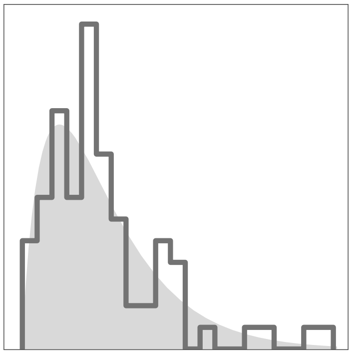

Appendix A

Introduction to Mathematica Commands

Mathematica (Wolfram Research, Inc.) is a system for doing mathematics on the computer and for reporting the results. *Mathematica* has numerical, graphical, and symbolic capabilities. Its basic features include arbitrary precision arithmetic; differential and integral calculus (routines for both symbolic and numerical evaluation); infinite and finite series, limits and products; expansion and factoring of algebraic expressions; linear algebra; solving systems of equations; and two- and three-dimensional graphics. *Mathematica* packages and custom tools include procedures for probability and statistics.

This chapter briefly describes the *Mathematica* commands used in the laboratory problems. The first section covers commands provided by the system, and commands loaded from *Mathematica* packages. The second section covers the customized tools provided with this book. An index to the commands appears at the end of the chapter.

A.1 Standard commands

This section introduces the standard *Mathematica* commands used in the laboratory problems. Standard commands include commands provided by the system and commands loaded from add-on packages when the `StatTools` packages are initialized. Note that commands from the

`Graphics‘FilledPlot‘`,
`Statistics‘DataManipulation‘`,
`Statistics‘DescriptiveStatistics‘`,
`Statistics‘DiscreteDistributions‘`, and
`Statistics‘ContinuousDistributions‘`

packages are loaded when `StatTools‘Group1‘` and `StatTools‘Group2‘` are initialized, commands from

`Statistics‘ConfidenceIntervals‘` and

Statistics‘HypothesisTests‘

are loaded when StatTools‘Group3‘ is initialized, and commands from

Statistics‘LinearRegression‘

are loaded when StatTools‘Group4‘ is initialized.

Online help for each Symbol discussed in this section is available by evaluating ?Symbol. Additional help and examples are available in the Help Browser.

A.1.1 Built-in constants and functions

Frequently used constants and arithmetic functions, functions for sums and products, functions from calculus, functions used in counting problems, the numerical approximation function (N), boolean functions, logical operators and connectors, and a function for producing random integer and random real numbers are summarized below.

Constants and arithmetic functions

Symbol	Value	Symbol	Function
Pi	$\pi = 3.14159\ldots$	$+, -$	add, subtract
E	$e = 2.71828\ldots$	$*, /$	multiply, divide
I	$i = \sqrt{-1}$	$\wedge$	exponentiation
Infinity	∞		
True, False	true, false		

Arithmetic expressions are evaluated from left to right with the following hierarchy: exponentiation is done first, followed by multiplication and division, followed by addition and subtraction. Parentheses can be used to enclose subexpressions. For example, $3 + 8 * 4$ returns 35 and $(3 + 8) * 4$ returns 44.

In addition, a space can be used instead of the multiply symbol (although this is not recommended). For example, $3 * 4 * 5$ and 3 4 5 each return 60.

Sum, Product functions

The functions Sum and Product can be used to compute sums (respectively, products) of one or more terms. For example,

1. Sum[$x \wedge 2, \{x, 1, 10\}$] returns $385 = 1 + 4 + 9 + \cdots + 100$.

2. Product[$x, \{x, 4, 12\}$] returns $79833600 = 4 * 5 * 6 * \cdots * 12$.

In each case, the variable x is known as an *iterator*. More generally, Sum and Product can be used to compute multiple sums (respectively, products). For example,

Sum[$i + j, \{i, 1, 4\}, \{j, 1, i\}$] returns $50 = 2 + 3 + 4 + 4 + 5 + 6 + 5 + 6 + 7 + 8$.

Functions from calculus

Mathematica Name	Function
`Log[x]`	natural logarithm, $\ln(x)$ or $\log(x)$
`Log[10,x]`	common logarithm, $\log_{10}(x)$
`Exp[x]`	exponential function, e^x
`Power[x,y]` or `x^y`	power function, x^y
`Sqrt[x]`	square root function, $\sqrt{x}$
`Abs[x]`	absolute value function, $\lvert x\rvert$
`Sign[x]`	sign function ($+1$ when $x > 0$, -1 when $x < 0$, 0 when $x = 0$)
`Round[x]`	round x to the nearest integer
`Floor[x]`	return the greatest integer less than or equal to x
`Sin[x]`, `Cos[x]`, `Tan[x]`, ...	$\sin(x)$, $\cos(x)$, $\tan(x)$, ...
`ArcSin[x]`, `ArcCos[x]`, `ArcTan[x]`, ...	$\arcsin(x)$, $\arccos(x)$, $\arctan(x)$, ...

Notice in particular that *Mathematica* uses the general function name `Log` for both natural and common logarithms.

Functions for counting problems

Mathematica Name	Function
`Factorial[x]` or `x!`	factorial, $x! = x * (x-1) * \cdots * 1$ when x is a non-negative integer
`Binomial[n,r]`	binomial coefficient, $\binom{n}{r}$
`Multinomial[`r_1`,`r_2`,`...`,`r_k`]`	multinomial coefficient, $\binom{n}{r_1,r_2,\ldots,r_k}$ where $n = r_1 + r_2 + \cdots + r_k$

For example, `Binomial[10,3]` returns 120 (the total number of subsets of size 3 from a set of size 10), and `Multinomial[5,2,3]` returns 2520 (the total number of partitions of a set of size 10 into distinguishable subsets of sizes 5, 2, 3).

Random numbers

Mathematica uses the general function name `Random` to return random numbers in many different situations. For example,

1. `Random[]` returns a real number chosen uniformly from the interval $(0, 1)$.
2. `Random[Real,{0,500}]` returns a real number chosen uniformly from the interval $(0, 500)$.
3. `Random[Integer,{10,200}]` returns an integer chosen uniformly from the range of integers $10, 11, 12, \ldots, 200$.

In each case, an algorithm called a *pseudo-random* number generator is used to produce the output.

Numerical approximation

Mathematica returns exact answers whenever possible. The `N` function can be used to compute approximate (decimal) values instead. For example, `N[E]` returns 2.71828 and `N[40!]` returns $8.15915\ 10^{47}$.

Logical symbols

Logical operators and connectors are summarized below:

Symbol	Meaning	Symbol	Meaning
$<$	less than	$==$	equal to
$<=$	less than or equal to	$!=$	not equal to
$>$	greater than	&&	logical and
$>=$	greater than or equal to	$\vert\vert$	logical or

There are often several ways to ask the same question. For example, if a, x, and b are specific real numbers, then the expression `Not[`$a <= x <= b$`]` and the expression $(x < a)||(x > b)$ each return `True` when the number x is not in the interval $[a, b]$, and `False` otherwise.

Note that, in many systems, if you type "<=", the symbol "≤" appears on the screen. Similarly, if you type ">=", the symbol "≥" appears on the screen.

Boolean functions

Functions returning `True` or `False` are summarized below:

Mathematica Name	Returns `True` when	Returns `False` when
`Positive[x]`	x is positive	x is 0 or negative
`Negative[x]`	x is negative	x is 0 or positive
`EvenQ[x]`	x is an even integer	otherwise
`OddQ[x]`	x is an odd integer	otherwise
`IntegerQ[x]`	x is an integer	otherwise
`MemberQ[list,x]`	x is a member of `list`	otherwise

In addition, `Not[x]` returns `True` if the value of x is `False` and returns `False` if the value of x is `True`.

Case-sensitivity

Mathematica is a case-sensitive language. For example, *Mathematica* recognizes `Sum` as the symbol representing the sum function, but does not recognize `sum` as the sum function. Be sure to pay close attention to the pattern of capital and lower case letters used in symbol names. In particular, all *Mathematica* commands begin with a capital letter.

A.1.2 User-defined variables and functions

Naming conventions for user-defined variables and functions, immediate and delayed assignments to symbols, echoing results to the screen, patterns and function definitions, functions for clearing and removing symbols, and functions for conditional statements and building multi-step procedures are discussed below.

Naming conventions, assignment, echoing

As a general rule, you should use lower case letters when defining variables and functions. *Mathematica* distinguishes between immediate assignment (`=`) and delayed assignment (`:=`), as described below.

`symbol = expression`
evaluates `expression`, stores the result in `symbol`, and echoes the result to the screen.
 	
`symbol = expression;`
evaluates `expression`, stores the result in `symbol`, but does *not* echo the result to the screen.
 	
`symbol := expression`
stores the *un*evaluated `expression` in `symbol`.

For example,

```
length=8; width=5;
area=length*width
```

stores the length, width, and area of a rectangle, and returns the area.

Functions

Delayed assignment is commonly used when defining functions. The form `x_` (a symbol followed by an underscore character) is used to represent an independent variable or *pattern*. Then

`f[x_] := expression in x`
defines the function $f(x)$ using a delayed assignment.
 	
`f[x_,y_] := expression in x and y`
defines the function $f(x, y)$ using a delayed assignment.
 	
Etcetera

Clear, Remove functions

The `Clear` function is used to clear the values of one or more user-defined symbols. The `Remove` function is used to remove one or more user-defined symbols from the *Mathematica* environment. It is a good idea to use `Clear` and `Remove` before defining functions. For example,

```
Clear[x]; Remove[f];
f[x_] := x^2
```

defines the function $f(x) = x^2$ and

```
Clear[x,y]; Remove[f];
f[x_,y_] := x<y
```

defines the function $f(x, y)$ whose value is `True` when x and y are numbers with $x < y$, and whose value is `False` when x and y are numbers with $x \geq y$.

If, Which functions

The `If` and `Which` functions can be used to implement split-definition functions:

`If[condition,`e_1`,`e_2`]`
returns the value of expression e_1 if the value of `condition` is `True`, and returns the value of expression e_2 if the value of `condition` is `False`.
 	
`Which[`c_1`,`e_1`,`c_2`,`e_2`,`$\ldots$`,`c_r`,`e_r`]`
returns the value of the first expression for which the condition c_i has value `True`.

For example,

```
Clear[x]; Remove[f];
f[x_] := If[x<0, 1+x, 1-x]
```

defines the function $f(x)$ whose value is $1 + x$ if x is negative ($x < 0$) and $1 - x$ if x is non-negative ($x \geq 0$), and

```
Clear[x]; Remove[f];
f[x_] := Which[-1<=x<0, 1+x, x<=1, 1-x, True, 0]
```

defines the function $f(x)$ whose value is $1 + x$ when x is in the half-closed interval $[-1, 0)$, $1 - x$ when x is in the closed interval $[0, 1]$, and 0 otherwise.

