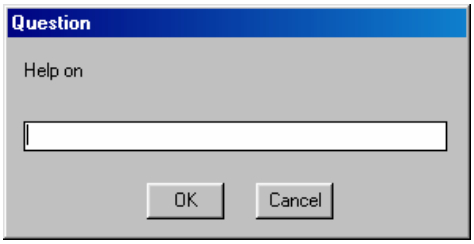


Figure 2-5

The menu option "R language (HTML)" will produce some HTML based documents containing information and links to more documentation. This should be available off-line as part of the R installation. The next option "Manuals" provides a secondary menu with several pdf files of R documents.

The remaining options on the Help menu are “Apropos” and “About”. “Apropos” pops up a dialog box similar to the help box depicted in Figure 2-5 but that you only need to enter a partial search term to search R documents. “About” pops up a little dialog box about R and the version you are using.

One of the most difficult tasks in R is finding documentation to help you. R is actually very extensively documented and only a fraction of this documentation is available directly using the help menu. However, much of the documentation is technical rather than tutorial, and geared more toward the programmer and developer rather than the applied user. More about getting help is discussed in the next chapter.

The Toolbar

Below the menu bar is the toolbar, depicted in Figure 2-5. This provides quick access icons to the main features of the menu bar. If you scroll over the icons with your mouse slowly you will get rollover messages about the feature of each icon. The stop icon can be useful as a panic button providing the same functionality as the Misc menu’s “Stop current computation” option.

Figure 2-5

Packages

The basic R installation contains the package base and several other packages considered essential enough to include in the main software installation. Exact packages included may vary with different versions of R. Installing and loading contributed packages adds additional specialized functionality. R is essentially a modular environment and you install and load the modules (packages) you need. You only need to install the packages once to you system, as they are saved locally, ready to be loaded whenever you want to use them. However

The easiest way to install and load packages is to use the Packages menu, although there are equivalent commands to use as well if you prefer the command line approach.

Installing Packages

In order to use an R package, it must be installed on your system. That is you must have a local copy of the package. Most packages are available from the CRAN site as contributed packages, and can be directly downloaded in R. In

Table 4-1: Arithmetic Operators

Operator	Functionality
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Raised to a power

Logical and Relational Operators

Logical and relational operators are used when you want to selectively execute code based on certain conditions. Table 4-3 lists the commonly used logical and relational operators. Using logical and relational operators is a form of flow control to determine the action the program will take. Essentially flow control of a program can be thought of as being in three layers – order (sequence of code written), selection (use of logical and relational operators), and repetition (or looping). Order is self-explanatory, selection is discussed in this section, and repetition is covered in the next section.

Table 4-3: Logical and Relational Operators

Operator	Functionality
&	And
|	Or
!	Not
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal

Table 5-1: Some Data Summary Functions

Function name	Task performed
sum(x)	Sums the elements in x
prod(x)	Product of the elements in x
max(x)	Maximum element in x
min(x)	Minimum element in x
range(x)	Range (min to max) of elements in x
length(x)	Number of elements in x
mean(x)	Mean (average value) of elements in x.
median(x)	Median (middle value) of elements in x
var(x)	Variance of elements in x
sd(x)	Standard deviation of element in x
cor(x,y)	Correlation between x and y
quantile(x,p)	The p^{th} quantile of x
cov(x,y)	Covariance between x and y

Let's apply some of these functions using an example.

```
> x<-c(0.5,0.2,0.24,0.12,0.3,0.12,0.2,0.13,0.12,0.12,0.32,0.19)
> sum(x)
[1] 2.56
> prod(x)
[1] 2.360122e-09
> max(x)
[1] 0.5
> min(x)
[1] 0.12
> range(x)
[1] 0.12 0.50
> length(x)
[1] 12
> mean(x)
[1] 0.2133333
> median(x)
[1] 0.195
> var(x)
```

As illustrated with the plots in Figures 5-6 and 5-7, even relatively simple plots in R can require quite a few lines of code and use various parameters. Most of the graphical examples in this book – and there are many of them - will use the simplest plotting code possible to illustrate examples, since our focus is on understanding techniques of data analysis. However, the graphic code in R can be as complicated as you wish, and only a snapshot of R's full graphic capabilities have been presented here. R allows for the user to code virtually every detail of a graph. This may seem complicated, but it is a useful capability. With a little practice, you can code R to produce custom, publication quality graphics to effectively illustrate almost any data analysis result.

Saving Graphics

Notice that when the graphics window is active the main menu is different, as illustrated in Figure 5-8. On the File menu there are many options for saving a graphic, including different graphical formats (png, bmp, jpg) and other formats (metafile, postscript,pdf). You could also use command line functionality to save, but using the options under the File menu is easier and pops up a save as dialog box allowing you to choose the directory you are saving the graphic file to.

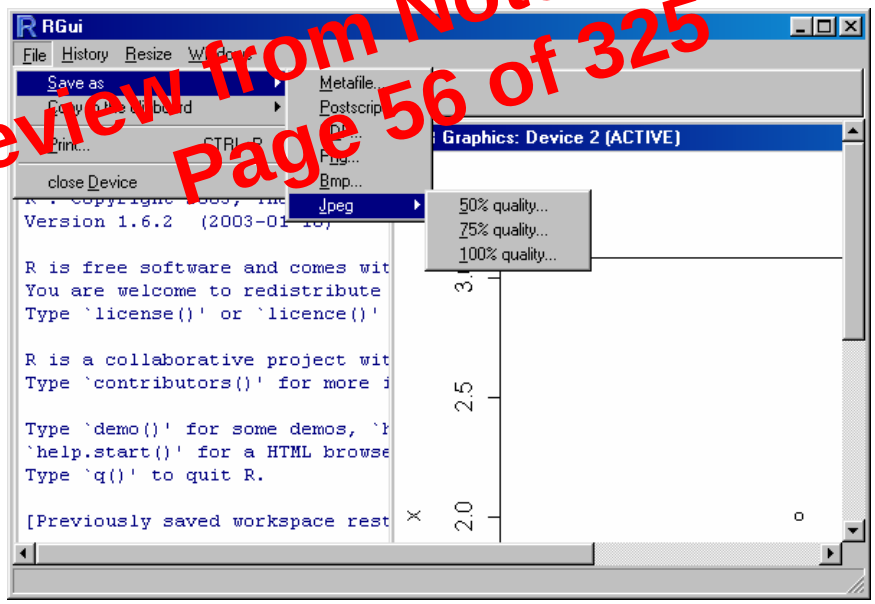


Figure 5-8

Another option to save a graphic is to simply right mouse click on the graphic, which will produce a pop up menu with options to copy or save the graphic in various formats as well as to directly print the graphic. In a Windows

Discrete versus Continuous Random Variables

Random variables can be discrete or continuous. Discrete random variables are used when the set of possible outcomes (sample space) for an experiment is countable. Although many discrete random variables define sample spaces with finite numbers of outcomes, countable does not mean finite. The outcomes can be countably infinite (the integers are countably infinite because they are discrete and go on forever). Examples of experimental outcomes that are modeled with discrete random variables include numbers of people standing in a line, number of A's in a nucleotide sequence, and the number of mutations, which occur during a certain time interval.

A random variable that is not discrete but can assume any value, usually within a defined interval, is defined as a continuous random variable. Measurable quantities are generally modeled with continuous random variables. Examples of experimental outcomes that can be modeled with continuous random variables are: height, weight, intensity of a signal, and time required for a radioactive substance to decay.

Because much of the information bioinformatics deals with is discrete data (sequence information is usually analyzed using discrete random variables) the emphasis of this book is on this type of data. However continuous random variables are not ignored and play an important role in some areas of bioinformatics, especially in Bayesian statistics and in discriminacy analysis.

Probability Distributions and Densities

Now with an understanding of the concept of a random variable, whether discrete or continuous, we can talk about probability models. In general terms, probability models are assumed forms of distributions of data. A probability model fits the data and describes it. Sometimes the fit is empirical such as the example above. Often the data is fit to a distribution of known form (to be discussed in the next two chapters) such as a beta or gamma distribution, other times in more complex scenarios the data is fixed to a distribution that is a mixture of known forms.

Every random variable has an associated probability distribution function. This function is called a probability mass function in the case of a discrete random variable or probability density function in the case of a continuous random variable. The distribution function is used to model how the outcome probabilities are associated with the values of the random variable. In addition all random variables (discrete and continuous) have a cumulative distribution function, or CDF. The CDF is a function giving the probability that the random variable X is less than or equal to x, for every value x, and models the accumulated probability up to that value.

For simple discrete random variables, the associated probability distributions can be described using a table to "model" the probability as above for the RNA analysis example, or alternatively a graph can be used. In R a simple histogram (show in Figure 6-9) can be used to model the probability distribution function for this example.

```
> X<-c(0,1,2,3)
> Prob<-c(0.208,0.167,0.25,0.375)
> N<-c ('A', 'C', 'G', 'U')
> barplot(Prob,names=N,ylab="Probability", main="RNA Residue Analysis")
```

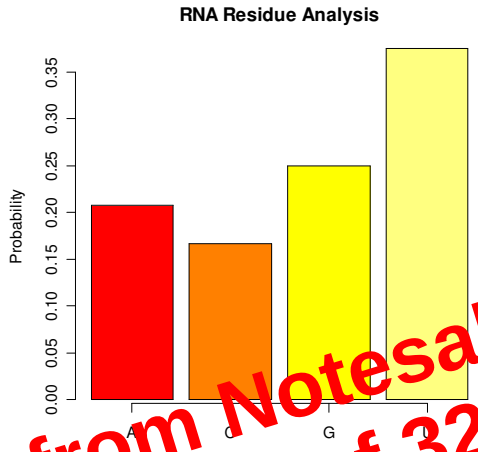


Figure 6-9: Histogram Illustrating the Probability Mass Function for RNA Residue Example

To find the cumulative distribution value for this example, simply add up the probabilities for each value of X for 0,1,2,3 and the value the CDF is the probability that random variable X assumes that or a lesser value. For example if X equals 2, the CDF is the probability that X=2 or X=1 or X=0. To calculate this simply total the values for P(X=2) plus P(X=1) plus P(X=0). For our RNA residue example, the calculations for the CDF are shown in Table 6-2.

Table 6-2: Probability Distribution and Cumulative Probability Distribution for RNA Residue Analysis Example

Residue	Value of X (=x)	P (X=x)	F(x)= P(X ≤ x)
A	0	5/24=0.208	0.208
C	1	4/24=0.167	0.375
G	2	9/24=0.375	0.625
U	3	6/24=0.25	1

step) and the F (x) is the interval from negative infinity (or wherever x is defined) to the value of x.

Empirical CDFs and Package stepfun

A simple method for drawing preliminary conclusions from data about an underlying probability model is the plotting of the empirical CDF. For calculating the empirical CDF from n data values we assign a probability of 1/n to each outcome and then plot the CDF to this set of probabilities. A useful R package when working with empirical data, particularly discrete data, to determine and plot empirical CDF is a package called stepfun. This package contains functions that will easily generate an empirical CDF given any data vector, and also contains functions to create CDF plot easily.

For example, suppose we collect data on how many times we spot the sequence ATC in 10 randomly chosen 100 base pair DNA stretches and get (2,4,2,1,3,4,2,1,3,5) as the result and went to obtain an empirical CDF for the distribution of this data. The data can simply be entered and the plot stepfun function used to easily generate a CDF plot, as depicted in Figure 6-11. Stepfun makes plotting CDF's and related graphs much easier.

```
> x<-c(2,4,2,1,3,4,2,1,3,5)
> plot.stepfun(x)
```

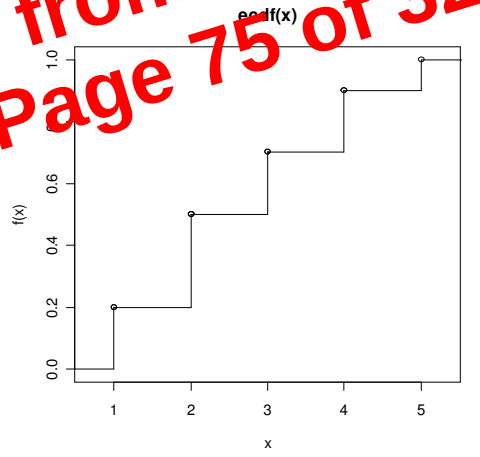



Figure 6-11: CDF Plot Example Produced Using Stepfun

Parameters

The most general definition of a parameter is "some constant" involved in a probability distribution, which although vague is actually a good definition. Random variables define the data in a probability model. Parameters serve to

This concludes discussion, for now, of discrete univariate probability distributions. You should have a feel for these distributions and how to work with them in R. These distributions will be used in applications in later chapters.

