

Statistical Methods for Programmers

Brent E. Edwards

February 1, 2018

Contents

1	Mathematical Fundamentals	1
1.1	Introduction	1
1.2	Intervals	1
1.3	Sets	3
1.3.1	Discrete and continuous	5
1.3.2	Implementing sets	5
1.4	Combinatorics	18
1.4.1	Summation and products	18
1.4.2	Product Rule	20
1.4.3	Factorial	21
1.4.4	Permutations	22
1.4.5	Combinations	24
1.5	One-dimensional summaries, finite version	26
1.5.1	sumX and sumXSquared	27
1.5.2	Measurements of Central Tendency	27
1.6	Measurements of dispersion	31
1.6.1	Percentiles, quartiles, inter-quartile range, and confidence interval	32
1.6.2	Order statistics	33
1.6.3	Variation	34
1.6.4	Less-used measurements	37
1.6.5	Other measurements	38
1.7	Calculus	40
1.7.1	Functors	40
1.7.2	Limits	41
1.7.3	Differentiation	42
1.7.4	Integration	47
2	Probability basics	53
2.1	Introduction	53
2.2	Relations between sample spaces and probability	53
2.2.1	Partitioning A Sample Space	53
2.2.2	Random Variables	54
2.2.3	Expected value	55

2.2.4	Naïve probability	56
2.2.5	Kolmogorov Axioms	59
2.2.6	Jensen's Inequality	63
2.3	Conditional Probabilities	65
2.3.1	Independent sets	66
2.3.2	The Monty Hall Problem	71
2.3.3	States and conditional probability	73
2.3.4	Bayes' Formula	75
2.3.5	Game Theory	77
2.4	Philosophy, Probability, and Information Theory	79
3	Distributions	81
3.1	Introduction	81
3.2	CDFs and PDFs	82
3.3	Distribution	84
3.4	Data distribution	85
3.5	Discrete Distributions	87
3.5.1	Implementation	87
3.5.2	Binomial distribution	92
3.5.3	Geometric distribution	99
3.5.4	Negative Binomial distribution	102
3.5.5	Hypergeometric	106
3.5.6	Poisson Distribution	111
3.6	Continuous Distributions	117
3.6.1	Implementation	117
3.6.2	The Exponential distribution	123
3.6.3	Uniform distribution	127
3.6.4	Normal Distribution	131
3.6.5	Gamma and its relations	137
3.6.6	The Beta distribution	144
3.7	Final Problems	146
4	Introduction to Statistical Tests	147
4.1	Introduction	147
4.2	Certainty	147
4.2.1	How to estimate	148
4.2.2	Accuracy and Precision	151
4.2.3	Null and Alternate Hypotheses	154
4.3	Outliers	158
4.4	Fitting equations	159
4.4.1	Least Squares	160
4.4.2	Maximum Likelihood	161

5 Statistical Tests for one variable	165
5.1 What statistical tests do	165
5.2 Bootstrapping	166
5.2.1 Estimating a distribution's mean using bootstrapping . .	167
5.2.2 The accuracy of bootstrapping	168
5.3 Estimating the mean of a normal distribution	169
5.3.1 The normal approximation	169
5.3.2 One-sided z-tests	170
5.3.3 Two-sided z-tests	173
5.3.4 Student's t-test	176
5.3.5 Wilcoxon Signed-Rank Test	178
5.4 Estimating the variance of a normal distribution	180
5.4.1 Normal means and variance	180
5.4.2 Fast estimates for the standard deviation of a normal distribution	182
5.4.3 Full-data estimates for the variance	182
5.5 Estimating parameters for other distributions	183
5.5.1 Verifying the distribution	183
5.5.2 Pearson's χ^2 -test (goodness of fit)	185
6 Correlation and Interpolation	189
6.1 Linear Regression	191
6.2 Distribution of linear regression	193
6.2.1 Distribution of the best-fit line	193
6.2.2 Point likelihood	194
6.3 Measuring the linear regression equation	194
6.3.1 covariance	194
6.3.2 The linear correlation coefficient	196
6.4 Abusing Regression and Correlation	199
6.4.1 Stock Market picks	199
6.4.2 Problem with the stock market pick method	202
7 Simple Classification	203
7.1 Supervised Learning	203
7.1.1 Example	203
7.1.2 Algorithm	208
7.1.3 Testing and splitting data	214
7.2 Unsupervised Learning	215
8 Statistical Tests for two variables	217
8.1 Pairs of variables	217
8.2 Comparing the mean of two samples	218
8.3 Two-sample Z- and T-test	218
8.3.1 Example: Go Handicaps	219
8.4 Pearson's chi-square test	220

9 Statistical Tests — Multiple	221
9.1 Multiple sets of variables	221
9.1.1 Fisher's Least Significant Difference	221
9.1.2 Bonferroni Correction	221
9.2 ANOVA	221
10 Sampling	223
10.1 Basic concepts in sampling	223
10.2 Monte Carlo Methods	226
10.3 Resampling	229
10.3.1 Resampling without parameters	229
10.3.2 Resampling with parameters	229
10.4 Markov Chain	229
10.4.1 Breaking Caesar ciphers	229
10.5 MCMC	229
10.5.1 Improving compression codes	229
11 Bayesian methods	237
11.1 Bayesian Inference	237
11.1.1 Priors and likelihoods	237
11.2 Hierarchical Models	237
12 Time Series	239
12.1 Time Series Decomposition	239
12.2 Time Series Regression	239
12.3 Nonseasonal Box-Jenkins Models	239
12.4 Seasonal Box-Jenkins Models	239
13 Markov Chain - Monte Carlo	241
13.1 EM Algorithm	241
13.1.1 EM Variance Estimation	241
13.2 Markov Chains	241
13.3 Monte Carlo Integration	241
13.4 Markov Chain - Monte Carlo	241
13.4.1 Metropolis-Hastings Algorithm	241
13.4.2 Gibbs Sampler	241
13.5 Bootstrapping	241
13.5.1 Bootstrap-t confidence interval	241
13.5.2 BCA Intervals	241
A Answers	243
A.1 Mathematical Fundamentals	244
A.2 Probability basics	267
A.3 Distributions	281
A.4 Introduction to Statistical Tests	293
A.5 Statistical Tests for one variable	296

A.6 Correlation and Interpolation	308
A.7 Simple Classification	311
A.8 Statistical Tests for two variables	312
A.9 Sampling	313

Chapter 1

Mathematical Fundamentals

1.1 Introduction

The topics in this chapter are:

1. Intervals
2. Sets
3. Summation, products, factorials, permutations, and combinations
4. Simple summaries of one-dimensional data
5. Measurements of dispersion
6. Differentiation and integration

This chapter should be easy. Even if you know all the topics in this chapter, please read this chapter to understand how mathematics and computer science interact in this book.

1.2 Intervals

An **interval** holds all numbers between two end points. Each end point can be either an **open end point** or a **closed end point**. A closed endpoint, which is bracketed by [or], includes the endpoint in the interval. An open endpoint, which is bracketed by (or), excludes the endpoint. An interval can be open on one end and closed in the other. In other words, if you have an interval between a and b , the interval can be of four different types with three names:

interval
open end point
closed end point

- A **closed interval** includes both endpoints. It is written $[a, b]$.
closed interval
- An **open interval** includes neither end point. It is written (a, b) .
open interval

- A **half-open interval** includes one end point but not the other end point. The kind of brackets show which end point is included and which is not. A half-open interval that includes the first end point but not the second is written $[a, b)$. A half-open interval in the other direction is written $(a, b]$.

Example 1.1 Intervals

- $(1, 3)$ represents the open interval between 1 and 3. It doesn't include 1 or 3.
 - $[-5, -2]$ represents the closed interval between -5 and -2. It includes both -5 and -2.
 - $(-3, 7]$ is a half-open interval between -3 and 7. It includes 7, but it does not include -3.
 - $[e, \pi)$ is a half-open interval between e (approximately 2.718) and π (approximately 3.1416). It includes e but it does not include π .
-

Example 1.2 Intervals in the real world

- All real-world measurements involve uncertainty: every measuring instrument has a maximum accuracy. When a person says that she is 175 cm tall, that usually means that she is between 174.5 and 175.5 cm tall.
 - Events are an example of intervals: all events have a starting time and an ending time.
-

Here's a header file for a class to hold an interval:

```
enum OpenOrClosed
{
    Open, Closed
};

class Interval
{
private:
    double m_start;
    OpenOrClosed m_oocStart;
    double m_end;
    OpenOrClosed m_oocEnd;

public:
    Interval( double start, OpenOrClosed oocStart,
              double end, OpenOrClosed oocEnd );
```

```

/* Single-value intervals. */
explicit Interval( double value);
Interval( const Interval& orig );
~Interval();
const Interval& operator =( const Interval& orig );

bool hasMember( double member ) const;
double getStartValue() const;
const OpenOrClosed getStartOpen() const;
double getEndValue() const;
const OpenOrClosed getEndOpen() const;

private:
    Interval();
};

```

The `explicit Interval(double dValue);` constructor handles single values. Given one double, d , it constructs the range $[d, d]$. The `explicit` keyword prevents you from accidentally writing “`Interval i = 3;`”; you must write it as “`Interval i = Interval(3);`”.

PROBLEMS: (Answers on page 244)

1. (Easy) Implement the `Interval` class. Make sure that it correctly handles error conditions. For example, what happens if you construct an interval like $[3, -2]$? What happens if you construct an interval like $[3, 3]$? How well does it work with infinities?
2. (Easy) What’s the difference between the interval $[0, 10)$ and the sequence $0, 1, 2, \dots, 9$?

1.3 Sets

A **set** is just a collection of items, any kind of items. We write a set with { and **set**} brackets surrounding the members of the set.

Example 1.3 Sets

- The set { Dewey, Cheatem, Howe } has three members.
 - The set {2, 3, {4, 5}} also has three members. The third member is a set containing two members.
-

The empty set or null set is the set of no members. It is written \emptyset or $\{\}$. Please note that the set $\{\emptyset\}$ is not the empty set. $\{\emptyset\}$ has one element (which is the empty set), so $\{\emptyset\}$ is not empty.

For this book, an **experiment** is anything that can be repeated, and that may produce different answers when it's repeated. Statistics is often interested in the results of experiments.

sample Every experiment results in exactly one **sample**. The word sample is a definition, not a restriction. A sample might represent anything, from *nothing happened* to *3.54 miles to seventeen cows painted blue, thirteen pigs dancing, three goats singing madrigals, and a confused ferret*. In C++ terms, the idea of a sample is similar to “all functions return one value, but that value may be `void` or may be a `struct` or `class` with multiple members”. When necessary, I use the word **measurement** for each member of a sample.

measurement

sample space The set of all possible samples is called the **sample space**.

Example 1.4 Three Marbles

Imagine that you have a bag of three marbles, with each marble numbered from one to three. You reach in, and gather any number of marbles — even zero or all three marbles. A marble can't appear twice within any sample. Then this experiment has eight possible samples in its sample space:

1. $\{1,2,3\}$
 2. $\{1,2\}$
 3. $\{1,3\}$
 4. $\{2,3\}$
 5. $\{1\}$
 6. $\{2\}$
 7. $\{3\}$
 8. $\{\} \text{ or } \emptyset$
-

Example 1.5 Real numbers

Most computers' random functions choose a random real number between 0 and 1. If computers had infinite precision, then the sample space would be infinitely large.

1.3.1 Discrete and continuous

discrete A sample space is **discrete** if one can count all the samples. Mathematically, a sample space is discrete if there is a one-to-one relation between the sample space and a subset of \mathcal{N} , the natural numbers. This book will tend to use the natural numbers to represent discrete sets.

continuous A sample space is **continuous** if it has more than one sample and for any two distinct samples in the space, there exists a third distinct sample between them.¹ The most famous continuous sample spaces are \mathcal{R} , the real number set.

This book assumes that measurements are numbers from a single axis. A discrete numeric set will almost always refer to the natural numbers (or a subset, or a closely-related set). A continuous numeric set will almost always refer to the real numbers (or a subset, or a closely-related set).

PROBLEMS: (Answers on page 244)

1. (Easy) Find an example in mathematics that is both discrete and continuous.
2. (Medium) Find an example in mathematics that is neither discrete nor continuous.

1.3.2 Implementing sets

Sets can be implemented in hundreds of correct ways. Some implementations are faster; some implementations use less memory; some are easier to understand. The code in this book is intended to be easy to understand.

We will often use finite sets: a set with a finite (usually small) number of elements. The following header will be filled in the next few sections:

```
template <typename T>
class FiniteSet
{
private:
    std::set<T> m_set;

public:
    FiniteSet<T> (const std::set<T>& oldSet);
    FiniteSet<T> (const FiniteSet<T>& oldFiniteSet);

    FiniteSet<T>& operator =(const std::set<T>& oldSet);
    FiniteSet<T>& operator =(const FiniteSet<T>& \
        oldFiniteSet);
```

¹Fixed-precision real numbers in a computer, like `double`, are not strictly continuous. They have a limit to their resolution; some pairs of numbers do not have a number between them. This book will usually ignore that problem.

```

~FiniteSet<T>();

bool isMember(const T& testMember) const;
FiniteSet<T> intersection(const FiniteSet<T>& rhs) \
const;
FiniteSet<T> join(const FiniteSet<T>& rhs) const;
bool subset(const FiniteSet<T>& rhs) const;
FiniteSet<T> difference(const FiniteSet<T>& rhs) const\ 
;
int size() const;

private:
    FiniteSet<T>() {}; // We don't want people to call \
    this method.
};

```

Membership

If a sample s is a member of a set E , then write $s \in E$. The C++ code for membership is:

```

template <typename T>
bool FiniteSet<T>::isMember(const T& testMember) const
{
    return m_set.find(testMember) != m_set.end();
}

```

Example 1.6 Membership Example: Real Numbers

Here's some examples when the sample space is the the real numbers:

- $0.5 \in \{0.3, 0.5, 0.7\}$
 - $0.5 \notin \{0.4, 0.6, 0.8\}$
 - $0.5 \in (0.4, 0.6]$
 - $0.5 \notin (0.3, 0.5)$
-

Example 1.7 Membership Example: Three Marbles

Here's some examples when the sample space is the three marbles:

- $\{1\} \in \{\{1\}, \{2, 3\}\}$
 - $\{2\} \notin \{\{1\}, \{2, 3\}\}$. The only members of the right-hand side are $\{1\}$ and $\{2, 3\}$. $\{2\} \neq \{1\}$ and $\{2\} \neq \{2, 3\}$.
-

PROBLEMS: (Answers on page 244)

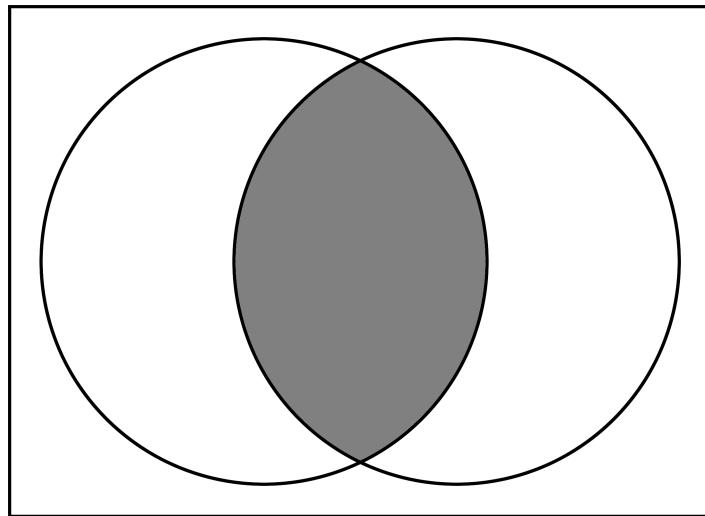
1. (Easy) Many books spend many pages asking whether for any sample E , it is true that $\emptyset \in E$. How does the code sidestep that question?

Intersections

You can build sets from other sets.

An **intersection** is a function. Its input is two or more sets. It returns the **intersection** members that appeared in *all* of the input sets.

