Missing Data

Anyone working with empirical data sooner or later deals with a data set that has missing values. R treats missing values by using a special NA value. You should encode missing data in R as NA and convert any data imports with missing data in other forms to NA as well, assuming you are not using a numerical convention (such as entering 0's).

```
> missingData<-c(1,3,1,NA,2,1)
> missingData
[1]  1  3  1 NA  2  1
```

If computations are performed on data objects with NA values the NA value is carried through to the result.

```
> missingData2<-missingData*2
> missingData2
[1]  2  6  2 NA  4  2
```

If you have a computation problem with an element of a data object and are not sure whether that is a missing value, the function is.na can be used to determine if the element in question is a NA value.

```
> is.na(missingData[1])
[1] FALSE

> is.na(missingData[4])
[1] TRUE
```

Controlling the Workspace

This section describes some basic housekeeping tasks of listing, deleting, and editing existing objects. Then there is discussion of the different ways of saving your workspace.

Listing and Deleting Objects in Memory

When working in R and using many data objects, you may lose track of the names of the objects you have already created. Two different functions ls() and objects() have redundant functionality in R to list the current objects in current workspace memory.

```
> ls()
[1] "q"  "v1" "v2" "x1" "x2" "x3" "z"
> objects()
[1] "q"  "v1" "v2" "x1" "x2" "x3" "z"
```

Sometimes you will want to remove specific objects from the workspace. This is easily accomplished with the remove function, rm(object) with the object name as the argument.

```
> rm(q)
```

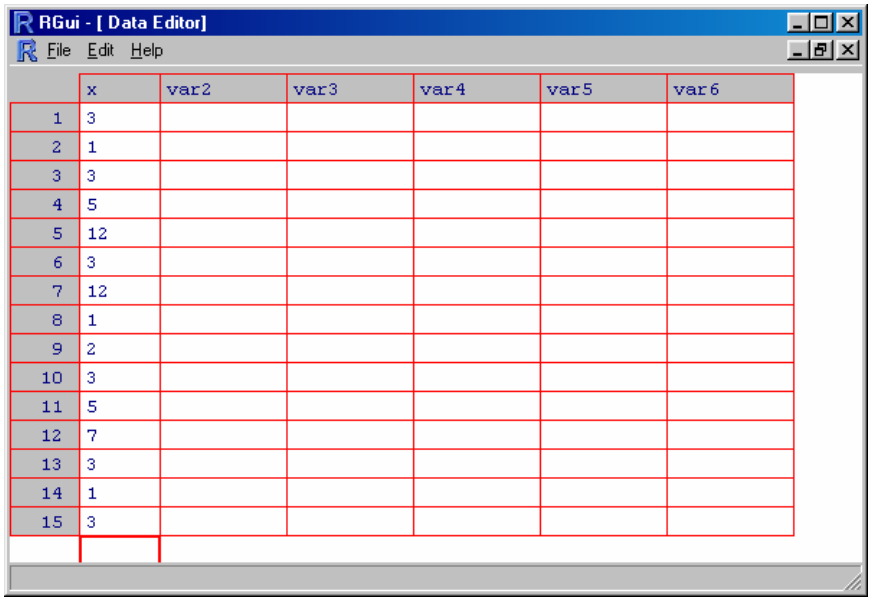
Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> ls()
[1] "v1" "v2" "x1" "x2" "x3" "z"
> rm(v1, z)
> ls()
[1] "v2" "x1" "x2" "x3"
```

Editing data objects

R has a built in data editor which you can use to edit existing data objects. This can be particularly helpful to edit imported files easily to correct entries or if you have multiple data entries to edit beyond just simple editing of a particular entry. The data editor has a spreadsheet like interface as depicted in Figure 3-1, but has no spreadsheet functionality.



	x	var2	var3	var4	var5	var6
1	3					
2	1					
3	3					
4	5					
5	12					
6	3					
7	12					
8	1					
9	2					
10	3					
11	5					
12	7					
13	3					
14	1					
15	3					

Figure 3-1

To use the data editor, use the data.entry function with the variable being edited as the argument:

```
> x<-c(3,1,3,5,12,3,12,1,2,3,5,7,3,1,3)
> data.entry(x)
```

All changes made using the data editor are automatically saved when you close the data editor. Using the Edit menu option “Data editor”, which brings up a dialog box asking which object to edit, is an alternative way to access the data editor.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Saving your work

R has a few different options for saving your work: save to file, savehistory, and save workspace. Save to file saves everything, savehistory saves commands and objects and save image just saves the objects in the workspace. Let's explain a little more about these and how to use them.

Save to file

Save to file is an option available under the file menu. This option saves everything – commands and output – to a file and is the most comprehensive method of saving your work. This option produces a save as dialog box, making saving the file in a specific directory easy. It produces a text file format viewable in a text editor or word processor or custom specified file type using the all files option and typed in file type. This method of saving is most familiar and simplest, but sometimes you may not want to save everything, particularly when you have large amounts of output and only want to save commands or objects.

Savehistory

This history of what you did in a session can be saved in a *.Rhistory file using the savehistory function. This will save everything typed into the command line prior to the savehistory() function call in the session without R formatting or specific output (versus Save to file which includes all output and formatting).

```
> x<-c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> savehistory(file="shortSession.Rhistory")
```

This creates a *.Rhistory file in the main R directory (C:\Program Files\R* where * is the current version of R) unless otherwise specified. This file should be readable by a text editor, such as notepad, as in Figure 3-2.

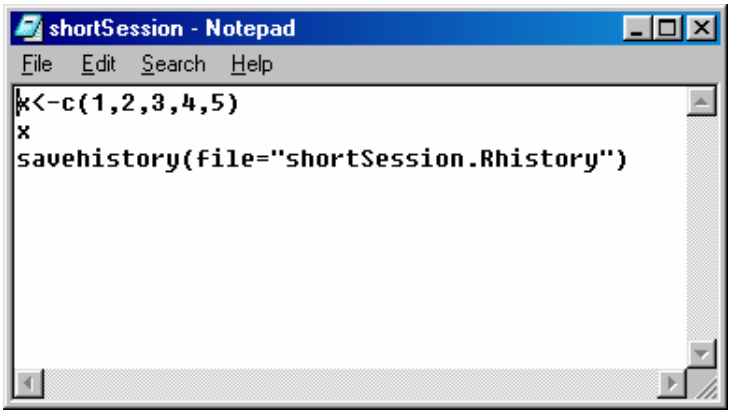


Figure 3-2

Permission granted to reproduce for nonprofit, educational use.

Saving workspace image

The option to save the workspace saves only the objects you have created, not any output you have produced using them. The option to save the workspace can be performed at any time using the save.image () command (also available as "Save Workspace" under the file menu) or at the end of a session, when R will ask you if you want to save the workspace.

```
> x<-c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> save.image()
```

This creates an R workspace file. It defaults to having no specific name (and will be overwritten the next time you save a workspace with no specific name), but you can do save.image(filename) if you want to save different workspaces and name them.

Note that R will automatically restore the latest saved workspace when you restart R with the following message

```
[Previously saved workspace restored]
```

To intentionally load a previously saved workspace use the load command (also available under the file menu as "Load Workspace").

```
> load("C:/Program Files/R/rw1062/.RData")
> x
[1] 1 2 3 4 5
```

Importing Files

We have seen that entering data can be done from within R by using the scan function, by directly entering data when creating a data object, and by using the data editor. What about when you have a data file that you want to import into R, which was made in another program? This section touches on the basics of answering these questions.

It is of note here that there is a manual available for free on the R site and on the R help menu (if manuals were installed as part of the installation) called "R Data Import/Export" which covers in detail the functionality R has to import and export data. Reading this is highly recommended to the user working extensively with importing or exporting data files. This manual covers importing data from spreadsheets, data, and networks.

The first thing to do before importing any file is to tell R what directory your file is in. Do this by going under the File menu and choosing the "Change dir" option", which produces the dialog box illustrated in Figure 3-3. Type in or browse to the directory of your data file and press the OK button.

Permission granted to reproduce for nonprofit, educational use.

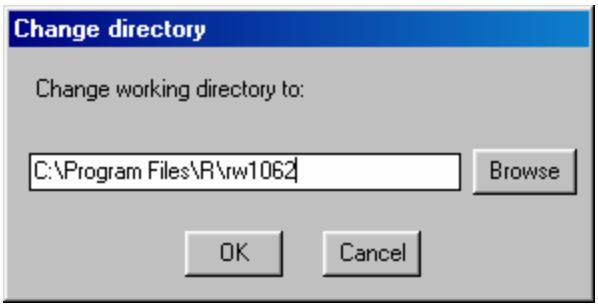


Figure 3-3

Importing using the function read. *()

The most convenient form to import data into R is to use the read functions, notably read.table(). This function will read in a flat file data file, created in ASCII text format. In Notepad you can simply save such a file as a regular text file (extension *.txt). Many spreadsheet programs can save data in this format. Figure 3-4 gives an example of what this format should look like in Notepad:

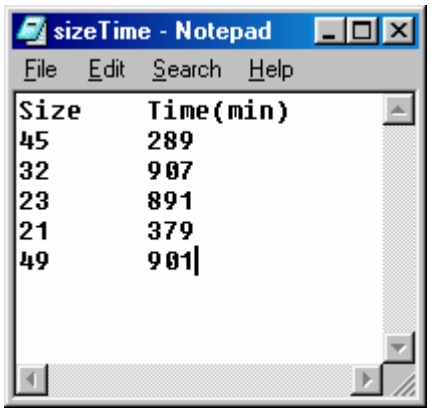


Figure 3-4

Using read.table with arguments of file name and header=T (to capture column headings), such a file can easily be read in R as a data frame object:

```
> sizeTime<-read.table("sizeTime.txt",header=T)
> sizeTime
  Size Time.min.
1   45       289
2   32       907
3   23       891
4   21       379
5   49       901
```

There are some additional read function variants. Notably read.csv() which will read comma delineated spreadsheet file data, which most spreadsheets can save files as.

Permission granted to reproduce for nonprofit, educational use.

Importing with scan ()

The scan function can be used with an argument of a file name to import files of different types, notably text files (extension *.txt) or comma separated data files (extension *.csv). Data from most spreadsheet programs can be saved in one or both of these file types. Note that scan() tends to produce formatting problems when reading files much more often than read and is not recommended for importing files.

Package foreign

Also of note is an R package called foreign. This package contains functionality for importing data into R that is formatted by most other statistical software packages, including SAS, SPSS, STRATA and others. Package foreign is available for download and installation from the CRAN site.

Troubleshooting Importing

Finally, sometimes you may have data in a format R will not understand. Sometimes for trouble imports with formatting that R cannot read, using scan () or the data editor to enter your data may be a simple and easy solution. Another trick is to try importing the data into another program, such as a spreadsheet program, and saving it as a different file type. In particular saving spreadsheet data in a text (comma or tab delineated) format is simple and useful. Caution should be used as some spreadsheet programs may restrict the number of data values to be stored.

Getting Help

Virtually everything in R has some type of accessible help documentation. The challenge is finding the documentation that answers your question. This section gives some suggestions for where to look for help.

Program Help Files

The quickest and most accessible source of help when using R is to use the on-line help system that is part of R. This includes on-line documentation for the R base packages, as well as on-line documentation for any loaded packages.

Finding help when you know the name of what it is your asking for help on is easy, just use the help function with the topic of interest as the argument of the help function. For example to get help on function sum:

```
> help(sum)
```

This produces a help file as depicted in Figure 3-5.

Permission granted to reproduce for nonprofit, educational use.

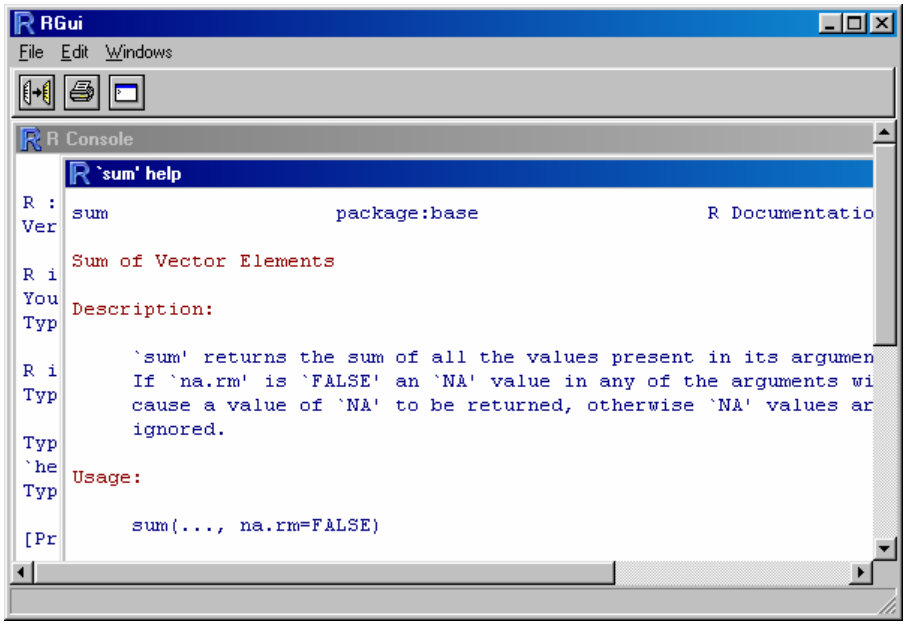


Figure 3-5

As an alternative to calling the help function you can just put a question mark in front of the topic of interest:

```
> ?sum
```

If you don't know the exact name of what you're looking for, the apropos() function can help. To use this function type in a part of the word you are looking for using quotes as the function argument. The result of calling apropos is that you will get a list of everything that matches that clue, as in Figure 3-6 for apropos("su"), and you can then do a help search on your term of interest.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

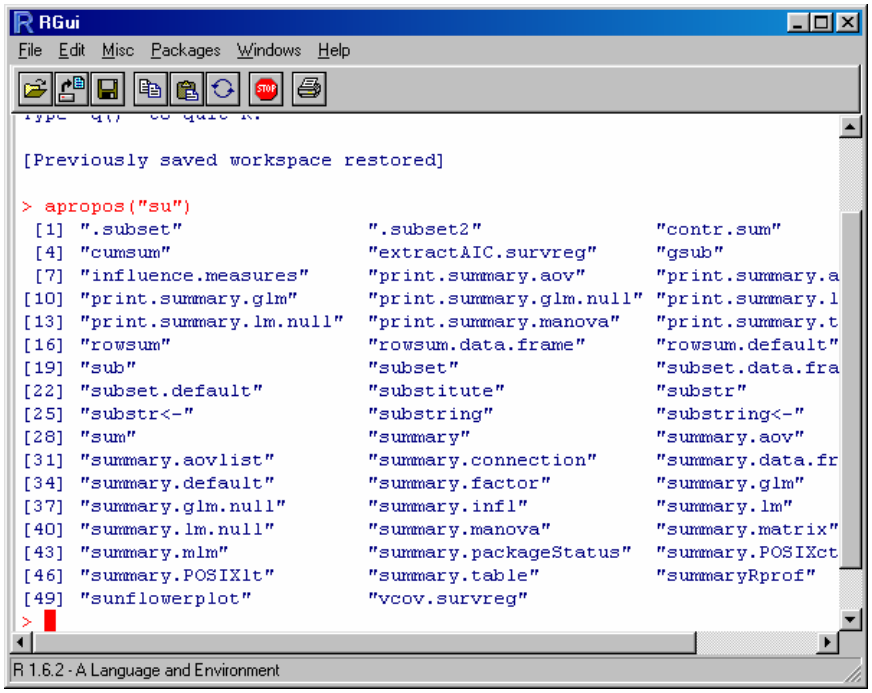


Figure 3-6

Note that on-line help is especially useful for describing function parameters, which this book will not discuss in great detail because many functions in R have lots of optional parameters (read.table for example has about 12 different optional parameters). The reader should be comfortable with looking up functions using help to get detailed parameter information to use R functions to suit their needs.

Documents

The R system and most packages are fairly well documented, although the documentation tends to be technical rather than tutorial. The R system itself comes with several documents that are installed as part of the system (recommended option). These are under the "Manuals" option on the "Help" menu.

In addition to manuals that cover R in general, most packages have their own documents. R has a system where package contributors create pdf files in standard formats with explain the technical details of the package, including a description of the package and its functionality. These documents also generally list the name and contact information for the author(s) of the package. These documents are available from the "Package Sources" section of the CRAN site (cran.r-project.org) where the relevant package is listed and are always listed as Reference Manual, although they save with a file name corresponding to the

Permission granted to reproduce for nonprofit, educational use.

package name. They can be downloaded and saved or viewed on-line using a web browser that reads pdf files.

Books

There are several books written that are about or utilize R. In addition, there are several books that cover the related topics of S and S-Plus. These are included in the resource section at the end of the book.

On-line Discussion Forums

One of the best sources of help if you have a specific question is the on-line discussion forums where people talk about R. These forums serve as "technical support" since R is open source and has no formal support system. There are usually many well-informed users who regularly read these discussion lists. A guide to such lists is found at *www.r-project.org* as depicted in Figure 3-7. In addition many other forums exist on the web where questions about R may be posted.

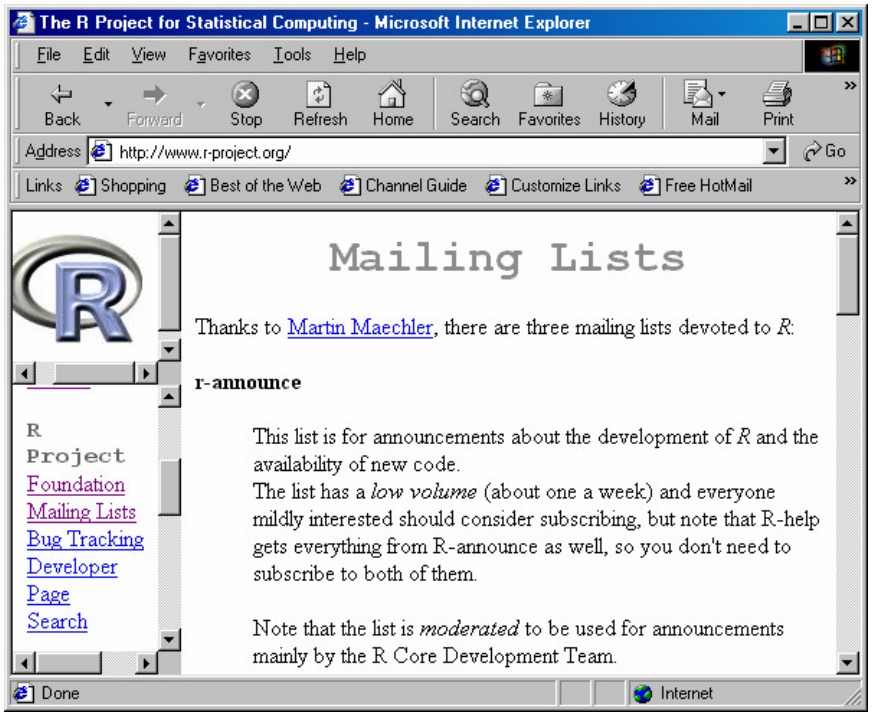


Figure 3-7

Permission granted to reproduce for nonprofit, educational use.

4

Programming with R

The goal of this chapter is for the reader to become acquainted with the basics of programming in R – using correct syntax, using logical statements, and writing function. This should empower the reader with a minimal set of programming skills required to effectively use R. No prior programming knowledge is assumed, however this would be helpful as many of the programming techniques R are similar to those of other high-level programming languages.

Variables and Assignment

The foundation of programming is using variables and assigning values to those variables. We have already seen in the previous chapter numerous examples of using the assignment operator "<-"to assign a value to a variable. Most of what we did in the previous chapter involved creating data objects and assigning the values and type of data object to a variable. This chapter will assign the results of function calls to variables as well.

Syntax

R is not incredibly fussy as far as syntax goes in comparison to other high-level languages. There are almost no restrictions on variable names, although you should use descriptive names whenever possible. As far as punctuation is concerned, semicolons are required to separate statements if they are typed on a single line, but are not required if statements are written on separate lines.

```
> x <- 5; y <- 7  # Same as the following two lines
> x <- 5
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> y <- 7
```

In R there is a great deal of room for personal style in terms of how you write your code. In particular you may insert spaces wherever you please. This is useful for making code look nice and easy to analyze.

```
> x<-y^2/5   # Same as the following
> x <- y^2 / 5
```

For most tasks there are many correct ways to achieve the goal

One standard however is comments, which are always written starting with "#"

```
# This is a comment
```

Many comments are written throughout the text.

Another important thing to note is that R is always case sensitive. This means that lower case letters have a different meaning from upper case letters. The word MyVariable is not the same as myvariable, and the two would be treated separately if used to hold variables as illustrated below:

```
> MyVariable<-5
> MyVariable
[1] 5
> myvariable
Error: Object "myvariable" not found
```

If you get the error object not found message (as above) and you are sure you created a variable of that name, check the case (or do an ls() to check all current objects in the workspace).

Arithmetic Operators

The basic arithmetic operators are listed in Table 4-1. The usual algebraic rules of precedence apply (multiplication and division take precedence over addition and subtraction). Use parenthesis "( )" to separate operations of priority out. Do not use bracket sets "[ ]" or "{ }" as these are for other purposes in R. Using arithmetic operators is very simple, and operations can be performed directly on numbers or on variables. Some trivial examples are listed below:

```
> 2+4
[1] 6
> y<-0
> x<-4
> x*y^2
[1] 0
> x^4
[1] 256
> z<-5
> (x+y)*z/x
[1] 5
```


Permission granted to reproduce for nonprofit, educational use.

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

Permission granted to reproduce for nonprofit, educational use.

Usually logical and relational operators are used with conditional programming statements if and else. The if statement can be used alone or with the else statement. The form of the if statement is:

```
if (condition is true)
       then do this
```

The condition in parenthesis usually will use a relational operator to determine if the condition is true. When the if statement is used alone, and the condition in parenthesis is not true then nothing happens. For example:

```
> x<-6
> y<-2
> #if x is less than or equal to y then add them to get z
> if(x<=y)z<-x+y
> #condition is not true so nothing happens
> z
Error: Object "z" not found
> #if the reverse relational operator is used
> #then the condition is true and z is assigned x+y
> if(x>=y)z<-x+y
> z
[1] 8
```

To get something to happen if the condition in parenthesis is not true, use if in conjunction with else to specify an alternative. Note that there is not a condition in parenthesis after the else statement. In this case the code following else will execute as long as the if condition is not true.

```
if (condition is true)
       then do this
else
       do this
```

For example:

```
> q<-3
> t<-5
> # if else conditional statement written on one line
> if(q<t){w<-q+t} else w<-q-t
> w
[1] 8
```

Note the use of { } brackets around some of the code. These curly bracket sets are frequently used to block sections of code, and will indicate code continues on the next line. This code can also be written:

```
> if(q<t){
+ w<-q+t
+ }else
+ w<-q-t
```

This separates the code onto different lines, which is unnecessary for this simple case but with longer code it becomes unwieldy to write all the code on one line.

Permission granted to reproduce for nonprofit, educational use.

The logical operators, &, |, and ! can be used to add additional selection criteria to a selection statement.

For example if you want to simultaneously select based on two criteria you can use the and (&) operator or the or (|) operator:

```
> a<-2
> b<-3
> c<-4

> #Using and to test two conditions, both true
> if(a<b & b<c) x<-a+b+c
> x
[1] 9

> #Using and to test two conditions, one is false
> if(a>b & b<c) y<-a-b-c
> y
Error: Object "y" not found

> #Using or to test two conditions, both false
> if(a==b | a>c) z<-a*b*c
> z
Error: Object "z" not found

> #Using or to test two conditions, one true
> if(a<b | a>c) z<-a*b*c
> z
[1] 24
```

The not operator (!) is used for many purposes including selecting based on inequality ("not equal to"):

```
> w<-2
> #
> if(w!=3)v<-w
> v
[1] 2

> #
> if(w!=2)u<-w
> u
Error: Object "u" not found
```

Looping

Control by repetition, or looping, allows you to efficiently repeat code without having to write the same code over and over. In R two common looping expressions are while and for.

The while loop repeats a condition while the expression in parenthesis holds true and takes the form

Permission granted to reproduce for nonprofit, educational use.

```
while (condition controlling flow is true)
       perform task
```

For example suppose we want to execute adding 1 to the variable x while x<=5:

```
> x<-0
> while(x<=5){x<-x+1}
> x
[1] 6
```

Note that x=6 at the end because the loop is still true at the beginning of the loop when x=5.

For loops are used to iterate through a process a specified number of times. A counter variable (usually designated by a lowercase letter "i") is used to count how many times the loop is executed:

```
for (i in start:finish)
       execute task
```

As an example, if you want to initialize a vector with values you can loop through it to assign the values:

```
> y<-vector(mode="numeric")
> for(i in 1:10){
+ y[i]<-i}
> y
 [1] 1 2 3 4 5 6 7 8 9 10
```

For loops can also be nested (one inside the other) to initialize matrices or perform similar tasks:

```
> z<-matrix(nrow=2,ncol=4)

> for(i in 1:2){
+ for(j in 1:4) z[i,j]<-i+j}
> z

     [,1] [,2] [,3] [,4]
[1,]    2    3    4    5
[2,]    3    4    5    6
```

Subsetting with Logical Operators

Although they are handy for doing simple repetitive tasks, for loops are not used as often in R as they are in other languages and are not recommended because they tend to be memory intensive, which can cause problems. Looping through 10,000 matrix data objects, for example, may not be a good idea. Fortunately R provides powerful alternatives to looping in the form of subsetting with logical operators. Subsetting is available for vectors, matrices, data frames and arrays using the [,] brackets.

Permission granted to reproduce for nonprofit, educational use.

Subsetting with logical operators is based on the following simple fact: Every logical statement produces one of the outcomes TRUE or FALSE. For example:

```
> x <- 7 ; y <- 3
> x > y
[1]  TRUE
> x < y
[1]  FALSE
```

Logical operators applied to vectors, matrices etc. will result in an object of the same dimension consisting of entries TRUE or FALSE depending on whether the statement is true for the particular element. For example:

```
> x <- 1:6
> y <- rep(4,6)
> x > y
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
> x <= y
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE
> x == y
[1] FALSE FALSE FALSE  TRUE FALSE FALSE
```

If we use the outcomes of a logical vector statement for subsetting a vector, only the elements where the outcomes are equals TRUE will be selected.

```
> # Select the elements of the vector (11:16) where x <= y
> (11:16)[x <=y ]
[1] 11 12 13 14
> # Select the elements of the vector (11:16) where x = y
> (11:16)[x == y]
[1] 14
```

Similar statements apply to matrices, data frames and arrays such as:

```
> A <- matrix(1:6,nrow=2)
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> A > 3
      [,1]  [,2] [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE  TRUE TRUE
```

An interesting way of "grabbing" elements out of a matrix is by subsetting as follows. Note that the result is a vector of elements.

```
> A[A > 3]
[1] 4 5 6
```

Functions

A function is a module of code that performs a specific task. Functions are called by another line of code that sends a request to the function to do

Permission granted to reproduce for nonprofit, educational use.

something or return a variable. The function call may or may not pass arguments to the function. A function takes the general form

```
functionName<-function(arg1, arg2…){
       do this }
```

Note that the assignment operator assigns the function to the function name. An alternative to this is to user an underline "functionName_function()" which you will see as well, although this may be deprecated in the most recent version. The functionName is a variable and creates an object. The word function is always explicitly used to define the variable as a function object.

The "do this" task is performed and/or the value is returned to the calling code. You should write functions when you have a repetitive task to do. This sections looks at using existing functions, and then at how to write functions. Throughout this book functions are used extensively.

Using Existing Functions

Even in the base package (with no additional packages installed) R supplies a large number of pre-written functions for you to use. Additional packages are filled with additional functions, usually with related functions for related tasks packaged together. The simplest functions are functions that perform basic mathematical tasks. Some selected mathematical functions for common operations are listed in Table 4-2. R has functionality pre-written for virtually any standard mathematical tasks. *Table 4-2*

Function	Operation Performed
sqrt(x)	Square root of x
abs(x)	Absolute value
sin(x), tan(x), cos(x)	Trigonometric functions
exp(x)	Exponential
log(x)	Natural logarithm
log10(x)	Base 10 logarithm
ceiling(x)	Closest integer not less than x
floor(x)	Closest integer not greater x
round(x)	Closest integer to the element

Permission granted to reproduce for nonprofit, educational use.

To use base package functions, simply call the function (these are always installed when you run R):

```
> z<-c(1,2,3,4,5,6)
> sum(z)
[1] 21
```

If you plan on using the results of the function call for other calculations, you should assign the result to a variable:

```
> y<-sum(z)
> y
[1] 21
```

To use functions from packages, always remember you have to have the package installed and loaded in order to use its functions!

Writing Functions

With R, you should always do some research before writing a complex statistical procedure function. Chances are, if it's a standard procedure, the functionality has already been written. Use the on-line resources and mailing list of the R community to search for what you need. If you do write novel functionality that may be of use to others, consider contributing it to the R community as a small package.

However, there will be many times you will want to write a task-oriented function to do some simple routine manipulations. Let's return to the data frame from the last chapter with data on gene counts and genome sizes for selected organisms:

```
comparativeGenomeSize
   organism genomeSizeBP estGeneCount
1     Human    3.000e+09        30000
2     Mouse    3.000e+09        30000
3 Fruit Fly    1.356e+08        13061
4 Roundworm    9.700e+07        19099
5     Yeast    1.210e+07         6034
```

Let's add a column to this data frame calculating base pairs per gene. To do this, let's write function that takes as arguments the genome size and estimated gene count information, and calculates from this the estimated bp per gene:

```
> #function geneDensity simply calculates bp per gene
> geneDensity<-function(bp,genes){
+ bp/genes}

> #pass data frame columns to function geneDensity
> #storing results in variable bpPerGene
> bpPerGene<-geneDensity(comparativeGenomeSize$genomeSizeBP,
+ comparativeGenomeSize$estGeneCount)
> #result of function computation
> bpPerGene
[1] 100000.000 100000.000  10382.053   5078.800   2005.303
```


Permission granted to reproduce for nonprofit, educational use.

To round the numbers to the nearest integer we can call round, one of the simple math functions from Table 4-2:

```
> bpPerGene<-round(bpPerGene)
> bpPerGene
[1] 100000 100000  10382   5079   2005
```

Next, to create a new data frame with all the information we can use techniques of data frame manipulation from the previous chapter:

```
> comparativeGenomes<-cbind(comparativeGenomeSize,bpPerGene)
> comparativeGenomes
   organism genomeSizeBP estGeneCount bpPerGene
1     Human    3.000e+09        30000    100000
2     Mouse    3.000e+09        30000    100000
3 Fruit Fly    1.356e+08        13061     10382
4 Roundworm    9.700e+07        19099      5079
5     Yeast    1.210e+07         6034      2005
```

This book is filled with additional examples of task-oriented functions and you should readily become fluent in their use.

Package Creation

One of the most important features of R is that it allows the user to create packages and contribute these to the R community for others to use. The discussion of how to create packages is beyond our coverage here. However, the document "Writing R Extensions" covers the process of creating a package (which is not technically difficult). This document is available as part of the (optional installation) help files with the program or by download from the CRAN site.

Permission granted to reproduce for nonprofit, educational use.

5

Exploratory Data Analysis and Graphics

This chapter presents some rudimentary ways to look at data using basic statistical summaries and graphs. This is an overview chapter that presents basics for topics that will be built on throughout this book.

Exploratory Data Analysis

This section covers ways to quickly look at and summarize a data set using R. Much of this material should be familiar to anyone who studied introductory statistics.

Data Summary Functions in R

Table 5-1 lists many of the basic functions in R that are used to summarize data. Notice that usually, the function name is what you would expect to be in most cases, as R is designed to be rather intuitive. For example, the function to find the mean (or average value) of a data vector x is simply mean(x).

For functions not listed in table 5-1, try help and the expected name or using the apropos() function with part of the expected name to find the exact function call. It will probably be what you expect it to be. Most of the functions in table 5-1 do not have additional parameters, and will work for a data vector, matrix or data frame.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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)
```


Permission granted to reproduce for nonprofit, educational use.

```
[1] 0.01313333
> sd(x)
[1] 0.1146008
```

Often, you will use these basic descriptive functions as part formulas for other calculations. For example, the standard deviation (supposing we didn't know it had its own function) is the square root of the variance and can be calculated as:

```
> sd<-var(x)^0.5
> sd
[1] 0.1146008
```

The summary() Function

Suppose we have an enzyme that breaks up protein chains, which we'll call ChopAse. Suppose we have three varieties of this enzyme, made by three companies (producing the same enzyme chemically but prepared differently). We do some assays on the same 200 amino acid protein chain and get the following results for digestion time based on variety:

```
> ChopAse
  varietyA timeA varietyB timeB varietyC timeC
1        a  0.12        b  0.12        c  0.13
2        a  0.13        b  0.14        c  0.12
3        a  0.13        b  0.13        c  0.11
4        a  0.12        b  0.15        c  0.13
5        a  0.13        b  0.13        c  0.12
6        a  0.12        b  0.13        c  0.13
```

Based on this data and the way it is presented it is difficult to determine any useful information about the three varieties of Chopase and how they differ in activity. Here is a case where using the summary function can be very useful as a screening tool to look for interesting trends in the data.

The summary function simultaneously calls many of the descriptive functions listed in Table 5-1, and can be very useful when working with large datasets in data frames to present quickly some basic descriptive statistics, as in the ChopAse example:

```
> summary(ChopAse)
 varietyA     timeA       varietyB     timeB        varietyC     timeC
 a:6      Min.   :0.120   b:6      Min.   :0.1200   c:6      Min.   :0.1100
          1st Qu.:0.120            1st Qu.:0.1300            1st Qu.:0.1200
          Median :0.125            Median :0.1300            Median :0.1250
          Mean   :0.125            Mean   :0.1333            Mean   :0.1233
          3rd Qu.:0.130            3rd Qu.:0.1375            3rd Qu.:0.1300
          Max.   :0.130            Max.   :0.1500            Max.   :0.1300
```

This gives some quick quantitative information about the dataset without having to break up the data frame or do multiple function calls. For example the mean for variety B appears higher then the mean time for varieties A or C. This may be statistically significant, and this observation can be utilized for statistical testing of differences (such as those covered in Chapter 13).

Permission granted to reproduce for nonprofit, educational use.

Working with Graphics

Of course, it is often most interesting to present data being explored in a graphical format. R supports a variety of graphical formats in the base package, and numerous other packages provide additional graphics functionality. This section focuses on understanding the graphics environment in R and how to control features of the graphics environment. Some basic graph types are introduced, however many more examples of graphs are introduced in various chapters in a more specialized context.

Graphics Technology in R

At startup, R initiates a graphics device drive. This driver controls a special graphics window for the display of graphics. To open this window without calling graphics, use the function call windows() (for Windows operating systems, it is x11() on Unix, and macintosh() on Mac O/S).

Commands for using graphics in R can be categorized into three types: high-level plotting functions, low-level plotting functions, and graphical parameter functions. Let's look at each of these types of plotting commands.

High-Level Plotting Functions

High-level plotting functions create a new plot on the graphics device. The simplest high level plotting function is plot(), as illustrated below

```
> x<-c(1,2,3,4,5,6,7,8)
> y<-c(1,2,3,4,5,6,7,8)
> plot(x,y)
```

This produces the simple graph in Figure 5-1. Plot also works with only one argument, ex: plot(x), except in this case the plot would be of the values of vector x on the y-axis and the indices of value x on the x- axis (which would appear identical to Figure 5-1 in this case, try it!). Plot is a generic function and has a diverse set of optional arguments. For example, if you type in help (plot) you get the help screen shown in Figure 5-2. Under the usage section is the list of the function and most of the parameters, which are described in more detail in the arguments section of help below. Note there are parameters for labeling the x and y axises, main for a main title, and parameters for controlling the axis lengths (for example, we could have included a y range from 0 to 20 if we wanted). These parameter arguments are pretty standard for most of the high level plotting functions in R. Unless specified all the arguments are set to defaults, such as those utilized in Figure 5-1.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

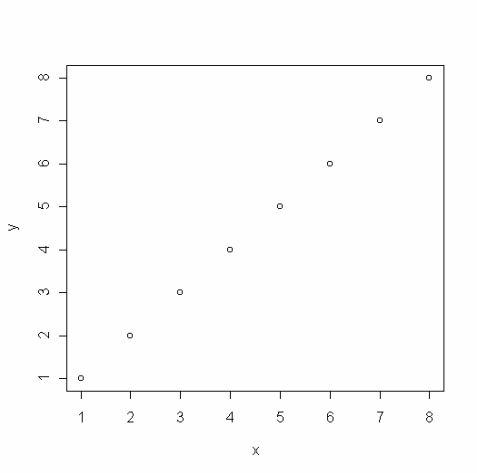


Figure 5-1

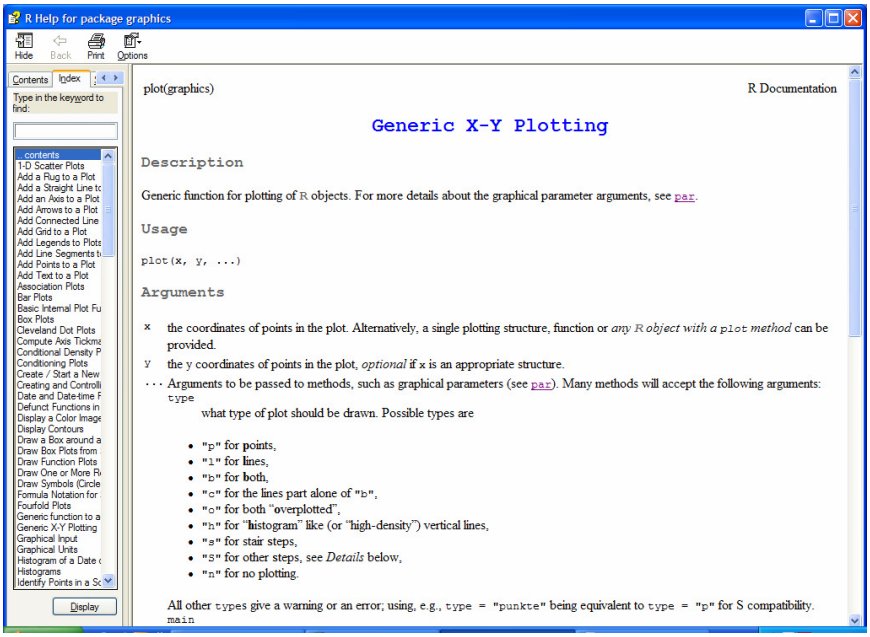


Figure 5-2

For example, suppose we want to make a little fancier plot than the one in Figure 5-1 and use some of the parameters available:

```
> plot(x,y,xlim=range(0:10),ylim=range(0:10),type='b',main="X vs Y")
```

This changes the x and y ranges on the graph from 1 to 8, to 0 to 10. It also changes the type of plot from the default of points, to both points and lines. In addition it adds the main title to the graph "X vs Y".

Permission granted to reproduce for nonprofit, educational use.

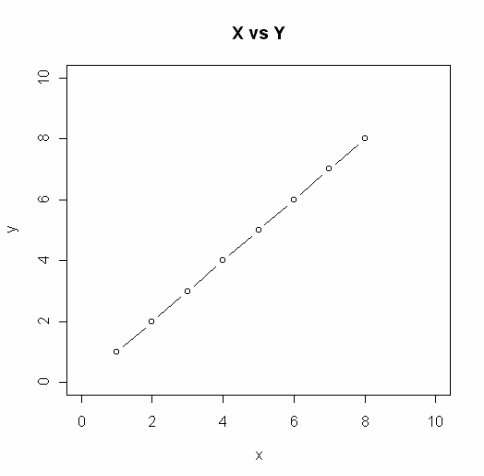


Figure 5-3

Table 5-2 lists other selected high-level plotting functions:

Table 5-2: Selected High-Level Plotting Functions

Function name	**Plot produced**
boxplot(x)	"Box and whiskers" plot
pie(x)	Circular pie chart
hist(x)	Histogram of the frequencies of x
barplot(x)	Histogram of the values of x
stripchart(x)	Plots values of x along a line
dotchart(x)	Cleveland dot plot
pairs(x)	For a matrix x, plots all bivariate pairs
plot.ts(x)	Plot of x with respect to time (index values of the vector unless specified)
contour(x,y,z)	Contour plot of vectors x and y, z must be a matrix of dimension rows=x and columns=y
image(x,y,z)	Same as contour plot but uses colors instead of lines
persp(x,y,z)	3-d contour plot

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Rather than illustrate further examples here, virtually all of the high-level plotting functions in Table 5-2 are utilized in coming chapters of this book.

Low-Level Plotting Functions

Low-level plotting functions add additional information to an existing plot, such as adding lines to an existing graph. Note that there is some redundancy of low-level plotting functions with arguments of high-level plotting functions. For example, adding titles can be done as arguments of a high-level function (main="", etc) or as a low-level plotting function (title(main="), etc).

For example, let's return to Figure 5-1. We could add to this plot in several ways using low-level plotting functions. Let's add a title, some text indicating the slope of the line and a line connecting the points.

```
> text(4,6,label="Slope=1")
> title("X vs Y")
> lines(x,y)
```

This embellishes the plot in Figure 5-1 to become the plot in Figure 5-4.

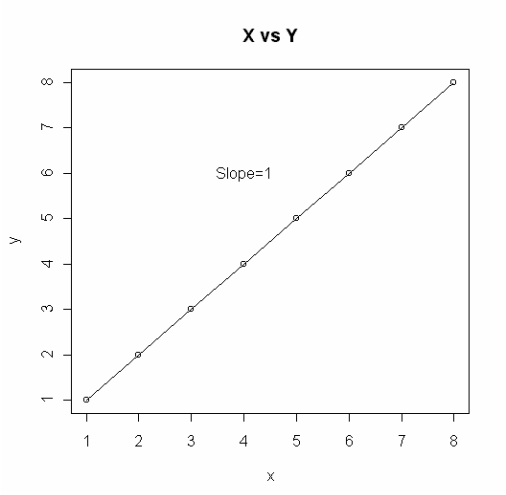


Figure 5-4

Table 5-3 lists additional low-level plotting functions. Note that most of these work not just with plot () but with the other high-level plotting functions as well. If you are working with multiple plots (as we shall soon see how to do) on one graphics window, the low-level plotting function used will apply to the most recently added plot only so you should write your code in the appropriate order.

Permission granted to reproduce for nonprofit, educational use.

Table 5-3: Selected Low-Level Plotting Functions

Function name	Effect on plot
points(x,y)	Adds points
lines(x,y)	Adds lines
text(x, y, label="")	Adds text (label="text") at coordinates (x,y)
segments(x0,y0,x1,y1)	Draws a line from point (x0,y0) to point (x1,y1)
abline(a,b)	Draws a line of slope a and intercept b; also abline(y=) and abline(x=) will draw horizontal and vertical lines respectively.
title("")	Adds a main title to the plot; also can add additional arguments to add subtitles
rug(x)	Draws the data on the x-axis with small vertical lines
rect(x0,y0,x1,y1)	Draws a rectangle with specified limits (note –good for pointing out a certain region of the plot)
legend(x,y,legend=,…)	Adds a legend at coordinate x,y; see help(legend) for further details
axis()	Adds additional axis to the current plot

Graphical Parameter Functions

Graphical parameter functions can be categorized into two types: those that control the graphics window and those that fine-tune the appearance of graphics with colors, text, fonts, etc. Most of these can be controlled with a function in the base package called par(), which can be used to access and modify settings of the graphics device.

par() is a very important graphics function, and it is well worth the time to read the help document for par, pictured in Figure 5-5. The settings changed by par remain in effect in the current workspace until they are explicitly changed.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

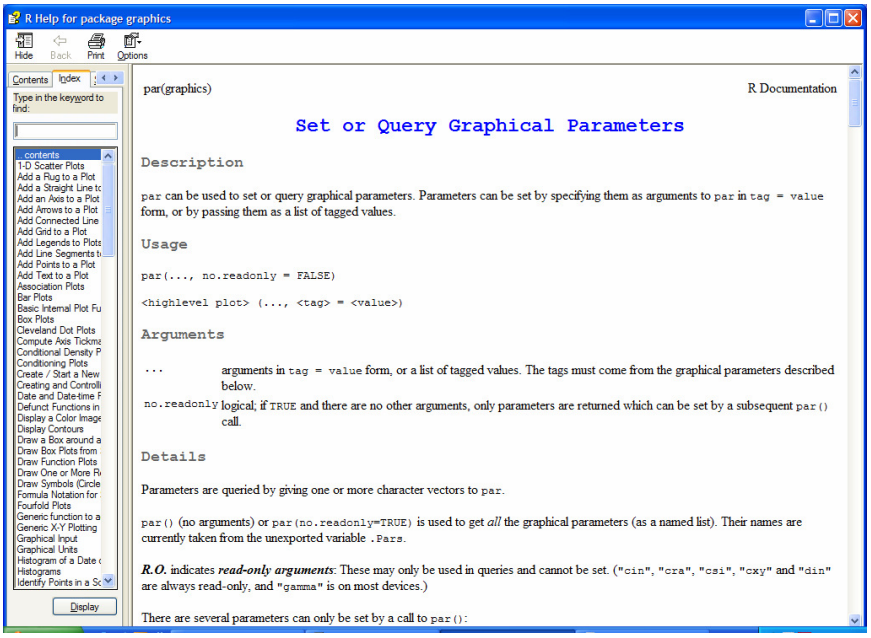


Figure 5-5: Par() function help

One of the most common tasks you will want to do with par is to split the graphics screen to display more than one plot on the graphic device at one time. You can do this with either the mfrow or mfcol parameters of the par function. Both mfrow and mfcol takes arguments (rows, columns) for the number of rows and columns respectively. Mfrow draws the plots in row order (row 1 column 1, row 1 column 2, etc) whereas mfcol draws plots in column order (row 1 column 1, row 2 column 1).

Graphics parameters also control the fine-tuning of how graphics appear in the graphics window. Table 5-4 lists some of the basic graphical parameters. Most of these parameters can be implemented as parameters of par, in which case they are implemented in all graphs in a graphics window, or as parameters of high or low level plotting functions, in which case they only affect the function specified.

Let's look at an example that utilizes some of par's functionality using data from NCBI on numbers of base pairs and sequences by year.

```
> NCBIdata
   Year   BasePairs Sequences
1  1982      680338       606
2  1983     2274029      2427
3  1984     3368765      4175
4  1985     5204420      5700
5  1986     9615371      9978
…
20 2001 15849921438  14976310
21 2002 28507990166  22318883
```


Permission granted to reproduce for nonprofit, educational use.

Table 5-4: Selected Graphical Parameters

Parameter	Specification
bg	Specifies (graphics window) background color
col	Controls the color of symbols, axis, title, etc (col.axis, col.lab, col.title, etc)
font	Controls text style (0=normal, 1-=italics, 2=bold, 3=bold italics)
lty	Specifies line type (1:solid, 2:dashed, 3: dotted, etc)
lwd	Controls the width of lines
cex	Controls the sizing of text and symbols (cex.axis,cex.lab,etc)
pch	Controls the type of symbols, etiher a number from 1 to 25, or any character within “”

Using the NCBI data, let’s plot base pairs by year plot and sequences by year plot on the same graphics window:

```
> #Convert Base Pair Data to Millions
> MBP<-NCBIdata$BasePairs/1000000

> #Convert Sequence Data to Thousands
> ThousSeq<-NCBIdata$Sequences/1000

> #Set par to have a 2-column (2 graph) setup
> #Use cex to set label sizes
> par(mfcol=c(1,2),cex.axis=0.7,cex.lab=1)

> #Plot base pair data by year
> plot(NCBIdata$Year,MBP,xlab="Year",ylab="BP in Millions",
+  main="Base Pairs by Year")

> #Add line to plot, color blue
> lines(NCBIdata$Year,MBP,col="Blue")

> #Similarily, plot sequence data and line
> plot(NCBIdata$Year,ThousSeq,xlab="Year",ylab="Seq. in Thousands",
+ main="Sequences by Year")
> lines(NCBIdata$Year,ThousSeq,col="red")
```

The resulting plot is shown in Figure 5-6.

Permission granted to reproduce for nonprofit, educational use.

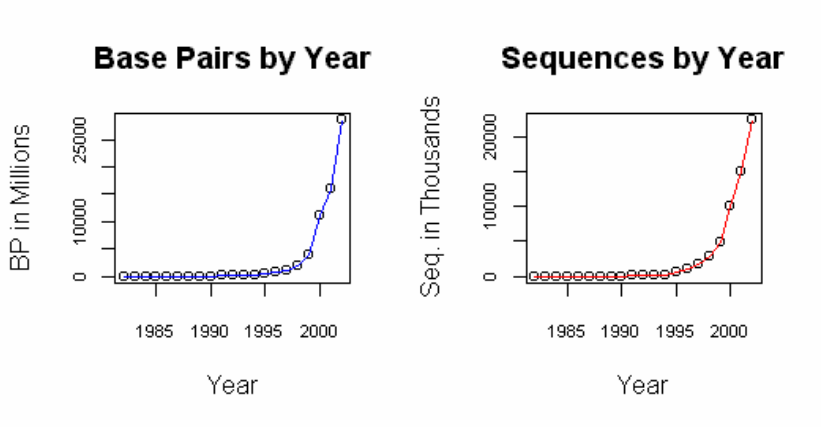


Figure 5-6

Another way to represent this data might be to use barplots, as illustrated in Figure 5-7.

```
> #Code for Figure 5-7
> par(mfcol=c(2,1),cex.axis=0.6,cex.lab=0.8)
> barplot(NCBIdata$BasePairs,names.arg=NCBIdata$Year,
+ col=grey,xlab="Years",ylab="Base Pairs",main="Base Pairs by Year")
> barplot(NCBIdata$Sequences,names.arg=NCBIdata$Year,
+ col=grey,xlab="Years",ylab="Sequences",main="Sequences by Year")
```

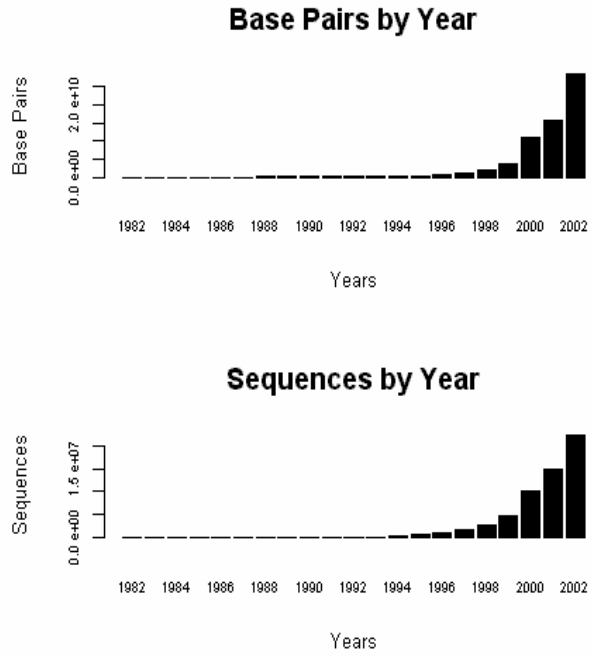


Figure 5-7

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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.

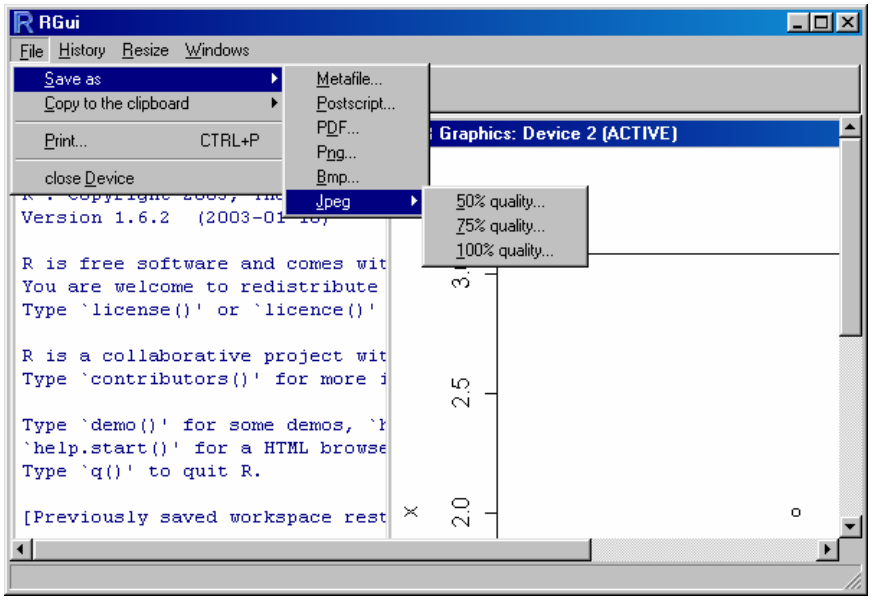


Figure 5-8

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

Permission granted to reproduce for nonprofit, educational use.

environment the copy options are as a bitmap or metafile, and the save options are as metafile or postscript.

Additional Graphics Packages

R has available many packages that add graphical capabilities, enhancing graphic capabilities available in the base package. This section presents some selected graphics packages that may be of interest, all of which should be available from the CRAN site.

mimR

mimR is a package that provides an R interface to a program called MIM. MIM is a software package, which is useful for creating graphical models, including directed and chain graphs. Although MIM is a commercial package, there is a free student edition available with limited functionality. We will see in coming chapters that these types of models, which are popular among computer scientists, are useful in advanced statistical modeling, such as Bayesian statistics and Markov Chain Monte Carlo modeling.

scatterplot3d

scatterplot3d is a valuable package that adds functionality that the base package lacks, that of producing effective 3d plots. It is also relatively simple for the beginner to use and contains one function scatterplot3d() with many flexible parameter options which create many different 3d plots, such as the demo plot in Figure 5-9.

Permission granted to reproduce for nonprofit, educational use.

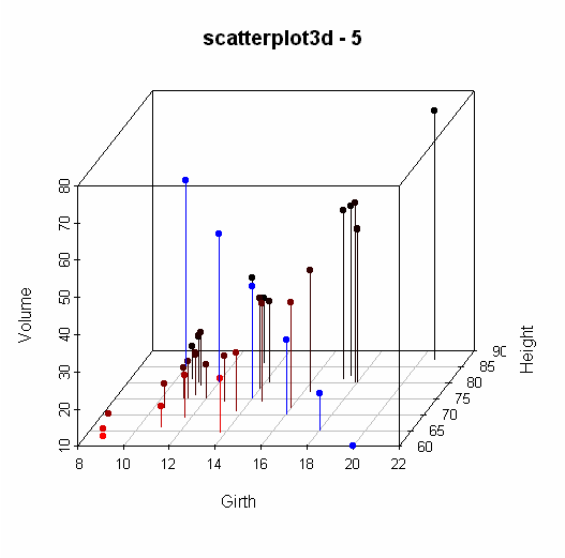


Figure 5-9

grid

Grid is a sophisticated and very useful R package, which provides a "rewrite of the graphics layout capabilities" (from package description on CRAN site). It works with base package functionality and adds some better capabilities for graphics in R. Some added functionality available with this package includes: allowing interactive editing of graphs, improving axis labeling capabilities, and primitive drawing capabilities.

lattice

The package lattice is quite sophisticated and contains extensive functionality for advanced graphics based on what are referred to often as Trellis graphics (the specific type used in other versions of S). This type of graphics is very useful for applications in multivariate statistics as they allow for presenting many graphs together using common x- and y-axis scales which is a visually effective way for doing comparisons between groups or subgroups. Cleveland (1993) presents the original foundation for this type of graphic. Figure 5-10 presents one of the demonstrations of a type of plot available in the lattice package.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

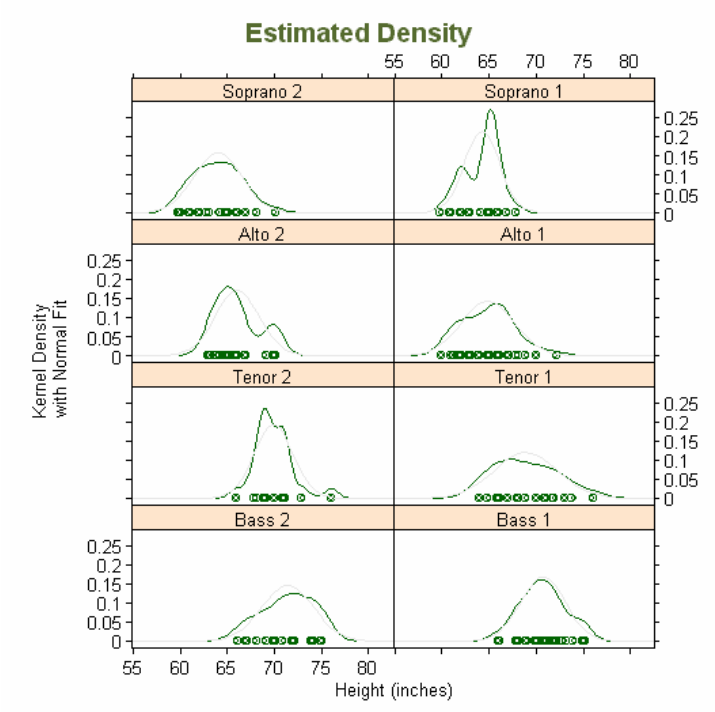


Figure 5-10

Permission granted to reproduce for nonprofit, educational use.

6

Foundations of Probability Theory

Aristotle The probable is what usually happens.

Cicero Probability is the very guide of life

Democritus Everything existing in the universe is the fruit of chance.

Probability is a mathematical language and framework that allows us to make statistical statements and analyze data. Probability focuses on the description and quantification of chance or random events. Understanding probability is key to being able to analyze data to yield meaningful and scientifically valid conclusions.

This chapter introduces some fundamentals of probability theory, beginning with an overview of the two schools of thought concerning how to think about probability. After this, some concepts of probability are introduced and finally, we begin the study of probability distributions. The contents of this chapter serve as an important foundation for subsequent chapters.

Two Schools of Thought

Most people who have formally studied probability are familiar with the more traditional way of thinking about probability. However it is important to note early on that there are two major approaches to understanding probability and statistics. The second, and less familiar (rarely incorporated into high school and college introductory courses), approach is of increasing importance in biological applications.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The most familiar way of thinking about probability is within a framework of repeatable random experiments. In this view the probability of an event is defined as the limiting proportion of times the event would occur given many repetitions. A good example of this frequentist definition of probability is defining the probability of a coin landing on heads by measuring the proportion of times a fair coin lands on its head out of the total times it is tossed. Application of the frequentist approach is limited to scenarios where frequent repetitions of the same random experiment are possible. Within the frequentist paradigm, applying probability theory to a novel scenario, with no clear sampling frame for repetitions of the same experiment, is virtually impossible.

The second way of thinking about probability allows the incorporation of an investigator's intuitive reasoning. Instead of exclusively relying on knowledge of the proportion of times an event occurs in repeated sampling, this approach allows the incorporation of subjective knowledge, such as historical information from similar experiments as the one under study, information from experiments related to the one under study, an educated guess about outcomes, or even, subjective beliefs of the investigator related to the problem under study. These so-called prior probabilities are then updated in a rational way after data are collected. The common name for this approach is Bayesian statistics, named after Sir Thomas Bayes, a nonconformist English minister who lived from 1702-1761. Bayes never published his work in his lifetime, but other mathematicians followed up on his work, resulting in the publication and promotion of his ideas.

Bayesian statistics has experienced explosive growth in recent years, with many applications in areas of science, economics, engineering and other fields. Until recently Bayesian statistics was largely theoretical and mathematically complex. However, powerful computers and software packages such as R are now common and Bayesian statistics theory is now commonly applied in the development of powerful algorithms and models that process data in new ways. Many of these algorithms and models have applications in bioinformatics, and some of these will be introduced in this book.

The two approaches are based on the same basic rules of probability. Understanding both, classical (or frequentist) and Bayesian statistics requires knowledge of the theory and methods of probability and probability models. Bayesian statistics essentially expands frequentist statistics by combining the interpretations of probability, thus increasing applicability of statistics to scenarios either not accessible by frequentist statistics or better served by the more complex analysis Bayesian statistics has to offer.