Univariate Continuous Distributions

Univariate Normal Distribution

The normal distribution is the typical bell curve distribution used to characterize many types of measurable data such as height, weight, test scores, etc. The normal is also the distribution that is used to model the distribution of data that is sampled, as will be discussed later in this book under the topic of inferential statistics. Sometimes the normal distribution is called the Gaussian distribution, in honor of Karl Gauss. It is a ritual that all introductory statistics students are saturated with details about the normal distribution, far more than will be covered here. The probability density equation for the normal distribution, presented below, should ring a bell of familiarity to graduates of statistics courses:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{\sigma^2}}$$

In the equation above, the Greek letter mu (μ) represents the mean of the distribution (aka: average value, expected value) and the Greek letter sigma (σ) represents the standard deviation of the distribution (sigma squared is the variance). Mu and sigma serve as the parameters for the distribution. For the normal distribution, the location and scale parameters correspond to the mean and standard deviation, respectively. However, this is not necessarily true for other distributions. In fact, it is not true for most distributions.

One of the tricks with the normal distribution is that it is easily standardized to a standard scale. If X is a continuous random variable with mean mu and standard deviation sigma it can be standardized by transforming X to Z where Z is a normally distributed variable with mean 0 and standard deviation 1 (which also equals the variance since $1^2=1$). This is useful if you have a bunch of different X's and want to put them all on the same Z system so you can compare them, with a scoring system called Z-scores (see your favorite introductory statistics book for further discussion). The transformation of X to Z is simply:

$$Z = \frac{X - \mu}{\sigma}$$

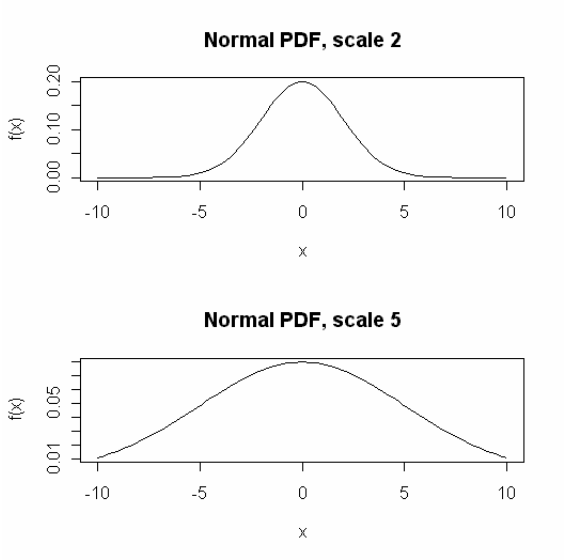


Figure 7-9: Changing the scale parameter (standard deviation) in the normal distribution

The normal also has a cumulative density function, which in R utilizes the pnorm function with the same parameters as shown. The code below computers some normal CDFs, using a few different scale paramters with the same mean of 0.

```
> par(mfrow=c(3,1))
> plot(x,pnorm(x,0,1),xlab="x",ylab="f(x)", type='l', main="Normal CDF scale
1")
> plot(x,pnorm(x,0,2),xlab="x", ylab="f(x)", type='l', main="Normal CDF
scale 2")
> plot(x,pnorm(x,0,5),xlab="x", ylab="f(x)", type='l', main="Normal CDF
scale 5")
```

```
data <- c(4.75, 3.4, 1.8, 2.9, 2.2, 2.4, 5.8, 2.6, 2.4, 5.25)
n <- length(data)
x <-seq(0,8,length=200)
plot(x,dgamma(x,shape=5.6,scale=0.6),type='l',ylab="f(x)")
points(data,rep(0,n))
```

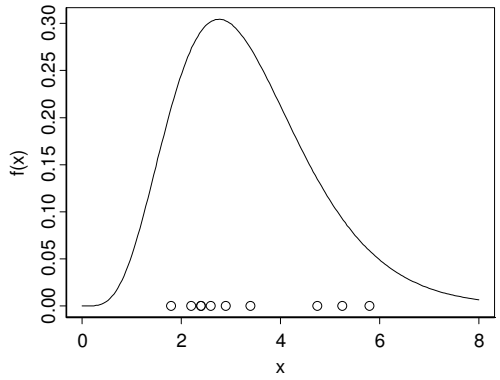


Figure 7-15: Comparing the data distribution to the gamma fit

As with other distributions the cumulative distribution function for the gamma is designated starting with a p, in this case pgamma. The CDF for the model used in this example can simply be graphed in R as:

```
plot(x,pgamma(x,shape=5.6,scale=0.6),type='l',ylab="P(X<=x)",
+ main="Gamma CDF Fit")
points(data,rep(0,n))
```

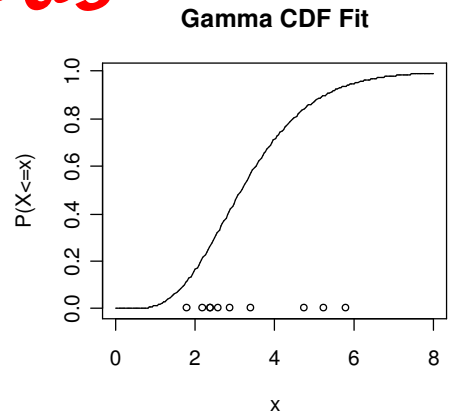


Figure 7-16:Gamma CDF

It looks from the CDF plot in Figure 7-16 that there may still be some probability density at x values higher than 5. We can perform a simple calculation to check this out.

8

Probability and Distributions Involving Multiple Variables

The previous two chapters have looked at the basic principles of probability and probability distributions of one random variable. In this chapter we introduce some more concepts of probability and then extend looking at probability distributions to include distributions modeling two or more random variables.

Expanded Probability Concepts

Conditional Probability

Conditional probability is a powerful concept that allows us to calculate the probability of an event given that some prior event, which we have probability information about, has occurred. Using the concept of conditional probability allows us to solve problems where "things happen sequentially" with rather simple probability models, instead of complicated mathematical models that would be the alternative if it were not for conditional probability. Understanding conditional probability, as we will see in the next chapter, is an essential foundation for Bayesian statistics. But understanding the concept is also of importance on its own.

Let's illustrate the use of conditional probability with an example from classical genetics by considering the case of pea color as a trait encoded by one gene that has two alleles. The dominant allele, which we will denote by Y, codes for yellow pea color. The recessive allele we will denote by y, which codes for green pea color.

You may look at this and wonder, why? The logic for this alternative definition of independence comes from the definition of conditional probability:

$$P(E|F)=\frac{P(E\cap F)}{P(F)}$$

This can be algebraically rewritten as

$$P(E\cap F)=P(E|F)P(F)$$

But since we just defined the independence of E and F as P(E|F)=P(E) this simplifies to

$$P(E\cap F) =P(E)P(F)$$

This form of the definition of independence comes in very handy for calculating joint probabilities of independent events.

It is important to note here that determining that events are independent is not equivalent to determining that events are disjoint or mutually exclusive, which was previously defined by two events having no common intersection ($P(E\cap F)= \varnothing$). Disjoint and mutually exclusive mean the same thing, but independence is a very different concept.

Independence can easily be extended to include more than two events. Three events A, B, and C are independent if $P(A\cap B\cap C)=P(A)P(B)P(C)$. In this case we can say that A, B, and C are mutually independent and independence of pairs of these events can be concluded (A is independent of B, B is independent of C, etc.). However, it is not always the case that the reverse is true, and it is possible to have three events, A, B, and C where A and B are independent, B and C are independent but A and C are not independent and therefore A, B, and C are not mutually independent.

In practice, determining whether events are independent can be tricky. Some times it's based on common logic. For example, most people would agree that the outcomes for each toss of a fair coin are independent, meaning the outcome of one toss of a coin (heads or tails) has no impact on the next toss of a coin. But in general, you should not assume independence without good reason to do so.

Independence is often utilized in bioinformatics in analyzing sequence information. Although this issue is often debatable, assuming independence of sequence elements is key in many data analysis algorithms commonly used. Independence makes calculations easy and the assumption of independence can greatly simplify a complicated algorithm.

For example, suppose nucleotides in a DNA sequence are mutually independent with equal probabilities (that is, P(A)=P(T)=P(C)=P(G)=1/4). The probability

Using combinatorics, we can calculate the number of possible divisions of n sequences into r groups of size x1, x2…xr with what is called the multinomial coefficient. This can be written as:

$$\binom{n}{x_1, x_2, \ldots x_r} = \frac{n!}{x_1! x_2! \ldots x_r!}$$

Combining these results produces the joint distribution of observed events (a formula that directly parallels the binomial case of two possible outcomes described in the previous chapter) under the multinomial model.

$$p(x_1, x_2, \ldots, x_r) = \binom{n}{x_1 x_2 \ldots x_r} p_1^{x_1} p_2^{x_2} \cdot \ldots \cdot p_r^{x_r}$$

Among its many applications in bioinformatics, the multinomial model is frequently used in modeling the joint distribution of the number of observed genotypes. Any number of loci and any number of alleles can be modeled this way, but the simplest example is the case of looking at a genetic locus which has two alleles, A and a. If we sample n diploid individual in the population and record their genotype at that locus, a number of individuals will be of genotype AA, which we can represent as just as n_{AA}. Likewise, a number of individuals will have Aa genotype and can be represented by n_{Aa}, and the number of individual of aa genotype can be represented by n_{aa}. To formalize this into a probability model, we can use the random variable X to represent n_{AA}, the random variable Y to represent n_{Aa}, and the random variable Z to represent n_{aa}. We can label these proportions (probabilities) as P_{AA}, P_{Aa}, and P_{aa} for each of the three respective possible genotypes.

The multinomial distribution formula represents the joint distribution of the three genotypes is given below.

$$P(X=n_{AA}, Y=n_{Aa}, Z=n_{aa}) = \frac{n!}{n_{AA}! n_{Aa}! n_{aa}!} (P_{AA})^{n_{AA}} (P_{Aa})^{n_{Aa}} (P_{aa})^{n_{aa}}$$

Since you probably wouldn't want to perform paper and pencil calculations using this formula, the question now is how would you work with such a model in R? Clearly models with 3 random variables are not as simple to work with as univariate models, but R can handle analyzing and performing simulations on these more complicated distributions quite easily.

As an example, suppose we have 20 individuals and genotype them and find that $n_{AA}=4$, $n_{Aa}=14$, and $n_{aa}=2$. Given this information, we can easily estimate our parameters for the multinomial distribution by simply using the sample proportions $P_{AA}=0.2$, $P_{Aa}=0.7$ and $P_{aa}=0.1$. Since we do not have a lot of data it is difficult to examine the properties of this model. However, using our empirical parameters we can extend our data set by doing simulations of more

```
> results2<-results/(results[,1]+results[,2]+results[,3])
> results2

 [1,] 0.20 0.55 0.25
 [2,] 0.20 0.65 0.15
 [3,] 0.10 0.60 0.30
 [4,] 0.15 0.65 0.20
 [5,] 0.20 0.65 0.15
 [6,] 0.30 0.55 0.15
 [7,] 0.15 0.65 0.20
 [8,] 0.20 0.75 0.05
 [9,] 0.05 0.65 0.30
[10,] 0.35 0.35 0.30
```

Looking at the proportions makes it clearer that the simulated values are indeed based on the empirical proportion parameters (0,2,0.7,0.1) supplied.

You could write your own functions like the above to sample from multinomial distributions, but there is a package called combinat that contains some pre-written functions to sample from multinomial distributions. This package also contains a number of other functions useful in combinatorial calculations.

Note that if you were interested in doing some statistical tests, you could simulate values from distributions with alternative parameters, and then perform tests to determine whether the empirical values differ from this theoretical distribution. For example, you could test the empirical values against a theoretical population with parameters P_{AA}=0.25, P_{Aa}=0.5, and P_{aa}=0.25. This will not be done here because it requires techniques of inferential statistics not yet discussed, but is presented here to illustrate some of the powerful applications you can perform using simulations of distributions.

The marginal distributions for each of the random variables X, Y and Z can easily be obtained from the multinomial. Suppose we are interested only in the marginal probability mass function of the random variable X? We could go about finding the marginal probability mass function using lots of messy algebra or instead we consider the following argument.