Figure 1.1: Venn Diagram of Intersection



The `set_intersection` function in C++ implements this operation for two sets, as does the following code:

```

template <typename T>
FiniteSet<T> FiniteSet<T>::intersection(const FiniteSet<T>& rhs) const
{
    std::multiset<T> retval;
    typename std::multiset<T>::const_iterator iter;

    for (iter = rhs.m_set.begin(); iter != rhs.m_set.end();
          iter++)
        if (find(m_set.begin(), m_set.end(), *iter) != m_set.end())
            retval.insert(*iter);

    return FiniteSet<T>(retval);
}

```

The intersection between two sets is written ' \cap ', like $A \cap B$. The intersection between S_1, S_2, \dots, S_n is written $\bigcap_{i=1}^n S_i$ in the same way as the summation symbol on page 18.

disjoint If A and B are sets, and if $A \cap B = \emptyset$, then A and B are said to be **disjoint**.

Example 1.8 Intersections: Real numbers

Here's examples of intersections that use the real numbers experiment:

- $\{0.2, 0.8\} \cap \{0.2\} = \{0.2\}$
 - $\{0.3, 0.5, 0.7\} \cap \{0.4, 0.5, 0.6\} = \{0.5\}$
 - $\{0.1, 0.3, 0.5\} \cap \{0.4, 0.6, 0.8\} = \emptyset$
 - $\emptyset \cap \{0.2, 0.5\} = \emptyset$
 - $[0.3, 0.6] \cap (0.5, 0.8) = (0.5, 0.6]$
 - $\{0.3, 0.5, 0.7\} \cap [0.4, 0.8] = \{0.5, 0.7\}$
-

Example 1.9 Intersections: Three Marbles

Here's examples of intersections that use the marbles experiment.

- $\{\{1\}\} \cap \{\{2\}\} = \emptyset$
 - $\{\{1\}\} \cap \{\{1\}\} = \{\{1\}\}$
 - $\{\{1\}\} \cap \{\{1, 2\}\} = \emptyset$
 - $\{\{1\}, \{2\}\} \cap \{\{2\}, \{3\}\} = \{\{2\}\}$
 - $\{\{1\}, \{2\}, \{1, 2\}\} \cap \{\{1, 2\}, \{2\}, \{2, 3\}\} = \{\{2\}, \{1, 2\}\}$
-

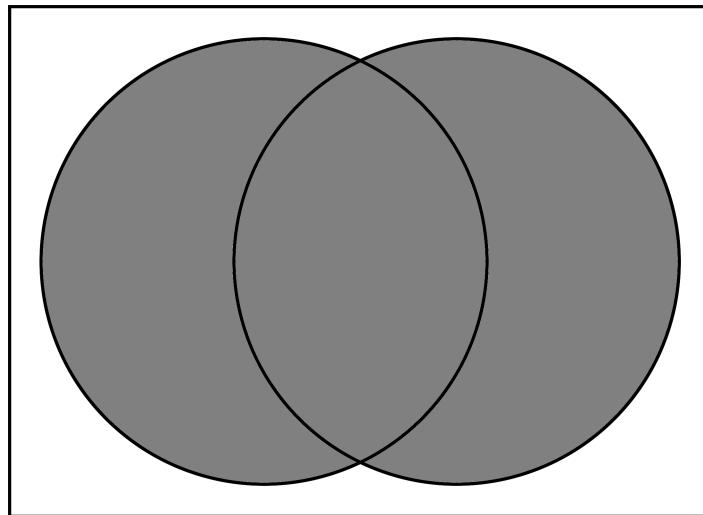
PROBLEMS: (Answers on page 244)

1. (Medium) If m is the number of members in the left-hand side and n is the number of members in the right-hand side, then the `FiniteSet<T>::intersection` code above takes $O(mn)$ time. However, C++ keeps the elements in a set sorted. Use this information to write a faster intersection routine that takes just $O(m + n)$ time.

Unions

A **union** is also a function. Its input is two or more sets. Its output set is all **union** members that appeared in *any* of the input sets.

Figure 1.2: Venn Diagram of Union



The `set_union` function in C++ implements this operation, as does the following code:

```
template <typename T>
FiniteSet<T> FiniteSet<T>::join(const FiniteSet<T>& rhs) \
const
{
    std::multiset <T> retval;
    typename std::multiset <T>::const_iterator iter;

    for (iter = m_set.begin(); iter != m_set.end(); iter++)
        retval.insert(*iter);
    for (iter = rhs.m_set.begin(); iter != rhs.m_set.end(); iter++)
        retval.insert(*iter);
    return retval;
}
```

```

    retval.insert(*iter);

    for (iter = rhs.m_set.begin(); iter != rhs.m_set.end()
        ); iter++)
        retval.insert(*iter);

    return retval;
}

```

Since `union` is a reserved word in C++, I cannot create a function with that name. Therefore, I named this function `join`.

Remember that sets can have at most one copy of any member. Therefore, if the same members were in both sets, it would appear only once in the union.

The union between two sets is written ' \cup ', like $A \cup B$. The union between S_1, S_2, \dots, S_n is written $\bigcup_{i=1}^n S_i$ in the same way as the summation symbol on page 18.

Example 1.10 Unions: Real Numbers

Here's examples of unions that use the real numbers experiment:

- $\{0.2\} \cup \{0.8\} = \{0.2, 0.8\}$
 - $\{0.3, 0.5\} \cup \{0.5, 0.7\} = \{0.3, 0.5, 0.7\}$, since 0.5 can only appear once.
 - $\emptyset \cup \{0.3, 0.6\} = \{0.3, 0.6\}$
 - $(0.3, 0.6) \cup [0.5, 0.7] = (0.3, 0.7]$
 - $\{0.5, 0.7\} \cup [0.4, 0.8] = [0.4, 0.8]$
-

Example 1.11 Unions: Three Marbles

Here's examples of unions that use the three marbles experiment:

- $\{\{1\}\} \cup \{\{2\}\} = \{\{1\}, \{2\}\}$
 - $\{\{1\}\} \cup \{\{1\}\} = \{\{1\}\}$, since it can have at most one copy of any member.
 - $\{\{1\}\} \cup \{\{1, 2\}\} = \{\{1\}, \{1, 2\}\}$, since $\{1\}$ and $\{1, 2\}$ are different members.
 - $\{\{1\}, \{2\}\} \cup \{\{2\}, \{3\}\} = \{\{1\}, \{2\}, \{3\}\}$, since $\{2\}$ can appear only once.
 - $\{\{1\}, \{2\}, \{1, 2\}\} \cup \{\{2\}, \{2, 3\}\} = \{\{1\}, \{2\}, \{1, 2\}, \{2, 3\}\}$
-

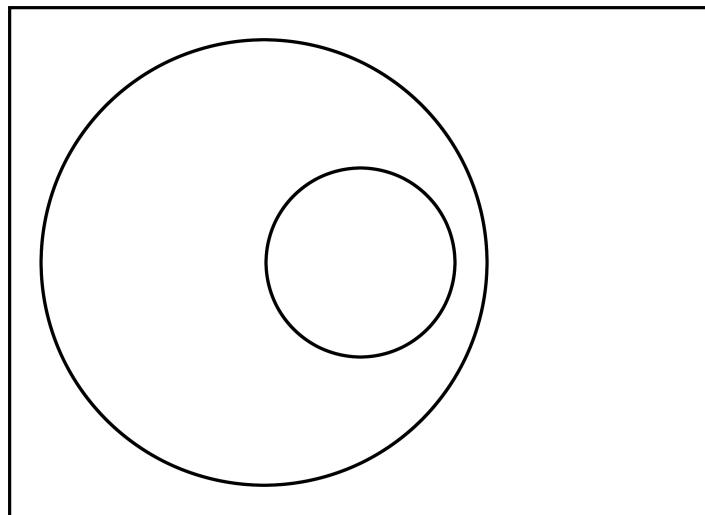
PROBLEMS: (Answers on page 245)

1. (Easy) Let the sample space have three green balls and seven red balls. You choose one set of balls, s_1 , put them back, then choose a second set of balls, s_2 . s_1 has one green ball and three red balls. s_2 has two green balls and four red balls. What other information do you need before you can answer how many red or green balls $s_1 \cup s_2$ has?

Subset

A **subset** is a function. Its input is two sets. Its output is whether all members **subset** in the first set occur in the second set.

Figure 1.3: Venn Diagram of Subset



Implementing it is easy:

```
template <typename T>
bool FiniteSet<T>::subset(const FiniteSet<T>& rhs) const
{
    typename std::multiset<T>::const_iterator iter;

    for (iter = m_set.begin(); iter != m_set.end(); iter++)
        if (find(rhs.m_set.begin(), rhs.m_set.end(), *iter)
            == rhs.m_set.end())
}
```

```

    return false;

    return true;
}

```

“ A is a subset of B ” is written $A \subseteq B$. “ A is not a subset of B ” is written $A \not\subseteq B$.

Example 1.12 Subsets: Three Marbles

Here’s examples of subsets that use the three marbles example:

- $\{\{1\}\} \subseteq \{\{1\}, \{2\}\}$
- $\{\{1\}, \{2\}\} \subseteq \{\{1\}, \{2\}\}$
- $\{\{1\}, \{2\}, \{3\}\} \not\subseteq \{\{1\}, \{2\}\}$ since $\{3\}$ appears nowhere on the right side.
- $\{\{1\}, \{2\}\} \not\subseteq \{\{1, 2\}\}$ since $\{1\}$ appears nowhere on the right side. Remember that $\{1\} \neq \{1, 2\}$.
- $\emptyset \subseteq \{\{1\}, \{1, 2\}\}$

Example 1.13 Subsets: Real Numbers

- $\{0.4, 0.6\} \subseteq (0.3, 0.7)$
- $\{0.4, 0.6\} \not\subseteq (0.4, 0.7)$ since $0.4 \notin (0.4, 0.7)$
- $(0.5, 0.6) \subseteq [0.5, 0.7]$

In set theory, two sets, A and B , are equal if $A \subseteq B$ and $B \subseteq A$. If $A \subseteq B$ but $B \not\subseteq A$, then A is a proper subset of B . This relation can be written $A \subset B$.

Sidebar 1.1: Equality and proper subsets

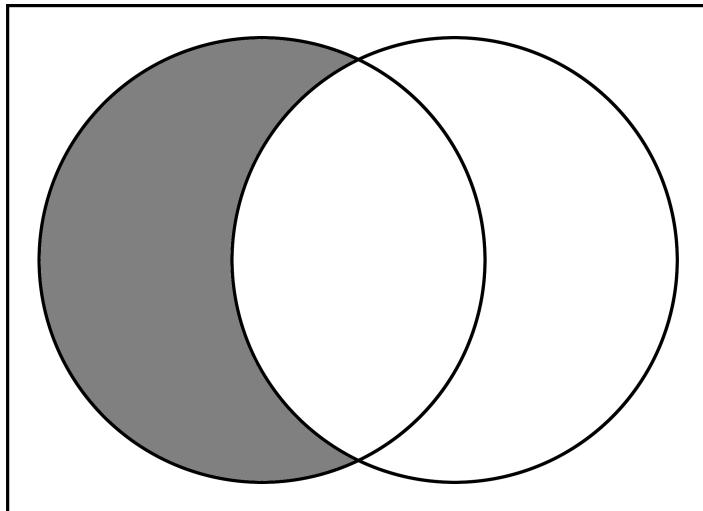
PROBLEMS: (Answers on page 246)

1. (Easy) Let A and B be sets. Which statements below are always true, always false, or sometimes true and sometimes false... and why?
 - (a) $A \subseteq A$
 - (b) $(A \cap B) \subseteq A$
 - (c) $(A \cup B) \subseteq A$
 - (d) $\emptyset \subseteq A$
 - (e) $A \subseteq \emptyset$
 - (f) $A \subseteq (A \cap B)$
 - (g) $A \subseteq (A \cup B)$
2. (Easy) The subset function can be written in terms of the intersection function and equality. Write it.

Set difference

The final important function among sets is the set difference function. Its input is two sets. Its output is all members that are in the first sample, but not in the second one.

Figure 1.4: Venn Diagram of Set Difference



The `set_difference` function in C++ implements this operation, as does the following code:

```

template <typename T>
FiniteSet<T> FiniteSet<T>::difference(const FiniteSet<T>
    >& rhs) const
{
    std::multiset <T> retval;
    typename std::multiset <T>::const_iterator iter;

    for (iter = m_set.begin(); iter != m_set.end(); iter++)
        if (find(rhs.m_set.begin(), rhs.m_set.end(), *iter)
            == rhs.m_set.end())
            retval.insert(*iter);

    return FiniteSet<T>(retval);
}

```

The set difference function will be written \setminus for this book.²

Example 1.14 Set Differences: Real Numbers

Here's examples of set differences that use the real numbers experiment:

- $\{0.2, 0.8\} \setminus \{0.2\} = \{0.8\}$
 - $\{0.3, 0.5, 0.7\} \setminus \{0.4, 0.5, 0.6\} = \{0.3, 0.7\}$, since neither 0.4 nor 0.6 occur in the left-hand side.
 - $(0.4, 0.6] \setminus [0.5, 0.7) = (0.4, 0.5)$
-

Example 1.15 Set Differences: Marbles

Here's examples of set differences that use the marbles experiment:

- $\{\{1\}\} \setminus \{\{2\}\} = \{\{1\}\}$. The $\{\{2\}\}$ doesn't appear in the first sample, so it does nothing.
 - $\{\{1\}\} \setminus \{\{1\}\} = \emptyset$.
 - $\{\{1\}\} \setminus \{\{1\}, \{1, 2\}\} = \emptyset$
 - $\{\{1\}, \{2\}, \{1, 2\}\} \setminus \{\{1, 2\}, \{2\}, \{2, 3\}\} = \{\{1\}\}$
 - $\{\{1\}, \{2\}\} \setminus \{\{2\}, \{3\}\} = \{\{1\}\}$
-

²Many other books use $-$, the minus sign, for set difference. I find that sign confusing: Is $(-1, 4) - 1$ equal to $(-1, 1) \cup (1, 4)$ due to set difference or is it equal to $(-2, 3)$ due to subtracting one from every element in $(-1, 4)$?

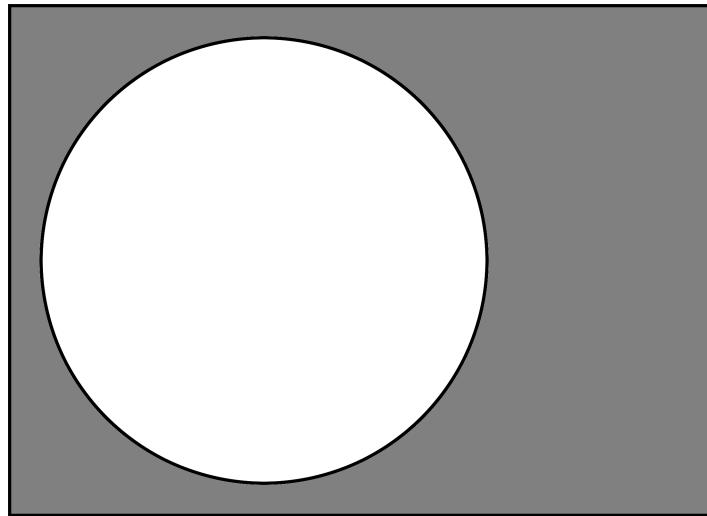
PROBLEMS: (Answers on page 246)

1. (Medium) If m is the number of members in the left-hand side and n is the number of members in the right-hand side, then the `FiniteSet<T>::difference` code above takes $O(mn)$ time. However, C++ keeps the elements in a set sorted. Use this information to write a faster difference routine that takes just $O(m + n)$ time.

Set Negation

Some books list the **set negation** function, instead of set difference.³ If the **set negation** sample space is U (for *universe*), then the set negation for a set S is $U \setminus S$, and it's written as S^c in this book.⁴ The letter c stands for the word **complement**. **complement**

Figure 1.5: Venn Diagram of Set Negation



³In the last section, I said that the set difference was the last important function. I have not forgotten that sentence.