This chapter reviews the essentials of probability and serves as a conceptual foundation for much of the material in the subsequent chapters of this section and for various other chapters in this book. Chapter 7 covers specific univariate (one-variable) probability models commonly used and chapter 8 covers some more advanced probability topics and multivariate probability models. Chapter 9 specifically introduces Bayesian theory and approaches to modeling. For a more in-depth coverage of frequentist probability theory the reader is referred to read

Permission granted to reproduce for nonprofit, educational use.

a text on probability such as Sheldon Ross's "A First Course in Probability" or other recommended references listed in the appendix.

Essentials of Probability Theory

Probability theory is based on the idea of studying random, or chance, outcomes of an experiment. A keyword here is random. Random means the outcomes of the event are not fixed in advanced, and hence when the same experiment is repeated the results will likely be somewhat different, resulting in some variability in the response. Randomness is an essential concept in most of statistics. The word experiment is used broadly, and is not limited to planned scientific experiments.

To analyze an experiment in terms of probability, first the set of outcomes of a random experiment must be defined. Then, probabilities can be assigned to outcomes of the experiment. This is best explained by an example. Suppose the experiment is one roll of a standard, six-sided die. The set of all possible outcomes is defined by the set of numbers that represent the number of dots on the face of the die that turns up as a result of the die roll. In the case of a standard 6-sided die the set is {1,2,3,4,5,6}. This set, containing all possible outcomes of the experiment, is known as the sample space. A subset of the sample space is defined to be an event. It is to events that probabilities are assigned.

For the sample space {1,2,3,4,5,6} different events can be defined. An event may be a single outcome, or a subset containing multiple outcomes. For example, the event could be that the die lands with one dot up. Because in this example all possible outcomes are equally likely and there are six possibilities, the probability for this event is 1/6. Another event could be the event that the die lands on an even number, which corresponds to the subset {2,4,6}. In this case the probability of the event is ½.

Set Operations

Since the sample space is defined by the set of possible outcomes, it should be clear that probability relies on basic operations of set theory. These basic operations provide a logical framework for understanding how events relate in the sample space. It is helpful to review these here.

Set theory makes use of Venn diagrams, as depicted in Figure 6-1. The entire diagram represents the sample space, and any sub-area, such as a circle represents a particular event.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

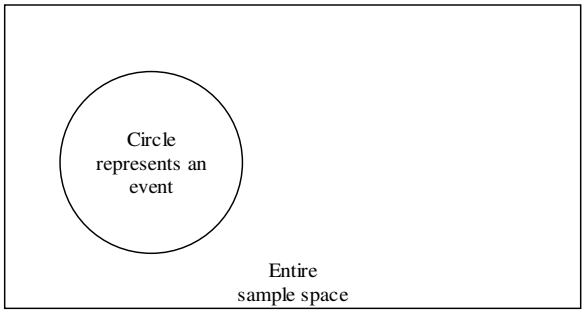


Figure 6-1: Basic Venn Diagram

Set Operation 1: Union

The union of two events, A and B, in the same sample space, is the event that either A or B occurs or both occur. In casual wording the union can be referred to as "A or B occurs" or "at least one of the events A and B occurs". Mathematically the statement for a union is $A \cup B$. The union of A and B is depicted in Figure 6-2.

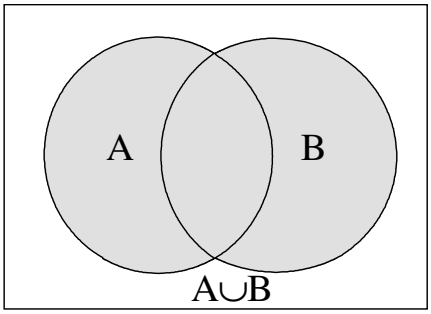


Figure 6-2: Union of Two Events

Set Operation 2: Intersection

The intersection of two events in the same sample space, A and B, is the event that contains all outcomes that are common to both A and B. In casual wording an intersection can be referred to as "both events A and B occur". Mathematically the statement for an intersection is $A \cap B$. Later in chapter 8 we will discuss joint probabilities and joint distributions where the concept of intersection plays a key role. The intersection of A and B is depicted in the Venn diagram in Figure 6-3.

Permission granted to reproduce for nonprofit, educational use.

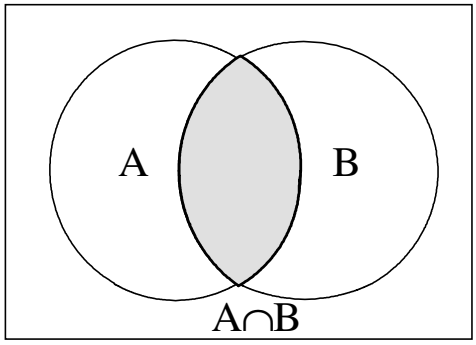


Figure 6 -3: Venn Diagram of the Intersection of Two Events

Set Operation 3: Complement

The complement of event A, depicted below and represented mathematically as A^c, is the event that A does not occur. The complement of A consists of all outcomes in the sample space that are not in A, as illustrated in Figure 6-4. It is a favorite statistical trick that if the probability of event A is complicated to calculate, it is sometimes easier to calculate instead the probability of the complement of A (which equals 1 minus the probability of event A).

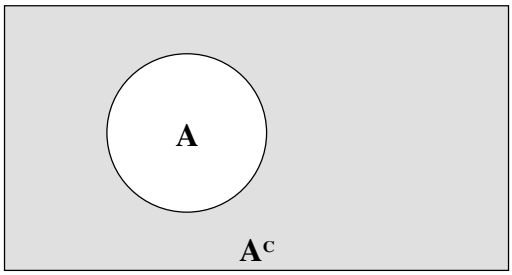


Figure 6-4: Venn Diagram of the Complement of an Event

Disjoint Events

Two events, A and B, in the same sample space, are disjoint if their intersection contains no elements. In other words, the intersection of A and B is the empty (or null) set, denoted by ∅, as depicted in Figure 6-5. To put the definition in plain English, disjoint events are events that have nothing in common with each other. Disjoint events are said to be mutually exclusive, and the terms disjoint and mutually exclusive are interchangeable. As a cautionary note, although the terms disjoint and mutually exclusive mean the same thing, these terms should not be interchanged with the term independence, which has a completely different meaning, to be discussed in chapter 8.

Permission granted to reproduce for nonprofit, educational use.

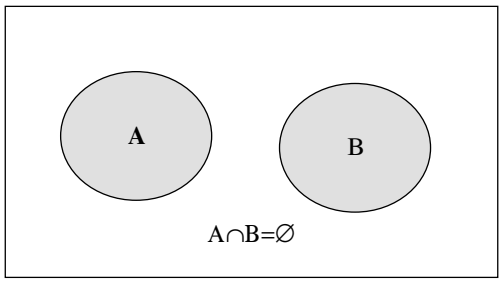


Figure 6-5: Intersection of Disjoint Events is the Null Set

The Fundamental Rules of Probability

Some simple rules (axioms) are used in probability to guarantee a consistent notion of how probability represents the chance of events related to random experiments. Axioms are statements of assumed truth, which do not require proof, and serve as the foundation for proving theorems.

Rule 1: Probability is always positive

Probability is always positive and the concept of negative probability does not exist. The values of probability of an event A, denoted P (A), must range from zero to one, reflecting the concept of probability as a measure of proportion. For an event to have probability equal to zero, it means the event is impossible. For an event to have probability equal one, it means the event is certain.

Rule 2: For a given sample space, the sum of probabilities is 1

For a simple example, consider the sample space of the experiment of picking a nucleotide at random from the four possible nucleotides. For this experiment the sample space is {A, T, C, G}, as depicted in Figure 6-6, and, assuming equal frequencies of all four nucleotides, P (A)=P (T)=P(C)=P (G)=1/4. Thus, the sum of all probabilities for this sample space is one.

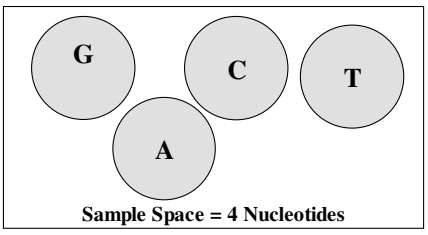


Figure 6-6: Sample Space for 4 Nucleotides

Permission granted to reproduce for nonprofit, educational use.

For each event in a sample space where the outcomes have equal probability of occurring, the probability of an event equals the frequency of the event over the total events in the sample space. If the sample space contains 6 nucleotides {G, C, A, T, A, C} as depicted in Figure 6-7, then P (A)=P (C) = 1/3 and P (T)=P (G)=1/6. But the sum of events in the sample space is still 1; the probabilities of individual nucleotides are changed to reflect the composition of events in the sample space.

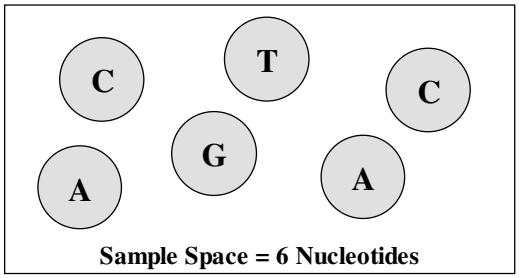


Figure 6-7:Sample Space for 6 Nucleotides

This axiom is also what allows the calculation of the complement of an event. Suppose in the 4-nucleotide example that only P (A) is known and we want to calculate the probability of any other nucleotide (which is the event of the complement of A). Using the formula P (A^c) = 1-P (A) this is a trivial task, for example if P (A) is 0.25, then P (A^c) =1-0.25 or 0.75. Figure 6-8 depicts the complement of A in the shaded out area.

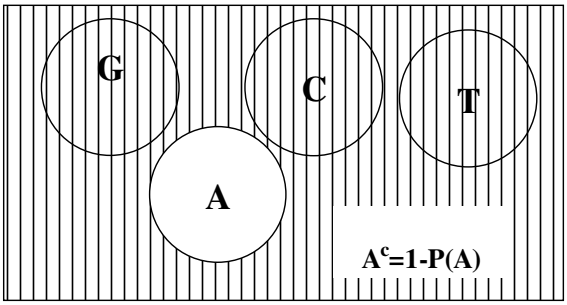


Figure 6-8:The Complement of A is everything in the Sample Space Except the Event A

Rule 3: For disjoint (mutually exclusive) events, P (A ∪ B)=P (A) + P (B)

In the case of disjoint events, there is no intersection of events, so the probability of their unions is simply the sum of their probabilities. For a sample space consisting of two disjoint events, the calculation of total probability is trivial (P(A)+P(B)=1). However, many sample spaces consist of many possible

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

outcomes (theoretically up to infinity, but denoted below as n). In this case, if the outcomes are disjoint events, then the probability of all events can be represented by

$P(E1) \cup P(E2) \cup P(E3) \ldots \cup P(En) = P(E1) + P(E2) + P(E3) + \ldots + P(En)$

Or, in shorthand notation

$$P\left(\bigcup_{i=1}^{n} Ei\right) = \sum_{i=1}^{n} P(Ei)$$

For events that are not disjoint, the calculation of P(A U B) must take into account the area of intersection for the events, and is calculated as $P(A \cup B)=P(A)+P(B)-P(A \cap B)$, subtracting the area of intersection (which otherwise would be double counted). For more than 2 events that are not disjoint this calculation becomes more complicated as it is necessary to count any regions of intersection only once.

Counting

It is also helpful to review here the methods of counting, and how R can be used as a tool for helping out with count calculations. Many times in probability you will need to be able to quantify things before assigning probabilities and applying counting methods is essential to doing this. The mathematical theory of counting is called combinatorial analysis.

The Fundamental Principle of Counting

The fundamental principle of counting (alternatively known as the multiplication principle), applied to two experiments, is that if the number of outcomes of experiment 1 is m, and then number of outcomes of experiment 2 is n, then the total number of outcomes for the two experiments is m * n.

For example, suppose the number of alleles possible for gene A is 3. In probability terms this can be considered as the number of outcomes of experiment 1 where the experiment is the choice of alleles for gene A. And suppose the number of alleles for gene B is 5 (outcomes of experiment 2). The total number of outcomes for these two experiments is 3*5 or 15, which is the number of possible combined outcomes of gene A and gene B.

The fundamental principle of counting can be applied to multiple experiments by extension of the two-experiment scenario. If k experiments have outcomes

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

n1, n2, n3…nk, then together the k experiments have a total of n1*n2*n3*…*nk possible outcomes.

For example, suppose a protein complex consists of 5 proteins encoded by 5 different genes. Suppose for protein 1 there are 4 genetic alleles, for protein 2 there are 2 genetic alleles, for protein 3 there are 9 genetic alleles, for protein 4 there are 11 alleles, and for protein 5 there are 6 alleles. How many different genetic alleles are involved in this protein complex? The answer is a straightforward multiplication of the number of alleles involved in all 5 proteins, which equals 4*2*9*11*6 for a total of 4752 possible combinations of alleles involved in this 5 protein complex.

Permutations (Order is Important)

When picking distinct objects and arranging them there are two scenarios to consider: order important and order unimportant. For example, let's pick the letters a, b, and c. We can pick three letters and count the number of unique ways we can arrange them (order important). In this case order is important and the results is the six combinations: abc, acb, bca, bac, cab, and cba. There are a total of 3!=3*2*1=6 possible permutations of arranging three distinct letters in groups of 3. Note the use of the factorial notation and recall that in general n! =n*(n-1)*…*1

A permutation of objects occurs when objects are arranged so that order is important. Mathematically the formula for a permutation or an arrangement of r out of n distinct objects (order is important) is:

$$Pn,r = \frac{n!}{(n-r)!}$$

Combinations (Order is Not Important)

What about the case where order is not important? For example, in the case of the letters a, b, c what if all we want to know is how many ways we can select 2 out of 3 letters, regardless of order? In this case the answer is 3 because it doesn't matter whether the order of the letters is ab or ba or any of the other combinations of two letters.

A combination of objects occurs when objects are selected and order of arrangements is not important. Mathematically the formula for a combination of selecting r out of n distinct objects (order unimportant) is:

$$Cn,r = \frac{n!}{r!(n-r)!}$$

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Note that the number of permutations is related to the number of combinations of objects by the following relationship:

$$Pn, r = r!Cn, r$$

Using permutations and combinatorial calculations, probabilities in a sample space where all outcomes are equally likely can be found as the number of outcomes that define a particular event over the total number of possible outcomes in a sample space.

Counting in R

In R, combinations can be calculated using the choose () function. The choose function calculates the combinatorial of the parameters it is given. Choose (n, r), with parameter n and r, calculates Cn,r. For example:

```
> #calculate number of combinations of
> #choosing 3 nucleotides from 4
> choose(4,3)
[1] 4
```

Introducing the Gamma function

The ! operator is the logical sign for negation in R and is not used for factorial calculations. This makes factorial calculations in R not so simple as coding "n!". Something else to use when calculating complicated counting tasks in R is the gamma function. The gamma function is a mathematical relationship defined by the following formula:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt \quad (x>0)$$

The Greek letter capital gamma is used in $\Gamma(x)$. Of course, in the formula written above this seems mathematically complex and exotic, and unless you're a big calculus fan you probably have no interest in evaluating it per se. However, the gamma function has a nice identity which can be evaluated for any a positive number n:

$$\Gamma(n) = (n-1)!$$

In other words, gamma of n is equal to (n-1) factorial. This can be handy in calculating permutations and combinations in R where you can use the function gamma (x) where x is the value of the factorial you want to calculate plus 1. For example, to calculate 4! which is equal to 4*3*2*1, you can use gamma (5) in R, as below:

Permission granted to reproduce for nonprofit, educational use.

```
> #gamma of n+1 calculates n!
> #to calculate 4! use gamma(5)
> gamma(5)
[1] 24
```

Gamma functions can be used together to perform any permutation or combinatorial counting procedure. For example, to calculate the numbers of unique 8-mer peptide arrangements taken from the 20 amino acids (order is important here so it is a permutation), simply use the formula translating the permutation into an equation of gamma functions as follows:

$$Pn,r = \frac{n!}{(n-r)!} = \frac{20!}{(20-8)!} = \frac{20!}{12!} = \frac{\Gamma(21)}{\Gamma(13)}$$

In R this can simply be calculated as:

```
> gamma(21)/gamma(13)
[1] 5079110400
```

Thus there are 5,079,110,400 possible permutations of 8 unique amino acids.

Although the gamma function may seem strange at first (and is not typically taught in lower-end mathematics courses), it is good to become familiar with the gamma function. The gamma function is involved in several continuous probability distributions (discussed later in this chapter) including the gamma distribution, the beta distribution and the Chi-Square distribution. Bayesian statistics makes use of these distributions and they are heavily used in bioinformatics applications.

Modeling Probability

Statistics in its mathematical formulation makes extensive use of models to represent relationships between variables. Models are written out in the form of equations. Perhaps the most familiar simple mathematical model is the linear relationship between a independent variable x and the dependent variable, y. In this case the model is:

$$y=mx + b$$

where m is the slope of the line and b is the y-intercept of the line. Another way to represent this is to consider y as a function of x, written f(x) so that:

$$f(x)=mx + b$$

The outcomes of an experiment are also modeled mathematically in a probability model. Probability models provide a way to structure and organize

Permission granted to reproduce for nonprofit, educational use.

the distribution of data that are the outcomes of an experiment. The next two chapters cover standard probability models commonly used in bioinformatics applications. The remainder of this chapter is devoted to several concepts important in understanding probability models: random variables, discrete versus continuous variables, probability distributions and densities, parameters, and univariate versus multivariate statistics.

Random Variables

Probability is based on observing random outcomes of an experiment. To model these outcomes using mathematical functions, we use variables called "random variables". Random variables assign a numerical value to each outcome of an experiment.

For example, consider the RNA sequence below:

AUGCUUCGAAUGCUGUAUGAUGUC

In this sequence there are 5 A's, 9 U's, 6 G's, and 4 C's with a total of 24 residues. To model this sequence, the random variable X can be used where X represents the nucleotide residues. Because there are advantages to working with quantitative information, when the data is described qualitatively a random variable is used to assign a number to the non-numerical outcomes. For this experiment let's assign the random variable values representing A as 0, C as 1, G as 2 and U as 3. A small letter represents the outcome of the random variable, so little x can be used here. So, in probability terms, the model represented using the random variable X for this experiment is given in Table 6-1.

Table 6-1: Using Random Variable X to Quantitatively Model Residues in a Particular RNA Sequence

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

If the experiment is to count the frequency of each nucleotide in another sequence of RNA, the values of the random variable will be the same but the probabilities of the random variable assuming that value will be slightly different reflecting the different trials of the experiment. Understanding this simple model (which does not even use an equation!) is key to understanding more complicated models. Probability models simply use random variables to represent the outcome of an experiment whether it is a simple experiment (as above) or a much more complicated experiment with many outcomes.

Permission granted to reproduce for nonprofit, educational use.

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 microarray 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.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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")
```

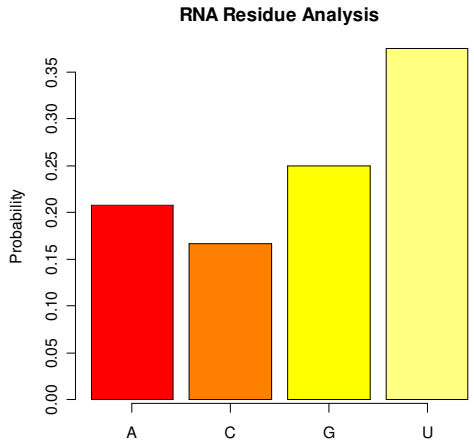


Figure 6-9: Histogram Illustrating the Probability Mass Function for RNA Residue Example

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

Permission granted to reproduce for nonprofit, educational use.

To visually analyze the CDF, a simple step graph of this CDF can be done in R by adding the commands below to the previous code. Figure 6-10 shows the plot produced.

```
>  CumProb<-c(0.208, 0.375, 0.625, 1)
> plot(X,CumProb,xlim=range(0,1,2,3,4), main="RNA Residue Analysis CDF",
xlab="X=", type="S")
```

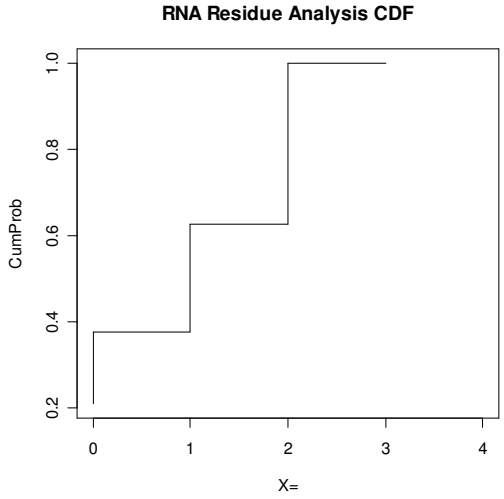


Figure 6-10: Cumulative Distribution Function for RNA Residue Analysis

Note the step shape of this CDF is characteristic of a discrete random variable and illustrates that the cumulative probability starts at 0 and stops at 1, thus obeying the basic laws of probability.

In the case of a continuous random variable, the model for how probability is distributed is called a probability density function and is denoted by f(x). For continuous values, probabilities are not assigned to specific discrete experimental outcomes but are instead assigned to intervals of values. There is no assigned probability that X=1 when X is a continuous random variable. Instead there is a probability assigned with an interval of values surrounding X, although that interval can be of values which are VERY close to X but not quite exactly X. Using a little calculus, the probability for a continuous random variable is:

$$P(a < x < b) = \int_a^b f(x)dx$$

This is also the "area under the curve" on the interval [a, b], which is part of the reason why the function is called a "density" function rather than a distribution function as in the discrete case. For a continuous random variable the CDF is just like for a discrete random variable except the graph will be continuous (not

Permission granted to reproduce for nonprofit, educational use.

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)
```

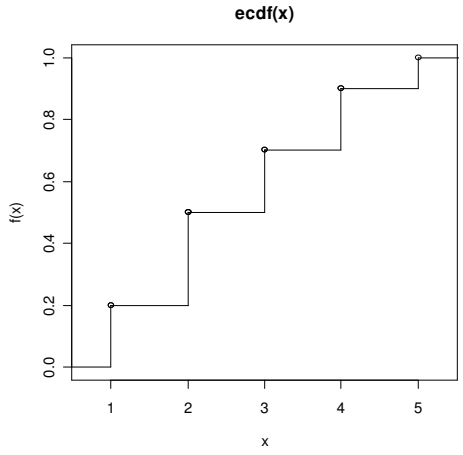


Figure 6-11: CDF Plot Example Produced Using Stepfun

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

Permission granted to reproduce for nonprofit, educational use.

mathematically frame how a probability model fits. Standard probability models use parameters extensively.

Parameters represent characteristics of the model they are used in and usually are classified as different types, such as shape, scale and location, discussed below in more detail. Some distributions have all three types of parameters, some have one and some have two. In some cases parameters serve hybrid functionality. Probability models (including those that we will introduce in the next two chapters) can be viewed as families of distributions that have certain mathematical forms defined by a function that includes which parameters they have. Changing the value of the parameter while utilizing the same mathematical model for the distribution will form different family members with distinctly appearing distributions. Fitting a distribution is an art and science of utmost importance in probability modeling. The idea is you want a distribution to fit your data model "just right" without a fit that is "overfit". Over fitting models is sometimes a problem in modern data mining methods because the models fit can be too specific to a particular data set to be of broader use.

Shape Parameters

To look at shape parameters, it is best to illustrate with an example. Figure 6-12 is an R generated plot of four probability density models, all of which are gamma densities (a family of continuous probability densities to be discussed later). The same data is used but modeled using different values for the shape parameter for this distribution (with all other parameters constant). Notice that changing the shape parameter changes how the model fits the data. Families of distributions such as the gamma family are particularly useful in modeling applications since they are flexible enough to model a variety of data sets by using different parameter values.

Permission granted to reproduce for nonprofit, educational use.

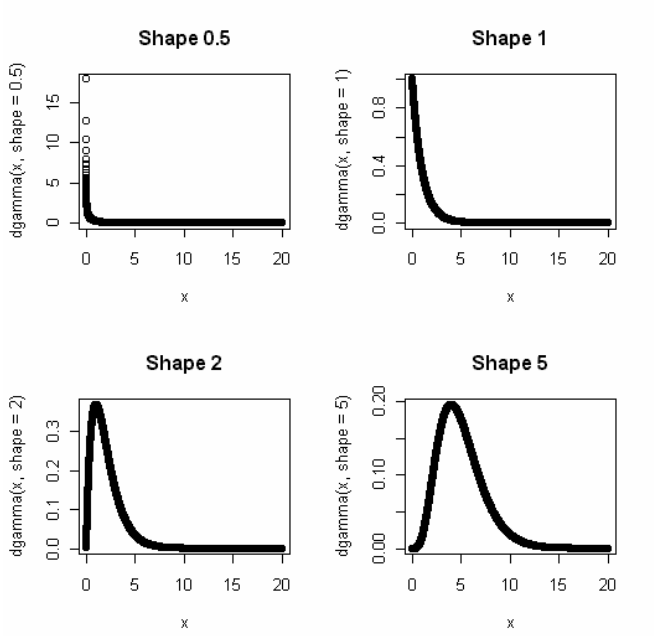


Figure 6-12: Altering a Shape Parameter with Other Parameters Held Constant

Scale Parameters

A second type of parameter is a scale parameter. Scale parameters do not change the shape of a distribution but change how spread out the distribution is. As an illustration, the plot in Figure 6-13 is also of the gamma family of distributions and uses the same data as the prior illustration. All plots in the figure use shape parameter 2 but use different scale parameters, as indicated on the plots. Although it may not initially appear so, the plots are the same shapes. Note that higher values for the scale parameter result in the shape of the distribution being increasingly spread out.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

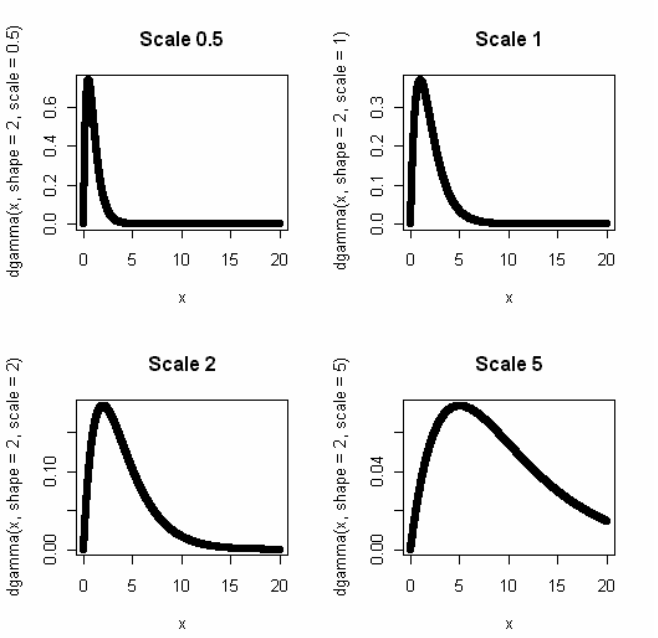


Figure 6-13: Altering a Scale Parameter with Other Parameters Held Constant

Location Parameters

The final type of parameter discussed here is the location parameter. The location parameter specifies where the graph lies along the horizontal (x) axis. Changing the location parameter translates the graph along the horizontal axis either to the left or to the right.

To illustrate this perhaps the easiest example uses the normal distribution, or the familiar bell curve. The normal distribution is a continuous probability density function that has some nice properties and is frequently used and especially well studied in most introductory statistics courses. One of those nice properties is that the mean of the distribution corresponds to the location parameter (and the standard deviation corresponds to the scale parameter). You may wonder that since the normal is such a nice model, why do we need all these other models? The answer is that in reality data are not often normally distributed and in probability many different types of models exist to model many different types of distributions of data.

Shifting the location parameter of the normal distribution shifts where the center of the distribution lies. This is illustrated in Figure 6-14 using two plots, one of which has mean (location parameter) 0 and the other mean (location parameter) 4. All other parameters are constant.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

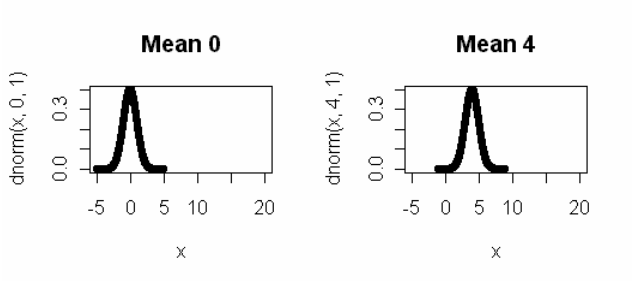


Figure 6-14: The Effect of changing the Location Parameter

Even before specific models are introduced you may be wondering how to choose a model to use and how to pick values of parameters. This can be a complicated and tricky question. As models are introduced it will become more apparent which models are used for which types of data. The determination of parameter values is done in different ways. Sometimes the parameters can be determined from the data, as is the case of the mean and standard deviation of normally distributed data. Sometimes R can be used as a tool to help in selecting parameter values for a distribution by computing parameters or by graphically looking at the data. Other times more sophisticated statistical methods are used, the techniques of which are beyond the scope of this book and consultation with a statistician is advised.

Univariate versus Multivariate Statistics

The models introduced in the next are all univariate models, meaning that there is only one random variable on which data is measured and one outcome probability measure associated with that random variable in the model. Often many random variables are measured at the same time. To model how multiple random variables affect an outcome of an experiment a more complicated branch of statistics, multivariate statistics, is used. To compare these, suppose you are examining the annealing temperature of a DNA PCR primer. Univariate statistics would look how the percentage of one nucleotide affected annealing temperature whereas multivariate statistics would take into consideration how the composition of all four nucleotides and perhaps environmental conditions such as salt concentration affected annealing temperature and other factors. Chapter 7 introduces some standard univariate models, and select multivariate models are introduced in Chapter 8.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

7

Univariate Distributions

Different random variables have different probability distributions. Technically there are infinitely many possible probability distributions, but some common forms appear over and over again in practical applications. This chapter discusses some of the common discrete and continuous univariate distributions commonly often encountered in modeling data in biological applications.

Univariate Discrete Distributions

Univariate discrete distributions are standard probability models that utilize a discrete random variable to define the outcomes of an experiment. Presented here are two models frequently used in analyzing biological data: the binomial model and the related Poisson model.

The Binomial Distribution

The foundation for the binomial distribution is the Bernoulli random variable. A Bernoulli random variable arises in an experiment where there are only two outcomes, generally referred to as "success" and "failure". For the success outcome the value of the random variable is assigned the value 1, and for the failure outcome the value of the random variable is assigned the value 0. The probability of success is a value p, a proportion between 0 and 1. The probability of failure (using the law that probability adds to 1 and that the complement probability is 1-probablity of all other events) is 1-p.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

For a one trial experiment the probability distribution function for a Bernoulli experiment is trivial and the distribution of a Bernoulli random variable can be written as follows:

$$p(x) = p^x (1-p)^{1-x}$$

where x can take either of the value 0 or 1.

Let's consider the case of having a child and use a Bernoulli random variable to represent whether the child has blue eyes. Let's assume the probability of the child having blue eyes is 0.16 (not empirically verified!) and this is the "success" outcome. For this experiment the distribution of the random Bernoulli variable X is given in Table 7-1.

Table 7-1: Distribution of outcomes of a Bernoulli Trial

Outcome	**Random Variable X =x**	**P(X=x)**	**Probability of outcome**
Blue eyes	1	p	0.16
Not blue eyes	0	1-p	0.84

What about if you really want a blue-eyed child, so you have 10 children and you want to know the probability that k out of the 10 have blue eyes? This is a more complicated question. Each outcome of having a child is independent of other children – meaning whether the first child had blue eyes has no statistical influence on the second child having blue eyes. Independence will be discussed in more detail in the next chapter.

In order to answer this you want to create a model for how many of 10 children will have blue eyes based on the probability that a given child has blue eyes is 0.16. Such a model is called a binomial model. Using the Bernoulli probability distribution function equation above we can extend it to work for more than one trial by changing the exponents to n=the number of trials and k=the number of successes as follows:

$$p^k (1-p)^{n-k}$$

Note this is not a probability distribution function anymore, as it will only model the probability of a particular sequence of n=10 children k of which have blue eyes. For example if k=3, n=10, the above represents the probability of a sequence such as {1,0,0,0,1,1,0,0,0,0}, where as indicated 1 denotes "blue eyes" and 0 denotes "not blue eyes". But when there are 10 children, the blue-eyed children can be any one of the 10 children. Using the counting method of combinations discussed in the previous chapter (formula below) we note that the number of such sequences with k ones and n-k zeros is

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

$$Cn,k = \frac{n!}{k!(n-k)!}$$

Remember that for a series of 10 children, one blue-eyed child could be positioned in 10 different ways (the blue eyed child could be first of 10, second of 10, etc.), corresponding to a combination of

$$C_{10,1} = \frac{10!}{1(9)!} = 10$$

In order to model the correct probability of observing 1 child of 10 children with blue eyes, the probability distribution function needs to account for the 10 different arrangements of children, so the proper way to write the probability distribution is

$$p(k) = P(k \text{ successes in n trials}) = \binom{n}{k} p^k (1-p)^{n-k}$$

Note $\binom{n}{k} = Cn,k = \frac{n!}{k!(n-k)!}$ is another popular notation used for the number of combinations of k out of n distinct objects. This symbol is commonly described as "n choose k" and is also called the "binomial coefficient" in some contexts.

Of course to make this a distribution function we want to calculate not only the probability of having 1 in 10 children with blue eyes, but the whole distribution of how many kids (0,1,2…10) in 10 will have blue eyes. This is tedious to do by hand and hence using R comes in handy.

In R the binomial distribution is the function dbinom. In R, all probability distributions (or densities in the case of continuous random variables) use the letter d as the first letter in the function and then part or the entire name of the distribution for the rest of the function name. dbinom takes as parameter arguments binom(x, size, prob) where x= the vector of k values to be used, size is the total number of trials (n), and prob is the probability of success on each trial.

Using some simple commands in R to generate the probability values for the binomial distribution for the number of children in 10 with blue eyes using p =0.16

```
> x<-0:10
> y<-dbinom(0:10,10,0.16)
> data.frame("Prob"=y,row.names=x)
```


Permission granted to reproduce for nonprofit, educational use.

we obtain the following:

```
Prob
0   1.749012e-01
1   3.331452e-01
2   2.855530e-01
3   1.450428e-01
4   4.834760e-02
5   1.105088e-02
6   1.754108e-03
7   1.909233e-04
8   1.363738e-05
9   5.772436e-07
10  1.099512e-08
```

Thus given p=0.16, the probability of 0 in 10 children with blue eyes is 0.175; the probability of one with blue eyes child is 0.333 and so forth.

Of course, writing out tables of probability as above is only practical for simple scenarios and in most cases a graphical model for the probability distribution will be used. To get a graphical model in R for the same distribution above, simply use the plot command and put the binomial function call right in the plot function call as follows:

```
> plot(0:10,dbinom(0:10,10,0.16),,type='h',xlab="",ylab="Probability",
sub="Number of kids with blue eyes")
```

Figure 7-1 illustrates the graphic model of the probability distribution function for this example.

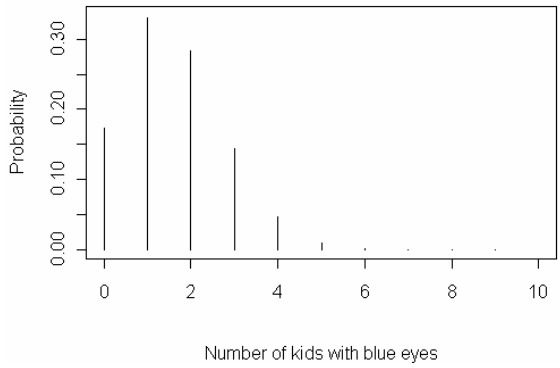


Figure 7-1: Example of a Binomial Distribution, p=0.16

Note that this distribution is pretty skewed toward lower values of the random variable (X=number of kids with blue eyes) because the value of p is 0.16. The graph should seem reasonable given the value of p. What if the value of p is changed?

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Let's re-run this example with probabilities that a child has blue eyes is 0.05, 0.2, 0.5, and 0.8 and see how this changes the distribution.

```
> par(mfrow=c(2,2))
> plot(0:10,dbinom(0:10,10,0.05),type='h',xlab="",ylab="Prob", sub="p=0.05")
> plot(0:10,dbinom(0:10,10,0.2),type='h',xlab="",ylab="Prob", sub="p=0.2")
> plot(0:10,dbinom(0:10,10,0.5),type='h',xlab="",ylab="Prob", sub="p=0.5")
> plot(0:10,dbinom(0:10,10,0.8),type='h',xlab="",ylab="Prob", sub="p=0.8")
```

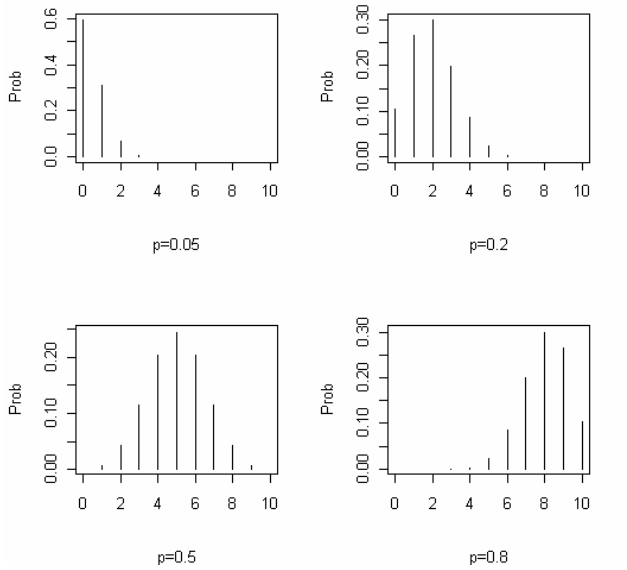


Figure 7-2: Illustrating the Effect of Changing the Value of p in the Binomial Distribution

Notice in Figure 7-2 how the larger p shifts the distribution more toward higher values of the random variable. This should make sense because a higher p makes it more likely a child will have blue eyes and therefore more children in 10 will have blue eyes, as represented by the shift of the graphical models with higher p. Note also for p=0.5 that the distribution is symmetric. This is always the case for a binomial distribution with p=0.5 since it equally likely that success or failure occurs.

So far we have only considered the probability distribution function for the binomial, but what about the cumulative distribution function? Recall that this is the function which models the total probability up to and including a certain value of the random variable X=x.

This is easy to do in R using the pbinom distribution function, which takes the same parameters as the dbinom. In fact we can use the same code as above to get plots of the CDF of the binomial for the example above changing only the

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

type of the plot to 's' for step and change the function used from dbinom to pbinom:

```
> par(mfrow=c(2,2))
> plot(0:10,pbinom(0:10,10,0.05),type='s',xlab="",ylab="Prob", sub="p=0.05")
> plot(0:10,pbinom(0:10,10,0.2),type='s',xlab="",ylab="Prob",sub="p=0.2")
> plot(0:10,pbinom(0:10,10,0.5),type='s',xlab="",ylab="Prob",sub="p=0.5")
> plot(0:10,pbinom(0:10,10,0.8),type='s',xlab="",ylab="Prob",sub="p=0.8")
```

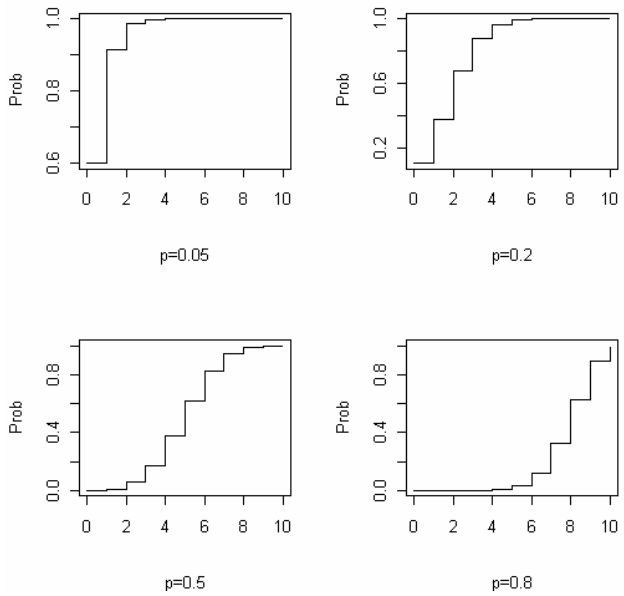


Figure 7-3: Binomial CDFs Using Different Values of p

The pattern of cumulative probability for binomials produced using different values of p is illustrated in Figure 7-3. When p is small (as in 0.05) the cumulative probability reaches 1 quickly whereas a large value of p results in the cumulative probability not reaching 1 until the higher range of values for the random variable.

If all you need is a simple calculation for one value in R all you need to do is enter the appropriate function and relevant parameter values. For example, suppose you want to know the probability that (exactly) 4 kids in 10 will have blue eyes when p=0.5. Simply use the dbinom function in R as follows and it calculates this value for you:

```
> dbinom(4,10,0.5)
[1] 0.2050781
```

Thus, the chance of 4 in 10 kids with blue eyes is 0.205 or 20.5% with p=0.5, which should make sense based on earlier graphical models.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The binomial distribution is an important model and one of the simplest to understand. Several other distributions are related to the binomial and also based on Bernoulli random variables (success or failure experiments). The geometric distribution (dgeom, pgeom in R) considers the random variable X as the number of failures before the first success. The negative binomial distribution (dnbinom, pnbinom in R) considers X as a measure of the number of failures before the r^{th} success. A multivariate version of the binomial, the multinomial distribution, will be introduced in the next chapter. The example used with the binomial could easily have been modeled using one of the related distributions. For example, the geometric could be used to model the number of children born before the first child with blue eyes. In many cases when you have data to model you have some choices how to model it based on choice of random variable measurement outcome and choice of distribution used.

The Poisson Distribution

The next discrete univariate distribution to be introduced is called the Poisson distribution, named after Simeon D. Poisson. The Poisson is one of the most utilized discrete probability distributions. Mathematically the Poisson is actually a limiting case of the binomial, the details of which will not be dealt with here but can be found in most mathematical probability books.

The Poisson has many applications, including numerous applications in biology. In general, the Poisson is used to model the counts of events occurring randomly in space or time. Simple real world examples which may use a Poisson model include the number of typing errors on a page, the number of accidents occurring at an intersection in a given time period and the number of discarded products made by a manufacturing process. In bioinformatics some examples where the Poisson could be used include: to model the instances of mutation or recombination in a genetic sequence, the distribution of errors produced in a sequencing process, the probability of random sequence matches, or in counting occurrences of rare DNA patterns.

The mathematical formula for the Poisson distribution is:

$$p(X = x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

Here x represents counts and thus can be any integer value ≥ 0. Note also that in this equation, there is one parameter, the Greek letter lambda λ that defines the distribution. In a random process (such as mutation) there will be lambda events per unit time interval. Thus, lambda represents a rate. Because of the relation of the Poisson to the binomial, lambda can be obtained by the relationship $\lambda = n * p$ where p is the probability of the event occurring and n the number of trials. The relation holds when p is small (rare events) and the number of trials n are large.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Suppose we are using a new sequencing technique and the error rate is one mistake per 10,000 base pairs. Suppose we are sequencing 2000 base pair regions at a time. What is the probability of 0 mistakes using this technique? Of 1 mistake, 4 mistakes? The Poisson model can be used to model these probabilities. To use the Poisson, you must first calculate lambda. In this case the "trial" can be considered an individual base pair, so n=2000 trials for a 2000 base pair sequence. The probability of "success" here is the probability of getting an error, where p = 1/10,000. To calculate lambda, multiply n*p, or 2000*(10,000) which results in a rate of 0.2 mistakes per 2000 base pairs so lambda is 0.2. Note it is common to adjust lambda based on n. If we were using 5000 base pair regions to sequence at a time we would use a lambda of 0.5.

To calculate the probability of one mistake in the 2000 base pair sequence, we could do this by hand with the following equation:

$$p(X = 1) = \frac{e^{-0.2}0.2^{1}}{1!} = 0.1637$$

However, we are interested in the whole distribution of probability values for the random variable X = number of mistakes in the sequence and it is much easier to computer these in R than doing individual hand calculations. In R the dpois function is used to compute Poisson distributions, and has parameters dpois (x, lambda) where x is the value or vector of values of the random variable to be calculated and lambda is the parameter.

As we did with the binomial, first let's generate a simple table of probabilities that X=x for the values of this distribution. We have a little bit of a problem in that in this case, theoretically X can be anywhere from 0 (no sequence errors) to 2000 (every bp an error). However, knowing lambda is 0.2 (also the mean or expected number of errors) the number of sequence errors is not likely to exceed 10, so the following code is used to generate the table:

```
> x<-0:10
> y<-dpois(0:10,0.2)
> data.frame("Prob"=y,row.names=x)
```

which produces the following results:

```
Prob
0   8.187308e-01
1   1.637462e-01
2   1.637462e-02
3   1.091641e-03
4   5.458205e-05
5   2.183282e-06
6   7.277607e-08
7   2.079316e-09
8   5.198290e-11
9   1.155176e-12
10  2.310351e-14
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The chance of no errors (X=0) is 0.818, the chance of 1 error (X=1) is 0.163, etc. in a 2000 base pair sequence. As expected, even at a value as low as 10 there is virtually no probability left. This can be viewed graphically as illustrated in Figure 7-4.

```
> plot(0:10, dpois(0:10,0.2), type='h', xlab="Sequence Errors",
ylab="Probability" )
```

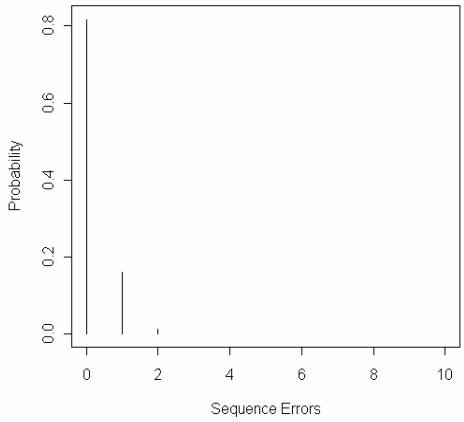


Figure 7-4: Poisson Distribution, Lambda =0.2.

The cumulative distribution in this case may be a bit more interesting. The CDF for the Poisson uses the ppois function call with the same parameters as dpois discussed above.

```
> plot(0:10,ppois(0:10,0.2),xlab="# Seq Errors", ylab="Cum Prob", type='s')
```

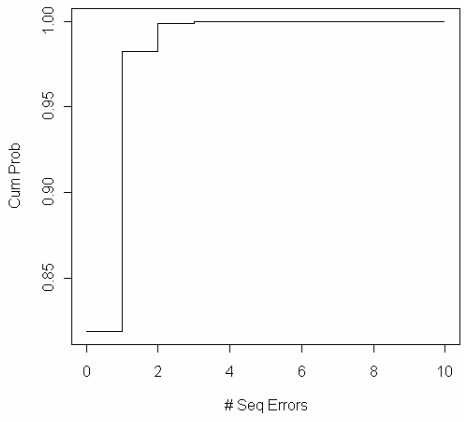


Figure 7-5: CDF for Poisson Model of Sequencing Errors with Lambda=0.2.

Permission granted to reproduce for nonprofit, educational use.

As is clear from the CDF graph in Figure 7-5, the number of sequence errors using this method in a 2000 base pair sequence is highly unlikely to be more than 3. This should leave you pretty confident the process is not going to produce a lot of errors (unless you are looking for more stringent reliability). Below R is used to find the probability of 1 or fewer, 2 or fewer and 3 or fewer errors in this example.

```
> ppois(1,0.2)
[1] 0.982477
> ppois(2,0.2)
[1] 0.9988515
> ppois(3,0.2)
[1] 0.9999432
```

What happens to the Poisson probability distribution when the parameter is changed? To examine this, let's look at a few examples with different values for lambda:

```
> par(mfrow=c(2,2))
> plot(0:10,dpois(0:10,0.5),xlab="",ylab="Prob",type='h',main="Lambda 0.5")
> plot(0:10,dpois(0:10,1),xlab="",ylab="Prob",type='h',main="Lambda 1")
> plot(0:10,dpois(0:10,2),xlab="",ylab="Prob",type='h',main="Lambda 2")
> plot(0:10,dpois(0:10,5),xlab="",ylab="Prob",type='h',main="Lambda 5")
```

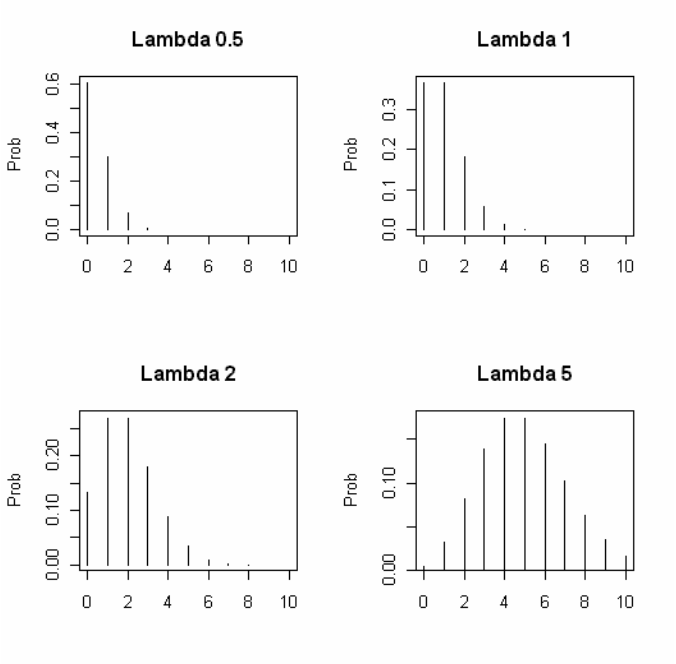


Figure 7-6: Poisson Distributions with Different Lambda Values

Not surprisingly the way the Poisson distribution changes (Figure 7-6) when lambda changes looks a lot like the way the binomial changes when p changes. Considering the relationship λ=n*p this should come as no surprise. Remember

Permission granted to reproduce for nonprofit, educational use.

that the plots above only consider X=x for the range of [0, 10] so in the case of λ=5 there is more of the distribution shifted to the right. To see where the probability levels off to 1 similar analysis can be done looking at the cumulative distributions (ppois for the Poisson) as was done with the binomial. Let's do this with the range X=x from 0 to 20 for all the values of lambda used in the previous plot.

```
> par(mfrow=c(2,2))
> plot(0:20,ppois(0:20,0.5),xlab="",ylab="Cum Prob", type='h', main="Lambda
0.5")
> plot(0:20,ppois(0:20,1),xlab="",ylab="Cum Prob", type='h',main="Lambda 1")
> plot(0:20,ppois(0:20,2),xlab="",ylab="Cum Prob", type='h',main="Lambda 2")
> plot(0:20,ppois(0:20,5),xlab="",ylab="Cum PRob", type='h',main="Lambda 5")
```

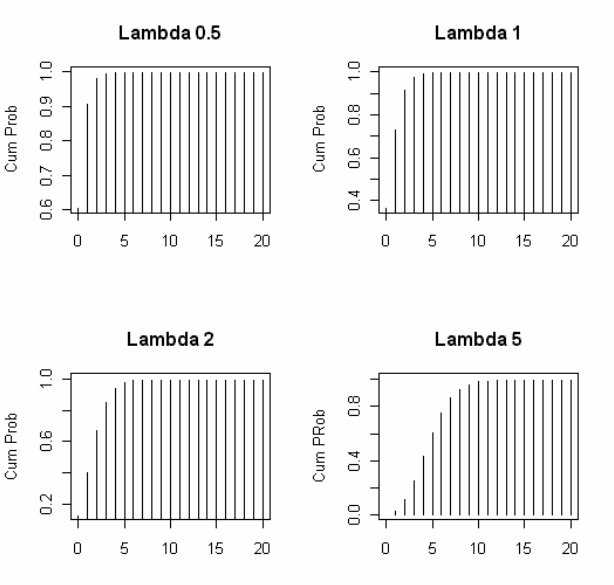


Figure 7-7: Poisson CDFs with Different Lambda Values

For the lambda values 2 or less it is pretty clear from the CDF plots (Figure 7-7) that it is unlikely more than 10 of 2000 base pairs would contain errors. Be careful though– although the graph for lambda=5 appear to level off at 1 around x=10 there is still some significant probability of obtaining a value of X higher than 10, which can be analyzed by doing some additional calculations in R as is done below:

```
> ppois(10,5)
[1] 0.9863047
> ppois(12,5)
[1] 0.9979811
> ppois(15,5)
[1] 0.999931
> ppois(20,5)
[1] 1
```


Permission granted to reproduce for nonprofit, educational use.

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:

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

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:

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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

R contains functionality for both the probability density function and cumulative distribution function for the normal model in the base package. Like previous distributions discussed, the command for the probability density function starts with "d" and is named "dnorm". The parameters of dnorm are the data vector of x's, the mean and standard deviation. As an example, let's plot a normal density for a range of x from –10 to 10 with mean 0 and standard deviation 1:

```
> x<-seq(-10,10,length=100)
> plot(x,dnorm(x,0,1),xlab="x", ylab="f(x)", type='l', main="Normal PDF")
```

This produces a nice bell shaped PDF plot depicted in Figure 7-8

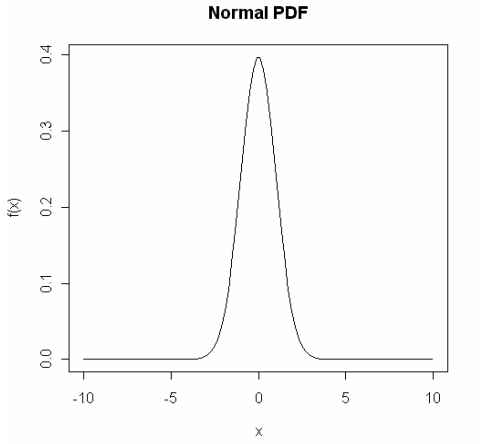


Figure 7-8: Univariate Standard Normal Distribution

In discussing location parameters in the previous chapter, we looked at an example of changing the location parameter for the normal. But what happens if we change the scale parameter? Remember that increasing the scale parameter, which is the standard deviation in the case of the normal, increases how spread out the data are. To look at this in R, simply change the standard deviation parameter of dnorm:

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

The change in the scale parameter from 2 to 5 is illustrated in Figure 7-9.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

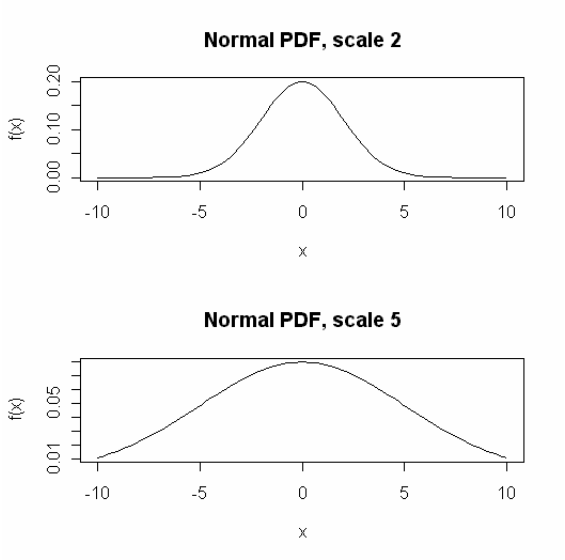


Figure 7-9: Changing the scale parameter (standard deviation) in the normal distribution

The normal also has a cumulative density function, which in R utilizes the pnorm function with the same parameters as dnorm. 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")
```


Permission granted to reproduce for nonprofit, educational use.

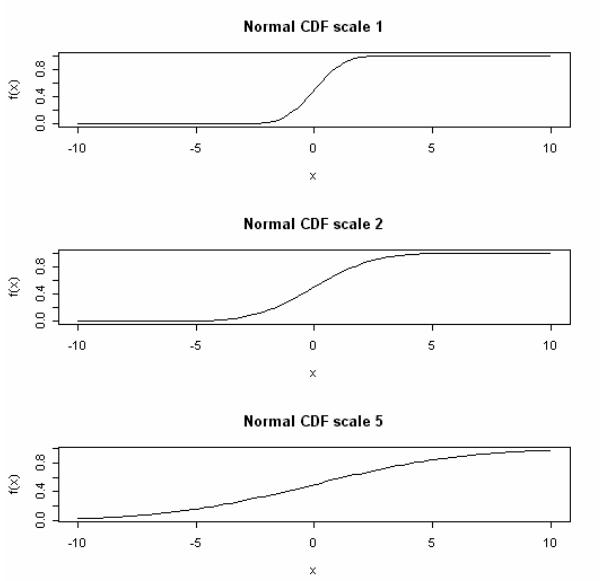


Figure 7-10: Normal Distribution CDFs with Different Scale Parameters

Notice in Figure 7-10 how increasing the scale parameter (standard deviation) causes the CDF to increase much less precipitously, reflecting the less concentrated (more spread out) data distribution from the probability density.

In inferential statistics, the normal is frequently used to answer questions with regard to test scores and other such measures. For example suppose you take a test and score 85, which sounds great until the professor announces the mean score is 92 with a standard deviation of 8. Then you begin to wonder, how badly did you do? To answer this use the pnorm function in R as follows:

```
> pnorm(85, mean=92,sd=8)
[1] 0.1907870
```

This means you scored better than 19.1% of the class. You could have answered this question looking at the other end of the distribution as well using 1-pnorm.

```
> 1-pnorm(85, mean=92, sd=8)
[1] 0.809213
```

This means that 80.9% of the class scored better than you (better luck next time). Knowing the mean and standard deviation of a normal distribution makes such calculations easy. And if all you have is a data set that you are assuming is normally distributed, you can enter your data, and then use R to find the mean and standard deviation. For example, suppose you chop up a piece of DNA with an enzyme and get 20 fragments with sizes you measure on a gel (to the nearest base pair) and you guess from the gel pattern that the size of the fragments is normally distributed. You can record (in data vector x) and analyze your data in R as follows:

Permission granted to reproduce for nonprofit, educational use.

```
> x<-c(321, 275, 345, 347, 297, 309, 312, 371, 330, 295, 299, 365,
+ 378, 387, 295, 322, 292, 270, 321, 277)
> mean(x)
[1] 320.4
> sd(x)
[1] 35.16787
```

And now, knowing the mean and standard deviation, you have a probability density function normal model for this data that you can use to perform further statistical tests on. It is very simple to graph this model and its CDF function using the code below, with results depicted in Figure 7-11.

```
> par(mfrow=c(1,2))
> xx <- seq(200,450,by=.25)
> plot(xx,dnorm(xx,mean(x),sd(x)),type='l',xlab="Frag Size in bp",
ylab="f(x)",
+ main="Restriction Fragments PDF")
> points(x,rep(0,n))
> plot(xx,pnorm(xx,mean(x),sd(x)),type='l',xlab="Frag Size in bp",
+ ylab="Cum Prob", main="Restriction Fragment CDF")
> points(x,rep(0,n))
```

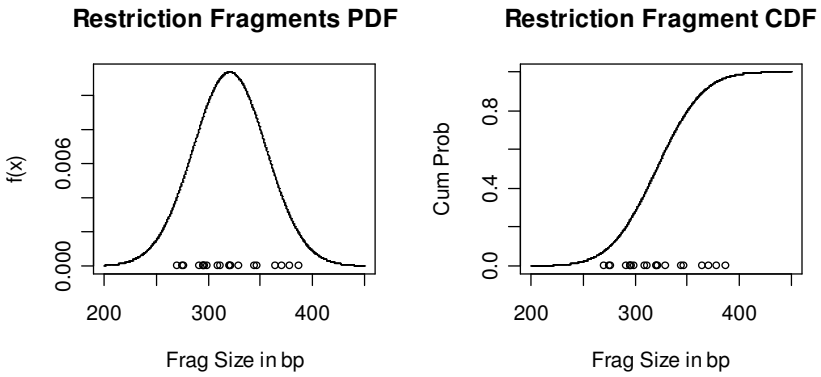


Figure 7-11: PDF and CDF for restriction fragment data

In addition, you now have a model you can make inferences on. For example, suppose you want to know the probability of getting a fragment bigger than 400 bp. A simply query R computes this.

```
> 1-pnorm(400,mean(x),sd(x))
[1] 0.01180460
```

Based on the above result, there is a 1.18% chance of getting a fragment this big from the given distribution.