If we are only interested in the number of outcomes that result in the first type, *X*, then we simply lump all the other types (Y and Z) into one category called "other". Now we have reduced this to a situation we have two outcomes. This should ring a bell of familiarity, as it has now become a case of Bernoulli trials. The number of times genotype X occurs resulting from n independent trials follows a Binomial probability distribution with parameters n and p_1. Note that the probability of "failure" = prob("other") = 1 - p_1 = p_2 +…+ p_r (sum of all the others). Thus, the one-dimensional marginal distribution for a multinomial distribution is simply a binomial:

The denominator of Bayes' rule P(B) is the marginal probability of event B, that is the probability of event B over all possibilities of A where there is joint probability. In the case where A is not a single event, but a set n of mutually exclusive and exhaustive events, , such as a set of hyptheses, we can use the law of total probability to calculate P(B):

$$P(B)=\sum_{n} P(B \mid An) * P(An)$$

In this situation, Bayes' rule provides the posterior probability of any particular of these n hypotheses, say Aj given that the even B has occurred:

$$P\left(A_j \mid B\right)=\frac{P\left(B \mid A_j\right) P\left(A_j\right)}{\sum_{n} P(B \mid An) * P(An)}$$

Because for a given situation, P(B) is a constant, Bayes theorem may be written as:

$$P(A|B) \propto P(B \mid A) * P(A)$$

Where $\propto$ is the symbol for "proportional to".

Sometimes Bayes' rule is written using E and H, where E stands for "evidence" and H stands for "hypothesis". Using E and H we can write:

$$P(H|E)=\frac{P(E \mid H)P(H)}{P(E)}$$

In this form, P(H) represents the prior degree of belief in the hypothesis before the evidence. P(H|E) is the updated probability of belief in the hypothesis given the evidence. In other words, Bayes' rule is updating the degree of belief in the hypothesis based on the evidence. This is where the usefulness of Bayes rule and Bayesian statistics in learning comes from, and this idea is a foundation of the usefulness of Bayesian statistics.

Applying Bayes' Rule

Let's apply Bayes' rule to two examples. In the first case, we will have complete information about the joint probability of two events. In the second case, we will have only select probability information to work with.

Table 9-1 shows the joint probability of two events, event A being a membrane bound protein and event B having a high proportion of hydrophobic (amino acid) residues. The two columns with data represent the marginal distributions of A, being a membrane bound protein, and the complement of A (written as ~A

computational complexity involved. Multiple parameter models, extensive amounts of data, can all be worked with using this same model.

The rest of this chapter discusses this algorithm further, taking a closer look at the prior choices, what a likelihood function is, and how the posterior is calculated.

Priors

As stated earlier, the use of the prior in the model is the controversial and distinctive element of Bayesian statistics. The use of a prior introduces subjective information from the statistician into the model and there are not hard and fast rules as to the "correct" prior to use as it is more of an educated guessing game. This is often criticized by non-Bayesians as being too subjective for their liking. In the coin-tossing example above, the prior was an estimate lower than the empirical data suggested, so based on that you may argue that the prior biased the data and the result would have been more accurate if no prior were used in calculating the posterior. However, what if the prior estimate for the proportion of heads was 0.5 and the empirical data result for the proportion of heads in a 10 toss trial is 0.3? In this case, if the coin is fair, the prior is more accurate than the data and the posterior would be much closer to the true value than it would be if it were for the data alone and the posterior accuracy would increase with the use of the prior, a counter argument to the criticisms of the non-Bayesians. Another counterargument is that there is always subjectivity in any model – data can be collected through a variety of experiments with different biases. Everything is subjective.

Priors however are not just about introducing subjective knowledge into the model; they are about introducing previous knowledge into the model. The Bayesian algorithm is all about updating previous knowledge with new evidence. Previous knowledge may be entirely subjective – based on a feel or a guess about what the outcome of a novel experiment may be, often in regard to the potential outcome of an experiment that has never been done. Prior knowledge however may also be knowledge and based on data from a previous similar empirical experiment. Meta analysis models can be done in Bayesian combining data from new experiments with data from old experiments.

The choice of a particular prior for a model is a science in and of itself. We will only deal with the simplest models of standard form, merely touching on the vast world of Bayesian models. Advanced model choices for priors are left to the experienced statistician and studying prior choices could be the topic of several good dissertation projects. However, in the Bayesian model the use of a prior is key. Some prior model must be chosen when using a Bayesian method. Sometimes, as in the example above the choice of prior is simple because it follows a convenient model pattern, as is the case with the beta prior for the binomial model. Such priors are called conjugate priors. Other times the basis of a prior is made for mathematical reasons (beyond our discussion) in a

What would happen in this example if instead of 10 data points we obtained 100 data points, with 50 times the coin landing on its head? For the posterior we can use a beta distribution with parameters alpha (new)=k+alpha(old), beta(new)=n-k+beta(old). With n=100 and x=50 we get alpha (new)=52 and beta (new)=55. Let's compare the results with n=10 with the resulting posterior distribution obtained with n=100 and the same success rate in the coin toss:

```
> p <- seq(0, 1, by = 0.001)
> plot(p, dbeta(p, 2, 5), ylim = range(0:10), ylab = "f(p)",
+ type = "l", main = "Effect of n on Posterior")
> lines(p, dbeta(p, 7, 10))
> lines(p, dbeta(p, 52, 55))
> legend(0, 4, "Prior")
> legend(0.2, 6, "Post.1")
> legend(0.5, 9, "Post.2")
```

Figure 9-8 shows the result of this analysis. Note that the second posterior is centered on the data value of p=0.5 and is also much more "precise" on this value with less spread. This reflects the general rule that the more the data the more the data affect the posterior given the same prior.

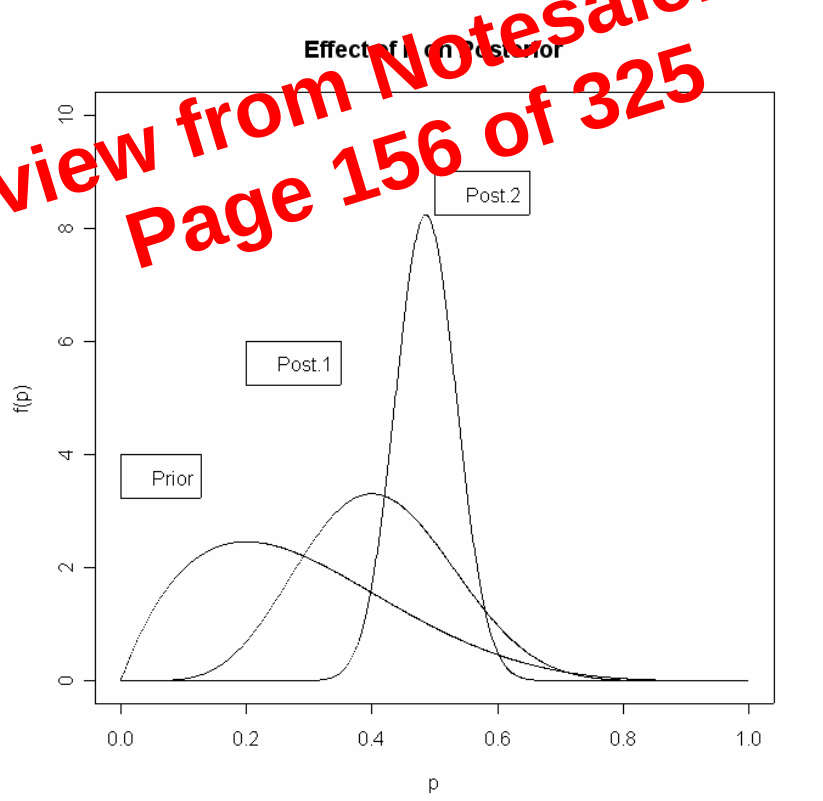


Figure 9-8

Often in our analysis we are concerned only with some parameters (so-called main parameters) and not others (so-called nuisance parameters). Nuisance parameters can be eliminated easily in Bayesian models by calculating marginal and conditional probability distribution for the main parameters Rather than introduce techniques here for dealing with multiparameter situations we shall only note that they exist and we will work with them in coming chapters when we learn computationally intense algorithms that do most of the work for us in evaluating such models to understand the parameter of interest. In particular we will introduce a chapter covering some special algorithms that may be used to evaluate multiparameter Bayesian models. We also cover Markov chain Monte Carlo methods, which are of growing use and popular in bioinformatics applications. Working with multiparameter models simply builds on the concepts used with single parameter models and an understanding of these basic concepts is essential for proceeding further in working with these models

Applications of Bayesian Statistics in Bioinformatics

Although coverage in this book is minimal and introductory, the hopes is that this will entice the reader to study more advanced statistical methods and fully appreciate the power of Bayesian methods in bioinformatics. Also related are artificial intelligence related subjects, logic, and of course biology related areas.

Some areas where Bayesian techniques are being researched for applications include:

- Prediction of protein structure and function
- RNA structure prediction
- Mechanisms of mammalian regulation
- Prokaryotic regulatory networks
- Large scale data analysis methods
- Algorithms for detecting subtle signals in DNA

The potential applications of Bayesian data analysis methods are without limit, and there is little doubt that such techniques will become increasingly common and have an increasing number of applications in the future. The appendix lists some reference papers from the literature where Bayesian methods are used.

R is a software tool that is adaptable, programmable, and powerful and very useful for working with complicated data models and for doing necessary calculations for estimation and hypothesis testing, both, frequentist and Bayesian.

randomly walks for n cycles. In the upper left corner, n=10, the samples are sequentially joined. Each successive sample is conditionally dependent on another in what is a Markov process, described later in this chapter. As the iterations continue the process produces a more refined approximation to the desired posterior distribution. The sampler used in this figure is called the Gibbs sampler, one of the algorithms discussed in Chapter 11 and the algorithm used by the software program WinBugs, which can be used as an accessory program with R and is discussed in Chapter 12.

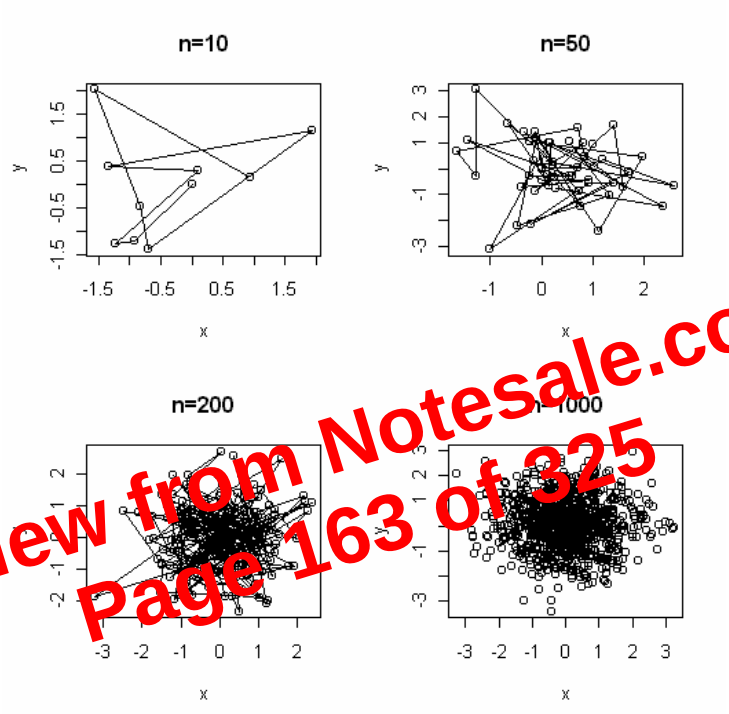


Figure 10-1: Posterior MCMC Simulation of a Bivariate Normal Distribution

By 1000 cycles the simulation in Figure 10-1 clearly resembles Figure 8-7, which depicts a directly simulated bivariate normal distribution plotted in the same way. Although the bivariate normal is not a very complicated distribution, hopefully this illustration has convinced you that MCMC techniques are capable of producing a posterior distribution from which analysis of posterior data and parameters can be performed.

Utilizing MCMC techniques require an understanding of Bayesian theory (covered in Chapter 9), Stochastic and Markov Processes described in this chapter, the algorithms covered in Chapter 11, and can be implemented using R and the auxiliary software tools introduced in Chapter 12. The coverage in this book is far from comprehensive, and serves as only a minimalist introduction to

probabilities and how to mathematically (using R) manipulate the matrix to compute transition probabilities for k>1 subsequent states and understanding the idea of convergence to a stationary state.

Let's investigate these concepts with more mathematical detail using a second, slightly more complex model involving a DNA sequence.

Modeling A DNA Sequence with a Markov Chain

Specifying the Model

The first step in working with a Markov Chain model is to determine what the model is. Consider again Figure 10-3, which could be part of any DNA sequence of interest. Each place in the sequence can be thought of as a state space, which has four possible states {A, T, C, G}. Each position in the sequence can be represented with a random variable X_0, X_1,…X_n that takes a value of one of the states in the state space for a particular place in the sequence. If we follow Figure 10-3 and go in the 5' to 3' direction then X_0=A, X_1=T and so forth.