⁴Many other books use \bar{S} to represent set negation. This book uses \bar{S} to represent the mean of a set S .

Example 1.16 Set Negation

- In the three marbles example, $\{\{1\}, \{2\}, \{3\}\}^c = \{\emptyset, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
 - In the real numbers example, $[-1, 1]^c = (-\infty, -1) \cup [1, \infty)$
-

The nice thing about using set negation instead of set difference is that set arithmetic is similar to boolean arithmetic. Table 1.1 on page 16 summarizes the similarities between the two groups.

Boolean Arithmetic	Set Arithmetic
True or false value	Presence or Absence of a member
Logical or ($\ $)	Union (\cup)
Logical and ($\&\&$)	Intersection (\cap)
Logical not ($!$)	Set Negation (S^c)

Table 1.1: Comparison of boolean arithmetic and set arithmetic

In particular, an analog of DeMorgan's Law works for these operators:

$$\begin{aligned} ! (B_1 \&\& B_2) &= (\ ! B_1) \| (\ ! B_2) & (S_1 \cup S_2)^c &= S_1^c \cap S_2^c \\ ! (B_1 \| B_2) &= (\ ! B_1) \&\& (\ ! B_2) & (S_1 \cap S_2)^c &= S_1^c \cup S_2^c \end{aligned}$$

On the other hand, the most general way to implement set negation in computer science uses set difference. In fact, the source code for a set negation function would be identical to a set difference function, except that it might refer to a global variable instead of a left-hand side.

Cardinality

cardinality The **cardinality** of a set S is written as $|S|$. It is the number of elements in the top level of a set.

Because cardinality only counts the number of elements in the top level, $|\{\{1, 2\}, \{3, 4\}\}|$ has only two elements, not four. On the other hand, $|(3, 5)|$ does not have a well-defined cardinality.

Among C++ STL classes, the `size()` method returns the cardinality of a set.

```
template <typename T>
FiniteSet<T> FiniteSet<T>::size() const
{
    return m_set.size();
}
```

PROBLEMS: (Answers on page 247)

1. (Easy) set_negation can be written in terms of set_difference and the sample space. Given the functions union, intersection, set_negation and the sample space, write the set_difference function.
 2. (Medium) If you know the universe that the sets will be constrained to, you can create another implementation of sets. A bitset is a number whose binary representation determines whether an element is in the set. For example, if the bitset had four members, then 0110_2 would be a bitset with the second and third member. Create a bitset with as many elements as members, and where presence of a member is indicated by a true in the bitset. Write code that fulfills the following header.

```
template <typename T>
class Bitset_FiniteSet
{
private:
    static std::set<T> ms_setUniverse;
    long m_set; // If your compiler has it, use long \
long.

public:
    Bitset_FiniteSet<T>(&const std::set<T>& oldSet);
    Bitset_FiniteSet<T>(&const Bitset_FiniteSet<T>& \
oldFiniteSet);

    Bitset_FiniteSet<T>& operator =(const std::set<T>& \
oldSet);
    Bitset_FiniteSet<T>& operator =(const \
Bitset_FiniteSet<T>& oldFiniteSet);

    ~Bitset_FiniteSet<T>();

    bool isMember(const T& testMember) const;
    Bitset_FiniteSet<T> intersection(const \
        Bitset_FiniteSet<T>& rhs) const;
    Bitset_FiniteSet<T> join(const Bitset_FiniteSet<T> \
        & rhs) const;
    bool subset(const Bitset_FiniteSet<T>& rhs) const;
    Bitset_FiniteSet<T> difference(const \
        Bitset_FiniteSet<T>& rhs) const;
    int size() const;

private:
    Bitset_FiniteSet<T>() {};// We don't want \
people to call this method.
```

} ;

Feel free to add public methods or private functions as you see fit.

1.4 Combinatorics

Combinatorics is the mathematics of counting. Every question in combinatorics asks, “How many...?”

1.4.1 Summation and products

summations Mathematicians use many **summations** and **products**. Often, these summations and products use many terms. Mathematicians have a convenient way to write them.

Example 1.17 Summation and products, part one

The phrase “A function of a sequence of natural numbers” may not be clear. These examples may make the idea clearer:

infinite sum

- The **infinite sum** $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$ can be written as $\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \dots$. The natural numbers appear in the denominator of the terms, and everything else remains the same.
 - The infinite sum $-1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4}$ has an alternating sign. The series $(-1^1, -1^2, -1^3, -1^4, \dots)$ is the same as $(-1, 1, -1, 1, \dots)$. Therefore, we can write the original infinite sum with $(-1^1 \times 1) + (-1^2 \times \frac{1}{2}) + (-1^3 \times \frac{1}{3}) + \dots$.
-

summation sign

Mathematicians write a sum with the **summation sign**, \sum . They use this symbol in two different ways:

- If the summation is a function of a sequence of natural numbers, then a mathematician writes the following:

$$\sum_{i=0}^{20} f(i)$$

The i is an arbitrary variable name. It’s defined below the summation sign, along with its initial value. The final, inclusive value is written above the summation sign. The variable increments by 1 with every iteration.

- If the summation is based on a previously-defined set, then a mathematician writes the following:

$$\sum_{s \in S} f(s)$$

The variable s takes on all values of the set S .

Example 1.18 Summation and products, part two

- The sum $1 + 2 + 3 + \dots + 100$ can be written as $\sum_{i=1}^{100} i$.
 - If $S = \{1, 2, 3, 5, 8\}$, then $\sum_{s \in S} s^2$ is $1^2 + 2^2 + 3^2 + 5^2 + 8^2$.
 - The infinite sum $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$ can be written as $\sum_{i=0}^{\infty} \frac{1}{2^i}$. The symbol ∞ means that the sum never stops.
 - The infinite sum $1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \dots$ can be written as $\sum_{i=1}^{\infty} (-1)^i \times \frac{1}{i}$.
 - Let S be the always-doubling, infinite set of numbers $\{1, 2, 4, 8, 16, 32, \dots\}$. Then the phrase $\sum_{s \in S} \frac{1}{s}$ is the same as $\sum_{i=0}^{\infty} \frac{1}{2^i}$
-

The first kind of summation acts like a for-loop. The expression

$$\sum_{i=a}^b f(i)$$

is exactly equal to the code

```
result = 0;
for (i=a; i<=b; i++)
    result += f(i);
```

The second kind of summation acts like an iterator. The expression

$$\sum_{s \in S} f(s)$$

is exactly equal to the code

```
result = 0;
for (s = S.begin(); s != S.end(); ++s)
    result += f(*s);
```

Not all infinite series have a proper sum. However, the summations in this book usually converge to some value.

Mathematicians also have a symbol for products, \prod . Instead of summing its values, it returns the product of them.

PROBLEMS: (Answers on page 247)

1. (Easy) Write the code that represents $\prod_{i=a}^b f(i)$
2. (Medium) The sum $\sum_{i=0}^{\infty} \frac{1}{2^i}$ has shown up several times. What is its value? Prove that your answer is correct.

1.4.2 Product Rule

Given two sets, $A = \{a_1, a_2, a_3, \dots, a_m\}$ and $B = \{b_1, b_2, b_3, \dots, b_n\}$, where A has m elements and B has n elements, how many pairs can be made with one element taken from A and one element taken from B ?

- Choose a_1 . Then n elements in B can be associated with a_1 .
- Choose a_2 . Then n elements in B can be associated with a_2 .
- ...
- Choose a_m . Then n elements in B can be associated with a_m .

Therefore, there are $m \times n$ possible pairs with one element taken from A and one element taken from B .

A visualization of this answer is in figure 1.6.

Figure 1.6: products

$$\begin{array}{cccccc} a_1b_1 & a_2b_1 & a_3b_1 & \dots & a_mb_1 \\ a_1b_2 & a_2b_2 & a_3b_2 & \dots & a_mb_2 \\ a_1b_3 & a_2b_3 & a_3b_3 & \dots & a_mb_3 \\ \dots & \dots & \dots & \dots & \dots \\ a_1b_n & a_2b_n & a_3b_n & \dots & a_mb_n \end{array}$$

product rule The final set has $m \times n$ elements. This idea is the **product rule**.

PROBLEMS: (Answers on page 248)

1. (Easy) Expand the product rule. Given c sets, $A_1 = \{a_{11}, a_{12}, \dots, a_{1n_1}\}$, $A_2 = \{a_{21}, a_{22}, \dots, a_{1n_2}\}$, ..., $A_c = \{a_{c1}, a_{c2}, \dots, a_{cn_c}\}$, how many arrays can be made with one element from each set?

1.4.3 Factorial

factorial The **factorial** is a simple function. If you have n objects, then the factorial of n represents the number of ways that all of these objects can be ordered.

The factorial of n is the product of all numbers from 1 to n , inclusive.

The following program also defines it:

```
double factorial(int n)
{
    double result;
    int i;

    if (n < 0)
        throw std::out_of_range("Cannot compute a negative\
                                factorial.");

    result = 1;
    for (i=2; i<=n; i++)
        result *= i;

    return result;
}
```

The factorial of n is written $n!$.

Example 1.19 Examples of factorials

- If you have three objects, a , b , and c , then they can be ordered in six ways: (a, b, c) , (a, c, b) , (b, a, c) , (b, c, a) , (c, a, b) , and (c, b, a) . Therefore, the factorial of 3 is 6. According to the formula, the factorial of 3 (written $3!$) is $1 \times 2 \times 3$, or 6.
 - If you have five objects, a , b , c , d , and e , then they can be written in $5!$ different ways: $1 \times 2 \times 3 \times 4 \times 5$, or 120.
 - If you have 20 objects: a, b, c, \dots, t , then the factorial of 20 (written $20!$) is $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10 \times 11 \times 12 \times 13 \times 14 \times 15 \times 16 \times 17 \times 18 \times 19 \times 20$, or 2,432,902,008,176,640,000. Listing all of these combinations is an exercise for the reader.
-

PROBLEMS: (Answers on page 248)

1. (Easy) Every morning, I get dressed, walk the dog, read the newspaper, eat breakfast, brush my teeth, and shave. It doesn't matter in which order I do these six things. How many different morning routines could I have?
2. (Easy) Use product notation to write $12!$
3. (Medium) Most textbooks that teach recursion use factorial as an example. You can define $n!$ as $n \times (n - 1)!$ (when $n > 1$) and as 1 (when $n = 0$ or $n = 1$). Why is this program worse than the one on page 21?
4. (Medium) This chapter gives two definitions for the factorial of n : the number of ways that n items can be ordered, and the product of all numbers from 1 to n . Prove that, for all numbers greater than zero, the two definitions give the same result.

1.4.4 Permutations

"Choose five letters of the alphabet." What does this sentence mean? Are you allowed to pick "AAAAA"? Is "UNITE" the same as "UNTIE"? Those two questions divide the meaning of "choose five letters from the alphabet" into four categories. Two of these four categories are more important for statistics.

Choosing *without replacement* means that once an item is chosen, it can never be chosen again. If you choose five letters from the alphabet without replacement, then you could not pick "AAAAA" or "ABCDA".

Order does matter means that items in a different order are considered different from each other. If you choose five letters, but order does matter, then choosing "UNITE" would be different than choosing "UNTIE".

Permutations happen when you choose items without replacement and when order matters. The `permutations` function counts the number of ways to choose r items from a population of size n without replacement and when order matters. In mathematics, the `permutations` function is written $P(n, r)$, and it is computed with the function

$$P(n, r) = n \times (n - 1) \times \cdots \times (n - r + 1) = \frac{n!}{(n - r)!}$$

or by the following code:

```
double permutations(int n, int r)
{
    int i;
    double permute;

    if (n < 0 || r < 0 || n < r)
        throw std::out_of_range("Illegal_permutation.");
}
```

```

    permute = 1.0;
    for (i = n-r+1; i <= n; i++)
        permute *= i;

    return permute;
}

```

Example 1.20 Permutation Example

I have a collection of 30 favorite hour-long albums. How many five-hour series can I create without repeating an album? According to the formula, I can create $\frac{30!}{25!} = 30 \times 29 \times 28 \times 27 \times 26$ different concerts, or 17,100,720 different series.

Sometimes, instead of counting the number of ways that a permutation happens, you need to create a random permutation of all members of a set. This permutation can be done with the C++ function `random_shuffle`.

PROBLEMS: (Answers on page 249)

1. (Easy) California's *Fantasy 5* Lottery draws 5 numbers, each from 1 to 39, without replacement, that you need to get in order. What is the chance that you will choose all 5?
2. (Easy) The tarot has 78 cards, and a reading chooses ten cards. Order matters in a reading. How many possible readings are there?
3. (Easy) Write $P(12, 5)$ using the product notation of section 1.4.1 on page 18
4. (Medium) The `permutations` code could have been written as:

```

double permutations(int n, int r)
{
    if (n <= 0 || r < 0 || n < r)
        throw std::out_of_range("Illegal_permutation.");
    }

    return factorial(n) / factorial(n - r);
}

```

Why is the code in the text better?

5. (Medium) Write an efficient function to choose a random permutation of all elements of a set, without using `random_shuffle`. Use the following header:

```
template <typename T>
const std::vector<T> random_permutation(const std::vector<T>& source);
```

6. (Hard) Write an efficient function to create all permutations of all members of a set. Since the number of possible permutations grows exponentially, the function should accept two parameters: the source, and which number permutation should be returned. Use the following header:

```
template <typename T>
const std::vector<T> all_permutations(const std::vector<T>& source, long int permutation);
```

1.4.5 Combinations

Combinations happen when you choose items without replacement and when order does not matter.

The combinations function counts the number of choices when you have n items, you choose r of them without replacement, and order does not matter. The combinations function is written $C(n, r)$ or $\binom{n}{r}$.

The combinations function is computed with the function

$$\binom{n}{r} = \frac{n(n-1)\cdots(n-r+1)}{r(r-1)\cdots 1} = \frac{n!}{r!(n-r)!}$$

or by the following code:

```
double combinations(int n, int r)
{
    if (n <= 0 || r < 0 || n < r)
        return 0;
    if (2 * r <= n)
        return permutations(n, r) / factorial(r);
    else
        return permutations(n, n-r) / factorial(n-r);
}
```

Example 1.21 Counting combinations

A Chinese restaurant has a special: choose any three items in their ten-item menu for \$20. How many different combinations are there? $\text{combinations}(10, 3)$ is $\frac{10!}{3! \times 7!} = 120$.

Choosing random combinations

Computer science is different than mathematics. Mathematics assumes a perfect world; computer science has small (but often significant) assumptions that break the perfect world.

Assume that you have n elements, and that you choose r elements. We would like each of the $\binom{N}{r}$ possible choices to be equally likely. This condition is called an **unrestricted sample**. Anything that does not have this condition is biased.

unrestricted sample

That condition sounds extremely simple. In the world of computers, it is often difficult to meet, due to the pseudo-random numbers that computers generate.

Imagine that a card game must shuffle a standard 52-card deck. The best, standard method to choose random numbers uses `drand48`. Could the shuffle be unrestricted?

`drand48` has 48 bits in its seed, and every random number uses one seed, so it allows at most 2^{48} different states without repetition. There are $52!$ ways to shuffle a deck. But:

$$\begin{aligned} 52! &= 52 \times 51 \times \cdots \times 2 \times 1 \\ &> 2^{51} \\ &> 2^{48} \end{aligned}$$

There are more ways to shuffle a deck of cards than there are states in `drand48`. No matter how code uses `drand48`, some orderings of the cards cannot be reached. Therefore, the sample is biased!

This book rarely comments about this problem. However, in production systems, you may want to investigate libraries with larger seeds or other sources of random numbers.

PROBLEMS: (Answers on page 252)

1. (Very Easy) On the island Sennomo in the Pacific, ten languages are common: Arabic, English, Esperanto, French, Korean, Mandarin, Japanese, Spanish, Tagalog, and Vietnamese. How many different ways could someone be trilingual?
2. (Medium) The text doesn't cover the combination of choosing items *with replacement* and when order matters. What is the mathematical formula for the number of ways one could select r items from a population of n items with replacement and when order matters?
3. (Medium) The code uses a few tricks that make for good problems:
 - (a) Prove that $\binom{n}{r} = \frac{P(n,r)}{r!}$.