Another useful feature in R is that all distributions have an associated quantile function built in, designated by a q before the distribution name code (qnorm, qbinom, etc). This automatically calculates what percentage of the distribution corresponds to the given cumulative probability regions. For example, suppose

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

for the distribution above you want to calculate the 10^{th} and 90^{th} percentiles of the distribution. Simply use the qnorm function in R to do this.

```
> qnorm(0.10,mean(x),sd(x))
[1] 275.3306
> qnorm(0.90,mean(x),sd(x))
[1] 365.4694
```

This means that 10% of the data is 275 or fewer base pairs in size, and 90% of the data is 365 or fewer base pairs in size, given this distribution. Quantiles are called quartiles for the 25%, 50% and 75% regions and also called percentiles on standardized tests. The quantile function can be helpful for analyzing data in a distribution.

Even though the normal is widely taught in statistics courses, and widely used in many areas of statistics, it is not the only continuous probability distribution to be familiar with for effective data analysis. Many times, especially when dealing with physical phenomena (as opposed to humanly generated measurable data such as that from standardized school tests) the data will not be normally distributed. For non-normally distributed continuous data modeling we now turn to two important families of distributions, the gamma family and the beta family.

The Gamma Family

The gamma family consists of a few related distributions including the gamma distribution, the exponential distribution and the Chi-Square distribution. The base distribution of the family is the gamma distribution, which provides a versatile model for working with continuous data that may not be normally distributed. Popular applications of the gamma distribution are to measurements of time until failure, concentrations of pollutants, etc. The gamma distribution is only defined for positive real numbers and it takes different forms depending on the parameter values. The probability density of the gamma distribution has the following general form:

$$f(x) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta}$$

Before you scream and shriek and give up, thinking you cannot possibly understand this crazy equation, relax and realize it's only a mathematical model for a probability density function. Long gone are the days when anyone would hand calculate f(x) values for this equation because computer packages such as R are very happy to do the calculations for us. The only things in the equation besides the familiar mathematical terms "x" and "e" are two parameters – alpha (designated by the Greek letter α) and beta (designated by the Greek letter β) and the gamma function (introduced in the pervious chapter) where the gamma

Permission granted to reproduce for nonprofit, educational use.

function of the parameter alpha is part of the equation. For the gamma distribution alpha is the shape parameter and beta is the scale parameter.

Like all the other distributions, the probability distribution for the gamma is simple to use in R. The dgamma function call takes as parameters the data vector and then the shape and scale parameters.

First, let's make a few graphs of changing the shape parameter, alpha, while keeping the scale parameter, beta, is constant at 1:

```
> x<-seq(0,10,length=100)
> par(mfrow=c(2,2))
> plot(x,dgamma(x,shape=1,scale=1), type='l',xlab="x",
ylab="Prob",main="Shape 1")
> plot(x,dgamma(x,shape=2,scale=1), type='l',xlab="x",
ylab="Prob",main="Shape 2")
> plot(x,dgamma(x,shape=5,scale=1), type='l',xlab="x",
ylab="Prob",main="Shape 5")
> plot(x,dgamma(x,shape=10,scale=1), type='l',xlab="x",
ylab="Prob",main="Shape 10")
```

Note in Figure 7-12 that for shape=10 the distribution shifts toward the higher end (and the graph doesn't depict the entire distribution only the same x range as the other graphs for comparison).

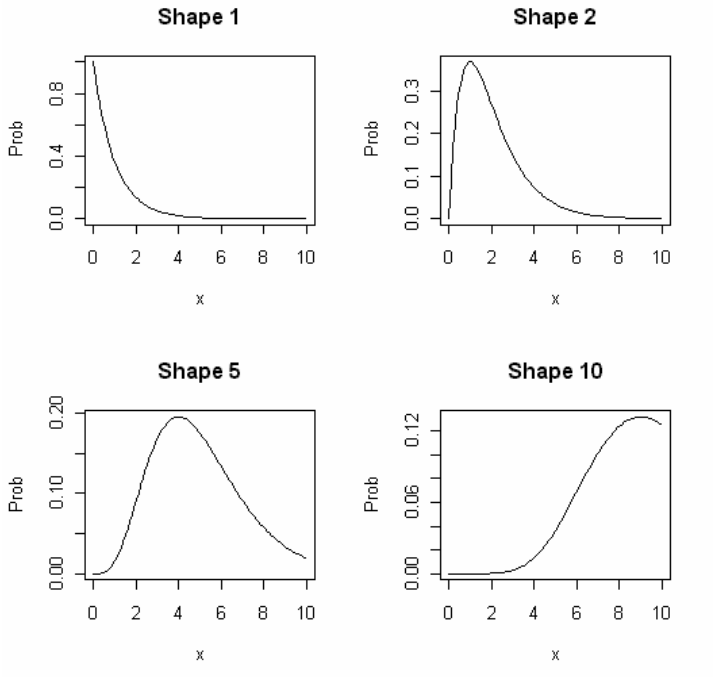


Figure 7-12: Gamma distributions with different shape parameters

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Next, let's look at how the gamma changes when the shape parameter, alpha, is held constant (at 2) and the scale parameter, beta, is changed.

```
> x <- seq(0,30,length=100)
> plot(x,dgamma(x,shape=2,scale=1), type='l',xlab="x", ylab="f(x)",
+ main="Gamma pdf's")
> lines(x,dgamma(x,shape=2,scale=2),lty=2)
> lines(x,dgamma(x,shape=2,scale=4),lty=3)
> lines(x,dgamma(x,shape=2,scale=8),lty=4)
> legend(x=10,y=.3,paste("Scale=",c(1,2,4,8)),lty=1:4)
```

Note that, although they might not look it, all of the graphs in Figure 7-13 are the same shape. The higher scale parameter values just spread the distribution out. For the higher values the distribution extends beyond the x range shown in the graph (but for comparison all the graphs are on the same x scale range).

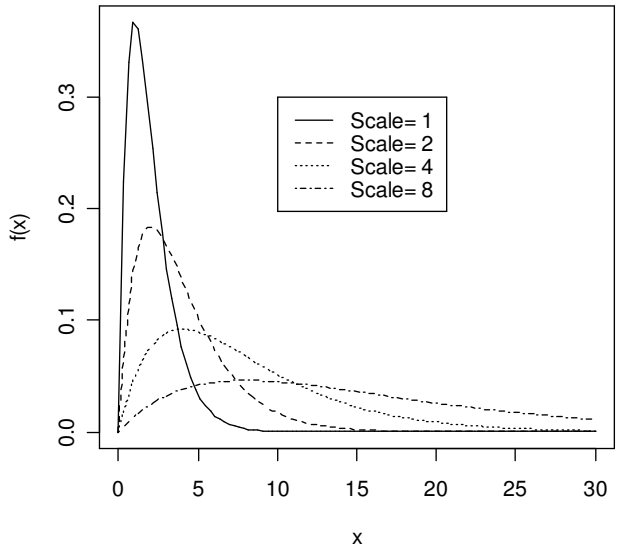


Figure 7-13: Gamma distributions with different scale parameters

You may be wondering how the gamma distribution is used to model the distribution of a set of data.

Suppose you are measuring survival times of an enzyme in a solution (as measured by some kind of assay for enzyme activity) and you get the following data in hours: 4.75, 3.4, 1.8, 2.9, 2.2, 2.4, 5.8, 2.6, 2.4, and 5.25. How could you decide on a probability model to model the probability of the enzyme surviving in solution?

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Because you know you cannot always assume data is normally distributed (although you often hope so) the first thing to do is to look at a plot of the data. There is actually a statistician's secret tool to check whether data are normally distributed. It is a plot called a Q-Q plot and what it does is line quantiles of the data against normal quantiles. If the line is a straight line, the data can be considered normally distributed and you can use the normal probability model.

All you have to do to run a Q-Q plot in R is enter the data and use the qqnorm and qqline functions.

```
> x<-c(4.75, 3.4, 1.8, 2.9, 2.2, 2.4, 5.8, 2.6, 2.4, 5.25)
> qqnorm(x)
> qqline(x)
```

Running this code produces the Q-Q plot in Figure 7-14.

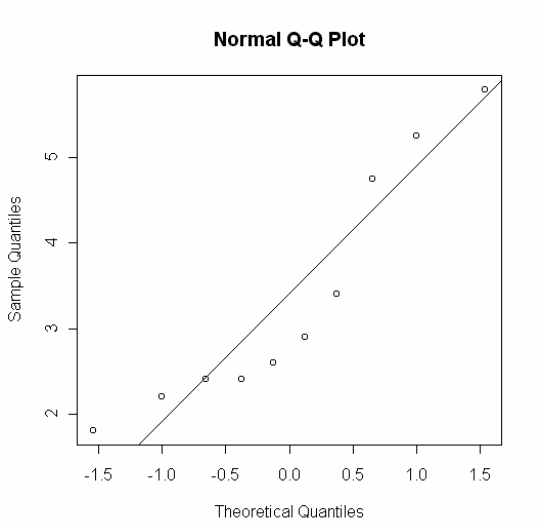


Figure 7-14: Q-Q plot of enzyme data

And you know, by looking at the squiggle pattern of the data that it does not align nicely with the expected normal line. When you see such a nonlinear data pattern on a Q-Q plot you should train yourself to think that you'd probably better find a model other than the normal distribution to model the distribution of this data.

Knowing about the flexibility of the gamma distribution, you suspect there may be a gamma distribution model for this data. But how do you determine the alpha (shape) and beta (scale) parameters you need in order to use the gamma model?

Actually it's not too hard. Although the scale and location parameters for the gamma model are not equal to the standard deviation and mean like in the normal case, there is a mathematical relationship between the mean and standard

Permission granted to reproduce for nonprofit, educational use.

deviation and the scale (beta) and shape (alpha) parameters for the gamma distribution.

The mean of the gamma distribution is equal to alpha * beta and the variance (=standard deviation squared) is related to alpha and beta by being equal to alpha*beta^2. In R it is always easy to get the mean and variance (or sd) of a data vector:

```
> mean(x)
[1] 3.35
> var(x)
[1] 1.985556
```

Let's be lazy and just call the variance =2. But based on this we know that:

$$\text{Mean}=3.35=\alpha\beta$$

and

$$\text{Var} = 2 =\alpha\beta^2$$

Doing a little algebra:

$$3.35/\beta=\alpha$$

and then substituting this into the variance equation for alpha allows you to solve for beta:

$$3.35*\beta=2, \text{ so } \beta=0.6 \text{ (roughly)}$$

and subsequently, you can solve for alpha

$$3.35=\alpha\ (0.6) \text{ so alpha} = 5.6$$

So the distribution of the data can be modeled using a gamma probability density function with shape (alpha) = 5.6 and scale (beta)=0.6. Note that because we don't have many data points (only 10) this might not be the best possible fit, but since we only have such a limited amount of data it's impossible to assess the goodness of fit (collecting more data values would be better of course). Also note that alpha and beta need not be integer values, allowing even greater flexibility in how the gamma model fits data. However, there is a requirement that both alpha and beta be positive values (so if you do the algebra and get negative values for alpha and beta, you did something wrong).

Let's look at a graphical model of the data and a dgamma plot using the parameters determined above:

Permission granted to reproduce for nonprofit, educational use.

```
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))
```

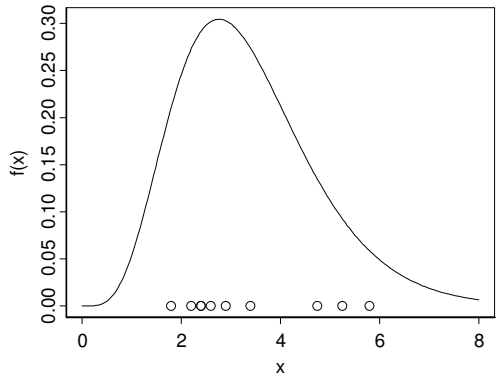


Figure 7-15: Comparing the data distribution to the gamma fit

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))
```

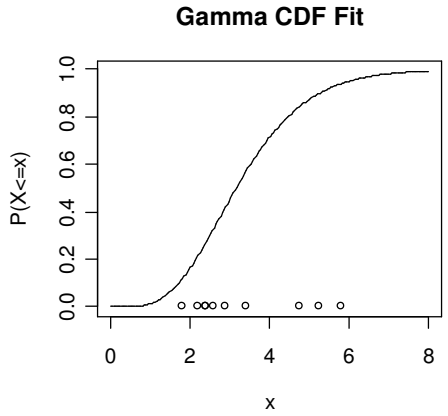


Figure 7-16:Gamma CDF

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.

Permission granted to reproduce for nonprofit, educational use.

```
> 1-pgamma(5,shape=5.6,scale=0.6)
[1] 0.1263511
```

Thus, there 12.6% of the density has values greater than 5 and you may want to re do the plots to reflect this (an exercise left to the reader).

The Exponential Distribution

The exponential distribution, famous for modeling survival times (as in the case with radioactive decay), is just a special case of the gamma distribution where the shape parameter, alpha = 1. This reduces the mathematical formula for the gamma to:

$$f(x) = \frac{1}{\beta} e^{-x/\beta}, \; x > 0$$

The exponential is often written in terms of a rate parameter lambda where $\lambda = 1/\beta$, or

$$f(x) = \lambda e^{-\lambda x}, \; x > 0$$

Most students of science have seen this form for radioactive decay rates, or survival rates of bacteria or something of that sort. Although details will not be discussed here, the R functions dexp (x, rate =) is used with the rate parameter lambda value to model the probability density, and the function pexp is used to model the CDF.

The Chi Square Distribution

The Chi-Square distribution is another gamma distribution variant, and the term "Chi-Square" should be familiar to genetics students for its role in analyzing count data and other applications. The Chi-Square distribution always uses a value of beta=2 for the scale parameter and a value of alpha=k/2 for the shape parameter where k is the number of "degrees of freedom". The Chi-Square distribution and concept of "degrees of freedom" will be returned to in later chapters, but is noted here to show its relationship to the gamma distribution. In R the probability density for this distribution is denoted as dchisq, and the cumulative density is pchisq. Both take as parameters the data vector and the degrees of freedom.

The Beta Family

Like the gamma family, the beta family is group of distributions that use alpha and beta parameters. The beta family has the following mathematically formula:

Permission granted to reproduce for nonprofit, educational use.

$$f(x) = \frac{1}{\beta(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1},\ 0<x<1$$

And you ask, what is that $\beta(\alpha,\beta)$ thing in the denominator? Well, that thing is called the beta function, and the beta function is actually a ratio of gamma functions, as follows:

$$\beta(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

One very important thing to note – the range of x in the equation for the beta probability density is clearly denoted as being between 0 and 1. This is key. The beta function is used to model data measured as proportions. For example, if you have data on the proportion of an amino acid in a protein motif (say a leucine zipper) and it is not likely to be normally distributed (check with a Q-Q plot), then you should model the data with a beta density function.

Unfortunately the interpretation of the parameters with the beta are not as clear as with the gamma, and with the beta distribution, the alpha and beta parameters are sometimes referred to as the "shape 1" and "shape 2" parameters. Both parameters play a role in how the data fits the distribution. As with the gamma, the alpha and beta parameters have a relationship to the mean and variance of the distribution, with the following mathematical formulas:

$$\text{mean} = \frac{\alpha}{\alpha+\beta}$$

$$\text{variance} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

The formula for the mean is not too bad, but remember to solve for alpha and beta you need to solve for both – and algebraically it's not quite so pretty to do with the above relationships. Because the beta distribution is a sine qua non in Bayesian statistics (and literally, used all over the place, along with its multivariate counterpart the Dirichlet) it is worth the time to write a small program to calculate these parameter values. Meanwhile, let's learn how to work with the dbeta density function and the pbeta cumulative distribution functions in R.

Let's use as an example the proportion of acidic amino acids found in a particular motif of proteins (a generic example with no particular type of motif). Assume we have already determined the parameters of the beta density that models our data. We have alpha ("shape 1") =2 and beta ("shape 2") = 10. A graphical model of this density is made in R with the following command:

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> x
 [1] 0.11 0.10 0.10 0.16 0.20 0.32 0.01 0.02 0.07 0.05 0.25 0.14 0.11 0.12
0.08
[16] 0.13 0.08 0.14 0.09 0.08
> data<-x
> n <- length(data)
> x <- seq(0,1,length=200)

> plot(x,dbeta(x,2,10),xlab="prop. of acidic residues", ylab="f(x)",
+ main="Beta PDF for residue data")
> points(data,rep(0,n))
```

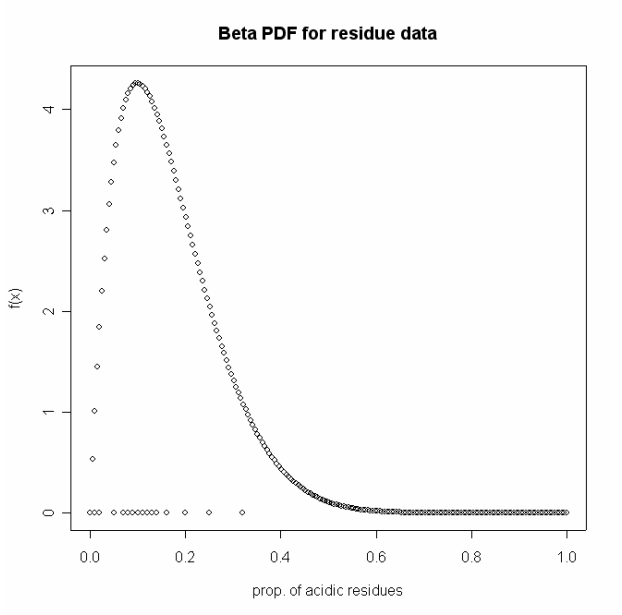


Figure 7-17: Beta PDF

Similarly a CDF plot can be generated using the pbeta function:

```
> plot(x,pbeta(x,2,10),xlab="prop. of acidic residues", ylab="Cum. prob",
+ main="Beta CDF for residue data")
> points(data,rep(0,n)
```


Permission granted to reproduce for nonprofit, educational use.

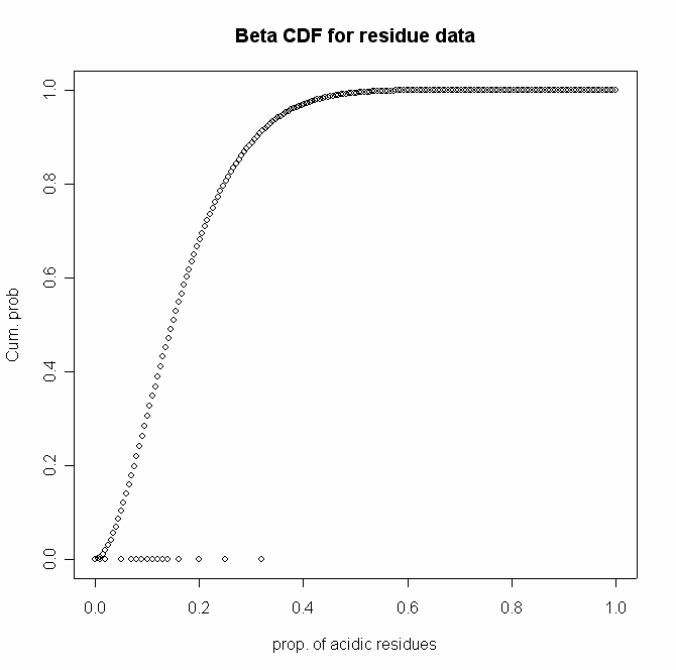


Figure 7-18: CDF for data distrbuted with a beta model

Other Continuous Distributions

Many other continuous distributions exist, but will not be discussed here. The interested reader is advised to consult a probability and/or mathematical statistics book for further information. R contains functionality to analyze many of these distributions and some distributions can be computed using a transformation of an existing distribution. For example, there is such a thing as an inverse gamma distribution, which can be computed using gamma distribution functionality. Two other continuous distributions which the reader may be familiar with will be introduced later – the t distribution and the F distribution. The t distribution will be introduced in the inferential statistics chapter and is the distribution which introductory statistics students are taught to use in the context of hypothesis testing. The F distribution will be introduced in the chapter on experimental design and plays an important role in microarray data analysis.

Simulations

One of the greatest powers of using a computer lies in the ability to simulate things. What if you don't have any data, but you know the probability model that you want to work with? Through the power of simulation, you can use the computer to generate sample values for you. It's like doing an experiment, only

Permission granted to reproduce for nonprofit, educational use.

in the virtual world instead of the real world. Simulation is an essential technique in computational statistics.

In R simulations are very easy to do. Every distribution mentioned in this chapter has a corresponding function in R that starts with "r" for random. All you have to do to obtain simulated values is specify the number of simulated values you want and the parameters of the function you are simulating from.

For example, let's simulate 20 values generated from a standard normal distribution. We just run the rnorm command with the appropriate parameters, storing the values in a data vector:

```
> y<-rnorm(10,mean=0,sd=1)
> y
 [1]  1.37100652  0.46028398 -0.83283766 -1.56743758  1.24318977 -0.43508915
 [7] -1.64050749  0.08383233 -1.56016713 -0.45454076
```

Note that every run of the above simulation should produce different values (do not expect the same results as above).

Once you have simulated data you often want to look at them graphically. One way to look at the simulated values is to plot a histogram, which is very simply coded below:

```
hist(y)
```

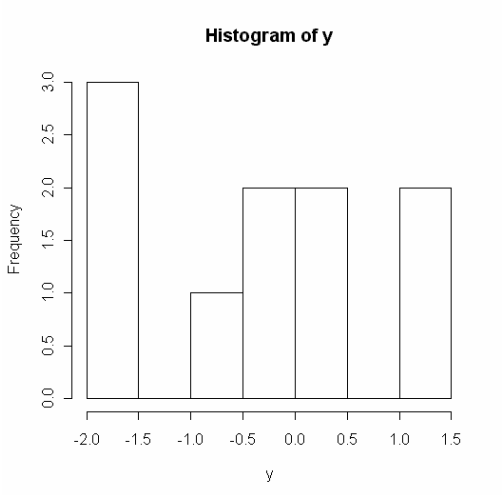


Figure 7-19

A histogram from a simulation of only 10 values is admiringly dull and usually hundreds or thousands of values are simulated. Let's try the above simulation a few more times, this time with 50, 100, 500, and 1000 values:

```
> y1<-rnorm(50,mean=0,sd=1)
> y2<-rnorm(100,mean=0,sd=1)
> y3<-rnorm(500,mean=0,sd=1)
> y4<-rnorm(1000,mean=0,sd=1)
```


Permission granted to reproduce for nonprofit, educational use.

```
> par(mfrow=c(2,2))
> hist(y1,nclass=10,main="N=50")
> hist(y2,nclass=10,main="N=100")
> hist(y3,nclass=10,main="N=500")
> hist(y4,nclass=10,main="N=1000")
```

As you can see from the plots in Figure 7-20, the more values you simulate the closer the histogram will appear to the distribution you are simulating from. The distribution with N=1000 appears to approximate a continuous standard normal distribution quite nicely.

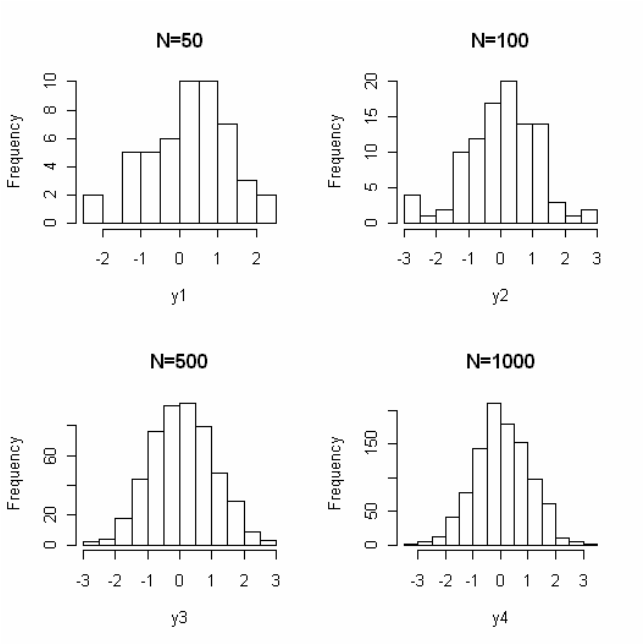


Figure 7-20: Increasing the number of simulations from a normal distribution

The reader is encouraged to try more simulation experiments and plots with the distributions discussed in this chapter. There will be much more discussion of simulations in coming chapters.

Permission granted to reproduce for nonprofit, educational use.

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.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Assuming they are diploid, peas can have a genotype that is homozygote (yy or YY) or heterozygote (Yy or yY). Assuming equal frequencies of both alleles, the probability of a heterozygote is ½, and the probability of the homozygote is ½ as well. Peas of genotype yy are green, with a probability of ¼, and peas of genotypes Yy, yY, and YY are yellow, and therefore the probability of a yellow pea is 3/4.

Next, suppose we have a yellow pea, and want the probability that the pea is also a heterozygote. In terms of probability theory, we are restricting the sample space for the event pea color to the event that the pea is yellow, and creating a new sample space consisting only of yellow pea color. Within this new restricted space, we are asking what is the probability of the event heterozygous pea. In conditional probability jargon, we are conditioning the event of heterozygous pea on the event of yellow pea. The event yellow pea is our prior event that we already have information about.

Mathematically we can look at this example using language and notation from probability theory. We know from our basic genetic information that the probability of a pea being yellow is ¾ and we know that the probability of a pea being heterozygous and yellow is 2/4, based on the calculation that peas can be of Yy, yy, yY, and YY and 2 of four of these events (Yy and yY) are both yellow and heterozygous (we look more at joint probabilities later in this chapter). We can then use the joint probability of the event "heterozygous and yellow" and divide this by the event of "yellow" to calculate the probability of being heterozygous and yellow as follows:

$$P(\text{yellow and heterozygous}) = 2/4$$

$$P(\text{yellow})=3/4$$

$$P(\text{heterozygous}|\text{yellow}) = \frac{P(\text{yellow and heterozygous})}{P(\text{yellow})} = 2/3$$

The notation P(heterozygous|yellow) is standard notation for conditional probability where the event being conditioned on comes after the "|" notation. P(A|B) is read as "the conditional probability of the event A given that the event B has occurred".

Using Trees to Represent Conditional Probability

Often looking at a graphical illustration helps to understand a concept. Trees are a visual way to represent conditional events and probabilities. Initial branches of the tree depend on the stem and finer branches depend on the previous branch. Let's use a simple tree diagram to illustrate our example (Figure 8-1).

Permission granted to reproduce for nonprofit, educational use.

The first branch of the tree represents the event of pea color and the second branch of the tree represents genotype (homozygous versus heterozygous) conditioned on the pea color.

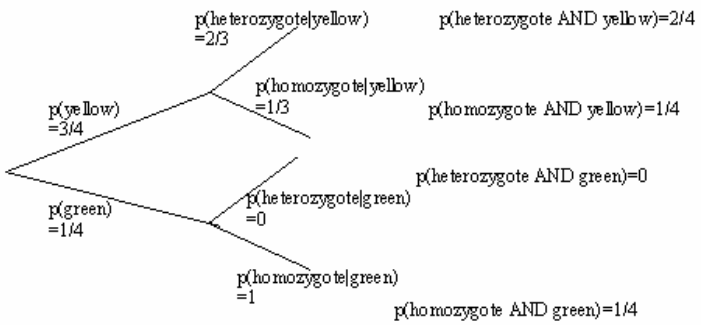


Figure 8-1: Conditional Probability Tree Diagram

Note that the second branch tree could have been written with branches for each of the four genotypes but is drawn here using the more simplified two branch version of heterozygous versus homozygous. The combined result at the end of the second branch is the joint probability of both events. Later in this chapter joint probability will be explained in more detail.

It is also important to observe in Figure 8-1 that probabilities within each set of branches add to 1. In the second branch, this stems from the fact that when we condition on an event, we define a new conditional sample space and the conditional probabilities obey the axioms and rules of probability within this new (conditional) sample space. In our example, the conditional sample space is based on pea colors.

Independence

It often happens that knowledge that a certain event E has occurred has no effect on the probability that some other event F occurs. In other words, the conditional probability of event F given event E is just the probability of event F. This can be written mathematically as $P(F \mid E) = P(F)$. One would expect that in this case, the equation $P(E \mid F) = P(E)$ would also be true. In fact each equation implies the other. If these equations are both true, then F is independent of E and this is formalized in the definition of independent events, which states that two events E and F are independent if $P(E|F)=P(E)$ and $P(F|E)=P(E)$.

Here is an alternative way to define independence. Two events E and F are independent if both E and F have positive probability and if $P(E \cap F) = P(E)P(F)$.

Permission granted to reproduce for nonprofit, educational use.

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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

of observing a sequence ATCGA is simply P(A)P(T)P(C)P(G)P(A) or $(1/4)^5$=1/1024.

In the case above the nucleotides are assumed equally likely. However the concept of independence can easily be applied to the case of nucleotides of different frequencies. Suppose that P(C)=P(G)=1/3 and P(A)=P(T)=1/6. Then, assuming independence, the probability of sequence ATCGA is $(1/6)^3(1/3)^2$ which calculates to 1/1944. The importance here is that the event of a particular nucleotide in a sequence is independent of other nucleotides in the sequence, not that the probabilities of each nucleotide be the same.

In many instances, in sequence analysis or in analyzing other events, it is clear events are not independent. Two events that are not independent are said to be dependent. For example, in analyzing the nucleotide sequence for a start codon (ATG) or other sequence motif independence does not hold and the subsequent nucleotides are dependent on the prior nucleotide within that sequence motif.

Joint and Marginal Probabilities

Joint probability is a pretty self-descriptive concept – it's the probability of two (or more) events at once. Here we will look at the concept of joint probabilities, which will serve as preparation for coverage of joint distributions later in this chapter, but is also a subject of use on its own. Joint probability is officially defined as the probability of the intersection of two events. Joint probability was diagrammed with a Venn diagram in chapter 6 when we discussed the set theory concept of intersection. Recall that the intersection of two events uses the symbol "$\cap$". Using this notation, P(A$\cap$B) symbolizes the joint probability of events A and B.

Tables are often used to display joint probabilities. For example, given a particular DNA sequence we obtain the following (hypothetical) joint probabilities for two adjacent nucleotide sites (Table 8-1):

Permission granted to reproduce for nonprofit, educational use.

Table 8-1: Joint Probabilities of Nucleotides at Adjacent Sites

		Nucleotide at position 1			
Nucleotide at position 2		**A**	**T**	**C**	**G**
	A	0.2	0.1	0	0.1
	T	0	0.1	0.1	0.1
	C	0.1	0	0.1	0
	G	0	0.1	0	0

Although not immediately obvious to the untrained eye, a skilled probability practitioner can harvest from Table 8-1 a lot of useful information. Each table cell contains the probability of the intersection (joint probability) of events. For example, in the first cell the entry is the joint probabilitiy of nucleotide A in position 1 and nucleotide A in position 2. Therefore, the joint probability is 0.2. Fundamental to a joint probability table is the fact that all probability entries add up to 1, which is simply an expression of the axiom of probability that the probabilities of all events have to sum to 1.

What if we want to know the probability of nucleotide A being at position 1 regardless of the nucleotide at position 2? This calculation is the column total of the nucleotide A in position 1 column, or 0.3. This probability is called the marginal probability. Table 8-2 expands this idea calculating all the marginal probabilities for all columns (marginal probabilities for nucleotide at position 1) and all rows (marginal probabilities for nucleotide at position 2).

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Table 8-2: Computing Marginal Probabilities

		Nucleotide at position 1				Marginal probabilities for rows
Nucleotide at position 2		**A**	**T**	**C**	**G**	
	A	0.2	0.1	0	0.1	0.4
	T	0	0.1	0.1	0.1	0.3
	C	0.1	0	0.1	0	0.2
	G	0	0.1	0	0	0.1
Marginal probabilities for columns		0.3	0.3	0.2	0.2	1

Calculating conditional probabilities from the information in the table is also a breeze. For example, to calculate the conditional probability of a nucleotide at position 2 being T given the nucleotide at position 1 is a G we use the following formula from the definition of conditional probability…

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

…and simply apply the formula to the desired conditions.

$$P\ (T \text{ at } P2 \mid G \text{ at } P1)=\frac{P(T \text{ at } P2\cap G \text{ at } P1)}{P(G \text{ at } P1)}=\frac{0.1}{0.2}=1/2$$

Here, the joint probability, P(T at P2 ∩ G at P1), is obtained from the cell in Table 8-2 containing the joint probability for nucleotide G at position 1 and T at position 2, and P (G at P1) is the marginal probability of nucleotide G at position 1 obtained from the column total for that column in Table 8-2.

The Law of Total Probability

The law of total probability provides a method of calculating the probability of an event, which we'll denote by A, by conditioning on a set of mutually exclusive and exhaustive events, which we'll denote by $B_1,B_2,\ldots,B_n$. Note that the B_k are usually the different outcomes of a sample experiment (all possible events in a given sample space). Remember that mutually exclusive means that two events have no common intersection and that their joint probability is 0, that is (Bi ∩ Bj) = ∅ for any i, j. Exhaustive means the entire sample space or union of all events, $B_1 \cup B_2 \cup \ldots \cup B_n$ = the sample space.

Permission granted to reproduce for nonprofit, educational use.

The law of total probability is best illustrated using the "Pizza Venn Diagram" in Figure 8-2, and can be summarized mathematically as follows:

$$P(A) = \sum_{i=1}^{n} P(A \mid B_i) P(B_i)$$

In the above formula and in Figure 8-2, A is the union of disjoint (mutually exclusive) sets, A∩Bi, for all i. P(A) can also be written as:

$$P(A) = \sum_{i=1}^{n} P(A \cap Bi)$$

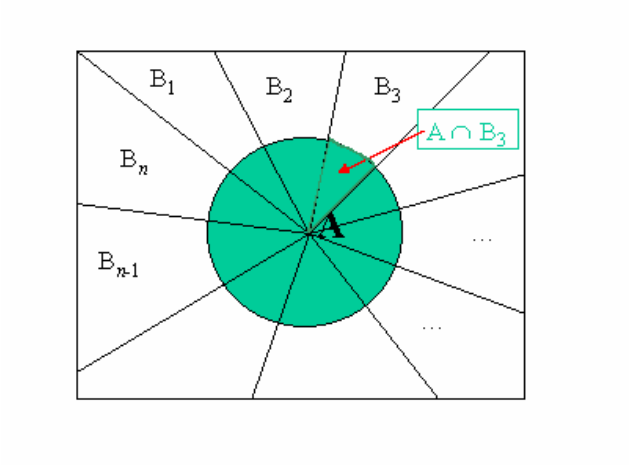


Figure 8-2: Illustrating the Law of Total Probability

Although the law of total probability may seem confusing, we just applied the law of total probability earlier in this chapter when calculating marginal probabilities in the adjacent nucleotide example. In this case we used the formula $P(A) = \sum_{i=1}^{n} P(A \cap Bi)$ where A is the nucleotide we are calculating the marginal probability of and the Bi's are the four nucleotides we are calculating the marginal probability over.

For example, to calculate the marginal probability of A in the first nucleotide position: P (A in position 1) = P(A in P1∩ A in P2) + P(A in P1∩ T in P2) + P(A in P1∩ C in P2) + P(A in P1∩ G in P2). Doing the math, P (A in P1) is 0.3.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Probability Distributions Involving More than One Random Variable

So far we have only considered the study of distributions where one random variable is modeled. However, modeling probability for most real life phenomena, and certainly most phenomena in bioinformatics, usually requires modeling the distribution of more than one random variable. Being able to use multiple variable distribution models provides us with a very powerful data analysis tool.

Examples of phenomena being modeled with more than one variable are abound in scientific applications. An environmental study may include measures of temperature, green house gases, moisture, and other conditions measured at different locations and each measure modeled by a random variable. In biochemistry it may be possible to develop a statistical model predicting tertiary protein structure using random variables to model factors such as percentages of certain amino acid residues or motifs. Understanding how to use probability distributions involving two or more random variables is key to being able to model data involving more than one measurable dimension.

Joint Distributions of Discrete Random Variables

Let's revisit our example of joint probability of nucleotides in two positions discussed earlier in this chapter. Previously we considered joint probabilities of events of a particular nucleotide being in position 1 and a particular nucleotide being in position 2. Let's step this up and model the scenario using random variables. Let X be the random variable to model the nucleotide at position 1 and Y be the random variable to model the nucleotide at position 2.

We can re-write our familiar table as a joint probability mass function (pmf) of two random variables (Table 8-3).

Permission granted to reproduce for nonprofit, educational use.

Table 8-3: Joint Probability Mass Function for Nucleotides at Two Adjacent Sites

		X= Nucleotide at position 1			
Y=Nucleotide at position 2		**A**	**T**	**C**	**G**
	A	0.2	0.1	0	0.1
	T	0	0.1	0.1	0.1
	C	0.1	0	0.1	0
	G	0	0.1	0	0

This is a small but important change to the table. The table now models probability distributions for the two random variables. We can view each cell as representing P (X=xi, Y=yi)=P (X=xi ∩Y=yi), an individual value of the joint probability mass function denoted by $p(x_i, y_i)$.

Although we will not do so here, extending joint distributions to include more than 2 variables is pretty straightforward. We may extend the nucleotide example to include the distribution of the nucleotide in position 3 of a sequence. We could use the random variable Z to model third nucleotide. The probability of any given 3-nucleotide sequence such as ATG would be given by the joint distribution of random variables X, Y, and Z representing the respective probabilities of each nucleotide at each position. We write this as P ((X=A) ∩(Y=T)∩(Z=G)).

Marginal Distributions

Using the same logic used in calculating marginal probabilities described earlier, we can take the joint distribution and sum over all values of the other variable to create the marginal distribution of one variable. The only novel idea here is that we are assigning a random variable to the marginal probability, creating a marginal probability mass function for a discrete random variable.

For example, summing over all values for the second nucleotide position produces the marginal distribution or marginal probability mass function (pmf) of the first nucleotide, X, as depicted in Table 8-4.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Table 8-4: Marginal Probability Mass Function for X

X= Nucleotide at position 1			
A	**T**	**C**	**G**
0.3	0.3	0.2	0.2

Similarly, we could calculate the marginal probability mass function (pmf) of random variable Y by summing over all X values, as in Table 8-5.

Table 8-5: Marginal Probability Mass Function for Y

Y=Nucleotide at position 2		
	A	0.4
	T	0.3
	C	0.2
	G	0.1

The marginal distribution can be denoted using mathematical shorthand. For example, to denote the marginal probability of X we would write the following:

$$p_x(x_i) = \sum_y p(x_i, y)$$

This formula denotes the probability of X summed over all Y values in the joint distribution. Note that the sum of probabilities for each marginal distribution adds to 1 (obeying the law sum of all probabilities in a sample space sums to 1), which should always be the case (and serves as a good check for determining if you did the correct calculations).

Conditional Distributions

Sometimes we may be interested in the distribution of one variable conditional on a specified value of the second variable.

For example, we may be interested in the distribution of second nucleotides (Y) given that the first nucleotide is an A. For each nucleotide modeled by the distribution of Y, the conditional probability is calculated by using the conditional probability formula:

$$P\,(Y=yi|X=A) = \frac{P(Y = yi \cap X = A)}{P(X = A)}$$

Permission granted to reproduce for nonprofit, educational use.

In this formula, the numerator is the joint probability of the first nucleotide being A and second nucleotide being nucleotide yi. The denominator is the marginal probability of the first nucleotide being A.

For example, using previous data from Table 8-1, to calculate the probability that the second nucleotide is an A (Y=A) conditioned on the probability that the first nucleotide is an A we perform the following probability calculation:

$$P\ (Y=A|X=A)=\frac{P(Y=A\cap X=A)}{P(X=A)}=\frac{0.2}{0.3}=0.66$$

Continuing this calculation for the other nucleotides, we obtain the conditional distribution of Y given X=A in Table 8-6.

Table 8-6: Conditional Distribution of Y given X=A

Y=Nucleotide at position 2 GIVEN X=A	**A**	0.66
	T	0
	C	0.33
	G	0

Again, note that the sum of conditional probabilities adds to 1 and fulfills the law of probability that the sum of probabilities of events in a sample space adds to 1. When calculating a conditional distribution we are redefining the sample space to a specific condition and redefining probability calculations to be valid within the new, redefined sample space.

Joint, Marginal and Conditional Distributions for Continuous Variables

Because of the importance of discrete data in bioinfomatics and the relative simplicity of working with discrete data, only discrete joint, marginal and conditional distributions have been discussed in detail. However, although most sequence analysis will concern itself with discrete data and distributions, other models often utilize continuous variables. Conceptually the joint, marginal, and conditional distributions are the same as for discrete variables, but the mathematics is more complicated.

The joint distribution of two continuous random variables can be modeled using a joint probability density function (pdf). The joint pdf of two continuous random variables X and Y is a two dimensional area A and can be evaluated by

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

integrating over this area with respect to each variable for given values of X and Y

$$P((X,Y) \in A) = \iint_A f(x,y)dydx$$

Since evaluating this requires integration techniques from multivariable calculus, we will not evaluate such integrals here. But conceptually the probability that (X, Y) lies in area A is equal to the volume underneath the function f(x,y) over the area A.

Calculating a marginal distribution of a continuous variable is similar to calculating a marginal discrete random variable distribution. In the case of the discrete random variable, this is done by summing over the other variable(s) whereas in the case of a continuous random variable, this is done by integrating over the other variable(s).

For example to determine the marginal pdf of X given the joint distribution of continuous random variables X and Y, integrate over the distribution of Y (which if X and Y were discrete would be summing over all distribution of Y) as follows:

$$f_x(x) = \int_y f(x,y)dy$$

Again the calculus of computing these distributions is beyond our coverage here, but a conceptual understanding of how to compute a marginal distribution for a continuous variable is important and seeing how discrete and random variable distribution concepts are very similar is important as well.

The conditional probability distribution for two continuous random variables can also be calculated using some simple calculus. If X and Y have joint probability density function f(x,y), then the conditional probability density function of X, given that Y=y, is defined for any values as the joint probability of X and Y divided by the marginal probability that Y=y. This can be written mathematically where the conditional distribution is denoted by $f_{x|y}(x \mid y)$.

$$f_{x|y}(x \mid y) = \frac{f(x,y)}{f_y(y)}$$

Working with more than two continuous random variables is a simple extension of the concepts and techniques presented here for two random variables. The analytical methods and calculus for performing such calculations can become quite tedious. However, using a computer program can greatly simplify these types of calculations as well as perform simulations from complex distributions and their derived distributions. Many examples presented in this book will

Permission granted to reproduce for nonprofit, educational use.

perform the task of working with high-dimensional distributions using the computational power of R.

Graphical models beyond two random variables are not very practical, so among other applications, marginal and conditional distributions are often used to look at graphics of higher dimensional distributions, as we shall in some examples in the next section of this chapter.

Common Multivariable Distributions

Distributions of more than one random variable are extensions of univariate distributions. The distributions presented here are not should not seem entirely novel, because they build on univariate distributions presented previously by including more than one random variable in the model. A solid understanding of the univariate binomial, normal, and beta distributions (which can be reviewed in the previous chapter) is the foundation for understanding the three distributions we will look at here: the multinomial, the multivariate normal, and the Dirichlet. These three distributions are selected because they are the key multivariable distributions used in modeling data in bioinformatics.

The Multinomial Distribution

The multinomial distribution is the most commonly used discrete, high-dimensional probability distribution. The multinomial is an extension of the binomial distribution. Instead of just two possible outcomes (as in the case of the binomial), the multinomial models the case of multiple possible outcomes.

Consider an experiment that models the outcome of n independent trials. Each trial can result in any of r different types of outcomes (compared to just r=2 in the binomial case). The probability of any of the outcomes is constant, just as in the binomial model the probability of success and probability of failure were held constant for a particular model. These probabilities are denoted by $p_1,p_2,\ldots,p_r$ and the sum of the probabilities for all outcomes sums to one, that is $p_1+ p_2 +\ldots+ p_r = 1$.

If we count how many outcomes of each type occur, we have a set of r random variables $X_1,X_2,\ldots,X_r$. Each X_j = the number of outcomes of the j^{th} type (where j=1 to r) and the actual counts are values of each random variable, denoted by $X_j = x_j$, etc. Note the sum of the values of the random variables is n, the total number of trials, that is $x_1+x_2+\ldots+x_r = n$.

Because we are dealing with a series of independent events, any particular sequence of outcomes consists of x_1 of the first kind, $x_2$of the second kind, etc and has probability

$$p_1^{x_1} p_2^{x_2} \cdot \ldots \cdot p_r^{x_r}$$

Permission granted to reproduce for nonprofit, educational use.

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

Permission granted to reproduce for nonprofit, educational use.

values to look at graphs and other details of the distribution. Later in chapter 13 we will do similar types of simulations using a technique called bootstrapping.

To perform simulations, we can write a simple function in R that generates values of the three random variables (genotype counts) from a multinomial distribution given the parameter values of the proportions of each genotype:

```
#function for drawing random values
#from a multinomial distribution

#takes as parameters
## parameter N number of simulations
## parameter n is number of trials simulated (sample size)
## parameter p is a vector of proportions

rmnomial_function(N,n,p){
    l<-length(p)
    x<-rbinom(N,n,p[1])
    if(l==2)
            {cbind(x,-x+n)}
    else
            {cbind(x,rmnomial(N,-x+n,p[2:1]/sum(p[2:1])))}

}
```

To illustrate the use of this function, let's perform 10 simulations of 20 individuals using our empirical parameters for the proportion vector.

```
> ## Define N to be 10 trials
> N<-10
> ## Define n to 20
> n<-10
> ## Define our p vector containing empirical values
> # pAA=0.2, pAa=0.7, paa=0.1
> p<-c(0.2,0.7,0.1)
> ## Call function with these parameters, store in results
> results<-rmnomial(N,n,p)
```

This produces the following matrix of simulated values of the three random variables we are modeling:

```
> results

 [1,] 4 11 5
 [2,] 4 13 3
 [3,] 2 12 6
 [4,] 3 13 4
 [5,] 4 13 3
 [6,] 6 11 3
 [7,] 3 13 4
 [8,] 4 15 1
 [9,] 1 13 6
[10,] 7  7 6
```

We could easily write some code to calculate proportions for the values for the simulated random variable values:

Permission granted to reproduce for nonprofit, educational use.

```
> 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:

Permission granted to reproduce for nonprofit, educational use.

$$p_{X_j}(x_j) = \binom{n}{x_j} p_j^{x_j} (1-p_j)^{n-x_j}$$

To examine this more concretely, let's continue our example and do some illustrative simulations. Only now let's just be concerned with the first variable, X. Instead of considering both Y and Z variables, let's consolidate them into a variable W, representing "everything else" which is not genotype AA, and make W=X+Y=(n_{Aa}+n_{aa}) and pW=(1-pAa-paa)=0.8. We now have a binomial model that models X (counting the trials of AA as "successes" and all other outcomes as "failures") alone. Using R, let's simulate this distribution in two ways and compare the results.

First let's simulate 10,000 values using the multinomial model and all 3 random variables. To do this, repeat the simulation described above but changing parameter N to 10,000.

```
> results<-rmnomial(10000,20,c(0.2,0.7,0.1))
> ## Change results to proportions
> results2<-results/(results[,1]+results[,2]+results[,3])
> ## Store column 1 of results in X
> X<-results2[,1]
> ## Store column 2 and 3 results in W
> W<-(results2[,2]+results2[,3])
```

Summarize values by getting the means for X and W:

```
> mean(X)
[1] 0.19913
> mean(W)
[1] 0.80087
```

The mean for X is roughly 0.2 and the mean (the proportion of "successes" in the binomial) and the mean for W is 0.8 (the proportion of "failures" in the binomial). These values should seem logical.

To look at the result visually, plotting the proportions with a histogram is simple, as coded below and shown in Figure 8-3.

```
>  hist(X,nclass=10,main="Simulated prop. AA using Multinomial")
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

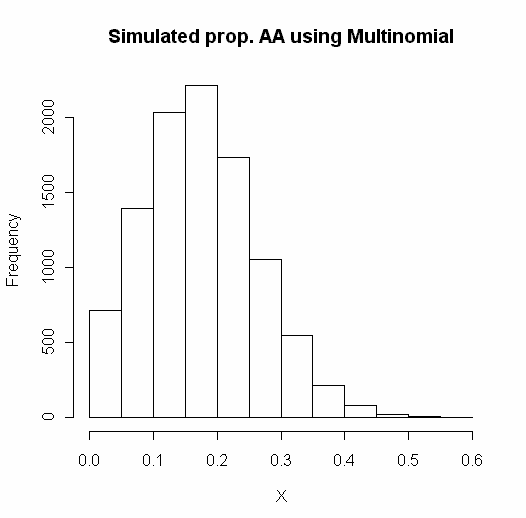


Figure 8-3

For comparison, a quick simulation from the binomial of 10,000 values using p=0.2 is simple to code using the rbinom function:

```
> ##Simulate 10000 values from binomial
> ##Simulate rv for n=20 and p=0.2
> B<-rbinom(10000,20,0.2)
> ##Convert to proportions by dividing by 20
> hist(B/20,nclass=10,main="Simulated prop. AA using Binomial")
```

The histogram using the binomial simulation is shown in Figure 8-4. It should be apparent from the graphs that the distributions in Figure 8-3 and Figure 8-4 are virtually identical.

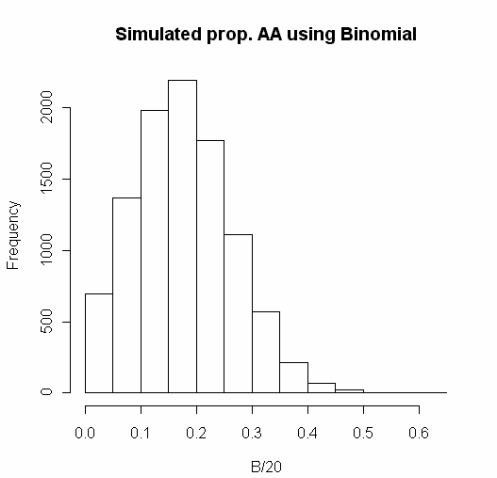


Figure8-4

Permission granted to reproduce for nonprofit, educational use.

Multivariate Normal Distribution

The multivariate normal is a favorite distribution in multivariate statistics. Often data are mathematically transformed to fit a normal model. Such transformation of data can be somewhat controversial, because sometimes they are performed for convenience and simplicity and the results may be misleading or difficult to translate back to the original variables. Nonetheless, most of inferential multivariate statistics utilize the multivariate normal. Understanding how the normal distribution can be extended to include more than one random variable is important.

Because most of the mathematical aspects of dealing with the multivariate normal involve advanced techniques of calculus and matrix algebra, we will consider the details of only one limited example of the multivariate normal. We will consider the bivariate normal model, which models the joint distribution of two independent normally distributed random variables.

Recall, from the previous chapter, the mathematical formula for the normal distribution:

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

We can model their joint distribution of the two random variables X and Y by simply looking at the product of the marginal distributions of their two marginal distributions since they are independent variables. Thus the model for the joint distribution of two independent normally distributed random variables X and Y (aka: the bivariate normal) is:

$$f(x, y) = f_x(x) f_y(x)$$

We can write this by using the normal distribution equation for both random variables and multiplying them:

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

Let's now take a look at what this looks like graphically using R. Let's reduce the case even further by considering only a standard bivariate normal distribution. Recall that the standard normal distribution has mean of 0 and standard deviation of 1. Thus, we can further re-write the equation above considering X and Y as standard normal random variables, each having mean 0 and standard deviation 1:

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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

By performing some algebra, this can be simplified to:

$$f(x,y) = \frac{1}{2\pi} e^{-\frac{(x^2+y^2)}{2}}$$

This is a pretty workable equation to plot. To do this, we can write a function in R to simulate draws from a bivariate standard normal. In the code below we create two vectors of length 20, x and y and then write a function to calculate values of f(x,y). The results of the function call are stored in a variable matrix z. Then the data can be viewed using a simple 3-d perspective plot, presented in Figure 8-5.

```
> x<-seq(-2,2,length=20)
> y<-x
> bvn_function(x,y){
+ (1/2*pi)*exp(-0.5*(x^2+y^2))
+ }
> z<-x%*%t(y)
> for(i in 1:20){
+ for(j in 1:20){z[i,j]<-bvn(x[i],y[j])}}
> persp(x,y,z)
```

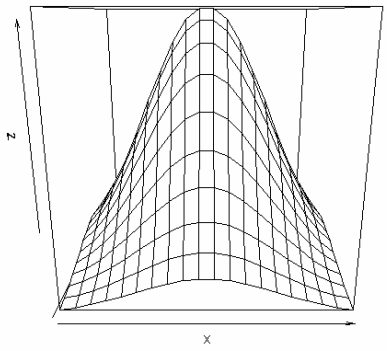


Figure8-5: Perspective Plot of Bivariate Standard Normal Distribution

Figure 8-5 is shaped like a symmetric upside down cone with its base on the XY plane. Along the X-axis (front view of the plot) the perspective lines show a pattern of the normal distribution reflecting that any "slice" of the cone would have the shape of a normal distribution.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Once again, we didn't have to write a function, because random generations of multivariate normal distributions of any dimension can also be done using the package mvtnorm. Mvtnorm contains functionality for generating random values from all kinds of multivariate normal distributions, and from related multivariate t distributions for any model, and with any parameter values, not just with standard parameter values. Let's use the package's function rmvnorm to generate 1000 values from a bivariate standard normal distribution:

```
> data<-rmvnorm(1000,mean=c(0,0))
> data
                  [,1]          [,2]
   [1,] -0.9152555414  0.5950708803
   [2,] -1.2240565493  0.3079036163
   [3,] -1.2205942482 -0.9042616927
…
  [999,]  2.0735458497 -1.7003787054
 [1000,] -0.0962237236  0.0056516042
```

In our result we have a matrix of data where the first column has values of random variable X values from standard normal simulation, and the second column is random variable Y values from a standard normal simulation. A scatter plot in Figure 8-6 demonstrates that the joint density is shaped as we would expect data from a bivariate normal simulation to be shaped.

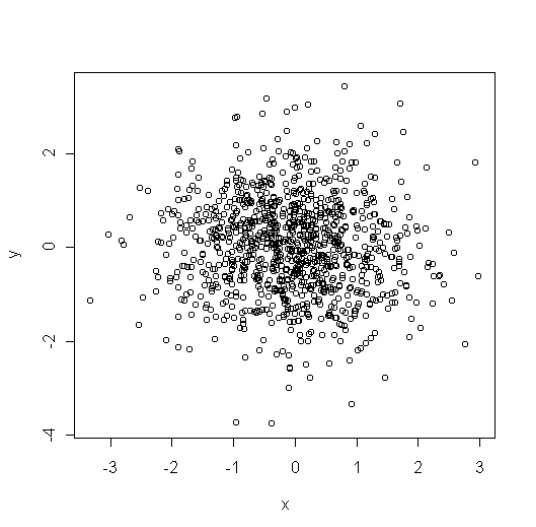


Figure8-6: Scatter plot of X and Y values from bivariate standard normal simulation

If we want to look at marginal distributions of X and Y, we can break down the data matrix into an X and Y matrix and look at the marginal distributions of X and Y very easily. All we have to do is store the X and Y values in new variables (although we technically don't even have to do this and could have just plotted the columns from the matrix) and do a histogram plot of the marginal distribution of interest.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> x<-data[,1]
> y<-data[,2]
> hist(x,nclass=20,main
+ ="Marginal distribution of x")
```

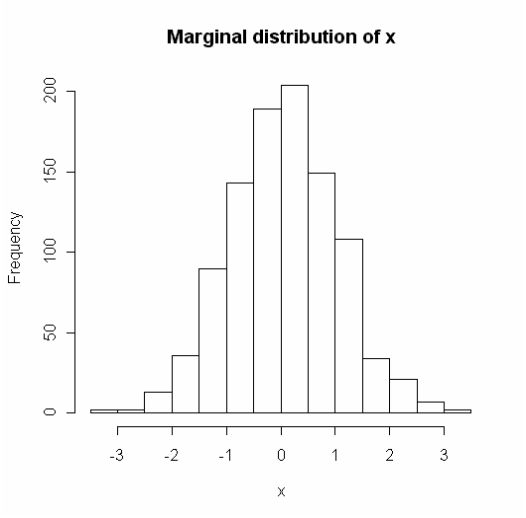


Figure 8-7: Marginal Distribution of X

From the histogram in Figure 8-7, the marginal distribution of X is quite normally distributed. If we did a simulation of 1000 values from a standard univariate normal distribution we would obtain a virtually identical result.

Dirichlet Distribution

The Dirichlet is the multivariable version of the beta distribution. Recall that the beta distribution is the distribution often used to model data in the form of proportions, i.e. values between 0 and 1. It should make sense that if you are modeling something such as the proportions of nucleotides in a DNA sequence, each proportion (A, T, C, G) can be modeled with an individual random variable, and the joint distribution for the proportions of all four nucleotides can be modeled using a Dirichlet.

We now denote by $X_1 \ldots X_k$ a set of proportions noting that $X_1 + \ldots + X_k = 1$ (and each $X_i > 0$). Mathematically the formula for the Dirichlet distribution, for k random proportions is:

$$f(X_1, X_2, ..X_K) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} X_i^{\alpha_i - 1}$$

Permission granted to reproduce for nonprofit, educational use.

Although this formula may look intimidating, it's not all that complicated! The Γ is just the symbol for the gamma function presented in chapter 6 and used in both the gamma and beta distributions in chapter 7. Capitol sigma $\sum$ is the symbol for addition (sum of), and the capitol pi, Π is the symbol for multiplication. The alpha's are the parameters for the random variable X's.

The constraint on the distribution is that the sum of proportions adds to one, that is, $\sum_{i=1}^{k} X_i = 1$ which should seem logical since when modeling the joint proportions of nucleotides in a sequence, for example, the total of all proportions is always 1. This follows the usual law that the total probability of all events in a sample space is 1.

As a simple example where we might use the Dirichlet, let's model the joint proportions of purines (A or G) and pyrimidines (C and T) in a given sequence. Let's use X1 to model the proportion of purines, and X2 to model the proportion of pyrimidines. Let's use the arbitrary choice of alpha=1 as a parameter for X1 and alpha=2 for X2. We model the proportion of purines as p1 and the proportion of pyrimidines as p2. Mathematically, with k=2 since there are two random variables being modeled, the joint distribution Dirichlet model is:

$$f(X_1, X_2) = \frac{\Gamma(\sum_{i=1}^{2} \alpha_i)}{\prod_{i=1}^{2} \Gamma(\alpha_i)} \prod_{i=1}^{2} X_i^{\alpha_i - 1}$$

Simplifying the above expression by substituting in the alpha values and p_i's and then performing addition and multiplication produces a formula that is not very intimidating and just involves algebra and calculation of a few gamma functions:

$$f(X_1, X_2) = \frac{\Gamma(3)}{\Gamma(1)\Gamma(2)} (p1)^{1-1} (p2)^{2-1}$$

Note that since $X_2 = 1\text{-}X_1$ we could simply view this as a marginal distribution of X_1 because for any given X_1, X_2 would be completely determined.

$$f_{X_1}(x_1) = \frac{\Gamma(3)}{\Gamma(1)\Gamma(2)} (x_1)^{1-1} (1 - x_1)^{2-1}$$

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

We note that this is simply the Beta distribution for X_1.with parameters $\alpha=\alpha 1=1$, and $\beta=\alpha 2=2$. Therefore the joint Dirichlet distribution for two random proportions X_1, X_2 ($X_1 + X_2 = 1$) is equivalent to the univariate Beta distribution for X_1 alone. Although in this example only two random variables are modeled and the model is pretty straightforward, when more random variables are modeled the Dirichlet can become quite complex. Because calculating the gamma functions in this equation can be computationally intense (even for quite high powered computers), sometimes the distribution is evaluated by taking logarithms (which make it computationally more efficient). It actually doesn't matter which base you use for your log calculations as long as you are consistent. The right side of the Dirichlet model can be calculated using logarithms like this:

$$=\log\left(\Gamma\left(\sum_{i=1}^{k}\alpha_i\right)\right)-\log\left(\prod_{i=1}^{k}\Gamma(\alpha_i)\right)+\log\left(\prod_{i=1}^{k}X_i^{\alpha_i-1}\right)$$

Another computation quirk of the Dirichlet distribution is that it is simpler to sample from the Dirichlet indirectly by using a method that draws k independent gamma samples and then computing random proportions (X_i) as the value of each sample divided by the sum of the k samples. It can be shown that the proportions X1,..,Xk have a Dirichlet distribution. We will not discuss mathematical details of this here, but in computer programs and in literature you will see Dirichlet being simulated using draws from the gamma distribution so it of interest to note this here and this trick is used in the code example below.

To simulate from the Dirichlet in R you could write your own function. The code below gives an example of a function that simulates n draws with a parameter vector of p:

```
rDir_function(n,a){
    l<-length(a)
    m<-matrix(nrow=n,ncol=p)
    for(i in 1:p){m[,i]<-rgamma(n,a[i])}
    sum<-m%*%rep(1,p)
    m/as.vector(sum)
}
```

Using this function to simulate values for n=20 with a vector of parameters (alpha values) for 3 random variables where alpha=1 for all three variables produces matrix of results where each column represents simulations for each random variable:

```
x<-rDir(20,c(1,1,1))
> x
              [,1]         [,2]       [,3]
 [1,] 0.0018936588 8.005894e-01 0.19751690
 [2,] 0.3344705893 1.894494e-03 0.66363492
 [3,] 0.0372989768 2.437322e-02 0.93832781
 [4,] 0.5159592455 6.830641e-03 0.47721011
…
```


Permission granted to reproduce for nonprofit, educational use.

Again there is no need to write a function to perform simulations, because the package gregmisc contains pre-written functions for computing density of or generating random values from the Dirichlet distribution. Performing the same computation as above using the function rdirichlet from this package produces the following:

```
>  x <- rdirichlet(20, c(1,1,1) )
> x
             [,1]        [,2]       [,3]
 [1,] 0.74226607 0.034630906 0.22310302
 [2,] 0.61271253 0.359267638 0.02801983
 [3,] 0.20446723 0.180993424 0.61453934
 [4,] 0.77386208 0.004850972 0.22128695
 …
```

Let's return to our example of modeling the joint distribution of the proportion of purines and pyrimidines and simulate 1000 values using the rdirichlet function:

```
> x<-rdirichlet(1000,c(1,2))
```

If we look at the mean values simulated for p1 and p2, these are the simulated proportions of x1 purines and x2 pyrimidines given that our parameters alpha=1 and alpha=2.