The model we are interested in is that nucleotide dependent on the prior nucleotide and only the prior nucleotide. In other words, the nucleotide sequence is not a series of independent nucleotides, but that each nucleotide is dependent on the nucleotide immediately upstream. In other words, for the sequence:

5' ATTGCC 3'

we can express the probability of this sequence using the Markov Property as follows:

$$P(X5=C|X4=C,X3=G,X2=T,X1=T,X0=A)=P(X5=C|X4=C)$$

Setting up the Transition Probability Matrix

To set up the matrix of transition probabilities, we need some idea of the initial probability distribution. Perhaps we sequence a few 100 base pair stretches from the DNA sample in question and determine the following for the probability of nucleotides adjacent to each other. We can use this as our one-step transition matrix to start the chain with

$$\begin{array}{c} \\ A \\ T \\ C \\ G \end{array}\begin{bmatrix} 0.3 & 0.2 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.4 & 0.3 \\ 0.2 & 0.2 & 0.2 & 0.4 \\ 0.1 & 0.8 & 0.1 & 0 \end{bmatrix}$$

```
> DNA8<-DNA4%*%DNA4
> DNA8
          [,1]      [,2]      [,3]      [,4]
[1,] 0.1556210 0.3497508 0.2450089 0.2496193
[2,] 0.1556207 0.3497695 0.2450017 0.2496081
[3,] 0.1556171 0.3497173 0.2450332 0.2496324
[4,] 0.1556375 0.3498298 0.2449286 0.2496041
> DNA16<-DNA8%*%DNA8
> DNA16
          [,1]      [,2]      [,3]      [,4]
[1,] 0.1556240 0.3497689 0.2449923 0.2496148
[2,] 0.1556240 0.3497689 0.2449923 0.2496148
[3,] 0.1556240 0.3497689 0.2449923 0.2496148
[4,] 0.1556240 0.3497689 0.2449923 0.2496148
```

DNA16 (P^{16}) appears to have converged and represents a stationary distribution. Just to be sure we run a few more powers….

```
> DNA32<-DNA16%*%DNA16
> DNA64<-DNA32%*%DNA32
> DNA64
          [,1]      [,2]      [,3]      [,4]
[1,] 0.1556240 0.3497689 0.2449923 0.2496148
[2,] 0.1556240 0.3497689 0.2449923 0.2496148
[3,] 0.1556240 0.3497689 0.2449923 0.2496148
[4,] 0.1556240 0.3497689 0.2449923 0.2496148
```

We can use the converged transition matrix to conclude that our stationary distribution of nucleotides, which we represent with the letter pi, is:

$$\pi = [0.15 \quad 0.35 \quad 0.25 \quad 0.25]$$

We can interpret this as given our initial distribution the posterior distribution of nucleotides is 15% A, 35%T, 25%C and 25%G, regardless of position. We can also use this distribution to calculate expected values of the numerical distribution of nucleotides. In a sample of 1000 nucleotides from this sample, we would expect 150 A, 350 T and 250 to be C or G.

Applications

Although the above is mostly a toy example, models based on these principles are used in sequence analysis in non-MCMC applications. Statistical comparisons of sequence elements can be made where the nucleotide composition of different types of sequence elements is distinguishable. For example, one could obtain initial distributions of nucleotides in gene coding and non-coding regions and use Markov chains to determine the stationary distributions for both. You could then perform statistical analysis to determine if there is a significant difference in nucleotide composition in coding or non-coding regions. Likewise, Markov models are frequently used to determine if a region is a CpG island using a more complicated Markov model called a Hidden Markov Model (HMM).

We have seen that the existence of a stationary state is NOT guaranteed, but is conditional on various properties of the Markov Chain. To have a unique stationary state a chain must be ergodic, possessing the characteristics of aperiodicity and irreducibility described earlier.

The convergence of Markov chains is a mathematical topic in and of itself, the details of which are beyond our discussion. However, not all chains converge with equal speed or fluidity and we do want do make sure our chain has converged before determining a stationary distribution. The package CODA will be introduced in chapter 12 and contains functionality to perform some convergence diagnostics.

Often our stationary distribution is a multivariable, high-dimensional distribution. This is difficult to analyze graphically or to interpret, so ordinarily we will be interested in analyzing individual variables and parameters separately. In order to do this we will look at marginal and conditional probability distributions for the parameters of interest. Recall the discussion and graphical illustrations in Chapter 8, which looked at these distributions for the multivariate normal, multinomial, and Dirichlet distributions. We will come back to this type of analysis.

Continuous State Space

So far we have considered discrete state spaces only – those where the states are distinct such as {A, T, C, G}. However, often we will be dealing with continuous state space models where the state space can take on a continuum of values. In the case of a continuous state space the transition matrix is replaced with a transition density often referred to as the transition kernel. This cannot be put into matrix form but is instead a joint continuous probability density. Transition probabilities are calculated with integrals, not sums. Except for the mathematical differences of dealing with continuous versus discrete values for the state space, the discrete and continuous state spaces are conceptually the same and there is no need to discuss continuous state space models in detail. We will work with continuous state space models in some of the examples used in Chapters 11 and 12

Non-MCMC Applications of Markov Chains in Bioinformatics

In this book, our primary interest is in working with probability models and using Markov Chains for modeling so that we may utilize MCMC methods to simulate posterior distributions in order to harvest results of interest. This is the

and range B, then f has an inverse function f-1 with domain B and range A and is defined by

$$f^{-1}(y) = x \Leftrightarrow f(x) = y$$

for any y in B, as depicted in Figure 11-1.

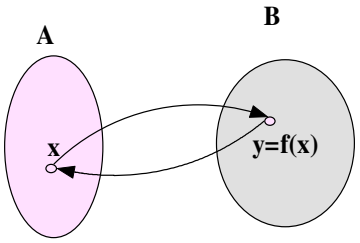


Figure 11-1

Of course, for a probability distribution function we know that the range of values in B is between 0 and 1 (obeying the probability theory that all probability values are between 0 and 1). Therefore to simulate values, we can randomly generate values between 0 and 1, which is called the uniform distribution (which is the probability distribution having equal probability of all values between 0 and 1). In R the function runif() can be used to draw a random uniform sample and then to transform the sample by the inverse CDF method to obtain a simulation of the desired distribution.

Let's do an example of the inverse CDF method to simulate an exponential distribution with parameter lambda =2. Recall that for the exponential, the CDF is

$$F(x)=1-e^{-\lambda x}$$

Letting F(x)=u

$$u=1-e^{-\lambda x}$$

Solving for x

$$1-u= e^{-\lambda x}$$

$$\log(1-u)=-\lambda x$$

$$x=-\frac{1}{\lambda}\log(1-u)$$

and Y. Values of rho between 0 and 1 indicate the degree of a linear relationship between the variables.

If the correlation coefficient is considered, the joint probability distribution of two normally distributed random variables can be written as:

$$(X,Y) \sim N\left(\begin{pmatrix}0\\0\end{pmatrix},\begin{pmatrix}1 & \rho\\ \rho & 1\end{pmatrix}\right)$$

This means X and Y are distributed normally with means 0 and 0 (the first parameter matrix) and variances of σ^2 and correlation of ρ between XY.

For reasons that we will not detail (see a textbook on multivariate statistics, such as Johnson&Wichern, listed in the appendix for details), the conditional distributions of X and Y for the bivariate normal are (here we assume $\sigma^2 = 1$ for simplicity)

$$P(X|Y=y) \sim N(\rho y, 1-\rho^2)$$

$$P(Y|X=x) \sim N(\rho x, 1-\rho^2)$$

We can write a function in R that uses the Gibbs sampler to simulate a biviariate normal distribution by iteratively sampling from these conditionals:

```
> gibbsBVN_function(x,y, n, rho){
+ #create a matrix to store values
+ m<-matrix(nrow=n,ncol=2)
+
+ #store initial values in matrix
+ m[1,]<-c(x,y)
+
+ #sampling iteration loop
+ for (i in 2:n){
+ #rnorm takes sd not variance
+ #update x conditional on y
+ x<-rnorm(1,rho*y,sqrt(1-rho^2))
+ #update y conditional on x from above
+ y<-rnorm(1,rho*x,sqrt(1-rho^2))
+
+ #store values in matrix
+ m[i,]<-c(x,y)
+ }
+ m
+ }
```

This works because it is a Markov chain. Refer back to figure 11-5. If $X^{(0)}=x0$ then the distribution of $X^{(n)}$ is $N(\rho^{2n}x0, 1-\rho^{4n})$. But as n goes to infinity this converges to N(0,1), a regular standard normal distribution. Therefore after enough runs, no matter where we start X and Y the marginal distributions of X

Therefore, overall the proportion of accepted move values will create a density of interest by accepting values most often when they are values of the most dense areas and by rejecting values mostly when they are in area of low density.

The distributions we are interested in are more complex and ratios more complex as well, but the general idea conveyed with the hill climber story is the basis for how both the Metropolis and Metropolis-Hastings algorithms work. We discuss these in more depth below. It is also of interest to note that many related algorithms exist as well and developing more specific versions of these algorithms is an active area of statistical research.

Metropolis Algorithm

The Metropolis algorithm is the simplified version of the full Metropolis-Hastings algorithm that works when the proposal distribution is symmetric around the old value θ. , as in Figure 11-12. Note that the proposal distribution is actually a conditional distribution for the new value given the old value of the parameter. In a symmetric distribution the ratio going up or down the hill doesn't matter which side of the hill you are on, whereas in a non-symmetric distribution the R values will be different for different sides of the hill. The uniform and normal distributions are common symmetric distributions (good for sampling distributions).

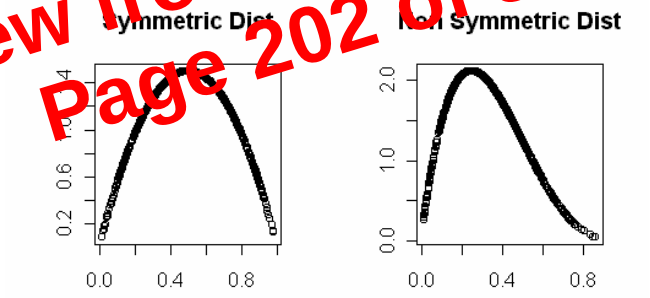


Figure 11-12

Because of the proposal distribution symmetry, with the Metropolis algorithm, the acceptance ratio R depends only on the value of the ratio of target distribution values. Note that in the hill-climbing example, this is what we did.

To make the general algorithm given earlier specific for the Metropolis algorithm

1. Generate a new value from the proposal distribution; call it thetaStar (θ*)

Table 12-1

Genotype Frequencies	Genotype	Phenotype (Blood Type)
p^2	AA	A
2pr	AO	
q^2	BB	B
2qr	BO	
2pq	AB	AB
r^2	OO	O

The easiest data to collect on blood types for a population under study is the phenotype data. It is simple to phenotype blood and collect data on numbers of individuals with each blood type. However, it is not technically or mathematically easy to determine frequencies of the individual alleles. Although for this example, it is algebraically possible, if there were more alleles involved it would soon become impossible algebraically to solve the equations necessary. Therefore a Bayesian MCMC method works better to solve this type of problem.

Recall the Bayesian paradigm

$$P(\text{Model}|\text{data}) \propto P(\text{Model})\, P(\text{data}|\text{Model})$$

$$\text{Posterior} \propto \text{Prior} * \text{Likelihood}$$

Our model of interest here is the posterior distribution of alleles (p, q, r) given the data of counts of blood type phenotypes. Recall that the discussion in Chapter 8 of using the multinomial distribution to model the distribution of phenotype data, based on phenotype counts. For this example the multinomial is the likelihood, or data, model, and can be can be written as follows:

$$P(A, B, AB, O) = \frac{n!}{n_A!\, n_B!\, n_{AB}!\, n_O!} (p_A)^{nA} (p_B)^{nB} (p_{AB})^{nAB} (p_O)^{nO}$$

Where the n's are the numbers of each phenotype and the p's are the proportions of each phenotype. Using Hardy-Weinberg equilibrium we can convert the proportions of phenotypes to allele proportions in terms of p,q, and r. Since the constant term is left out in Bayesian calculations we can re-write the likelihood model as:

$$P(A, B, AB, O) \alpha (p^2 + 2pr)^{nA} (q^2 + 2qr)^{nB} (2pq)^{nAB} (r^2)^{nO}$$

Our prior distribution of interest is the distribution of individual alleles A, B, O that are modeled respectively with p, q, and r. Recall (Chapter 8) that the Dirichlet model is used to model multivariable proportion data. For this example we have no specific knowledge of the proportions so we will use a noninformative Dirichlet prior assuming all proportions are equal (the equivalent of a beta (1,1) distribution for all priors). We can write our Dirichlet prior with parameter alpha=1 and ignoring the constant term as:

$$P(p, q, r) \quad \alpha \quad (p)^{\alpha-1}(q)^{\alpha-1}(r)^{\alpha-1} = 1$$

Gibbs Sampling to Determine Posterior

The posterior for our model is: the product of the prior and the likelihood:

$$P(p,q,r|data) \alpha \quad (p)^{\alpha-1}(q)^{\alpha-1}(r)^{\alpha-1}(p^2 + 2pr)^{nA}(q^2 + 2qr)^{nB}(2pq)^{nAB}(r^2)^{nO}$$

However, it is not easy to solve this analytically for the posterior parameters p, q, r which are the proportions of alleles in the population given the phenotype count data. We note that the posterior is of the form of a high-order polynomial in p,q,r which is actually a complicated mixture of several Dirichlet distributions.