- (b) Prove that $\frac{P(n,r)}{r!} = \frac{P(n,n-r)}{(n-r)!}$
4. (Hard) Write an efficient function to compute all combinations of a vector.
Use the header

```
template <typename T>
const std::vector< T > all_combinations(const std::vector<T>& source,
                                         long int combination)
```

where `combination` is the number of the combination.

1.5 One-dimensional summaries, finite version

statistic Statistics summarizes data. In fact, a **statistic** is defined as a computation on data.

The next two sections will implement the following header for a class called `Data`:

```
template <typename T>
class Data : public virtual FiniteSet<T>
{
public:
    Data<T>(const std::multiset<T>& oldSet);
    Data<T>(const FiniteSet<T>& oldSet);
    Data<T>(const Data<T>& oldSet);

    T sumX() const;
    T sumXSquared() const;
    int size() const;

    T mean() const;
    T median() const;
    T mode() const;
    T variance() const;
    T standard_deviation() const;
    T percentile(double percent) const;
    T order(int n) const;

private:
    Data<T>() { }
};
```

1.5.1 sumX and sumXSquared

If we write the data as x_1, x_2, \dots, x_n , then `sumX()` computes $\sum_{i=1}^n x_i$ and `sumXSquared()` computes $\sum_{i=1}^n x_i^2$.

```
template <typename T>
T Data<T>::sumX() const
{
    typename std::multiset<T>::const_iterator iter;
    T total;

    total = 0;
    for (iter = FiniteSet<T>::m_set.begin(); iter != \
          FiniteSet<T>::m_set.end(); iter++)
        total += *iter;

    return total;
}

template <typename T>
T Data<T>::sumXSquared() const
{
    typename std::multiset<T>::const_iterator iter;
    T total;

    total = 0;
    for (iter = FiniteSet<T>::m_set.begin(); iter != \
          FiniteSet<T>::m_set.end(); iter++)
        total += *iter * *iter;

    return total;
}

template <typename T>
int Data<T>::size() const
{
    return FiniteSet<T>::m_set.size();
}
```

1.5.2 Measurements of Central Tendency

The **central tendency** is a fancy term for “middle”. There are many measurements for the central tendency; the three most common are the mode, median, and mean.

Mode

mode A **mode** is a most frequent item in the data. A mode may not be unique; data may have many modes.

Modes have several features:

- The data doesn't need to have an order for it to have a mode. For example, if data held favorite flavors of ice cream, one could take the mode of this data.
- Modes are not useful for sets. Sets may have only one copy of any element. Therefore, if an element is in a set, then it's a mode of the set. However, modes are useful in multisets, which can have multiple copies of an element.

```
template <class T>
T Data<T>::mode() const
{
    typename std::multiset<T>::size_type count;
    typename std::multiset<T>::const_iterator iter;
    typename std::multiset<T>::size_type max_count;
    T max_elem;

    max_count = 0;
    for (iter = FiniteSet<T>::m_set.begin(); iter != \
        FiniteSet<T>::m_set.end(); iter++) {
        count = FiniteSet<T>::m_set.count(*iter);

        if (count > max_count) {
            max_count = count;
            max_elem = *iter;
        }
    }

    return max_elem;
}
```

Median

median A **median** is an element at which half or more of the data is below or equal to it, and half or more of the data is above or equal to it. A median might not be unique, or even part of the dataset. For example, if the data were 1,3,5,7,9,11, then 6 would be a median.

To have a median, data needs a complete ordering: every pair of elements must be comparable so that either one is less than the other, or they are equal.

Because C++'s sets are ordered, the following code returns a "middlemost" element.

```

template <typename T>
T Data<T>::median() const
{
    typename std::multiset<T>::const_iterator iter;
    typename std::multiset<T>::size_type median;
    typename std::multiset<T>::size_type position;
    typename std::multiset<T>::size_type size;

    size = FiniteSet<T>::m_set.size();
    median = (typename std::multiset<T>::size_type) std\.
        ::floor((size-1) / 2);

    iter = FiniteSet<T>::m_set.begin();
    for (position = 0; position < median; position++)
        iter++;

    return *iter;
}

```

Mean

If you write the data as a_1, a_2, \dots, a_m , then the **unweighted mean** is defined as **unweighted mean** $\sum_{i=1}^m \frac{a_i}{m}$. It is written as \bar{a} .

When there are many copies of data, the mean can be computed differently. If the measurement a_i has n_i copies, then the mean is $(\sum_{i=1}^m a_i n_i) / (\sum_{i=1}^m n_i)$.

To have a mean, data needs to have both a sum and a division operator against natural numbers defined.

```

template <typename T>
T Data<T>::mean() const
{
    return sumX() / FiniteSet<T>::size();
}

```

Means can change dramatically if a very large or very small member is added to the data. This behavior is very different than that of medians or modes: single members rarely change the median much, and if a single member changes the mode, then the new mode was already near to being a mode.

Example 1.22 Means, medians, and modes, example 1

Suzanne has counted the blood types of her fellow students. She discovered that, in her class, there are 14 type-O students, 5 type-A students, 6 type-B students, and 1 type-AB student. Then the mode of her data is type-O. Since no natural order exists among the different blood types, her data has neither a median nor a mean.

Example 1.23 Means, medians, and modes, example 2

Imagine a set of gray colors. Grey have a natural ordering: we can say that one gray is darker or lighter than another. Given many hues of gray, you can put them in order from lightest to darkest, and therefore find a median. But since there is no (natural) addition among gray colors, there is no mean among the grays.

Example 1.24 Means, medians, and modes, example 3

Leona teaches Esperanto at Tutmonda University. She has the set of scores of the final exams of her students. Since these scores have a computable scale, Leona can find the mean, median, and mode of the scores.

PROBLEMS: (Answers on page 253)

1. (Easy) The code for the median always gives an element of the set, even when the median might be better expressed as a number not in the set. An alternate implementation might count the size of the set; if the size is even, return a number between `set[floor(size / 2)]` and `set[ceil(size / 2)]`. Why might this implementation be a bad idea?
2. (Easy) You are a philanthropist. You want to raise the “average” wage for a town by donating \$1000 per year to a deserving citizen.
 - (a) If you wanted to raise the *mean* wage in a town, does it matter who you give the \$1000 per year to?
 - (b) If you wanted to raise the *median* wage in a town, does it matter who you give the \$1000 per year to?
3. (Easy) Why does the definition for median say a element at which half *or more* of the data is below or equal to it, and half *or more* of the data is above or equal to it?

The mean and median can be very different. In particular, consider a company where the six peons make \$20,000 per year, the two managers make \$50,000 per year, and the president makes \$500,000 per year. In such a company, the mean salary would be \$80,000, but that number would not represent the experience of most employees. The median or mode, \$20,000, would better represent the salary of the average employee. This asymmetry is called skew.

Sidebar 1.2: Skew

4. (Easy) A psychologist wants to compare the grades of men and women in two majors. She generates the following table:

Gender	Number	Subject	Grade
Male	70	Engineering	72
Male	20	Literature	94
Female	30	Engineering	68
Female	80	Literature	91

There are three questions:

- (a) Who scores higher in engineering: males or females?
 - (b) Who scores higher in literature: males or females?
 - (c) Who has the higher average score: males or females?
5. (Medium) An Internet connection is rather intermittent. There are three sets of data to be transmitted, of exactly equal length. The first set takes 3 minutes to transmit. The second set takes 5 minutes to transmit. What is the best estimate for how long the third set will take to send? (Hint: The answer is not 4 minutes!)

1.6 Measurements of dispersion

In section 1.5 on page 26, we went over several ways to summarize data. We are sometimes interested not only in the mean, median, or mode, but also how much **variation**, also known as **dispersion**, is in the data.

Consider graph 1.7 and graph 1.8 on page 32. They both have the same mean, but the first has less variation than the second. This section gives several ways to measure that variation.

variation
dispersion

Figure 1.7: Variation example 1

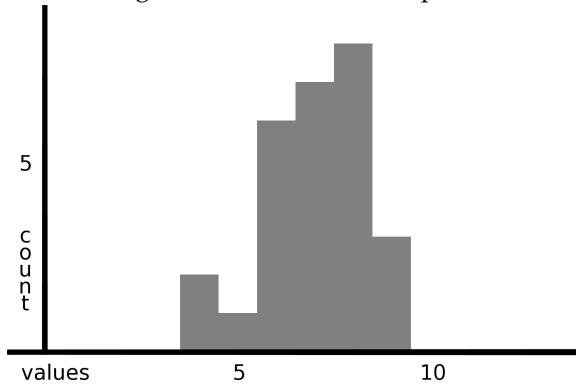
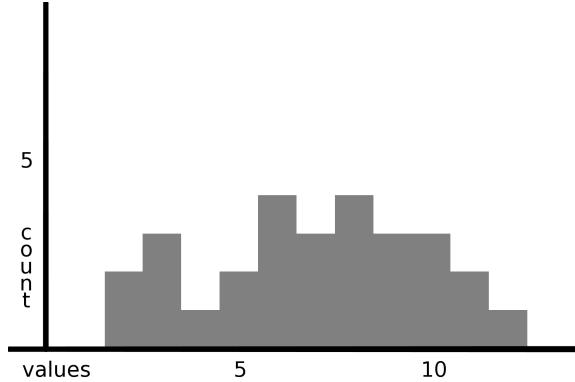


Figure 1.8: Variation example 2



1.6.1 Percentiles, quartiles, inter-quartile range, and confidence interval

range The most basic way to describe the variation in data is the **range**. The range is the difference between the largest and the smallest member.

median Extend the idea of the median, as defined in section 1.5.2 on page 28. A **median** is an element at which 50% of the data is below or equal, and 50% of the data is above or equal. In the same way, an *n*th **percentile** is the smallest element at which *n*% of the data is below or equal, and (100 – *n*)% of the data is above or equal.

quartile The first **quartile** is defined as the twenty-fifth percentile: the element where 25% of the data is below or equal, and where 75% of the data is above or equal. The first quartile can be thought of as the median of all data between the smallest member and the median. The second quartile is the median. The third quartile is defined as the seventy-fifth percentile: the element where 75% of the data is below or equal, and where 25% of the data is above or equal.

inter-quartile range One measurement of variance, the **inter-quartile range**, is the difference between the third quartile and the first quartile.

confidence interval An *n*% **confidence interval** for data is an interval that covers *n*% of the data.

Think about any skill that can be objectively measured, like reading speed. Imagine that we measure millions of ten-year-old students around the country, and learn that, on average, boys and girls read at the same rate. Nevertheless, in this world, teachers discover that almost all of the fast readers are girls. This situation could happen if girls had a wider variation in their reading speed than boys. In this world, teachers would discover that almost all of the slow readers are girls, too.

Sidebar 1.3: Why means aren't enough

It is sometimes but not always defined as the interval from the $\frac{n}{2}$ percentile to the $100 - \frac{n}{2}$ percentile. The inter-quartile range is a 50% confidence interval. We'll often use the 95% confidence interval to describe data.

Example 1.25 Percentiles

In Sennomo Island, Ms. Flore teaches Esperanto. The exam scores of her twenty students were:

60	63	67	68	69	71	71	73	77	78
82	82	85	88	93	97	97	98	99	100

What is an inter-quartile range?

In this data, the first quartile is 69, the third quartile is 93, so the inter-quartile range is 24.

What is an 95% confidence interval?

We don't have enough data to specify a 2.5% percentile: the lowest score, 60, represents a 5% percentile. Since we have the 5th percentile and the 100th percentile, we can say that a 95% confidence interval for the data is [60, 100].

PROBLEMS: (Answers on page 254)

1. (Easy) Write code for the percentile. Use the following header:

```
template <class T>
T Data<T>::percentile(double percent) const
```

1.6.2 Order statistics

Another way to examine the dispersion of data is through **order statistics**. **order statistics** Given data x_1, x_2, \dots, x_n , then sort the data. The order statistics are the 1-based indexes of the sorted data. The smallest member (the minimum) is written $x_{(1)}$, the second-smallest is written $x_{(2)}$, and the largest member is written $x_{(n)}$.⁵ Notice that order statistics use 1-based indexes⁶, while in computers, most arrays use 0-based indexes⁷.

⁵Some books write the order statistics with superscripts, as $x^{(1)}, x^{(2)}, \dots, x^{(n)}$. I write order statistics with subscripts because $x^{(n/2)}$ is confusing whether I'm writing x to the $n/2$ power, or I'm writing the $n/2$ th order statistic of x .

⁶The smallest element is 1.

⁷The smallest element is 0.

PROBLEMS: (Answers on page 254)

1. (Easy) Write code for order statistics. Use the following header:

```
template <class T>
T order(const std::multiset<T>& data, int order);
```

2. (Easy) The C function `drand48()` generates uniformly-distributed⁸ random numbers from 0 to 1. Write a function that generates a set of n uniformly-distributed random numbers from 0 to 1, and finds $x_{(1)}$ for them. Run this function thousands of times where $n = 2$ and take the average. Then run this function thousands of times where $n = 3$ and take the average. Repeat until $n = 9$. What is the pattern?

1.6.3 Variation

Another way to measure variation in data is to measure the distance between the mean and the data.

Euclidean distance

The most common distance is the **Euclidean distance**. In two dimensions, the Euclidean distance between (x_1, x_2) and (y_1, y_2) is $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$. In more dimensions, the Euclidean distance between (x_1, x_2, \dots, x_n) and (y_1, y_2, \dots, y_n) is $\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$.

How could we find a distance between one mean and many data? We can repeat the mean as many times as we have data. So one possible distance between the mean, \bar{a} , and a set of data, $\{a_1, a_2, \dots, a_n\}$ is $\sqrt{(\bar{a} - a_1)^2 + (\bar{a} - a_2)^2 + \dots + (\bar{a} - a_n)^2}$.

However, there's a problem with that definition of mean. Think about two sets of data: $A_1 = 1, 1, 3, 3$ and $A_2 = 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3$. The mean of these two sets is the same, 2. The amount of variation of these two sets should be about the same. Compute using the above distance:

$$\begin{aligned} d(\{2, 2, 2, 2\}, A_1) &= \sqrt{1^2 + 1^2 + (-1)^2 + (-1)^2} \\ &= \sqrt{4} \\ &= 2 \\ d(\{2, 2, \dots, 2\}, A_2) &= \sqrt{1^2 + 1^2 + \dots + (-1)^2} \\ &= \sqrt{16} \\ &= 4 \end{aligned}$$

This distance depends on the number of elements in data. Although it's a good definition of distance, it's not a good measurement of variation.

variance

When we have data, a first definition⁹ of **variance** is $s^2 = \frac{\sum_{i=1}^n (\bar{a} - a_i)^2}{n-1}$.

⁸where any random number is as equally likely as any other random number.

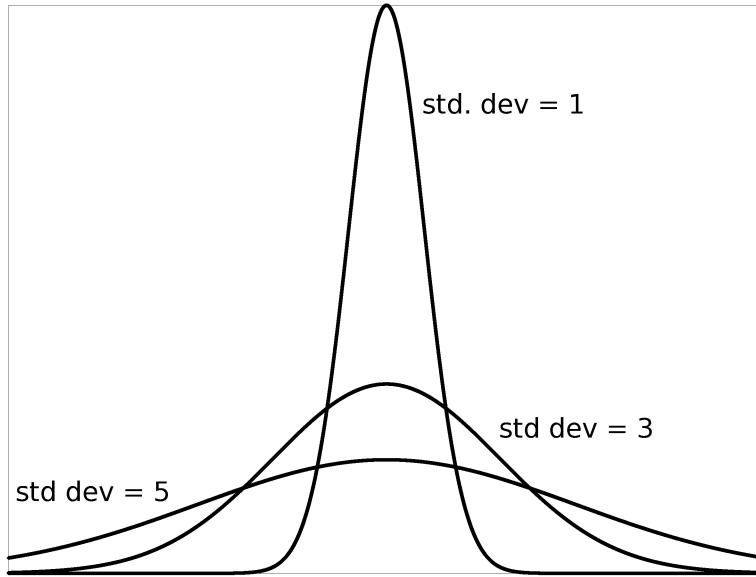
⁹A second definition is on page 55

The denominator comes from linear algebra. We divide by $n - 1$ instead of by n because the numbers only have $n - 1$ degrees of freedom. In other words, if the data is a_1, a_2, \dots, a_n , and you are given \bar{a} and a_1, a_2, \dots, a_{n-1} , then you can easily compute a_n :

$$\begin{aligned} n\bar{a} &= \sum_{i=1}^n (a_i) \\ n\bar{a} - \sum_{i=1}^{n-1} (a_i) &= \sum_{i=1}^n (a_i) - \sum_{i=1}^{n-1} (a_i) \\ &= a_n \end{aligned}$$

The **standard deviation** is the square root of the variance: $s = \sqrt{s^2} = \text{standard deviation}$ $\sqrt{\frac{\sum_{i=1}^n (\bar{a} - a_i)^2}{n-1}}$. A comparison of the size of standard deviations is illustrated in graph 1.9.