```
> mean(x[,1])
[1] 0.3354725
> mean(x[,2])
[1] 0.6645275
```

Each proportion has a marginal distribution. If we plot the marginal of x[,1] we get the following as depicted in Figure 8-8:

```
> hist(x[,1],nclass=20,main="Marginal of x[,1]")
```

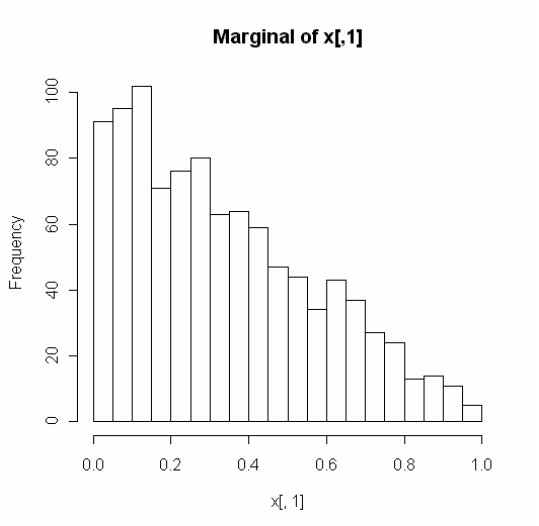


Figure 8-8

Permission granted to reproduce for nonprofit, educational use.

Likewise if we plot the marginal of x[,2] we get the plot in Figure 8-9

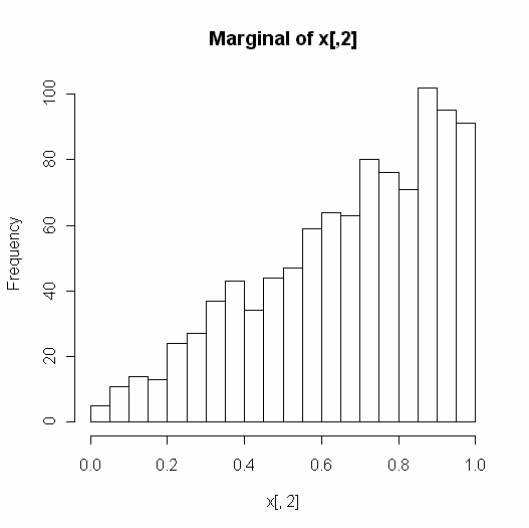


Figure 8-9

The Dirichlet is sometimes used on its own, but is used extensively in Bayesian statistics in conjunction with the multinomial distribution to form powerful models for working with counts and proportion data. It is important here to grasp the basics of the Dirichlet and its mathematical model and how to draw simulations from the Dirichlet using R.

Based on the discussions in chapters 6, 7, and 8 you should have an understanding of the basics of probability theory and a feel for working with univariate distributions and some select distributions of more than one variable. If you can understand the different models, what they are used for, how to perform simulations and how to graphically analyze results, you should be prepared to utilize these capabilities in some more advanced applications. The next few chapters contain an introduction to Bayesian statistics and an overview of Markov Chain methods, and coming discussions will build upon the concepts and methods presented thus far.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

9

An Introduction to Bayesian Data Analysis

Chapter 6 introduced the discussion of Bayesian statistics by discussing the two major schools of thought in statistics. We discussed frequentists statistics being the historic mainstream type of statistics usually taught in introductory statistics course and Bayesian being a school of statistical thought which build upon frequentists methods but incorporates subjective, as well as objective, thinking about probability.

Building on the concepts of probability and models introduced in chapters 6 though 8, this chapter introduces the Bayesian way of thinking about data models, and presents some basic examples of working with Bayesian models in R. The coverage here is primarily conceptual and only serves as a brief introduction to the extensive world of Bayesian statistics. The goal primarily is to provide a foundation for understanding the computationally intense methods (using R) presented in Chapter 10 that utilize Bayesian theory and are of increasing use in applications in bioinformatics. These methods will be applied in Chapters 11 and 12 in the study of Markov Chain methods.

There is sometimes a rift, even on occasion referred to as a holy war, among statisticians to be either frequentists or Bayesian, and to fight for the cause of their beliefs. This book does not present a biased view of either approach being superior to the other, but presents each view in terms of strengths and utilizes the approach most likely to be of use in bioinformatics applications discussed. For the applications in Chapters 10,11, and 12 Bayesian methods are favored for many reasons, highlights of which are discussed. However, after Chapter 12 we will return to working with frequentist statistics to study how to use R for

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

inferential statistics and microarray data analysis. Some topics discussed after that will utilize a combination of frequentists and Bayesian thinking.

The Essential Difference

The essential difference between Bayesian and frequentists is in interpreting what probability means. Since probability is the language and foundation of statistics, this difference affects not only how Bayesians think about and model probability, but also how they make inferences about data as well. Therefore, Bayesians have different methods of inferential statistics as well. However we are mainly concerned with Bayesian statistics in the context of working with probability models and will only touch on the differences in making inferences.

To illustrate the difference in how a frequentist and a Bayesian view probability, consider the statement that "a fair coin has a probability of landing on its head of ½".

To a frequentist, this has an objective interpretation. A frequentist would reason that if the experiment of tossing a coin were repeated many times, the proportion of times the coin would land on its head would be ½. This probability value is a true, fixed measure of the chance that a coin lands on its head, mathematically approximated by repetitions of an experiment. The probability is the arithmetic average of measurements of the experimental outcome, sometimes called a point estimate and the sampling distribution would be normal. When we return to inferential statistics in Chapter 13 this will be studied in more detail.

A Bayesian however would not be concerned with the idea of a fixed, true proportion of times a coin lands on its head. To a Bayesian there is no such thing as a fixed probability statement. Bayesians consider probability statements to be measures of personal degree of belief in a certain hypothesis. For a Bayesian, there are no true measures, only certain probability distributions associated with their subjective beliefs. A Bayesian will therefore look at the outcome of a coin toss as a random variable, with an associated probability distribution.

Using subjective knowledge, the Bayesian would first guesses on what the probability of a parameter is (in this case, the outcome of a coin toss). This is called a prior distribution (and for the coin toss, which is a Bernoulli trial based experiment, a likely prior is the beta model, discussed in chapter 7). Then the Bayesian would collect data (tossing the coin) and based on the data collected would update the model. Most of this chapter will be concerned with understanding this process: how the Bayesian chooses a prior, how the data are modeled, and how the model is updated to produce what is called a posterior distribution. This process is the paradigm of Bayesian statistics.

Note, that the Bayesian might not choose ½ as a first guess (p success) but might choose 1/10. Perhaps the Bayesian has experienced in the past that coins do not land on their heads and has previously experimented with unfair coins. One

Permission granted to reproduce for nonprofit, educational use.

Bayesian may have a different belief about a prior distribution from another Bayesian's belief of prior distribution for the same process. What is important is that the Bayesian first uses subjective belief to produce an initial model of probability distribution. It is this incorporation of subjective belief into the model that distinguishes the Bayesian from the frequentists.

Why the Bayesian Approach?

Although at first the Bayesian approach may seem odd, and the incorporation of subjective belief into a probability model may seem controversial, there are many reasons why the Bayesian approach has an advantage for modeling probability in bioinformatics as well as many other applications.

Here are some good reasons to use the Bayesian approach, presented here briefly and without technical details.

- Easier Interpretation of Parameters.

  Anyone who has ever interpreted a confidence interval in frequentists statistics course (discussed in Chapter 13) knows that frequentist's ways of interpreting things can become a bit confusing. Having a distribution for a parameter, as the Bayesian approach does, is easier to understand then the frequentist's way of point estimates and standard errors.

- Less Theoretical.

  All of Bayesian statistics essentially revolves around one paradigm, that of Bayes theorem. The forms of distributions modeled may vary, but the paradigm for working with them is essentially the same.

- Computationally Intensive.

  The Bayesian approach is ideal for utilizing computational power. Software packages such as R can take advantage of the Bayesian approach because they (in combination with the appropriate hardware) can perform computationally intensive tasks.

- More Flexible.

  The Bayesian approach is extremely flexible in how to model distributions of data and can be applied to many situations.

- Deals Well with Missing Data Values.

  Methods of Bayesian data analysis are able to effectively "cover" for missing data values without lost of statistical robustness.

- Effectively Models High-Dimensional Data

Permission granted to reproduce for nonprofit, educational use.

As a consequence of its computational intensity, the Bayesian approach can handle situations involving multiple variables (parameters)

- Is a Method of Learning.

  As we shall see with Bayes Theorem, the Bayesian approach involves constantly updating the model based on new data. This is a method of statistical learning. Artificial intelligence and other disciplines take advantage of the Bayesian approach and utilize it in learning algorithms in data mining and other applications.

- Does Not Require Large Samples

  Bayesian methods do not require mathematical methods of asymptotic approximations for valid inferences. Algorithms based on Bayesian theory, including Markov Chain Monte Carlo, can be carried out and produce results with small samples very effectively.

Some potential disadvantages of the Bayesian approach include:

- Criticism of Using Subjectivity in the Model.

  This may be viewed as nonscientific and biased and is the most frequent source of dispute of Bayesians and Frequentists. The use of a prior distribution and its influence on the posterior model are discussed in depth in this chapter.

- Misunderstood.

  Bayesian models are not widely taught or utilized (and even when utilized, may not be utilized correctly) and are still not mainstream statistics. Therefore others may not understand the Bayesian model and may misinterpret it.

- Complicated?

  When Bayesian models are presented mathematically they may appear very complicated. However, whether they are more complex than frequentist statistics is debatable. In actuality Bayesian methods, especially in complicated models, are somewhat easier to work with then Frequentist models for the same model. The bases for all Bayesian models are fundamentally the same algorithm, which may be simpler than Frequentists models.

Bayes' Rule

The foundation of Bayesian statistics is a rather simple probability rule known as Bayes' rule (also called Bayes' theorem, Bayes' law). Bayes' rule is an accounting identity that obeys the axioms of probability. It is by itself

Permission granted to reproduce for nonprofit, educational use.

uncontroversial, but it is this simple rule that is the source of the rich applications and controversy over subjective elements in Bayesian statistics.

Bayes' rule begins with relating joint and conditional probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Figure 9-1 illustrates this, where A is the event inside the larger circle, B is the event inside the smaller circle, and C is the intersection of events A and B and represents the joint probability of A and B.

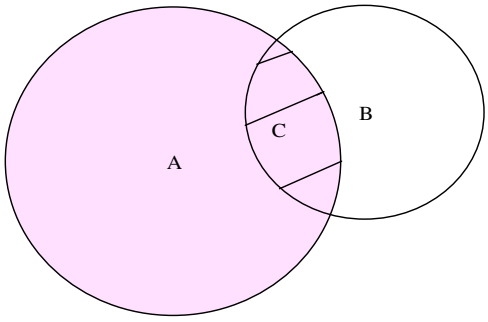


Figure 9-1

However, the probability of A and B can also be rewritten as:

$$P(A \cap B) = P(A|B) * P(A)$$

$$\text{Or as } P(A \cap B) = P(B|A) * P(B)$$

We can rewrite the relationship between conditional and joint probability of A and B given earlier as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \mid A) * P(A)}{P(B)}$$

Where the relation

$$P(A|B) = \frac{P(B \mid A) * P(A)}{P(B)}$$

Is known as Bayes' rule. Bayes' rule is sometimes called the rule of inverse probability. This is because it shows how a conditional probability P(B|A) can be turned into, or inverted, into a conditional probability P(A|B).

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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

Permission granted to reproduce for nonprofit, educational use.

or A^c), not being a membrane bound protein. The two rows represent the marginal distributions of B, having a high hydrophobic content and the complement of B (~B or B^c). Each cell represents the joint probability of two events.

Let's say we want to calculate the probability of a protein having a high hydrophobic content given that it is a membrane bound protein. To do this we can apply Bayes' rule in this form:

$$P(B|A)=\frac{P(A\mid B)P(A)}{P(B)}=\frac{P(A\cap B)}{P(B)}$$

$P(A\cap B)$ is the joint probability of A and B is simply found from the cell in the joint probability table and is 0.3. P(B) can be calculated by the law of total probability, calculating the $P(B|A)P(A)+P(B|\sim A)P(\sim A)$ or since we have the table by the sum of the row for event B, which is 0.5.

Therefore

$$P(B|A)=\frac{P(A\cap B)}{P(B)}=\frac{0.3}{0.5}=0.6$$

And we conclude given protein is membrane bound that there is a 60% chance the protein has a high hydrophobic residue content.

Table 9-1

		Type of Protein	
		Membrane Bound (A)	Non-membrane Bound (~A)
Prop. Hydrophobic residues	High (B)	0.3	0.2
	Low (~B)	0.1	0.4

In the first example (above) the computation of the desired conditional probability and use of Bayes rule are quite straightforward, since all the information needed is available from the joint probability table. Now let's consider a second example, where the computation would be quite difficult were it not for Bayes formula.

Suppose having a gene X results in the onset of a particular disease 50% of the time. Suppose the prevalence of having the gene X is 1/1000 and the prevalence of having a particular disease is 1%. From this, compute the probability of having gene X given that you have the disease

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

We could use this information and try to produce a joint probability table for the two events – having the gene and having the disease. Or, now that we know there is Bayes' rule, we can use it to solve this problem.

From the information above we are given:

P(Gene)=1/1000

P(Disease)=1/100

And

P(Disease|Gene)=0.5

Note that what we are doing here is inverting the probability P(Disease|Gene) to calculate P(Gene|Disease) in light of the prior knowledge that the P(Gene) is 1/1000 and the P(Disease)=1/100. The P(Disease) can be assumed to be the marginal probability of having the disease in the population, across those with and those without the gene X.

$$P(\text{Gene}|\text{Disease})=\frac{P(\text{Disease}\mid\text{Gene})P(\text{Gene})}{P(\text{Disease})}$$

Solving this is as simple as plugging in the numbers:

$$P(\text{Gene}|\text{Disease})=\frac{0.5*(1/1000)}{1/100}=0.05$$

This result is interpreted as if you have the disease (condition) the probability that you have the gene is 5%. Note that this is very different from the condition we started with which was the probability that you have the gene, then there is a 50% probability you will have the disease.

Bayes' rule could be applied over and over in order to update probabilities of hypotheses in light of new evidence, a process known as Bayesian updating in artificial intelligence and related areas. In such a sequential framework, the posterior from one updating step becomes the prior for the subsequent step. The evidence from the data usually starts to drive the results fairly quickly and the influence of the initial (subjective) prior will be diminished. However, since we are interested in Bayes formula with respect to Bayesian statistics and probability models, our discussion here will continue in this direction.

Extending Bayes' Rule to Working with Distributions

In a similar way to the previous chapter when we first looked at probability theory (of joint, conditional and marginal probability) and then applied it to

Permission granted to reproduce for nonprofit, educational use.

understanding how these principles apply to distributions, here we first looked at Bayes rule (above) but now, we will extend Bayes' rule and apply it to working with probability models (distributions).

Let's return to the "evidence" and "hypothesis" view of Bayes' rule, and apply it to our discussion at the beginning of the chapter regarding how a Bayesian would view the probability of a fair coin landing on its head.

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

We have already discussed that this can be written as a proportional relationship eliminating P(E), which is a constant for a given scenario, so:

$$P(H|E)\propto P(E\mid H)P(H)$$

Recall that P(H) represents the prior degree of belief in the hypothesis before the evidence. In this case, this will be the Bayesian's subjective belief of the proportion of times the coin will land on heads before the coin is flipped. Recall that the Bayesian would view this as a probability distribution. Therefore, we have to model P(H) with a distribution. We refer to the distribution modeled with P(H) as a prior.

But what distribution should we use for the prior? We have not collected empirical data so we have nothing to base a binomial distribution on. All we have is the Bayesian's subjective belief of the proportion of times that the coin will land on its head. Let's say the Bayesian believes that this proportion is ¼. Recall that the beta distribution is used to model probability distributions of proportion data for one random variable. How can we fit a beta distribution to model our P(H) for a proportion of ¼? Recall the beta model presented in chapter 7:

$$f(x)=\frac{1}{\beta(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1},\ 0<x<1$$

So really all we need here are some alpha and beta parameters which center the distribution around 1/4 (or 0.25). Having some experience with understanding how using different alpha and beta parameters affects the shape of the beta distribution (making a page of different beta distributions with various combinations of parameters is helpful for learning these things), we suspect that using alpha=2 and beta=5 for parameters will produce an appropriate distribution.

Let's go ahead in R and graph the prior distribution.

```
> x<-rbeta(5000,2,5)
> plot(x,dbeta(x,2,5),main="Prior dist. P(H)")
```


Permission granted to reproduce for nonprofit, educational use.

Figure 9-2 shows the resulting distribution. Is this an adequate prior? It is centered on 0.25, our Bayesian's subjective belief. It is not too precise and has a reasonable spread, reflecting both some certainty on the part of the Bayesian that the value is around 0.25, but also reflecting that the Bayesian views this as a distribution and there is a reasonable spread of variation in the distribution between 0 and 0.5, reflecting the Bayesian is fairly certain of his beliefs, but not so certain as to use an overly precise (invariable prior).

You may think this reasoning for defining a prior is quite non scientific. The purpose of a prior is to model the initial subjective belief of the Bayesian regarding the parameter of interest, before data are collected. We will discuss priors in more detail later in this chapter.

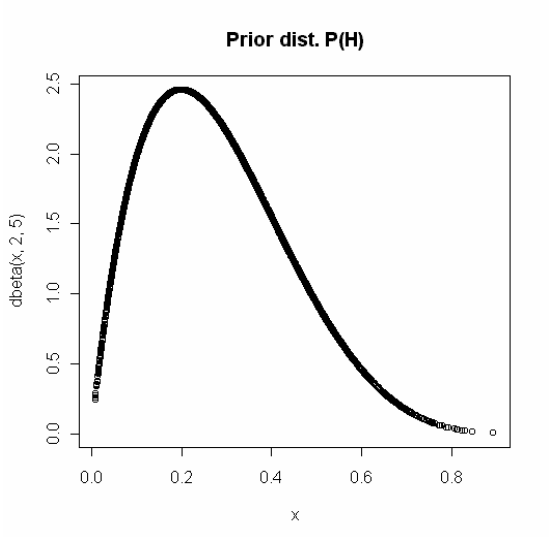


Figure 9-2

We now have the P(H) part of the model $P(H|E) \propto P(E \mid H)P(H)$. In order to calculate P(H|E) we need P(E|H), the probability of the evidence given the hypothesis. In order to get this, we need some evidence. Evidence, of course, is data. In order to get data, we need to perform an experiment, so let's collect some evidence.

Suppose we toss the coin 10 times and record whether it lands on heads or tails. Each trial of tossing the coin is a Bernoulli trial, and the combination of 10 trials can be modeled using the binomial distribution, which is (from chapter 7):

$$p(k) = P(k \text{ successes in n trials}) = \binom{n}{k} p^k (1-p)^{n-k}$$

Permission granted to reproduce for nonprofit, educational use.

Suppose our coin tossing experiments resulted in 5 of the 10 coins landing on heads. In frequentist statistics our estimate for the Binomial parameter p would be p=0.5 (the sample proportion) and the frequentist estimated binomial model for k=0 to 10 would be

$$p(k) = P(k \text{ successes in } 10 \text{ trials}) = \binom{10}{k} 0.5^{k} * 0.5^{10-k}$$

which can be modeled in R using the simple code:

```
> plot(0:10,dbinom(0:10,10,0.5),,type='h',xlab="",main="Data dist. P(E|H)")
```

With the resulting plot shown in Figure 9-3.

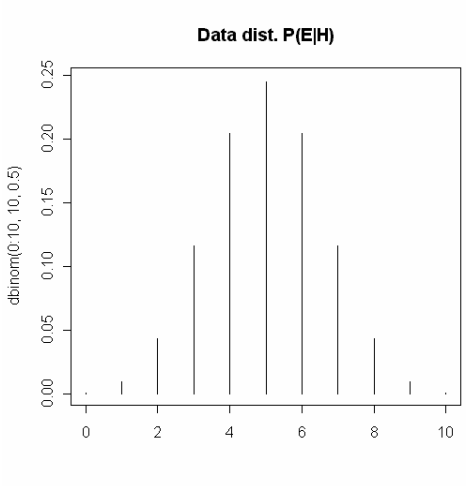


Figure 9-3

Since in the Bayesian paradigm p is a random variable, there are infinitely many plausible models for the data, namely one for each value of p. Two such models are depicted in Figure 9.3. The dependence of the probability of the data on the parameter value is also referred to as the likelihood, or likelihood function. In our case since the experiment resulted in k=5 the likelihood is

$$\text{Likelihood}(p) = P(5 \text{ successes in } 10 \text{ trials} \mid \text{parameter } p) = \binom{10}{5} p^5 * (1-p)^5$$

Likelihoods will be discussed more later in this chapter, but let's continue working with our extension of Bayes' rule to apply it to distributions.

With our prior distribution model and our data distribution model, we have now completed the right side of the proportion

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

$$P(H|E) \propto P(E \mid H)P(H)$$

Now what we want is P (H|E), the updated probability of belief in the hypothesis given the evidence. We want this of course, to be in the form of a probability model that we call the posterior.

Recall our beta model for P(H)

$$f(x) = \frac{1}{\beta(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1},\ 0<x<1$$

we concluded that alpha=2, and beta=5 are adequate parameters for modeling our P(H) with an Bayesian estimate of 0.25, this is our x value but x here represents p , the proportion value, so we can rewrite this as

$$f(p) = \frac{1}{\beta(\alpha,\beta)} p^{\alpha-1}(1-p)^{\beta-1},\ 0<p<1$$

Again, looking at the binomial data model equation:

$$P(k) = \binom{n}{k} p^{k}(1-p)^{n-k}$$

It should be apparent that the two models BOTH contain terms of p raised to an exponent, and (1-p) raised to an exponent. Hopefully you recall a day in math class sometime in secondary school when you encountered the rule for multiplying exponents of the same base, $a^m a^n = a^{m+n}$

Let's combine our models:

$$P(H|E) \propto P(E \mid H)P(H)$$

$$P(H|E) \propto \binom{n}{k} p^{k}(1-p)^{n-k} \frac{1}{\beta(\alpha,\beta)} p^{\alpha-1}(1-p)^{\beta-1}$$

Multiplication produces the posterior:

$$P(H|E) \propto \binom{n}{k} \frac{1}{\beta(\alpha,\beta)} p^{k+\alpha-1}(1-p)^{n-k+\beta-1}$$

Combining constant (for this particular) terms and replacing them with c (c is a normalizing constant which we will not be concerned with evaluating here):

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

$$P(H|E) \propto cp^{k+\alpha-1}(1-p)^{n-k+\beta-1}$$

Note that this follows the form of a beta distribution with parameters alpha(new)=k+alpha(old)-1, beta(new)=n-k+beta(old)-1.

Thus our posterior distribution has parameters alpha=5+2=7 and beta=10-5+5=10, so we have a beta (7,10) distribution to model the posterior distribution of P(H|E). Remember that in the Bayesian paradigm P(H|E) now represents the plausibility of the values p given our data which resulted in a value of k=5 in a binomial model with parameter p,. The plausibility is naturally represented by the posterior probability density function of the Beta (7,10) distribution.

Let's use R to graphically analyze our posterior solution (Figure 9-4):

```
> x<-rbeta(5000,6,9)
> plot(x,dbeta(x,6,9),main="Posterior dist P(H|E)")
```

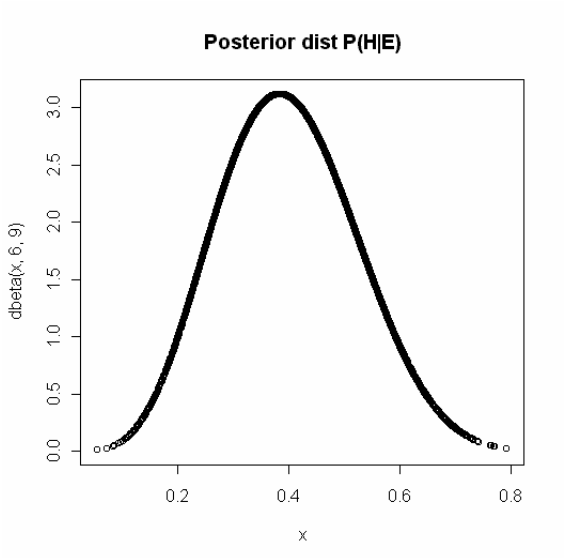


Figure 9-4

A more interesting plot can be produced in R using the following code, which uses the points function to plot the posterior distribution and the legend function to label the two distributions, as shown in Figure 9-5.

```
> p <- (0:1000)/1000
> plot(p,dbeta(p,2,5),,ylim=range(0:4),type="l"
+ ylab="f(p)", main="Comparing Prior and Posterior")

> lines(p,dbeta(p,7,10))
> legend(0,3,legend="Prior")
> legend(0.4,3.6,legend="Posterior")
```


Permission granted to reproduce for nonprofit, educational use.

Notice in Figure 9-5 that the posterior is shifted toward higher proportions, reflecting the updating of the model based on the data. Note though that the posterior is still not nearly centered on 0.5, the data value for p. We will return to discussing how the likelihood and prior influence the posterior distribution calculation later in this chapter.

We could update the posterior again, this time utilizing the current posterior distribution as the prior distribution and repeat the algorithm assuming the same data were achieved in a duplicate experiment. The readers are encouraged to do this experiment on their own with plots resulting in R. The result would be a posterior with updated parameters, which graphically shifts more toward 0.5.

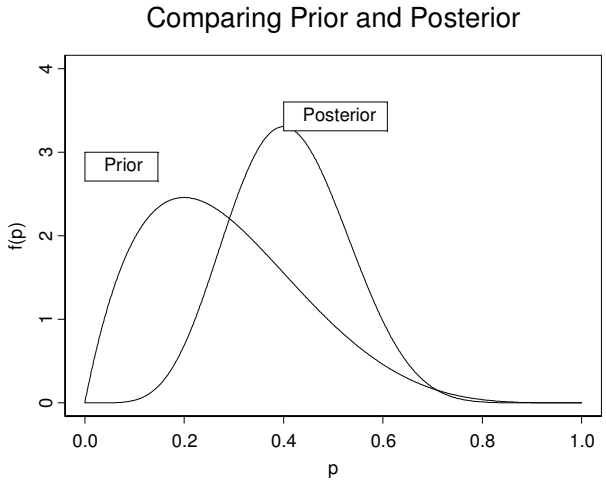


Figure 9-5

What is critical to understand in this example is not the details of the distributions used (which have previously been discussed) but the essence of the Bayesian algorithm as it applies to working with probability distribution models:

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

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

The essence of this is that the Bayesian's subjective prior model is updated by the data model (the Likelihood) to produce a posterior distribution that combines the Bayesian subjective knowledge with the objective data observed.

Although the mathematical complexity of the applications of this algorithms are broad and can be complex, this algorithm remains the same for the Bayesian methods. Although only simple examples are used in this book that touch on the

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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 more close 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

Permission granted to reproduce for nonprofit, educational use.

situation where there is little or no information about what the prior distribution is. Such priors are referred to as noninformative. Let's explore both classes of priors a bit further.

Conjugate Priors

The type of prior we will use for all our examples in this and coming chapters falls into the type of prior referred to as an informative prior or a conjugate prior. Conjugate priors have the property that the posterior distribution will follow the same parametric form as the prior distribution, using a likelihood model of a particular form as well (but the likelihood form is not the same as the prior and posterior form). This property is called conjugacy. In our example the beta prior combined with a binomial likelihood produces a beta posterior. This is always the case given these models. Therefore when a binomial model is used for the data model, one uses a beta for a prior and this will produce a beta posterior distribution. The example used earlier in this chapter for a coin toss can be applied to numerous scenarios. You may have noticed that we are mixing a discrete model for the data, the binomial, with a continuous model for the parameter p, the beta. That's OK. The subjective belief is quantified on a continuous scale for all possible values for p: $0 < p < 1$, and the data are Bernoulli trials modeled with a binomial model, this produces a continuous scale posterior model. There is no requirement for conjugate pairs to be both continuous or both discrete distributions. In an estimation problem, parameters are usually on a continuous scale, and hence the prior and posterior distributions are continuous.

There are other conjugate pairs of prior-likelihood to yield a posterior of the same form as the prior. Let's consider one more example using standard univariate distributions, the conjugate pair of a Poisson distribution with a gamma prior.

Recall the form of the Poisson distributions (from chapter 7). This is the data model for the occurrence of x events with rate lambda (=rate of occurrence, or average number of events per unit time or space). Let's use the symbol theta (θ) instead of lambda, where theta represents the rate parameter (all we are doing here is changing the symbol used, a perfectly legal mathematical maneuver).

If we observe a single data value x that follows the Poisson distribution with rate theta then the probability distribution of such a single "Poisson count" is:

$$P(x|\theta)=\frac{\theta^x e^{-\theta}}{x!} \text{ where x can take values } 0,1,2,3,\ldots..$$

Note that the rate can be any positive (real) number. If we observe n independent occurrences x1,x2,…,xn (or Poisson counts) they are modeled using the likelihood l:

Permission granted to reproduce for nonprofit, educational use.

$$P(\text{data}|\text{model})=P(x1,x2,\ldots,xn|\theta)=\prod_{i=1}^{n}\frac{e^{-\theta}\theta^{x_i}}{x_i!}$$

Leaving out the constant term (factorial in denominator):

$$P(\text{data}|\text{model})\propto\theta^{\Sigma xi}e^{-n\theta}$$

Recall also the gamma model, a continuous model with great flexibility of particular shape and location parameters, alpha and beta, used to model data which are greater than zero and continuous in nature but do not follow the normal distribution. The gamma is the conjugate prior for the Poisson distribution and can be used to model the prior distribution of the rate parameter theta:

$$f(\theta)=\frac{1}{\beta^{\alpha}\Gamma(\alpha)}\theta^{\alpha-1}e-\theta^{\beta}$$

Or reducing the model (eliminating the constant term) to produce a prior for theta distributed as a gamma:

$$p(\theta)\propto\theta^{\alpha-1}e^{-\beta\theta}$$

Let's use this model to measure the rate of a particular mutation in a particular gene, which has rate theta. According to prior knowledge, the rate per 1000 people at which this mutation occurs is 0.4. Based on this we can model this with a gamma distribution with alpha parameter of 2 and beta parameter of 5 (the mean of which is 0.4 from our discussion in chapter 7). That is the distribution of theta (here denoting the rate per 1000 people) for the prior is gamma (2,5). We then obtain some new data that finds that the in n=10 samples of 1000 people each, the number of mutations is as follows: (0,2,1,0,1,2,0,1,1,0)

Using the Poisson likelihood model and our gamma prior, we need to determine our posterior distribution based on this data where

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

$$P(\text{model}|\text{data})\propto P(\text{data}|\text{model})*P(\text{model})$$

$$P(\theta|x)\propto P(x|\theta)*P(\theta)$$

$$P(\theta|x)\propto\theta^{\Sigma xi}e^{-n\theta}*\theta^{\alpha-1}e^{-\beta\theta}$$

Adding exponents, this produces a posterior distribution with alpha(post)=Σxi+alpha(prior) and beta(post)=beta(prior)+n. Since in our 10 samples we found a total of eight people with the mutation, Σxi is 8. So our

Permission granted to reproduce for nonprofit, educational use.

updated parameters are alpha (post)=10 and beta (post)=15 with a gamma (10,15) distribution for theta. Let's look at a plot (Figure 9-6) of the prior and posterior distributions in light of the new data:

```
> theta<-seq(0,2,0.001)
> y<-dgamma(theta,2,5)
> plot(theta,y,ylim=range(0:3),ylab="",xlab="theta")
> y2<-dgamma(theta,10,15)
> lines(theta,y2)
> legend(0,2.5,"Prior")
> legend(0.6,2.8,"Posterior")
```

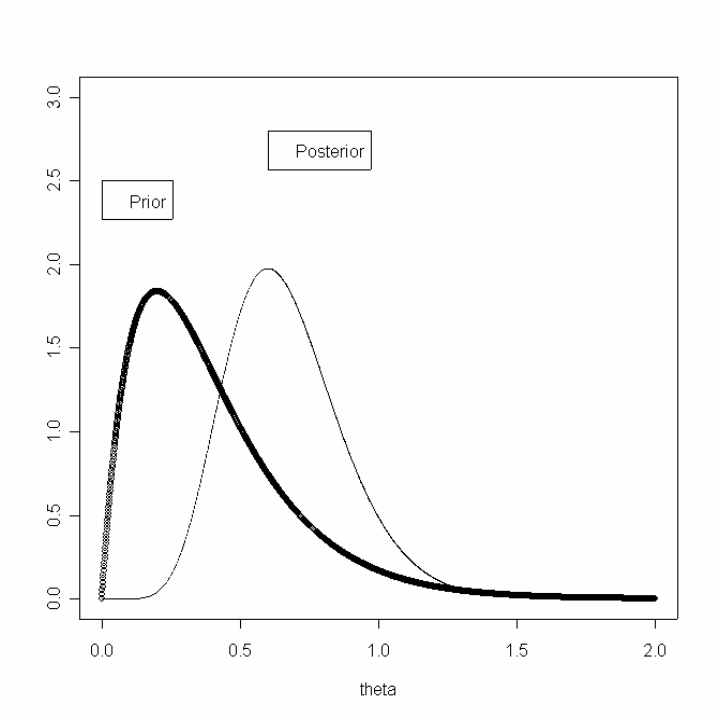


Figure 9-6

Note in Figure 9-6 shows the posterior distribution of theta shifts toward higher values (the empirical data would suggest a theta of 0.8 per thousand) but still not as high as the data would suggests. The sample size (10) is not very large, an issue we will discuss below in discussing calculating the posterior. Working with conjugate pairs greatly simplies the mathematics and modeling involved in Bayesian data analysis. Table 9-2 summarizes commonly used conjugate pairs.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Table 9-2: Conjugate Pairs

Prior	Likelihood	Posterior
Univariate Models		
Beta	Binomial	Beta
Gamma	Poisson	Gamma
Gamma	Exponential	Gamma
Normal	Normal (known variance, unknown mean)	Normal
Multivariable Models		
Dirichlet	Multinomial	Dirichlet
Multivariate Normal	Multivariate Normal (known variance matrix)	Multivariate Normal

Noninformative Priors

What do we do when we want to use a Bayesian model but are ignorant about the prior distribution, and have much to guess on, if any information? The solution to this is to use a so-called noninformative prior. The study of noninformative priors in Bayesian statistics is relatively complicated and an active area of research. Other terms used for noninformative prior include: reference prior, vague prior, diffuse prior, and flat prior.

Basically, a noninformative prior is used to conform to the Bayesian model in a correct parametric form. The choice of noninformative prior may be purely mathematical and used so the posterior distribution is proper (integrates to 1) and thus is a correct density function. Noninformative priors may be special distribution forms, or versions of common probability distributions that reflect no knowledge about the distribution of the random variable(s) in the prior model.

Here we will introduce only one specific example of a noninformative prior, because it is a prior distribution of interest in modeling proportion data and of interest in bioinformatics. If there is no specific prior information known about the proportion of something (say a proportion of nucleotide in a sequence motif), then we can use a noninformative form of the beta distribution. The beta (1,1) distribution, shown in Figure 9-7, is a flat noninformative prior that is uniform across the distribution of all proportions. This is a special case of a

Permission granted to reproduce for nonprofit, educational use.

uniform distribution (which is any distribution of uniform density). If you want to use a Bayesian model and have proportion data to model with no prior information, you should use a beta(1,1) prior. The multivariate version of the beta(1,1) is a Dirichlet model with prior parameters set to 1, and will be used in examples in coming chapters.

```
> x<-seq(0,1,by=.001)
> plot(x,dbeta(x,1,1),ylab="",main="Beta (1,1) Prior")
```

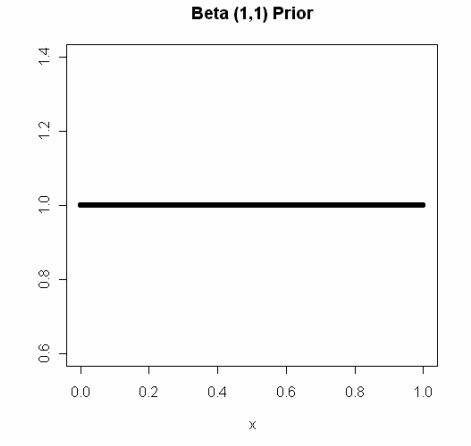


Figure 9 -7

Likelihood Functions

You may have noticed that we do not refer to the likelihood function as the likelihood distribution (which would be WRONG!) but instead call it the likelihood or the data model. There is a difference between the likelihood and a distribution model. Although, in Bayesian applications, the likelihood will often assume a form of a common probability distribution model, likelihood functions are used in many areas of statistics and are not just specific to use in Bayesian modeling.

With a distribution model we are asking what the probability of the sample outcomes are given the parameter values. For example, in the binomial model as a distribution we are asking what the probability of x successes is given the parameter value p. The parameter, p, is fixed, and the observations (x's) are variable. The data are a function of the parameters, as:

$$f(x \mid n, p) = \binom{n}{p} p^{x} (1-p)^{n-x}$$

With a likelihood function we are asking how likely the parameter values are given the sample outcomes. In essence, the interpretation of the likelihood is the reverse of the distribution. In the likelihood we already have the data and we are

Permission granted to reproduce for nonprofit, educational use.

trying to determine the parameters given the data. The parameters are a function of the data:

$$L(p \mid x, n) = \binom{n}{p} p^{x} (1-p)^{n-x}$$

In frequentist statistics the likelihood function is used to find a single value of the parameter, a so-called point estimate. A popular estimate is the value that is most likely, that is, the value that makes the likelihood function as big as possible (which you could prove by taking the likelihood and finding its maximum using calculus). We call this the maximum likelihood estimate (MLE) and it has some nice properties. The interpretation is that the MLE is the most plausible value of the parameter, in the sense that it produces the largest possible probability of the observed data occurring. Many common estimates in statistics, such as the mean and variance of a normal distribution are MLE's. Obviously making such a "true" estimate is more of a frequentist than a Bayesian thing, but it is of note here because it is an important use of the likelihood function.

Evaluating the Posterior

The whole purpose of setting up the model, determining a prior, and collecting data is of course to evaluate the posterior distribution to determine the results desired. In the examples used so far, the posterior distribution was quite simple and the trick of using conjugate pairs resulted in a familiar posterior distribution whose parameters could easily be determined by taking advantage of the property of conjugacy. Of course, this is not so easy for more complicated models, and this is where the beauty of a computational package such as R comes in. Let's take a look at some of the issues involved in the posterior outcome of the Bayesian model.

The Relative Role of the Prior and Likelihood

The posterior distribution reflects both the prior and the likelihood models, although in most cases not equally. Let's investigate the role of the likelihood and role of the prior in determining the posterior.

First, let's look at the role of the likelihood. It is pretty straightforward that the more data there are, the more the data will influence the posterior and eventually the choice of the prior will become irrelevant. For example, let's return to our example used earlier in this chapter regarding the coin tossing experiments. In this example we had a prior estimate for p of 0.25, which we modeled with a beta (2,5) prior distribution. We collected n=10 data values and obtained k=5 successes (coin landed on head) for the binomial likelihood model. The comparison of prior and posterior was shown in Figure 9-5.

Permission granted to reproduce for nonprofit, educational use.

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.

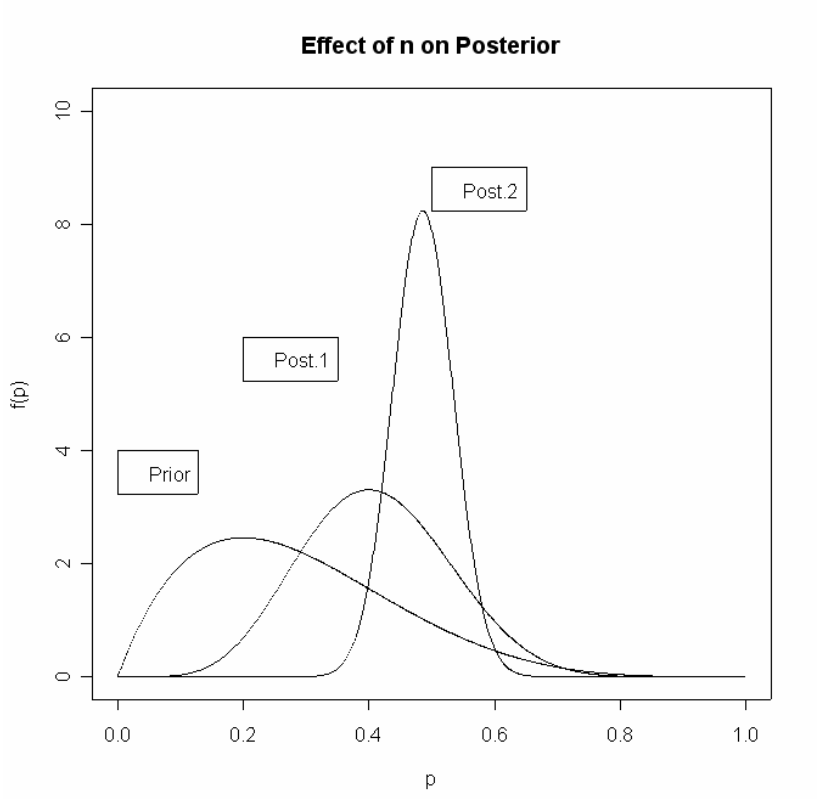


Figure 9-8

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

In terms of the role of the prior, the more vague the prior, the less the prior influences the posterior. Let's see what would happen in the example above if we used the same data for n=10 but used instead a beta(1,1) noninformative prior to start. Our posterior would follow the form of a beta distribution wit parameters alpha=6, beta=6. Let's analyze and graph this model and compare it with the prior and first posterior in Figure 9-8

```
> p <- seq(0.001, 0.999, by = 0.001)
> plot(p, dbeta(p, 1, 1), ylim = range(0:10), ylab = "f(p)",
+ type = "l", main = "Effect of noninformative prior")
> lines(p, dbeta(p, 6, 6))
> legend(0.5, 4, "Posterior")
> legend(0, 3, "Prior")
```

Figure 9-9 shows the result of the posterior when a noninformative beta prior is used. This figure is intentionally drawn on the same scale as Figure 9-8 for comparison. Posterior 1 in Figure 9-1 is updated with the same data as the posterior distribution in Figure 9-9 except in Figure 9-9 a noninformative beta prior is used. The effect of the noninformative prior is essentially no effect. When the noninformative prior is used the data dominate the model, whereas when the informative prior in 9-8 is used the informative prior influences the posterior along with the data.

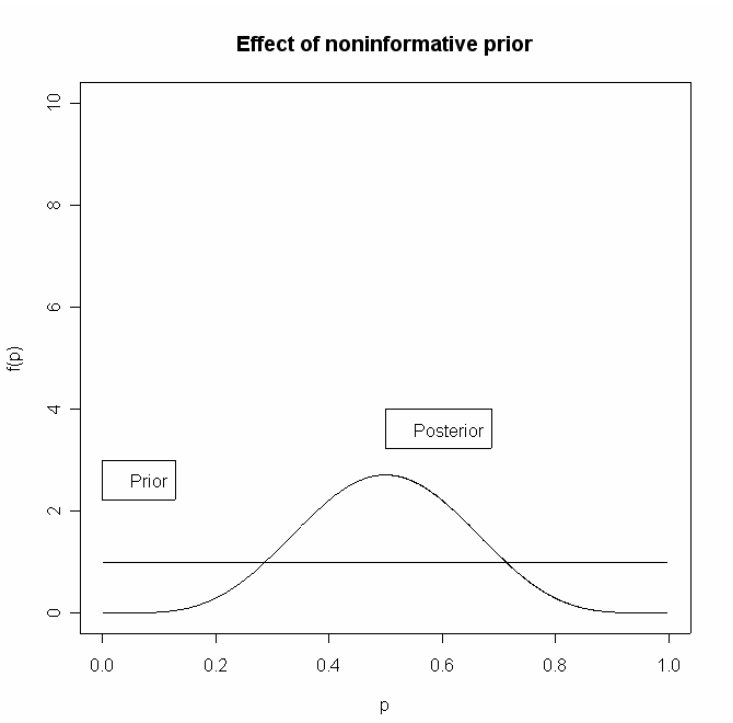


Figure 9-9

Permission granted to reproduce for nonprofit, educational use.

The important point is that in Bayesian analysis the posterior distribution will reflect with relative weights the data model and the prior model. With large datasets, the data may dominate, with a small dataset and a precise prior the prior will dominate. This relative influence of data and prior knowledge is important aspect of Bayesian analysis.

The goal in Bayesian analysis is often to determine the posterior distribution and sample from it. Using analytical methods (the paper and pencil method) without a computer it is nearly impossible to evaluate a posterior except in simple cases. However, computationally intensive methods utilizing software tools such as R and a little bit of programming skills enable us to powerfully evaluate posterior distributions. We shall see that the focus of Markov Chains and other methods are in fact tools for evaluating a posterior distribution of interest.

The Normalizing Constant

Notice that in many examples in this chapter we have used the model that was written with a proportionality factor, thus we have left out constant terms and ignored them working with the model

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

$$P(\text{model}|\text{data}) \propto P(\text{data}|\text{model}) * P(\text{model})$$

Don't forget though that with the proportionality factor the models are only given "modulo a constant" hence they are not completely defined. In order to make the models fully explicit we would need to add back in a constant term c: (Note that the notation "c" is generic. The constant c takes different values in each of the two lines below).

$$\text{Posterior} = c * \text{Likelihood} * \text{Prior}$$

$$P(\text{model}|\text{data}) = c * P(\text{data}|\text{model}) * P(\text{model})$$

Note that the constant terms, grouped as c, are called the normalizing constant and this can be calculated so that the posterior distribution integrates to 1, but it is only a scaling factor and does not affect the distribution otherwise, so it is acceptable to ignore the c term in many calculations and only calculate it when necessary.

One versus Multiparameter Models

So far we have used only one-parameter models in our discussion. Bayesian statistics can of course be utilized in models with multiple parameters, and those are of primary interest in bioinformatics. We have worked in the previous chapter with some multivariable models, and in this book we will focus examples on the multinomial-Dirichlet conjugate pair in future examples.

Permission granted to reproduce for nonprofit, educational use.

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.

Permission granted to reproduce for nonprofit, educational use.

10

Stochastic Processes and Markov Chains

This chapter begins a three chapter series devoted to the basics of Markov Chains (Ch 10), the computational algorithms involved in evaluating them for MCMC methods (Ch 11), and utilizing the program WinBugs in conjunction with R and the CODA package to be able to perform simple applications using MCMC techniques of interest to bioinformatics (Ch 12).

Markov chains have many applications in many areas of bioinformatics and other fields. Our focus in this chapter is providing an elementary introduction to Markov Chains for the purpose of utilizing them in the Markov Chain Monte Carlo methods for simulating posterior probability distributions for complex data models. Ten years ago, if you brought up the term MCMC to a room of biologists, few if any would have heard of such a technique. Today MCMC is a common buzzword especially in bioniformatics. Understanding of the mathematical techniques behind MCMC is still mostly the domain of statisticians and mathematicians. This and the next two chapters present a conceptual and applied approach to a primarily non-mathematical audience interested in being able to understand the lingo and appreciate the computational power of using MCMC. References given in the appendix for further study are provided for the interested reader.

Beginning with the End in Mind

Let's begin our study of Markov Chains by keeping the end in mind. Before introducing the basics of the MCMC technique, let's discuss what the ultimate

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

goal of MCMC technique is. The previous chapter introduced Bayesian statistics in which the major goal is to determine a posterior distribution given a likelihood data model and a prior distribution model. Some simple examples were given, and the trick of using conjugate pairs of distributions was illustrated. However, determining a posterior distribution is easier said then done and traditional analytical methods (using advanced calculus, numerical methods, etc) are often extremely difficult.

For example, let's consider what is actually a rather is simple case using a conjugate pair of a multinomial likelihood and Dirichlet prior which produce a Dirichlet posterior distribution.

The prior is:

$$f(p_1, p_2, \ldots p_K) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} p_i^{\alpha_i - 1}$$

The likelihood is:

$$f(X_1, X_2, \ldots X_K) = \binom{n}{X_1 X_2 \ldots X_K} p_1^{X1} p_2^{X2} \ldots p_k^{Xk}$$

The posterior distribution is:

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

$$\text{Posterior} = \binom{n}{X_1 X_2 \ldots X_K} p_1^{X1} p_2^{X2} \ldots p_k^{Xk} \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} p_i^{\alpha_i - 1}$$

It is easy to evaluate the Dirichlet - that's why it is used as a conjugate model. Each individual pi has a beta distribution with parameter (alpha. post = alpha + Xi, beta.post = sum (alpha_j (j not=i) + sum (Xj (j not = i). Perhaps we should use another model to illustrate "difficult to evaluate". For example, let's consider the two parameter normal model. In this example we consider what is actually a rather simple case using a normal likelihood with parameters μ and σ^2.

$$\text{Data: } X1, X2, \ldots, Xn \sim N(\text{mean}=\mu, \text{variance}=\sigma^2)$$

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

It turns out to be a bit easier for Bayesian computations to work with the inverse of the variance parameter, the so-called precision parameter $\phi = 1/\sigma^2$. We then assume the following popular independent priors:

$$\text{Priors: } \mu \sim \text{N(mean=0, variance} = \tau^2)$$

$$\phi \sim \text{gamma}(\alpha,\beta)$$

If little prior information is available, τ^2 is usually chosen to be very large. .For this example the likelihood function is, remembering that we use $\phi = 1/\sigma^2$:

$$\text{lik}(\mu,\phi) = \frac{\phi^{n/2}}{(2\pi)^{n/2}} e^{-\frac{\phi}{2}\sum_{i=1}^{n}(X_i-\mu)^2}$$

The joint prior density of the parameters μ and ϕ is:

$$f(\mu,\phi) = f_1(\mu)\, f_2(\phi) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{1}{2\tau^2}\mu^2} \cdot \frac{1}{\Gamma(\alpha)\beta^{\alpha}} \phi^{\alpha-1} e^{-\left(\phi/\beta\right)}$$

The posterior distribution is:

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

$$\text{posterior} = f(\mu,\phi \mid X_1,...,X_n) \propto \phi^{n/2} e^{-\frac{\phi}{2}\sum_{i=1}^{n}(X_i-\mu)^2} \frac{1}{\tau} e^{-\frac{1}{2\tau^2}\mu^2} \phi^{\alpha-1} e^{-\left(\phi/\beta\right)}$$

The above two examples for the posterior are not easy to evaluate, and these are simple cases of multivariable posterior distributions. Even those in love with multidimensional calculus are not very interested in performing such evaluations. Indeed, most posterior distributions of interest involving multiple variables are difficult to evaluate and have no easy analytical solution. But, the dilemma is, in bioinformatics and other areas working with high-dimensional datasets, what we want is the information in the posterior distribution – the one we can't evaluate directly!

Do not despair, because there is a clever trick. Markov Chain Monte Carlo (MCMC) methods are the golden key to producing a posterior distribution that samples can be obtained from and high dimensional data can be studied. MCMC techniques simulate a posterior that can be explored. We can then use the results to draw inferences about models and parameters.

A brief view of how MCMC works is shown in Figure 10-1. Here we simulate a bivariate normal distribution normal. The approximation starts at the origin and

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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.

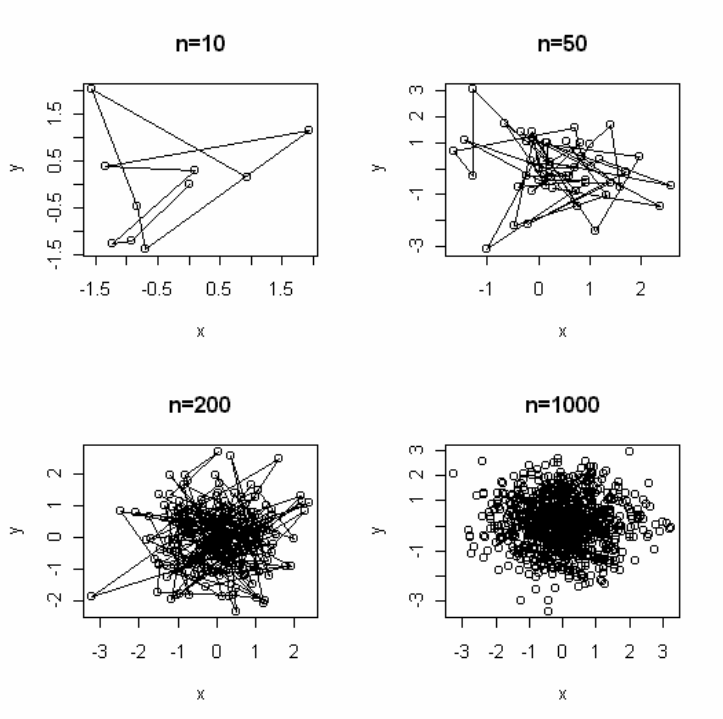


Figure 10-1: Posterior MCMC Simulation of a Bivariate Normal Distribution

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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

the power of MCMC. To take full advantage of MCMC requires much more statistical training.

The idea here is to introduce the reader to the power of this technique so that he/she may understand the terminology in the literature, he/she may be able to perform simple exploratory applications using R and auxiliary programs, and to encourage the reader interested to collaborate with statisticians who can assist them in their modeling and data analysis. Some readers might even become interested enough to pursue further formal studies in statistics (such as online graduate courese offered at the Univesity of New Hampshire).

Stochastic Modeling

As we have mentioned before, models are used to represent behaviors and properties of real world systems. A major goal of statistics is to provide useful models in order to understand such systems in particular when there is variability. The use of statistical models in bioinformatics is rapidly growing, but many other fields use such models as well. Understanding models is of increasing importance for even those not involved in directly making them because of the importance of their use.

Models can be generally classified as stochastic or deterministic. The term stochastic is used to describe uncertainty (=randomness) in the model. A deterministic model is a model where the input components of the model are not random. The output of a deterministic model is determined by the input. A deterministic model has an outcome that can be computed by direct calculation or numerical approximation.

A stochastic model is one where the input components are random. The output of a stochastic model is also random, and is a snapshot or estimate of the characteristics of a model for a given set of inputs. The results of a stochastic model are not usually determined by direct methods (analytical methods) but usually involve some simulation technique (such as Monte Carlo methods, discussed in Chapter 11). Stochastic models work well in the Bayesian paradigm.

A deterministic model is a simplified version of a stochastic model, eliminating much of the randomness. A deterministic model gives results of a single scenario, whereas a stochastic model can be used to give a distribution of results for a distribution of scenarios. Deterministic is more mathematical, stochastic is more statistical.

Permission granted to reproduce for nonprofit, educational use.

Stochastic Processes

A stochastic process is a stochastic mathematical model for a probabilistic experiment that evolves in time (or space) and generates a series of values.

Recall that in probability, a single random phenomenon (such as the value of the role of a die) is modeled using a random variable representing the outcome of an experiment. Such an experiment can be referred to as static, because such an experiment (rolling a die) does not (or at least should not) depend on time (when the die is rolled) or space (my house or yours).

In a stochastic process, the "game" changes in time or space in some way. Each step in the process is defined by a random variable, and the entire stochastic process can be defined as a set of random variables {X1, X2, X3…Xn} that are indexed by subsequent integer values (i from 1 to n). Each random variable in the sequence represents the current state of the process. The state space though (consisting of possible values of the random variables) remains the same. In a stochastic process the subsequently indexed random variables are not independent and there is some pattern of dependency from Xn to Xn+1.

There are different types of stochastic processes. Two major categories of stochastic processes are arrival-time processes and Markov processes. Arrival time processes play an important role in probability applications and include Bernoulli processes and Poisson processes. We will not cover arrival times proceses. Let's instead focus on Markov processes.

Markov Processes

A Markov process is a stochastic process where the set of random variables {X1, X2, X3…Xn…} models a certain process over time where it is assumed these random variables represent measurements (or counts) that were taken at regularly spaced time points. Markov processes exhibit a very special type of dependence. The next value (Xn+1) depends only on the current value (Xn). In other words the future evolution exhibits a probabilistic dependence on the past through the present, and ONLY the present.

Mathematically this is written in terms of conditional probability:

$$P(X_{n+1}=x_{n+1}|X_n=x_n, X_{n-1}=x_{n-1}, \ldots X_0=x_0)=P(X_{n+1}=x_{n+1}|X_n=x_n)$$

This is known as the Markov condition.

As an example of a Markov process, consider the evolution of a DNA sequence over time. In Figure 10-2 the state of the sequence depends only on the time interval prior to it (t=4 depends only on t=3, etc). This logic is the basis for some models of studying sequence evolution using Markov Chains.

Permission granted to reproduce for nonprofit, educational use.

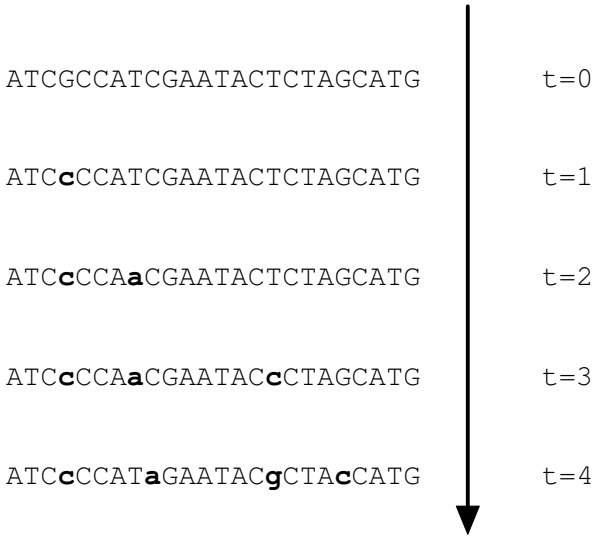


Figure 10-2: DNA Sequence Evolution as a Markov Process Over Time

The nucleotide pattern 5' to 3' in a DNA sequence is not necessarily independent and often is modeled so that the nucleotides are dependent on the nucleotide immediately upstream of them. This is an alternative Markov process model. Figure 10-3 depicts an abbreviated version of the above sequence, this time each sequential nucleotide represented as an indexed random variable. Each state of the nucleotide {A, T, C, G} is sequentially dependent on the nucleotide adjacent and immediately upstream of it, and only that nucleotide. This type of Markov model is used in analyzing sequences in non-MCMC applications, discussed very briefly at the end of this chapter.

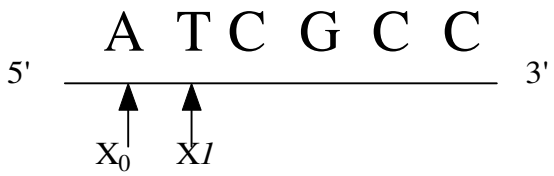


Figure 10-3: 5' to 3' Nucleotide Dependencies as a Markov Process

Classifications of Stochastic Processes

Time (Position)– Discrete or Continuous?

This refers to the index of the random variables. As we saw in the example above the index can be in terms of time or position (place, space, sequence). If the index is discrete then it's discrete time/space. If the index is continuous it is

Permission granted to reproduce for nonprofit, educational use.

continuous time/space. To avoid confusion with state space (below) it is better to use the term sequence, position or place for indexes that represent locations.

State Space – Discrete or Continuous?

This refers to the space of values for the random variable. In the discrete state space case there are finite and countable values and the random variables used are discrete random variables. For example, a discrete state space is the set of four nucleotides in a DNA sequence and modeled using random variables that will be discrete random variables (since they can only take on four distinct values). If there is a continuum of values (any value) in the state space then a continuous random variable is used and the state space is referred to as continuous.

Four Combinations of Possibilities

Of course, you can have any combination of time and state space. Here we will be concerned mostly with discrete time, discrete space Markov processes which are called Markov Chains. Towards the end of the chapter, we will discuss briefly how the discrete models generalize to continuous models.

Random Walks

A random walk is a simple Markov Chain, yet the study of random walks is a complicated subject in probability and has many aspects. Given that the system is in state x (where x can be any integer value) at time n, the system will at time n+1 move either up, or down one point (integer) usually with equal probability. The basic probability calculations for the alignment algorithm used by BLAST are based on random walks and many other powerful applications are based on this simple theory.

The simple function below simulates a random walk with respect to the y-axis using R. The x-axis is the index and represents time. At any given time a random step up or down one integer amounts to sampling with equal probabilities from the numbers (+1,-1) and adding the result to the current state. We initiate the chain with a starting value of y(t=1)=0. The results of one simulation are shown in Figure 10-4.