Module command

The `Module` command is used to define functions whose computations require several steps. The form of the command is

```
Module[{local variables},expression]
```

where the `local variables` (separated by commas) are used in the multi-step `expression`, the steps in `expression` are separated by semi-colons, and the value of the last step is returned. For example,

```
Clear[x,y,z]; Remove[f];
f[x_,y_,z_] := Module[{s},
  s = (x+y+z)/2;
  Sqrt[s*(s-x)*(s-y)*(s-z)]]
```

defines the function $f(x, y, z)$ whose value is $\sqrt{s*(s-x)*(s-y)*(s-z)}$ where $s = (x+y+z)/2$. ($f(x, y, z)$ is the area of a triangle with sides x, y, and z computed using its semi-perimeter, s.)

A.1.3 Mathematica lists

The basic *Mathematica* data structure is the list (defined below). Functions for computing the length of a list, for sorting a list, for building lists, and for extracting elements and sublists are summarized below.

Definition of list

A *list* is a sequence of elements separated by commas and enclosed in curly braces. For example, $\{1.3, -12, 8.3, 15, 7\}$ is a list of numbers,

$$\{\{5, 3.3\}, \{12.1, 2\}, \{-8.3, 4\}, \{-12, -5.7\}\}$$

is a list of pairs of numbers, and $\{1, \{5, 3, 8\}, \{-8.3, 4\}, \texttt{"Hello"}\}$ is a list with a mixture of types.

Arithmetic operations on lists of numbers are similar to arithmetic operations on numbers. For example, if

```
list1={8,6,1,12,4};
list2={2,-2,1,-1,0};
```

then `4*list1` is the list $\{32, 24, 4, 48, 16\}$ and `list1-list2` is the list $\{6, 8, 0, 13, 4\}$.

Length, Sort functions

The command `Length[list]` returns the number of elements in `list`. The command `Sort[list]` returns a list of the elements of `list` in non-decreasing order. For example, if

`list=`$\{-3, 8, 12, -3, 4, 6, 2\}$`;`

then `Length[list]` returns 7, and `Sort[list]` returns $\{-3, -3, 2, 4, 6, 8, 12\}$.

Range command

The `Range` command is used to produce regularly-spaced lists of numbers. For example,

1. `Range[8]` returns $\{1, 2, 3, 4, 5, 6, 7, 8\}$.
2. `Range[5,12]` returns $\{5, 6, 7, 8, 9, 10, 11, 12\}$.
3. `Range[6,10,.5]` returns $\{6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\}$.

In the first two cases, the spacing is in whole-number units. In the third case, the spacing is in half-number units.

Table command

The `Table` command is used to build lists over specific ranges. For example,

1. `Table[`$x \wedge 2, \{x, 1, 10\}$`]` returns $\{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$.

2. `Table[`$\{x, x!\}, \{x, 0, 5\}$`]` returns $\{\{0,1\},\{1,1\},\{2,2\},\{3,6\},\{4,24\},\{5,120\}\}$.
3. `Table[`$x \wedge 2, \{x, 1, 4, 0.5\}$`]` returns $\{1, 2.25, 4., 6.25, 9., 12.25, 16.\}$.
4. `Table[Random[],`$\{20\}$`]` returns a list of 20 pseudo-random numbers from the interval $(0, 1)$.

In the first two cases, the form {`x,xmin,xmax`} indicates that the expression is evaluated for each whole number between `xmin` and `xmax`. In the third case, the form {`x,xmin,xmax,`δ} indicates that the expression is evaluated in increments of δ instead of 1. In the last case, no iterator is needed; each time the command is evaluated, a new list of 20 numbers is returned.

More generally, `Table` can produce multiply-indexed lists. For example, the command `Table[`$i - j, \{i, 1, 3\}, \{j, 1, 4\}$`]` returns the list

$$\{\{0, -1, -2, -3\}, \{1, 0, -1, -2\}, \{2, 1, 0, -1\}\}.$$

(A list of lists of the form $\{i-1, i-2, i-3, i-4\}$ is produced.)

Additional functions for operating on lists

The following table gives additional list functions. Note that the `CumulativeSums` function is from the `Statistics‘DataManipulation‘` package.

Mathematica Function	Returns
`Map[f,`$\{x_1, x_2, \ldots, x_n\}$`]`	$\{f(x_1), f(x_2), \ldots, f(x_n)\}$
`Outer[f,`$\{x_1, x_2, \ldots, x_n\}$`,` $\{y_1, y_2, \ldots, y_m\}$`]`	$\{\{f(x_1,y_1), f(x_1,y_2), \ldots, f(x_1,y_m)\},$ $\{f(x_2,y_1), f(x_2,y_2), \ldots, f(x_2,y_m)\}, \ldots\}$
`CumulativeSums[`$\{x_1, x_2, \ldots, x_n\}$`]`	$\{x_1, x_1+x_2, x_1+x_2+x_3, \ldots\}$
`Append[`$\{x_1, x_2, \ldots, x_n\}, x_{n+1}$`]`	$\{x_1, x_2, \ldots, x_n, x_{n+1}\}$
`Prepend[`$\{x_1, x_2, \ldots, x_n\}, x_0$`]`	$\{x_0, x_1, x_2, \ldots, x_n\}$
`Permutations[`ℓ`]`	the list of all permutations of objects in the list ℓ
`Join[`$\ell_1, \ell_2, \ldots$`]`	the list with the elements of list ℓ_1 followed by the elements of list ℓ_2, ...
`Union[`$\ell_1, \ell_2, \ldots$`]`	the list of distinct elements of the lists $\ell_1, \ell_2, \ldots$ written in ascending order
`Intersection[`$\ell_1, \ell_2, \ldots$`]`	the list of elements common to all lists $\ell_1, \ell_2, \ldots$

For example, the code

```
Clear[x]; Remove[f];
f[x_] := x∧3;
Map[f,{1,3,−2,4}]
```

returns the list $\{1, 27, -8, 64\}$, and the command `Permutations[`$\{-2, 1, 3, 3\}$`]` returns the following list of `Multinomial[1,1,2]`=12 lists

$$\{\{-2,1,3,3\},\{-2,3,1,3\},\{-2,3,3,1\},\{1,-2,3,3\},\{1,3,-2,3\},\{1,3,3,-2\},$$
$$\{3,-2,1,3\},\{3,-2,3,1\},\{3,1,-2,3\},\{3,1,3,-2\},\{3,3,-2,1\},\{3,3,1,-2\}\}.$$

Select function

The `Select` function can be used to choose elements of a list satisfying a given criterion. For example,

```
Clear[x]; Remove[f];
f[x_] := x < 10;
Select[{18,3,-2,12,5,-19,3},f]
```

returns the sublist of elements less than 10, $\{3, -2, 5, -19, 3\}$.

Note that "unnamed" (or *pure*) functions are often used in `Select` commands in the laboratory problems. For example, the single command

```
Select[{18,3,-2,12,5,-19,3},(#<10)&]
```

gives the same result as above. (The & says that the expression in parentheses is an unnamed function whose argument is #.)

Additional list-extraction functions

The following table gives additional functions for extracting elements from a list, or sublists of a list:

Mathematica Function	Returns
`First[list]`	the first element of `list`
`Last[list]`	the last element of `list`
`Part[list,n]` or `list[[n]]`	the n^{th} element of `list`
`Delete[list,n]`	a list with the n^{th} element of `list` removed
`Take[list,n]`	a list of the first n elements of `list`
`Take[list,-n]`	a list of the last n elements of `list`
`Drop[list,n]`	a list with the first n elements of `list` removed
`Drop[list,-n]`	a list with the last n elements of `list` removed

Notice in particular that a double bracket is used to extract the n^{th} element of a list. For example, if

```
samples={{1,8,3,12},{2,-4,14,6},{-8,7,1,2},{8,7,3,20}};
```

then `samples[[3]]` is the third element, $\{-8, 7, 1, 2\}$.

The `Part`, `Delete`, `Take`, and `Drop` commands can be defined more generally. For example, the command `list[[{5,8,12}]]` returns the sublist with the 5th, 8th, and 12th elements only. See the Help Browser for additional examples.

A.1.4 Summarizing lists of numbers

Functions for counting the number of times elements appear in lists, the numbers of elements in specific intervals, the sum and product of elements in lists, and for computing other summaries are described below. The `Frequencies` and `RangeCounts` functions are from the `Statistics‘DataManipulation‘` package, and the `Standardize` and `TrimmedMean` functions are from the `Statistics‘DescriptiveStatistics‘` package.

Functions for counting elements

`Count[list,`x`]`
returns the number of times x appears in `list`.

`Frequencies[list]`
returns the list of pairs $\{\{f_1, x_1\}, \{f_2, x_2\}, \ldots\}$ where x_1, x_2, $\ldots$ are the unique elements in `list` and x_i occurs f_i times, for $i = 1, 2, \ldots$.

`RangeCounts[list,`$\{x_1, x_2, \ldots, x_r\}$`]`
returns the list $\{f_0, f_1, \ldots, f_r\}$ where f_0 is the number of elements in the interval $x < x_1$, f_1 is the number of elements in the interval $x_1 \leq x < x_2$, f_2 is the number of elements in the interval $x_2 \leq x < x_3$, $\ldots$, and f_r is the number of elements in the interval $x \geq x_r$.

For example, if `sample` is the following list of 100 numbers

$$\{0,1,4,1,0,0,0,0,0,0,1,2,0,3,0,1,0,2,0,0,0,0,1,0,0,2,0,0,3,0,1,2,0,1,3,\\ 0,0,0,0,1,2,2,0,0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,1,3,0,3,0,0,0,0,0,0,0,\\ 1,1,0,0,1,1,0,1,0,1,0,0,0,1,2,2,1,0,0,1,0,0,7,0,0,1,2,3,1,0\}$$

then `Count[sample,2]` returns 9 and `Frequencies[sample]` returns the list

$$\{\{59,0\},\{24,1\},\{9,2\},\{6,3\},\{1,4\},\{1,7\}\}.$$

The unique elements in `sample` are 0, 1, 2, 3, 4, and 7. Zero appears in `sample` 59 times, one appears 24 times, etc.

Similarly, if `sample` is the following list of 50 numbers

$$\{3.77,\ 3.95,\ 5.16,\ 13.10,\ 0.48,\ 2.42,\ 0.16,\ 1.21,\ 1.81,\ 0.67,\ 3.62,\ 4.70,\ 3.98,\\ 10.29,\ 3.90,\ 0.06,\ 1.19,\ 8.11,\ 6.01,\ 2.77,\ 4.74,\ 6.93,\ 4.75,\ 5.90,\ 0.28,\ 0.66,\\ 11.99,\ 6.30,\ 9.74,\ 3.65,\ 8.64,\ 2.63,\ 2.50,\ 2.21,\ 1.02,\ 5.20,\ 5.93,\ 2.52,\ 2.84,\\ 4.94,\ 2.52,\ 3.20,\ 4.46,\ 0.78,\ 4.92,\ 4.53,\ 12.74,\ 1.59,\ 3.90,\ 2.75\}$$

then `RangeCounts[sample,`$\{1,3,5\}$`]` returns $\{7, 14, 15, 14\}$, which corresponds to the numbers of elements in the intervals

$$x < 1,\ 1 \leq x < 3,\ 3 \leq x < 5, \text{ and } x \geq 5, \text{ respectively.}$$

See the Help Browser for additional applications of these functions.