Figure 1.9: Standard Deviation



An programming definition for the variance of a set follows:

```
template <typename T>
T Data<T>::variance() const
{
    typename std::multiset<T>::const_iterator iter;
    typename std::multiset<T>::size_type my_size;
```

```

T sum_of_squares;
T square_of_sum;
T sum;

my_size = size();
sum_of_squares = sumX_squared();
sum = sumX();
square_of_sum = sum * sum;

if (my_size > 1)
    return (sum_of_squares - (square_of_sum / \
        my_size)) / (my_size - 1);
else
    return 0;
}

```

PROBLEMS: (Answers on page 256)

1. (Easy) I wrote that the Euclidean distance between data and the mean depends on the number of members. Under what circumstances does it not depend on the number of members?
2. (Easy) You are coding the variance function for an embedded system, and your program must use as little memory as possible. There can be between 2 and 32,767 doubles to take the variance of. How many variables do you need to keep track of, in order to compute the variance?
3. (Medium) I define s in the text as $\frac{\sum_{i=1}^n (\bar{a} - a_i)^2}{n-1}$. The program uses a different formula. Show that the two formulas are the same.
4. (Easy) The variance and standard deviation seems difficult. Why couldn't we just measure the average difference between each data element and the mean: $\sum_{i=1}^n (a_i - \bar{a}) / n$?
5. (Medium) Write code that stores 1000 random numbers between 0 and 100 generated by `drand48`. Find the mean and standard deviation of these numbers. Find what percentage of the data lies between $\mu \pm 2\sigma$. Does this percentage come close to the expected percentage of 95% listed in the sidebar?
6. (Medium) Write code that flips 100 coins, and counts the number of times 'heads' comes up. Have the code repeat this measurement 1000 times. Find the mean and standard deviation of these measurements. Find what percentage of the measurements lies between $\mu \pm 2\sigma$. Does this percentage come close to the expected percentage of 95% listed in the sidebar?

1.6.4 Less-used measurements

You might occasionally see these other measurements in other texts. Many of them delete extreme data.

Other measurements of central tendency

The $n\%$ - **trimmed mean** is sometimes used instead of the median. This measurement removes the smallest $n\%$ and largest $n\%$ of the data, then takes the mean of the remaining $(100 - 2n)\%$ of the data.

The $n\%$ - **winsorized mean** substitutes the smallest $n\%$ with the n percentile, the largest $n\%$ with the $100 - n$ percentile, then takes the mean of the remaining $(100 - 2n)\%$ of the data.

The **midrange** is the halfway point between the highest and lowest members.

Example 1.26 Other measurements of central tendency

Assume that we have test scores 0, 0, 50, 60, 70, 80, 85, 90, 95, 100.

Then:

The mean is 63.

The 10%-trimmed mean removes a 0 and the 100 from the data, then take the mean of the remaining eight members, or 66.25.

The 20%-winsorized mean substitutes 50 for the two 0s, substitutes 90 for the 95 and 100, then takes the mean of the ten members, or 71.5.

The midrange is halfway between 0 and 100, or 50.

PROBLEMS: (Answers on page 259)

1. (Medium) According to rumor, the trimmed mean can have less variance than the mean or the median. Is this true or false? Here's a way to find out:

First, we need a source of data. For this experiment, make a loop that counts the number of times drand48() is called until it generates a random number less than 1. (This source is called the exponential distribution, and it's described on page 123.)

Second, we need to measure the variation of the source of data. To get one measurement, gather n pieces of data, trim it, and get the variance. Vary the amount of trim so that the trim ranges from 0% (which is identical to the mean) to nearly 50% (which is identical to the median).

Does this graph show the variance increasing, staying the same, or decreasing?

Other measurements of dispersion

Every measure of distance also gives rise to a measure of variation.

Manhattan distance

The **Manhattan distance**, also called the city block distance or the taxicab metric, between two points at (x_1, x_2, \dots, x_n) and y_1, y_2, \dots, y_n is $\sum_{i=1}^n |x_i - y_i|$.

mean absolute deviation

The Manhattan distance gives rise to the **mean absolute deviation**. If the data is $\{a_1, a_2, \dots, a_n\}$, the Manhattan distance is defined as:

$$\frac{\sum_{i=1}^n |a_i - \bar{a}|}{n - 1}$$

Because it uses the absolute value function, it is not differentiable everywhere. Therefore, it's less mathematically tractable than other measurements of variation.

Chebyshev distance

The **Chebyshev distance** between two points at (x_1, x_2, \dots, x_n) and y_1, y_2, \dots, y_n is $\max |x_i - y_i|$.

maximum absolute deviation

The Chebyshev distance gives rise to the **maximum absolute deviation**. If the data is $\{a_1, a_2, \dots, a_n\}$, the maximum absolute deviation is defined as:

$$\max_i |a_i - \bar{a}|$$

Example 1.27 Other measurements of dispersion

Assume that we have the data 60, 70, 80, 90, 100.

The mean absolute deviation would be $\frac{20+10+0+10+20}{5} = 12$.

The maximum absolute deviation would be 20.

1.6.5 Other measurements

A few other measurements are occasionally used to describe distributions. I list them for completeness' sake; they won't be used outside of this subsection.

moments of a distribution

The **moments of a distribution** are easy to understand. If x_i are the data, and there are n members, then the moments of its distribution are:

$$\begin{aligned}\mu_1 &= \frac{\sum_{i=1}^n (x_i - \bar{x})^1}{n} \\ \mu_2 &= \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n} \\ \mu_3 &= \frac{\sum_{i=1}^n (x_i - \bar{x})^3}{n} \\ \mu_4 &= \frac{\sum_{i=1}^n (x_i - \bar{x})^4}{n}\end{aligned}$$

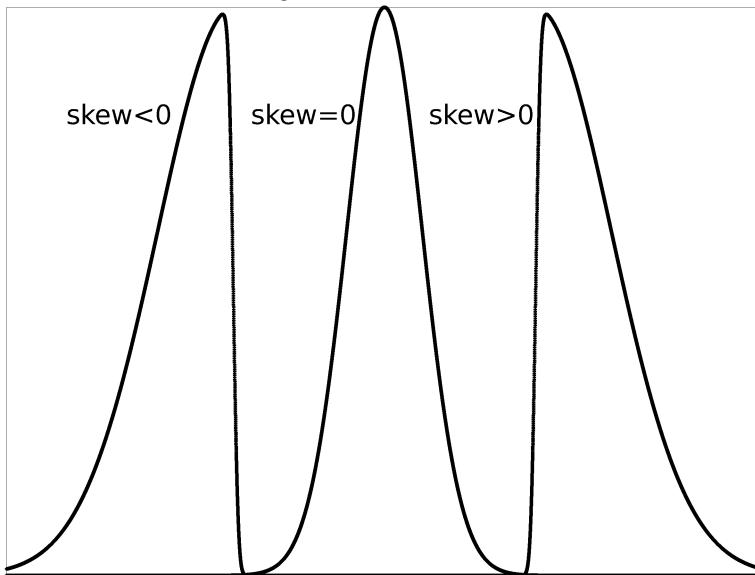
The first moment is always 0.

The second moment is very similar to the variance. The variance is divided by $n - 1$; the second moment is divided by n .

The higher moments are not very useful because of their units. If x_i is measured in seconds, then μ_3 is in seconds³ and μ_4 is in seconds⁴. However, two measurements based on them are sometimes useful.

The **skewness** of data is $\gamma_3 = \frac{\mu_3}{\mu_2^{3/2}}$. It is a measurement of how biased data **skewness** is toward the positive or negative side. It is illustrated in graph 1.10.

Figure 1.10: Skewness



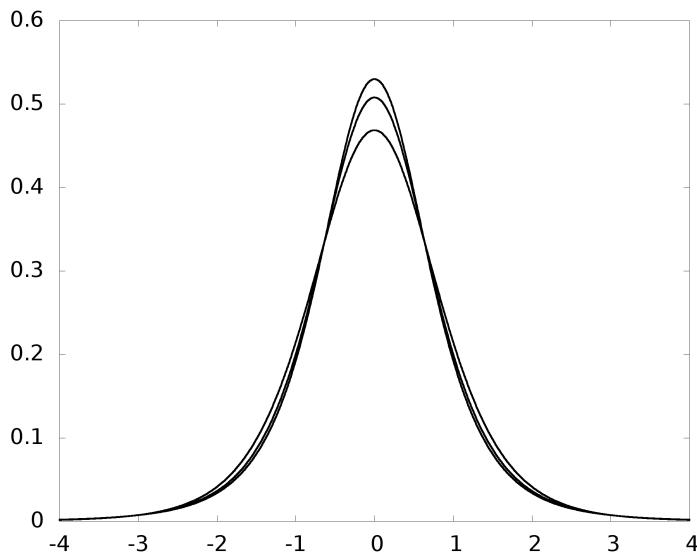
Three overlapping graphs, with three different skewnesses.

When skewness is positive, the data is biased toward the positive side. When skewness is negative, the data is biased toward the negative side. When skewness is zero, the data has no bias in this measurement.

The **kurtosis** of data is $\gamma_4 = \frac{\mu_4}{\mu_2^2}$. It measures the spread of data: whether **kurtosis** data has infrequent, extreme deviations (high kurtosis) or data has frequent, moderate deviations (low kurtosis). It is illustrated in graph 1.11 on page 40.

The kurtosis of the normal distribution (the most common distribution) is 3, so statisticians occasionally talk about **excess kurtosis** — the kurtosis minus **excess kurtosis** three.

Figure 1.11: Kurtosis: kurtosis of 3, 13, and 1003.



1.7 Calculus

Calculus

Calculus brings both good news and bad news. The bad news is that statistics depends on Calculus. The good news is that Statistics depends on numeric results from Calculus, not on the means of computing those results.

1.7.1 Functors

functor In C++, a **functor** is a class that can look like an operation. Functors have `operator()` as a method; if `foo` is a functor that takes no parameters, then `foo()` is valid code.

For this book, functors will inherit from the following interface:

```
class IFunction
{
public:
    virtual ~IFunction() {}

    virtual double operator() (double parameter) = 0;
};
```

For this book, all functors will be functions with one parameter.

1.7.2 Limits

We don't use the idea of a **limit** frequently in computer science. However, this **limit** idea is fundamental to differentiation.

Let $f(x)$ be defined for values of x around (but not necessarily including) some value a .

If, as x is confined to smaller intervals around a , then $f(x)$ becomes confined to smaller intervals around some value l , then l is called the *limit* of $f(x)$ as x approaches a .

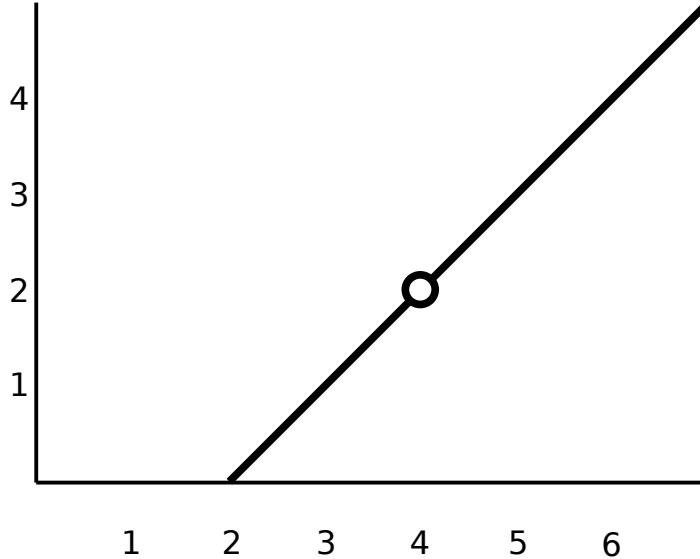
This idea can be written either as "the limit of $f(x)$ as x approaches a is l " or as $\lim_{x \rightarrow a} f(x) = L$.¹⁰

Example 1.28 Limits, simple example.

Consider the function $f(x) = \frac{x^2 - 6x + 8}{x - 4}$, which is graphed below.

Now, $f(4) = \frac{0}{0}$, so the function isn't defined at 4. But division shows that the function would be $f(x) = x - 2$, when $x \neq 4$. For any x , as the distance from x to 4 shrinks, so does the distance from $f(x)$ to 2. Therefore, $\lim_{x \rightarrow 4} \frac{x^2 - 6x + 8}{x - 4} = 2$.

Figure 1.12: Limits



¹⁰The definition of limit is based on [Kleppner and Ramsey, 1985, page 54]

PROBLEMS: (Answers on page 261)

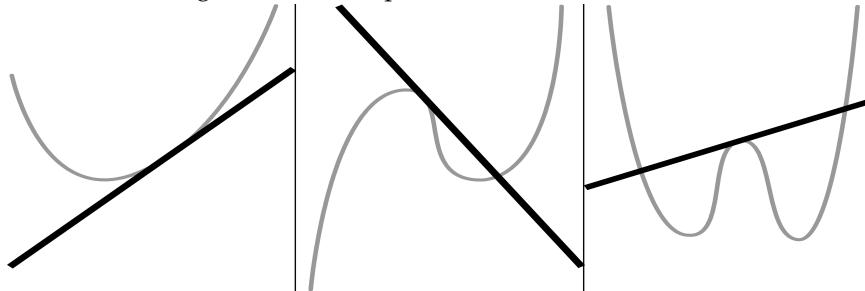
1. (Easy) The most famous limit is $\lim_{x \rightarrow 0} (1 + x)^{1/x}$. This limit is called e, and its value is about 2.718281828459. Write a program that approximates e using this formula.

1.7.3 Differentiation

slope Straight lines have **slope**. The slope is the number of units that it increases or decreases on the y-axis per unit of the x-axis. If a straight line is written in the form $y = mx + b$, then m is the slope of that line.

Only straight lines have slope. However, small sections of most continuous **differential** functions look like straight lines.¹¹ The **differential** of a function, $f(x)$, can be thought of as a function, $f'(x)$ that returns the very local slope of $f(x)$. This idea is diagrammed on figure 1.13.

Figure 1.13: Examples of differentiation lines.



There are several ways to write the derivative of a function, $f(x)$. The most common way is $\frac{df(x)}{dx}$ or $\frac{d}{x}f(x)$. The second-most common way was used above: $f'(x)$.

Computing Differentials

The mathematical definition of the differential of a function is

$$f'(x) = \lim_{\delta \rightarrow 0} \frac{f(x + \delta) - f(x)}{\delta}$$

The differential is a function that takes as input one function, and that returns as output another function.¹²

Using functors, approximating the differential is easy:

¹¹Some continuous functions don't look like straight lines, no matter how small of a section you look at. For example, $f(x) = |x|$ always has a sharp edge — can't be differentiated — at 0.

¹²If you haven't experienced these kinds of functions, take a class in functional programming.

```

Differentiate::Differentiate(IFunctor *pifunctor, double delta)
{
    if (delta <= 0)
        throw new std::out_of_range("Cannot_use_a_nonpositive_delta.");
    m_pifunctor = pifunctor;
    m_delta = delta;
}

double Differentiate::operator() (double parameter)
{
    double dResult;

    dResult = ((*m_pifunctor)(parameter + m_delta) - (*m_pifunctor)(parameter)) / m_delta;

    return dResult;
}

```

Notice that `Differentiate` is an `IFunctor`. Differentiation can be chained.