```
> Walk1d <- function() {
+ n <- 100
+ y <- vector(length=n)
+ y[1] <- 0
+ for (i in 2:n) y[i] <- y[(i-1)] + sample(c(-1,1),1)
+ plot(1:n,y,type='l',ylim=c(-20,20))
+ }
> Walk1d()
```


Permission granted to reproduce for nonprofit, educational use.

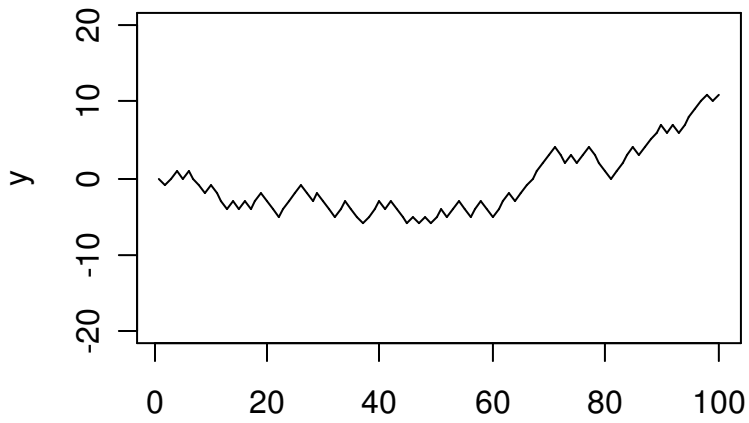


Figure 10-4: Simple Random Walk

Likewise, it is simple to simulate a two dimensional random walk in R as coded below. Four sample outcomes are shown in Figure 10-5. This code could represent (among other things) an experiment of flipping two coins. One uses the sample function to "flip" a 1 (heads) or -1 (tails). For the x and y direction if the coin is 1, you move + 1 in that direction, if the coin is -1 you move –1 in that direction. In the code below you see that we first generate 500 coin flips representing the movements in the X direction (vector "xmove") and likewise for the Y-direction (vector "ymove"). The "cumsum" function accumulates these movements.

```
> par(mfrow=c(2,2))
> Walk2d<-function(){
+ xstart <- 0
+ ystart <- 0
+ xmove <- sample(c(-1,1),500,repl=T)
+ ymove <- sample(c(-1,1),500,repl=T)
+ xmove <- xstart + cumsum(xmove)
+ ymove <- ystart + cumsum(ymove)
+ plot(xmove,ymove,xlim=c(-40,40),ylim=c(-
+40,40),xlab="x",ylab="y",type='l')
+}
> for (i in 1:4) Walk2d()
```


Permission granted to reproduce for nonprofit, educational use.

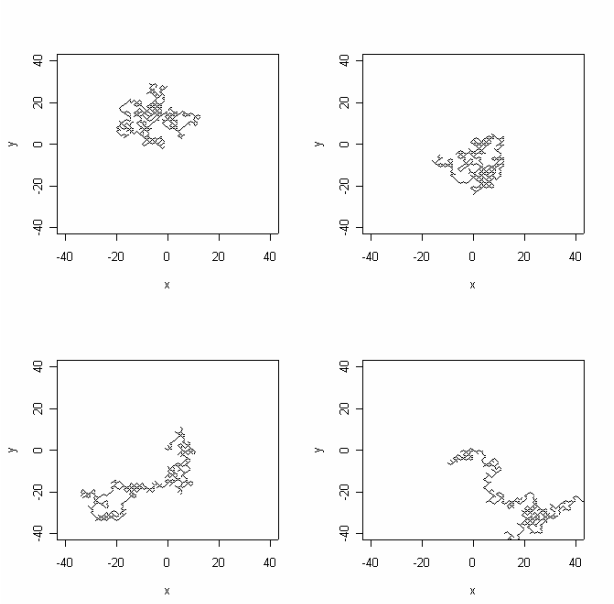


Figure 10-5: Two Dimensional Random Walk Outcomes (same simulation)

Real world applications of random walks include fractals, dendritic growth (for example, tree roots), models for stock options and portfolios, and gambling game outcomes. Random walks underlie the theory of Markov Chains. Let's now explore Markov Chain models in more detail.

Probability Models Using Markov Chains

A Markov Chain is a probability model for a Markov process that evolves in time or space. Because some matrix math is required to compute Markov Chain probabilities, we will first review how to do this using R. Then this section introduces two examples of models using Markov Chains. The first example is very simple and will be presented mostly conceptually. The second example is more complex and will be presented more mathematically. The next section of this chapter will investigate the mathematical details of Markov Chain models in more depth.

Matrix Basics

A matrix is a rectangular table of numbers. The numbers are called entries (or elements) of the matrix. The dimension of a matrix is the number of rows and columns it contains. You have likely encountered matrices before in a math course (although you may not remember them!).

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

For our purpose here, we will be concerned mostly with square matrices. Square matrices have an equal number of rows and columns. And the only operation we will be concerned with is multiplying matrices.

For example, consider matrices A and B, which are both 2 by 2 matrices:

$$A = \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}$$

$$B = \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix}$$

The elements of the matrix are indexed by ij, where i is the row number, and j is the column number. To multiply matrix A by matrix B to create a new matrix C, we multiply row 1 elements of A by corresponding elements of column 1 of B and sum the results to get the first entry in C (c11). Then we multiply row 1 elements of A by column 2 elements of B to get entry c12 of C, etc….

$$C = AB = \begin{bmatrix} a11*b11+a12*b21 & a11*b12+a12*b22 \\ a21*b11+a22*b21 & a21*b12+a22*b22 \end{bmatrix} = \begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix}$$

In R, matrices are a distinct type of object with distinct behaviors declared using the matrix function call. Matrices are NOT the same as data frames nor are they the same as arrays, which may look the same but behave differently and are different object types in R which behave very differently!. To declare a matrix in R use the matrix function below with parameters data (for the data vector), nrow (for number of rows) and ncol (for number of columns).

matrix(data, nrow, ncol)

There are other parameters that we will not be concerned with here (if interested use help(matrix) and R will show the function details). Data can be a preexisting data vector, or can be entered as a parameter directly in the function call. By default columns are filled first and then rows. Thus, if 4 data values are entered as a data parameter, the first two become the first column and the second two values become the second column.

If you have existing data and you are not sure whether it is already a matrix object, using function is.matrix(x) will return true or false depending on whether object x is a matrix or not.

Multiplying matrices is done not using "*" alone but using the special notation "*%*%*".

Let's declare two square matrices and multiply them in R

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> a<-matrix(c(1,2,3,4),nrow=2,ncol=2)
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> b<-matrix(c(4,3,2,1),nrow=2,ncol=2)
> b
     [,1] [,2]
[1,]    4    2
[2,]    3    1
> #Multiply matrix a times matrix b the CORRECT way
> a%*%b
     [,1] [,2]
[1,]   13    5
[2,]   20    8
> Note using a*b is NOT CORRECT
> a*b
     [,1] [,2]
[1,]    4    6
[2,]    6    4
```

A Simple Markov Chain Model

Suppose a frog lives alone in a small pond with two Lilly pads and we record where he is every 5 minutes. At each time point the frog (who is disabled and cannot swim but can only leap from pad to pad) is on one of the Lilly pads (state space consisting of A and B). Between the observations (every 5 minutes) the frog may move to the other Lilly pad or stay where he is.

What if we know the frog is on Lilly pad A at the initial time, and we want to know the probability that he is on pad A in 10 minutes?

We can model this scenario as follows. Our initial model suggests that during the first interval we observe (time 0 to time 1 (5 minutes)) if the frog is on pad A at time 0, the probability of the frog staying on pad A is 0.8, the probability of the frog going to pad B is 0.2 if the frog is on pad A. If the frog is on pad B, the probability of staying on pad B is 0.6 and the probability of the frog going from pad B to pad A is 0.4. This model is depicted graphically in Figure 10-6.

Permission granted to reproduce for nonprofit, educational use.

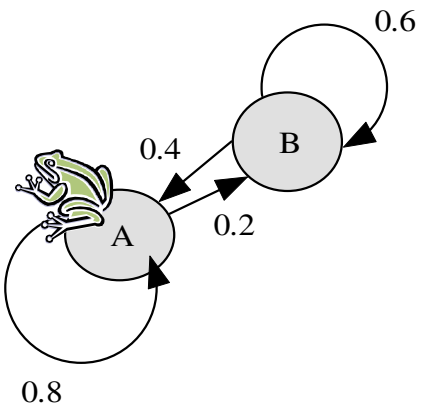


Figure 10-6: Transition Probabilities for Frog's First Move

Alternatively, we can use a matrix to model this situation for the probabilities of transition from state space 1 (time 0=0 minutes) to state space 2 (time 1=5 minutes). We call this a transition matrix. The row is the starting location and the column is the ending location for each move. The matrix below is for transition probabilities for the first move given the initial state

$$\text{Transition matrix for initial state} = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}$$

To determine how he could be on pad A in 2 time periods given he is on pad A in the initial state, we can observe that in order for this to happen, there are two possibilities. The first is that the frog is on pad A stays on pad A during the first and second transitions. The second is that the frog goes from pad A to pad B and then back to pad A.

The total probability of being on pad A after two time periods given he started in pad A is the probability of the mutually exclusive events (disjoint events) of the first and second possibilities. Each transitional event is independent (frog can go from A to A or A to B, these events are independent) so we can multiply them. Notice that in the matrix this is the first row times the first column.

P(Frog on A after two time intervals)

= P(AA)P(AA)+P(AB)P(BA)

=(0.8*0.8)+(0.2*0.4)

=0.64+0.08=0.72

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

To get the next transition matrix (t=1 to t=2) we can use R to multiply the first transition matrix by itself, using the logic described above.

```
> frog1<-matrix(c(0.8,0.4,0.2,0.6),nrow=2,ncol=2)
> frog1
     [,1] [,2]
[1,]  0.8  0.2
[2,]  0.4  0.6
> frog2<-frog1%*%frog1
> frog2
     [,1] [,2]
[1,] 0.72 0.28
[2,] 0.56 0.44
```

Figure 10-7 depicts graphically the new transition probabilities.

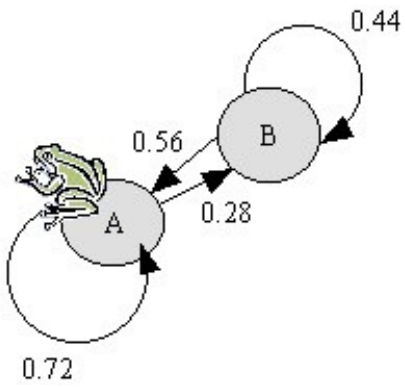


Figure 10-7: Transition Probabilities for Frog's First Two Moves

Let's consider the longer-term behavior of the transition probability matrix (which can be referred to as "the chain"). As we increase the powers of the transition matrix (time 3, time 4….) a peculiar thing happens….

```
> frog3<-frog2%*%frog1
> frog3
      [,1]  [,2]
[1,] 0.688 0.312
[2,] 0.624 0.376

> frog4<-frog3%*%frog1
> frog4
       [,1]   [,2]
[1,] 0.6752 0.3248
[2,] 0.6496 0.3504

> frog5<-frog4%*%frog1
> frog6<-frog5%*%frog1
> frog7<-frog6%*%frog1
> frog8<-frog7%*%frog1
> frog9<-frog8%*%frog1
> frog10<-frog9%*%frog1
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> frog10
          [,1]      [,2]
[1,] 0.6667016 0.3332984
[2,] 0.6665968 0.3334032

> frog11<-frog10%*%frog1
> frog12<-frog11%*%frog1
> frog13<-frog12%*%frog1
> frog14<-frog13%*%frog1
> frog15<-frog14%*%frog1
> frog15
         [,1]      [,2]
[1,] 0.666667 0.3333330
[2,] 0.666666 0.3333340

> frog16<-frog15%*%frog1
> frog17<-frog16%*%frog1
> frog18<-frog17%*%frog1
> frog19<-frog18%*%frog1
> frog20<-frog19%*%frog1
> frog20
          [,1]      [,2]
[1,] 0.6666667 0.3333333
[2,] 0.6666667 0.3333333
```

After around 20 intervals the transition matrix no longer changes. At this point, the frog's probability of going from A to A is the same as going from B to A, and B to B is the same as A to B, as depicted in Figure 10-8.

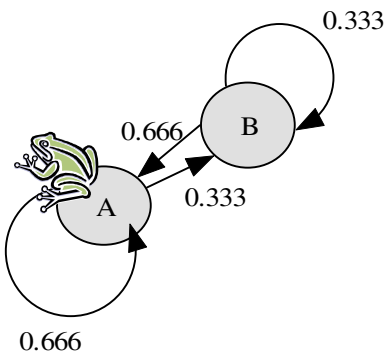


Figure 10-8: Transition Matrix for Frog's First 20 Jumps

After 20 jumps the frog's position is in a so-called stationary distribution and the chain has converged (ended) with this stationary distribution.

In the beginning we didn't really mention a "starting probability distribution" for the probability of the frog being in pad A or B at the beginning of the chain. We specified only the transition probabilities for his moving from pad to pad. We can easily incorporate such a "prior distribution" in the model.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Let's say there is an equal probability (p=0.5 for starting on A or B). We can update this distribution by multiplying it by each transition matrix.

```
> start0<-matrix(c(0.5,0.5),nrow=1,ncol=2)
> start0
     [,1] [,2]
[1,]  0.5  0.5
> start1<-start0%*%frog1
> start1
     [,1] [,2]
[1,]  0.6  0.4
> start2<-start1%*%frog2
> start2
      [,1]  [,2]
[1,] 0.656 0.344
> start3<-start2%*%frog3
> start3
         [,1]     [,2]
[1,] 0.665984 0.334016
```

By the third iteration, the starting probability distribution is converging to the stationary distribution.

What if we used a different starting probability distribution? Let's use p=0.1 for the frog starting on pad A and p=0.9 for the frog starting on pad B and see what happens when we update this with the transition matrices:

```
> altstart0<-matrix(c(0.1,0.9),nrow=1,ncol=2)
> altstart0
     [,1] [,2]
[1,]  0.1  0.9
> altstart1<-altstart0%*%frog1
> altstart1
     [,1] [,2]
[1,] 0.44 0.56
> altstart2<-altstart1%*%frog2
> altstart2
       [,1]   [,2]
[1,] 0.6304 0.3696
> altstart3<-altstart2%*%frog3
> altstart3
          [,1]      [,2]
[1,] 0.6643456 0.3356544
```

This alternative starting distribution also converges to the stationary probabilities. This can be interpreted as the process eventually "forgets" the starting condition and no matter where it starts, converges to some stationary distribution given the initial transition probabilities.

Important concepts which are illustrated with this simple example are the concept of a Markov Chain as a probabilistic model, the concepts of states of the Markov chain (Lilly pad A or B at a given time), understanding transition probabilities between states and how to use a matrix to display transition

Permission granted to reproduce for nonprofit, educational use.

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 nucleotides depend 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

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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Rows represent starting states and columns the next state. Row 1, column 1 represents the probability of going from nucleotide A in position 0 (X_0=A) to nucleotide A in position 1 (X_1=A) or P(X_1=A| X_0=A). Row 3, column 2 represents the probability of going from nucleotide C in position 0 (X_0=C) to nucleotide T in position 1 (X_1=T) or P(X_1=T|X_0=C). The transition matrix contains all information needed to compute all the probabilities for future states. Notice the sum of every row is 1, obeying the law of total probability. The columns do not have to sum to 1 and usually do not.

In addition to the matrix we can use a graphical representation of the one-step transition probabilities as illustrated in Figure 10-9:

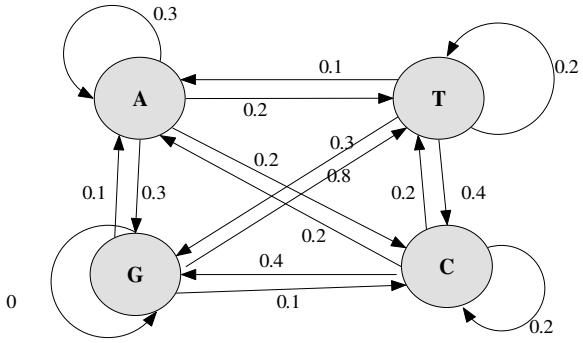


Figure 10-9

Determining Convergence and Stationary Distributions

Let's go ahead and enter this model in R to calculate a stationary distribution using R's capabilities to multiply matrices:

```
> DNA<-
matrix(c(0.3,0.1,0.2,0.1,0.2,0.2,0.2,0.8,0.2,0.4,0.2,0.1,0.3,0.3,0.4,0),
+ nrow=4,ncol=4)
> DNA
     [,1] [,2] [,3] [,4]
[1,]  0.3  0.2  0.2  0.3
[2,]  0.1  0.2  0.4  0.3
[3,]  0.2  0.2  0.2  0.4
[4,]  0.1  0.8  0.1  0.0
```

Multiplying matrices by powers of 2:

```
> DNA2<-DNA%*%DNA
> DNA4<-DNA2%*%DNA2
> DNA4
       [,1]   [,2]   [,3]   [,4]
[1,] 0.1567 0.3512 0.2424 0.2497
[2,] 0.1557 0.3476 0.2454 0.2513
[3,] 0.1572 0.3560 0.2380 0.2488
[4,] 0.1533 0.3458 0.2529 0.2480
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> 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:

$$\pi = \begin{bmatrix} 0.15 & 0.35 & 0.25 & 0.25 \end{bmatrix}$$

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).

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Characteristics of a Markov Chain

Markov Chains can be characterized by certain behaviors and properties. When using MCMC, we are interested in chains that result in a stable and unique stationary distribution. Not all Markov chains have properties we want in order to be usable for determining a unique posterior distribution! In order for a chain to converge to a unique state it must be ergodic, that is be aperiodic and irreducible. These terms and other important terms are explained below.

Finiteness

Finiteness means that there are a finite number of possible states. Clearly in the case of the Lilly pads there were only 2 states, and in the case of a DNA sequence there are only 4 states. These systems are finite. Finite does not necessarily mean small, it only means the number of possible states is not infinite. We are only interested in Markov models where the state space is finite.

Aperiodicity

Aperiodicity means the chain is not periodic. In other words, the chain does not behave like a sine wave and keep visiting the same states over and over again.

Consider a three state chain {A, B, C} where the only possible -transitions are A->B, B->C and C->A as in Figure 10-10.

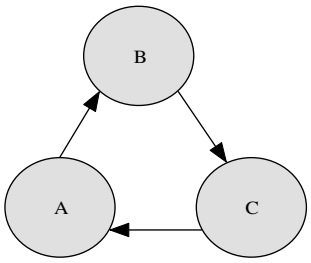


Figure 10-10

This chain is periodic because if we observe that the chain is in state A at time 0, then in time 3 the chain will be in state A again because this is the only route possible given the allowable transitions (note there is not a transition to stay in the same state). Again in time 6 and time 9 the chain will be in state A. This is a very predictable and periodic pattern. Since the greatest common divisor of these is 3, we say that this chain has period of 3.

We could work with this chain in R as follows:

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> periodic<-matrix(c(0,1,0,0,0,1,1,0,0),nrow=3,ncol=3)
> periodic
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    1    0    0
[3,]    0    1    0

> periodic1<-periodic%*%periodic
> periodic1
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    0    0    1
[3,]    1    0    0

> periodic2<-periodic%*%periodic1
> periodic2
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1

> periodic3<-periodic2%*%periodic
> periodic3
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    1    0    0
[3,]    0    1    0
```

If we keep going the chain will just rotate around and around without ever converging to a unique distribution. This is NOT a behavior we want for a chain we use for determining a posterior distribution. Instead, we want chains that are aperiodic.

Irreducibility

Irreducible means that every state can be reached from every state. To illustrate this with a counter example, consider the chain in Figure 10-11.

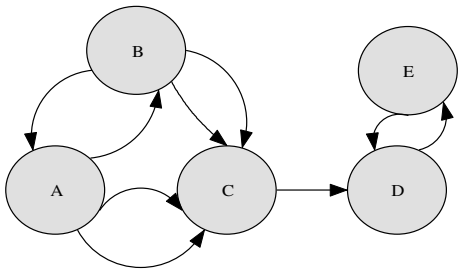


Figure 10-11

Look at the figure and ask yourself the simple question, how do you go from state E to state A? The answer is that you can't. If in a state space the transition

Permission granted to reproduce for nonprofit, educational use.

probability of going from one state to another state is zero, then the state space is reducible.

An example of what a stationary state transition matrix for a reducible state space {A, B, C, D} would look like is:

$$\begin{pmatrix} 0.4 & 0.6 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 0.4 & 0.6 \\ 0 & 0 & 0.3 & 0.7 \end{pmatrix}$$

In this case there are two stationary distributions. In a reducible system there does not exist a UNIQUE stationary distribution. Therefore we desire our systems to be IRREDUCIBLE when using them for determining a unique posterior distribution.

Ergodicity

A Markov chain, which is BOTH aperiodic and irreducible, is ergodic. An ergodic chain has a unique stationary distribution that is positive on all states when the number of states is finite.

Mixing

A Markov chain has good mixing properties if it moves fluidly through all possible states. Good mixing means the chain does not get "stuck" for periods of time around one state. We will analyze mixing when we look at time series plots of Markov chains using WinBugs as mixing is a diagnostic tool for chain behavior.

Reversibility

Reversibility is a special and somewhat mathematically magical property of a Markov Chain. It is NOT necessary for a chain to be reversible in order to have a unique, stable, stationary distribution. In other words, an ergodic chain can be reversible, but does not have to be reversible. However, some of the algorithms we will use take advantage of reversibility.

An ergodic Markov chain is reversible if it satisfies the detailed balance equation:

$$\pi(x)p(x, y) = \pi(y)p(y, x)$$

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

In the detailed balance equation for a chain with state space S, x is the current state of the system and y is the state at the next step. The transition probability from x to y is p (x, y) and the transition probability of going from y to x is p(y,x). If the system has more than two states the detailed balance holds for any two states in the system's state space.

In essence the property of reversibility means that the probability of going forward in time (or space) equals the probability of going back in time (or space) for a given state space pair.

Let's look at a simple example of a reversible Markov Chain. Going back to the frog example, what if the one-step transition matrix for the frog is as follows:

$$P^0 = \text{Start} \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{bmatrix}$$

Although it takes a while (about 60 transitions, try it in R and check) the stationary matrix for this system is:

$$P^n = \text{Start} \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

Thus the unique invariant distribution for states A and B is:

$$\pi = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$$

To check for reversibility we check the detailed balance equation:

$$\pi(A)p(A, B) = \pi(B)p(B, A)$$

$$0.5*0.9 = 0.5*0.9$$

Therefore this system is reversible. You could take the stationary matrix and perform some calculations to travel backward through the chain and you would return to the starting state.

The Stationary State

The stationary state is the gold in the mining process of Markov chain models. The stationary state is also referred to as the target distribution, the limiting distribution, and various other terms. In the Bayesian sense, the stationary state is the posterior distribution (or the posterior distribution of interest can be determined using the stationary state).

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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 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

Permission granted to reproduce for nonprofit, educational use.

area where R and WinBugs can be very useful tools and applications presented in this book will focus on this type of analysis.

However, it is of note here that Markov Chains play important roles in many other applications in bioinformatics which may be stand-alone Markov Chain applications and do not utilize MCMC techniques. For these applications, the interest is not in posterior distribution determination. For example, there are many applications of Markov Chains in sequence analysis, and the use of Markov Chains for additional applications is increasing all across bioinformatics related disciplines. Many books of interest are listed in the appendix for those interested in further exploration of the use of Markov Chains in sequence analysis; in particular the book by Durbin is devoted to this subject.

The study of Markov Chains is a rich mix of algorithmic, mathematical and statistical methods and only the basics have been introduced here. However the basic properties of Markov Chains introduced in this chapter apply to other types of applications and should serve as a foundation for further study.

A Brief Word About Modeling

Sometimes you may start feeling a little suspicious about whether stochastic modeling and Bayesian methodology is "correct". This is especially true if you come from a background using primarily analytical and more "exact" methods of measuring and working with data.

Keep in mind, the goal of any type of mathematical modeling is to approximate "reality" in order to understand some phenomena. A major area of modern statistics is to develop better models that more accurately reflect reality, an ongoing process where models are iteratively refined. To be useful, it is not always necessarily that a model be extremely accurate. In statistics, models take randomness and some inaccuracy into account. In many cases, models are the best system we have for understanding phenomena, especially phenomena that cannot be accessed or manipulated empirically. An emerging area of bioinformatics involves modeling systems at a higher level (systems biology is already emerging as a science in and of itself) and statistical models will play a key role in these models.

One of the goals of this book is to impart upon the reader an understanding of the basics of some of the statistical models in use. We focus on methods of modeling and on how R can be used as a software tool to work with these models. At this point, we have introduced general probability models using one or many variables, and the basics of stochastic models. We will later work with linear models and methods of organizing large data sets in meaningful ways.

Permission granted to reproduce for nonprofit, educational use.

11

Algorithms for MCMC

This is a chapter introducing some commonly used algorithms that take advantage of Monte Carlo methods to simulate distributions. There are many algorithms and many variants of algorithms to perform these tasks, but our focus here will be on a few specific algorithms often used in bioinformatics. These algorithms can easily be used in R and are relatively mathematically simple yet extremely versatile.

We have seen already that using R it is easy to simulate draws from common distributions with either R's built-in functionality in the base package (as in rnorm, rgamma, etc) or using supplementary functions either written directly or using a pre-written package (as in the multinomial and Dirichlet simulations in chapter 8).

However, we have also introduced the powerful techniques of Bayesian statistics and the idea of a complicated high-dimensional posterior distribution that is difficult to determine. In bioinformatics, this is the type of distribution we are interested in applying in order to solve complex problems. Determining solutions for such complex posterior distributions is beyond the limits of traditional analytical mathematical methods, and hence the need for Monte Carlo simulation. We have introduced Markov Chains as a method of approximating a posterior distribution.

The goal now is to work with Bayesian methodology and Markov Chain techniques to be able to simulate and study posterior distributions of interest. This chapter introduces the algorithmic approaches that are used to do this, and the next chapter introduces some applications where we will explore complicated posterior distributions using some examples from genetics. To do

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

so, we will use the program WinBugs and the package CODA in conjunction with R.

WinBugs utilizes the Gibbs sampler, one of the algorithms covered in this chapter. It is possible for a user to work in WinBugs without understanding the algorithmic background the program uses, but not wise or advantageous. Therefore a focus of this chapter is to cover the theory of why the program works. However, the algorithms introduced here have stand-alone value and are used for many other applications in bioinformatics as well.

Monte Carlo Simulations of Distributions

Monte Carlo method in general means any technique for obtaining solutions to problems using random numbers. The name obviously derives from the famous casino resort on the French/Italian Riviera. Random numbers were first studied in the context of games of chance and gambling. Monte Carlo basically means simulating random outcomes. Before the availability of fast computers methods of generating random numbers were unreliable or tedious. However, in the recent past because of the development of the personal computer and the exponential increase of processor speed (over time) Monte Carlo methods have experienced explosive growth. Monte Carlo techniques are now standard in statistics and all sciences to solve many types of problems. Our problem of interest here is using Monte Carlo methods to simulate probability distributions. This section explores two methods of simulating simple distributions. The next two sections introduce more complicated algorithms for Monte Carlo simulation and the last section of this chapter ties it all together.

Inverse CDF Method

The inverse CDF method, or inverse cumulative distribution function, method is the simplest method to simulate a probability distribution directly. It allows us to utilize a computer's built-in random number generator (technically called pseudo-random number generator). A random number generator is an algorithm that produces random sequences of the digits 0,1,2,…,9 – imagine repeated shuffling of these digits and then drawing one digit at a time between shuffling. Any real number between 0 and 1 can be represented as a sequence of such digits (up to a certain precision), hence it can be generated using the random number generator. If many such numbers are generated the resulting sample is that of a uniform distribution with range [0,1]. In R this is the command "runif". The inverse CDF method transforms a set of uniform(0,1) random draws so that the resulting values represent random draws from any desired distribution.

Recall from chapter 6 the discussion of cumulative distribution functions. ecall also from precalculus mathematics that if f is a 1 to 1 function with domain A

Permission granted to reproduce for nonprofit, educational use.

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.

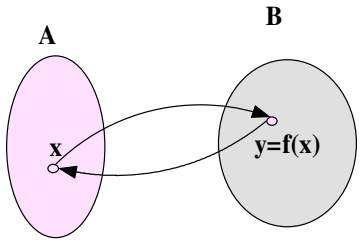


Figure 11-1

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

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

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

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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Therefore we can write a simple function in R to simulate an exponential distribution. We can call this function with the desired n, the number of values to be randomly drawn, and parameter lambda desired.

```
> simExp_function(n,lambda){
+ u<-runif(n)
+ x<-(-1/lambda)*log(1-u)}
```

Doing this to simulate an exponential with lambda=2 using 1000 draws (n=1000) produces a simulated distribution using the inverse CDF method which appears exactly as a simulated distribution of the exponential using R's command "rexp" with the same parameters does, as in Figure 11-2. In fact R utilizes the inverse CDF method internally to provide simulation of several distributions. However the inverse CDF method is easy to implement when faced with a new distribution that is not available in software.

```
> x1<-simExp(1000,2)
> x2<-rexp(1000,2)
> par(mfrow=c(1,2))
> hist(x1,30,main="Inverse CDF")
> hist(x2,30,main="Direct Simulation")
```

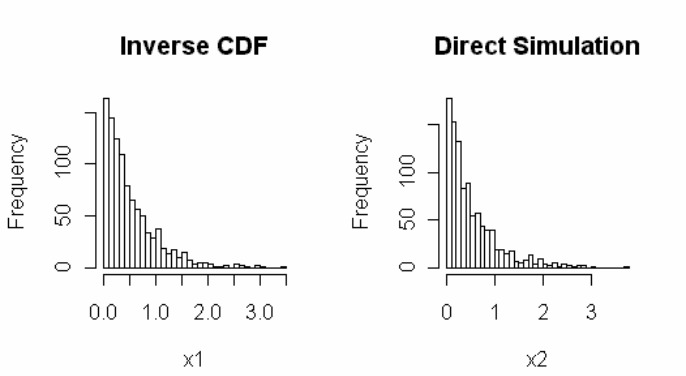


Figure 11-2

Rejection Sampling

The second method of using Monte Carlo methods to simulate a distribution that we will look at is called rejection sampling. Rejection sampling can be useful to simulate a distribution when the inverse CDF function is unobtainable or complicated. This is the case even with some univariate distributions such as the beta and gamma.

The general strategy with rejection sampling is to sample instead from another appropriate distribution and then use a correction mechanism to redirect the sample to make it approximately representative of the distribution of interest. Rejection sampling is a technique of approximating a distribution using another close distribution for the actual sampling and a rule to determine (accept or

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

reject) which values sampled approximate the distribution of interest. The distribution sampled from is sometimes called an envelope distribution, since it must frame the distribution of interest and thus contain all values for the distribution of interest.

Rejection sampling is best illustrated with an example. Let's simulate via rejection sampling a beta distribution with alpha=2 and beta=2. Mathematically this distribution is given by its pdf as follows.

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

For alpha=2, beta=2 this is

$$f(x) = 6 * x * (1-x)$$

We note from the plot of a beta (2,2) in Figure 11-3 that the maximum of the beta curve is 1.5. Therefore we can use a multiple of the uniform pdf with y=1.5 over the range of x from 0 to 1 as the envelope for the beta distribution. Here we simply sample from a uniform(0,1) ignoring the multiplicity factor 1.5. .

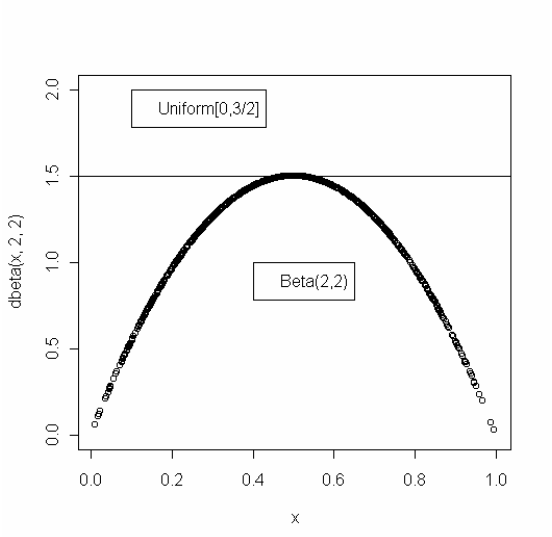


Figure 11-3

The method we will use to do the rejection sampling is as follows:

1. Draw a sample x from the distribution whose pdf is a multiple of the envelope – in our case we are sampling from (uniform [0,1]) . The multiplier is M=1.5 (to be used in step 4)
2. Draw a sample u from the uniform 0 to 1

Permission granted to reproduce for nonprofit, educational use.

3. Compute f(x) – that is beta(2,2) pdf evaluated at the value x

4. Compare if f(x) > M*u then accept x as a valid value. Otherwise reject x and repeat steps 1 through 4.

In R we can code these four steps as follows, recording accepted values as "acceptx". Here we give an example of 5000 random draws:

```
> x <- runif(5000,0,1)  # Step 1
> u <- runif(5000,0,1)  # Step 2
> fx <- dbeta(x,2,2)   # Step 3
> acceptx <- x[fx > 1.5*u]   # Step 4
```

Since a picture can speak a thousand words, let's do a plot comparing where the accepted and rejected values lie:

```
> hist(acceptx,prob=T,ylim=c(0,1.8))
> points(x,fx)
```

Figure 11-4 is convincing evidence that the rejection sampler works.

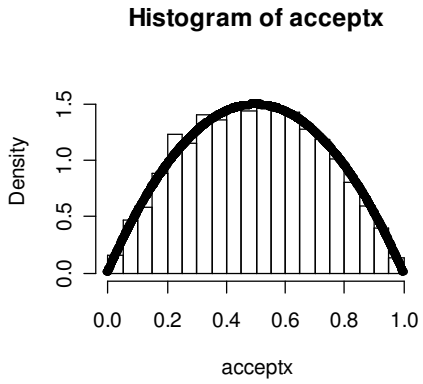


Figure 11-4

The Gibbs Sampler

In 1984, Geman and Geman introduced the Gibbs sampler in the context of image restoration using Bayesian thinking, Besag(1974) used the Gibbs sampler for spatial statistics but it wasn't until 1990 when Gelfand and Smith first used this sampler in the context of Bayesian statistics. Since then, probably no other algorithm has received so much attention and become so popular in statistical applications. The Gibbs sampler is remarkably simple and versatile, yet powerful.

Basically, the Gibbs sampler is a tool for sampling from a multivariate distribution, say of a m-dimensional set of variables U1,…,Um. It does so by

Permission granted to reproduce for nonprofit, educational use.

iterating through the individual variables Ui by sampling from the full conditionals f(Ui| U(-i) and then updating, and repeating this for all Ui. The full conditionals are simply the univariate conditional distributions of any one component Ui given the values of all the remaining components denoted by U(-i)

In the Bayesian context we are interested in sampling from the multivariate posterior distribution of all parameters. Here the full conditional is the conditional distribution of a single parameter given the data and all the other parameters. We have mentioned in earlier chapters that one way to study high dimensional distributions is to look at marginal and conditional distributions of parameters. Indeed, many times this is the only way to look at such distributions especially beyond 3 dimensional distributions since it impossible to visualize or graph higher-dimensional distributions.

The conditional and marginal distributions of parameters of interest have much simpler forms. A high dimensional posterior distribution can be broken down into conditional distributions for each of the parameters of interest. We then can look at marginal and conditional distributions of individual parameters. For example, let's say we have a hypothetical distribution which has three parameters that we will call A, B, and C. Together these parameters have posterior distribution p(A,B,C) which is a joint distribution. The Gibbs sampler enables us to obtain samples from this distribution by simply using the three full-conditional distributions: p(A|B,C), p(B|A,C) and p(C|A,B). Each of these is a one-dimensional distribution that is much easier to work with than the joint distribution. Ultimately we will not be studying the joint posterior distribution of the sampled values but rather marginal distributions of individual parameters since that is what we are mainly interested in.

The Gibbs sampler works by beginning with an initial state of the distribution and iterating through the distribution by updating each of the full conditionals until the distribution reaches a stable, converged posterior distribution. Essentially the Gibbs sampler is a Markov Chain process, illustrated in Figure 11-5 for a two parameter, x and y scenario. The sampler goes from state X to state Y iteratively using Markov dependence and the full conditionals represent the Markov transition probability distributions.

To illustrate use of the Gibb's sampler we present two examples. The first is a toy example using basic probability. In this case the posterior distribution of interest is known, but all of the steps can be illustrated so this example is presented as a review of probability and as an illustration of the Gibb's sampler, which can be easily followed. The second example is using the Gibb's sampler to simulate the posterior of a bivariate normal distribution, a more complex situation.

Permission granted to reproduce for nonprofit, educational use.

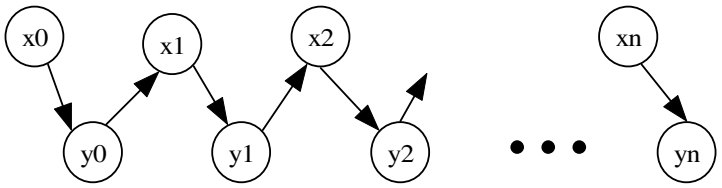


Figure 11-5

A Gibbs Example Using Simple Probability

To illustrate the Gibbs sampler, let's do a simple experiment using only basic probability distributions. Suppose we are interested in two parameters, A and B, whose joint distribution is known and illustrated in Table 11-1

Table 11-1

		A		
		1	**2**	**3**
B	**1**	0.3	0.2	0.1
	2	0.2	0.1	0.1

Recall that a marginal probability is the probability of a parameter considering only values of that parameter across all values of other parameters. The marginal probabilities of A are illustrated in Table 11-2.

Table 11-2

A		
1	**2**	**3**
0.5	0.3	0.2

The marginal probabilities of B are illustrated in Table 11-3.

Table 11-3

B	**1**	0.6
	2	0.4

Recall that a conditional probability distribution is the value of one parameter given a particular value of another parameter. From the joint distribution the full conditional distribution of B given A is given in Table 11-4. For example, in the upper left cell the probability that B=1 GIVEN A=1 is 0.6.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Table 11-4

		A		
		1	**2**	**3**
P(B|A)	**1**	0.6	0.667	0.5
	2	0.4	0.333	0.5

Likewise, the full conditional distribution of A given B is presented in Table 11-5. This time the value in the upper left cell is the probability that A=1 GIVEN B=1 is 0.5. If this does not make sense, you should review the coverage of conditional and marginal probabilities presented in chapter 8.

Table 11-5

		P(A|B)		
		1	**2**	**3**
B	**1**	0.5	0.333	0.166
	2	0.5	0.25	0.25

Here is a little program in R, which uses the Gibb's sampler to compute the marginal distributions of A and B, by sampling from the full conditionals.

First we initialize all variables involved.

```
> #initialize a and b to (1,1) first row, first column
> a<-1
> b<-1
> #initialize variables to hold results
> margA<-NULL
> margB<-NULL
> joint<-matrix(c(0.3,0.2,0.2,0.1,0.1,0.1),nrow=2,ncol=3)
> samples<-matrix(0,nrow=2,ncol=3)
```

Next, run a loop of 1000 runs which chooses values for the variable conditioned on using the sample () function (not to be confused with the samples results matrix variable!) and then stores the selection in the appropriate marginal results variable using the append function.

```
> for(i in 1:1000){
+ #sample a given value of b
+ #sample comes from full conditional dist of b given a
+ b<-sample(c(1,2),1,prob=joint[,a]/sum(joint[,a]))
+ #sample b given value of a from above
+ #sample comes from full conditional dist of a given b
+ a<-sample(c(1,2,3),1,prob=joint[b,]/sum(joint[b,]))
+ margA<-append(margA,a)
+ margB<-append(margB,b)
+ samples[b,a]<-samples[b,a]+1}
```


Permission granted to reproduce for nonprofit, educational use.

The samples matrix contains the counts of a's and b's from the sampler above:

```
> samples
     [,1] [,2] [,3]
[1,]  288  206  117
[2,]  194   95  100
```

Converting the samples data to probabilities and comparing with the original joint distribution shows the results obtained using the iterative Gibb's sampling from the conditional distributions reproduced the joint distribution.

```
> sampleProp<-samples/1000
> sampleProp
      [,1]  [,2]  [,3]
[1,] 0.288 0.206 0.117
[2,] 0.194 0.095 0.100
> joint
     [,1] [,2] [,3]
[1,]  0.3  0.2  0.1
[2,]  0.2  0.1  0.1
```

Figure 11-5 shows the marginal distributions of A and B produced by the sampler. Compare these results to the marginal distributions in Tables 11-2 and 11-3.

```
> par(mfrow=c(1,2))
> hist(margA,main="Marg.Dist of A")
> hist(margB,main="Marg.Dist of B")
```

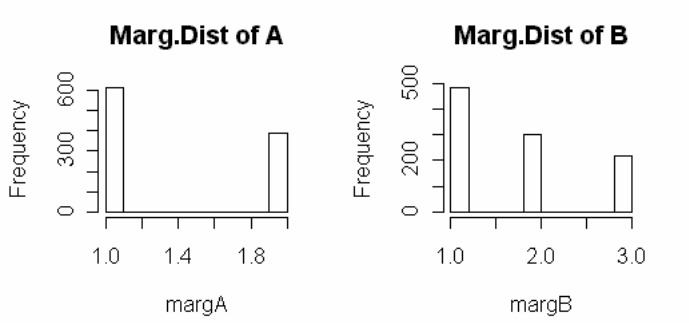


Figure 11-5

A Gibbs Example Using the Bivariate Normal

Recall from chapter 8 the discussion of the bivariate normal distribution as the joint distribution of two normally distributed variables, which we can refer to as X and Y. In chapter 8 we only considered the case where X and Y are independent. Let's look at slightly more complicated bivariate normal distribution where X and Y are correlated. That is, there is a statistical relationship between X and Y. Correlation is measured by a correlation coefficient, usually symbolized by the Greek letter rho (ρ). If rho=0 then the variables are uncorrelated. If rho=1 then there is a perfect correlation between X

Permission granted to reproduce for nonprofit, educational use.

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(ncol=2,nrow=n)
+
+ #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

Permission granted to reproduce for nonprofit, educational use.

and Y will be normal distributions with mean of 0 and variance (=standard deviation) of 1.

Let's run this function with different starting values for X and Y, using rho=0.

```
> par(mfrow=c(2,2))
> startX0Y0<-gibbsBVN(0,0,200,0)
> startX5Y5<-gibbsBVN(5,5,200,0)
> startXn5Y5<-gibbsBVN(-5,5,200,0)
> startXn5Yn5<-gibbsBVN(-5,-5,200,0)
```

Figure 11-6 shows that no matter where you start, after 200 runs the joint distribution simulated using the Gibbs algorithm appears the same as if started from (0,0).

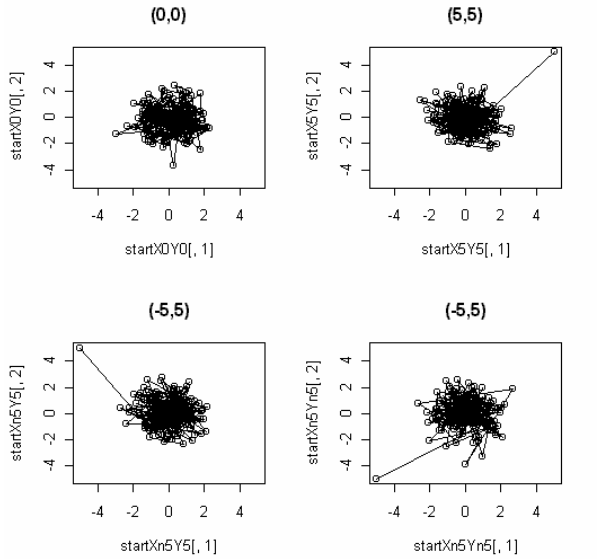


Figure 11-6

Next, let's see how the sampler behaves using different correlation coefficients (rho values). With a plot of these four values in Figure 11-7.

```
> corr0<-gibbsBVN(0,0,1000,0)
> corr3<-gibbsBVN(0,0,1000,0.3)
> corr5<-gibbsBVN(0,0,1000,0.5)
> corr98<-gibbsBVN(0,0,1000,0.98)
> par(mfrow=c(2,2))
> plot(corr0[,1],corr0[,2],main="XYCorr=0")
> plot(corr3[,1],corr3[,2],main="XYCorr=0.3")
> plot(corr5[,1],corr5[,2],main="XYCorr=0.5")
> plot(corr98[,1],corr98[,2],main="XYCorr=0.98")
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

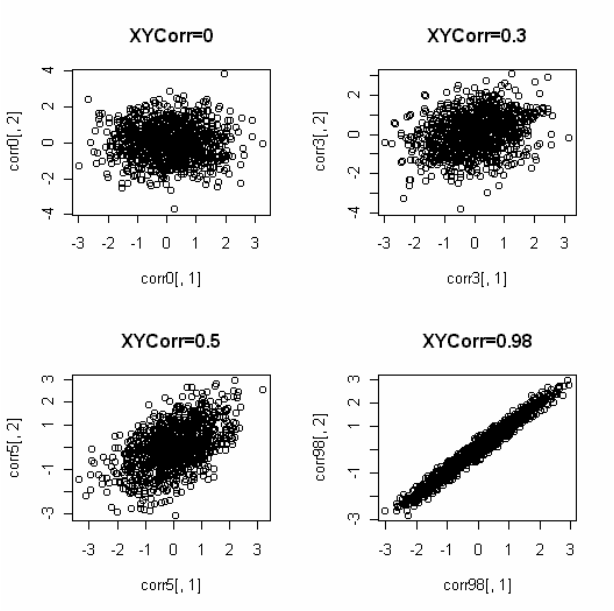


Figure 11-7

Note in Figure 11-6 that the bivariate distributions using different correlation coefficients show the effect of correlation on the relationship of X and Y. Let's plot marginal distributions for some of these distributions.

```
> par(mfrow=c(2,2))
> hist(corr3[,1],nclass=20,main="X Marg, Corr=0.3")
> hist(corr3[,2],nclass=20,main="Y Marg, Corr=0.3")
> hist(corr98[,1],nclass=20,main="X Marg, Corr=0.98")
> hist(corr98[,1],nclass=20,main="Y Marg, Corr=0.98")
```

Note in Figure 11-8, which shows plots of selected marginal distributions, that all of the marginal distributions show a standard normal distribution pattern. This is not only a small study in how correlation coefficients affect the bivariate normal, but also an important point about the Gibb's sampler. When examining individual variables using the Gibb's sampler's output one essentially obtains results regarding the marginal distributions of parameters of interest. Although in this particular example we can look at the joint distribution, since it is only two dimensional, in many cases of higher-dimensional distributions we cannot study directly the joint distribution. Usually this does not matter since the marginal distribution gives us the information we need. Note in Figure 11-6 if you look only at the X or Y axis and the distribution of data along one axis, it is normally distributed and the joint distribution pattern and correlation of variables do not matter when we are studying X or Y individually

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

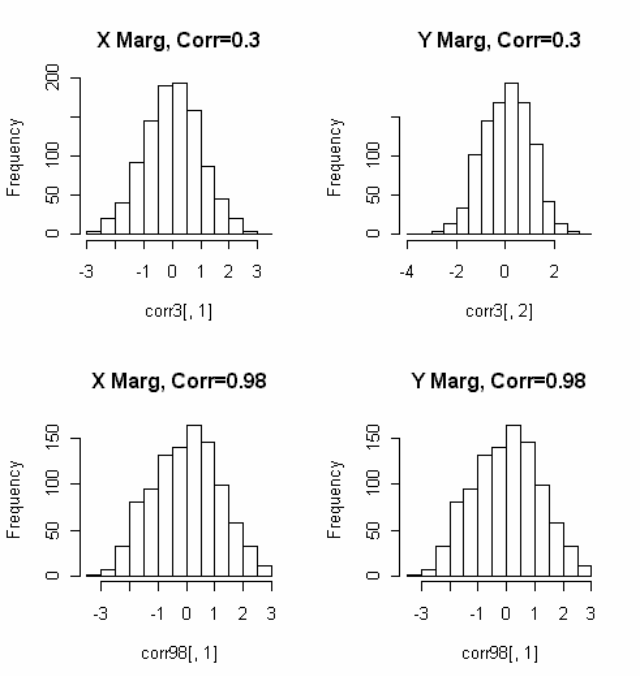


Figure 11-8

Generalized Gibbs Sampler

The Gibbs sampler can be written as a general algorithm. Our interest is in any multivariate distribution of the *n* parameters θ1, θ2, θ3,… θn. which we write as p(θ1, θ2, θ3,… ,… θn). To perform the Gibbs sample follow the following algorithm:

1. Initialize the parameters $\theta=(\theta1^{(0)}, \theta2^{(0)}, \theta3^{(0)},\ldots \theta n^{(0)})$
2. Simulate a value for $\theta1^{(1)}$ from the full conditional $\theta1|\theta2^{(0)}, \theta3^{(0)},\ldots \theta n^{(0)}$
3. Simulate a value for $\theta2^{(1)}$ from the full conditional $\theta2|\theta1^{(1)}, \theta3^{(0)},\ldots \theta n^{(0)}$
4. …..(Simulate values for θ3 through θn-1)
5. Simulate a value for $\theta n^{(1)}$ from the full conditional $\theta n|\theta1^{(1)}, \theta2^{(1)},\ldots \theta n\text{-}1^{(1)}$
6. Repeat 2 through 5 until a stationary distribution of the parameters is reached.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The Metropolis-Hastings Algorithm

The Gibbs sampler assumes sampling from the full conditional distributions if it can be done. Usually the Gibbs sampler will be useful, but if the full conditional distributions cannot be sampled from there needs to be an alternative. Even when feasible it is not necessarily easy to use the Gibbs sampler. In this case a more general algorithm, the Metropolis-Hastings algorithm can be of help. This algorithm uses a method that is similar to the rejection method algorithm. The Gibb's sampler is actually a special case of this algorithm where the sampled value is always accepted. Here we look at both the Metropolis Algorithm and its modified and more versatile form, the Metropolis-Hastings Algorithm.

For both algorithms we will be concerned with two distributions. One distribution, called a proposal distribution, we will use only to initially sample from. The proposal distribution is in essence a transition probability distribution as we discussed in the context of Markov chains. The choice of its particular form is flexible. Often there are many proposal distributions that can be used – pick the simplest one! Sometimes the proposal distribution is called a jumping or sampling distribution.

The other distribution is our posterior distribution of interest, which we will refer to as the target distribution. This is our distribution π or stationary distribution we get as a result of an ergodic Markov chain. The idea here is that we simulate values from the proposal distribution and apply criteria whether to accept or reject the simulated values. The beauty of the Metropolis algorithm lies in the fact that for the acceptance/rejection criterion we only need to calculate ratios of the posterior densities, which is much easier than calculating the posteriors densities themselves or simulating from the joint posterior distributions. For example the normalizing constant for posterior distributions can be ignored since it cancels out when using ratios. In general terms (which will be translated into specific distributions later) what we do is:

1. Generate a new value from the proposal distribution, call it θ^*
2. Calculate the ratio of the posterior of the new value to the old value =
   $$R = \frac{p(\theta^* \mid data)}{p(\theta^{old} \mid data)}$$
   Note since $p(\theta \mid data) \propto f(data \mid \theta)\, p(\theta)$
   this is equals $$R = \frac{f(data \mid \theta^*)\, p(\theta^*)}{f(data \mid \theta^{old})\, p(\theta^{old})}$$
3. Draw a uniform [0,1] value, u
4. Accept the new value θ^* if u < min (1, R). That is, accept the proposed value if the uniform value is less than the minimum of 1 and R.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The purpose of the simulation is to find values of the posterior distribution where there is higher density or more area under the curve. This can be thought of as looking for hills on a flat landscape. Whether or not a proposed value is accepted depends on some hill-climbing rules.

Rule 1 – Moving UP a hill is always good. It means you are finding an area of higher density. For example, in Figure 11-9 moving from a value of about 0.5 to 1.5 is good. The ratio of the values from finish/start is 1.5/0.5=3=R. Therefore the min(1,R)=min(1,3)=1. Since u will always be 1 or less a value of R which causes uphill climbing will always be accepted.

Rule 2 – Going SLIGHTLY downhill is OK, as illustrated in Figure 11-10. The R value here is roughly 1.4/1.7=0.82. Therefore min(1,R)=0.82. Compared to a uniform draw there is a good chance that a value between 0 and 1 will be less than 0.82, so usually a slight downhill move will be accepted although not always.

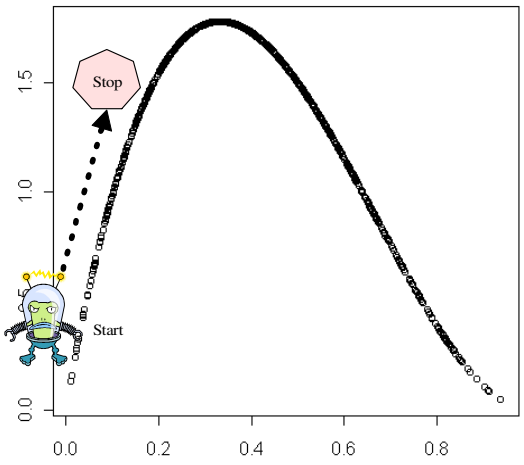


Figure 11-9

Permission granted to reproduce for nonprofit, educational use.

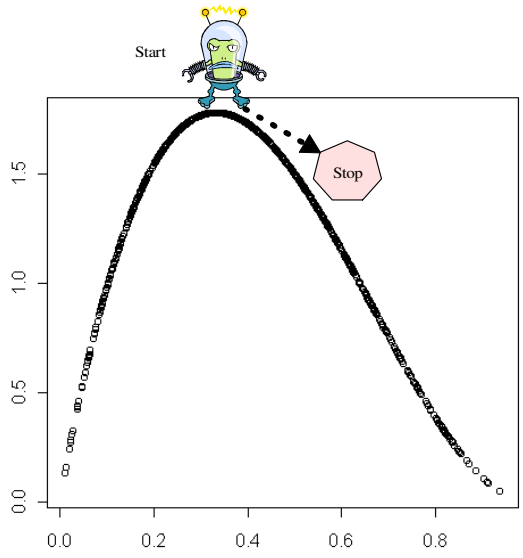


Figure 11-10

Rule 3 – Dropping off a cliff is generally not good. In Figure 11-11 Here R would be roughly 0.3/1.7=0.176. A uniform value has only a 17.6% chance of being this small. Therefore in the majority of cases the proposed value will be rejected, but not always.

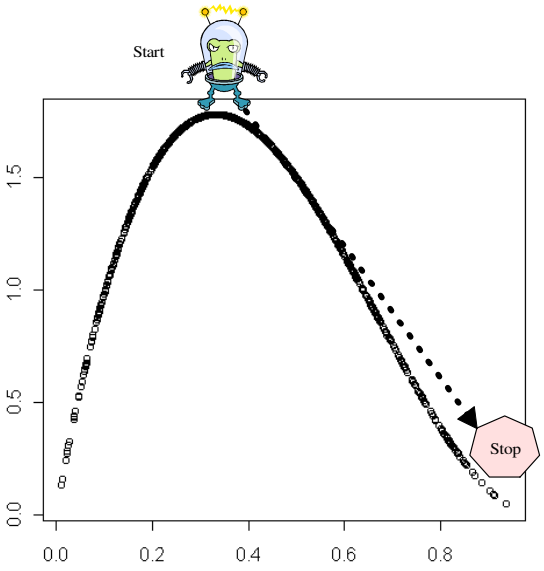


Figure 11-11

Permission granted to reproduce for nonprofit, educational use.

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).

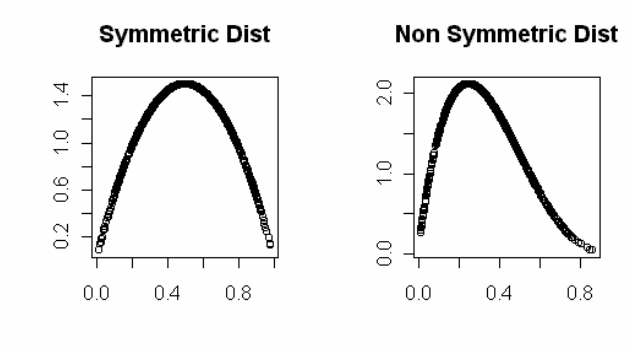


Figure 11-12

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*)

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

2. Calculate the ratio of the posterior density at thetaStar (new value) as compared to as theta (previous value). That is:

$$R=\frac{p(\theta^*)}{p(\theta)}$$

3. Draw a uniform [0,1] value, u

4. Accept the new value if u < min (1, R). That is, accept the proposed value if the uniform value is less than the minimum of 1 and R.

The constant updating of the values of the posterior distribution is a Markov chain that results in a stationary posterior distribution.

Let's run a short example in R using the Metropolis algorithm for which we know the result but pretend that we do not Let's simulate the posterior distribution of theta where theta is the parameter of the probability of success in a binomial distribution. We assume that we have one data outcome y=3 from a binomial with n=5 trials. We further assume that the prior distribution of theta is uniform(0,1). We have seen earlier that the posterior distribution in this case is a beta distribution with parameters alpha=1+y=4, and beta = 1 + n –y = 3, and a posterior mean of 4/7 =.5714. Without knowing this posterior distribution, for the Metropolis algorithm, we use the simple representation:

$$f(\theta \mid Y=3) \propto \Pr(Y=3 \mid \theta) f(\theta) = \theta^3 (1-\theta)^2 \cdot 1$$

First, initialize values. We will initialize theta to 0.04 for the first run of the Metropolis algorithm, an arbitrary starting value.

```
> nchain<-1000
> y<-3
> n<-5
> theta<-vector(length=nchain)
> theta[1]<-0.04
```

Next run a loop sampling thetaStar from the uniform distribution, a symmetric proposal distribution, and calculating the value of the ratio of p(θ*|Y=6)/p(θ|Y=6). Draw a uniform value for the acceptance criteria, and if u<r accept the new value of theta (thetaStar), otherwise reject thetaStar and use the same value of theta. Iterate through the loop the predetermined nchain length.