Finding sums and products

The `Apply` function is a general function that can be used to find sums and products without using an iterator. For example, the command

`Apply[Plus,`$\{1.43, 17.75, 6.33, 10.34, 29.25, 4.29, 8.26, 8.58\}$`]`

returns the sum, 86.23. Similarly, the command

`Apply[Times,`$\{2.2, 0.95, 5.09, 0.2, 1.87, 3.05, 2.46, 3.66\}$`]`

returns the product, 109.258.

Note that the shorthand `Plus@@list` can be used to find the sum of the numbers in `list`, and the shorthand `Times@@list` can be used to find the product of the numbers in `list`.

Additional summary functions

Mathematica Function	Returns
`Min[list]`	the minimum element of `list`
`Max[list]`	the maximum element of `list`
`Median[list]`	the median of `list`
`Mean[`$\{x_1, x_2, \ldots, x_n\}$`]`	$\overline{x} = (\sum_{i=1}^{n} x_i)/n$
`Variance[`$\{x_1, x_2, \ldots, x_n\}$`]`	$s^2 = (\sum_{i=1}^{n}(x_i - \overline{x})^2)/(n-1)$
`StandardDeviation[`$\{x_1, x_2, \ldots, x_n\}$`]`	$s = \sqrt{s^2}$
`Standardize[`$\{x_1, x_2, \ldots, x_n\}$`]`	$\{z_1, z_2, \ldots, z_n\}$ where $z_i = (x_i - \overline{x})/s$
`TrimmedMean[`$\{x_1, x_2, \ldots, x_n\}$`,`$\alpha$`]`	`Mean[`$\{x_{(k+1)}, x_{(k+2)}, \ldots, x_{(n-k)}\}$`]` where $k =$ `Floor[n*`α`]`
`Dot[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_n\}$`]` or $\{x_1, x_2, \ldots, x_n\}.\{y_1, y_2, \ldots, y_n\}$	$x_1y_1 + x_2y_2 + \cdots + x_ny_n$

Note in particular that `Standardize` returns a list with mean 0 and variance 1.

A.1.5 Structured lists and matrices

Structured lists and matrices (defined below) are used in many statistical applications. Functions for extracting elements of structured lists, for extracting rows and columns of matrices, for transposing matrices, for computing the inverse of a square matrix, for computing products of matrices, for constructing matrices by partitioning, for removing structure, and for displaying structured lists in row-column form are summarized below.

Definitions; rows, columns, elements

A *structured list* is a list whose elements are themselves lists. An n-by-m *matrix* is a structured list of n lists (the rows) of m elements each (the columns).

If `matrix` is an n-by-m matrix, then `matrix[[i]]` is the i^{th} row of the matrix, and `Column[matrix,j]` is the j^{th} column of the matrix. For example, if `matrix` is

$$\{\{3,5,8\},\{2,6,4\},\{1,2,10\},\{0,2,4\}\}$$

then `matrix[[2]]` is $\{2,6,4\}$ and `Column[matrix,2]` is $\{5,6,2,2\}$.

If `list` is a structured list, then `list[[`i,j`]]` is the j^{th} part of the i^{th} element of `list`. For example, if `matrix` is the 4-by-3 matrix above, then `matrix[[4,1]]` is the first element in the fourth row, 0.

Exchanging rows and columns

The `Transpose` function can be used to exchange the rows and columns of an n-by-m matrix. For the 4-by-3 matrix above, for example, `Transpose[matrix]` returns the following 3-by-4 matrix

$$\{\{3,2,1,0\},\{5,6,2,2\},\{8,4,10,4\}\}.$$

Inverse of a matrix, product of matrices

If `matrix` is an n-by-n matrix of numbers, then `Inverse[matrix]` returns the inverse of the matrix (if it exists). For example, the commands

```
matrix = {{2,5},{1,3}};
Inverse[matrix]
```

return the inverse of the 2-by-2 matrix, $\{\{3,-5\},\{-1,2\}\}$.

If `m1` is an n-by-m matrix of numbers and `m2` is an m-by-p matrix of numbers, then the command `m1.m2` returns the n-by-p matrix product. For example, the commands

```
m1={{8,0,2},{1,-4,1}};
m2={{4,1},{0,3},{-1,-3}};
m1.m2
```

return the 2-by-2 matrix $\{\{30,2\},\{3,-14\}\}$. Similarly, given the 2-by-2 invertible `matrix` above, the command `matrix.Inverse[matrix]` returns the 2-by-2 identity matrix, $\{\{1,0\},\{0,1\}\}$.

Additional functions for structured lists

Mathematica Function	Returns
`Partition[`$\{x_1,x_2,\ldots,x_{nm}\}$`,m]`	the n-by-m matrix with rows $\{x_1,x_2\ldots,x_m\}$, $\{x_{m+1},x_{m+2},\ldots,x_{2m}\}$, $\ldots$, $\{x_{nm-m+1},\ldots,x_{nm}\}$.
`Flatten[list]`	the non-structured list obtained by removing all internal curly braces.
`Flatten[list,`i`]`	the list obtained by flattening to level i.
`TableForm[list]`	the elements of `list` printed in rows and columns.

For example, if `samples` is the following 2-by-3 matrix of sample lists

$$\{\{\{3,9,7,6\},\{1,18,4,14\},\{2,8,20,14\}\},\{\{3,7,9,1\},\{6,7,7,5\},\{9,13,10,9\}\}\},$$

then `Flatten[samples,1]` produces the following list of 6 sample lists

$$\{\{3,9,7,6\},\{1,18,4,14\},\{2,8,20,14\},\{3,7,9,1\},\{6,7,7,5\},\{9,13,10,9\}\}.$$

(One level of curly braces is removed.)

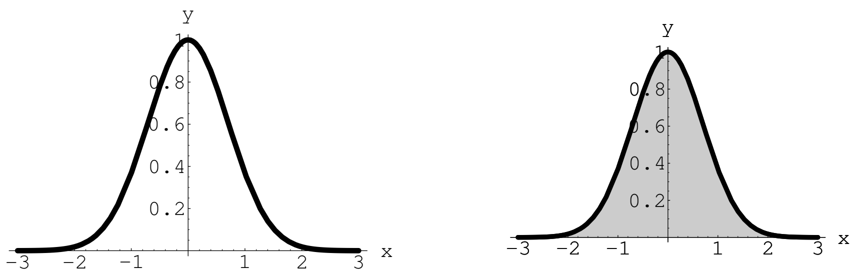

Figure A.1. *Unfilled (left) and filled (right) plots of* $y = \exp(-x^2)$.

A.1.6 Basic graphing

Functions for constructing two-dimensional plots are summarized below. A general scatter plot function, used to produce plots of one or more lists of pairs of numbers, is introduced in Section A.2.6. See the Help Browser for additional examples and for instructions on constructing three-dimensional plots.

Plot and FilledPlot functions

The `Plot` and `FilledPlot` commands can be used to construct two-dimensional plots of functions $y = f(x)$, as described below. Note that `FilledPlot` is from the `Graphics‘FilledPlot‘` package.

`Plot[f[x],{x,xmin,xmax},options]`
returns a plot of $f(x)$ for x in the interval `[xmin,xmax]`.
 	
`FilledPlot[f[x],{x,xmin,xmax},options]`
returns a plot with the area between $f(x)$ and the x-axis filled.

See the Help Browser for the complete list of options available for these functions. For example, the command

```
Plot[Exp[-x∧2],{x,-3,3},
  PlotStyle→Thickness[.015],
  AxesLabel→{"x","y"}];
```

produced the left part of Figure A.1, and the command

```
FilledPlot[Exp[-x∧2],{x,-3,3},
  PlotStyle→Thickness[.015],
  Fills→GrayLevel[0.80],
  AxesLabel→{"x","y"}];
```

produced the right part of the figure.

A.1.7 Derivatives and integrals

Functions for computing derivatives and partial derivatives, and for computing definite and indefinite integrals are summarized below.

Derivatives and partial derivatives

The command `D[expression,x]` returns the derivative (or partial derivative) of `expression` with respect to x. The `expression` can contain any number of symbols. For example, if `expression` is the polynomial

$$7 + a\,x^2 - 2\,b\,x\,y + 15\,c\,x^3\,z^2,$$

then `D[expression,x]` returns $2\,a\,x - 2\,b\,y + 45\,c\,x^2\,z^2$.

For functions of one variable, the shorthand `f'[x]` can also be used to return the derivative. For example,

```
Clear[x]; Remove[f];
f[x_] := Sin[2*x];
f'[x]
```

returns `2 Cos[2 x]`. Similarly, `f''[x]` returns `-4 Sin[2 x]`.

Definite and indefinite integrals

The `Integrate` command is used to compute antiderivatives and to compute definite integrals. For the function above, for example, `Integrate[f[x],x]` returns the antiderivative `-Cos[2 x]/2` and the command

```
Integrate[f[x], {x,0,Pi/4}]
```

returns 1/2. (The area under the curve $y = f(x)$ and above the interval $[0, \pi/4]$ on the the x-axis is 1/2 square units.)

More generally, `Integrate` can be used to compute multiple integrals. For example, the commands

```
Clear[x,y];
Integrate[x∧2 + 4y∧2, {x,0,3}, {y,0,x}]
```

return 189/4. (The volume under the surface $z = x^2 + 4y^2$ and above the triangular region in the xy-plane with corners $(0,0)$, $(3,0)$, $(3,3)$ is 189/4 cubic units.)

Assumptions in definite integrals

The option `Assumptions` is used to specify conditions on symbols in the integrand. For example, the commands

```
Clear[a,x];
Integrate[Exp[-a*x], {x,0,∞}, Assumptions → a> 0]
```

return $1/a$. (If $a > 0$, then the value of the integral exists and equals $1/a$.) See the Help Browser for additional situations where the `Assumptions` option is used.

A.1.8 Symbolic operations and solving equations

Functions for working with polynomial and rational expressions, for replacing occurrences of one subexpression with another, and for solving equations exactly and approximately are summarized below.

Working with polynomial and rational expressions

Mathematica Function	Action
`Expand[expression]`	multiplies out the products and powers in `expression`.
`Factor[expression]`	returns an equivalent product of factors.
`Together[expression]`	combines terms of `expression` over a common denominator.
`Apart[expression]`	splits `expression` into partial fractions.
`Simplify[expression]`	returns a simplied form.
`Coefficient[polynomial,x,r]`	returns the coefficient of x^r in the expanded `polynomial`.