Rule set 1.1: Some rules of differentiation

Only a book about Calculus can give all of the rules of differentiation. The rules below are the most important.

For the following rules, x is a variable, u and v are variables that depend on x , f and g are functions of x , a and n are constants, and e is the base of the natural logarithm, or about 2.718.

1. $\frac{da}{dx} = 0$
2. $\frac{d}{dx}(ax) = a$
3. $\frac{dx^n}{dx} = nx^{n-1}$ if $n \neq 0$
4. $\frac{d}{dx}(u + v) = \frac{du}{dx} + \frac{dv}{dx}$
5. $\frac{d}{dx}(uv) = u \frac{dv}{dx} + v \frac{du}{dx}$
6. $\frac{d \ln x}{dx} = \frac{1}{x}$
7. $\frac{de^x}{dx} = e^x$
8. $\frac{df}{dx} = \frac{df}{dg} \times \frac{dg}{dx}$ (the chain rule)

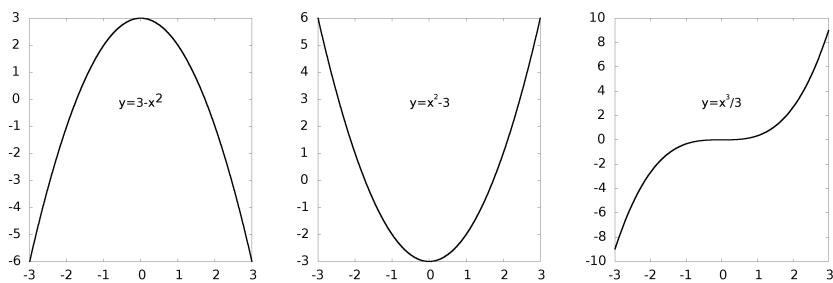
[Kleppner and Ramsey, 1985, page 254]

Maximums and Minimums

Differentiation is important because it lets you find local minimums and maximums for functions.

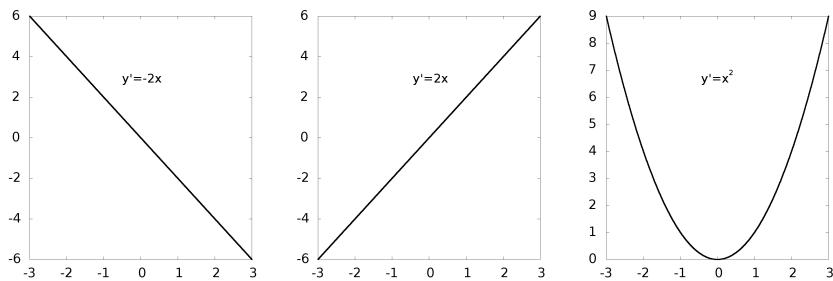
A local maximum occurs when values to either side are less than or equal to a current value. A local minimum occurs when values to either side are greater than or equal to a current value. In both cases, the function is “flat” at that element: its slope is zero.

Figure 1.14: Three curves



In graph 1.14, the first curve is a maximum, and the second curve is a minimum.

Figure 1.15: The derivatives of the three curves

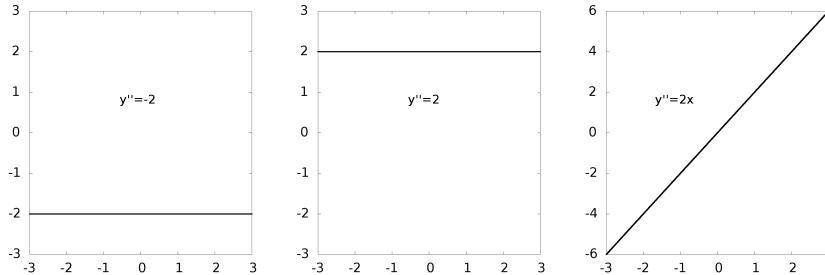


The derivatives of graph 1.14 are in graph 1.15. Notice that all three graphs have a derivative of zero at zero. If the function has a maximum or minimum at x , then its derivative will be zero at x . However, the opposite is not the case: even if the derivative at x is zero, x may not be a maximum or minimum.

We have a strange situation: we have a test that will recognize all maximums and minimums, but also other elements. Further, the test doesn't distinguish between maximums, minimums, or other elements.

A way to distinguish most of them is to take the derivative of the derivative, called the second derivative.

Figure 1.16: The second derivatives of the three curves



The second derivatives are in graph 1.16 on page 45. When the first derivative is zero and the second derivative is positive, then the curve is at a local minimum. When the first derivative is zero and the second derivative is negative, then the curve is at a local maximum. When both the first and second derivatives are zero, then assume nothing about the location; it could be a local maximum, minimum, or inflection point¹³

PROBLEMS: (Answers on page 262)

1. (Easy) Find a curve where the first and second derivatives are zero at a point, and where the point is a local maximum.
2. (Easy) Find a curve where the first and second derivatives are zero at a point, and where the point is a local minimum.

Newton-Raphson Method

Differentiation is also important because it sometimes allows you to find the solution to a function quickly through the Newton-Raphson Method.

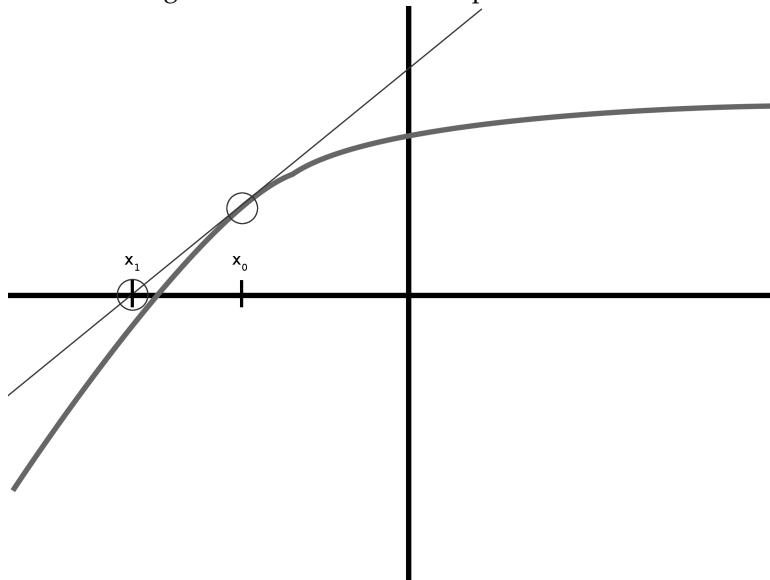
Imagine that you have a function, $f(x) = 0$ and an approximation, x_0 to the solution. You want to find a better approximation. The Newton-Raphson Method says that you should use the local slope around x_0 to create a tangent line, then find where this tangent line crosses zero. That element, x_1 , is likely to be a better approximation than x_0 .

In figure 1.17 on page 46, the right circle elements to the first approximation and the left circle elements to a better approximation.

Repeat the Newton-Raphson method, starting at x_1 , to find another approximation x_2 . If each action finds a better approximation, then this method converges to the solution.

¹³An inflection point happens when the first derivative is zero, but the curve is not at a local minimum or maximum. In short, nothing happens there.

Figure 1.17: The Newton-Raphson Method



```

double Newton_Raphson(IFunctor *pifunctor,
                        double deltaFunctor,
                        double initialEstimate,
                        double minDeltaEstimate)
{
    double currentDifference = initialEstimate;
    double currentEstimate = 1E10;
    double previousDifference;
    double previousEstimate;

    Differentiate diff(pifunctor, deltaFunctor);

    do {
        previousDifference = currentDifference;
        previousEstimate = currentEstimate;

        currentEstimate = previousEstimate -
            (*pifunctor)(previousEstimate) / diff(\_
            previousEstimate);

        currentDifference = fabs(currentEstimate - \
            previousEstimate);
    }
}
```

```

if (currentDifference >= previousDifference)
    // The sequence diverged.
    break;
} while (currentDifference > minDeltaEstimate);

return currentEstimate;
}

```

PROBLEMS: (Answers on page 263)

1. (Medium) Look at the line that defines `currentEstimate`. Explain how that line was derived.

1.7.4 Integration

In Calculus, the opposite of differentiation is **integration**.

integration
indefinite integration

Like differentiation, **indefinite integration** is a function that takes a function and returns a function. Indefinite integration is an inverse of differentiation: if an indefinite integral of $f(x)$ is $F(x)$, then the derivative of $F(x)$ will be $f(x)$.

Rule set 1.2: Some basic integrals

There are many rules of integration, just as there are rules of differentiation. The rules below are not complete.

For the following rules, x is a variable, u and v are variables that depend on x , a and n are constants, and e is the base of the natural logarithm.

1. $\int a \, dx = ax$
2. $\int af(x) \, dx = a \int f(x) \, dx$
3. $\int(u+v) \, dx = \int u \, dx + \int v \, dx$
4. $\int x^n \, dx = \frac{x^{n+1}}{n+1}$ if $n \neq -1$
5. $\int \frac{dx}{x} = \ln x$
6. $\int e^x \, dx = e^x$
7. $\int e^{ax} \, dx = \frac{e^{ax}}{a}$
8. $\int b^{ax} \, dx = \frac{b^{ax}}{a \ln b}$
9. $\int \ln(x) \, dx = x \ln(x) - x$
10. $\int u \, dv = uv - \int v \, du$
11. $\int_a^b f(x) \, dx + \int_b^c f(x) \, dx = \int_a^c f(x) \, dx$

$$12. \int_a^b f(x) dx = - \int_b^a f(x) dx$$

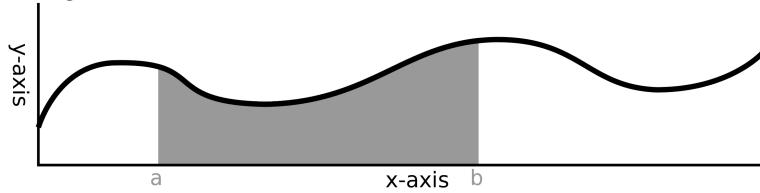
$$13. \int u dv = uv - \int v du$$

[Kleppner and Ramsey, 1985, page 256]

definite integral This book will mostly use the **definite integral**, which can be computed numerically.

The definite integral can be thought of as the area under a curve between two elements. It looks like graph 1.18.

Figure 1.18: The area under a curve between two elements



The definite integral for a function $f(x)$ from a to b is either written as $F(x)|_a^b$, or as $F(b) - F(a)$, or as

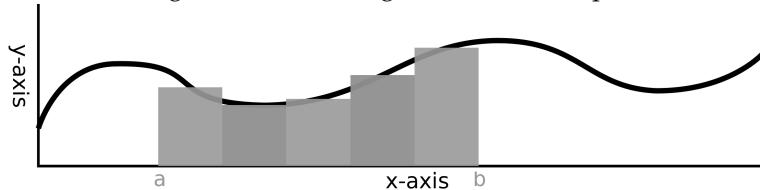
$$\int_a^b f(x) dx$$

The definite integral from a to b on the function $f(x)$ is defined as

$$\int_a^b f(x) dx = \lim_{dx_0 \rightarrow 0} \sum_{x_0=a/dx_0}^{b/dx_0} f(x_0 \times dx_0) \times dx_0$$

Given a function $f(x)$, how can you compute its definite integral from a to b ? The easiest way is to choose a small number δ , divide the length from a to b into vertical strips each δ wide, and sum the area of the strips. It looks like graph 1.19.

Figure 1.19: Dividing the area into strips



Using `Interval` and the `IFunctor` gives an implementation:

```

Integrate::Integrate(IFunctor *pifunctor, double delta)
{
    if (delta <= 0)
        throw new std::out_of_range("Cannot_use_a_\
nonpositive_delta.");
    m_pifunctor = pifunctor;
    m_delta = delta;
}

double Integrate::integrate(Interval interval)
{
    double integralResult;
    double slice;

    if (interval.isEmpty())
        return 0;

    if (interval.getEndValue() - interval.getStartValue() \
        < m_delta)
        throw new std::out_of_range("The_delta_is_too_large_\
for_this_interval.");

    integralResult = 0;
    for (slice = interval.getBeginValue() + (m_delta/2);
          slice - (m_delta/2) < interval.getEndValue();
          slice += m_delta)
        integralResult += (*m_pifunctor)(slice) * m_delta;

    return integralResult;
}

```

This method of approximating an integral is called the **Riemann Rule**.

Riemann Rule

PROBLEMS: (Answers on page 263)

1. (Medium) The **Standard Normal Curve**, which we'll be dealing with in **Standard Normal Curve** section 3.6.4, has the following definition:

$$\Phi(Z) = \int_{-\infty}^Z \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

This function maps from any real number Z to a real number between 0 and 1. It has two important features:

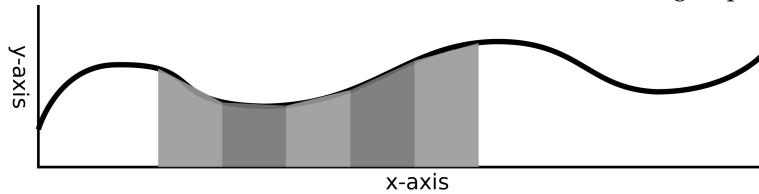
- (a) $\Phi(0) = 0.5$

$$(b) \Phi(-Z) = 1 - \Phi(Z)$$

Write a program that computes the standard normal curve for every 0.01 from -5 through 5. (The standard normal curve can only be computed by numeric methods.)

2. (Medium) An alternate way to compute the definite integral uses trapezoids instead of rectangles. Instead of computing $f(x)$ at the center, this method computes it at both ends: $f(x - \frac{\delta}{2})$ and $f(x + \frac{\delta}{2})$, then draws a line between them. This computation is demonstrated in graph 1.20.

Figure 1.20: The area under a curve between two elements, using trapezoidal



The area for this kind of trapezoidal is $\frac{1}{2} \times \left(f\left(x - \frac{\delta}{2}\right) + f\left(x + \frac{\delta}{2}\right) \right) \times \delta$.

Create a new implementation of `integrate` that uses the above method. Compare its results with definite integrals that you know. Given the same size δ , is it more accurate? Why or why not?

If a measurement is based on many different factors, it usually can be described by the normal distribution. Measurements like height, weight, and test scores fall under this distribution.

If a measurement is described by the normal distribution, its mean is μ , and its standard deviation is σ , then there are three relatively good estimates:

1. About 68% of all data will be in the range $[\mu - \sigma, \mu + \sigma]$.
2. About 95% of all data will be in the range $[\mu - 2\sigma, \mu + 2\sigma]$.
3. About 99.8% of all data will be in the range $[\mu - 3\sigma, \mu + 3\sigma]$

These are magic numbers; when you read section 3.6.4 on page 131, you will have the tools to generate them.

Sidebar 1.4: Normal Distribution

Chapter 2

Probability basics

2.1 Introduction

We use probabilities every day. “If you flip a coin, there’s a fifty-percent chance that it will come up heads.” “The 49ers have a 70% chance of beating the Yankees in the Stanley Cup.”¹

This book is about computer science, not philosophy. In this book, anything that acts like a probability is a probability.²

2.2 Relations between sample spaces and probability

2.2.1 Partitioning A Sample Space

A **partition** is an abstraction where samples in a sample space are considered **partition** in groups.

A partition is a set of samples. Let the samples be called s_1, s_2, \dots, s_n , and the sample space as a whole is S . Then p_1, p_2, \dots, p_m are a partition if they follow the rules:

Rule set 2.1: Rules for partitions

1. $\bigcup_i p_i = S$
2. For any different p_i and p_j , $p_i \cap p_j = \emptyset$

¹If you’re not from North America, then substitute “Your favourite rugby team have a 70% chance of beating your favourite cricket team in the World Cup.”

²A self-aware writer would point out that this sentence is a very strong philosophical assumption. A well-balanced book would note that this position is not accepted by all statisticians, outline the disagreement, and let the reader make up his or her mind. This book is not well-balanced.