```
> for(i in 2:nchain){
+ thetastar<-runif(1)
+ r<-thetastar^y*(1-thetastar)^(n-y)/(theta[i-1]^y*(1-theta[i-1])^(n-y))
+ u<-runif(1)
+ if(u<r){
+ theta[i]<-thetastar}
+ else theta[i]<-theta[i-1]
+ }
```

Next, do some plots of the output, as depicted in Figure 11-9.

Permission granted to reproduce for nonprofit, educational use.

```
> par(mfrow=c(3,1))
> plot(1:100,theta[1:100],main="First 100 Runs")
> lines(1:100,theta[1:100])
> plot(1:1000,theta[1:1000],main="All Runs")
> lines(1:1000,theta[1:1000])
> plot(901:1000,theta[901:1000],main="Last 100 Runs")
> lines(901:1000,theta[901:1000])
```

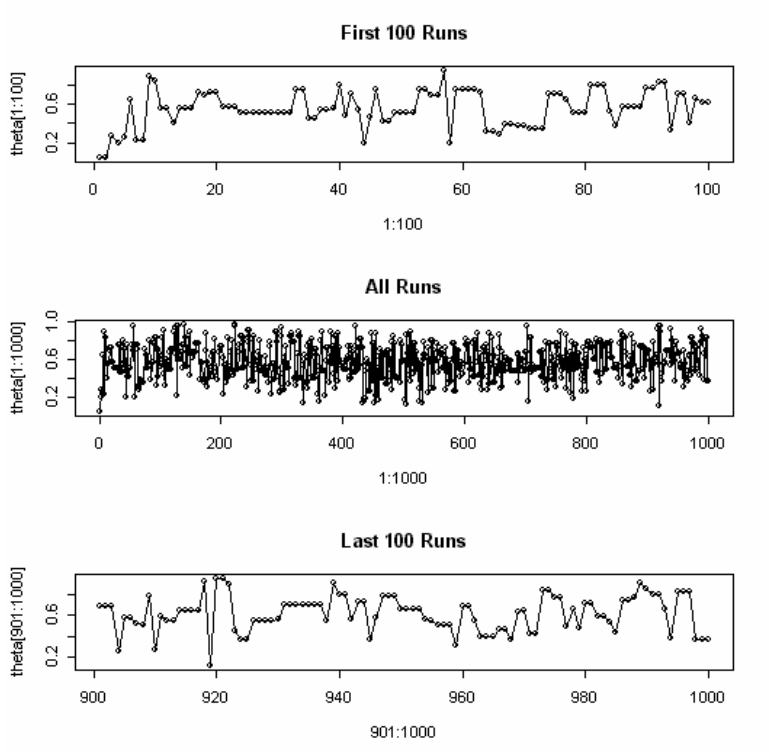


Figure 11-13

These plots are a time series of the 1000 runs. The first 100 runs are depicted separately and you can see that from runs 0 to 20 the chain is more erratic in behavior and takes about 20 cycles reach a more stable state. The initial period is called the burn-in period. Typically we discard data from the burn in period – often it is the first few hundred runs. Because this is a simple univariate example, the number of runs before the chain reaches a stable stationary state is very few. In more complex scenarios it is hundreds of runs that compromise the burn-in period. It is important to do time series plots of all runs and look at them.

Time series plots also are good visual diagnostics of how well a chain mixes. The chain in this example is well mixing. There are no hang-ups where the chain spends unusually long periods of time in areas that are substantially above or below the mean value of the posterior distribution As a check we can plot a

Permission granted to reproduce for nonprofit, educational use.

histogram of the generated values and overlay the theoretical beta(4,3) density function:

```
> hist(theta[101:1000],nclass=25,prob=T)
> xx <- (1:100)/100
> lines(xx,dbeta(xx,4,3))
```

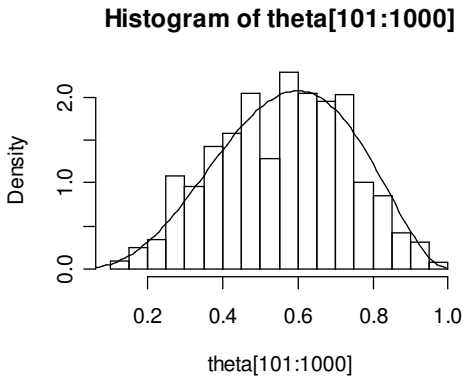


Figure 11-14

However be aware that the theta values that were generated by the Metropolis algorithm do not represent a random sample from the posterior distribution, rather they represent a Markov chain whose subsequent values are not statistically independent.

Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm extends the use of the Metropolis algorithm to situations where the proposal distribution is not symmetric. The algorithm is the same as with the Metropolis algorithm except that the proposal distribution results are added to the acceptance ratio term. Using the Metropolis Hastings algorithm the acceptance ratio is:

$$R = \frac{p(\theta^*)}{p(\theta)} \frac{q(\theta \mid \theta^*)}{q(\theta^* \mid \theta)}$$

The added term is the ratio of the proposal distribution q of theta (old value) conditioned on the new value thetaStar over the value of q of thetaStar conditioned on theta. This term accounts for the asymmetry of the distribution so that the proper acceptance ratio is used.

We will not do a full example using the Metropolis Hastings algorithm here, but examples are provided in many of the references listed in the appendix and other examples abound in the research literature.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Putting it All Together: Markov Chain Monte Carlo

In Chapter 10, we learned that Markov chains which are ergodic (aperiodic and irreducible) converge to a stationary equilibrium distribution. In this chapter we learned some algorithms for simulating posterior distributions of interest both without using Markov chains (inverse CDF, and rejection sampler) and using Markov chains (Gibbs, Metropolis-Hastings).

The two Markov chain methods discussed in this chapter are often the only way we have of implementing many Bayesian methods to determine a posterior distribution of interest. Although only simple examples were presented in this chapter, the concepts and algorithms discussed here apply to more complex high-dimensional distributions.

Gibbs versus Metropolis-Hastings

As mentioned previously, the Gibbs sampler is actually a special case of the Metropolis-Hastings algorithm, a statement that may make no sense if no one tells you that with the Gibbs sampler the proposal distribution is the target distribution. Therefore the acceptance ratio with the Gibb's sampler is:

$$R = \frac{p(\theta^*)}{p(\theta)} \frac{q(\theta \mid \theta^*)}{q(\theta^* \mid \theta)} = \frac{p(\theta^*)}{p(\theta)} \frac{p(\theta \mid \theta^*)}{p(\theta^* \mid \theta)} = 1$$

The acceptance ratio is always 1, therefore the value is always accepted.

The Gibbs sampler has many advantages over the Metropolis-Hastings sampler in that it is computationally much simpler. The Metropolis-Hastings performs evaluations on full distributions that can become computationally intense, and even computationally impossible (even on the most state of the art machines). This is because the Gibb's only computes from the conditional distributions, which is less computationally intense.

Of course, there are times when sampling from full conditionals is impossible (for various mathematical reasons) and the Metropolis-Hastings algorithms must be used. The next chapter introduces WinBugs, which is a program you can use in conjunction with R that utilizes the Gibbs sampler. There is no comparable easy to use program that utilizes the Metropolis-Hastings algorithm and implementing this algorithm probably requires original statistical programming.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Issues in Chain Efficacy

In conclusion to this chapter, it is fitting to briefly highly some of the areas of concern when using Markov chain algorithms to simulate a posterior distribution with regard to how well a chain runs. These issues apply to both Gibbs and Metropolis-Hastings algorithm. In the next chapter we will look at some specific examples and how these issues are addressed. All of these issues are areas of active statistical research.

Mixing

We have already brought up the issue of mixing on a few occasions. Mixing refers to how well the chain moves, or speed at which chain forgets its past. Mixing can be analyzed by looking at a time series plot (such as those in Figure 11-9) of the values simulated for each parameter of interest. It is generally impossible to tell in advance how well a chain will mix. However if you are comparing more than proposal distribution you can determine which is the better at mixing and use that one. Rapid mixing can indicate that the chain requires a shorter number of runs to reach a posterior distribution.

How Many Chains to Run?

The issue of whether to run one chain or multiple chains is of debate. Some suggest many short runs (Gelfand and Smith, 1990), whereas others suggest several long runs (Gelman and Rubin 1992), and others suggest one very long run (Geyer 1992). The idea behind running many runs is that it is possible that the chain reaches an incomplete posterior distribution because the chain gets hung up on one area of high density of a posterior density that has multiple distinct areas of high density where there is little distribution of density elsewhere. No official comment is made here about which method is correct, but running more than one chain is generally a good idea.

Burn in and run length

As discussed earlier the burn in consists of the initial iterations before the chain reaches a stationary state. How long the burn in is varies tremendously. It can be very short as in our example in Figure 11-13, or very long. Therefore the run length (number of times you run the chain) should be long enough to ensure than chain has indeed reached a stationary state. How long this is depends on the particular situation. If in doubt, do more runs.

Convergence

After burn-in we say chain has converged to a stationary state (posterior distribution). Many models will work, but some may be slow to converge. Sometimes a slow converging model may be improved (using a different proposal distribution and other ways), yet sometimes it is easier to do longer

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

runs. It is important to check convergence with convergent diagnostics, however there is no universal convergence diagnostic. This can be done using the package CODA.

Permission granted to reproduce for nonprofit, educational use.

12

MCMC using BRugs

This chapter applies the theories of MCMC and Gibbs sampling by introducing the use of the package BRugs and associated package CODA. These software tools make running MCMC analysis simple and accessible for those interested in simple applied models without having to do any complicated programming tasks.

About BRugs

Package BRugs incoroporates what used to be a separate program (WinBUGS, still available from the website http://www.mrc-bsu.cam.ac.uk/bugs/.). WinBUGS is a small, freeware program designed specifically to run Bayesian analysis for MCMC applications. BUGS is an acronym for Bayesian analysis Utilizing Gibbs Sampling. The original version is called classic BUGS and runs in a DOS-based environment. This version is also still available to interested users.

In biological science, BUGS is used widely in public health and genetics applications. Genetics aplications include including linkage analysis, recreating evolutionary trees, pedigree analysis, haplotype analysis, and other applications. This chapter will present a simple genetic example looking at the distribution of ABO blood types. Other applications of WinBUGS in bioinformatics are fathomlessly possible, and those skilled in Bayesian modeling can take advantage of WinBUGS to assist with MCMC analysis.

Package BRugs incoprorates the functionality of WinBUGS but does nto do much model diagnositics. Instead, convergence diagnostics are performed using the CODA package, a package designed for use in S-Plus or R.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

ABO Blood Types Example

The Model

Different markers determine blood types in humans. One of the marker systems often used is the ABO system. Humans are typed according to one of the four possible types: A, B, AB, and O. Different genes are encoded by variations called alleles. For the ABO blood type, there are three possible alleles, which we will refer to as A, B, and O. There are six combinations of alleles a human can have on their two chromosomes: AA, AB, BB, AO, BO, and OO.

In this system there is a pattern of co-dominance, where A and B are co-dominant. Allele O is always recessive. This means that if a person has the AB allele combination, he/she will be AB blood type since both markers show up (co-dominance). However if a person is type AO only marker A shows up. Likewise if a person is type BO only marker B shows up. The only case where a person can be type O is if he/she has both alleles of type O.

We call the allele combination a person has a genotype, with possible genotypes for blood types being AA, AB, BB, AO, BO, and OO. We call the way the genotype displays itself a phenotype and the possible phenotypes here are AB, A, B, and O.

If we consider the three possible alleles in the gene pool as proportions and assume these are the only alleles in the gene pool, the sum of proportions is 1. We can write the equation below using p for the proportion of allele A, q for the proportion of allele b, and r for the proportion of allele 0.

$$p+ q + r =1$$

Mathematically we can relate the allele frequency to phenotype frequency using the Hardy-Weinberg equilibrium for a three-allele locus:

$$(p+ q + r)^2=1$$

Doing the algebra:

$$p^2 + 2pr + 2pq + q^2 + 2qr + r^2 = 1$$

Table 12-1 summarizes the relationship between alleles (p, q, r), genotype frequencies, genotypes and phenotypes for the blood types.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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:

Permission granted to reproduce for nonprofit, educational use.

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

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.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Running the Model in BRugs

Create the Model

Open a new notepad document and enter the following model:

```
model BloodTypes {

##Prior for allele frequencies

#Gamma trick to generate uniform Dirichlet dist
a~dgamma(1,1)
b~dgamma(1,1)
o~dgamma(1,1)

#Scaled frequency of alleles  prior  for each allele
#frequence allele A
p<-a/(a+b+o)
#frequency allele B
q<-b/(a+b+o)
#frequency allele O
r<-o/(a+b+o)

##Multinomial likelihood models data

#A phenotype frequency
x[1]<-p*p+2*p*r
#B phenotype frequency
x[2]<-q*q+2*q*r
#AB phenotype frequency
x[3]<-2*p*q
#O phenotype frequency
x[4]<-r*r

n[1:4]~dmulti(x[ ],total)
```

Save this document as “bt.txt” in the R working directory.

Create the Data Set

Likewise, create a small file “btData.txt” in the R working directory.

```
#data
list(n=c(750,250,75,925),total=2000)

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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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 small 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'
>
>
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> 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.

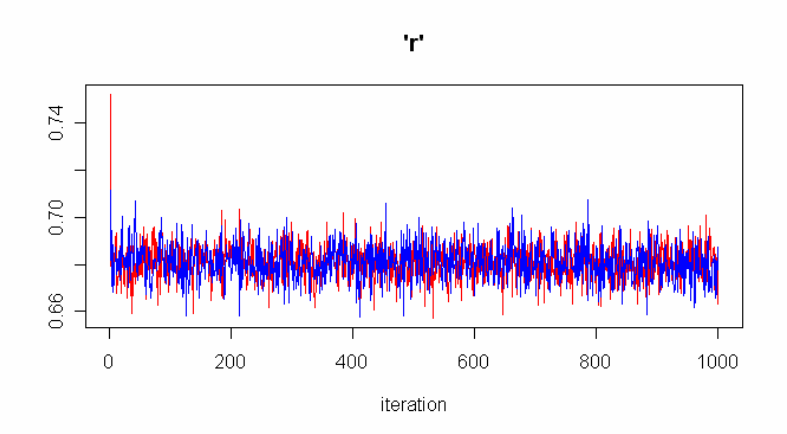


Figure 12-1: Time Series of Parameter r

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))
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

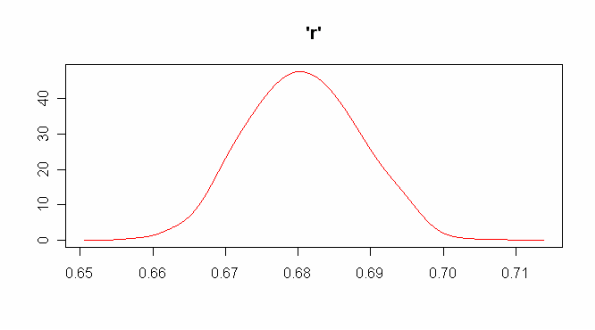


Figure 12-2: Marginal Density of Parameter r

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  0.07661 0.08516   0.09471   502    998
r 0.68070 0.007752 0.0002584  0.66640 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.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

13

Foundations of Statistical Inference

Statistical inference is about analyzing data and drawing conclusions, or inferences, about the data at hand. Often the reason for doing this is to fit a mathematical model to the data, which can be either a probability model or some type of predictive model, like a regression model, as discussed in Chapter 15. Obtaining such a model can be very useful for drawing further conclusions and testing (the subject of Chapter 14). The value derived from fitting a useful model is often the payoff of laborious experimentation and data collection.

This chapter discusses the process of analyzing sample data and the techniques to discover what we want to learn about a dataset in order to fit parameters for models and how R can help with these tasks. It should be noted here that classical and not Bayesian techniques are presented in this and subsequent chapters, but that there are parallel Bayesian methods. It is also noted that much of the material in this chapter is covered extensively in any introductory level statistics textbook. Therefore the emphasis here is on a review of topics and how they work in R. The interested reader should consult one of the books in the appendix for details of the concepts presented.

Sampling Theory

Usually when we collect data we are "sampling from a population". In the idealistic world we could collect every possible data point of interest from the entire population under study. In practice, except in unusual circumstances, this is impossible. Therefore, taking a sample from the population is the best we can do.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

A sample is a subset of the underlying population. In statistics, a good sample is characterized by being random and therefore representative of the population at hand. Underlying the principle of randomness is the property of independence. This means that whatever the i^{th} outcome of the sampling process is, the next (i^{th} + 1) outcome will have no dependence on the ith outcome. In other words all outcomes from a random sampling process are mutually independent. Outcomes are viewed as realizations of a random variable and therefore the outcomes for any experiment share the same underlying probability distribution. These two principles of a random sample are sometimes referred to as "i.i.d." which stands for independent and identically distributed.

A key issue with sampling is the size of the sample (which is almost always designated by the letter n). It should be intuitive that the larger the sample, the better the sample in terms of using the data to characterize the underlying population. Figure 13-1 illustrates the effect of sample size when the underling population is a standard normal.

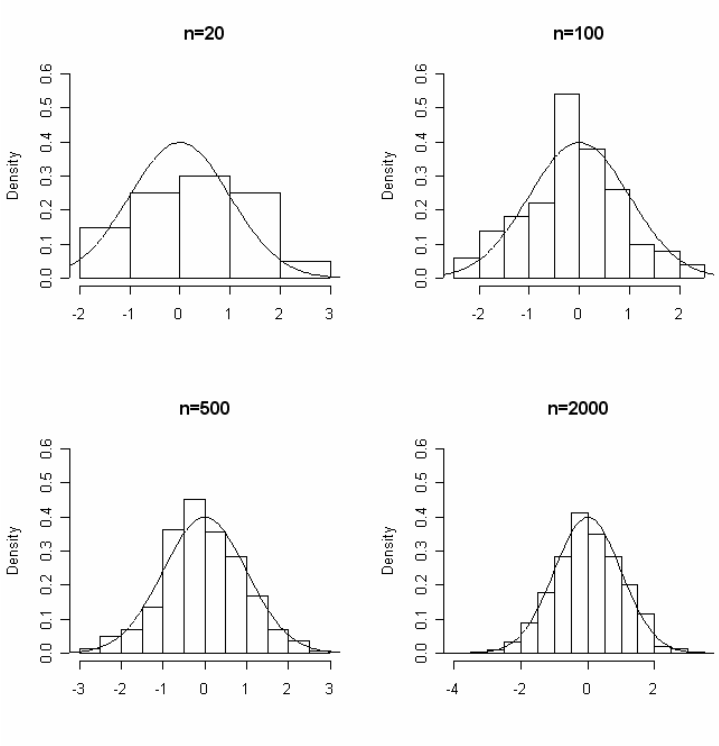


Figure 13-1: Effect of Sample Size on Approximating the Underlying Distribution

After a sample is collected, it is mathematically characterized by certain functions of the sample data. A statistic, by definition, is a function of the data. Some statistics you are likely already familiar with and that were discussed in Chapter 5 include mean, median, variance, etc. These concepts are introduced

Permission granted to reproduce for nonprofit, educational use.

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))
```

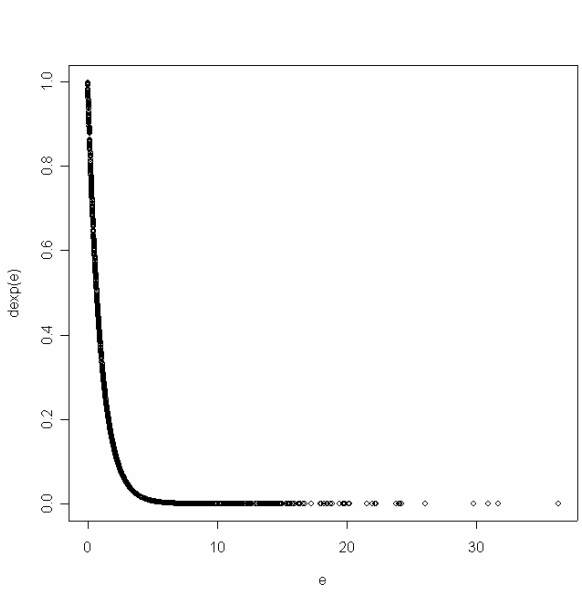


Figure 13-2: Exponential with lambda=4.

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.

Permission granted to reproduce for nonprofit, educational use.

```
>#Create matrices with 50 rows and n=samples size columns
> n5<-matrix(rexp(50*5,rate=0.25),nrow=50)
> n20<-matrix(rexp(50*20,rate=0.25),nrow=50)
> n50<-matrix(rexp(50*50,rate=0.25),nrow=50)
> n200<-matrix(rexp(50*200,rate=0.25),nrow=50)


> #Compute row means
> n5means<-rowMeans(n5)
> n20means<-rowMeans(n20)
> n50means<-rowMeans(n50)
> n200means<-rowMeans(n200)

>#Plot results
> par(mfrow=c(2,2))
> hist(n5means,xlab="",prob=T,xlim=range(0:12),ylim=
+ range(0,0.2,0.4,0.6,0.8),main="n=5",nclass=10)
> hist(n20means,xlab="",prob=T,xlim=range(0:12),ylim=
+ range(0,0.2,0.4,0.6,0.8),main="n=20",nclass=10)
> hist(n50means,xlab="",prob=T,xlim=range(0:12),ylim=
+ range(0,0.2,0.4,0.6,0.8),main="n=50",nclass=10)
> hist(n200means,xlab="",prob=T,xlim=range(0:12),ylim=
+ range(0,0.2,0.4,0.6,0.8),main="n=200",nclass=10)
```

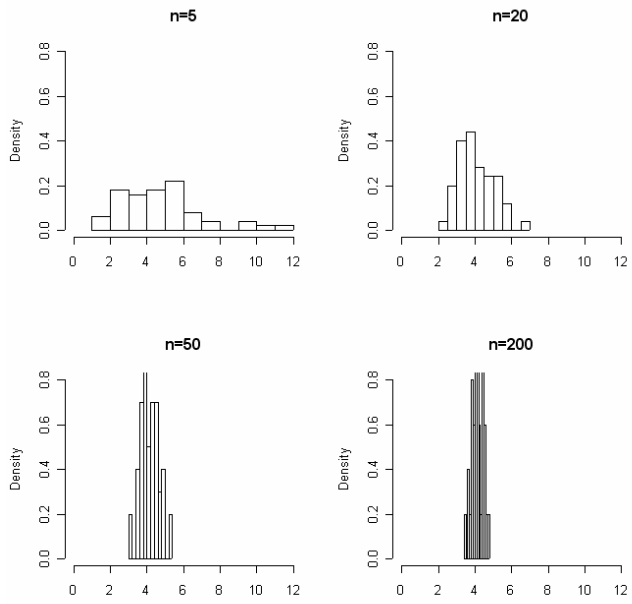


Figure 13-3 Illustrating the Law of Averages

Figure 13-3 plots the results of the distribution of the sample means based on sample size. . With this smallest sample size the calculated sample means do not show a normal distribution tendency. When the sample size is increased to 20 a normal pattern begins to appear. With sample size of n=200 a normal distribution pattern for the distribution of sample means is clear.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Figure 13-3 also illustrates another important point about the distribution of a sample mean. Notice that the spread of the data decreases as the sample size n increases. This is because the measure of variation, the standard deviation, which when applied to repeat samples of sample means is called "standard error of the mean", is inversely proportional to the square root of the sample size. Standard error can be written using the following equation, where sigma is the standard deviation of the data and n is the sample size:

$$\text{Stand. Error}\left(\bar{X}\right) = \frac{\sigma}{\sqrt{n}}$$

In other words if you want to increase precision of your sample mean, increasing the sample size always works. This actually has serious implication in statistical testing, where to meet a criterion to reject a hypothesis, sample size will undoubtedly play a role. We will discuss this issue again when we discuss hypothesis testing in the next chapter.

The variability, or spread of a sample is usually measured by the standard deviation (s). Mathematically (s) is the square root of the average of squared deviations from the mean. For technical reasons, to eliminate something called bias, dividing by n-1 instead of n usually performs the calculation.

$$s = \sqrt{\frac{\sum(X_i - \bar{X})^2}{n-1}}$$

In addition to measuring the mean and variability of the data, we often like to compute parameters of the underlying distribution, a process referred to as parameter estimation. This issue was touched upon in earlier chapters when parameters of standard probability distributions were discussed, but estimating and working with parameter estimates is a major focus of this chapter. Statistics include both parameters and other metrics of the data. We have already seen that for a normal distribution the major characteristics of the distribution, the mean and standard deviation are also the parameters of the distribution, but for other distributions such as gamma distributions the mean and standard distribution can be computed using the parameters of the distribution (alpha and beta for the gamma) but are not equal to the parameters.

In coming sections of this chapter we discuss two ways to estimate parameters – point estimates and interval estimates. Now, let's look at probability distributions that model sampled data.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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:

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

$$t = \frac{\bar{X} - \mu}{s\left(\bar{X}\right)}$$

where $\bar{X}$ is the sample mean based on a sample of size n, and $s\left(\bar{X}\right) = s/\sqrt{n}$ is the estimated standard error of the sample mean. Note that μ is both the mean of the data as well as the mean of the sampling distribution of the sample mean. If the data of the sample are more or less normally distributed, then the sampling distribution of the t-score is again a t distribution with n-1 degrees of freedom, where n is the size of the sample from which $\bar{X}$ is calculated.

The t-distribution is symmetric like the normal and has mean 0 but its standard deviation is > 1. Let's take the 50 mean values obtained earlier for the n=200 sample size of means from an exponential distribution with scale (= mean) = 4. Remember, that by the law of averages, these values approximately follow a normal distribution. We obtain the t scores of these 50 values.

```
> t<-(n200means-  4)/sd(n200means)
```

Next, let's plot these t-scores and overlay a curve for a standard t-distribution with n-1 degrees of freedom (where n=sample size). In R the density curve of the t distribution is called with the function

```
dt(x, df)
```

Where x is the data vector and df is the degrees of freedom.

```
> hist(t,prob=T,main="Standardized data wtih overlay of t-distribution")
> curve(dt(x,49),add=T)
```

Figure 13-4 shows the resulting plot.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

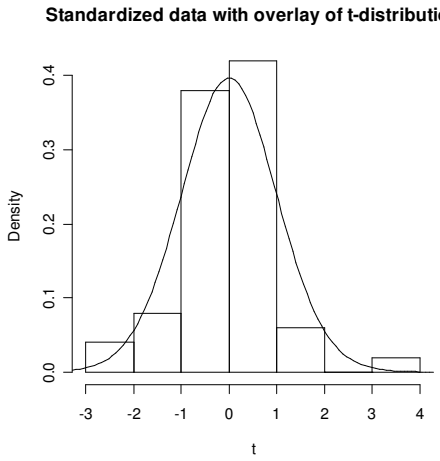


Figure 13-4

Let's look at the relationship between degrees of freedom and the shape of the curve. As an aside, the term "degrees of freedom" is a description of the number of observations that are free to vary after the sample statistics have been calculated. When we compute the sample standard deviation, the mean (a statistic) is used in the formula to compute the standard deviation. So the degrees of freedom left over is n-1, used in the parameter for the t-distribution. You will see degrees of freedom again with the Chi-Square and F-distribution. It is a confusing concept, and pay attention to how it is calculated for a particular distribution.

For a t-distribution as the degrees of freedom (sample size) increases, the distribution in the limiting case (n approaching infinity) becomes normal. In statistics this is referred to as an asymptotic approximation. When the t-distribution has a smaller number of degrees of freedom (smaller sample size) the distribution is more variable and less centered. This is correlated with the earlier discussions of when the sample size is smaller it is more variable and the mean is less precise (review Figure 13-3).

Let's plot some t-distributions with different degrees of freedom:

```
> x <- seq(-8,8,by=.1)
> par(mfrow=c(2,2))
> plot(x,dnorm(x),type='l',ylab="",main="df=2")
> lines(x,dt(x,df=2),lty=2)
> plot(x,dnorm(x),type='l',ylab="",main="df=5")
> lines(x,dt(x,df=5),lty=2)
> plot(x,dnorm(x),type='l',ylab="",main="df=10")
> lines(x,dt(x,df=10),lty=2)
> plot(x,dnorm(x),type='l',ylab="",main="df=20")
> lines(x,dt(x,df=20),lty=2)
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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.

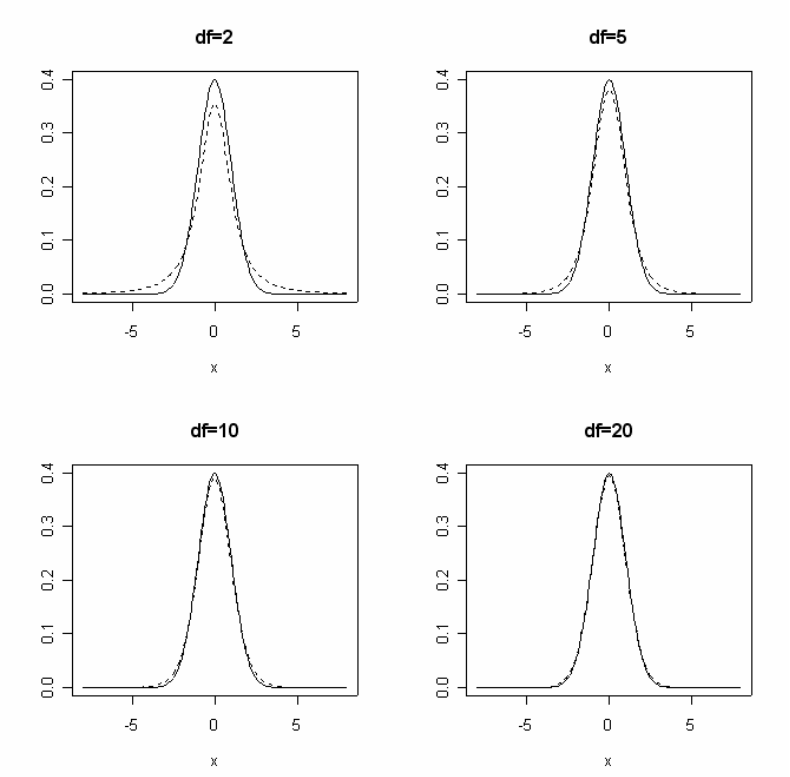


Figure 13-5

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:

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

Permission granted to reproduce for nonprofit, educational use.

The Chi-Square is a sampling distribution because it depends on the sample size through n-1 degrees of freedom. Varying the degrees of freedom changes the shape of the distribution. In R the density curve of the Chi-Square distribution is called by the following function:

```
dchisq(x, df)
```

Figure 13-6 plots the Chi Square distribution with 2, 4 and 9 degrees of freedom. As the degrees of freedom increase the shape of the Chi Square becomes more normally distributed and the distribution moves to the right.

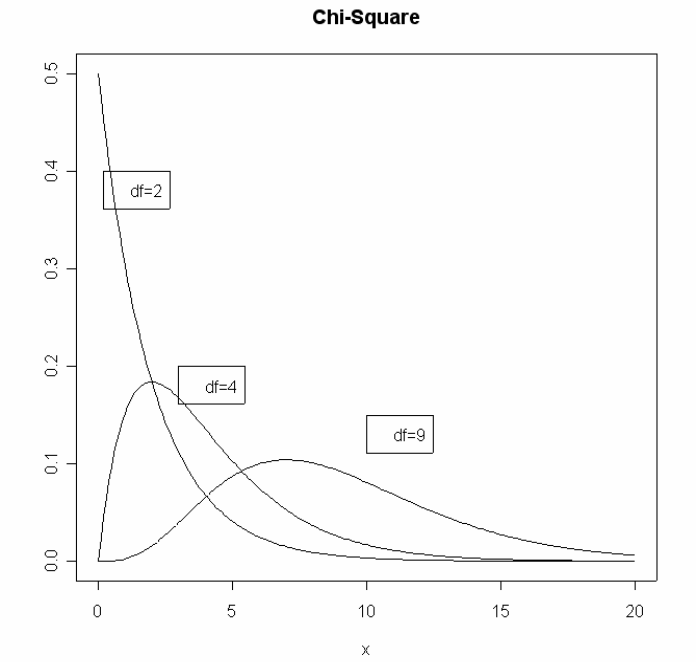


Figure 13-6: Chi-Square Distribution with 2, 4, and 9 Degrees of Freedom

The Chi-Square, like any other probability distribution, has an area under the curve of 1. Because it is a member of the gamma family, it is somewhat mathematically complicated. Computing specific values of the Chi-Square is difficult to do directly, so to compute such values tables or computers are used. In R the `qchisq()` function produces Chi-Square test statistic values.

The F Distribution

The Chi-Square distribution is not so interesting in and of itself, although testing using the Chi-Square has wide applications. However, what is very interesting is the distribution of the ratio of two random variables, each having a Chi-

Permission granted to reproduce for nonprofit, educational use.

Square distribution. Mathematically such a ratio is described and modeled with the F distribution. Let U and V be independent random variables, where U has a Chi-Square distribution with m degrees of freedom and V has Chi-Square distribution with n degrees of freedom. Define the ratio W as follows:

$$W = \frac{U/m}{V/n} = \frac{n}{m}\frac{U}{V}$$

This ratio creates a random variable W whose distribution is an F distribution with numerator degrees of freedom m and denominator degrees of freedom n.

At first glance, the F distribution may make no sense or seem useless. Au contraire! The F distribution is an incredibly useful distribution and the basis for many statistical inference tests. What the F ratio creates is a testable "signal to noise" ratio. Suppose the variability of background noise is modeled with random variable U and variability due to the experimental effects is modeled with random variable V. Computing the ratio of variability and modeling it with an F distribution provides a criterion to determine if the experimental procedure achieves a significant effect over the background noise level. This type of testing has a wide range of applications. In bioinformatics, F ratios are used extensively in microarray data analysis. Indeed F ratios serve as the basis for ANOVA (analysis of variance), a procedure discussed in the next chapter, which in turn is the basis for the branch of statistics involving the design of experiments.

Meanwhile, let's work with the F distribution using R. The F distribution is a sampling distribution that depends on two sample sizes. The formula for calling the density curve of the F distribution in R is:

```
df(x, df1, df2)
```

Where `df1` is the numerator degrees of freedom and `df2` is the denominator degrees of freedom. We will work more with the specifics of the F distribution in the next chapters, but meanwhile let's just look at how changing the `df1` and df2 affects the shape of the F distribution curve.

The following code produces the F distribution plots in Figure 13-7:

```
> x <- seq(.1,5,by=.005)
> m <- c(1,5,10,30)
> n <- c(1,5,10,30)
> par(mfrow=c(4,4))
> for (i in 1:4){
+   for (j in 1:4) {
+           plot(x,df(x,m[i],n[j]),type='l',ylab="f(x)",cex=.6)
+           title(paste(paste("dof =",m[i]),n[j],sep=","))
>       }
>     }
```


Permission granted to reproduce for nonprofit, educational use.

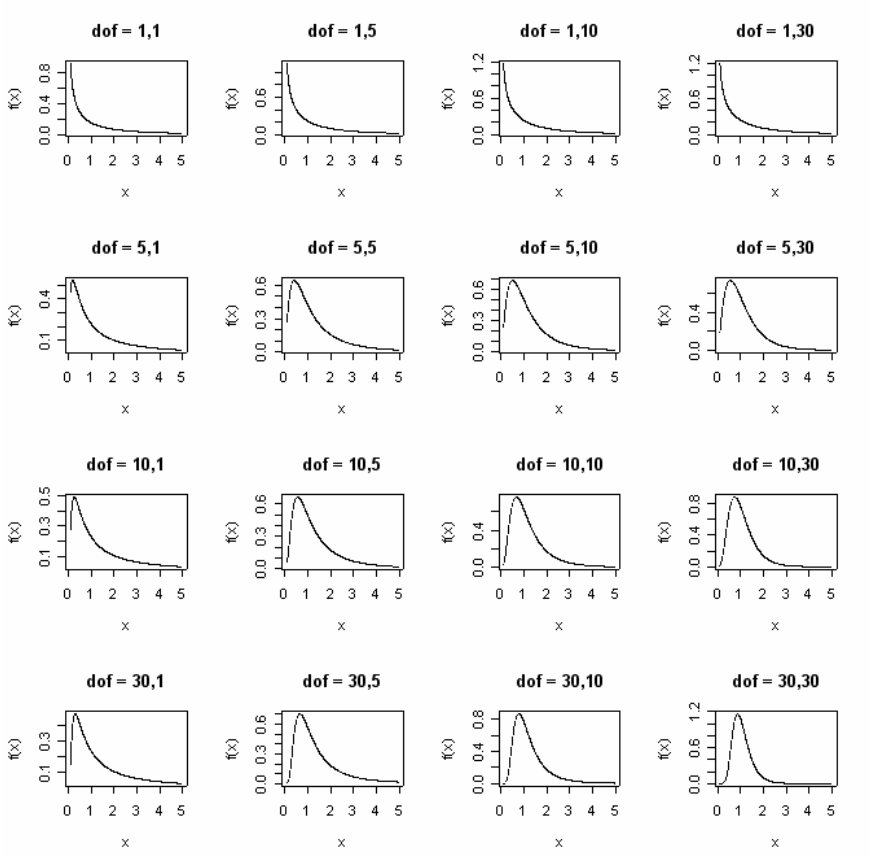


Figure 13-7: F distributions with varying df1, df2

Like for the Chi-Square, specific percentiles of the F distribution are difficult to determine analytically. However the `qf()` function in R produces the appropriate test statistics. We will use this function extensively when we discuss ANOVA. Let's now shift gears and investigate methods of estimating parameters.

Parameter Estimation

Given our sample data we want to fit a model to the data. Often we would like to fit a standard probability model (see Chapters 7 & 8). In order to do this, we need to determine the best fitting parameters for the particular model we have in mind. Therefore we need a method of estimating the parameters. Parameter estimates take two forms, point estimates and interval estimates. Point estimates have their merit in being very useful in defining the model. Interval estimates have merit in quantifying how precise a parameter estimate is. Often you will want both a point estimate and an interval estimate for a particular parameter.

Permission granted to reproduce for nonprofit, educational use.

This section reviews some basics of both types of estimates. Parameter estimation is a key topic in mathematical statistics and selected appendix references are listed which cover this topic in mathematical detail for the interested reader.

Point Estimates

A point estimate is a single value estimate of a population parameter calculated from sample data. Earlier in this chapter we calculated means of samples from the exponential. Each individual mean is a point estimate and as we resampled we had a distribution of point estimates (means), which by the law of averages follow a normal distribution pattern when n is large. We can also measure based on the samples of data the standard error of the mean and model its distribution as well (proportional to a Chi Square). In any given sample the point estimate differs from the true underlying population parameter. If in many repeated (conceptual) samples this difference averages out to zero, we say that the point estimate is unbiased.

There are different methods of calculating point estimates, including method of moments estimates, maximum likelihood methods, least squares estimation, Bayesian methods, and others. We will only work in detail with maximum likelihood estimates specifically here, but a good mathematical statistics book will cover the other methods in great depth.

For some cases, point estimates are quite simple to make. Table 13-1 lists some common point estimates of parameters that can be calculated with simple algebra or using basic functions in R.

Table 13-1: Point Estimates that are Simple to Calculate

Point Estimate	Underlying Parameter	Algebraic Calculation	R Function
$\overline{X}$	Mean (μ) of a normal population	$\frac{\sum X_i}{n}$	mean()
s^2	Variance (σ^2) of a normal population	$\frac{\sum (X_i - \overline{X})^2}{n-1}$	var()
s	Standard deviation (σ) of a normal population	$\sqrt{s^2}$	sd()
p	Parameter (p) for binomial model	X/n where X=number of successes	Calculate manually

Permission granted to reproduce for nonprofit, educational use.

However what if we want to estimate more complicated parameters? For example, calculate point estimates for the alpha and beta parameters of a gamma distribution? Reviewing the formula in Chapter 7 for the gamma distribution indicates that there exist no simple calculations of point estimates of these population parameters. Therefore in many cases we need a more sophisticated method to make point estimates. Often the maximum likelihood method (MLE) works best.

Maximum Likelihood Estimation (MLE)

Recall from Chapter 9 that a likelihood function is the function of the parameters given the data. The principle of maximum likelihood estimation is that somewhere on the range of parameter values, the likelihood function hits a maximum value. The parameter values for which the likelihood function obtains its maximum are called the maximum likelihood estimates for the parameters

Analytically, maximum likelihood estimates can be solved using basic calculus. Recall from calculus that a maximum value occurs when the first derivative of a function is set to zero. Often the analytical solution utilizes the practice of taking logarithms to make the analytical solution easier (the resulting estimate is equivalent). Figure 13-8 illustrates the analytical MLE solution for a binomial parameter. Although the binomial parameter is easily estimated algebraically it is illustrated here because is perhaps the easiest to calculate using the analytical MLE method.

$$
\begin{aligned}
lik(p) &= p^{x_1}(1-p)^{1-x_1}\, p^{x_2}(1-p)^{1-x_2} \cdot \ldots \cdot p^{x_n}(1-p)^{1-x_n} \\
&= p^{\sum x_i}(1-p)^{n-\sum x_i} \\
l(p) &= \log lik(p) = \sum x_i \log(p) + \left(n - \sum x_i\right)\log(1-p) \\
l'(p) &= \frac{dl(p)}{dp} = \frac{\sum x_i}{p} - \frac{n-\sum x_i}{1-p} = 0 \quad \text{for critical point} \\
(1-p)\sum x_i &= p\left(n - \sum x_i\right) \;\rightarrow\; \sum x_i = pn \\
\rightarrow\; \hat{p} &= \frac{\sum x_i}{n} = \frac{\#\text{ successes}}{\#\text{ trials}}
\end{aligned}
$$

Figure 13-8: Analytical MLE for Binomial Parameter

However, even the "simple" analytical solution in Figure 13-8 looks complicated, and instead of using analytical methods and a pencil, it is handy to use R to help calculate MLE estimates.

A common use of maximum likelihood estimation in genetics is to estimate parameters for the Hardy Weinberg equilibrium model for the distribution of

Permission granted to reproduce for nonprofit, educational use.

genotypes in a population at equilibrium. In a given population gene pool, for a two allele gene locus, alleles A and a are at frequencies p and q, respectively. According to Hardy Weinberg equilibrium the genetic frequencies of genotypes are modeled by the simple equation: $p^2+2pq+q^2=1$. The likelihood for this follows a multinomial probability distribution that we can model, where parameter theta,θ=p (and therefore 1-θ=q, theta between 0 and 1) as:

$$L(\theta)\alpha(\theta)^{2nAA}(2\theta(1-\theta))^{nAa}(1-\theta)^{2naa}$$

Letting n1=nAA and n2=nAa and n3=naa and logarithms (does not matter which base):

$$\text{Log } L(\theta)=2n_1\log(\theta) + n_2\log(2\theta(1-\theta)) + 2n_3\log(1-\theta)$$

Eliminating the multiplicative term:

$$\text{Log } L(\theta)= 2n_1\log(\theta) + n_2\log(2) + n_2\log\theta + n_2\log(1-\theta) + 2n_3\log(1-\theta)$$

To analytically solve for the maximum likelihood take the first derivative of this (which we designate with a lowercase letter "l" prime):

$$l'(\theta)=2n_1/\theta + n_2/\theta - n_2/(1-\theta) - 2n_3/(1-\theta)$$

To find the value of theta that maximizes the above we could solve it analytically (which is not hard in this case, but with other models is often analytically impossible) by setting $l'(\theta)=0$ and solving for theta, or we can solve it using R and finding the maximum value of the likelihood function over a grid of theta values, which is very simple and illustrated below.

Suppose we collect data from a population sample of 500 (assume the population obeys Hardy Weinberg equilibrium) and obtain the data in Table 13-2. Note that simply calculating theta from the proportion of AA does not work, since this does not account for the information about theta contained in the rest of the data.

Table 13-2

Genotype	**Data**	**Variable**
AA	134	n_1
Aa	266	n_2
aa	100	n_3

Let's program R to solve for the maximum likelihood estimate of theta given this data.

Permission granted to reproduce for nonprofit, educational use.

```
> #Create a grid of theta values
> theta<-1:1000/1000

> #Create a data vector to store (log)likelihood values
> lik<-vector(length=1000)

> #Enter data
> n1<-134
> n2<-266
> n3<-100

> #Given data, evaluate log likelihood
> for(i in 1:1000){
+ lik[i]<-2*n1*log(theta[i])+n2*log(2)+n2*log(theta[i])+
+ n2*log(1-theta[i])+2*n3*log(1-theta[i])}

>#  Use which function to determine max value of lik
> which(lik==max(lik))
[1] 534
> lik[534]
[1] -506.4562
> #MLE value for theta (corresponding vector index to lik[534])
> theta[534]
[1] 0.534

>#Plot of results
> plot(theta,lik,xlab="theta",ylab="log likelihood",
+ main="MLE estimation for theta")
> abline(v=theta[534],lty=2)
> legend(x=0.54,y=-2000,legend="MLE theta=0.534")
```

Figure 13-9 illustrates the resulting plot.

Permission granted to reproduce for nonprofit, educational use.

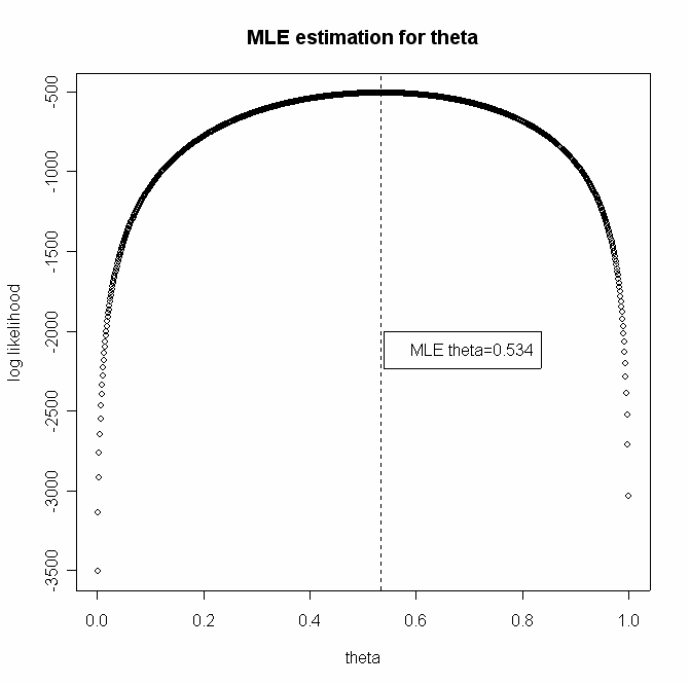


Figure 13-9: MLE for theta

To solve other MLE parameter estimates, apply the same principle illustrated above: find the likelihood function, take the first derivative, and find the maximum value for the parameter under study. There are a lot of R packages with specific applications that utilize maximum likelihood estimation. All build on the principle described above. Maximum likelihood methods are covered in most mathematical statistics texts.

Interval Estimates

When we obtain a point estimate for a parameter we only have a single value. That value in and of itself says nothing about how accurate or precise the estimate is. Interval estimates provide an alternative method for estimating a parameter by providing a probable range of values the parameter is likely to take. Often it is good to use both a point estimate and an interval estimate for a parameter under study. The most common type of interval estimate is called a confidence interval, which is what we will discuss here. Other types of interval estimates exist, such as predictive intervals and tolerance intervals. These have a similar theme but different purposes from a confidence interval.

The classic and most common method of constructing a confidence interval requires a point estimate of the parameter and follows the general formula:

point estimate +/- standard error of point estimate * test statistic

Permission granted to reproduce for nonprofit, educational use.

This yields an interval that is symmetric with midpoint equals the point estimate and with a lower bound of "point estimate – se * test statistic" and an upper bound of "point estimate + se *test statistic". The width of the interval depends on the standard error and the confidence level of the test statistic chosen. Smaller standard errors result in narrower intervals, indicating more precision in the point estimate. This is because the standard error is a measure of variation in the date on which the point estimate is made.

The test statistic comes from the appropriate sampling distribution of the parameter estimate. We define the level of a confidence interval to be 1-alpha, which produces a $(1-a)\cdot 100\%$ confidence interval, where alpha is the error level. Therefore we compute the test statistic for the lower end to be that of alpha/2 and for the upper end to be that of 1-alpha/2, distributing the error equally between the upper and lower bounds of the confidence interval. An example below will demonstrate this. Using alpha=0.05 is the norm for most confidence intervals. Using more extreme values of the test statistic (say a 95% confidence level) produces wider confidence intervals than less extreme values of the test statistic (say a 80% confidence interval), which would be narrower.

Although R doesn't have a generic function to compute confidence intervals, knowing the basic formula above and the specifics of what you're making a confidence interval of, it's vary easy to construct confidence intervals in R.

For example, let's create a confidence interval for a sample mean derived from a random sample of size 20 from a standard normal density. To do this we can use R to first generate a random sample:

```
> #Random sample of 20 from standard normal
> x<-rnorm(20,0,1)
> #Computer mean and standard error
> xBar<-mean(x)
> s <- sd(x)
> se<- s/(19^.5)
> xBar
[1] -0.1041
> se
[1] 0.2821
```

Recall from the discussion of sampling distributions earlier in this chapter that the distribution of the mean from a small sample can be modeled using a student's t-distribution. To do a 95% confidence interval, the alpha level is 0.05 (or 5%). We distribute this evenly between the upper and lower confidence limits, so for a symmetric distribution like the t-distribution, we can use as a test statistic the 0.975 percentile of the t-distribution with n-1=19 degrees of freedom.

```
> test<-qt(0.975,19)
> test
[1] 2.093024
```


Permission granted to reproduce for nonprofit, educational use.

Now that we have all the necessary information – the point estimate, the standard error, and the test statistic values – we can easily make a 95% confidence interval using R:

```
> clLower<-xBar-se*test
> clLower
[1] -0.6947
> clUpper<-xBar+se*test
> clUpper
[1] 0.4864
```

So a 95% confidence interval for our parameter estimate for the mean of our random sample is (-0.695, 0.486). But what does this mean? The way a confidence interval is interpreted is as follows: if we repeated this experiment (drawing random samples of size 20 from a standard normal) and made confidence intervals for each sample mean point estimate, then 95% of the time the confidence interval derived would contain the true mean. Note the probability statement is made about the confidence interval itself, not about the probability of an interval containing the true parameter. Making confidence intervals for other point estimates follows the same algorithm discussed here. The specific values will differ, but the general formula will hold.

Bootstrapping

To motivate the technique of bootstrapping, consider the following scenario. You collect gene expression data for a particular gene from 200 samples of cells (note this data is available in the rbioinfo package as bootstrapGE). Because you are wise and look at the distribution of data before performing further analysis, you do a histogram and get the distribution depicted in Figure 13-10.

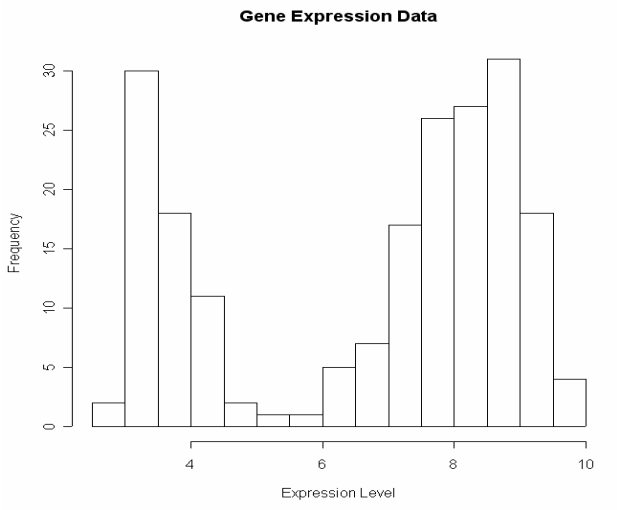


Figure 13-10

Permission granted to reproduce for nonprofit, educational use.

Clearly this is not a nice distribution pattern in terms of being able to fit a standard probability model. Computing a mean expression level alone given this distribution may not have meaning and does not describe the situation well. Hence, a more important objective of study is primarily to understand the variability of the expression level for this particular gene. Also the law of averages does not work well for this kind of distribution, unless you have a very large data set, say n=1000 or more. This is a case where the bootstrap can be a useful technique.

The bootstrap is a technique developed by Brad Efron (a statistics professor at Stanford) to find standard errors (measures of variability in the data) or confidence intervals in complicated situations where analytical computation is impossible. It is a simple, yet powerful, computational technique that in recent years has shown explosive growth in applications. In bioinformatics literature bootstrapping is used extensively in phylogeny analysis and in microarray data analysis. More applications of the bootstrap are sure to come, but here we focus on the basics of the bootstrap and how to do basic bootstrapping in R.

Bootstrap techniques are generally categorized as either nonparametric or parametric. Parametric bootstrap techniques assume that the data are generated from a standard parametric probability model (such as the normal, Poisson, and other models discussed in earlier chapters). We will not discuss parametric bootstrap techniques here although the principles and techniques are very similar to nonparametric techniques. Nonparametric bootstrap techniques are more versatile and suit our example situation better. Because of their versatility, nonparametric bootstrap techniques are the more popular type of bootstrap applications.

Nonparametric Bootstrapping

The beauty of the nonparametric bootstrap is that, since there are no assumptions of the underlying model, you can apply it to any dataset. Let's return to the bootstrapGE dataset discussion. Our goal is to determine how variable the expression of the gene is given the data. However, we have no formulas for the standard error formulation to use. So how should we go about determining the standard error?

The bootstrap estimates the standard error of a metric (such as a parameter estimate, mean, median, etc) by repeatedly drawing bootstrap samples from the original data. The samples are drawn with replacement and the sample size of each sample is the same as the original sample size. For each bootstrap sample, the metric is measured. Typically about 1000 bootstrap samples are taken. Then, the standard error of the metric under study is measured using the observed variation of the bootstrap samples. Essentially the process is a process of pseudo sampling from the original dataset to determine the variability of the dataset. Although it may sound too easy and simplistic, this is a very robust and

Permission granted to reproduce for nonprofit, educational use.

statistically sound technique for measuring standard errors. Figure 13-11 presents a schematic of the bootstrap process.

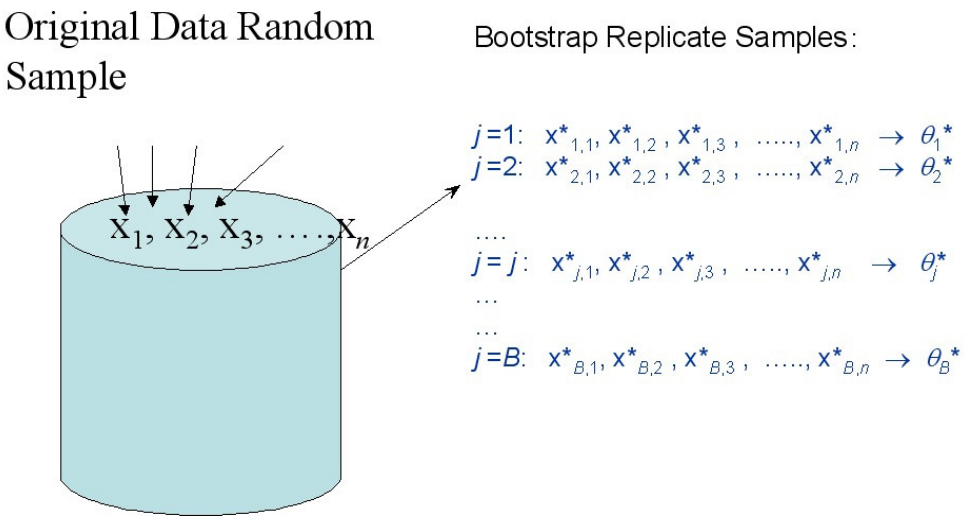


Figure 13-11

The essence of what the nonparametric bootstrap is doing is sampling from the empirical cumulative density function (cdf) of the data, which was introduced in Chapter 6. The empirical probability distribution assigns an equal probability to each of the data points, 1/n. Therefore when we resample every data point has an equal chance of being sampled. Using the cdf is what allows us to not rely on a particular probability model.

Recall that to plot the empirical cdf for any dataset in R, you can use package `stepfun` (which is usually included with the base installation, or can be obtained from the CRAN site). From this package you can simply plot the empirical cdf with the `plot.stepfun` functionality. Let's do this for our dataset, with the resulting plot in Figure 13-12. The line in the middle is the line of the median of the distribution, with 50% of the data above and below the median.

```
> #use package step fun to plot CDF
> plot.stepfun(bootstrapGE)

> #add a median line
> abline(h=0.5,lty=2)
```


Permission granted to reproduce for nonprofit, educational use.