For example, the command

```
Clear[t];
expression = Expand[(1 + t)∧8]
```

returns the polynomial

$$1 + 8\,t + 28\,t^2 + 56\,t^3 + 70\,t^4 + 56\,t^5 + 28\,t^6 + 8\,t^7 + t^8$$

and the command `Coefficient[expression,t,4]` returns 70.

Replacing subexpressions

The `ReplaceAll` function is used to replace all occurrences of one subexpression with another (or all occurrences of many subexpressions with others).

Mathematica Function	Action
`ReplaceAll[expression,x→a]` or `expression /. x→a`	replaces all occurrences of x in `expression` with a.
`ReplaceAll[expression,`$\{x \to a, y \to b, \ldots\}$`]` or `expression /.` $\{x \to a, y \to b, \ldots\}$	replaces all occurrences of x with a, y with b, etc.

An expression of the form `lhs→rhs` is called a *rule*. The rule `lhs→rhs` means "replace the lefthand side (`lhs`) with the value of the righthand side (`rhs`)."

For example, if `expression` contains the polynomial

$$5\,x^2 - 15\,x\,y + 27\,z^2$$

then the command

```
expression /. x→(y+z)∧2
```

returns the polynomial

$$27\,z^2 - 15\,y\,(y+z)^2 + 5\,(y+z)^4$$

Solving equations

The `Solve` and `NSolve` functions can be used to find solutions to polynomial and rational equations:

Mathematica Function	Action
`Solve[equation, x]`	solves `equation` for x.
`NSolve[equation,x]`	solves `equation` for x numerically.
`Solve[`$\{e_1, e_2, \ldots\}$`, `$\{x_1, x_2, \ldots\}$`]`	solves the system of equations e_1, e_2, ..., for variables x_1, x_2, ...
`NSolve[`$\{e_1, e_2, \ldots\}$`, `$\{x_1, x_2, \ldots\}$`]`	solves the system numerically.

For example, the command

```
Solve[240+10*x-45*x^2+5*x^3==0,x]
```

returns the list of replacement rules $\{\{x \to -2\}, \{x \to 3\}, \{x \to 8\}\}$. (Each rule corresponds to a root of the polynomial equation $240 + 10x - 45x^2 + 5x^3 = 0$.) Note that a double equal sign (==) is used in the first argument of `Solve`.

FindRoot command

More generally, `FindRoot` can be used to find approximate solutions to equations using a sequence of iterations:

`FindRoot[equation,`$\{x, x_0\}$`]` searches for a numerical solution to `equation` starting from $x = x_0$.
`FindRoot[`$\{e_1, e_2, \ldots\}$`, `$\{x, x_0\}$`, `$\{y, y_0\}$`, ...]` searches for a numerical solution to the system of equations e_1, e_2, ... in variables x, y, ... using the starting values $x = x_0$, $y = y_0$, ...

For example, the graphs of $y = x + 4$ and $y = e^{x/3}$ have a point of intersection between $x = 6$ and $x = 8$. Using $x = 7$ as a starting value, the command

```
FindRoot[x+4 == Exp[x/3], {x,7}]
```

returns the list $\{x \to 7.26514\}$, indicating that the point of intersection has x-coordinate approximately 7.26514. Note that a double equal sign (==) is used in the first argument of `FindRoot`.

A.1.9 Univariate probability distributions

The following tables give information on the discrete and continuous families of distributions used in the book. The information is from the

`Statistics‘DiscreteDistributions‘` and
`Statistics‘ContinuousDistributions‘` packages.

Discrete Distributions	Continuous Distributions
BernoulliDistribution[p]	CauchyDistribution[a,b]
BinomialDistribution[n,p]	ChiSquareDistribution[n]
DiscreteUniformDistribution[n]	ExponentialDistribution[λ]
GeometricDistribution[p]	FRatioDistribution[ν_1,ν_2]
HypergeometricDistribution[n,mm,nn]	GammaDistribution[α,β]
NegativeBinomialDistribution[r,p]	LaplaceDistribution[μ,β]
PoissonDistribution[λ]	LognormalDistribution[μ,σ]
	NormalDistribution[μ,σ]
	StudentTDistribution[n]
	UniformDistribution[a,b]

If X has the `model` distribution, then

Mathematica Function	Returns
Mean[model]	$E(X)$
Variance[model]	$Var(X)$
StandardDeviation[model]	$SD(X)$
CDF[model,x]	$F(x) = P(X \leq x)$
PDF[model,x]	$P(X = x)$ when X is discrete, and $F'(x)$ when X is continuous.
Quantile[model,p]	the p^{th} quantile ($100p^{\text{th}}$ percentile)
Random[model]	a pseudo-random number from the `model` distribution
RandomArray[model,r]	a list of r pseudo-random numbers
RandomArray[model,$\{r,s\}$]	an r-by-s matrix of pseudo-random numbers

For example, if

```
model=ExponentialDistribution[1/10];
```

then `Mean[model]` returns `10` and `Quantile[model,1/2]` returns `10 Log[2]`.

A.1.10 Confidence interval procedures for normal samples

Procedures to return confidence intervals for the mean or variance of a normal distribution when both parameters are unknown, and to return confidence intervals for the difference in means or ratio of variances of normal distributions when all parameters are unknown are summarized below. In each case, if the confidence level option is omitted, then a 95% confidence interval is returned.

The procedures are from the `Statistics‘ConfidenceIntervals‘` package.

MeanCI[$\{x_1, x_2, \ldots, x_n\}$,ConfidenceLevel$\rightarrow 1-\alpha$]
returns the interval $\overline{x} \pm t_{n-1}(\alpha/2)\sqrt{s^2/n}$.

VarianceCI[$\{x_1, x_2, \ldots, x_n\}$,ConfidenceLevel$\rightarrow 1-\alpha$]
returns the interval $\left[(n-1)s^2/\chi^2_{n-1}(1-\alpha/2), (n-1)s^2/\chi^2_{n-1}(\alpha/2)\right]$.

`MeanDifferenceCI[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}$`,ConfidenceLevel`$\rightarrow 1-\alpha$`]`
returns the interval $(\overline{x}-\overline{y}) \pm t_{df}(\alpha/2)\sqrt{s_x^2/n + s_y^2/m}$, where *df* is computed using Welch's formula.

`MeanDifferenceCI[`$\{x_1, x_2, \ldots, x_n\}$`, `$\{y_1, y_2, \ldots, y_m\}$`,EqualVariances→True,`
`ConfidenceLevel`$\rightarrow 1-\alpha$`]`
returns the interval $(\overline{x}-\overline{y}) \pm t_{n+m-2}(\alpha/2)\sqrt{s_p^2\ (1/n + 1/m)}$.

`VarianceRatioCI[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}$`,ConfidenceLevel`$\rightarrow 1-\alpha$`]`
returns the interval $\left[(s_x^2/s_y^2)/f_{n-1,m-1}(1-\alpha/2), (s_x^2/s_y^2)/f_{n-1,m-1}(\alpha/2)\right]$.

For example, the commands

`sample=`$\{4.79, 5.04, 7.63, 6.99, 5.81, 5.08, 3.81, 4.27, 6.27, 1.53, 6.46, 6.32\}$`;`
`MeanCI[sample,ConfidenceLevel→0.90]`

returns the 90% confidence interval for the mean, $[4.48106, 6.18561]$.

A.1.11 Hypothesis test procedures for normal samples

Procedures for tests concerning the mean or variance of a normal distribution when both parameters are unknown, and for tests concerning the difference in means or ratio of variances of normal distributions when all parameters are unknown are summarized below. In each case, use the option `TwoSided→True` to return the p value for a two sided test of the null hypothesis. In each case, use the option `FullReport→True` to return a report containing the observed values of the unknown parameter and test statistic, and the appropriate sampling distribution.

The procedures are from the `Statistics‘HypothesisTests‘` package.

`MeanTest[`$\{x_1, x_2, \ldots, x_n\}, \mu_0$`]`
returns the p value for a one sided t test test of $\mu = \mu_0$.

`VarianceTest[`$\{x_1, x_2, \ldots, x_n\}, \sigma_0^2$`]`
returns the p value for a one sided chi square test of $\sigma^2 = \sigma_0^2$.

`MeanDifferenceTest[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}, \delta_0$`]`
returns the p value for a one sided Welch t test of $\mu_x - \mu_y = \delta_0$.

`MeanDifferenceTest[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}, \delta_0$`, EqualVariances→True]`
returns the p value for a one sided pooled t test of $\mu_x - \mu_y = \delta_0$.

`VarianceRatioTest[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}$`,`$r_0$`]`
returns the p value for a one sided f test of $\sigma_x^2/\sigma_y^2 = r_0$.

For example, the commands

`s1=`$\{3.68, 4.29, 4.62, 1.09, 5.54, 2.68, 4.17, 5.12, 6.25, 7.11, 6.94, 5.49\}$`;`
`s2=`$\{2.17, 0.71, 5.67, 1.87, 2.06, 1.05, 1.76, 4.73\}$`;`
`MeanDifferenceTest[s1,s2,0,EqualVariances→True,TwoSided→True]`

return the p value for a two sided pooled t test of the equality of means ($\delta_o = 0$) as a *Mathematica* rule, `TwoSidedPValue→0.0113969`.

A.1.12 Linear and nonlinear least squares

Functions for finding linear and nonlinear least squares formulas, and for linear regression analyses are summarized below.

Fit function

The `Fit` command for linear least squares is summarized below:

`Fit[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,functions,`$x$`]`
returns the least squares fit of $y = f(x)$ to the data list, where the expression for $f(x)$ is a linear combination of `functions` of x.
 	
`Fit[`$\{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \ldots\}$`,functions,`$\{x, y\}$`]`
returns the least squares fit of $z = f(x, y)$ to the data list, where the expression for $f(x, y)$ is a linear combination of `functions` of x and y.
 	
Etcetera

For example, if

`pairs=`$\{\{25.10, 2.88\}, \{72.17, 34.99\}, \{25.98, 1.28\}, \{54.93, 16.24\}, \{69.64, 18.15\},$
$\{45.75, 32.16\}, \{0.67, -11.38\}, \{69.18, 8.44\}, \{2.41, -10.21\}, \{15.52, -14.62\}\}$;

is the list of data `pairs`, then the command

```
Clear[x]; Remove[f];
f[x_] = Fit[pairs,{1,x},x]
```

computes the least squares linear fit formula (the formula $-11.9815+0.51854\,x$) and defines the function $y = f(x)$ whose value is that formula. Note that the formula is computed and stored since an equal sign is used instead of a colon-equal. (This is a situation where immediate assignment should be used.)

FindFit function

The `FindFit` command is summarized below:

`FindFit[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,function,parameters,`$x$`]`
returns numerical values of the `parameters` for the least squares fit of $y = f(x)$ to the data list, where `function` is the expression for $f(x)$ as a function of x and the unknown `parameters`.
 	