Example 2.1 Partitions on three marbles

One partition on the three-marbles example is to group them based on the number of marbles in the sample. These partitions would be

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

This partition contains all eight samples, and no sample appears in two partitions.

Example 2.2 Partitions on the real line

We can create a strange partition on the set $[0, 1]$:

- $[0, 0.1)$
- $[0.1, \frac{\pi}{10})$
- $\{\frac{\pi}{10}, \frac{5}{7}\}$
- $(\frac{\pi}{10}, 0.5] \cup (\frac{5}{7}, 1]$
- $(0.5, \frac{5}{7})$

These five sets contain all real numbers in $[0, 1]$, and no real number appears in two different partitions. The partitions don't need to be continuous.

PROBLEMS: (Answers on page 267)

For the following problems, assume that the sample space is S , and that the samples in S are s_1, s_2, \dots, s_n .

1. (Easy) Prove that $p_1 = \{s_1\}, p_2 = \{s_2\}, \dots, p_n = \{s_n\}$ is a partition on S .
2. (Easy) Prove that $p_1 = \{s_1, s_2, \dots, s_n\}$ is a partition on S .

2.2.2 Random Variables

random variable A central idea of statistics and probability is the **random variable**.

For this book, a random variable is the combination of a sample set, S , and a probability function, \Pr_S , from the sample set to positive, real numbers. The restrictions on the probability function are covered in section 2.2.5 on page 59.

Functions of random variables are themselves random variables.

2.2.3 Expected value

expected value The most important function of a random variable is the **expected value**. For a random variable, F , the expected value is written $E[F]$. It represents $\sum_{f \in F} \Pr(f)f$ when f is discrete and $\int_{f \in F} \Pr(f)f \, df$ when f is continuous.

If k is a constant, then $E[k] = k$

Since k is discrete, $E[X] = \sum_{x \in X} \Pr(X)x$. When k consists of a single constant, then the probability of the single constant is 1, and the probability of any other value is 0. Therefore, $E[k] = 1 \times k = k$.

Theorem 2.1: The expected value of a constant

I (and most introductory textbooks) mislead with the first definition of **mean** **mean** and **variance** on page 29 and page 34. When you have a distribution, there is a **variance** more general definition:

The mean for a distribution X is defined as $\mu = E[X]$.

The variance for a distribution X is defined as $\sigma_X^2 = E[(X - E[X])^2]$. The standard deviation is still the square root of the variance.

PROBLEMS: (Answers on page 267)

1. (Easy) Let $A = \{a_1, a_2, \dots, a_n\}$. Prove that when the measurements are equiprobable, that the mean is the same as $E[A]$.
2. (Easy) If X is a random variable, f and g are functions on X , then prove that $E[f(X) + g(X)] = E[f(X)] + E[g(X)]$.
3. (Easy) If X is a random variable, f is a function on X , and k is a constant, then prove that $E[k \times f(X)] = k \times E[f(X)]$.
4. (Easy) If X is a random variable, then prove that $E[E[X]] = E[X]$.
5. (Medium) Let $A = \{a_1, a_2, \dots, a_n\}$. (They may or may not be equiprobable.) Prove that $E[(A - E[A])^2] = E[A^2] - (E[A])^2$.
6. (Medium) I pulled a fast one. I gave two definitions of variance when the samples are equiprobable: $s^2 = \frac{\sum_{i=1}^n (a_i - \bar{A})^2}{n-1}$ and $\sigma^2 = E[(A - E[A])^2]$. Not only are they not the same, unless the variance is zero, they always give different answers! What's the difference?

2.2.4 Naïve probability

A naïve but useful form of probability makes several assumptions:

1. Every sample has the same probability as every other sample. Therefore, if the sample space, E , has n elements, then every sample $e \in E$ has a probability of $\frac{1}{n}$.
2. The probability of a partition depends only on the number of unique samples in the partition, not on which samples are in the partition. Therefore, if a partition has k samples, and the sample space as a whole has n samples, then the probability of that partition is $\frac{k}{n}$.

In most games of chance, samples are divided into two partitions, called *success* and *failure*.

This naïve definition can handle many simple questions of probability. It's useful for examining many games of chance.

Example 2.3 Payoffs in roulette

American roulette is simple: A croupier spins a wheel then rolls a ball in the opposite direction so that it randomly chooses a slot, numbered in the set $\{1, 2, 3 \dots, 36, 0, 00\}$. A gambler bets on where the ball will land. Each bet can be on an individual number, or on a specific set of numbers. If the ball lands on a chosen number, then the gambler wins.

The expected value for a bet is the product of the value and the probability of the value. The expected values for a \$1 bet in roulette are:

Kind of bet	Winning Value	Probability	Expected value
Straight (single number)	\$36	$\frac{1}{38}$	$36 \times \frac{1}{38} = \frac{36}{38}$
Split (two numbers)	\$18	$\frac{2}{38}$	$18 \times \frac{2}{38} = \frac{36}{38}$
Street (three numbers)	\$12	$\frac{3}{38}$	$12 \times \frac{3}{38} = \frac{36}{38}$
Corner (four numbers)	\$9	$\frac{4}{38}$	$9 \times \frac{4}{38} = \frac{36}{38}$
Five numbers	\$7	$\frac{5}{38}$	$7 \times \frac{5}{38} = \frac{35}{38}$
Sixline (six numbers)	\$6	$\frac{6}{38}$	$6 \times \frac{6}{38} = \frac{36}{38}$
Any twelve-number bet	\$3	$\frac{12}{38}$	$3 \times \frac{12}{38} = \frac{36}{38}$
Any eighteen-number bet	\$2	$\frac{18}{38}$	$2 \times \frac{18}{38} = \frac{36}{38}$

The only bet with an expected value other than $36/38$ is the five-number bet, which has an expected value of $35/38$.

Example 2.4 Poker hands

Poker is played with a 52-card deck, which has every combination of thirteen ranks of cards (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) and four suits (\clubsuit , \diamondsuit , \heartsuit , \spadesuit). In five-card stud poker, each player gets five of these cards. A player has a pair when two of the five cards have the same rank.

What is the chance in five-card stud poker that a player has a pair?

Sometimes, it's easier to find the probability that something is not in the set, then subtract from one to find the probability that something is in the set. What is the chance that, among five cards, no two cards have the same rank?

There are $\binom{52}{5}$ different poker hands. (Refer back to section 1.4.5 on page 24 if you don't remember this notation.) This number is our denominator.

Think about the number of permutations of poker hands with no two cards of the same rank. The first card has 52 possible choices, one for each card in the deck; the second card has 48 possible choices: we may not reuse any card of the same rank; the third card has 44 possible choices; and so forth. Therefore, there are $52 \times 48 \times 44 \times 40 \times 36$ permutations. Since the number of combinations is the number of permutations divided by the number of ways that the permutations can be ordered, there are $\frac{52 \times 48 \times 44 \times 40 \times 36}{5!} = 1317888$ hands with no two cards alike.

Therefore, the probability that a hand has at least two cards of the same rank is $1 - \frac{1317888}{2598960} = \frac{1281072}{2598960} \approx 49.4\%$.

PROBLEMS: (Answers on page 269) Almost every book about probability has dozens of problems that belong to this section. Naive probability is one of the most commonly-known parts of probability.

1. (Easy) There's a common bar bet. Take seven cards: $A\diamondsuit$, $2\diamondsuit$, $3\diamondsuit$, $4\diamondsuit$, $5\diamondsuit$, $Q\clubsuit$, and $Q\spadesuit$. The fellow betting against you will double your bet if you choose three cards at random, and none of the three cards are the queens. What is the chance that you won't hit any queens?
2. (Medium) Assume that the year has only 365 days. How many people do you need to have in a room, so that the chance that two people share the same birthday is over 50%?
3. (Medium) People write books claiming to show how to win money gambling. One popular method is called the **Martingale system**.³ Its simplest **Martingale system** form is this algorithm:

³<http://casinogambling.about.com/od/moneymanagement/a/martingale.htm>

- (a) Choose some small amount of money, b_0 . Let the initial amount of the bet b be b_0 .
- (b) Make a bet of size b on a double-or-nothing gamble.
- (c) If the bet wins, reset your bet b to b_0 .
- (d) If the bet loses, double the bet: set b to $2b$.
- (e) Go back to step (b).

Every time we win, we gain back all previous bets, plus b_0 :

$$\begin{aligned}
 b_0 &= b_0 \\
 2b_0 &= (b_0) + b_0 \\
 4b_0 &= (b_0 + 2b_0) + b_0 \\
 8b_0 &= (b_0 + 2b_0 + 4b_0) + b_0 \\
 16b_0 &= (b_0 + 2b_0 + 4b_0 + 8b_0) + b_0 \\
 &\dots
 \end{aligned}$$

Write code with three parameters: b , g , and p . It should implement the Martingale system for someone with a initial bankroll of b , an initial bet size of \$1, and that stops if the gambler reaches a bankroll of g or goes bankrupt. The probability of winning any round would be p . If the gambler is not bankrupt, but does not have enough money to cover the whole Martingale bet, assume that the gambler bets everything.

4. (Medium) When most people gamble, they bet the minimum bet size, no matter whether they win or lose. Before you write any code, how well do you think this strategy works? Write new code for someone who starts with a b bankroll, a constant bet size of \$1, and who will stop if she reaches g or goes bankrupt. As in the previous questions, the probability of winning is p , and if the gambler wins, she doubles her bet.
5. (Medium) When extremely bold people gamble, they put all of their money on one bet. Before you write any code, how well do you think this strategy works? Write code for someone who starts with a b bankroll, is interested in a g goal, and who always bets either all of her bankroll or $g - b$ (if $2b > g$).
6. (Medium) [HowstuffWorks.com](http://entertainment.howstuffworks.com/how-to-play-blackjack2.htm)⁴ recommends a different betting strategy.
 - (a) Choose some small amount of money, b_0 . Let the initial amount of the bet be $b = b_0$.
 - (b) Make a bet of size b on a double-or nothing gamble.

⁴<http://entertainment.howstuffworks.com/how-to-play-blackjack2.htm>

- (c) If you have a winning streak lasting w bets, then you should bet $b = \lfloor \frac{w}{2} \rfloor b_0$. For example, if you've won 7 games in a row, you should bet $\lfloor 7/2 \rfloor * b_0 = 3b_0$.
- (d) If you lose, return back to the original level, $b = b_0$.
- (e) Go back to step (b).

Write a function that implements the above idea and uses the same parameters as the above three functions.

7. (Medium) Which of the four above gambling strategies is most likely to bring you from $b = 1000$ to $g = 5000$ with an initial bet of \$1 when $p = 0.4736$ (the chance that a person would double their money on an “even” bet in roulette)?
8. (Medium) In example 2.4 on page 56, I oversimplified. My poker hands example counts hands like $7\clubsuit 6\Diamond 3\clubsuit 3\heartsuit 3\spadesuit$ as pairs, when every poker player knows that hand is a three-of-a-kind. Find the real probability for getting exactly a pair in a poker hand.
9. (Medium) (The Ellsberg Paradox) Assume that there are ninety balls in an urn: 30 balls are red, and 60 balls are green or blue in some unknown proportion. There are four lotteries:

	l_1	l_2	l_3	l_4
red	\$100	\$0	\$0	\$100
green	\$0	\$100	\$100	\$0
blue	\$0	\$0	\$100	\$100
	red	green	blue	
l_1	\$100	\$0	\$0	
l_2	\$0	\$100	\$0	
l_3	\$0	\$100	\$100	
l_4	\$100	\$0	\$100	

The player has to choose either l_1 or l_2 and either l_3 or l_4 .

Most people prefer lottery l_1 over l_2 and prefer lottery l_3 over lottery l_4 , because they give known chances for the prize.

What do you think about this strategy?

(Taken from [Pratt et al., 1995, page 46])

2.2.5 Kolmogorov Axioms

This section gives a more mathematical definition for the probability function.

Once the sample space, S has been chosen, then the probability function must be chosen. If the sample space is discrete, then the function is called the **probability distribution function**. If the sample space is continuous, then the

probability distribution function

function is called the **probability density function**.

For both rule-sets, assume that your sample space is partitioned by $\{s_1, s_2, \dots\}$ where $\bigcup s_i = S$. **probability density function**

If your sample space is discrete, then the rules 2.2 apply:

Rule set 2.2: Kolmogorov Rules for discrete sets

1. \Pr maps from S into $[0, 1]$.
2. $\sum_{s_i \in S} \Pr(s_i) = 1$.
3. If $S_1 \subseteq S$, $S_2 \subseteq S$, and $S_1 \cap S_2 = \emptyset$, then $\Pr(S_1 \cup S_2) = \Pr(S_1) + \Pr(S_2)$.

What do these rules mean?

1. \Pr maps from S into $[0, 1]$.
Every element s_i in S has a probability between 0 and 1 (inclusive).
2. $\sum_{s_i \in S} \Pr(s_i) = 1$
The sum of all probabilities will be exactly one.
3. If $S_1 \subseteq S$, $S_2 \subseteq S$, and $S_1 \cap S_2 = \emptyset$, then $\Pr(S_1 \cup S_2) = \Pr(S_1) + \Pr(S_2)$.
If two sets in S are disjoint, then the probability of their union must be the sum of their probabilities.

If your sample space is continuous, then rules 2.3 apply.⁵

Rule set 2.3: Kolmogorov Rules for continuous sets

1. pdf maps from S to $\mathcal{R}^+ \cup 0$, the set of nonnegative real numbers.
2. $\int_{s_i \in S} \text{pdf}(s_i) dx = 1$.
3. If $I = (s_1, s_2)$ is an interval in S , then $\Pr(I) = \int_{s_2}^{s_1} \text{pdf}(x) dx$
4. If $S_1 \subseteq S$, $S_2 \subseteq S$, and $S_1 \cap S_2 = \emptyset$, then $\Pr(S_1 \cup S_2) = \Pr(S_1) + \Pr(S_2)$.

Kolmogorov Axioms

These rule-sets are called the **Kolmogorov Axioms**, and they define what a probability is. In this book, anything that follows those rules is a probability; anything that doesn't, isn't.

The rules are hard to understand. But they create several simpler theorems about combining two probabilities.

⁵If S is neither discrete nor continuous, then you've gone well beyond what this book can teach.

$$\Pr(\emptyset) = 0$$

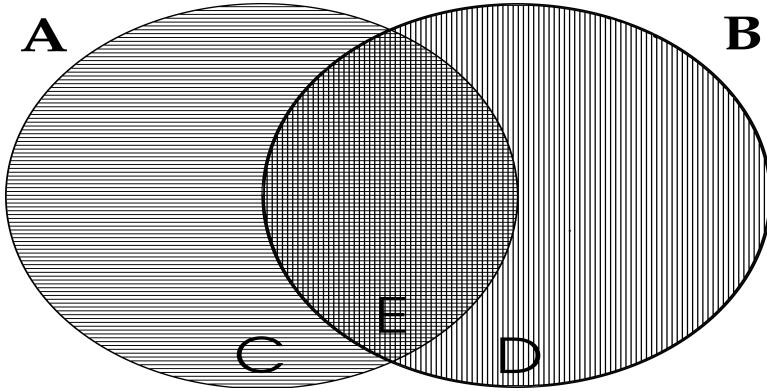
For any set S , $S \cap \emptyset = \emptyset$ and $S \cup \emptyset = S$. Then:

$$\begin{aligned}\Pr(S) &= \Pr(S \cup \emptyset) \\ &= \Pr(S) + \Pr(\emptyset) \\ \Pr(S) - \Pr(S) &= \Pr(S) + \Pr(\emptyset) - \Pr(S) \\ 0 &= \Pr(\emptyset)\end{aligned}$$

Theorem 2.2: Probability of \emptyset

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$$

Given any two sets, A and B , we can create three disjoint sets: $C = A \setminus B$, $D = B \setminus A$, and $E = A \cap B$.