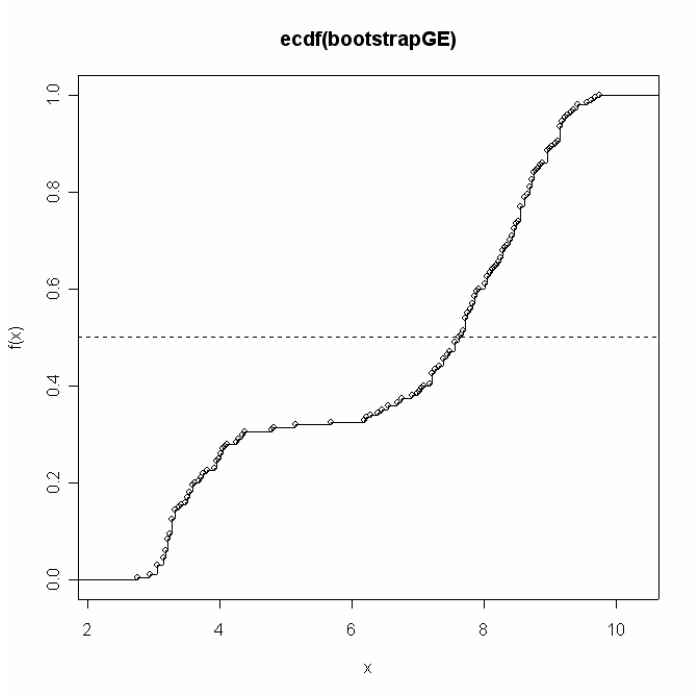


Figure 13-12

Let's go ahead and take 1000 bootstrap samples from our original dataset:

```
> #create a matrix to hold boostrap samples
> #note number of rows=number of bootstrap samples
> #note number of columns=sample size (equal to origninal n=200)
> bootsamples<-matrix(nrow=1000,ncol=200)

> #perform sampling using sample function, note replace=T
> for(i in 1:1000){
+ bootsamples[i,]<-sample(bootstrapGE,200,replace=T)}
```

Next, let's use the median as our metric to study the standard error of. The median provides a good central measure for a distribution of unknown form, which is the case here. We calculate medians for each of our 1000 bootstrap samples and store them in a new vector:

```
> #create vector to store medians
> bootmedstar<-vector(length=1000)

> #calculate and store median values
> for(i in 1:1000){bootmedstar[i]<-median(bootsamples[i,])}
```

For interest, let's look at the distribution of the bootstrap sample medians, depicted in Figure 13-13.

```
> hist(bootmedstar,nclass=40,xlab="median",main=
+ "Dist. of bootstrap sample medians")
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

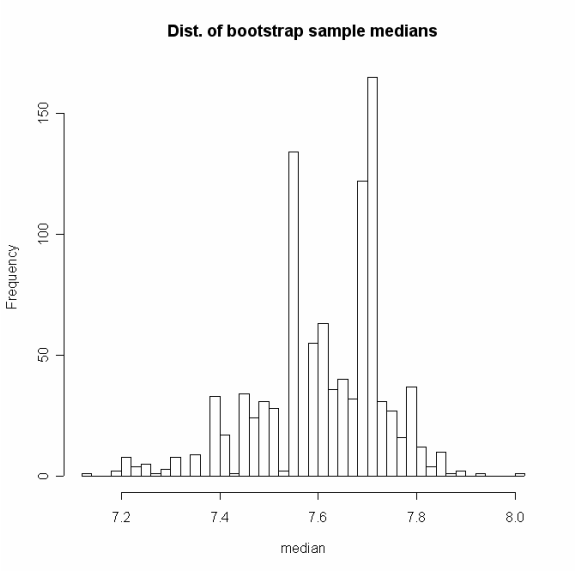


Figure 13-13

Next, the easiest thing to do to get a sense of variability of the data is to simply take the standard deviation of the bootstrap sample medians:

```
> sd(bootmedstar)
[1] 0.1298797
```

This is the bootstrap standard error of the sample median and can serve as a measure of variability of the central location of gene expression for the original data. Such a measure could serve as a test statistic to compare with variability of expression of other genes, or of the same gene under different conditions.

The above is just one example of bootstrapping presented to illustrate the basics of the process. There are many other uses of the bootstrap, but most involve the theme of hunting for a measure of standard error or variability for a dataset with an unknown distribution.

R Packages for Bootstrapping

R has two specific packages devoted to the bootstrap: `bootstrap` and `boot`. Both packages are worth looking at and include various applications of the bootstrap and further examples. In addition, many other packages, such as the `genetics` package, contain application specific bootstrap functionality.

Permission granted to reproduce for nonprofit, educational use.

14

Hypothesis Testing

The previous chapter reviewed the basics of statistical inference, including sampling theory and parameter estimation methods. This chapter continues on the topic of statistical inference, discussing hypothesis testing and how to perform hypothesis tests using R. Hypothesis testing is a very involved topic with broad applications, and we generally cover only the essentials here, focusing on tests likely to be of application in bioinformatics. Many introductory statistics texts cover hypothesis testing in great depth and almost any text should serve as a good reference to the interested reader.

The philosophical paradigm of hypothesis testing is that an experimental hypothesis is constructed, then the experiment is performed and data are collected, statistical tests based on the data are preformed and the results are compared with the initial hypothesis. Hypothesis testing is the foundation of the scientific method and therefore crucial to empirical data analysis. Hypothesis testing is a very broad topic and there are hundreds of possible tests. Luckily the different tests follow the same basic theory. Therefore we will cover the basics of the theory here, and then introduce some specific examples of commonly used tests.

Basic Theory

Hypothesis testing begins with the formulation of a null hypothesis (H_0) and an alternative hypothesis (H_A) testing the value of a statistic calculated from a sample of data collected in an experiment. Usually the alternative hypothesis is the logical negation of the null hypothesis. Often the null hypothesis is a clearly specified situation, which can easily be translated to a probability distribution that models a set of data. For example, a common null hypothesis is that the

Permission granted to reproduce for nonprofit, educational use.

mean of a population is 0. The testing goal often is to disprove the null hypothesis, if possible. . In this case, this would be to show the mean is not zero and differs significantly enough from zero based on a statistical test. The test uses information from the sample data. In this case we would base our evidence on the sample mean (remember this follows a normal approximation for large samples according to the central limit theorem discussed in the last chapter)

When we use an alternative hypothesis that "the mean is different from zero" we need to be a bit more specific. To be more specific, we can classify alternative hypothesis as being one-sided (also referred to as one-tailed or simple) or two-sided (also referred to as two-tailed or composite). A one-sided alternative hypothesis in this case may say the mean is greater than zero, or the mean is less than zero, but not either case. A two-sided alternative would state the mean is not equal to zero, and accounts for the mean being less than or greater than zero, either way. Two-sided tests are generally more common. When performing a hypothesis test it is important to be clear about what exactly you are testing and whether the alternative is a one or two-sided test.

After (1) clearly defining the null and alternative hypotheses, (2) you collect data. Using the data, you then (3) calculate the appropriate test statistic, (4) perform the appropriate test, (5) and then draw a conclusion based on the test result. Let's illustrate this with a simple example.

In-Depth Example to Illustrate Theory

One interesting application of statistics to bioinformatics concerns the ability to predict the secondary structure of protein molecules. Four proteins of known crystal structure (empirically verified secondary structures) were used to test various methods available on the Internet that allowed the user to enter primary structures of proteins and then used algorithms to predict secondary structure motifs, returning results to the user with a record of which amino acids corresponded to which predicted secondary structure. The proteins used are listed in Table 14-1 and the prediction algorithms and Internet sites used are listed in Table 14-2. The information in these tables is presented for informational purposes to illustrate where the data came from (and a way of doing an experiment using technology widely available). Crystal structure data was obtained using the PDB, or protein database, at *http://www.rcsb.org/pdb/*.

Permission granted to reproduce for nonprofit, educational use.

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 1BMZ:A)	GPTGTGESKCPLMVKVLDAVRGSPAINVAVHVFRKAADD TWEPFASGKTSESGELHGLTTEEEFVEGIYKVEIDTKSY WKALGISPFHEHAEVVFTANDSGPRRYTIAALLSPYSYS TTAVVTNPKE (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

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

	heidelberg.de/predictprotein/submit_def.html	local window comparison with sequence homology comparison

The experiment is to obtain results from the prediction sites and compare these results with empirical crystal structure data for each protein and score each amino acid residue as predicted correct or incorrect. Statistically we could come up with several tests using data collected from this experiment. To do a broad and simple test of the efficacy of the prediction methods, let's test that the average (mean) % Correct is 50% across the different proteins and different methods.

Step 1: Clearly define the null and alternative hypothesis.

The null hypothesis (H_o) is that the mean percentage correct is 50% (=0.5). The alternative hypothesis (H_a) is that the mean percentage is not equal to 50%.

Step 2: Collect the data

The total percentage of amino acids that followed correct secondary structural motif by method is recorded in Table 14-3. Note for the Cho Fassman test the average of beta and alpha structure prediction is used.

Table 14-3

Protein	Method	% Correct
Ubiquitin	CF AVG	0.467
Ubiquitin	GOR	0.645
Ubiquitin	PHD	0.868
DeoxyHb	CF AVG	0.472
DeoxyHb	GOR	0.844
DeoxyHb	PHD	0.879
Rab5c	CF AVG	0.405
Rab5c	GOR	0.604
Rab5c	PHD	0.787
Prealbumin	CF AVG	0.449
Prealbumin	GOR	0.772
Prealbumin	PHD	0.780

Step 3: Calculate the appropriate test statistic

What is the appropriate test statistic here? Recall the discussion in Chapter 13 about the t-distribution being the sampling distribution of the mean for a small size sample (which cannot be approximated as normal when n<30 roughly). Since our interest here is in testing the mean of the sample, it seems reasonable to use a t-distribution as our sample distribution. Recall that

Permission granted to reproduce for nonprofit, educational use.

$$t = \frac{\overline{X} - \mu}{s/\sqrt{n}}$$

has a t distribution on *n-1* dfs, where **s** is the estimate of **σ**.

So, under the null hypothesis with the theoretical mean, μ, equal to 0.5 the quantity

$$\frac{\overline{X} - 0.5}{s/\sqrt{n}}$$

is distributed as a t-distribution with n-1 degrees of freedom. When we use this quantity and substitute in our sample data (calculated mean of the sample for $\overline{X}$ and $s/\sqrt{n}$) this evaluated quantity is our test statistic.

Let's use R to calculate the test statistic:

```
> percentCorrect
 [1] 0.467 0.645 0.868 0.472 0.844 0.879 0.405 0.604 0.787 0.449 0.772 0.780

> sampleMean<-mean(percentCorrect)
> sampleMean
[1] 0.6643333

> mu<-0.5

> s<-sd(percentCorrect)
> s
[1] 0.1792481

> n<-length(percentCorrect)
> n
[1] 12

> testStatistic<-(sampleMean-mu)/(s/n^0.5)
> testStatistic
[1] 3.175863
```

Step 4: Perform the appropriate test

Our test is to determine where our test statistic, as evaluated, falls on the appropriate sampling distribution and whether this location on the sampling distribution indicates that the test statistic value is too extreme to be probabilistically considered as part of the distribution of the null hypothesis.

What we want is whether this test statistic value falls within the range of our hypothesized value (0.5) of the mean (and thus is it is reasonable to assume our test statistic comes from the hypothesized t-distribution) or whether our test statistic falls in the extreme regions of the distribution and thus we should reject

Permission granted to reproduce for nonprofit, educational use.

the null hypothesis. Notice that what we have just evaluated in calculating the test statistic is the distance between the target and the sample average in terms of standard errors. The test statistic tells us that that the sample average is about 3.175863 standard errors more than the target mean of 0.5. Is this a significant distance?

We need some sort of a cutoff criterion in order to determine this. The criterion is called the alpha-level, or the type one error level. It is set by the researcher and is the percentage of area under the sampling distribution curve that we consider too extreme to accept the null hypothesis. An ordinary alpha level for a two-tailed t-test (which is what we are doing here) is 0.05, indicating that a value of the test statistic that falls in the extreme 5% of the t-distribution (with n-1 degrees of freedom) is too extreme to accept the null hypothesis. For a two-tailed test this is split into 2.5% of the lower end of the curve, and 2.5% of the upper end of the curve. Other common alpha levels are 0.10, 0.01, and 0.001. The lower the alpha level the more stringent the decision criteria, or in other words, in order to be able to reject the null hypothesis the test statistics has to be very extreme.

Let's use an alpha level of 0.05 to perform our test and reach our conclusion with the help of R. Let's plot where our test statistic falls on the sampling distribution under the null hypothesis:

```
> x <- seq(-5,5,by=.1)
> plot(x,dt(x,df=n-1),xlab="t",ylab="",type='l')
> abline(v=testStatistic,lty=2)
> legend(3,0.2,legend="3.175863")
```

Figure 14-1 shows the result:

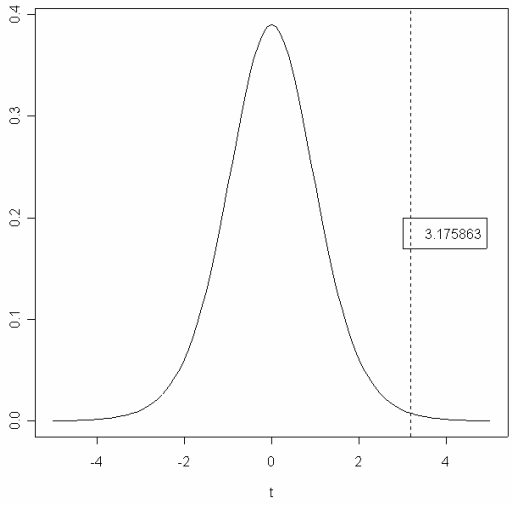


Figure 14-1

Permission granted to reproduce for nonprofit, educational use.

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.

Permission granted to reproduce for nonprofit, educational use.

Statisticians therefore have some formal decision criteria, the terminology of which is introduced here.

Whenever you perform a hypothesis test, there are four possible outcomes, listed in Table 14-3.

Table 14-3

	Actual Validity of H_o		
Decision Made		**H_o is true**	**H_o is false**
	Accept H_o	True negative	False Negative (Type II Error)
	Reject H_o	False Positive (Type I Error)	True Positive

Two of these outcomes are "correct" decisions – you reject a hypothesis when it should be rejected (true positive), or you accept a hypothesis when it should be accepted (true negative). The other two outcomes are not "correct" but a result of the statistical uncertainty inherent in a hypothesis test. We call these outcomes false results or "errors".

To understand errors, imagine a production assembly line and the null hypothesis (H_0) is that there is nothing wrong with production (product failure rate below a certain hypothesized criterion). A type I error is the rejection of H_0 given that H_0 is true (this is a conditional probability statement). The consequence of this is wrongly stopping production and monetary loss to the company. The type I error rate is called the significance level of the test, and is exactly the alpha level discussed earlier. If the alpha level for a test is 0.05, this means that 5% of the time we reject the null hypothesis and shut down the production line even though there is nothing significantly wrong. When we set the alpha level we are setting what is an acceptable type I error rate for the experiment.

A type II error is the failure to reject the null hypothesis when in fact H_0 is false. In this case production goes on even though there really does exist a higher than acceptable product failure rate. This may be bad for business in the long run. The type II error rate is sometimes referred to as the beta (β) level.

Power of a Test

The rate at which a statistical test can detect a true positive (rejecting the null when it should be rejected and accepting the alternative) is referred to as the "power" of the test. This is 1 minus the false negative rate or 1- β. We will not get into a theoretical discussion of power here, as it is covered in most

Permission granted to reproduce for nonprofit, educational use.

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
```


Permission granted to reproduce for nonprofit, educational use.

```
              sd = 1
       sig.level = 0.05
           power = 0.9886
     alternative = two.sided
```

Indeed the power of this test is 0.9886 meaning 98.86% of the time the test will reject the null hypothesis when it should. This is not bad and much better than the 0.5645 result to detect a delta of 0.5.

Maybe we think it would be a good idea to use fewer samples, perhaps to save money. Let's see what effect reducing the sample size to 10 has on the same test:

```
> power.t.test(n=10,delta=1,sig.level=0.05, type="one.sample")

     One-sample t test power calculation

              n = 10
          delta = 1
             sd = 1
      sig.level = 0.05
          power = 0.8031
    alternative = two.sided
```

Reducing the sample size reduces the power of the test to 0.8031 all other factors being equal. Maybe it pays to use n=20 as a sample size? To illustrate the effect of significance level perhaps we can go with the n=10 sample size but instead use a higher alpha level.

```
> power.t.test(n=10,delta=1,sig.level=0.1, type="one.sample")

     One-sample t test power calculation

              n = 10
          delta = 1
             sd = 1
      sig.level = 0.1
          power = 0.8975
    alternative = two.sided
```

Using a higher alpha level does increase the power to.0.8975. It is up to the researcher to determine what criteria for the test are acceptable and what is an acceptable power. Using the `power.t.test` function is a little tricky at first, but if you become skilled at using it (and the help function provides more details about the parameter options) it can be a valuable tool to use when determining sample sizes and significance levels to use for a particular test you want to work with

Permission granted to reproduce for nonprofit, educational use.

Hypothesis Testing Using R

Most standard hypothesis test in R have pre-written functionality so all you need to know to effectively use R for hypothesis testing is which test to use, how to call the test function with the appropriate parameters (use `help(name of test)` to get a help window with parameter details) and be able to interpret the test output (especially the p-value). Different tests differ in what is tested, in test statistics and how they're calculated and sampling distributions of the test statistic (or other criteria for nonparametric tests). The next two sections introduce how to do some very common tests in R using some examples. The R outputs for different tests show similar forms. The first section covers basic parametric tests, and the second section covers some nonparametric tests and introduces the rationale for using nonparametric tests.

Package ctest

Package `ctest` should be included with the basic installation of R (if not it is available on the CRAN site). Depending on the version of R you are using you may or may not specifically need to load package `ctest` in order to use it (version 1.6 has it preloaded).

Package ctest contains a standard set of hypothesis test functions that are typical of those taught in a basic statistics course. Most of the tests discussed in the remainder of this chapter are in this package. In addition, most of the test functions are flexible and have variable parameter options, allowing for different types of test possible within the same test function. Use the help function with the appropriate test name for further details about test functionality and parameters. Use function `ls(package:ctest)` to produce the list of specific functions available. For version 1.6, these are depicted in Figure 14-2.

Permission granted to reproduce for nonprofit, educational use.

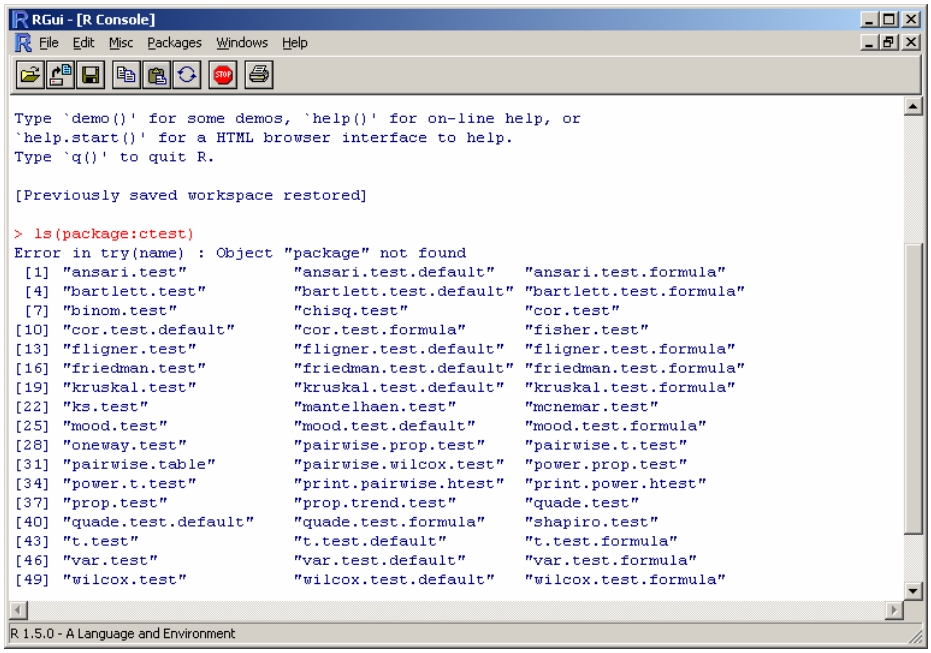


Figure 14-2: Contents of package ctest

Other Packages

Many other packages come with more sophisticated tests or tests that are application specific. For example, many of the microarray analysis packages available for R come with specialized hypothesis test functionality.

Select Parametric Tests

Parametric tests are based on the assumption the data come from a parameterized probability density. For most parametric tests the distribution of the test statistic is one of the sampling distributions discussed in Chapter 13 – the t-distribution, the Chi-Square distribution or the F-distribution.

The t-tests

The t-tests are a group of tests based on using the student's t distribution as a sampling distribution. Recall from the previous chapter that using the student's t distribution is appropriate to model the mean when you are sampling from a distribution that can be approximated as normal when the sample size becomes large. Normality of the data should be checked using a Q-Q plot, as described in Chapter 7, as normality is a critical assumption for t-tests results to be valid. There are three common forms of the t-test discussed here: the one sample t-test,

Permission granted to reproduce for nonprofit, educational use.

the two-sample t-test, and the paired t-test. All three tests are called using the function `t.test` but use different parameters to specify the test requirements.

One Sample t-test

The one sample student's t-test was illustrated earlier in our in depth analysis of a hypothesis test, but let's include an example here for comparison purposes to the other two forms of the t-test and to show how R automates functionality for this test.

A one-sample t-test is used to compare a single value of a mean of a test statistic result to a hypothetical mean under the null hypothesis. The test statistic for the one-sample t-test, detailed earlier, is:

$$t = \frac{\overline{X} - \mu}{s / \sqrt{n}}$$

Let's consider a small dataset of gene expression data (available as geHT data in package rbioinfo). This dataset consists of gene expression measurements for ten genes under control and treatment conditions, with four replicates each.

```
> geHTdata
     names   c1   t1   c2   t2   c3   t3   c4   t4
1   Gene 1 2650 3115 2619 2933 2331 2799 2750 3200
2   Gene 2 1200 1101 1200 1309 1888 1901 1315  980
3   Gene 3 1541 1358 1401 1499 1256 1238 1625 1421
4   Gene 4 1545 1910 1652 2028 1449 1901 1399 2002
5   Gene 5 1956 2999 2066 2880 1777 2898 1999 2798
6   Gene 6 1599 2710 1754 2765 1434 2689 1702 2402
7   Gene 7 2430 2589 2789 2899 2332 2300 2250 2741
8   Gene 8 1902 1910 2028 2100 1888 1901 2000 1899
9   Gene 9 1530 2329 1660 2332 1501 2298 1478 2287
10 Gene 10 2008 2485 2104 2871 1987 2650 2100 2520
```

Let's use a one-sample t-test to compare the hypothesis that the mean of the control expression values is 2000:

```
> #create data vector of all control data
> controls<-c(geHTdata$c1,geHTdata$c2,geHTdata$c3,geHTdata$c4)

> #perform one-sample t test that true mean is 2000
> t.test(controls,mu=2000)

        One Sample t-test
data:  controls
t = -2.174, df = 39, p-value = 0.03583
alternative hypothesis: true mean is not equal to 2000
95 percent confidence interval:
 1715.028 1989.722
sample estimates:
mean of x
 1852.375
```


Permission granted to reproduce for nonprofit, educational use.

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
```


Permission granted to reproduce for nonprofit, educational use.

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 measures 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, so we can pool the data for these two genes:

```
> g4g9ctrl
[1] 1545 1652 1449 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
```


Permission granted to reproduce for nonprofit, educational use.

The p-value for the test indicates there is a significant difference for this particular gene in treatment conditions versus control, and we can reject the null hypothesis of no difference.

Binomial Test

Bernoulli trials and the binomial distribution were discussed in Chapter 7. Recall that a Bernoulli trial models an experiment that has one of two discrete outcomes. The binomial test is, as described in the R help for `binom.test`, "an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment".

For example, suppose we perform an experiment crossing flowers of two genotypes that produce progeny with white flowers (recessive) of assumed proportion ¼ and progeny with purple flowers (dominant) of assumed proportion ¾. Suppose we want to test our assumption (null hypothesis) that these proportions are correct given that we have empirical data from 900 plants, 625 of which have purple flowers, and the remainder (275) have white flowers.

To do this we can use the binom.test function with parameters x=number of successes (purple flowers), n=total in sample, and p=proportion of successes to be tested (3/4).

```
> binom.test(x=625,n=900,p=3/4)

        Exact binomial test

data:  625 and 900
number of successes = 625, number of trials = 900, p-value = 0.0001593
alternative hypothesis: true probability of success is not equal to 0.75
95 percent confidence interval:
 0.663193 0.724417
sample estimates:
probability of success
             0.6944444
```

The results here give the empirical proportion of successes (0.6944) as well as a very small p-value, which significantly rejects the null hypothesis that the proportion of successes can be modeled at 0.75. Scientifically this can be interpreted as the relationship between dominant and recessive cannot be explained by simple Mendelian ratios for this genotype. The rejection of the null may seem surprising here, but since the sample size is large and the test is exact it is a convincing result. It is often useful to compute a confidence interval after a null hypothesis has been rejected. Again the R command shows the 95% confidence interval for the proportion of successes being between 0.663 and 0.724, with the hypothesized value of 0.75 lying outside the 95% confidence interval.

Permission granted to reproduce for nonprofit, educational use.

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 13 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

Permission granted to reproduce for nonprofit, educational use.

be approximated as normal, and when using non numerical (rank, categorical) data. Many nonparametric tests are built into R, either as part of package ctest or as part of an add-on package. There are dozen's of nonparametric tests in use. A good resource for further study is Conover's text "Practical Nonparametric Statistics" (more information given in the appendix). Test statistics for most nonparametric tests are not from a standard distribution, but are instead calculated from a test-specific test statistic and values for which are obtained from a standard table or built into R.

Wilcoxon Tests

Wilcoxon tests are the nonparametric analog of the t-tests. Instead of using a mean value derived from a t-distribution they use a median value as a test criteria. The median is simply the central point of the data, 50% of the data fall above the median and 50% fall below. For small datasets or data not normally distributed the median provides a good central measure of the data and the Wilcoxon tests are a good alternative to the t-tests in these cases.

The one-sample Wilcoxon test is often called the "Wilcoxon Signed Rank" test. This test determines whether the median of the empirical data differs significantly from a hypothesized median. This test is very simple to use in R, simply enter the data vector and the hypothetical median ("mu") as parameters. For example, let's use the Wilcoxon test to test whether the median of the protein prediction percent correct differs significantly from 0.5 (note this is the parallel of the one-sample t-test performed earlier)

```
> wilcox.test(percentCorrect,mu=0.5)

        Wilcoxon signed rank test

data:  percentCorrect
V = 68, p-value = 0.02100
alternative hypothesis: true mu is not equal to 0.5
```

The results of this test concur with the results of the t-test that the central measure (mean or median) of the data differs significantly from 0.5.

The `wilcox.test` function, like the `t.test` function, also provides for two-sample and paired tests. Use `help(wilcox.test)` for information on how to perform these tests and what parameters to use. These tests work in a parallel fashion to their t-test counterparts.

Chi-Square Goodness of Fit Test

The Chi Square goodness of fit test is a very simple and versatile test that quantitatively determines whether a random variable really should be modeled with a particular probability distribution. It literally tests the "goodness" of the density fit. The way the test works is it partitions the observed data into bins and

Permission granted to reproduce for nonprofit, educational use.

calculates the frequencies in each bin, similar to a histogram construction. It then compares the observed frequencies with the expected frequencies that would result from a perfect fit to the proposed distribution. It then calculates a test statistic that follows a chi square distribution, where o_i are the observed frequencies and e_i are the expected frequencies values for n bins:

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{o_i}, \mathrm{df} = n - 1$$

Note that this does use the Chi Square distribution but is a nonparametric test because the sample is not from the Chi Square distribution but we are testing the fitness of the distribution that the sample does come from.

This test is best illustrated with an example. We think that that the proportion of nucleotides A, T, C, G should be equally present in a given DNA sequence, with proportion 0.25 for each. This is modeled by a multinomial distribution (see Chapter 8). Let's see for a particular gene how good a fit this really is. That is our null hypothesis is that the data fit a multinomial distribution with equal probability for each nucleotide.

To perform an example using the proportion test in R, it is useful to note here that package `ape` (Analysis of Phylogenetics and Evolution) contains a function called `read.Genbank` that connects to Genbank (http://www.ncbi.nlm.nih.gov/) and downloads sequences into R for you. Make sure you have the package installed and loaded and you are connected to the Internet to use this.

Let's use this function to download a sequence into R, and then test that sequence with a chi square goodness of fit test to see if it conforms to a multinomial distribution. For this example we will use the sequence for human myoglobin, a muscle protein which carries oxygen. First we need to obtain the sequence.

```
> #Use nucleotide accession number for reference call
> ref<-c("NM_005368")

> #Store sequence in variable myoglobin
> #Call genbank with read.Genbank function and references
> myoglobin<-read.GenBank(ref)

> #Myoglobin sequence is now an R data object
> myoglobin
$"NM_005368"
   [1] "a" "c" "c" "c" "c" "a" "g" "c" "t" "g" "t" "t" "g" "g" "g" "g" "c"
….
[1059] "a" "a" "c" "a" "t" "c" "t" "c"
```

The goodness of fit test functionality is built into R as a default for function `chisq.test` when the input parameter is one vector of data and can be performed as below.

Permission granted to reproduce for nonprofit, educational use.

```
> #Data are already binned using table function
> table(myoglobin)
myoglobin
  a   c   g   t
237 278 309 242

> obs <- c(237, 278, 309, 242)
> chisq.test(obs,p=rep(1/4,4))

        Chi-squared test for given probabilities

data:  obs
X-squared = 12.79, df = 3, p-value = 0.005109

> # or even simpler using default p's
> chisq.test(obs)

        Chi-squared test for given probabilities

data:  obs
X-squared = 12.79, df = 3, p-value = 0.005109

>
> #Calculate where the alpha=0.5 level
> #of a df=n-1=3 Chi Square distribution is
> qchisq(0.95,df=3)
[1] 7.814728
```

Based on result of a test statistic 12.79 (more extreme on the Chi-Square curve with 3 degrees of freedom than 7.81) we fail to accept our null hypothesis at an alpha=0.05 significance level and conclude that the fit differs significantly from a multinomial with equal probabilities for each nucleotide.

Analyzing Contingency Tables

Contingency tables are a simple, yet powerful, method of analyzing count data that fits into categories. The data in them is count data as all we are doing is counting how many we observe which fit into which category, and not data measured on a continuous scale. This is also called categorical data analysis. We organize the data for analysis in a table format. The most typical case of a contingency table is a 2by 2 table (2 rows, 2 columns), although the table size can be extended to r (rows) by c (columns). We will only consider 2 x 2 tables here, looking at the setup of these tables and then considering how to analyze the data using some standard test methods available in R.

In a 2 x 2 contingency table there are four cells for the possible combinations of two factors. Where o11 is the number of observations that fit into row 1, column 1, etc.

Permission granted to reproduce for nonprofit, educational use.

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.

Permission granted to reproduce for nonprofit, educational use.

Chi-Square Test (for Independence)

The Chi-Square test for independence is similar to the goodness of fit test, but uses for the expected values the values calculated from the marginal probabilities of the data rather than expected values based on a distribution. That is, the expected value is the total number N times the expected probability based on the marginals, for example for the first cell:

$$e11=N^* p11=N(\frac{R_1}{N} * \frac{C_1}{N})$$

The Chi-Square test statistic is therefore the sum over all cells the squared observed minus expected value divided by the observed value. The degrees of freedom for this test are the number of rows minus 1 times the number of columns minus 1, which is 1 in the case of a 2 x 2 table.

$$\chi 2 = \sum_j \sum_i \frac{(o_{ij} - e_{ij})^2}{o_{ij}}, df = (r-1)(c-1)$$

To perform the test in R use the function `chisq.test` with the appropriately formatted table as a parameter:

```
> chisq.test(contingencyTestData)

        Pearson's Chi-squared test with Yates' continuity correction

data:  contingencyTestData
X-squared = 34.6624, df = 1, p-value = 3.921e-09
```

Based on this result (p-value of 3.921e-9) we would strongly reject the null hypothesis that the factors gene allele and disease are independent and conclude that there is a significant relation between the disease and which allele of the gene a person has.

Fisher's Exact Test

Fisher's exact test calculates every possible combination of the N values in the table (that is every possible permutation of N values into the 4 categories), creating a distribution of possible values, and calculates how extreme the observed data are on this exact distribution. It answers the question "How extreme is this table" out of all possible tables with the same sample size N. This test is extremely precise but computationally intensive (impossible to do without a computer). This type of test is referred to as a permutation test, and such tests are becoming popular in bioinformatics applications because of their precision.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

R is one of the few statistical software packages, which easily computes such test values. Use function `fisher.test` and enter the data matrix as a parameter.

```
> fisher.test(contingencyTestData)

        Fisher's Exact Test for Count Data

data:  contingencyTestData
p-value = 2.041e-09
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.1195958 0.3652159
sample estimates:
odds ratio
 0.2105543
```

The p-value result here is slightly different from the Chi-Square test value, but indicates the same result.

Likelihood Ratio Tests

Likelihood ratio tests are commonly used in bioinformatics and are category of hypothesis test. Because the mathematical theory behind them is somewhat complex, we will not cover them in depth here. Basically the likelihood ratio test consists of taking the ratio of the likelihood function under the null hypothesis over the likelihood function under the alternative hypothesis to determine a critical test region. This provides the "best test" providing the highest power with the lowest error (for mathematical reasons beyond our scope here). The BLAST database uses a likelihood ratio test to detect homologies between gene sequences. Likelihood ratio tests are common in other genetics applications as well. The R package `emplik` contains specific functions for testing various types of likelihood ratios, and several other genetics related packages (discussed in Chapter 18) contain some likelihood ratio test functionality.

Permission granted to reproduce for nonprofit, educational use.

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.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
 > 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.

Permission granted to reproduce for nonprofit, educational use.

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

Permission granted to reproduce for nonprofit, educational use.

squares between is the sum of squared distances from the group average to the grand average. The sum of squares within is the squared distance from each data value to the group average.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Method	Xij	Correct	Xi average	(Xij - Xi av)	(Xij - Xi av)^2	X average	(Xi av-X av)	(Xi av-X av)^2	(Xij - X av)	(Xij - X av)^2
CF AVG	X11	0.467	0.448	0.019	0.000	0.664	-0.216	0.047	-0.197	0.039
CF AVG	X12	0.472	0.448	0.024	0.001	0.664	-0.216	0.047	-0.192	0.037
CF AVG	X13	0.405	0.448	-0.043	0.002	0.664	-0.216	0.047	-0.259	0.067
CF AVG	X14	0.449	0.448	0.001	0.000	0.664	-0.216	0.047	-0.215	0.046
GOR	X21	0.645	0.716	-0.071	0.005	0.664	0.052	0.003	-0.019	0.000
GOR	X22	0.844	0.716	0.128	0.016	0.664	0.052	0.003	0.180	0.032
GOR	X23	0.604	0.716	-0.112	0.013	0.664	0.052	0.003	-0.060	0.004
GOR	X24	0.772	0.716	0.056	0.003	0.664	0.052	0.003	0.108	0.012
PHD	X31	0.868	0.828	0.040	0.002	0.664	0.164	0.027	0.204	0.042
PHD	X32	0.879	0.828	0.051	0.003	0.664	0.164	0.027	0.215	0.046
PHD	X33	0.787	0.828	-0.041	0.002	0.664	0.164	0.027	0.123	0.015
PHD	X34	0.780	0.828	-0.048	0.002	0.664	0.164	0.027	0.116	0.013
SUM				**0**	**0.048**			**0.305**	**0**	**0.353**

Sum of Squares WITHIN (SSW)

Sum of Squares BETWEEN (SSB)

Sum of Squares TOTAL (SST)

Figure 15-1

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Note that in Figure 15-1 the following relation holds:

Sum of Squares TOTAL

= Sum of Squares WITHIN + Sum of Squares BETWEEN

This can be simplified as:

SST = SSW + SSB

Thus the total variation is partitioned into two parts – the variation due to treatment (SSB) and the variation due to random variation (SSW). This partitioning of the variance is the core essence of how ANOVA works as this relation always holds.

However, the values of the sum of squares in and of themselves are not interesting and do not provide us within enough information to perform a statistical inference test. To do such a test we must work some more with these values and their interpretations.

Recall the brief discussion in Chapter 13 regarding using the F distribution as the distribution of the ratio of variances which can be used to analyze signal to noise ratios. ANOVA is a case of such a signal to noise ratio, and significance tests using ANOVA use the F distribution. The signal here is the between variation which represents changes in the mean response due to treatments (prediction methods in this case). The noise is the within variability (sometimes called error) since this represents random variation in repeated observations under the same factor level (same protein prediction method in this case).

Therefore, what we want a test of is whether we have a real signal – a real difference between the treatments, in relation the amount of noise we see in the experimental data. However, in order to compare our signal to our noise we cannot directly compare our SSW and SSB values. Why is this? If you look at the data in Figure 15-1 you may notice that for the sum of squares between there are only three unique values. And if you took the SSB value and subtracted two of the values, you automatically get the third value. The SSW has more than 3 unique values.

Each sum of squares value has associated with it a number called the degrees of freedom, a concept we have encountered before. For the sum of squares between, the degrees of freedom is the number of treatments (which we'll designate with a lowercase letter a) minus 1. For the sum of squares total the degrees of freedom is N (total number of data values) minus 1. Like the sum of squares the degrees of freedom also have the relation:

Total df = Within df + Between df

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

So the degrees of freedom for the sum of squares within is N-a, yielding the relation:

$$\text{Total df} = \text{Within df} + \text{Between df}$$

$$(N-1) = (N-a) + (a-1)$$

Which algebraically makes sense. In our example we have 11 total degrees of freedom, 9 degrees of freedom within, and 2 degrees of freedom between. We cannot directly compare SSW and SSB to get our signal to noise value because these values are based on different numbers. So we need to standardize our values to make the comparison level. Taking averages does this and we call these averages "mean squares".

Mean squares are calculated by taking the sum of squares and dividing them by their respective degrees of freedom. We are interested in the Mean Square Between (MSB) and the Mean Square Within, usually called Mean Square Error (MSE), which are calculated as follows, by taking the respective sum of squares and dividing by the respective degrees of freedom:

$$\text{MSB} = \frac{\text{SSB}}{a-1}$$

$$\text{MSE} = \frac{\text{SSW}}{N-a}$$

For our example these calculations are:

$$\text{MSB} = \frac{0.305}{2} = 0.1525$$

$$\text{MSE} = \frac{0.048}{9} = 0.00533$$

Finally we have the values we need to conduct a statistical test of significant differences between the protein structure prediction methods. The test utilizes the F distribution and takes for a test statistic the ratio of the MSB/MSE, our "signal to noise" ratio. The F test statistic value here is 0.1525/0.00533 or approximately 28.6. Recall from the discussion of the F distribution that this distribution depends on numerator and denominator degrees of freedom, which are the SSB degrees of freedom and the SSW degrees of freedom. In this case these are 2 and 9 respectively. Therefore we can use R to determine the critical value of this F distribution given alpha=0.05:

```
> qf(0.95,2,9)
[1] 4.256495
```


Permission granted to reproduce for nonprofit, educational use.

Thus, our test statistic value of ~28.6 is more extreme on the F distribution than the critical value of 4.257 and we conclude there is a significant difference among the three protein secondary structure prediction methods.

The ANOVA Table

All of the computations done in the discussion above are easily summarized into a convenient standardized format known as the ANOVA table, depicted in Table 15-2. R (and other statistical software packages) formats out for ANOVA analysis in this type of format.

Table 15-2

Source	Degrees of Freedom	Sum of Squares	Mean Squares	F Ratio
Factor	a-1	SS(between)	MSB	$\frac{MSB}{MSE}$
Error	N-a	SS(error)	MSE	
Total	N-1	SS(total)		

Performing ANOVA using R

Performing ANOVA analysis using R is fairly simple. To do so, first create a linear model object using function `lm`. This function will be discussed later in this chapter when it is used for regression. Apply the ANOVA function to the linear model object created. To do this, you can nest the function calls, as demonstrated below for our sample data:

```
> anova(lm(Correct~Method,data=protStruct))
Analysis of Variance Table

Response: Correct
          Df   Sum Sq  Mean Sq F value    Pr(>F)
Method     2 0.305352 0.152676  28.581 0.0001263 ***
Residuals  9 0.048077 0.005342
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

R refers to the sum of squares between by the group name (method row) and the sum of squares within as part of the "residuals" row.

The above example is the simplest possible case of ANOVA. This simple model can be built upon to create much more complicated models but they all follow the same paradigm of partitioning the variance and are all run in R in a similar fashion. We will do some two-factor ANOVA later in this chapter but

Permission granted to reproduce for nonprofit, educational use.

now let's consider another issue, determining which means are significantly different.

Determining Which Groups Differ

ANOVA analysis tells us whether there is a significant difference between 3 or more groups (treatments, factor levels). However, it does not determine which groups differ. For example based on the ANOVA analysis above we know that there is a significant difference somewhere between protein prediction methods PHD, GOR and CF (Cho Fasman) AVG. However, we don't know the specifics of whether PHD differs from GOR significantly, or GOR differs from CF AVG, etc.

To determine which methods differ from one another, we can use the `pairewise.t.test` function. This will produce a pairwise comparison of the methods and produce a table of p-values for the significant differences of specific pairs. This command requires us to attach the data frame, so that its variables can be used as individual objects.

```
> attach(prot.Struct)
> pairwise.t.test(Correct,Method)

        Pairwise comparisons using t tests with pooled SD

data:  Correct and Method

    CF AVG  GOR
GOR 0.00115 -
PHD 0.00013 0.05793

P value adjustment method: holm
```

Based on this output for this experiment, it can safely be concluded there is no difference between the GOR and PHD methods but that both of these differ significantly from the CF AVG method.

In much of microarray gene expression data analysis researchers are faced with performing sometimes as many as hundreds to thousands of comparisons tests. In such situations the Bonferroni adjustment would require to select individual alpha levels of 0.0001 or smaller, which makes it virtually impossible to detect any significant differences at all, in other words the power of such tests is near zero. Even the Holm adjustment, which is less stringent, would still not work for so many tests. Therefore researchers have developed a less restricting adjustment of the α-level that can be applied to a larger number of simultaneous tests. It is called the False Discovery Rate (FDR) and was proposed by Benjamini and Hochberg(1995). The False Discovery Rate controls the false discovery rate, the expected proportion of false discoveries amongst the rejected hypotheses. So, a FDR rate of 0.05 says that we can tolerate at most 5% of false rejections among all our tests. Let's see how a 5% FDR rate compares with the

Permission granted to reproduce for nonprofit, educational use.

5% Bonferroni adjusted and the 5% Holm adjusted P-values for our 3 pairwise comparisons.

```
> pairwise.t.test(Correct,Method,p.adj="fdr")

        Pairwise comparisons using t tests with pooled SD

data:  Correct and Method

    CF.AVG  GOR
GOR 0.00086 -
PHD 0.00013 0.05793

P value adjustment method: fdr
```

The result is that with only 3 pairwise tests, the FDR controlled P-values are nearly identical to those using the Holm adjustment for family-wise error rate. However the difference between the two methods would be dramatic if we performed several hundred tests simultaneously.

Graphical Analysis of ANOVA Comparisons

Often it is helpful to view means comparison data visually. This can be done in R using the `stripchart` function, which will plot the spread of data values for each group as depicted in Figure 15-2.

```
> stripchart(Correct~Method, vert=T)
> title("Comparing Secondary Structure Prediction Methods")
> #Calculating group means, sd, and sem
> xbar<-tapply(Correct,Method,mean)
> s<-tapply(Correct,Method,sd)
> sem<-s/sqrt(12)
> arrows(1:3,xbar+2*sem,1:3,xbar-2*sem,angle=90,code=3)
```


Permission granted to reproduce for nonprofit, educational use.

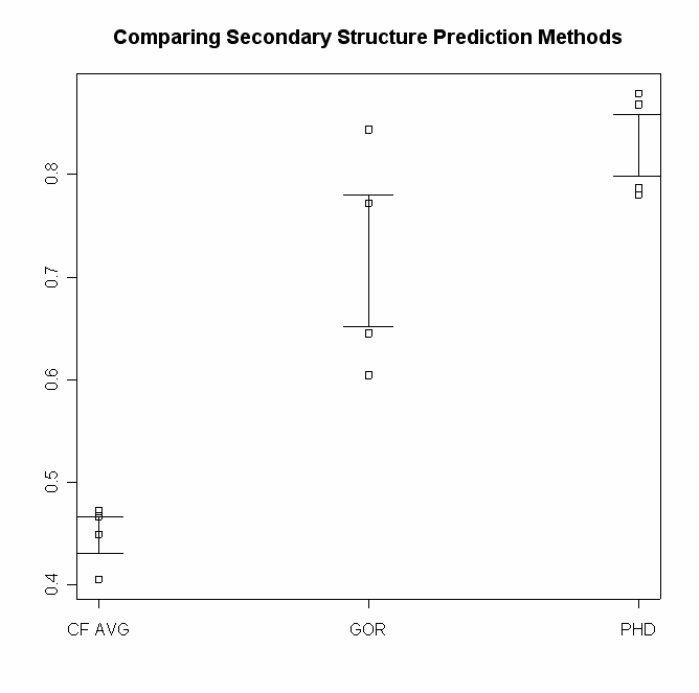


Figure 15-2

The addition of arrows (with flat heads at 90 degree angles) depicts the spread range of the mean within 2 standard errors, which is usually about the approximate range of a confidence interval. Using this range, if the arrow lines overlap (or come close) it indicates there is no significant difference between the groups, whereas if the arrows do not overlap there does exist a significant difference. Figure 15-2 provides a visual confirmation of our earlier conclusions.

Two-Factor ANOVA

Let's return to our initial dataset, which included which protein as well as the method used to determine secondary structure prediction. Suppose we want to analyze not only whether the percentage correct (result) is affected by the method of prediction used, but also whether the percentage correct is affected by which protein is being tested. In this analysis, both protein and method are factors. We call such an analysis a two factor ANOVA.

Adding another factor to the model is vary simple in R, simply use a plus sign and add the next factor to the `lm` function parameter. Nesting lm inside ANOVA creates a new ANOVA table as a result, this time with an additional line for the additional factor.

```
> anova(lm(Correct~Method+Protein))
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
Analysis of Variance Table

Response: Correct
          Df   Sum Sq  Mean Sq F value    Pr(>F)
Method     2 0.305352 0.152676 42.6844 0.0002832 ***
Protein    3 0.026615 0.008872  2.4803 0.1583825
Residuals  6 0.021461 0.003577
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Based on this analysis the method factor is significant but the protein factor is not. In the case of two-factor ANOVA the sum of squares between is split into two factors like this:

SS (between) = SS (Method) + SS (Protein)

The sum of squares is simply partitioned into partial sums of square components with one for each effect in the model. The relationship for total sum of squares still holds:

SST = SSW + **SSB**

Only now we can split SSB into two components, so:

SST = SSW + **[SS(method) +SS (protein)]**

The F-values are the mean square for the sum of squares for that method divided by the sum of squares error. An additional concern with ANOVA models incorporating two or more factors is the possibility that the factors interact and the interaction of factors is a significant concern. The analysis of interactions is not presented here, but this is a point of interest to note.

ANOVA Beyond Two Factors

The ANOVA analysis technique can be extended to all kinds of analyses and models. Although the complexity of these models is beyond coverage here, all of these rely on the basic analysis method discussed above. To study such models, some good references are presented in the appendix. In particular, books on the statistical discipline of experimental design have in-depth coverage of various types of ANOVA models. In bioinformatics, complex ANOVA models are often used in microarray data analysis.

Meanwhile let's proceed to studying the second major form of linear model, the regression model.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Regression Analysis

Regression analysis is the statistical technique used to model relationships between a set of input (or predictor) variables and one or more output (response) variables. The simplest form of regression analysis, called simple liner regression or straight-line regression involves the statistical modeling between a single input factor X (the "regerssor") and a single output variable Y (the "response"). We discuss simple linear regression and how to perform this analysis in some detail here.

First let's consider how to do a simple look at the relationship of two variables before producing a formal regression model.

Correlations

Correlation metrics are measures of associations between variables. It is important to note that association is a concept that has no implication of causation. Two variables can correlate quite nicely, yet have no cause/effect relationship. For example, in warm weather both the consumption of ice cream and the population of mosquitoes go up. Variables measuring each of these quantities would have a high correlation. This relationship however, is nothing but a coincidental relationship and is not of further statistical interest.

Nevertheless looking at correlations does provide a useful preliminary analysis for looking at relationships between variables. A correlation metric is called a correlation coefficient, usually designated by the letter r. There are three most commonly used correlation metrics: Pearson's correlation coefficient, Spearman's rho, and Kendall's tau. Pearson's correlation coefficient is determined by the mathematical formula:

$$r = \frac{\frac{1}{n-1}\sum(xi - \bar{x})(yi - \bar{y})}{SxSy}$$

where n is the number of data points, Sx and Sy are the standard deviations of the x and y data respectively, and $\bar{x}$ and $\bar{y}$ are the mean values of the data. Spearman's rho and Kendall's tau are nonparametric correlation coefficients and based on the ranks of the data.

All three correlation metrics have values between 1 and -1 where 1 is a perfect positive correlation, -1 is a perfect negative correlation, and 0 implies there is no correlation. All three correlations measure the strength of a linear relationship between the two variables. In the case of a perfect correlation of 1, the (x,y) data points are all exactly on a line with positive slope. This usually does not

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

happen with real data. As an example, consider the small dataset geneCorrData (this was available as part of the `rbioinfo` package which contains expression measurements for a time course of two genes under the same treatment condition in a prior release of R but has been retired). We can use R to analyze the correlation of gene expression for these two genes, linked in functionality, to study how they express together

First, just looking at the data, does there appear to be a relationship?

```
> geneCorrData
   gene1 gene2
1  -1.06 -1.08
2  -0.81 -1.02
3  -0.48 -0.39
4  -0.42 -0.48
5  -0.30 -0.58
6  -0.35 -0.24
7  -0.31 -0.05
8  -0.18 -0.33
9  -0.20  0.51
10 -0.11 -0.53
11 -0.09 -0.47
12  0.16  0.10
13  0.45  0.39
14  0.53  0.11
15  0.67  0.52
16  0.80  0.34
17  0.87  1.08
18  0.92  1.21
```

It helps with correlations to do a simple plot of the data, as depicted in Figure 15-3:

```
> plot(gene1,gene2)
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

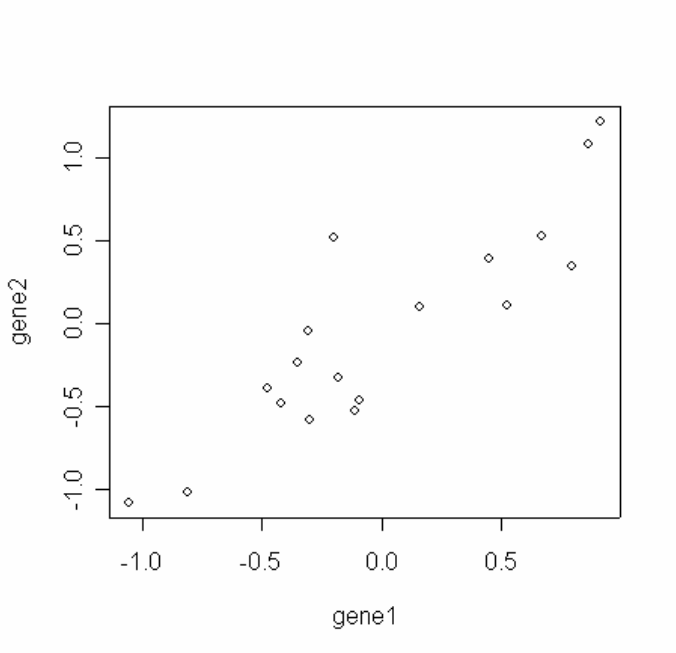


Figure 15-3

The plot demonstrates visually that there is evidence of a relationship but does not quantify it. Let's measure the strength of the linear relationship between gene 1 and gene 2 expression values using correlation coefficients. To do this we can use the R function cor.test. By default the test calculates Pearson's correlation coefficient, but other coefficients can be specified using the optional method parameter.

```
> cor.test(gene1,gene2)

        Pearson's product-moment correlation

data:  gene1 and gene2
t = 7.5105, df = 16, p-value = 1.246e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7071560 0.9556856
sample estimates:
      cor
0.8826268

> cor.test(gene1,gene2,method="spearman")

        Spearman's rank correlation rho
data:  gene1 and gene2
S = 192, p-value = 8.498e-05
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.8018576
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

This correlation, 0.88 or 0.80 depending which metric you use, is pretty strong. It says the association between the expression values for gene 1 (X values) and gene 2 (Y values) is positive and the values are highly correlated (as one goes up, so does the other). What it doesn't do is create any type of statistical model that relates how we would predict a new Y value given a new X data point. A simple linear regression model is required to do this.

The Basic Simple Linear Regression Model

Surely at some point in your mathematical career, probably in algebra class, you encountered the equation of a straight line, y=mx+b. In this equation m is the "slope" of the line (change in y over change in x) and b is the "intercept" of the line where the y-axis is intersected by the line. The plot of a line with slope 2 and intercept 1 is depicted in Figure 15-4.

Although the straight-line model is perfect for algebra class, in the real world finding such a perfect linear relationship is next to impossible. Most real world relationships are not perfectly linear models, but imperfect models where the relationship between x and y is more like Figure 15-3. In this case, the question literally is "Where do you draw the line?". Simple linear regression is the statistical technique to correctly answer to this question.

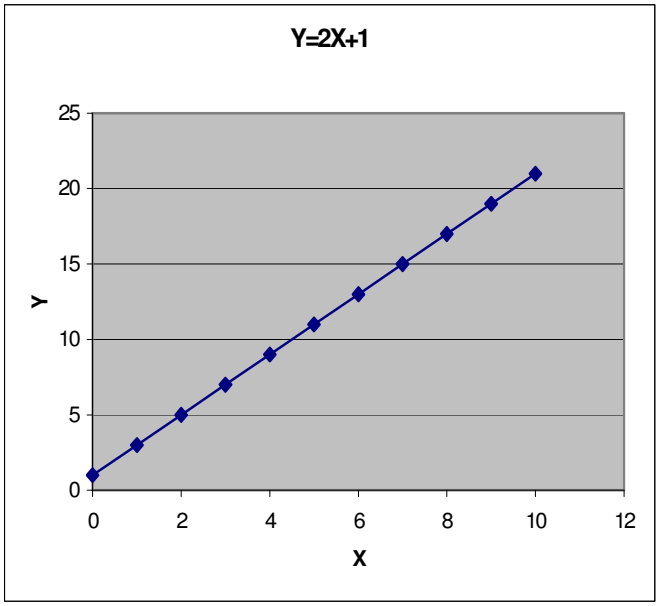


Figure 15-4

Permission granted to reproduce for nonprofit, educational use.

Simple linear regression is the statistical model between X and Y in the real world, where there is random variation associated with measured variable quantities. To study the relationship between X and Y, the simplest relationship is that of a straight line, as opposed to a more complex relationship such as a polynomial. Therefore in most cases we want to try to fit the data to a linear model. Plotting X versus Y, as done in Figure 15-3 is a good first step to determine if a linear model is appropriate. Sometimes data can be transformed (often by taking logarithms or other mathematical methods) to fit a linear pattern if they do not do so in the original measurement scale. Determining if a straight-line relationship between X and Y is appropriate is the first step. The second step, once it is determined a linear model is a good idea, is to determine the best fitting line that represents the relationship.

Fitting the Regression Model

Obviously, given Figure 15-3, you could simply take a ruler and draw in a line, which, according to your subjective eye, best goes through the data. This method is subject to much error and is unlikely you will produce the "best fitting" line. Therefore a more sophisticated method is needed.

To fit a regression model, some assumptions are made (presented here for thoroughness), these include:

1. For any given value of X, the true mean value of Y depends on X, which can be written $\mu_{y|x}$. In regression, the line represents mean values of Y not individual data values.
2. Each observation Yi is independent of all other Yj≠Yi.
3. We assume linearity between X and the mean of Y. The mean of Y is determined by the straight line relationship which can be written as:

   $\mu_{y|x} = \beta_0 + \beta_1 X$ where the betas are the slope and intercept of the line (note this closely resembles y=b+mx, the equation of a line).
4. The variance of the Yi's is constant (homoscedasticity property).
5. The response Yi's are normally distributed with a constant variance (see property 4).

Let's build on the third assumption. The actual observed values of Y differ from the mean of Y ($\mu_{y|x}$) by an error term (reflecting random variation). We can write this as:

$$Yi = = \beta_0 + \beta_1 X + \varepsilon i$$

Permission granted to reproduce for nonprofit, educational use.

The errors, ε i's, are the difference between the mean line and the actual Yi data values. We call these errors residuals.

In order to fit a regression model, we have parameters to estimate. In this case, we need good estimates for β1, the slope of the line, and β0, the y-axis intercept. We have a special method for estimating these called the method of least squares. Sometimes simple linear regression is referred to as least squares regression. The least square estimates minimize the sum of squares of the vertical distances between the points and the line, which corresponds with minimizing the squared residual values.

The model above with Greek letters for the beta values is the theoretical model. For the estimates of the theoretical parameters, we use the convention of lower case b's (an alternative notation uses hats on the Greek letters, such as $\hat{\beta}$) to denote that these are estimates based on the data not theoretical values.

The formulas for the least square estimates are as follows:

For b1 (slope of the line)

$$b1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{Sxy}{Sxx}$$

Where the numerator is also known as the sum of squares xy (Sxy) and the denominator is know as the sum of squares x (SSxx).

For b0 (intercept)

$$b0 = \bar{y} - b1\bar{x}$$

Of course, performing such calculations with large datasets is tedious. However, there is no need to do these calculations manually, as R provides simple functionality to perform them.

Once again, as with ANOVA, we will use the linear model function `lm`. To perform simple linear regression analysis in R, simply model the y and x data vectors of interest as lm(y~x) as follows for the gene expression data described earlier:

```
> geneLM<-lm(gene2~gene1)
> geneLM

Call:
lm(formula = gene2 ~ gene1)
Coefficients:
(Intercept)        gene1
   -0.05541      0.97070
```


Permission granted to reproduce for nonprofit, educational use.

In the above output R has calculated the b0 term (-0.05541) that is the y-intercept and the slope (listed as 0.9707). These terms are referred to as coefficients. As a challenge, you can verify these results using the formulas given for the coefficients and the empirical data. It is a good idea to store the linear model object created in a new variable so that it can be analyzed further.

Plotting the regression line makes the analysis more interesting. We can do this with a simple plot function, then adding a line with the fitted data values for y.

```
> plot(gene1,gene2)
> lines(gene1,fitted(geneLM))
```

To make things even more interesting we can add segments connecting the empirical data to the regression line. The length of these segments is the residual or error, representing the distance from the empirical data to the fitted regression value (mean of y line).

```
> segments(gene1,fitted(geneLM),gene1,gene2,lty=2)
> title("Fitted Model with Residual Line Segments")
```

Figure 15-5 shows the resulting plot. This model allows us to predict new y data (designated as y hat to indicate that it is an estimate) values based on an empirical x value with the formula.

$$\hat{y}_i = -0.05541 + 0.9707 * x_i$$

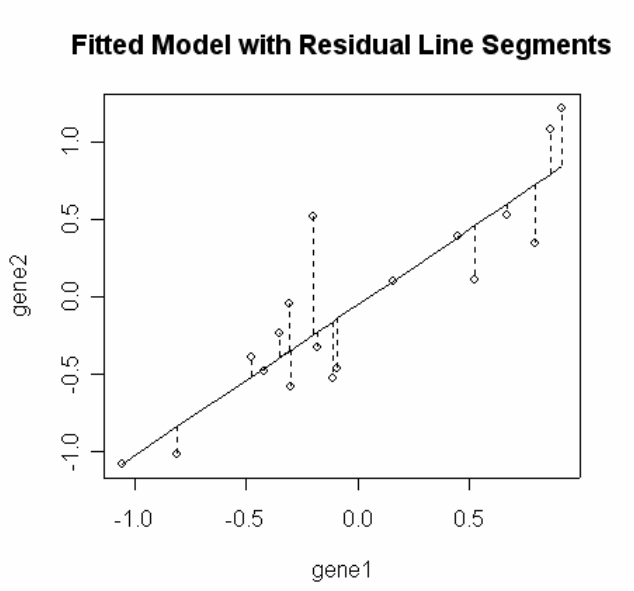


Figure 15-5

Note that regression is a tool to understand existing causal relationships not to create them. There are many misuses of regression where researchers falsely

Permission granted to reproduce for nonprofit, educational use.

conclude two variables are causally related just because they can find a good fitting regression model relating the variables.

If you want to look at the fitted values and the residuals, R provides functions `fitted` and `resid`. Supply these functions with the saved variable containing the linear model object to obtain the output of results:

```
> fitted(geneLM)
          1           2           3           4           5           6
-1.08435135 -0.84167628 -0.52134519 -0.46310317 -0.34661914 -0.39515415
          7           8           9          10          11          12
-0.35632614 -0.23013511 -0.24954911 -0.16218609 -0.14277208  0.09990299
         13          14          15          16          17          18
 0.38140607  0.45906209  0.59496013  0.72115116  0.78910018  0.83763519

> resid(geneLM)
            1             2             3             4             5
 4.351347e-03 -1.783237e-01  1.313452e-01 -1.689683e-02 -2.333809e-01
            6             7             8             9            10
 1.551542e-01  3.063261e-01 -9.986489e-02  7.595491e-01 -3.678139e-01
           11            12            13            14            15
-3.272279e-01  9.701311e-05  8.593934e-03 -3.490621e-01 -7.496013e-02
           16            17            18
-3.811512e-01  2.908998e-01  3.723648e-01
```

In fact, if you add the fitted values and the residual values together you get the empirical Y data values (gene 2).