`FindFit[`$\{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \ldots\}$`,function,parameters,`$\{x, y\}$`]`
returns numerical values of the `parameters` for the least squares fit of $z = f(x, y)$ to the data list, where `function` is the expression for $f(x, y)$ as a function of x, y and the unknown `parameters`.
 	
Etcetera

For example, if

pairs=$\{\{1,-0.15\},\{1,0.93\},\{1,0.35\},\{2,1.87\},\{2,8.99\},\{2,7.68\},\{3,8.43\},\{3,9.04\},$
$\{3,15.35\},\{4,17.95\},\{4,20.04\},\{4,21.03\},\{5,37.27\},\{5,35.56\},\{5,39.54\}\}$;

is the list of data `pairs`, then the commands

```
Clear[a,b,c,x]; Remove[f];
function = (a + b*x + c*x∧2)∧2;
f[x_] = function /. FindFit[pairs,function,{a, b, c},x]
```

do the following:

(i) Initialize the expression for the square-quadratic function;
(ii) Use FindFit to find numerical values for the parameters in the function and replace the parameters by their numerical values (for the data above the expression becomes $(0.700554 + 0.505929x + 0.11439x^2)^2$); and
(iii) Define the function $y = f(x)$ whose value is the estimated function. Note that the estimated function is stored since an equal sign is used instead of a colon-equal. (This is a situation where immediate assignment should be used.)

Consult the Help Browser for additional information on the FindFit function.

Regress function

The `Regress` command for linear regression analyses is summarized below. Note that `Regress` is from the `Statistics‘LinearRegression‘` package.

`Regress[`$\{\{x_1,y_1\},\{x_2,y_2\},\ldots\}$`,functions,`$x$`,`
`RegressionReport→options]`
returns a list of replacement rules for linear regression analysis based on the least squares fit of $y = f(x)$ to the data list, where the expression for $f(x)$ is a linear combination of `functions` of x.

`Regress[`$\{\{x_1,y_1,z_1\},\{x_2,y_2,z_2\},\ldots\}$`,functions,`$\{x,y\}$`,`
`RegressionReport→options]`
returns a list of replacement rules for linear regression analysis based on the least squares fit of $z = f(x,y)$ to the data list, where the expression for $f(x,y)$ is a linear combination of `functions` of x and y.

Etcetera

Options used in Chapter 14 include `ANOVATable`, `ParameterCITable`, `EstimatedVariance`, `RSquared`, `StandardizedResiduals`, `PredictedResponse`, `FitResiduals`, `PredictedResponseDelta`. For example, if

pairs=$\{\{25.10,2.88\},\{72.17,34.99\},\{25.98,1.28\},\{54.93,16.24\},\{69.64,18.15\},$
$\{45.75,32.16\},\{0.67,-11.38\},\{69.18,8.44\},\{2.41,-10.21\},\{15.52,-14.62\}\}$;

is the list of data `pairs`, then the command

```
Clear[x];
results = Regress[pairs,{1, x},x,
  RegressionReport→{ANOVATable, RSquared}];
```

does the following:

(i) Carries out a linear regression analysis where the hypothesized conditional expectation is $E(Y|X=x) = \beta_0 + \beta_1 x$; and

(ii) Stores a list containing the analysis of variance f test and the coefficient of determination in `results`.

Each member of the `results` list can be retrieved using replacement commands. In particular, the command `ANOVATable /. results` returns the table

Source	df	SS	MS	F	p value
Model	1	1872.45	1872.45	16.6324	0.0035
Error	8	900.626	112.578		
Total	9	2773.08			

and the command `RSquared /. results` returns 0.675225.

The linear regression package has many options. Consult the Help Browser for additional information and examples.

A.2 Custom tools

This section introduces the customized tools available with this book. Online help for a particular `Symbol` is available by evaluating `?Symbol`. Note that since these tools are not part of the standard *Mathematica* system, additional help is not available in the Help Browser.

A.2.1 Bivariate probability distributions

Three bivariate distributions are included:

`TrinomialDistribution[`n, $\{p_1, p_2, p_3\}$`]`
represents the trinomial distribution based for n trials with probabilities p_1, p_2, and p_3.

`BivariateHypergeometricDistribution[`n, $\{m_1, m_2, m_3\}$`]`
represents the bivariate hypergeometric distribution for a sample of size n drawn without replacement from a population with m_i objects of type i $(i = 1, 2, 3)$.

`BivariateNormalDistribution[`ρ`]`
represents the standard bivariate normal distribution with correlation ρ.

If (X, Y) has the `model` distribution, then

Mathematica Function	Returns
`Mean[model]`	$\{E(X), E(Y)\}$
`Variance[model]`	$\{Var(X), Var(Y)\}$
`StandardDeviation[model]`	$\{SD(X), SD(Y)\}$
`Correlation[model]`	$Corr(X, Y)$
`PDF[model,x,y]`	the joint PDF at (x, y)
`Random[model]`	a pseudo-random pair from the `model` distribution
`RandomArray[model,`r`]`	a list of r pseudo-random pairs
`RandomArray[model,`$\{r, s\}$`]`	an r-by-s matrix of pseudo-random pairs

For example, if

`model=TrinomialDistribution[10,`$\{0.2, 0.3, 0.5\}$`];`

then `Mean[model]` returns $\{2., 3.\}$ and `Correlation[model]` returns -0.327327.

A.2.2 Multinomial distribution and goodness-of-fit

The multinomial model is included:

`MultinomialDistribution[`n, $\{p_1, p_2, \ldots, p_k\}$`]`
represents the multinomial distribution for n trials with probabilities p_1, p_2, ..., p_k.

If $(X_1, X_2, \ldots, X_k)$ has the `model` distribution, then

Mathematica Function	Returns
`Mean[model]`	$\{E(X_1), E(X_2), \ldots, E(X_k)\}$
`Variance[model]`	$\{Var(X_1), Var(X_2), \ldots, Var(X_k)\}$
`StandardDeviation[model]`	$\{SD(X_1), SD(X_2), \ldots, SD(X_k)\}$
`Random[model]`	a pseudo-random observation of the form $\{x_1, x_2, \ldots, x_k\}$
`RandomArray[model,`r`]`	a list of r pseudo-random observations

Goodness-of-fit

The `GOFTest` command implements Pearson's goodness-of-fit test. The command returns the observed value of Pearson' statistic, the p value computed using the chi-square distribution with *df* degrees of freedom, and a diagnostic plot of standardized residuals $r_i = (x_i - np_i)/\sqrt{np_i}$, $i = 1, 2, \ldots, k$.

`GOFTest[MultinomialDistribution[`n, $\{p_1, p_2, \ldots, p_k\}$`],`$\{x_1, x_2, \ldots, x_k\}$`,`$df$`]`
returns a multinomial goodness-of-fit analysis based on Pearson's statistic. The observed frequency list, $\{x_1, x_2, \ldots, x_k\}$, must have sum n. The degrees of freedom, *df*, must be an integer between 1 and $k - 1$. Each expected value must be 4.0 or more.

For example, the commands

```
observed = {16, 11, 6, 17};
model = MultinomialDistribution[50, {0.25, 0.25, 0.25, 0.25}];
GOFTest[model, observed, 3]
```

return a plot of standardized residuals from the test of the null hypothesis that the observed frequency list is consistent with an equiprobable model, the observed value of Pearson's statistic (6.16), and the p value based on the chi-square distribution with 3 df (0.104). Further, the commands

```
expected = Mean[model];
N[(observed - expected)/Sqrt[expected]]
```

return the list of standardized residuals, $\{0.989949, -0.424264, -1.83848, 1.27279\}$. The sum of squares of the standardized residuals is the value of Pearson's statistic.

A.2.3 Generating subsets

Functions for generating random subsets, and lists of subsets are included.

`RandomSubset[`n, r`]`
returns a randomly chosen subset of r distinct elements from the collection $\{1, 2, ..., n\}$, written in ascending order. Each subset is chosen with probability $1/\binom{n}{r}$.

`RandomSubset[`n`]`
returns a randomly chosen subset of distinct elements from the collection $\{1, 2, ..., n\}$, written in ascending order. Each subset is chosen with probability $1/2^n$.

`AllSubsets[`n, r`]`
returns the list of all $\binom{n}{r}$ subsets of r distinct elements from the collection $\{1, 2, ..., n\}$.

`AllSubsets[`n`]`
returns the list of all 2^n subsets of distinct elements from the collection $\{1, 2, ..., n\}$.

For example, `AllSubsets[8,3]` returns the following list of 56 subsets:

$$\{\{1,2,3\},\{1,2,4\},\{1,2,5\},\{1,2,6\},\{1,2,7\},\{1,2,8\},\{1,3,4\},\{1,3,5\},\{1,3,6\},\{1,3,7\},\{1,3,8\},\{1,4,5\},\{1,4,6\},\{1,4,7\},\{1,4,8\},\{1,5,6\},\{1,5,7\},\{1,5,8\},\{1,6,7\},\{1,6,8\},\{1,7,8\},\{2,3,4\},\{2,3,5\},\{2,3,6\},\{2,3,7\},\{2,3,8\},\{2,4,5\},\{2,4,6\},\{2,4,7\},\{2,4,8\},\{2,5,6\},\{2,5,7\},\{2,5,8\},\{2,6,7\},\{2,6,8\},\{2,7,8\},\{3,4,5\},\{3,4,6\},\{3,4,7\},\{3,4,8\},\{3,5,6\},\{3,5,7\},\{3,5,8\},\{3,6,7\},\{3,6,8\},\{3,7,8\},\{4,5,6\},\{4,5,7\},\{4,5,8\},\{4,6,7\},\{4,6,8\},\{4,7,8\},\{5,6,7\},\{5,6,8\},\{5,7,8\},\{6,7,8\}\}$$

The command `RandomSubset[8,3]` will return one of the 56 subsets shown above.

A.2.4 Probability theory tools

Procedures supporting introductory probability concepts (Chapter 1) and ideas related to limit theorems (Chapter 5) are included.

Card and lottery games

`RandomCardHand[`n`]`
returns a plot of a randomly chosen hand of n distinct cards from a standard deck of 52 cards. Each hand is chosen with probability $1/\binom{52}{n}$.

`RandomCardHand[`n`,CardSet→list]`
returns a plot of a randomly chosen hand of n distinct cards with the cards whose numerical values are in `list` highlighted. Add the option `Repetitions→` r to repeat the experiment "choose n cards and record the number of cards in `list`" r times and return the results.

`RandomCardHand[]`
returns a plot of the numerical values of each card in a standard deck.

`RandomLotteryGame[`nn`,`n`]`
returns the results of a random state lottery game: (1) the state randomly chooses a subset of n distinct numbers from the collection $\{1, 2, ..., nn\}$ (shown as vertical lines); (2) the player randomly chooses a subset of n distinct numbers from the same collection (shown as black dots for numbers matching the state's picks, and red dots otherwise). Each choice of subset is equally likely.