Our method of solving this problem is to use the Gibb's sampler and an MCMC simulation iterating through the full conditionals as follows:

$$p_i = p(p \mid data, q_{i-1}, r_{i-1})$$
$$q_i = p(q \mid data, p_i, r_{i-1})$$
$$r_i = p(r \mid data, p_i, q_i)$$

Each step of the iteration has the Markov property of being dependent only on the prior step. The chain iterates like this updating each individual parameter for that step by sampling from the full conditional distribution. The number of i's is the number of cycles the chain runs. If the chain is ergodic the chain will reach a steady state distribution that is our posterior distribution of interest, the posterior distribution of the alleles given the phenotype count data.

Check the Model

After you enter the model, the first thing you want to do is check the model and make sure it is syntactically correct. To do this load the BRugs package and type the following:

```
> modelCheck("bt.txt")
model is syntactically correct
```

Load the Data

Once the model is syntactically correct, you want to load the data. Note that the data are given as a list. In this example, the data used are just made up and do not reflect empirical results.

```
> modelData("btData.txt")
data loaded
```

Compile the Model

The next step once the data are loaded is to compile the model.

```
> modelCompile(numChains=2)
model compiled
```

Initialize Values

With a successfully compiled model, you are now ready to initialize values of parameters in preparation for running the sampler. To initialize parameter values make a text file in the R working directory "btInits.txt" containing the following:

```
#inits
list(a=1,b=1,o=1)
```

Do the initializing in R with function modelInits

```
> modelInits("btInits.txt")
```

Run the Sampler

Now that we have a model, which has data loaded, is correctly compiled, and has prior parameters initialized, we are ready to run some samplers. The function samplesSet tells which parameters should be monitored and the modelUpdate function runs the sampler the specified number of times:

```
> samplesSet(c("p","q","r"))
monitor set for variable 'p'
monitor set for variable 'q'
monitor set for variable 'r'
>
>
```

```
> modelUpdate(1000)
1000 updates took 0 s
```

Analyzing Results

Once you produce sampled data, you have many options for analyzing the results. Remember that our result of interest is the Dirichlet posterior joint distribution of p, q, and r – the allele proportions for the blood type alleles given the data.

One of the simplest things to do is look at the time series plots for the chain for each of the parameters. To produce such a plot, simply use function samplesHistory() with the parameters of interest. For the code illustrated above, a time series plot of parameter r is illustrated in Figure 12-1.

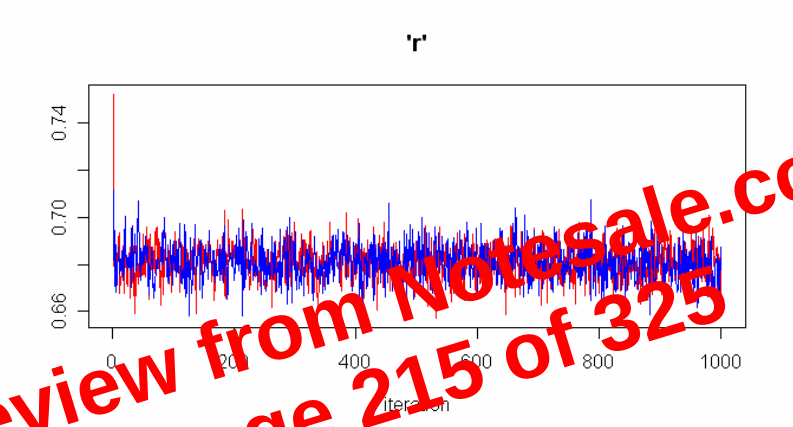


Figure 12-1: Time Series of Parameter r

A time series trace gives quick visuals check for two things – how well the chain mixes and whether the chain has converged. In Figure 12-1 the chains are well mixing (even up and down moves without a pattern of being hung up in one area or having correlated moves) and appear to have quickly converged. Note that a time series trace is NOT a formal statistical analysis, but a visual check, and although quite good at diagnosing good and bad runs and convergence, should not be used as the sole diagnostic criteria.

Another way to look at the parameters is to view a density plot of the marginal distribution of the parameter of interest.

```
samplesDensity("r", mfrow=c(1,1))
```

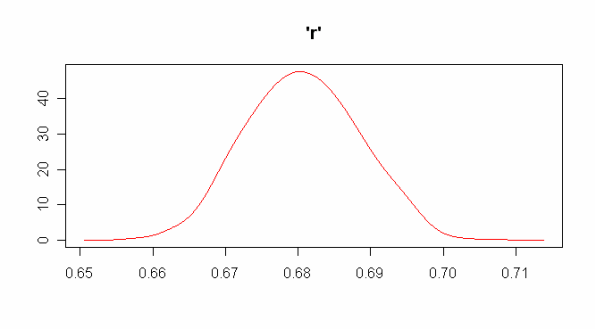


Figure 12-2: Marginal Density of Parameter r

Another way to analyze the results is to look at the summary statistics for each parameter. These results can also be used in statistical inference testing comparing parameters from different models, etc. In later chapters when we cover inferential statistics and introduce some different testing methods. Function samplesStats will give this information for all parameters of interest:

```
> samplesStats("*")
     mean       sd  MC_error val2.5pc  median val97.5pc  start sample
p 0.23410 0.007205 0.0002394  0.22100 0.23420   0.24860    502    998
q 0.08527 0.004643 0.0001428  0.07663 0.08500   0.09471    502    998
r 0.68070 0.007752 0.0002514  0.66440 0.68050   0.69570    502    998
```


CODA

The most important diagnostic to do with MCMC output however is to make sure the chains have really converged. All of the results discussed so far are invalid if the sampler has not produced a chain that has converged to a stable posterior state. CODA, the R package that is used in collaboration with BRugs will utilize various convergence diagnostics techniques. Interested users should install and explore the functionality of this package.

in great depth in most elementary statistics courses and books. But let's review some of them here.

A sample mean is simply the average of the data. To get the sample mean add all the values of data up and divide by the sample size (n). R will calculate a sample mean using `mean(x)` where x is the data vector. The sample mean is the most common measure of central tendency or location. If you take repeated data samples from the same underlying population and calculate sample means, the distribution of the sample means will follow the central limit theorem, commonly known as the law of averages. That is, the distribution will approximate a normal distribution in the limiting case (n goes to infinity).

The law of averages holds for any underlying population distribution, even though the data themselves may be far from normally distributed. Let's use R to draw samples from an exponential distribution with scale parameter 4, or rate parameter ¼ = 0.25 (rate=lambda in R) (see Chapter 7) and calculate means of each sample. To illustrate this effect we will vary the sample sizes.

```
> #First a graph of the distribution from which to sample from
> e<-seq(.1,30,by=.1)
> plot(e,dexp(e))
```

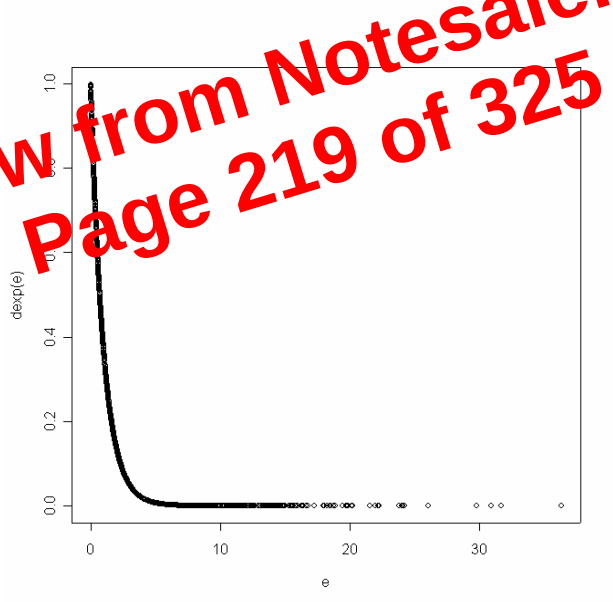



Figure 13-2: Exponential with lambda=4.

Next we will take 50 random samples of sample size n=5, n=20, n=50 and n=200 from the exponential model, calculate the mean of each of these and do a histogram of the 50 calculated means for each sample size. Note that because generated data are all i.i.d. (independent and identically distributed), we can simply draw a total of n*50 samples, and arrange them in a matrix with 50 rows, all in one command.

Sampling Distributions

Sampling distributions are a special class of probability distributions where the shape of the curve changes based on the sample size, n. The criteria "degrees of freedom" which is based in part on sample size, is part of the defining parameter for plotting a sampling distribution. Sampling distributions are distributions of statistics rather than distributions of individual data values. Every statistic has it's own sampling distribution – mean, mode, median, etc. Here we consider the sampling distribution for the mean (the t-distribution) and the sampling distributions of statistics that are based on variance (the Chi Square, and the F distributions). These common distributions serve as the basis for many statistical inference tasks.

Student's *t* Distribution

This distribution describes the sampling distribution of the sample mean when the true population variance is unknown, as is usually the case with sampling. This distribution is the basis for t-testing, comparing means drawn from different samples, and we use it in hypothesis testing. Meanwhile let's look at the mathematical properties of this distribution and how to use this distribution in R.

Recall from Chapter 7 the discussion of the normal distribution and that a random variable (X) following the normal distribution may be transformed to a Z score with the following relationship where the distribution of Z becomes the standard normal distribution with mean of 0 and standard deviation of 1.

$$Z = \frac{X - \mu}{\sigma}$$

We have already illustrated above that when we are sampling the standard deviation (true σ) is not known but an estimated standard deviation from the data is used. This estimated standard deviation is a random variable based on sample size. A t-distribution is a modification of the standard normal distribution to account for the variability of the standard deviation. A standardized t-score takes the following form:

$$t = \frac{X - \mu}{s}$$

If the data values x1,…,xn follow a normal distribution, then we call the distribution of the corresponding t scores a t-distribution with n-1 degrees of freedom. It is modeled using a t density curve. The t distribution also applies to the sampling distribution of sample means as follows:

The resulting plots are shown in Figure 13-5. In each case the solid line is the normal distribution and the dashed line is the t-distribution. Notice how when the degrees of freedom are increased the t-distribution becomes closer and closer to the normal. Indeed when sample size is roughly 30 or so (a specific number is subject to debate) we often use the normal distribution instead of the t-distribution because of this close approximation. With relatively small degrees of freedom the t-distribution has what are referred to as "heavy tails" or "thicker tails". It is important to review as well that as a probability distribution the area under the curve for any t-distribution is always 1.

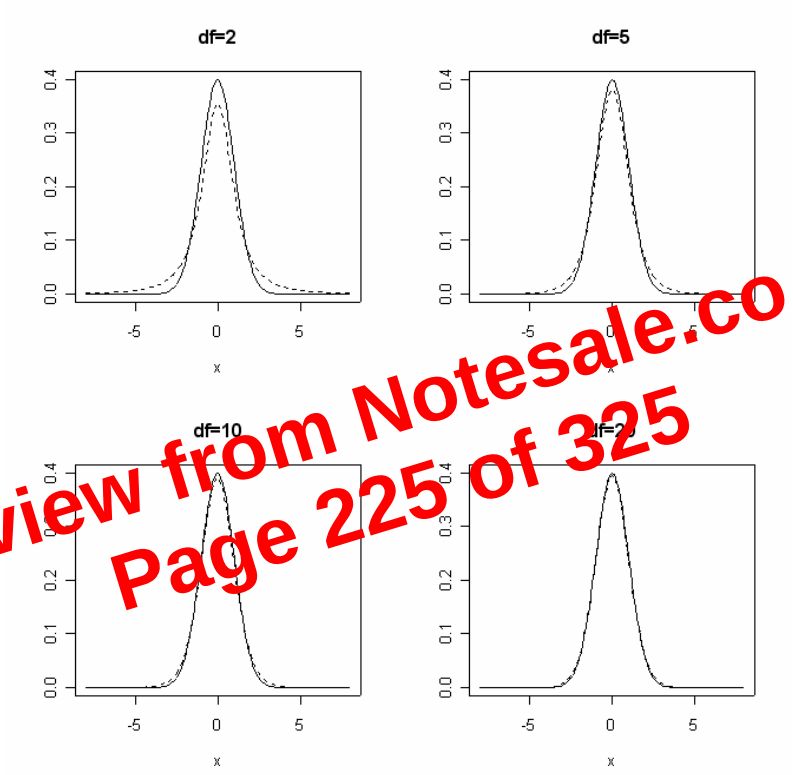


Figure 13-5

The Chi-Square Distribution

The Chi-Square distribution was briefly introduced in Chapter 7 as part of the gamma family of probability distributions. The Chi-Square distribution indirectly models the sample variance. The ratio of the sample variance to the true population variance is modeled as a Chi-Square according to the following:

$$\frac{(n-1)s^2}{\sigma^2} \sim \chi^2_{(n-1)}$$

Table 14-1

Protein Name	Primary sequence (length)	Alpha helical regions (amino acids in alpha helical form)	Beta sheet regions
Deoxy Human Hemoglobin (Chain 1A3N:A)	VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTT KTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPN ALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEF TPAVHASLDKFLASVSTVLTSKYR (141)	4-17, 21-35, 53-71, 76-79. 81-89, 96-112, 119-136 (68.07%)	none
Rab5C (mouse)	ICQFKLVLLGESAVGKSSLVLRFVKGQFHEYQESTIGAA FLTQTVCLDDTTVKFEIWDTAGQERYHSLAPMYYRGAQA AIVVYDITNTDTFARAKNWVKELQRQASPNIVIALAGNK ADLASKRAVEFQEAQAYADDNSLLFMETSAKTAMNVNEI FMAIAKKL (164)	16-25, 69-73, 88-104, 128-137, 153-162 (31.71%)	2-9, 38-47, 50-59, 78-84, 110-116, 141-144 (28.66%)
Ubiquitin	MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPD QQRLIFAGKQLEDGRTLSDYN IQKESTLHLVLRLRGG (76)	23-34(15.79%)	2-7, 12-16, 41-45, 48-49, 66-71(30.26%)
Prealbunin (Human Transthyretin, Chain A 1BMZ:A)	GPTGTGESKCPLMVKVLDAVRGSPAINVAVHVFRKAADD TWEPFASGKTSESGELHGLTTEEEFVEGIYKVEIDTKS WKALGISPFHEHAEVVFTANDSGPRRYTIAALLSPYSY TTTAVVTNPKE (127)	75-81 (5.51%)	12-18, 23-24, 29-35, 41-43, 45-48, 54-55, 67-73, 91-97,104-112, 115-123 (44.88%)

Table 14-2

Method	URL	Description
Chou-Fasman	http://fasta.bioch.virginia.edu-/o_fasta/chofas.htm	Statistical method which is based on individual amino acid "propensities" to form structure
GORIV	http://npsa-pbil.ibcp.fr/cgi-bin/npsa_automat.pl?page=/NPSA/npsa_gor4.html	Statistical method which takes into consideration local interactions ("windows") in addition to aa propensities
PHD	http://www.embl-	Neural network based; combines

Based on the graph, it looks like our value is pretty extreme and may be probable cause to doubt that the sample mean comes from the distribution of the null hypothesized mean, but let's be sure before making a decision.

We can use `1 - pt` function in R to determine the probability of being to the right of the test statistic:

```
> 1-pt(testStatistic,df=n-1)
[1
[1] 0.004413
```

This tells us that only 0.44%of the probability mass function is to the right of our test statistic. Given a cutoff alpha value of 5% (2.5% on each tail of the distribution) our test statistic is more extreme than 2.5% so we reach the decision that we reject the null hypothesis and conclude that the true mean of our data differs significantly from 0.5 and follows a different distribution than the distribution under the null hypothesis (in other words, our sample test statistic did not come from the null distribution).

Alternatively to determine if our value is too extreme we could have computed the values of the t-distribution for the critical points of both tails of the distribution using the qt function:

```
> alpha<-0.05
> qt(alpha/2,df=n-1)
[1] -2.200985
> qt(1-alpha/2,df=n-1)
[1] 2.200985
```

This would have told us that any values below –2.200985 or above 2.200985 are more extreme given the alpha level of 0.05 and we could reject any test statistic more extreme than these values.

We call the probability of being as extreme or more extreme than the observed test statistic (2 times above for a two-tailed alternative) the p-value or "observed significance level". If the p-value is less than the alpha-level the experimenter set, then we reject the null hypothesis for a given test. The smaller the p-value, the more significant the test result. Our p-value of 2 times 0.004413 = 0.008826 is significant, but a value of 0.000088 for example would be more extreme and even more significant. The next section discusses significance and statistical decision making in more detail.

Making Statistical Decisions

When you perform a hypothesis test you are subject to some gray areas. Remember nothing in statistics is ever absolute and in any type of statistical analysis there is always the randomness factor. Hypothesis testing is a process of making statistical decisions, and in hypothesis testing there is always some margin of error where it is possible that the test result is not correct.

introductory statistics books. But there are a few key things to be mentioned with regard to power and using R to compute power.

First, an important thing to know about power is that as the sample size increases, the power of a test increases. The second thing to know is that decreasing the alpha level (significance level) decreases the power of a test, given the same sample size. That is a test with an alpha level of 0.01 has less power than a test with an alpha level of 0.05. The bottom line is that both the alpha level and the sample size play a role in how powerful a test is and how well a test will correctly reject a null hypothesis. Do not confuse power with significance levels!

For some tests, R provides some relief in computing power. Two functions, `power.t.test` and `power.prop.test`, built into the ctest package automate power computations that have flexible parameters, allowing the user to enter the criteria they wish in order to have R compute other criteria.

For example, `power.t.test` is specific for computing power related values for the t-test (and has optional parameters for different versions of the t-test, one-sided versus two sided, etc). For example, suppose we want to know the power of a test given a sample of size n=20 at a significance level of alpha=0.05. We can specify these parameters, and also a value for delta, which is the difference between populations that we would like to be able to detect. For example, suppose we want to detect a difference of 0.5 units between our null and our alternative hypothesis, for this we use delta=0.5. It doesn't matter what the measurement units are, but a default standard deviation of 1 is assumed (which can be changed if necessary using the "sd" option). For the type of test we need to specify "one.sample". Later on we will discuss other types of t tests, such as two sample and paired for which power can also be calculated with this command.


```
> power.t.test(n=20,delta=0.5,sig.level=0.05, type="one.sample")

     One-sample t test power calculation

              n = 20
          delta = 0.5
             sd = 1
      sig.level = 0.05
          power = 0.5645
    alternative = two.sided
```

Note the power for the test above is only 0.5645, which is not very high. Perhaps if we look for a less subtle difference, say delta=1, we should get a higher power test, let's see:

```
> power.t.test(n=20,delta=1,sig.level=0.05, type="one.sample")

     One-sample t test power calculation

              n = 20
          delta = 1
```

The test result with a p-value of 0.03583 rejects the null hypothesis at a critical value (alpha level) of 0.05, but would accept the null hypothesis at a lower critical value such as 0.01. Therefore for this test the interpretation is dependent on the critical value set by the experimenter. For convenience `t.test` also computes a 95% confidence interval for the mean expression level (see the description in Chapter 13). As is expected the hypothesized value of 2000 lies outside the 95% confidence interval, since we rejected that value at a significance level of 5%. There is an exact correspondence between two-sided hypothesis tests and (1-α)100%confidence intervals.

Two sample t-test

The two-sample t-test is used to directly compare the mean of two groups (X and Y). It is required that measurements in the two groups are statistically independent. The null hypothesis states that the means of two groups are equal, or equivalently, the difference of the means is zero:

$$\text{Ho: } \mu(X)=\mu(Y), \text{ or } \mu(X) - \mu(Y) = 0$$

The test statistic for the two-sample t-test used by default in R (for Welch's test) is:

$$t = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{s_X^2}{m} + \frac{s_Y^2}{n}}}$$

where s^2_X is the sample variance of the X group, s^2_Y is the sample variance of the Y group, and m and n are the sizes of groups X and Y.

A good two sample t-test for our gene expression data is that there is no difference overall between the treatments and controls for any of the genes (test that the whole experiment didn't work or there are no differentially expressed genes). This is very simple in R by just entering the two vectors whose means are being compared as parameters to function `t.test`:

```
> treatments<-c(geHTdata$t1, geHTdata$t2, geHTdata$t3, geHTdata$t4)
> t.test(controls,treatments)

        Welch Two Sample t-test

data:  controls and treatments
t = -3.6163, df = 70.732, p-value = 0.0005564
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -653.6098 -188.9902
sample estimates:
mean of x mean of y
 1852.375  2273.675
```

The p-value for this test very strongly rejects the null hypothesis of no difference between the mean of the treatment group and control group, indicating there are some genes which exhibit significantly different gene expression levels, although the test does not provide specifics as to which genes these are.

Paired t-test

We have so far considered t-test testing one mean against a hypothesized mean (one-sample t-test) and comparing two statistically independent group means (two sample t-test). The final variant of the t-test is called the paired t-test. This test is used with paired data to determine if the difference in the data pairs is significantly different. The test statistic here is:

$$t = \frac{\bar{d}}{s_d / \sqrt{n}}$$

Where $\bar{d}$ is the average difference between the pairs, n is the number of pairs and s_d is the sample variance of the paired differences. We notice that the paired t-test is really the one-sample t-test applied to the differences of measurements within the pairs. The example of paired data we will use here is the difference between treatment and control on the states of the same gene. For example control 1minus treatment 1 for gene 4 for the same experiment can be considered a difference of paired data. Suppose gene 4 and gene 9 are really the same gene. If we can pool the data for these two genes:

```
> g4g9ctrl
[1] 1545 1265 1440 1399 1530 1660 1501 1478
> g4g9trt
[1] 1910 2028 1901 2002 2329 2332 2298 2287
```

Each of the 8 data values in the two data vectors with corresponding vector indices created above is a data pair. We can test for a significant difference in treatment-control for this gene using a paired t-test. Note that in parameters we indicate "paired=TRUE" to perform the test.

```
> t.test(g4g9ctrl,g4g9trt,paired=TRUE)

        Paired t-test

data:  g4g9ctrl and g4g9trt
t = -9.0459, df = 7, p-value = 4.127e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -768.3529 -449.8971
sample estimates:
mean of the differences
                -609.125
```

Comparing Variances

Sometimes we will be interested in testing whether two groups have equal variances, as this is often an assumption when performing the t-test and other statistical tests. If the different groups have significantly different variation (spread of the data) it can impact the validity of the test result. Let's do a simple test to determine if the variances for the gene expression data for the gene 4/gene 9 data (same gene) are the same under treatment or control conditions. To do so is very simple in R using the `var.test` and the desired data vectors as parameters:

```
> var.test(g4g9ctrl,g4g9trt)

        F test to compare two variances

data:  g4g9ctrl and g4g9trt
F = 0.2271, num df = 7, denom df = 7, p-value = 0.06907
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.045468 1.134387
sample estimates:
ratio of variances
         0.2271085
```

This test uses an F-test comparing the ratio of the variances of the two groups to a critical value on the F-distribution (discussed in Chapter 3 as the sampling distribution for the ratio of variances). This distribution will be used extensively in ANOVA discussed in the next chapter. The p-value result above indicates it is OK to assume at an alpha=0.05 ratio that there is no significant difference in variances between the two groups (and hence, t-tests can be assumed reliable since they assume equal variances). Note this holds only if you assume equal variances for the pooled variance version. This is not assumed in the Welch version (see above)

Indeed if we look at the variances of the two data groups and calculate the ratio of the variances, we get the same result as above:

```
> var(g4g9ctrl)
[1] 8455.929
> var(g4g9trt)
[1] 37232.98
> 8455.929/37232.98
[1] 0.2271086
```

Select Nonparametric Tests

Nonparametric tests make no or very minimal assumptions about the probability density from which the data are derived. They are used when the sample size is small, when the data are not normally distributed (always test data with a Q-Q plot described in chapter 7, if the normal assumption is in question) and cannot

When testing a contingency table we assume under the null hypothesis that the two factors are independent. Therefore under the null hypothesis the expected probability of an observation falling into category oij (i=roe, j=column) is just the product of the marginal probabilities for that row and column. For example, the probability of being categorized into the first row and first column is:

$$p_{11} = \frac{R_1}{N} * \frac{C_1}{N}$$

Note that the values can also be expressed in terms of marginal probabilities (see Chapter 8 for a discussion of marginal probability) for each row or column. R1/N for example, is the marginal probability of being in row 1.

The alternative hypothesis in contingency analysis is to disprove the null hypothesis by showing that there is a relationship between the factors being analyzed and that they are not independent of each other. To do this we will use the Chi-Square test and Fisher's exact test, both of which are available as predefined R functions available as part of package ctest.