```
> fit<-fitted(geneLM)
> res<-resid(geneLM)
> sumFnR<-fit+res
> compare<-data.frame(sumFnR,gene2)


> compare
   sumFnR gene2
1   -1.08 -1.08
2   -1.02 -1.02
3   -0.39 -0.39
4   -0.48 -0.48
5   -0.58 -0.58
6   -0.24 -0.24
7   -0.05 -0.05
8   -0.33 -0.33
9    0.51  0.51
10  -0.53 -0.53
11  -0.47 -0.47
12   0.10  0.10
13   0.39  0.39
14   0.11  0.11
15   0.52  0.52
16   0.34  0.34
17   1.08  1.08
18   1.21  1.21
```


Permission granted to reproduce for nonprofit, educational use.

Testing the Regression Model

Once we have a regression model we often want to do some inferential tests and analysis to see if the model is indeed a good fit or if we should adjust the model accordingly. In R the lm function is more than just an ordinary function. It is a special type of function that creates a model object. Those familiar with object-oriented programming know about the principle of encapsulation, where information is "hidden" inside an object. The linear model object when created contains hidden information that can be extracted with other functions. The summary function can be used to extract details of a linear model object. These details include information of use to diagnose the regression fit.

Applying the summary function to our linear model object yields the following output:

```
> summary(geneLM)

Call:
lm(formula = gene2 ~ gene1)

Residuals:
    Min      1Q  Median      3Q     Max
-0.3812 -0.2196 -0.0084  0.1492  0.7595

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.05541    0.07330  -0.756    0.461
gene1        0.97070    0.12925   7.511 1.25e-06 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 0.311 on 16 degrees of freedom
Multiple R-Squared: 0.779,      Adjusted R-squared: 0.7652
F-statistic: 56.41 on 1 and 16 DF,  p-value: 1.246e-06
```

The first section of output simply states the model. The next section is a summary of the residuals. Often in regression we look at the distribution of the residuals the check the assumption of normality. The distribution should be centered around zero, so the median value should be near zero, and it is so for our data.

Another good way to check normality by looking at the distribution of residuals is to do a Q-Q plot of the residuals to look for a linear pattern:

```
> qqnorm(resid(geneLM))
```

Figure 15-6 displays the result, which seems reasonably linear indicating the normality assumption of the model is not violated. This is one indication of a good model fit. A nonlinear pattern on a Q-Q plot casts doubt on the assumption of normality. This may or may not indicate a problem with the model fit, but deserves further exploration.

Permission granted to reproduce for nonprofit, educational use.

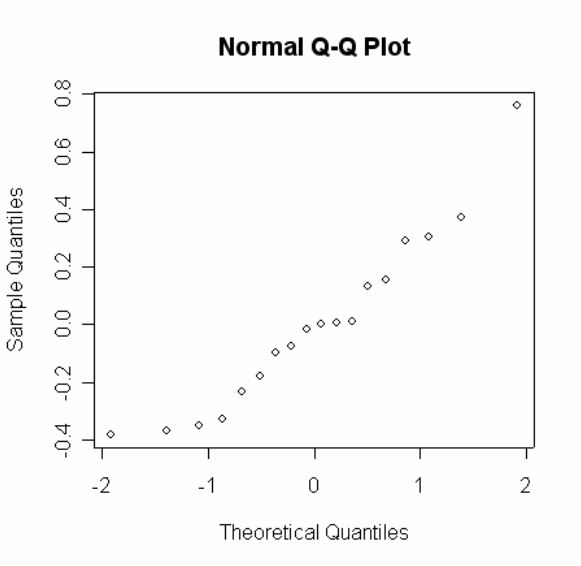


Figure 15-6

The other section of summary output that is of interest for testing simple linear regression models is the coefficients section. This section repeats least square estimate values for the slope and intercept coefficients but also gives output of statistical tests of these values, with a t-test statistics value and a p-value for the test statistic.

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.05541    0.07330  -0.756    0.461
gene1        0.97070    0.12925   7.511 1.25e-06 ***
```

The interesting test here is the test on the slope, which ids a test to see if the slope is significantly different from zero. In this case, with a p-value of 1.25e-06 our slope is highly significantly different from zero. However, if this test provides an insignificant p-value (0.05 or higher) this cast doubt on whether there is a strong linear relationship between x and y and perhaps there is no relationship or maybe another type of model would be a better fit. The intercept test is of little interest in practice.

Generalizing the Linear Model

Underneath the details, ANOVA and regression are really the same thing. Both are linear models that relate a response (Y) to input (X) through a linear relationship. In the cases above we looked at the case of one input variable, although this can easily be extended to include multiple input variables. Both ANOVA and regression take the form:

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Observed data = fit + residual

$$Y_{ij} = \text{fit} + \varepsilon_{ij}$$

In the case of one-way ANOVA we have the following:

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij}$$

Where the fit is the grand mean, μ plus the treatment effect for that group, designated with the Greek letter tau. If we expand this to two-way ANOVA the model is:

$$Y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij}$$

Now instead of one treatment effect we have two, alpha and beta, representing the effect of the two groups (in our example earlier, protein and prediction structure method).

For the regression model we have

$$Yi = = \beta_0 + \beta_1 X + \varepsilon i$$

Where the slope and intercept determine the fit based on the input data. In regression the input data is continuous data, whereas with ANOVA we are working with factor (discrete) variables.

The general paradigm for a linear model is:

$$Y = f(X) + \varepsilon$$

Where the output is a linear function of input variables plus an error term. There are dozens of variations of the linear model, which is a topic of advanced statistics courses. In all cases of the linear model the "fit" function f(X) is also linear in the parameters.. It is of note though that the input variables themselves need not be linear and can have exponents other than 1. Models where f(X) does not follow a linear relation in the coefficients are known as nonlinear models.

In bioinformatics, the use of linear models occurs frequently in quantitative genetics applications such as QTL mapping. ANOVA models can be used in designing experiments and analyzing microarray data, discussed further in chapter 19. R has extensive functionality for dealing with all types of linear and nonlinear regression models and many packages are devoted to specific advanced regression modeling applications. Select packages are featured in Table 15-3.

Permission granted to reproduce for nonprofit, educational use.

Table 15-3 Selected R Functionality for Linear Models

Package	**Select functions**	**Description**
Base installation	lm()	Linear regression model
Base installation	glm()	Generalised linear models
survival	coxph()	Cox model
	survreg()	Survival models
	clogit()	Conditional logistic regression
nlme	lme()	Linear mixed effects models
	nlme()	Non linear mixed effect models
gee	gee()	Generalised estimating equations
rmeta	Various	Models for meta analysis (fixed effects, random effects)

Permission granted to reproduce for nonprofit, educational use.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

16

Working with Multivariate Data

Many of the probability models introduced in earlier chapters are univariate models. However there is a large body of work in statistics devoted to the simultaneous analysis of several response variables. This area of statistics is called multivariate statistics. We have already introduced multivariate probability models; in particular the multinomial distribution is a multivariate model for discrete data. We have also discussed the bivariate normal distribution (Chapter 8), which is a continuous multivariate model.

Multivariate statistics can be challenging. In high dimensions there is the "curse of dimensionality": data spaces are vast, and it requires huge amounts of data to even moderately fill the space. Another serious problem is the edge effect issue. As the number of variables grows, a larger percentage of the data tends to be near the edge of the data space, yet many statistical approaches attempt to draw conclusions about the mean (center) of the data.

Another challenge is data and pattern description and visualization. We can only display low-dimensional projections of the data, such as histograms, scatterplots or perspective plots and rotating point clouds (three-dimensional projections). It is likely that important features of multivariate data get lost in these projections. In this chapter we discuss how multivariate data is summarized, and we introduce the general multivariate normal distribution. Furthermore we will introduce some techniques of analyze multivariate data which are becoming commonplace in biomedical literature.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The Multivariate Normal Distribution

We first generalize the bivariate normal distribution, discussed in Chapter 8, to the case of several variables. So far we only considered the bivariate standard normal distribution with zero correlation. The probability density function for higher dimensional normal probability models is fairly complex and requires the notation of linear algebra. Starting first with the case of two variables, say Y_1 and Y_2 we simply state the parameters that define the distribution, avoiding complicated algebra. They are: μ_1 and μ_2, the mean of Y_1 and mean of Y_2, respectively; σ_1 and σ_2, the standard deviation of Y_1 and of Y_2, respectively, and the correlation coefficient ρ_{12} between Y_1 and Y_2. Figure 16.1 displays random samples of size 500 from several different normal distributions whose parameters are given below in Table 16.1.

Table 16.1 Bivariate Normal Parameters of Figure 16.1

	μ_1	μ_2	σ_1	σ_2	ρ_{12}	σ_{12}
(a)	0	0	1	1	0	0
(b)	2	1	1.732	1	0	0
(c)	0	0	1.414	1	-0.5	-0.707
(d)	2	1	1.732	1	0.9	1.5588

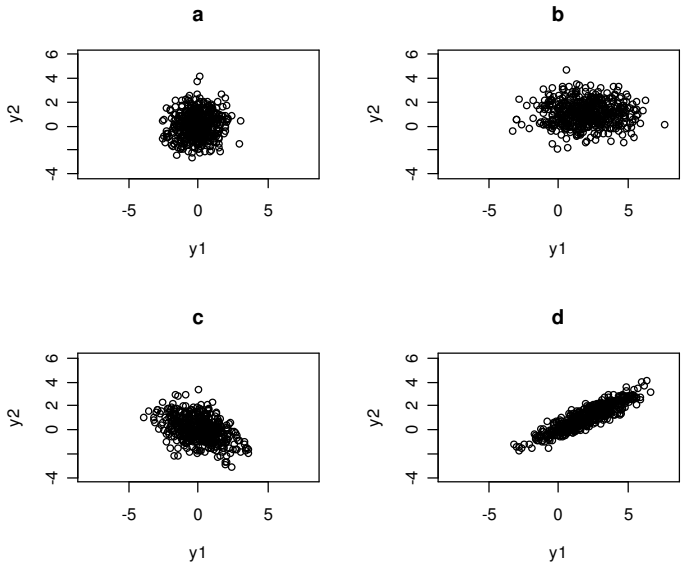


Figure 16.1: Simulated Bivariate Normal Data

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

The effects of changes in the parameters are obvious. The bivariate means determine the location of the center of the distribution. The standard deviations reflect the scale in the directions of the variables. For example in plot (b), the standard deviation of Y_1 is 1.732 times larger than that of Y_2 which is reflected in the larger range of Y_1. Finally the correlation describes the strength of a linear association between the two variables. Correlations vary between –1 and +1, where these limits denote perfect negative and positive correlations. Plot (d) depicts a strong positive correlation of 0.9. Notice the last column for the parameter σ_{12} in the table. This is the covariance of Y_1 and Y_2, which is the expected product of the deviations from the means:

$$\sigma_{12} = \text{Covariance} = \text{Cov}(Y_1,Y_2) = \text{Expected Value}\,(Y_1 - \mu_1)(Y_2 - \mu_2)$$

The correlation coefficient ρ_{12} is simply a scale-free version of the covariance, namely:

$$\rho_{12} = \text{Correlation}(Y_1,Y_2) = \text{Expected Value}\left[\left(\frac{Y_1-\mu_1}{\sigma_1}\right)\left(\frac{Y_2-\mu_2}{\sigma_2}\right)\right]$$

$= \sigma_{12}/(\sigma_1\,\sigma_2)$, or, equivalently $\sigma_{12} = \rho_{12}\,\sigma_1\,\sigma_2$

It is customary in multivariate statistics to provide the variances and the pairwise covariances arranged in a matrix as follows:

$$\Sigma = \begin{pmatrix} \text{var}(Y_1) & \text{cov}(Y_1,Y_2) \\ \text{cov}(Y_1,Y_2) & \text{var}(Y_2) \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}$$

This is called the variance-covariance or the dispersion matrix and is equivalent to providing the individual standard deviations and the correlations. Note that this matrix is symmetric: the off-diagonal elements above the diagonal correspond to those below the diagonal.

The R package mvtnorm provides commands for the multivariate normal distribution, rmvnorm for random number generation, dmvnorm for density calculations, and pmvnorm for cumulative probabilities. These commands require as inputs a vector of the means of the variables, and the variance-covariance matrix. For example, plot(d) in Figure 16.1 was generated using the following commands:

```
> library(mvtnorm)
> mean4 <- c(2,1)
> Sigma4 <- matrix(c(3,.9*(3^.5),.9*(3^.5),1),ncol=2)
> mat4 <- rmvnorm(500,mean4,Sigma4)
> plot(mat4,xlim=c(-8,8),ylim=c(-4,6),xlab="y1",ylab="y2")
```


Permission granted to reproduce for nonprofit, educational use.

The multivariate normal distribution serves as a baseline for most multivariate statistical analyses. Although clearly not all continuous data have a multivariate normal distribution, it is very difficult to work with other multivariate continuous distributions, one exception being the Dirichlet distribution that is used for modeling continuous fractions (see Chapter 8). The multivariate normal distribution is also preserved under linear data transformation. Geometrically speaking such transformations include translations, rotations, stretching and contracting. Virtually all of the classical dimension reduction techniques and classification methods are based on linear transformations, and therefore the multivariate normal distribution arises as a natural candidate for probabilistic modeling.

Another nice feature of the multivariate normal distribution is that any marginal distributions are again normal. To illustrate this point we randomly generate 4 variables Y_1, Y_2, Y_3, Y_4 that jointly have a multivariate normal distribution with respective means (2,4,6,8) and with variance covariance matrix

$$\Sigma = \begin{pmatrix} 1 & 1 & -.4 & -0.9 \\ 1 & 4 & 1.6 & -1.2 \\ -.4 & 1.6 & 4 & 1.8 \\ -0.9 & -1.2 & 1.8 & 9 \end{pmatrix}$$

We first check that the one-dimensional (univariate) marginal distributions are normal by plotting individual histograms which are displayed in Figure 16.2.

```
> Sigma <- matrix(c(1,1,-.4,-.9,1,4,1.6,-1.2,-.4,1.6,4,
+  1.8,-.9,-1.2,1.8,9),ncol=4)
> Y <- rmvnorm(500,c(2,4,6,8),Sigma)
> par(mfrow=c(1,4))
> for (i in 1:4) hist(Y[,i], main="", nclass=15,
  + xlab = paste("Y",i, sep=""))
```

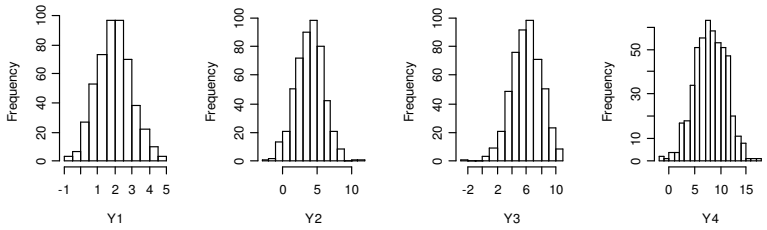


Figure 16.2. One-dimensional marginal distributions: Histograms

Note and check that the pairwise correlation coefficients are $\rho_{12} = 0.5$, $\rho_{13} = -0.2$, $\rho_{14} = -0.3$, $\rho_{23} = 0.4$, $\rho_{24} = -0.2$, $\rho_{34} = 0.3$. We examine the two-

Permission granted to reproduce for nonprofit, educational use.

dimensional marginal distributions by producing scatterplots of pairs of variables. For example, scatterplots of the pairs (1,2), (1,4), and (2,4) are shown in Figure 16.3.

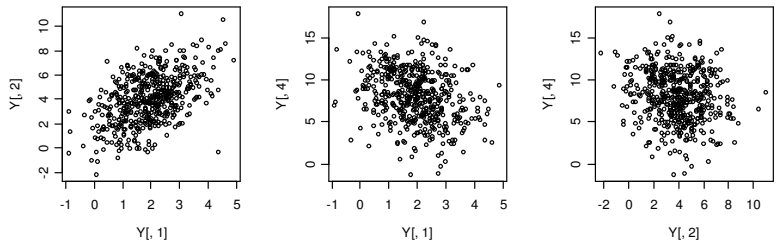


Figure 16.3. Two-dimensional marginal distributions: Scatterplots.

These are typical shapes for bivariate normal distributions as we have seen in Figure 16.1.

We now illustrate the property that normal distributions are preserved under linear transformation. A general linear transformation of the above four variables is defined by any arbitrary coefficients c, b_1, b_2, b_3, b_4 which then define a new random variable U:

$$U = c + b_1Y_1 + b_2Y_2 + b_3Y_3 + b_4Y_4$$

We consider the following two transformations:

$$U_1 = 3 + 2Y_1 - 2Y_2 + 4Y_3 - 4Y_4$$

$$U_2 = -6 - Y_1 - Y_2 + 6Y_3 + 2Y_4$$

By examining the histograms and the scatterplots (Figure 16.4) we verify that the U_1 and U_2 each have marginally a normal distribution, and both jointly follow a bivariate normal distribution.

```
> # Generate linear combinations
> U1 <- 3 + 2*Y[,1] -2*Y[,2] + 4*Y[,3] - 4*Y[,4]
> U2 <- -6 - Y[,1] - Y[,2] + 6*Y[,3] + 2*Y[,4]
> # Create graphs
> par(mfrow=c(1,3))
> hist(U1,nclass=12);hist(U2,nclass=12)
> plot(U1,U2)
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

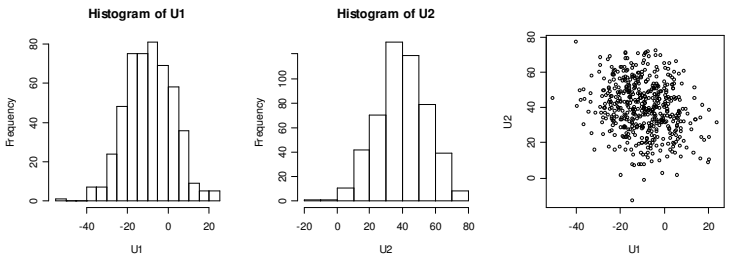


Figure 16.4: Distributions of Linear Combinations of Normal Variates

So far we have only examined simulated data, which by design are very well behaved. When being faced with actual continuous multivariate data in scientific applications, we will need to examine, explore and summarize the data before proceeding to use more sophisticated statistical techniques. Whenever possible we will judge as to whether the data follow a multivariate normal distribution. Note that most of the techniques in this text are not very sensitive to mild departures of the data from normality. However they can be influenced by individual extreme data values, so-called outliers, especially if the dataset is small to moderately large. If the distributions of some or all variables are consistently skewed a nonlinear transformation of the data such as the logarithm, or the square root transformation, often does wonders in creating transformed data that fairly closely follow a multivariate normal distribution.

Multivariate Sample Statistics

A routine first step in multivariate data analysis involves calculating sample statistics for the purpose of summarizing the data. The R-command "summary" gives the mean, median and sample quantiles of the variables. It is best to arrange the data into a data frame, instead of into a matrix. In a data frame each column can be named with the "names" command, and is treated as a separate variable.

```
> Y <- as.data.frame(Y)
> # Provide summaries for each variable
> summary(Y)
       Y1                 Y2                 Y3                 Y4
 Min.   :-0.8827   Min.   :-2.173   Min.   :-2.062   Min.   :-1.266
 1st Qu.: 1.2324   1st Qu.: 2.653   1st Qu.: 4.534   1st Qu.: 5.918
 Median : 1.9557   Median : 4.008   Median : 5.989   Median : 8.052
 Mean   : 1.9506   Mean   : 3.947   Mean   : 5.943   Mean   : 8.020
 3rd Qu.: 2.5636   3rd Qu.: 5.315   3rd Qu.: 7.311   3rd Qu.:10.316
 Max.   : 4.9096   Max.   :11.023   Max.   :10.927   Max.   :17.822
```

We note how close the medians and the means are to each other, indicating that the data are symmetric. In the following we obtain the standard deviations, the

Permission granted to reproduce for nonprofit, educational use.

sample variances and covariances arranged in a variance-covariance matrix, and the sample (Pearson) correlation coefficients arranged in a correlation matrix.

```
> # In the following only print 4 digits
> options(digits=4)
> # Get standard deviations
> sd(Y)

    Y1     Y2     Y3     Y4
0.9997 2.0017 2.0116 3.0502

> # Print the sample variance - covariance matrix
> var(Y)
        Y1      Y2      Y3      Y4
Y1  0.9993  0.9756 -0.4111 -0.8692
Y2  0.9756  4.0069  1.6587 -1.0377
Y3 -0.4111  1.6587  4.0466  1.7394
Y4 -0.8692 -1.0377  1.7394  9.3039

> # Print the correlation matrix
> cor(Y)

        Y1      Y2      Y3      Y4
Y1  1.0000  0.4875 -0.2044 -0.2851
Y2  0.4875  1.0000  0.4119 -0.1700
Y3 -0.2044  0.4119  1.0000  0.2835
Y4 -0.2851 -0.1700  0.2835  1.0000
>
```

Note how close the sample variance – covariance matrix is to the corresponding matrix Σ of the theoretical model from which the data was simulated. We use the symbol **S** (bold faced S) to denote the sample variance covariance matrix, which is customary in multivariate statistics. Remember again that the diagonal elements of **S** are the sample variances of the individual variables, and the off-diagonal elements are the covariances between pairs of the variables.

Displaying Multivariate Data

The best way to familiarize oneself with what is in a data set is through examining graphs. This is particularly important for multivariate data sets. While high dimensionality (many variables) prevents us from clearly seeing all features in lower dimensional plots, graphing is still a powerful method particularly for identifying outliers, relationships between variables, and for suggesting transformations to achieve distributions that are closer to a normal.

Let's use as an example the famous (Fisher's or Anderson's) iris data set that gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of the 3 species of iris: Iris setosa, Iris versicolor, and Iris virginica. (see Anderson (1935). The iris data is included in the R base package as a dataframe. We will analyze the species "versicolor".

Permission granted to reproduce for nonprofit, educational use.

```
> 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]) }
```

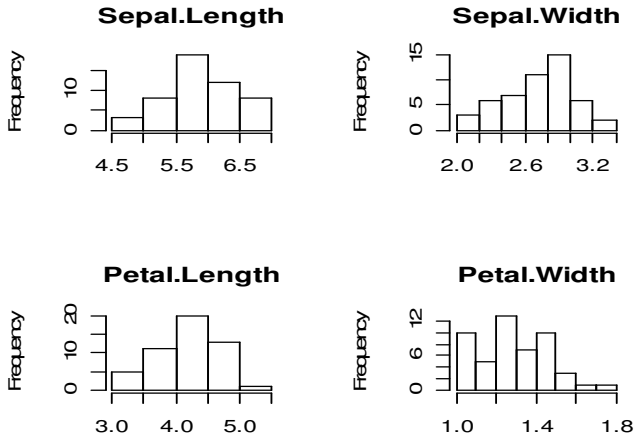


Figure 16.5: Histograms of Iris Versicolor Variables

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

Permission granted to reproduce for nonprofit, educational use.

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)
```

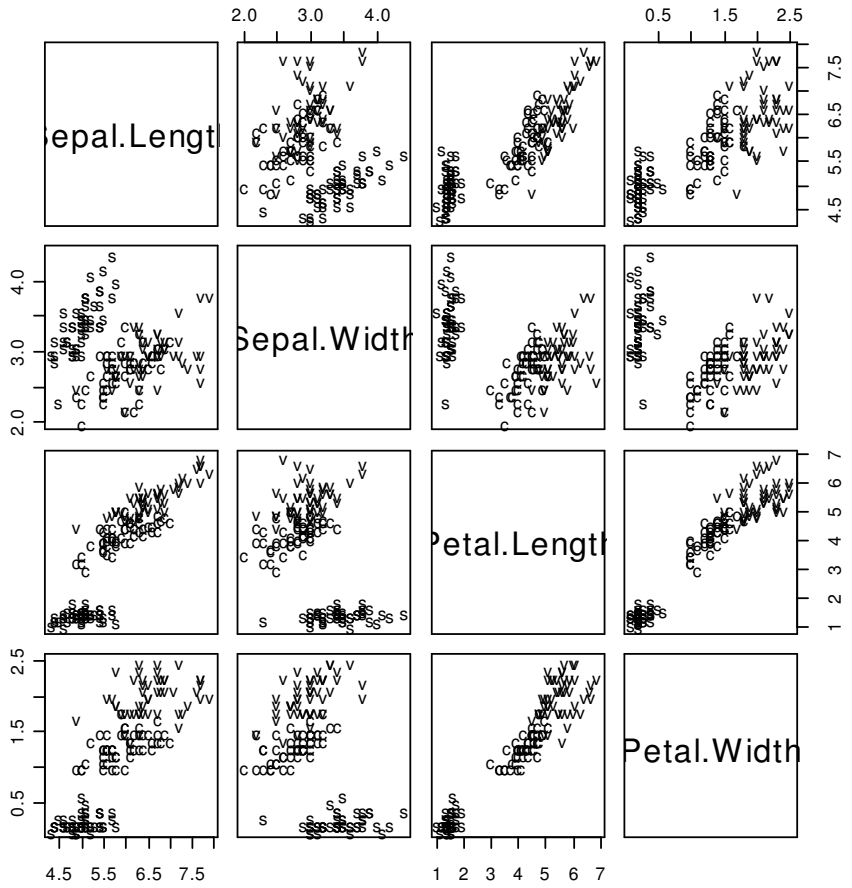


Figure 16.6: Scatterplot Matrix of Iris Versicolor Data

```
> # 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
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
Petal.Width     0.5464611   0.6639987   0.7866681   1.0000000
```

We note that the pairs Sepal.Length and Petal. Length, and Petal.Length and Petal Width are most strongly correlated with respective correlations of 0.7540 and 0.7867.

Within the last 10 years many statistical and graphical software packages have provided 3-dimensional data plots as point clouds where points are colored or shaded in a clever way to make the 3-dimensional features more visible to the human eye. Software may also include interactive graphics that allow the rotation of the point cloud which further enhances the subtle features in the data. The R package "scatterplot3d" provides a single command of the same name that creates a 3-d point cloud. For example plotting the four 3-d point clouds of the iris versicolor data results in the following (Figure 16.7)

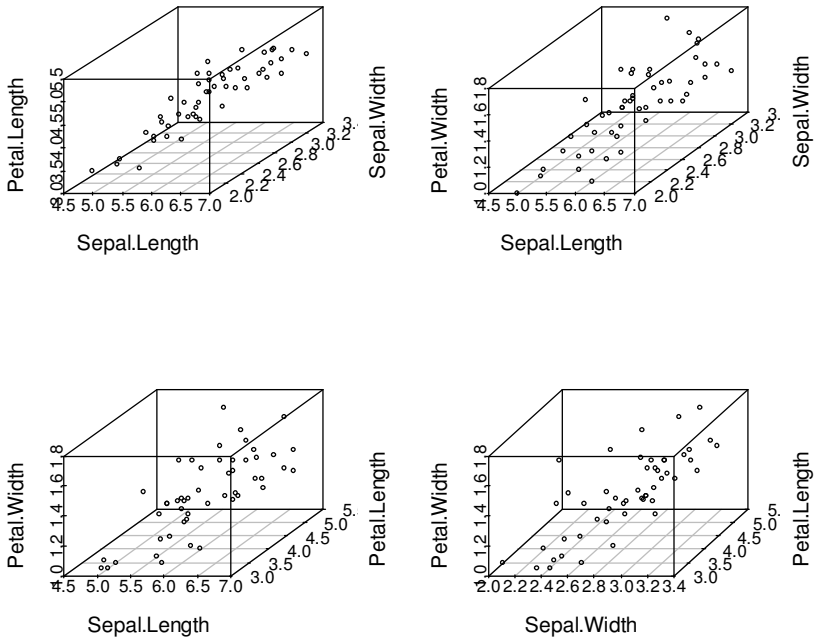


Figure 16.7: 3-d Point Clouds of Iris Versicolor Data

We notice a few subtle patterns in these point cloud graphs. In particular, in graph 1 (top left) there appear to be 4 points with low Sepal values slightly separated from the rest of the data. Also, in graph 2 (top right) there is a hint of a hook-like pattern for large values of Sepal Width and Petal Width. Again these patterns are subtle and would only be considered mild departures from multivariate normality. In fact for multivariate normal data, the surfaces of constant point density are ellipsoids in 3-d space that are centered at the mean.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Ellipsoids in 3-d space look like American footballs that have been stretched or compressed. In the special case of uncorrelated normal data, the football becomes aligned with the axes, and in the special case of uncorrelated data and equal variances among the variables, the football becomes a sphere.

Outliers

Outliers are extreme observations that are either the product of a data reading or measurement error, or they a caused by accidentally sampling from a different population than the rest of the data. For one and two-variable data it is easy to identify outliers in a scatterplot. For 3-variable data, rotating a 3-d point cloud helps in identifying outliers. For multivariate data with more than 3 variables, there is the chance that lower-dimensional projections do not reveal all outliers. The following gives an example of a two-variable data set with one outlier that is masked when the data is projected onto one-dimensional subspaces. The one-dimensional projections are represented as rugplots along the x and y axes.

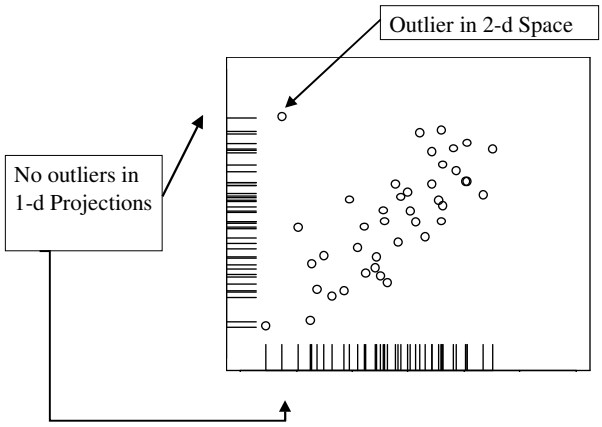


Figure 16.8: Masking of Outlier in Projections

One very simple classical method is to obtain for each data point $\mathbf{x}_i$, the unit-free measure of how many standard deviations it lies away from the center of the data, the multivariate mean. Remember that in univariate statistics this number, when using sample data, is simply the t-score:

$$t_i = \frac{x_i - \overline{x}}{s_X},$$

which for large data is approximates the z-score:

Permission granted to reproduce for nonprofit, educational use.

$$z_i = \frac{x_i - \mu_X}{\sigma_X}$$

Typically we are only interested in the absolute value (positive number) of the t-score, in which case we can write

$$u_i = \sqrt{t_i^2} = \sqrt{\left(\frac{x_i - \bar{x}}{s_X}\right)^2} = \sqrt{(x_i - \bar{x})\frac{1}{s_X^2}(x_i - \bar{x})}$$

For normally distributed data, t_i^2 has for large data approximately a chi-square distribution with 1 degree of freedom.

In multivariate data we will need to consider information from all the variances and covariances in order to obtain the measure that is equivalent to the t-score. It turns out that the measure, d_i^2 , which is a generalized squared distance, is:

$$d_i^2 = (\mathbf{x_i} - \bar{\mathbf{x}})' \mathbf{S}^{-1} (\mathbf{x_i} - \bar{\mathbf{x}})$$

This is a vector-matrix expression, and $\mathbf{S}^{-1}$ denotes the inverse matrix of the sample variance-covariance matrix **S**, and (..)**'** denotes the transpose of a vector or matrix. Any p by p matrix, such as **S** defines the coefficients of a system of p linear equations which is a transformation from one p-vector to another. The inverse matrix simply consists of the coefficients of the inverse transformation, and can be calculated by solving the linear system. For any p>2, such systems invariably require computer software for quick calculations. Notice the correspondence of d_i^2 with the right hand side of the equation for u_i. The d_i^2 are also called Mahalanobis distances, after the Indian statistician Mahalanobis. A result from multivariate statistics that generalizes the chi-square result above states that Mahalanobis distances approximately follow a chi square distribution with p degrees of freedom where p is the number of variables. R provides the Mahalanobis distances with the `mahalanobis` command for the rows of a data matrix. It requires the specification of the mean vector and of the variance covariance matrix: We plot the square roots of the Mahalanobis distances representing the distances in standard deviation units from the center for the Iris versicolor data in Figure 16.9.

```
> dist2 <- mahalanobis(iris.versicolor,
  + mean(iris.versicolor),var(iris.versicolor))
> plot(dist2^.5)
```


Permission granted to reproduce for nonprofit, educational use.

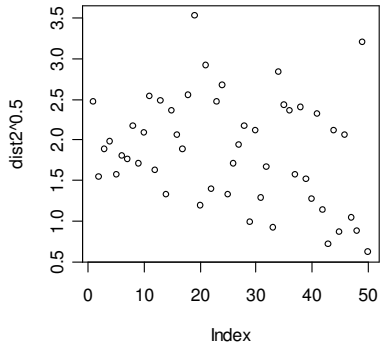


Figure 16.9 Multivariate distances of Iris Versicolor Data

We recognize 2 observations that are more than 3 units away from the center. Note that the square root of the 95th percentile of the chi squared distribution with 4 degrees of freedom equals 3.0802. We would use this number as a rough cut-off to flag extreme observations that give an indication of departure from normality. Notice however that none of the 2 flagged observations have an extremely large generalized distance, hence these are very moderate outliers, if we want to call them outliers at all.

Principal Components

In order to effectively analyze multivariate data the scientist is often forced to resort to what is known as dimension reduction techniques, the first we will introduce is known as principal component analysis. The primary goal of the method is to describe the variability in a multivariate data of p correlated variables by a smaller set of derived variables that are uncorrelated. The derived variables, called principal components, are linear combinations of the original variables, and are usually listed in the order of their respective importance. In other words, the first principal component (PC1) explains the largest amount of variation, the second principal component (PC2) is uncorrelated to PC1, and explains the second largest amount of variation, and so on. The aim is to discard subsequent principal components after a large percentage of the variation (90% – 99 %) has been explained by the first k principal components, where k < p.

Most statistical software provides methods for principal component analysis. In R we use the command `princomp` followed by the `summary` command to obtain standard PC analysis output given below. We apply this to the iris versicolor data. The procedure resides in the package `mva`.

```
> # Perform a Prinicpal Component Analysis
> library(mva)
> pc.iris <- princomp(iris.versicolor)
> summary(pc.iris)

Importance of components:
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
                          Comp.1    Comp.2     Comp.3     Comp.4
Standard deviation     0.6914597 0.2663389 0.23169066 0.09795181
Proportion of Variance 0.7808176 0.1158471 0.08766635 0.01566898
Cumulative Proportion  0.7808176 0.8966647 0.98433102 1.00000000
```

The Screeplot given in Figure 16.10 visualizes the amount of the variance that is explained by the four principal components. A dimension reducing transformation of the data into the first two PC's would capture 89.7% of the variability, while using the first three PC's would capture 98.4% of the variability.

```
> # Create screeplot of a princomp object
> screeplot(pc.iris)
```

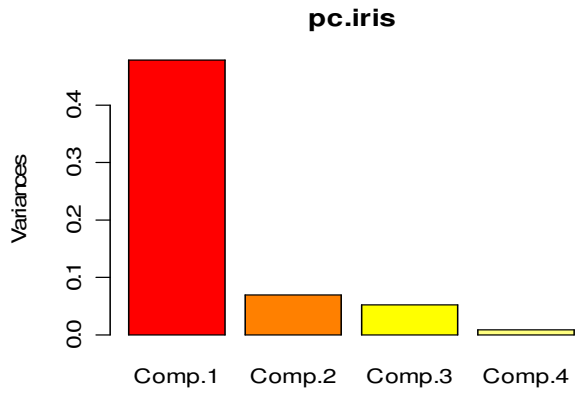


Figure 16.10: Screeplot of PC Analysis for Iris Versicolor

We note that all four original variables are size measurements (length and width). The essence of what principal components has done is to capture most of the variability of the data in one component variable (PC1) which can be used to reduce the dimensionality of the data from four original variables to one new component variable. PC1 can be interpreted as a derived variable of size. Principal components arise naturally in several other statistical procedures, notably in classification, which is the subject of the next section.

Classification and Discriminant Analysis

The goal of classification is to develop a predictive model that is capable of assigning membership to a class based on several measured characteristics. A standard example arises in biology: Several size measurements are obtained from samples of several species and the species of each sample is determined as well. It is assumed that taking size measurements is much easier than determining the exact species. In order to avoid the difficult task of species determination, a classification rule, capable of determining species from the size measurements alone, is developed.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

Classification for Two Populations

The easiest classification methods are those related to two populations, called binary classification. As the name suggests the goal is to predict membership in any one of two classes. The two classes are often coded as 0 and 1, but could also be thought of success and failure, or Yes and No.

Reaven and Miller (1979) examined the relationship between chemical subclinical and overt nonketotic diabetes in 145 non-obese adult subjects. The three primary variables used in the analysis are glucose intolerance, insulin response to oral glucose and insulin resistance. In addition, the relative weight and fasting plasma glucose were measured for each individual in the study. This data is available from the StatLib website. http://lib.stat.cmu.edu/datasets/Andrews/, Table 36.1. The data and a summary of related research is also described in the text *Data,* by Andrews and Herzberg (1985).

We attempt to derive a classification into the two categories normal (coded as 3 in the data) and subclinical diabetes (coded as 2) using the two variables glucose intolerance, and insulin response. We retrieve and plot the data using the following set of commands.

```
> glucose <- read.table('blood.glucose.data.txt')
> # Only use the 6 relevant variables. Last variable is class variable
> glucose <- glucose[,5:10]
> names(glucose) <- c('gluc.intol','i.respons','i.resist','rel.wt',
+'plas.gluc','diagn')
> # Only use normal (diagn=3) and subclinical diab (diagn=2)
> gluc1 <- glucose[glucose$diagn != 1,]
> attach(gluc1)
> # Plot the data
> plot(gluc.intol,i.respons,type='n')
> text(gluc.intol,i.respons,labels=diagn,cex=0.8)
> # Plot means of each group using point-character 15: filled square
> mean2 <- mean(gluc1[diagn==2,1:2])
> mean3 <- mean(gluc1[diagn==3,1:2])
> points(rbind(mean2,mean3),pch=15)
```

From the scatterplot of the data in Figure 16.11, we see that the responses of the two groups are quite interspersed. Diabetics overall have higher insulin response and glucose intolerance, as expected. The means of each group are indicated as filled squares.

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

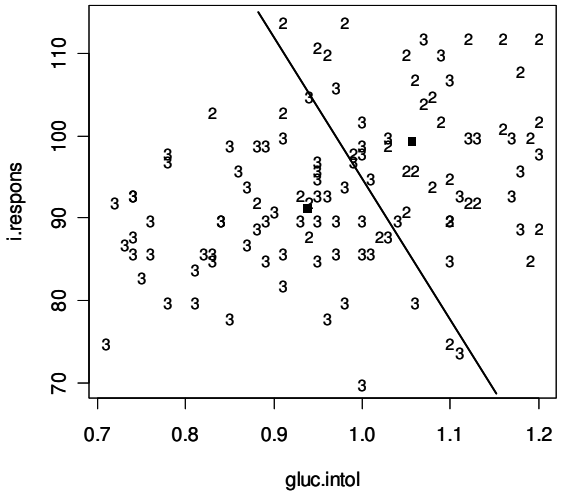


Figure 16.11: Glucose Intolerance and Insulin Response of Healthy (3) and Subclinically Diabetic (2) Individuals

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-word 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).

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

$$ECM = C(A|B)P(A|B)p_B + C(B|A)P(B|A)p_A ,$$

where C(A|B) is the cost of misclassifying X into A, when in fact it is in B, and similarly for C(B|A). Most often we consider a "0-1" cost function which means that there is a cost of 1 when misclassification occurs. With a "0-1' cost function it can easily be shown that the optimal classification rule is the one that maximizes the posterior probability.

$$p(A \mid X \text{ in } A) = \frac{p(X \text{ in } A \mid A)p_A}{p(X \text{ in } A \mid A)p_A + p(X \text{ in } A \mid B)p_B}$$

$$p(B \mid X \text{ in } A) = \frac{p(X \text{ in } A \mid B)p_B}{p(X \text{ in } A \mid A)p_A + p(X \text{ in } A \mid B)p_B}$$

Note that for binary classification, these two probabilities add to one. Deriving a minimum ECM rule can be difficult in practice. In particular expressions like p(X in A|A) may involve complicated multivariate density functions.

Linear discriminant analysis (LDA), was developed by R.A. Fisher in 1936 when he analyzed the iris data in the context of genetics. Fisher's LDA is an optimal classification rule if the data in each group follow a multivariate normal distribution that have different mean vectors but with the same variance-covariance matrix. That is a lot of assumption but is analogous to the assumptions made in ANOVA (see Chapter 15).

The package "MASS" contributed to the R project in conjunction with the text "Modern Applied Statistics with S-Plus" (Venables and Ripley, 1997) provides command `lda` linear discriminant analysis. The commands require as inputs a multivariate data matrix or dataframe, and a factor that denotes class membership. Let's first apply `lda`:

```
> # Perform Linear Discriminant Analysis
> library(MASS)
> gluc.lda <- lda(gluc1[,1:2],gluc1$diagn)
> gluc.lda

Prior probabilities of groups:
     2      3
0.3214 0.6786
```

The prior probabilities are assumed according to the fraction of data that fall in each class, unless otherwise specified.

```
Group means:
  gluc.intol i.respons
2     1.0558     99.31
3     0.9372     91.18

Coefficients of linear discriminants:
                LD1
gluc.intol -5.57342
```


Permission granted to reproduce for nonprofit, educational use.

```
i.respons  -0.07247
```

The coefficients of `lda` determine the linear transformation. One must be careful in interpreting these, because in R, as in other packages, they have been conveniently rescaled. An easier interpretation is the following: The separation line crosses the line connecting the group means at the halfpoint, and has a slope that is minus the ratio of the lda coefficients. In our example the slope is –5.573/0.072 = -76.9. (see Figure 16.11).

Let's examine how well this classification works. The command `predict(lda)` provides a printable list with the following objects:

```
> # 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 3 3 2 2
 [29] 3 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3
 [57] 3 3 3 2 2 3 2 3 2 3 2 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3
 [85] 3 2 2 2 2 2 2 2 3 3 2 2 2 3 3 2 2 3 2 3 3 3 3 3 2 3 2 3
Levels: 2 3
> # posterior probabilities of membership
> gluc.pred$posterior
          2      3
1   0.03147 0.9685
2   0.28652 0.7135
3   0.43596 0.5640
4……..

> # lda scores induced by the transformation
> gluc.pred$x

           LD1
  1    1.92131
  2   -0.09097
  3   -0.61500
..4…….
```

Inspecting `gluc.pred$class` we notice that the first 3 observations are classified as 3 because p(3|X) > 0.5 in all three. We could now count the misclassifications by comparing the actual diagnosis, the `diagn` variable with the predicted class. The `table` command in R provides for a convenient summary:

```
> table(gluc1$diagn,gluc.pred$class)

     2  3
  2 17 19
  3 10 66
```

Thus the estimated misclassification errors are as follows

Permission granted to reproduce for nonprofit, educational use.

P(X is “3”|”2”) = 19/36 = 0.528, and P(X is in “2”| “3”) = 10 / 76 = 0.132. We can obtain the classification probabilities by dividing the table by its row sums as follows:

```
> tab <- table(gluc1$diagn,gluc.pred$class)
> tab/rowSums(tab)

         2      3
  2 0.4722 0.5278
  3 0.1316 0.8684
```

These misclassification error probabilities are quite high.

Classification for More Than Two Populations

The discussion of binary classification directly generalizes to more than two populations without much difficulty. More challenging is the case where there are much more than two response variables. We now attempt to classify the full diabetes data using all five variables and all three diagnoses (1: overt diabetic, 2: chemical diabetic, 3: normal). We may first examine the scatterplot matrix. Linear discriminant analysis attempts to transform the data into one or two new variables, so called canonical variates, so that the separation of diagnosis is strongest in the first, and second strongest in the second. Note the similarity to principal component analysis. The following gives the R calculations.

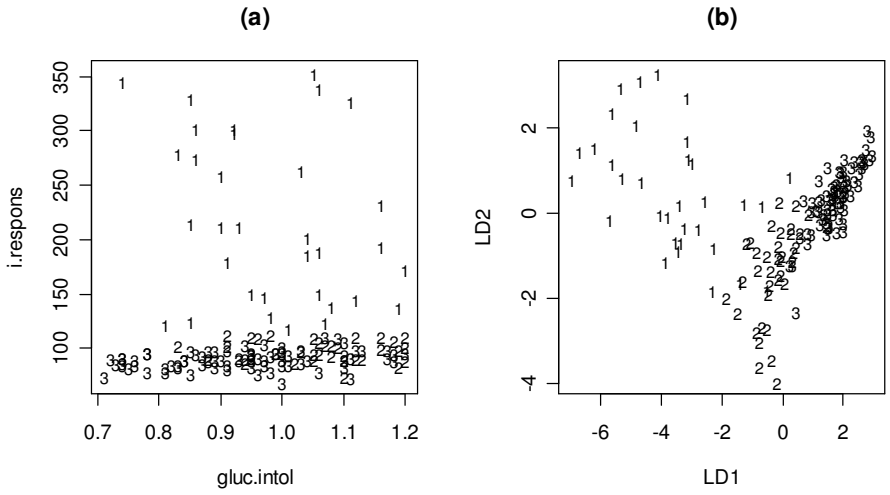
```
> # Perform LDA for three diagnoses; use all variables
> library(MASS)
> gluc.lda <- lda(glucose[,1:5],glucose$diagn)
> gluc.lda

Prior probabilities of groups:
     1      2      3
0.2276 0.2483 0.5241

Group means:
  gluc.intol i.respons i.resist rel.wt plas.gluc
1     0.9839    217.67   1043.8  106.0     318.9
2     1.0558     99.31    493.9  288.0     209.0
3     0.9372     91.18    350.0  172.6     114.0

Coefficients of linear discriminants:
                  LD1       LD2
gluc.intol -1.3624357 -3.784142
i.respons   0.0336488  0.036633
i.resist   -0.0125764 -0.007092
rel.wt      0.0001022 -0.006173
plas.gluc  -0.0042432  0.001134

Proportion of trace:
   LD1    LD2
0.8812 0.1188
```


Permission granted to reproduce for nonprofit, educational use.

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 into 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).



Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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.

Permission granted to reproduce for nonprofit, educational use.

We have deleted the samples of the types of cancers for which there were 2 or fewer samples, resulting in a sample size of 57. We selected a set of 35 genes that exhibited minimal expression of at least –0.55 for all cancers. To make the results more understandable we only consider the 6 most expressive genes, which corresponds to a minimal expression level over the 64 cancer samples of –0.42.

We thus work with 6 variables and attempt to partition into 8 different cancer types. The cancer types are BREAST, CNS, COLON, LEUKEMIA, MELANOMA, NSCLC, OVARIAN, RENAL. They are assigned in this order in the following output.

```
> # Classification Trees
> # Only use the 6 most expressed genes
> rowmin <- apply(MAdat,1,min)
> sort(rowmin,decr=T)[1:20]
> MAdat.red <- MAdat[(rowmin > -.475),]
> # Delete cancers with few cases
> delet <- c(21,24,30,35,36,49,51)
> MAdat.red <- MAdat.red[,-delet]
> Genedat <- data.frame(t(MAdat.red))
> # Names of Genes
> names(Genedat)

[1] "X2838" "X3234" "X3320" "X4831" "X5680" "X6596"

> cancers <- as.factor(names(MAdat.red))
> Genedat <- cbind.data.frame(Genedat,cancers)
```

The command `tree`, which requires a model formula, produces a tree object that can be plotted with the plot command and summarized with the summary command. We use an abbreviated model formula: `cancers ~.` , which means that cancer is the response variable, and all others are the predictor variables.

```
> # Classification Tree via Recursive Partitioning
> library(tree)
> MA.tr <- tree(cancers ~.,data=Genedat)
```

We can get an overall summary of the tree procedure:

```
 > summary(MA.tr)

Classification tree:
tree(formula = cancers ~ ., data = Genedat)
Variables actually used in tree construction:
[1] "X5680" "X4831" "X2838" "X3234"
Number of terminal nodes:  8
Residual mean deviance:  2.1 = 103 / 49
Misclassification error rate: 0.439 = 25 / 57
```

And we can plot the tree, which uses these commands. The graph is given in Figure 16.13.

```
> plot(MA.tr)
> # Add labels to the splits
> text(MA.tr)
```


Permission granted to reproduce for nonprofit, educational use.

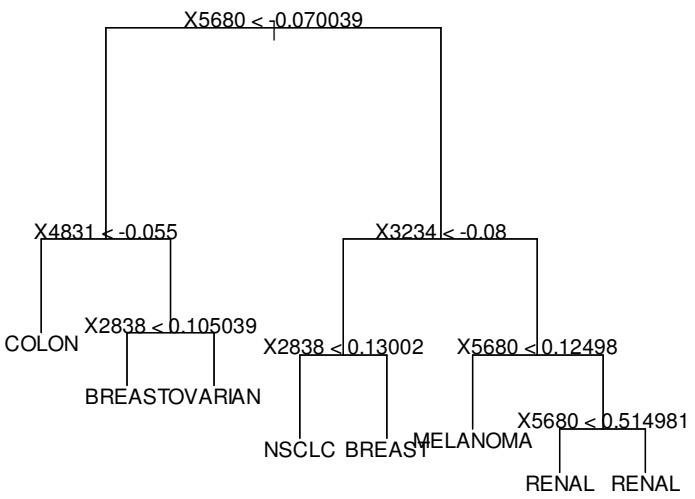


Figure 16.13 Classification Tree for Gene Expression Data

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 `rpart` 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.

Permission granted to reproduce for nonprofit, educational use.

A basic reality of clustering is the fact that for even a small number of items to be grouped, there are thousands of possible ways to cluster – that's why "cleaning house" can be so frustrating! In fact the number of possible clusters of k items is $2^k - 1$, a number that grows exponentially with k.

Number of possible clusters of k items

# items	2	4	6	8	10	12
#clusters	3	15	63	255	1023	4095
# items	14	16	18	20	22	24
#clusters	16383	65535	262143	1048575	4194303	16777215

Obviously we need computers to check through so many possible solutions. But even with today's high-speed computers, it is still a computational challenge to repeatedly check and sort billions of possible clusters. Clustering techniques have been developed as clever algorithms that make it possible for the computer to arrive at an optimal set of clusters without having to repeatedly check and count through all possible clusters. There are two broad categories of clustering techniques. In the first, called hierarchical clustering, the number of clusters is determined hierarchically either going from smallest to largest clusters (agglomerative) or from largest to smallest (divisive). In the second, which can be called non-hierarchical clustering, the number (k) of clusters is predetermined and one simply finds the best possible partition of the samples into k clusters.

Measures of Dissimilarity

A clustering of a data is considered optimal if samples within clusters are as similar as possible, and samples between clusters are as dissimilar as possible. We first need to define what we use as measure of similarity, or dissimilarity .Let's do a mini-microarray hypothetical experiment. We are looking at six genes and the expression of these genes under treatment and control conditions. Scores below are standardized expression scores.

Gene	Treatment	Control
G1	0.5	0.5

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

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.

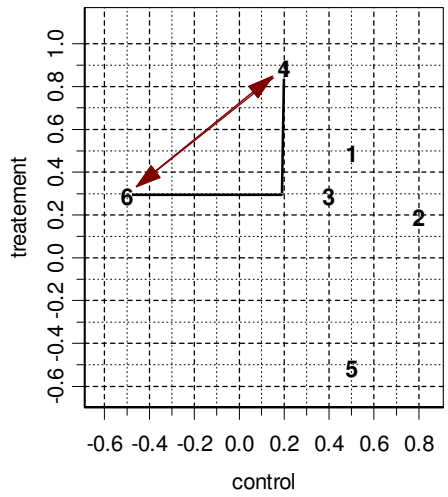


Figure 16.14 Toy Example of Gene Expression

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..

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

```
> library(mva)
> genes <- cbind(c(.5,.8,.4,.2,.5,-.5),c(.5,.2,.3,.9,-.5,.3))
> # Euclidean Distances
> dist(genes)
      1      2      3     4     5
2 0.4243
3 0.2236 0.4123
4 0.5000 0.9220 0.6325
5 1.0000 0.7616 0.8062 1.432
6 1.0198 1.3038 0.9000 0.922 1.281
```

Two other popular distance metrics used for clustering are the city block, or Manhattan distance (or L1 norm) and the max component distance.

In R, simply indicate the distance type in a subcommand inside `dist`. Any recognizable abbreviation is sufficient. The following shows the distances between the six genes using these two alternative metrics.

```
> # Manhattan Distances
> dist(genes,'manh')
    1   2   3   4   5
2 0.6
3 0.3 0.5
4 0.7 1.3 0.8
5 1.0 1.0 0.9 1.7
6 1.2 1.4 0.9 1.3 1.8

> # Max Component Distances
> dist(genes,'max')
    1   2   3   4 5
2 0.3
3 0.2 0.4
4 0.4 0.7 0.6
5 1.0 0.7 0.8 1.4
6 1.0 1.3 0.9 0.7 1
```

K-means Clustering

K-means clustering is probably the most widely used non-hierarchical clustering method in biological applications. The major advantage of this type of clustering comes from computational efficiency. Because K-means clustering does not require initial computation of a large distance matrix stored during the computer run, it is computationally efficient on a large dataset. As the name suggests, the aim is to optimally assign n data points to k clusters ($k < n$) where optimal means that each point belongs to the cluster whose mean it is closest to with respect to a chosen dissimilarity metric. The number of clusters (k) is a fixed number and needs to be selected prior to the cluster calculation.

Mathematically, K-means clustering works in a three steps algorithm:

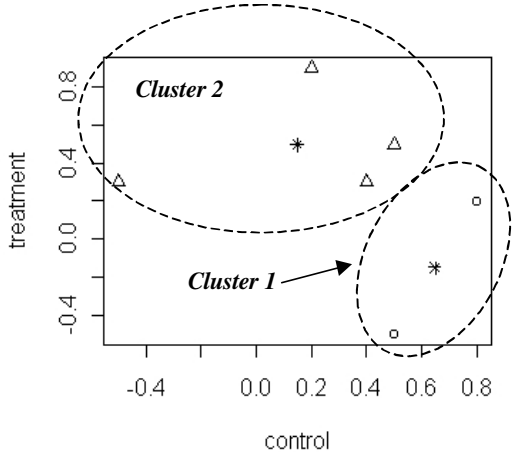
Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

- 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] 0.29 0.85

$size
[1] 2 4

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



Permission granted to reproduce for nonprofit, educational use.

*Figure 16.15. K means clustering of 6 Genes.(* denotes cluster means)*

We can see by eye from that all 6 genes are closest (in Euclidean distance) to their respective cluster means.

K-Medoid Clustering

Euclidean distances that are used for clustering as in the K-means procedure are very sensitive to extreme observations and outliers. Furthermore, the within - cluster means that are used as centroids are equally sensitive to outliers. Many so-called robust alternatives have been suggested, but they tend to be heavy on the computing requirement. The R package `cluster` is a suite of robust clustering methods for the most part based on the text by Kaufman and Rousseeuw (1990). Furthermore the graphics capabilities provided by `cluster` are quite impressive. This package is highly recommended as an add-on to the standard `mva` package.

In univariate statistics the sample median, or the middle observation, is often considered as a robust alternative to the sample mean. The median is insensitive, or robust, to outliers, no matter how extreme an outlier. The sample mean minimizes the sum of the squared distances (i.e. squared Euclidean distance) to the data points, while the median minimizes the sum of the absolute differences (L1 metric) to the data points. For clustering of p-dimensional data we consider so-called medoids as centers for the clusters. A medoid of a set of data points in p-space is the one data point for which the sum of the dissimilarities with all the other points is at a minimum. Medoids can be defined for any distance metric, but usually the Manhattan metric is chosen in order to reduce the influence of potential outliers.

The command `pam` in the package `cluster`, that is an acronym for "partition using medoids", performs k-medoid clustering. The input for the command can be either a data matrix or, instead, a distance matrix. Also the command provides the option (`stand = TRUE`)of using standardized data. Standardizing variables is highly recommended when the variables represent measurements that do not have a common natural scale. Data are standardized by subtracting each variable's mean value and by dividing by the variable's mean absolute deviation.

Hierarchical Clustering

In most large data in bioinformatics, such as in gene expression microarrays, there is no a-priori information as to what the number of clusters should be. For exploring structure in such data the biologist will perform a sequence of

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

clustering calculations from small to large number of clusters. It is much easier to interpret results if clusters of a finer partition are restricted to be subclusters of those of a coarser partition. Such a restriction results in what is called hierarchical clustering. Because of the hierarchy results can be displayed in a tree-like fashion, called a dendrogram.

There are two ways to do hierarchical clustering: (1) from small to large (so-called agglomerative), and (2) from large to small (so-called divisive). Agglomerative clustering *starts* with each data point being a one-point cluster. Clusters are then successively combined until ultimately all data reside in a single big cluster. Divisive clustering proceeds in the opposite way, starting with a single big cluster, and then optimally dividing clusters into subclusters, until ultimately each data point forms a separate one-point cluster. The question of course arises as to when to stop the procedure. The y axis of the dendrogram is the height, and usually measures the distance between the two clusters that are formed by the split. Therefore it is best to stop before branches in successive splits become crowded over short distances.

Agglomerative hierarchical clustering

The first thing in agglomerative clustering is to choose a dissimilarity measure, such as Euclidean or Manhattan. At any given step in the algorithm, all pairs of dissimilarities between clusters are examined, and the pair of clusters with smallest dissimilarity will be joined to form a single cluster. There are a variety of dissimilarities between clusters that we can choose from. These are commonly called the methods of linkage. The most common linkages are (1) average, (2) single, (3) complete, (4) Ward linakge. In the average linkage method the distance between two clusters is the average of the dissimilarities between the points in one cluster and the points in the other cluster. In the single linkage method, we use the smallest dissimilarity between a point in the first cluster and a point in the second cluster. In the complete linkage method, we use the largest dissimilarity between a point in the first cluster and a point in the second cluster. The Ward linkage method uses sums of squared deviations from the mean within clusters as a criterion. The following graph illustrates the first three different linkage methods

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

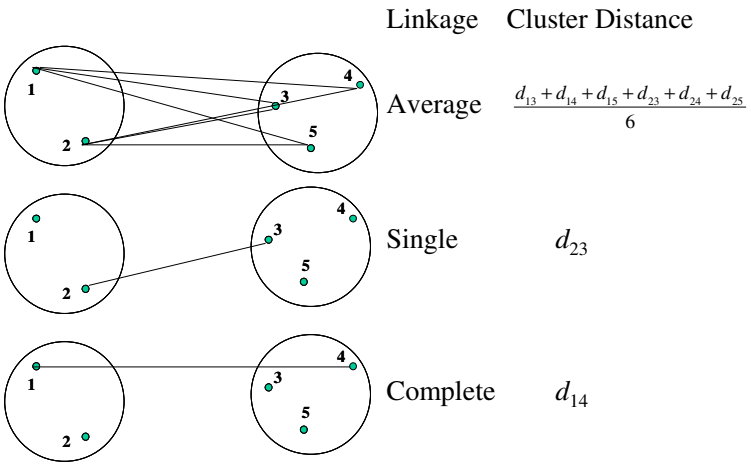


Figure 16.16: Linkage Methods for Hierarchical Clustering

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 a interactive stop 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")
```

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.

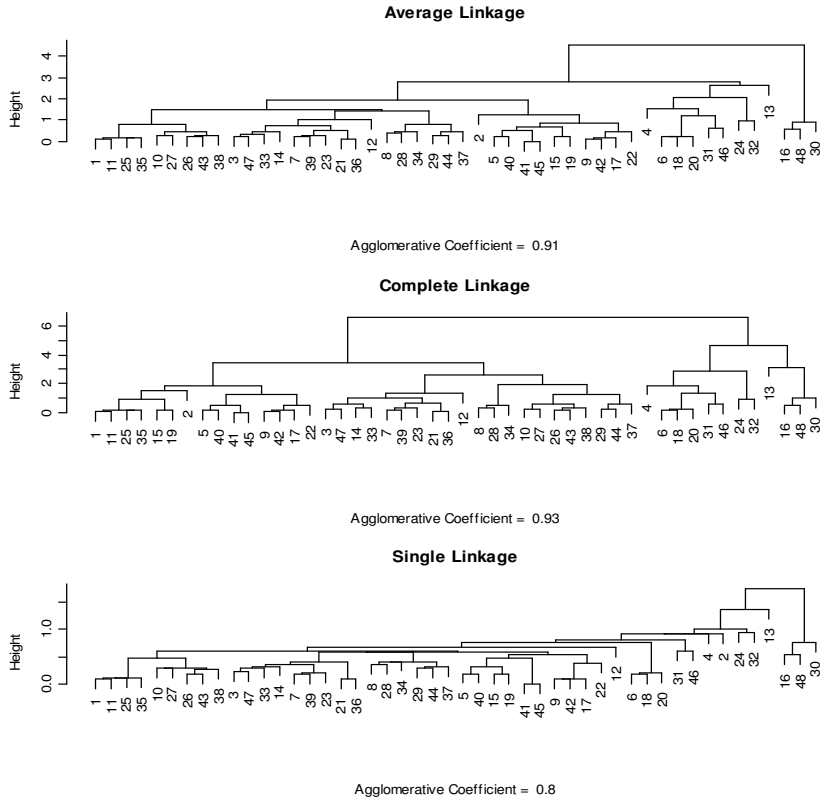


Figure 16.17. Agglomerative Clustering of Sleep Data using Three Different Linkage Methods

Divisive Hierarchical Clustering

Divisive clustering basically works in the opposite direction of agglomerative clustering. For a given dissimilarity metric the algorithm begins with a single cluster for the entire data set. In subsequent steps clusters are divided into sub-clusters. There are many different possibilities for dividing clusters, and hence many different algorithms exist. At any given stage, which cluster should be divided next? The R command `diana` of the library `cluster` chooses at each stage the cluster with the largest diameter, which is the largest dissimilarity between any two of its observations. For this cluster it then isolates the point that is farthest away from all the other points within the clusters. This point will initiate the new cluster split. Then more points will be aggregated to this new cluster, if they are closer in distance to this new cluster than to the old cluster, until no more such points are found. The command `diana` works just as the other stwo

Copyright May 2007, K Seefeld

Permission granted to reproduce for nonprofit, educational use.