`RandomLotteryGame[`nn`,`n`,Repetitions→` r`]`
repeats the experiment "play the game and record the number of matches" r times and returns the list of results.

Simple urn model

`UrnProbability[`nn, mm, n, m`]`
returns the probability that a randomly chosen subset of n distinct objects will contain exactly m special objects, where nn is the total number of objects and mm is the number of special objects in the urn: $\left(\binom{mm}{m}\binom{nn-mm}{n-m}\right)/\binom{nn}{n}$.

`UrnProbabilityPlot[`nn, mm, n, m`]`
returns a plot of urn probability as a function of nn or mm, and returns a parameter value with maximum probability. Use a symbol to represent the parameter of interest.

`UrnProbabilityPlot[`nn, mm, n, m`,ProbabilityRatio→` p`]`
returns a plot of urn probability as a function of nn or mm, a parameter value with maximum probability, and the list of parameter values whose probability is at least p times the maximum probability ($0 < p \leq 1$). Use a symbol to represent the parameter of interest.

For example, suppose that an urn contains exactly 120 marbles, that each marble is either red or blue, that the exact number of red marbles is not known, and that in a simple random sample of 10 marbles, exactly 8 are red. Then the command

```
Clear[mm];
UrnProbabilityPlot[120,mm,10,8,ProbabilityRatio→0.25]
```

returns a plot of urn probabilities as a function of the unknown total number of red marbles (mm), a report identifying 96 as the most likely number of red marbles, the urn probability when mm=96 (0.315314), and a list of estimates of mm with urn probabilities at least 0.25×0.315314 (the list is $68, 69, \ldots, 113$).

Sequences of running sums and averages

RandomSumPath[model,n] generates a random sample of size n from the univariate model distribution and returns a line plot of running sums, along with the value of the n^{th} sum. RandomAveragePath[model,n] generates a random sample of size n from the univariate model distribution and returns a line plot of running averages, along with the value of the n^{th} average.

RandomWalk2D[$\{p_{NE}, p_{NW}, p_{SE}, p_{SW}\}$,$n$] returns a plot of a random walk of n steps in the plane beginning at the origin, along with the final position. The walk goes northeast one step with probability p_{NE}, northwest with probability p_{NW}, southeast with probability p_{SE}, and southwest with probability p_{SW}. RandomWalk2D[$\{p_{NE}, p_{NW}, p_{SE}, p_{SW}\}$,$n$,Repetitions→ r] returns a list of r final positions of random walks of n steps beginning at the origin.

A.2.5 Statistics theory tools

Procedures supporting estimation theory concepts (Chapter 7) and hypothesis testing theory concepts (Chapter 8) are included.

Maximum likelihood estimation

The LikelihoodPlot function allows you to visualize ML estimation in specific situations, as described below.

LikelihoodPlot[model,data] returns a plot of the likelihood function for the given single-parameter model and data. Use a symbol to represent the parameter to be estimated. LikelihoodPlot[] returns the list of allowed models and parameters.

For example, the commands

```
sample={5.51,0.05,1.57,4.72,4.09,2.40,6.90,5.37,3.15,3.87};
Clear[σ];
LikelihoodPlot[NormalDistribution[3,σ],sample]
```

return a normal likelihood plot for the `sample` data and the ML estimate of σ^2, 4.31603. The computations assume that the `sample` data are the values of a random sample from a normal distribution with $\mu = 3$.

Constructing tests and computing power

The `PowerPlot` function allows you to visualize sampling distributions of test statistics under specific null and alternative models, and compute rejection regions and power, as described below.

> `PowerPlot[LowerTail,model0,model1,`n`,`α`]` and
> `PowerPlot[UpperTail,model0,model1,`n`,`α`]`
> return information for one-sided $100\alpha\%$ tests, where `model0` is the null model, `model1` is the alternative model, and n is the number of independent observations. In each case the function returns plots of the distribution of the test statistic when sampling from the null model (in gray) and when sampling from the alternative model (the acceptance region in blue; the rejection region in red) and displays the rejection region and the power of the test.
>
> `PowerPlot[]` lists the allowed models and null hypotheses.

For example, consider testing $p = 0.40$ versus $p < 0.40$ at the 5% significance level using a random sample of size 50 from a Bernoulli distribution and the sample sum, $Y = \sum_i X_i$, as test statistic. (Y has a binomial distribution with $n = 50$.) The following code

```
model0 = BernoulliDistribution[0.4];
model1 = BernoulliDistribution[0.3];
PowerPlot[LowerTail, model0, model1, 50, 0.05]
```

returns a plot of the Y distribution when $p = 0.40$ (the null model) and when $p = 0.30$ (a specific alternative model) on the same set of axes, and the information below

Rejection Region:	Significance Level:	Power when $p = 0.30$:
$\sum_i X_i \leq 14$	0.053955	0.446832

Note that the significance level is not exactly 0.05 since the test statistic is discrete.

Simulation functions

The `RandomTCI`, `RandomChiSquareCI`, `RandomTTest`, and `RandomChiSquareTest` functions allow you to study the behavior of test and confidence interval procedures for normal samples under a variety of different sampling situations.

`RandomTCI[model,`n`,`$1-\alpha$`,`r`]`
returns a plot of r random $100(1-\alpha)\%$ confidence intervals for $\mu = E(X)$. Intervals containing μ are shown in blue; those not containing μ are shown in red. Each interval is constructed using a random sample of size n from the `model` distribution and the procedure to construct a confidence interval for data from a normal distribution when σ is unknown. The proportion of intervals containing μ is displayed.

`RandomChiSquareCI[model,`n`,`$1-\alpha$`,`r`]`
returns a plot of r random $100(1-\alpha)\%$ confidence intervals for $\sigma^2 = Var(X)$. Intervals containing σ^2 are shown in blue; those not containing σ^2 are shown in red. Each interval is constructed using a random sample of size n from the `model` distribution and the procedure to construct a confidence interval for data from a normal distribution when μ is unknown. The proportion of intervals containing σ^2 is displayed.

`RandomTTest[LowerTail,`μ_0`,model,`n`,`α`,`r`]` and
`RandomTTest[UpperTail,`μ_0`,model,`n`,`α`,`r`]`
return information from a simulation study of one-sided $100\alpha\%$ tests of $\mu = \mu_0$. In each case the function generates r random t statistics, each based on a random sample of size n from the `model` distribution, and returns a histogram of the statistics (the acceptance region in blue; the rejection region in red) superimposed on the Student t distribution with $(n-1)$ degrees of freedom (in gray). The proportion of statistics in the rejection region is reported.

`RandomChiSquareTest[LowerTail,`σ_0`,model,`n`,`α`,`r`]` and
`RandomChiSquareTest[UpperTail,`σ_0`,model,`n`,`α`,`r`]`
return information from a simulation study of one-sided $100\alpha\%$ tests of $\sigma = \sigma_0$. In each case the function generates r random chi-square statistics, each based on a random sample of size n from the `model` distribution, and returns a histogram of the statistics (the acceptance region in blue; the rejection region in red) superimposed on the chi-square distribution with $(n-1)$ degrees of freedom (in gray). The proportion of statistics in the rejection region is reported.

A.2.6 Graphics for models and data

Procedures for plotting univariate and bivariate distributions, for comparing models with data, for plotting pairs lists, for constructing histograms and box plots, for constructing plots supporting least squares analyses, and for constructing a variety of diagnostic plots are included.

Plotting univariate distributions

The `ModelPlot` command can be used to visualize one or more univariate distributions, and the `OrderStatisticPlot` command can be used to compare a continuous distribution with the distribution of an order statistic, as described below.

`ModelPlot[model1,model2,...,modelk]` and
`ModelPlot[{model1,model2,...,modelk}]`
return a plot of k model distributions superimposed on the same set of axes. Models are distinguished by color.

`PlotColors[`k`]`
displays a legend showing the k colors in the plot.

`OrderStatisticPlot[model,`n`,`k`]`
returns a plot of the continuous `model` distribution (in gray) with the density function of the k^{th} order statistic from a random sample of size n superimposed (in blue). A vertical line is drawn at the value of the $(k/(n+1))^{\text{st}}$ quantile of the `model` distribution.

Note that the commands `ModelPlot[]` and `OrderStatisticPlot[]` return lists of the allowed models in each case.

Plotting bivariate distributions

The `ModelPlot3D` and `IndependentPlot` functions can be used to visualize bivariate distributions, as described below.

`ModelPlot3D[model]`
returns a plot of the bivariate `model`.

`IndependentPlot[model1,model2]`
returns a plot of the joint (X, Y) distribution, where X and Y are independent discrete random variables or independent continuous random variables, X has the `model1` distribution, and Y has the `model2` distribution.

Note that the commands `ModelPlot3D[]` and `IndependentPlot[]` return lists of the allowed models in each case.

Comparing models with data

The `ComparisonPlot` and `ProbabilityPlot` functions can be used to compare samples to theoretical models, as described below.

`ComparisonPlot[model,sample,options]`
returns a plot of the univariate `model` distribution (filled plot) with a histogram of the `sample` superimposed. Add the option `NumberIntervals`$\rightarrow n$ to use n subintervals of equal length in the histogram.

`ComparisonPlot[BivariateNormalDistribution[`ρ`],pairs]`
returns a contour plot of the standard bivariate normal density function with a scatter plot of `pairs` superimposed.

`ProbabilityPlot[model,sample]`
returns a probability plot of `sample` quantiles (vertical axis) against `model` quantiles (horizontal axis) of the continuous `model`. Add the option `SimulationBands`→`True` to add bands showing the minimum and maximum values for order statistics in 100 random samples of size `Length[sample]` from the `model` distribution.

Note that the commands `ComparisonPlot[]` and `ProbabilityPlot[]` return lists of the allowed models in each case.

Plotting pairs lists, and lists of triples

The `ScatterPlot` command is a multi-purpose function designed to produce single and multiple list plots, as described below.

`ScatterPlot[pairs1,pairs2,...,pairsk,options]` and
`ScatterPlot[{pairs1, pairs2, ..., pairsk},options]`
return a multiple list plot of k lists of pairs using `PlotColors[`k`]` to distinguish the samples.

`ScatterPlot[`$\{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \ldots\}$`,options]`
returns a list plot of $\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$ using color to distinguish the unique z-values.

`ScatterPlot[]`
returns a list of the allowed options.

The options include adding axes and plot labels, producing *line plots* (where successive pairs are joined by line segments), adding a least squares fit line and displaying the sample correlation (for a single list of pairs), and visualizing permutation and bootstrap methods (for a single list of pairs).

Constructing histograms and box plots

The `SamplePlot` command is used to construct an empirical histogram of sample data (or several histograms on the same set of axes), and the `BoxPlot` command is used to construct side-by-side box plots of sample data, as described below.

`SamplePlot[list1,list2,..,listk],options` and
`SamplePlot[{list1,list2,...,listk},options]`
return a plot of k empirical histograms superimposed on the same set of axes using `PlotColors[k]` to distinguish the samples. Labels may be added to the plot. Add the option `NumberIntervals`→n to use n subintervals of equal length in each histogram.