Let's define a contingency table dataset to use. Suppose we are doing a genetic study and studying the effect on which of two alleles for a gene a person has and the presence of a disease. We perform a genetic test to determine which allele the test subjects have and a disease test to determine whether the person has a disease. The data for a 2 x 2 contingency analysis should be entered in the format below, which works for both tests.

```
> contingencyTestData<-
+ matrix(c(45,67,122,38),
+ nr=2,
+ dimnames=list("Gene"=c("Allele 1","Allele 2"),
+ "Disease"=c("Yes","No")))

> contingencyTestData
          Disease
Gene       Yes  No
  Allele 1  45 122
  Allele 2  67  38
```

The tests we want to perform with this contingency table are whether or not the two factors, disease and gene allele, are independent or whether there is a significant relationship between the factors. The null hypothesis is that there is no relation and the factors are independent.

15

ANOVA and Regression

A lot of this book has been concerned with probability models, which are used to model the probability of a range of data. Now we talk about another type of models, linear modeling. Linear models can be used to predict variables and to evaluate patterns between variables. The most common types of linear models are linear regression and ANOVA models, which will be introduced here. Using R makes working with linear models very easy as R has extensive built-in functionality to handle such models. Let's first look at ANOVA, then linear regression, and then the end of the chapter discusses general linear models.

ANOVA

The previous chapter discussed techniques for comparing means of two groups of data using the two-sample t-test (illustrated comparing control and treatment means for gene expression data). This test, however, is limited because in many cases we wish to compare more than two group means. ANOVA, or analysis of variance (somewhat misnamed, but we shall soon see why) is the statistical method used to determine differences in means when there are more than two groups under study.

Let's return to the dataset illustrated in Table 14-3, made into a data frame in R (available as protStruct). Note that the non-numerical data are factor data types. Factors are non-numerical variables. Different values of the factor are called levels. For example, Method has 3 levels: CF AVG, GOR, and PHD, representing the three different methods used to evaluate protein secondary structure (see discussion in Chapter 14). ANOVA always uses some type of factor variable. The function `as.factor` can be used to convert data to data type factor.

```
 > protStruct
      Protein Method Correct
1   Ubiquitin CF AVG   0.467
2   Ubiquitin    GOR   0.645
3   Ubiquitin    PHD   0.868
4     DeoxyHb CF AVG   0.472
5     DeoxyHb    GOR   0.844
6     DeoxyHb    PHD   0.879
7       Rab5c CF AVG   0.405
8       Rab5c    GOR   0.604
9       Rab5c    PHD   0.787
10 Prealbumin CF AVG   0.449
11 Prealbumin    GOR   0.772
12 Prealbumin    PHD   0.780

> # NOTE - Protein and Method are FACTOR data types
> is.factor(Protein)
[1] TRUE
```

Suppose we want to test whether the percentages correct differ by method (ignoring the protein factor for now). Essentially what we want to test is whether the mean percent correct is different based on method. Because we have three groups, a two-sample t test is inadequate for this analysis.

Let's extract this data from the data frame (not a necessary step, just to simplify things here)

```
> compareMethod
   Method Correct
1  CF AVG   0.467
2     GOR   0.645
3     PHD   0.868
4  CF AVG   0.472
5     GOR   0.844
6     PHD   0.879
7  CF AVG   0.405
8     GOR   0.604
9     PHD   0.787
10 CF AVG   0.449
11    GOR   0.772
12    PHD   0.780
```

The trick with ANOVA (and why it is called analysis of variance) is to look at the response of interest (percent correct in this case) and analyze the variability in that response. The way ANOVA does this is to break up the variability into two categories – variability within groups, and variability between groups.

If we look at the data and regroup it as depicted in Table 15-1 we notice that we can computer an average (sum of the data divided by the number of data values) for each of the groups as well as a "grand" average, which is the average of all the data.

Table 15-1

Method	Correct	Group Averages
CF AVG	0.467	0.448
CF AVG	0.472	
CF AVG	0.405	
CF AVG	0.449	
GOR	0.645	0.716
GOR	0.844	
GOR	0.604	
GOR	0.772	
PHD	0.868	0.828
PHD	0.879	
PHD	0.787	
PHD	0.780	
Grand Average	**0.664**	

In order to statistically determine if a significant difference exists between the methods, we need to determine how much variability in the result is due to random variation and how much variability is due to the different method. Different factor levels are sometimes referred to as "treatments". In this case the treatment is the secondary structure determination method.

ANOVA partitions the observed variability into two components. One component is random variation, also known as pure error or within factor level variation. This "within" variation is calculated by observing the variability of the replicates within each level of the experimental factor. For example, for the CF AVG method, the within variation is calculated by using the variation of each individual measure minus the group average. Changing the factor levels causes the other component of variability. This is called "between" factor variation. This type of variability is measured using group averages as compared to the grand average. The total variation is the sum of the within variation and the between variation, and is measured using each data value as compared to the grand average.

Note, we have not yet discussed how to calculate within, between and total variation. Although it seems easiest to subtract average values from data points to perform these calculations, this alone does not work. The sum of distances of data from any average of the data is always zero. Therefore, we use the sum of the squared distances as the metric of variability of data values from average values.

Figure 15-1 shows the details of these calculations for the protein structure data. Of key interest are the three columns noted. The sum of squares total is the sum of the squared distances from each data point to the grand average. The sum of

```
> data(iris)
> # first print the data
> print(iris)

    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
….
….
49           5.3         3.7          1.5         0.2     setosa
50           5.0         3.3          1.4         0.2     setosa
51           7.0         3.2          4.7         1.4 versicolor
52           6.4         3.2          4.5         1.5 versicolor
..
…
100          5.7         2.8          4.1         1.3 versicolor
101          6.3         3.3          6.0         2.5  virginica
102          5.8         2.7          5.1         1.9  virginica
..
…..
149          6.2         3.4          5.4         2.3  virginica
150          5.9         3.0          5.1         1.8  virginica

> # Create subset of Species versicolor
> iris.versicolor <- iris[iris$Species=="versicolor",1:4]
> # Plot 4 histograms
> par(mfrow=c(2,2))
> for (i in 1:4) { hist(iris.versicolor[,i],xlab=NULL,
main=names(iris.versicolor)[i]) }
```

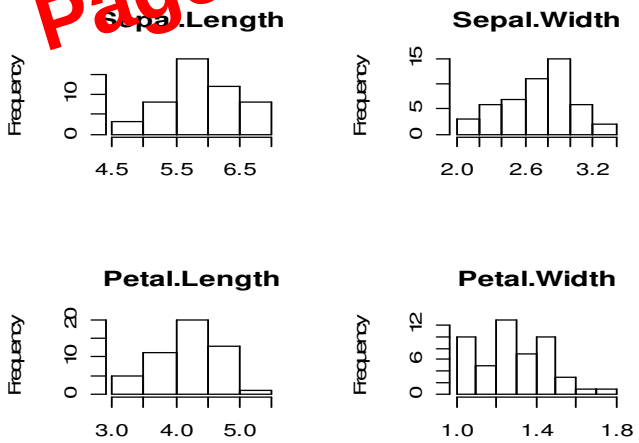


Figure 16.5: Histograms of Iris Versicolor Variables

With the exception of petal width, the variables fairly closely follow a normal distribution as we can see in Figure 16.5. Let's next examine pairwise correlations and scatterplots. R provides a convenient command, "pairs" that produces all scatterplots of all pairs of variables in a data set arranged as a

matrix (see Figure 16.6). We also create labels for plotting that indicate the iris species. There is of course redundancy in this figure. For example the first row second column entry is the scatterplot of Sepal Width on the x-axis versus Sepal Length on the y-axis, while the second row first column entry is the scatterplot of the same pair with the axes interchanged.

```
> n <- nrow(iris)
> # Create labels
> ir.labels <- rep("v",n)
> ir.labels[iris$Species=="versicolor"]<- "c"
> ir.labels[iris$Species=="setosa"]<- "s"
> pairs(iris[,1:4],pch=ir.labels)
```

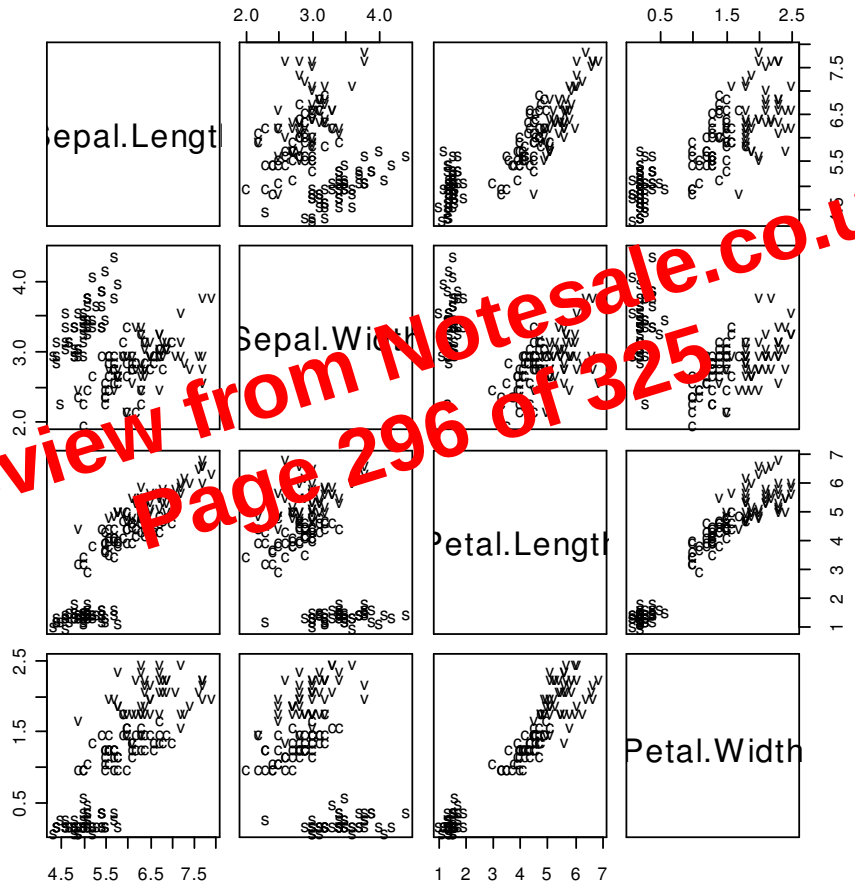


Figure 16.6: Scatterplot Matrix of Iris Versicolor Data

```
> # Pairwise Correlations
> cor(iris.versicolor)

             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.5259107    0.7540490   0.5464611
Sepal.Width     0.5259107   1.0000000    0.5605221   0.6639987
Petal.Length    0.7540490   0.5605221    1.0000000   0.7866681
```

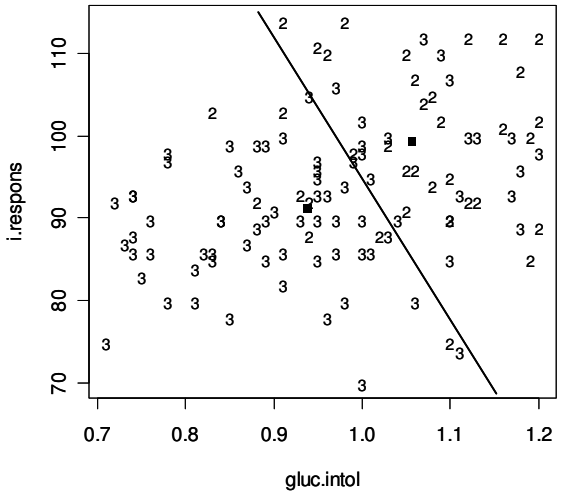


Figure 16.11: Glucose Intolerance and Insulin Response of Healthy (3) and Subclinically Diabetic (2) Individuals

Geometrically speaking a classification rule for this example is a partition of the 2-dimensional sample space where each side of the partition is assigned to one of the two groups. In the ideal case the observations are totally separated in sample space, in which case we can easily draw a line, or a curve of separation that classifies the data perfectly. In the majority of real-world scenarios there will be overlap of the groups, as illustrated by this example. Because of the overlap there is some risk of misclassifying a future sample into group "2" when it should be classified into group "3" and vice versa. The probabilities of correct classification and misclassification of an object X into groups A and B can be written as

$$P(X \text{ is correctly classified and } X \text{ in A}) = P(X \text{ in A}|A)p_A$$

$$P(X \text{ is misclassified as A and } X \text{ in B}) = P(X \text{ in A}|B)p_B$$

$$P(X \text{ is correctly classified and } X \text{ in B}) = P(X \text{ in B}|B)p_B$$

$$P(X \text{ is misclassified as B and } X \text{ in A}) = P(X \text{ in B}|A)p_A$$

In the above, p_A and p_B represent the prior probability of group A and group B respectively. Thus, the classification/misclassification probability is the conditional probability of the classification multiplied by the prior probability. Now we are ready to derive an optimal classification rule by minimizing the expected cost of misclassification (ECM).