`BoxPlot[list1,list2,...,listk,options]` and
`BoxPlot[{list1,list2,...,listk},options]`
return a plot of k side-by-side box plots. Labels may be added to the plot.

Plots supporting least squares analyses

The `SmoothPlot` and `PartialPlot` functions support least squares analyses (Chapter 14). The `SmoothPlot` command returns a scatter plot of one or more lists of pairs with smoothed values superimposed, and the `PartialPlot` command returns a partial linear regression plot, as described below.

`SmoothPlot[{pairs1,s1},{pairs2,s2},...,{pairsk,sk},options]` and
`SmoothPlot[{{pairs1,s1},{pairs2,s2},...,{pairsk,sk}},options]`
return a multiple list plot of k pairs lists with k smooths superimposed. If `si` is a symbol representing a one variable function, then `si[x]` is superimposed. If `si` is a list of pairs, then a line plot (with successive pairs joined) is superimposed. The samples are distinguished using `PlotColors[`k`]` colors. Labels may be added to the plot.

`PartialPlot[cases,`i`]`
returns the partial regression plot of the residuals of the regression of y on all predictors except predictor i (vertical axis) against the residuals of the regression of predictor i on the remaining predictors (horizontal axis). Each element of the cases list must be of the form $\{x_1, x_2, \ldots, x_k, y\}$, and i must be an integer in the range $\{1, k\}$.

Empirical cumulative distribution function plot

The `ECDFPlot` command can be used to plot empirical CDFs of two samples, display the maximum difference in ECDFs (the value of the Smirnov statistic), and visualize permutation and bootstrap methods, as described below.

`ECDFPlot[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}$`,options]`
returns a plot of the empirical cumulative distribution functions of the x and y samples. The maximum difference in ECDFs, and results of random resampling and random partitioning may be added to the plot.

Plots supporting shift model analyses

The `QQPlot` and `SymmetryPlot` functions support shift model analyses in the two sample and paired sample settings, respectively, as described below.

`QQPlot[`$\{x_1, x_2, \ldots, x_n\}$`,`$\{y_1, y_2, \ldots, y_m\}$`,options]`
returns a quantile-quantile plot of y-sample quantiles (vertical axis) versus x-sample quantiles (horizontal axis). The HL estimate of the shift parameter, and results of random resampling and random partitioning may be added to the plot.

`SymmetryPlot[`$\{x_1 - y_1, x_2 - y_2, \ldots, x_n - y_n\}$`,options]` and
`SymmetryPlot[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_n, y_n\}\}$`,options]`
returns a symmetry plot of differences $d_i = x_i - y_i$ for $i = 1, 2, \ldots, n$. The HL estimate of the shift parameter, and results of random resampling and random assignments of signs to differences may be added to the plot.

A.2.7 Test statistics and parameter estimates

Functions supporting parametric, nonparametric, permutation, and locally weighted regression analyses are provided.

Pooled t and Welch t statistics

`PooledTStatistic[`$\{x_1, x_2, \ldots\}, \{y_1, y_2, \ldots\}$`,`$\delta_0$`]`
returns the pooled t statistic for the test of the null hypothesis $E(X) - E(Y) = \delta_0$.

`WelchTStatistic[`$\{x_1, x_2, \ldots\}, \{y_1, y_2, \ldots\}$`,`$\delta_0$`]`
returns the Welch t statistic for the test of the null hypothesis $E(X) - E(Y) = \delta_0$.

Sample correlation and rank correlation statistics

`Correlation[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_n\}$`]` and
`Correlation[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`]`
return the sample correlation coefficient.

`RankCorrelation[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_n\}$`]` and
`RankCorrelation[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`]`
return the value of Spearman's rank correlation coefficient.

Note that each function allows two types of input: either separate lists of x- and y- coordinates, or a list of (x, y) pairs.

Statistics for one or more samples

`RankSumStatistic[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_m\}$`]`
returns the value of Wilcoxon's rank sum statistic for the first sample, R_1.

`SignedRankStatistic[`$\{x_1 - y_1, x_2 - y_2, \ldots\}$`]` and
`SignedRankStatistic[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`]`
return the value of Wilcoxon's signed-rank statistic, W_+.

`SmirnovStatistic[`$\{x_1, x_2, \ldots\}, \{y_1, y_2, \ldots\}$`]`
returns the maximum absolute difference between the empirical cumulative distribution functions for the x and y lists.

`TrendStatistic[`$\{x_1, x_2, \ldots x_n\}$`]`
returns the value of Mann's trend statistic.

`KruskalWallisStatistic[{sample1,sample2,...,sampleI}]`
returns the value of the Kruskal-Wallis statistic for the list of samples, where each sample is a list of two or more numbers.

`FriedmanStatistic[{sample1,sample2,...,sampleI}]`
returns the value of the Friedman statistic, where the input is a matrix of real numbers. The number of samples, and the common number of observations in each sample, must be at least 2.

Shift parameter estimation

The `HodgesLehmannDelta` function can be used to compute the HL estimate of the shift parameter in two sample and paired sample analyses, as described below.

`HodgesLehmannDelta[sample1,sample2]`
returns the HL estimate of Δ=Median(X)−Median(Y) for independent samples.
 	
`HodgesLehmannDelta[`$\{x_1 - y_1, x_2 - y_2, \ldots\}$`]` and
`HodgesLehmannDelta[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`]`
return the HL estimate of Δ=Median($X - Y$) for paired samples.

Locally weighted regression estimation

The `LowessSmooth` function can be used to approximate the conditional expectation of Y given X for fixed values of X, as described below.

`LowessSmooth[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,`$p$`]`
returns a list of pairs of the form (unique x-value, smoothed y-value) using a lowess smooth with 100p% of the data included at each point.
 	
`LowessSmooth[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,`$p$`,`$x_0$`]`
returns the smoothed y-value when $x = x_0$.

The `LowessSmooth` function implements Cleveland's locally weighted regression method using tricube weights. The robustness step is omitted. The algorithm is described in the book by Chambers, Cleveland, Kleiner, and Tukey (Wadsworth International Group, 1983, page 121). (See Section 14.4.)

A.2.8 Methods for quantiles and proportions

Functions for computing sample quantiles and quantile confidence intervals using the methods discussed in Chapter 9, and for constructing confidence intervals for the probability of success in Bernoulli/binomial experiments are included. In each confidence interval procedure, if the confidence level option is omitted, then a 95% confidence interval is returned.

Estimating quantiles

`SampleQuantile[`$\{x_1, x_2, \ldots, x_n\}$`,`$p$`]`
returns the p^{th} sample quantile, where $1/(n+1) \leq p \leq n/(n+1)$.
 	
`SampleQuartiles[`$\{x_1, x_2, \ldots, x_n\}$`]`
returns the sample 25^{th}, 50^{th}, and 75^{th} percentiles, where $n \geq 3$.
 	
`SampleInterquartileRange[`$\{x_1, x_2, \ldots, x_n\}$`]`
returns the sample interquartile range, where $n \geq 3$.

`QuantileCI[`$\{x_1, x_2, \ldots, x_n\}$`,`$p$`,ConfidenceLevel`$\rightarrow 1-\alpha$`]`
returns a $100(1-\alpha)\%$ confidence interval for the p^{th} model quantile.

Estimating Bernoulli/binomial probability of success

`BinomialCI[`x`,`n`,ConfidenceLevel`$\rightarrow 1-\alpha$`]`
returns the $100(1-\alpha)\%$ confidence interval for the binomial success probability p based on inverting hypothesis tests, where x is the number of successes in n trials ($0 \le x \le n$). The maximum number of iterations used to search for the endpoints is set at 50; to increase this limit, add the option `MaxIterations`$\rightarrow$`m` (for some m).

A.2.9 Rank-based methods

Functions for (1) computing ranks and signed ranks, (2) conducting rank sum, signed rank, rank correlation, trend, Kruskal-Wallis, and Friedman tests, and (3) constructing confidence intervals for the shift parameter in two sample and paired sample settings are provided. Each test and confidence interval procedure takes ties into account. (Reference: Lehmann, Holden-Day, Inc., 1975.)

Computing ranks and signed ranks

`Ranks[`$\{x_1, x_2, \ldots\}$`]`
returns a list of ranks for the single sample.

`Ranks[`$\{\{x_1, x_2, \ldots\}, \{y_1, y_2, \ldots\}, \ldots\}$`]`
returns a list of lists of ranks for each sample in the combined sample.

`SignedRanks[`$\{x_1 - y_1, x_2 - y_2, \ldots\}$`]` and
`SignedRanks[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`]`
return the list of signed ranks for the sample of differences $\{x_1 - y_1, x_2 - y_2, \ldots\}$.

For example, if `samples` is the following list of two samples,

$$\{\{104.6, 82.5, 66.8, 113.4, 135.6, 122.4\}, \{93.6, 110.2, 74.8, 81.8\}\}$$

then `Ranks[samples]` returns $\{\{6, 4, 1, 8, 10, 9\}, \{5, 7, 2, 3\}\}$.

Rank sum and signed rank tests

`RankSumTest[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_m\}$`,options]`
returns the results of a one sided rank sum test using R_1 as test statistic.

`SignedRankTest[`$\{x_1 - y_1, x_2 - y_2, \ldots\}$`,options]` and
`SignedRankTest[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,options]`
returns the results of a one-sided Wilcoxon signed rank test using W_+ as test statistic.

In each case, p values are computed using the normal approximation to the sampling distribution of the test statistic; to use the exact sampling distribution, add the option `ExactDistribution→True`. The exact distribution should be used when sample sizes are small.

To conduct a two sided test, add the option `TwoSided→True`. For additional options, evaluate the commands `?RankSumTest` and `?SignedRankTest`.

Rank correlation test

`RankCorrelationTest[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_n\}$`,options]` and
`RankCorrelationTest[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,options]`
return the results of a one sided test of randomness using the Spearman rank correlation coefficient as test statistic.

P values are computed using the normal approximation to the sampling distribution of the rank correlation statistic. To conduct a two sided test, add the option `TwoSided→True`.

Trend test

`TrendTest[`$\{x_1, x_2, \ldots, x_n\}$`,options]`
returns the results of a one sided test of randomness using Mann's trend statistic.

P values for the trend test are computed using the normal approximation to the sampling distribution of the test statistic; to use the exact sampling distribution, add the option `ExactDistribution→True`. The exact distribution should be used when sample sizes are small.

To conduct a two sided test, add the option `TwoSided→True`.

Kruskal-Wallis and Friedman tests

`KruskalWallisTest[{sample1,sample2,...,sampleI}]`
returns results of a Kruskal-Wallis test for the list of samples, where each sample is a list of two or more numbers.

`FriedmanTest[{sample1,sample2,...,sampleI}]`
returns results of a Friedman test, where the input is a matrix of real numbers. The number of samples, and the common number of observations in each sample, must be at least 2.

In each case, p values are computed using the chi-square approximation to the sampling distribution of the test statistic.

Confidence interval procedures for shift parameters

`RankSumCI[`$\{x_1, x_2, \ldots, x_n\}, \{y_1, y_2, \ldots, y_m\}$`,ConfidenceLevel→` $1-\alpha$`]`
returns a $100(1-\alpha)\%$ confidence interval for Median$(X)-$Median(Y).

`SignedRankCI[`$\{x_1 - y_1, x_2 - y_2, \ldots\}$`,ConfidenceLevel→` $1-\alpha$`]` and `SignedRankCI[`$\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$`,ConfidenceLevel→` $1-\alpha$`]` returns a $100(1-\alpha)\%$ confidence interval for Median$(X - Y)$.

In each case, confidence intervals are computed using the normal approximation to the sampling distribution of the statistics; to use the exact sampling distribution, add the option `ExactDistribution→True`. The exact distribution should be used when sample sizes are small. If the confidence level option is omitted, then a 95% confidence interval is returned.

A.2.10 Permutation methods

Functions for computing random reorderings, and permutation confidence intervals for the slope in a simple linear model are provided. The confidence interval method is discussed in the book by Maritz (Chapman & Hall, 1995, page 120).

Random reordering of sample data

`RandomPartition[{sample1,sample2,...,samplek}]`
returns a random partition of the objects in
`Flatten[{sample1,sample2,...,samplek},1]`
into lists of lengths n_1=`Length[sample1]`, ..., n_k=`Length[samplek]`, respectively. The choice of each partition is equally likely.

`RandomPermutation[sample]` returns a random permutation of the objects in the sample list. The choice of each permutation is equally likely.

`RandomSign[`$\{x_1, x_2, \ldots\}$`]` returns a list of numbers whose k^{th} term is either $+x_k$ or $-x_k$. Each list is chosen with probability $1/2^m$, where m is the number of non-zero elements in $\{x_1, x_2, \ldots\}$.

For example, if `samples` is the following list of 3 lists:

$$\{\{a, b, c, d\}, \{e, f, g\}, \{h, i, j, k, \ell\}\}$$

then `RandomPartition[samples]` could return any one of the `Multinomial[4,3,5]` = 27720 partitions of the objects $a, b, \ldots, \ell$ into sublists of lengths 4, 3, 5. One possibility is

$$\{\{k, \ell, i, c\}, \{g, a, j\}, \{f, h, b, d, e\}\}.$$

Permutation confidence interval for slope

`SlopeCI[pairs, ConfidenceLevel→` $1-\alpha$`, RandomPermutations→` r`]` returns an approximate $100(1-\alpha)\%$ permutation confidence interval for the slope based on r random permutations, where pairs is a list of pairs of numbers.

For example, if

`pairs=`$\{\{19.53, -115.42\}, \{12.17, -106.79\}, \{13.75, -47.13\}, \{0.92, 36.09\}, \{11.71, 33.95\}, \{4.33, 24.06\}, \{5.74, 6.59\}, \{19.81, -93.56\}, \{3.06, 8.26\}, \{11.00, -29.24\}\}$;

is the list of data `pairs`, then the command `Fit[pairs,{1,x},x]` returns the linear least squares fit formula, $47.2251 - 7.40484\, x$, and the command

```
SlopeCI[pairs, RandomPermutations→2000]
```

returns an approximate 95% confidence interval for the slope based on 2000 random permutations. A possible result is $[-11.441, -3.09877]$.

A.2.11 Bootstrap methods

Functions for computing random resamples, summarizing bootstrap results, and constructing approximate bootstrap confidence intervals using Efron's improved procedure (the BC_a method) in the nonparametric case are provided. The confidence interval procedure is discussed in the book by Efron and Tibshirani (Chapman & Hall, 1993, page 184).

Random resample of sample data

`RandomResample[sample]` returns a random sample of size Length[sample] from the observed distribution of the sample data.

Note that a random resample is not the same as a random permutation of the sample data. In a random resample, observations are chosen *with* replacement from the original sample list.

Summarizing bootstrap estimation results

The `BootstrapSummary` command returns a histogram of estimated errors superimposed on the normal distribution with the same mean and standard deviation, estimates of bias and standard error, and simple approximate confidence intervals (Section 12.2), as described below.

`BootstrapSummary[results,observed,ConfidenceLevel→` $1-\alpha$`]` returns a summary of results from a bootstrap analysis, where observed is the observed value of the scalar parameter of interest. The summary includes simple $100(1-\alpha)\%$ confidence intervals for the parameter of interest.

The `BootstrapSummary` function is appropriate for summarizing the results of both parametric and nonparametric bootstrap analyses.

Improved bootstrap confidence intervals

`BootstrapCI1[sample,`f`, ConfidenceLevel`$\rightarrow 1-\alpha$`,`
`RandomResamples`$\rightarrow r$`]`
returns a $100(1-\alpha)\%$ bootstrap confidence interval for a parameter θ based on r random resamples from `sample`. The symbol f represents a real-valued function of one variable used to estimate θ from a sample list.

`BootstrapCI2[{sample1,sample2,...},`f`, ConfidenceLevel`$\rightarrow 1-\alpha$`,`
`RandomResamples`$\rightarrow r$`]`
returns a $100(1-\alpha)\%$ bootstrap confidence interval for a parameter θ based on r random resamples from each sample in the list. The symbol f represents a real-valued function of one variable (a list of sample lists) used to estimate θ.

In each case, if the confidence level option is omitted, then a 95% confidence interval is returned. If the random resamples option is omitted, then 1000 random resamples are used.

A.2.12 Analysis of variance methods

Functions for one-way layouts, blocked designs, and balanced two-way layouts, and for Bonferroni analyses are provided. (See Chapter 13.)

One-way layout

`AnovaOneWay[{sample1,sample2,...,sampleI}]`
returns analysis of variance results for the one-way layout, where each sample is a list of 2 or more numbers. The output includes an analysis of variance table and a line plot of estimated group means.

`BonferroniOneWay[{sample1,sample2,...,sampleI},`α`]`
returns results of a Bonferroni analysis for the one-way layout, where each sample is a list of 2 or more numbers. The overall significance level is at most α. The output includes the total number of comparisons, the rejection region for each test, the appropriate sampling distribution for the test statistics, and a list of significant mean differences.

Note that an estimated difference in means, say $\widehat{\mu_i} - \widehat{\mu_k}$, is included in the output list of the Bonferroni analysis if the two sided pooled t test of $\mu_i = \mu_k$ is rejected at the α/m level, where $m = \binom{I}{2}$.

Blocked design

`AnovaBlocked[{sample1,sample2,...,sampleI}]`
returns results of analysis of variance for the blocked design, where the input is a matrix of real numbers. The number of samples, and the common number of observations in each sample, must be at least 2. The output includes an analysis of variance table and a line plot of estimated group effects.

`BonferroniBlocked[{sample1,sample2,...,sampleI},`α`]` returns results of a Bonferroni analysis for the blocked design, where the input is a matrix of real numbers. The number of samples, and the common number of observations in each sample, must be at least 2. The overall significance level is at most α. The output includes the total number of comparisons, the rejection region for each test, the appropriate sampling distribution for the test statistics, and a list of significant mean differences.

Note that an estimated difference in means, say $\widehat{\mu_{i.}} - \widehat{\mu_{k.}}$, is included in the output list of the Bonferroni analysis if the two sided paired t test of $\mu_{i.} = \mu_{k.}$ is rejected at the α/m level, where $m = \binom{I}{2}$.

Balanced two-way layout

`AnovaTwoWay[{row1,row2,...,rowI}]` returns results of analysis of variance for the balanced two-way layout, where the input is a matrix of samples of equal lengths. The number of rows, the number of columns, and the common number of observations per sample must each be at least 2. The output includes an analysis of variance table and line plots of estimated group means by row, using `PlotColors[`I`]` to distinguish the plots.

A.2.13 Contingency table methods

Functions for analyzing I-by-J frequency tables, generating random tables for permutation analyses, analyzing the odds ratio in fourfold tables, Fisher's exact test, and McNemar's test are provided. (See Chapter 15.)

Analysis of two-way tables

The function `TwoWayTable` is a general routine for analyzing I-by-J frequency tables using Pearson's statistic, the Kruskal-Wallis statistic, or the Spearman rank correlation statistic, as described below.

`TwoWayTable[table]` returns results of an analysis of independence of row and column factors, or of homogeneity of row or column models, using Pearson's chi-square test. The output includes a table of standardized residuals.

`TwoWayTable[table,Method→MeanTable]` returns a table of estimated cell expectations for tests of independence of row and column factors, or of homogeneity of row or column models.

`TwoWayTable[table,Method→KruskalWallis]` returns results of a Kruskal-Wallis test of equality of row distributions.

`TwoWayTable[table,Method→RankCorrelation,options]` returns results of a one sided test of independence of row and column distributions; for a two sided test, add the option `TwoSided→True`.

Note that the chi-square approximation to the sampling distribution of Pearson's statistic is adequate when all estimated cell expectations are 5.0 or more. For tables with small cell expectations, a permutation analysis can be done.

Generating random tables

`RandomTable[table]` returns a random two-way frequency table with the same row and column totals as `table` based on the general permutation strategy.

Odds ratio analyses in fourfold tables

The `OddsRatioCI` function returns a large sample approximate confidence interval for the odds ratio based on the normal approximation to the sampling distribution of $\log(\widehat{\mathrm{OR}})$; use the `ExactMethod`→`True` option to construct an interval when sample sizes are small (Section 15.4). If the confidence level option is omitted, then a 95% confidence interval is returned. The small sample method is discussed in the book by Agresti (John Wiley & Sons, 1990, page 67).

`OddsRatioCI[`$\{\{x_{11}, x_{12}\}, \{x_{21}, x_{22}\}\}$`, ConfidenceLevel`→ $1-\alpha$`]` returns an approximate $100(1-\alpha)\%$ confidence interval for the odds ratio. All cell frequencies must be positive.

`OddsRatioCI[`$\{\{x_{11}, x_{12}\}, \{x_{21}, x_{22}\}\}$`,ConfidenceLevel`→ $1-\alpha$`,ExactMethod`→`True]` returns a $100(1-\alpha)\%$ confidence interval for the odds ratio based on inverting hypothesis tests. All cell frequencies must be positive. The maximum number of iterations used to search for the endpoints is set at 50; to increase this limit, add the option `MaxIterations`→`m`.

Tests for fourfold tables

`FisherExactTest[`$\{\{x_{11}, x_{12}\}, \{x_{21}, x_{22}\}\}$`]` returns results of the two sided Fisher exact test.

`McNemarTest[`$\{\{x_{11}, x_{12}\}, \{x_{21}, x_{22}\}\}$`]` returns exact and approximate results of McNemar's test for paired samples.

Index