The optimal classification results in two canonical variates (LD1 and LD2) given by the coefficients above. About 88% of the separation results from the first LD transform. Let's examine class membership and misclassification error.

```
> # Store the list of output
> gluc.pred <- predict(gluc.lda)
> # class membership prediction
> gluc.pred$class

  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3
 [30] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [59] 3 3 3 2 2 3 2 2 3 3 2 3 2 3 3 3 3 3 2 3 3 3 3 2 2 3 2 2 2
 [88] 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 3 1 1 1 1
[117] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 3 2 2 2 1 1 1 1 1 1 1 1
Levels: 1 2 3

> tab <- table(glucose$diagn,gluc.pred$class)
> tab/rowSums(tab)

          1       2       3
  1 0.8182 0.15152 0.03030
  2 0.0000 0.86111 0.13889
  3 0.0000 0.03947 0.96053
```


This classification has much lower misclassification error than the previous one where we only used two response variables. In particular, three out of 76 healthy individuals are misclassified as chemically diabetic, with a misclassification probability of $3/76 = 0.039$. We see in Figure 16.12 plot (b) that the separation of the three groups shows clearly when the data are represented in the two canonical variates, whereas the groups appear interspersed, particularly groups 2 and 3, when represented in the first two variables glucose intolerance, and insulin response, as shown in plot (a).

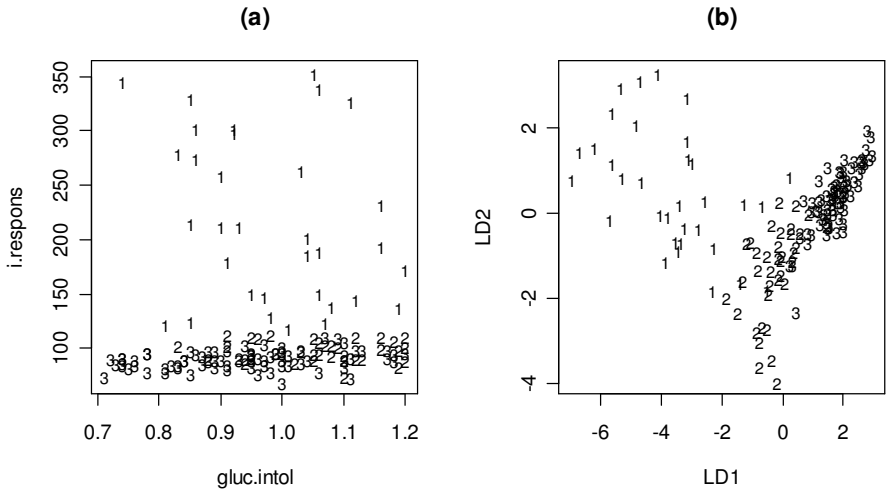


Figure 16.12: Comparison of Group Separation in (a) the first two variables and (b) the first two canonical variates

Cross-Validation

A note of caution is in order. Any of the linear methods for multivariate statistics are optimal when the data more or less follow a multivariate normal distribution. More importantly, since the calculations depend on the sample variance-covariance matrix, they can be highly sensitive to outliers. This outlier sensitivity can have a major impact on the misclassification error: Outliers can have a pull-effect on the separation curve that tends to reduce their probability of being misclassified. It is a well-known fact that, if the data from which the classification rule is derived (the "training" set) is also used to evaluate the rule, then the misclassification error are too low, or biased towards zero.

In order to adequately evaluate the classification procedure we need to have a separate data set, the so-called test set, for which we evaluate how well the classification rule works. This is called cross-validation. The test set can be a separated part of the dataset originally used which was not incorporated when making the model or a different dataset. R provides methods for cross validation which the interested reader should explore.

Classification Trees

Classification trees, or recursive binary partition schemes originated as tools for decision making in the social sciences and were introduced to main-stream statistics by Breiman et al. (1984). Classification trees essentially provide a sequence of rectangular blocks of the data space, where one of the groups is assigned to each block. The method is recursive, which means that the partitioning at a given step depends on the partitioning of previous steps, hence the method lends itself to a tree-like representation. Classification trees are similar to the widely used "keys" in botany for plant identification.

Constructing a Tree

The R library "tree" provides convenient commands that produce and validate classification trees. Let's use this methodology on the cancer microarray gene expression data, available at *http://www-stat.stanford.edu/ElemStatLearn* ,the website for the textbook by Hastie et al. (2001). This data has been fully preprocessed. A total of 64 tissue samples of a total of 14 different cancers have been obtained and their genetic responses were analyzed with DNA microarray technology.

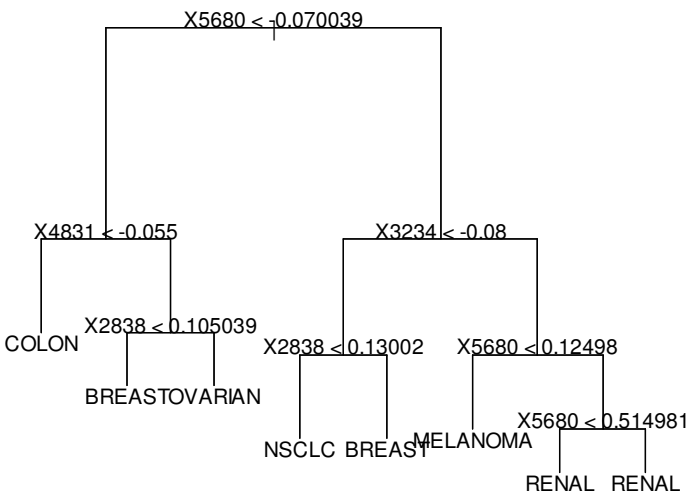


Figure 16.13 Classification Tree for Gene Expression Data

Hastie et al. (2001) delves much further into creating clasification trees. We can assess the predictive power of a tree using cross-validation, as described previously. The relevant command here is `cv.tree`. Interested users should familiarize themselves with this functionality.

Classification trees have their natural counterpart in regression analysis where the variable to be predicted is continuous. Here the method is called regression trees. The commercially available computer software CART (Classification and Regression trees) is a full-fledged stand-alone package for tree-based analyses. The R package `tree` provides functionality for both, classification and regression trees.

Clustering Methods

Clustering of data is usually a first step in organizing a multivariate dataset. Clustering is common in everyday life. Let's say you purchased a new bookshelf that holds 100 of your books. How do you arrange your books on the new shelf? You obviously want to place the books that are similar next to each other, and those that are dissimilar far from each other. In such a way you create groups, or clusters of books, where books within a cluster are similar. Now, how do you define similar? You may form groups based on qualifiers (nominal variables) like fiction, essays, or others, or educational books versus leisure reading. Or you may use quantitative characteristics such as book size, thickness, etc. In the end you are likely to have used several characteristics for determining the clusters, and you will have switched several books back and forth between different clusters (i.e. places on the shelf). The final outcome will depend on what "measures of similarity" or "dissimilarity" you have used.

G2	0.2	0.8
G3	0.3	0.4
G4	0.9	0.2
G5	-0.5	0.5
G6	0.3	-0.5

The data are plotted in Figure 16.14.

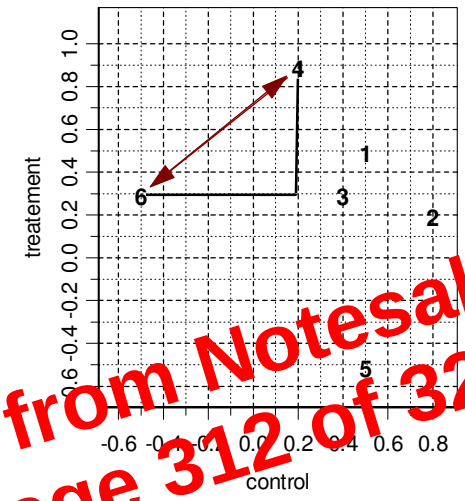


Figure 16.14 Toy Example of Gene Expression

The R package `mva` provides basic functionality for obtaining dissimilarity measures and for traditional clustering methods. For continuous data dissimilarity is measured using traditional distance metrics, the most obvious one being the Euclidean (geometric) distance. For genes **x** and **y** this is:

Euclidean Distance:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_p - y_p)^2} = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

In our case, we only have two variables, so p=2, and for genes 4 and 6 the distance is

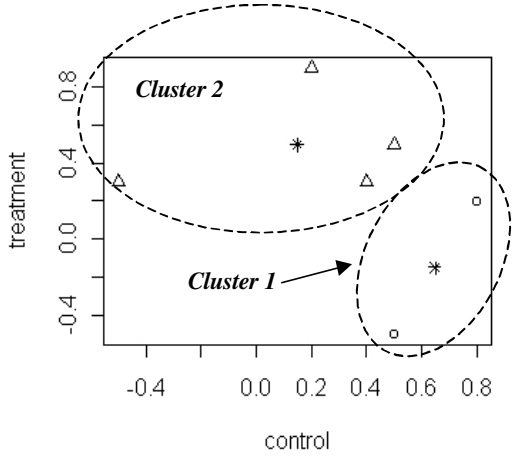
$$d(G4, G6) = \sqrt{(0.9 - 0.3)^2 + (0.2 - (-0.5))^2} = \sqrt{0.36 + 0.49} = 0.922$$

The R command `dist` creates a lower triangular matrix of pairwise distances..

- First, decide on the number k of clusters to be calculated, and then separate the data arbitrarily into k initial clusters. Calculated the centroid (=mean value or center) coordinates for every cluster selected
- Second, check through the data, assigning each data item to the cluster whose centroid is nearest (based on a distance metric). If a data item needs reassignment, create new clusters and recalculate the centroid coordinates for the clusters losing and gaining items.
- Third, repeat the second step until no more reassignment takes place.

There exist slight variations to this algorithm. In some versions the new centroid coordinates are calculated only after all data points have been checked and possibly reassigned. The R command `kmeans` is part of the package `mva`.

```
> clus <- kmeans(genes, 2, 20)
> # Note: the second number (20) denotes the number of iterations
> clus

$cluster
[1] 2 1 2 2 1 2

$centers
  [,1]  [,2]
1 0.65 -0.15
2 0.15  0.50

$withinss
[1] 0.21 0.85

$size
[1] 2 4

> plot(genes, pch = clus$cluster,xlab='control',ylab='treatment')
> # Plot Cluster Centers
> points(clus$centers,pch=8)
```

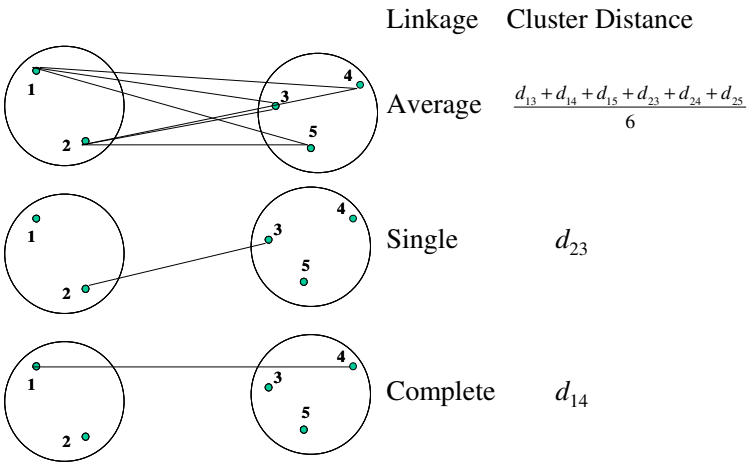


Figure 16.16: Linkage Methods for Hierarchical Clustering

We examine the different linkage methods for the sleep data. We use the R command `agnes` ("agglomerative nesting") of the `cluster` library. We note that `agnes` is virtually identical to the command `hclust` of the `mva` package but is more convenient to use when standardization of the data is necessary. The choice of measurement units strongly affects the resulting clustering. The variable with the largest variance will have the largest impact on the clustering. If all variables are considered equally important, the data need to be standardized first. The `plot` command provides two graphs, (1) banner plot, (2) dendrogram. We find the dendrogram more useful. It is obtained using the `which=2` option. Without the `which` option R plots both graphs in sequence with an interactive step in between. We display dendrograms of the clustering that results from the three linkage methods average, complete, and single. Ordinarily the rownames of the data are used as labels in the dendrogram. In order to avoid cluttering we create simple numberings for labels.

```
> # Agglomerative Hierarchical Clustering using number labels
> n <- nrow(sleep1)
> row.names(sleep1) <- 1:n
> cl1 <- agnes(sleep1[,-1],method='aver',metric='eucl',stand=T)
> cl2 <- agnes(sleep1[,-1],method='comp',metric='eucl',stand=T)
> cl3 <- agnes(sleep1[,-1],method='sing',metric='eucl',stand=T)
> plot(cl1,which=2,main="Average Linkage")
> plot(cl2,which=2,main="Complete Linkage")
> plot(cl3,which=2,main="Single Linkage")
```