From the above diagram, $C \cup E = A$ and $D \cup E = B$.

$$\begin{aligned}\Pr(A \cup B) &= \Pr(C \cup D \cup E) \\ &= \Pr(C) + \Pr(D) + \Pr(E) \\ &= (\Pr(C) + \Pr(E)) + (\Pr(D) + \Pr(E)) - \Pr(E) \\ &= \Pr(C \cup E) + \Pr(D \cup E) - \Pr(E)\end{aligned}$$

Substitute the definitions of $C \cup E$, $D \cup E$, and E :

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$$

Theorem 2.3: Sum of Probabilities (discrete version)

Example 2.5 Three marbles probabilities

Assume that, when you choose from the set of marbles, for each marble, you are $\frac{2}{3}$ likely to choose it and $\frac{1}{3}$ likely not to choose it. Then the chances of each of the sets of marbles is:

- $\Pr(\{\emptyset\}) = \frac{1}{27}$
- $\Pr(\{1\}) = \Pr(\{2\}) = \Pr(\{3\}) = \frac{2}{27}$
- $\Pr(\{1, 2\}) = \Pr(\{1, 3\}) = \Pr(\{2, 3\}) = \frac{4}{27}$
- $\Pr(\{1, 2, 3\}) = \frac{8}{27}$

This example is a binomial distribution, described in section 3.5.2 on page 92.

Example 2.6 Real Line probabilities

Assume that every point on the line $[0, 1]$ has the same probability. Then some very simple chances emerge:

- $\Pr(0.3) = \Pr(0.222222...) = \Pr(\frac{\pi}{10}) = 0$, since the chance of any particular real number is infinitesimally small.
- $\Pr([0.5, 0.6]) = 0.1$
- $\Pr([0.2, 0.7)) = 0.5$

This example is a uniform distribution, described in section 3.6.3 on page 127.

PROBLEMS: (Answers on page 275)

1. (Medium) Prove that the intuitive probability, of section 2.2.4 on page 56, meets all of the axioms for a probability.
2. (Medium) Why can a continuous probability function have values greater than 1?

2.2.6 Jensen's Inequality

Jensen's Inequality is not part of most people's intuitions of probability. A whole book ([Savage, 2009]) has been written about real-world applications of this inequality.

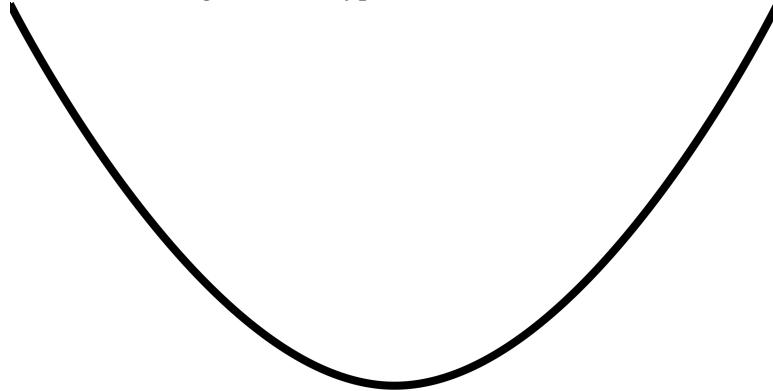
Let's first talk about it in mathematics. Then talk about it practically.

convex function

A function f is a **convex function**⁶ if and only if for any two points x_1 and x_2 and any λ where $0 < \lambda < 1$,

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

Figure 2.1: A typical convex function



A **concave function** is exactly the opposite of a convex function: for any **concave function** two points x_1 and x_2 and any λ where $0 < \lambda < 1$,

$$f(\lambda x_1 + (1 - \lambda)x_2) \geq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

For the same proof, if f is a concave function, then $f(E[X]) \geq E[f(x)]$.

The above inequality seems extremely remote from reality. But let's look at an example.

⁶This definition is modified from Weisstein, Eric W. "Convex Function." From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/ConvexFunction.html>

If f is a convex function, then $f(E[X]) \leq E[f(X)]$.

I will prove this result for a discrete distribution with two values; it is easy to extend the proof.

Let x_1 and x_2 be the two values with nonzero probabilities.

From the above definition, substituting $\Pr(x_1)$ for λ :

$$\begin{aligned} f(\Pr(x_1)x_1 + (1 - \Pr(x_1))x_2) &\leq \Pr(x_1)f(x_1) + (1 - \Pr(x_1))f(x_2) \\ f(\Pr(x_1)x_1 + (\Pr(x_2))x_2) &\leq \Pr(x_1)f(x_1) + (\Pr(x_2))f(x_2) \\ f\left(\sum_{x \in X} \Pr(x)x\right) &\leq \sum_{x \in X} \Pr(x)f(x) \\ f(E[x]) &\leq E[f(x)] \end{aligned}$$

Theorem 2.4: Jensen's Inequality

Example 2.7 Chocolate profit

You run a chocolate shop on the boardwalk, and you sell your chocolates for \$2 per chocolate from ingredients that cost \$1 per chocolate. The distribution for your sales is between 500 and 1,500 chocolates sold per day. You have two restrictions:

1. If you sell out of chocolates, you won't have more until the next day.
2. Your chocolates must be fresh; they are no good if you don't sell them on the day that they're made.

Without knowing Jensen's inequality, it makes most sense to make 1,000 chocolates per day and to expect \$1,000 profit per day. But let's try an experiment:

```
const int k_costIngredients = 1;
const int k_maxExperiment = 1000000;
const int k_price = 2;
const int k_supply = 1000;

int main(int argc, char* argv[]) {
    int demand;
    int experiment;
    int profit;
    double totalProfit;
```

```

    srand48( (long int) std::time(NULL) );

    totalProfit = 0.0;

    for (experiment = 0; experiment < \
        k_maxExperiment; experiment++) {
        profit = 0 - k_supply * k_costIngredients; \
        // Cost of making chocolates.

        demand = (int) std::floor((drand48() * 1000) \
            + 500);

        if (demand > k_supply)
            demand = k_supply;

        profit += demand * k_price;

        totalProfit += profit;
    }

    std::cout << "The average profit is " << \
        totalProfit / (double) k_maxExperiment << \
        std::endl;
}

```

When we run the experiment, the average profit per day is actually about \$749.36.

2.3 Conditional Probabilities

The above probabilities were **unconditional probabilities**; they depended on no other information. In reality, most probabilities change as new information comes in. This section explains the idea of **conditional probability**.

Conditional probability is written as $\Pr(A|B)$, and it represents the probability of A , given that B is true. It is defined:

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

The difference between conditional probability and unconditional probability is among the most misused and misinterpreted ideas in probability. The sections below will help you to understand conditional probability.

Example 2.8 Conditional Probability: Ferrets

Imagine three ferrets: Sasha is brown and energetic. Speedy is white and energetic. Russell is brown and mellow.

The unconditional probability of choosing an energetic ferret is $\frac{2}{3}$. But if you know that you have a brown ferret, the conditional probability that the ferret is energetic is $\Pr(\text{energetic}|\text{brown}) = \frac{\Pr(\text{energetic} \cap \text{brown})}{\Pr(\text{brown})} = \frac{1/3}{2/3} = \frac{1}{2}$.

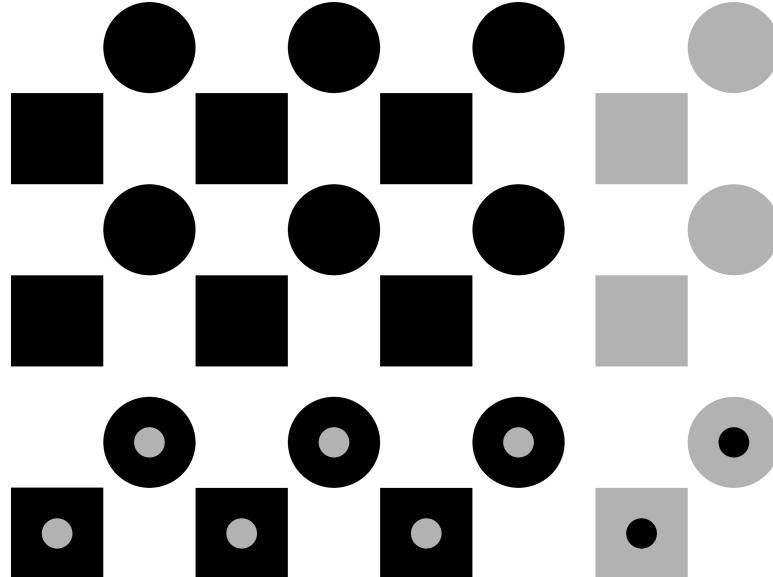
PROBLEMS: (Answers on page 275)

1. (Medium) Prove or disprove: If $\Pr(A|B) = \Pr(A)$, then $\Pr(B|A) = \Pr(B)$.
2. (Medium) Prove or disprove: $\Pr(A) = \Pr(A|B) + \Pr(A|B^c)$.

2.3.1 Independent sets

independent If $\Pr(A|B) = \Pr(A)$, then sets A and B are **independent** or **uncorrelated**. Independence usually means that A and B have no relationship.

Figure 2.2: Independent sets



Example 2.9 Independent Sets

In graph 2.2 on page 66, the sets of items with dots is disjoint to the set of items without dots, the set of squares is disjoint to the set of circles, and the set of light-colored items is disjoint to the set of dark-colored items.

Also, the set of items with dots is independent to the set of squares, the set of circles is independent to the set of light-colored items, and the set of dark-colored items is independent to the set of items without dots.

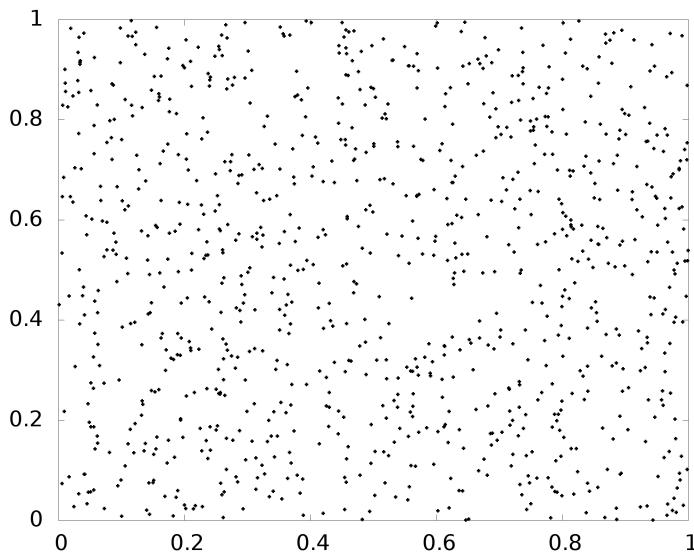
Independence is a relative idea. Two sets may be minimally correlated, or heavily correlated. For example, gas prices are heavily correlated with oil prices, and food prices are somewhat correlated with oil prices because of transportation, costs of running machinery, and costs of oil-based fertilizers.

Unless we have other reasons, statisticians tend to assume that sets are independent. This assumption isn't always accurate!

One way to understand independence is through graphs of two variables. Assume that X and Y are random numbers.

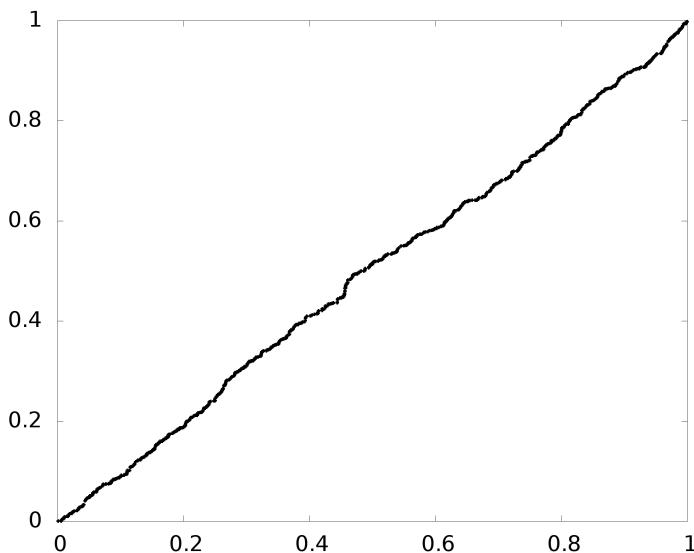
Plot X and Y on different axes. If X and Y are independent, then the choice of X will not change the value of Y , and vice-versa. An example of independent random variables is in graph 2.3.

Figure 2.3: Two independent random variables



An example of extremely correlated random variables is in graph 2.4 on page 68.

Figure 2.4: Two very correlated random variables



Both graph 2.4 and graph 2.3 on page 67 come from the same distribution. Only by comparing the two variables can one determine whether they are correlated.

The phrase “The choice of X will not change the value of Y ” might not be clear. Look at figure 2.5 on page 69: are these two variables independent or correlated?

Take two horizontal “stripes” of the graph, as done in figure 2.6 on page 69.

These two stripes are not likely to have come from the same distribution: they seem to have very different limits and medians. Therefore, these variables seem to be dependent.

However, different densities may cause variables to look dependent when they are not. When fewer points are available at an x or y range, the variables look like they have a smaller range than they do.

Look at figure 2.7 on page 70: are these two variables independent or correlated?

When you take horizontal “stripes” of the graph, it looks like the further up the graph, the smaller the range.

In this case, the two variables are uncorrelated. The difference of the range comes from the difference in the number of points. There are fewer points toward the top of the graph, and the points always tend toward the left part of the graph. This makes the graph look like a right triangle.

Figure 2.5: Does this graph show dependent or independent variables?

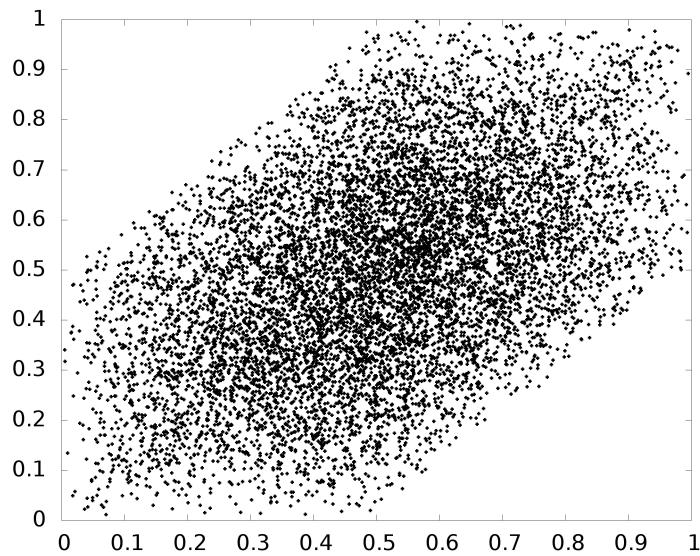


Figure 2.6: Does this graph show dependent or independent variables?

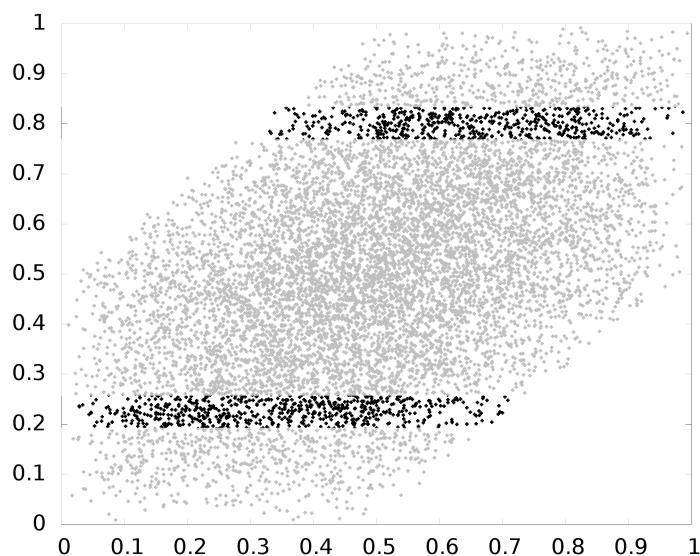


Figure 2.7: Does this graph show dependent or independent variables, take 2

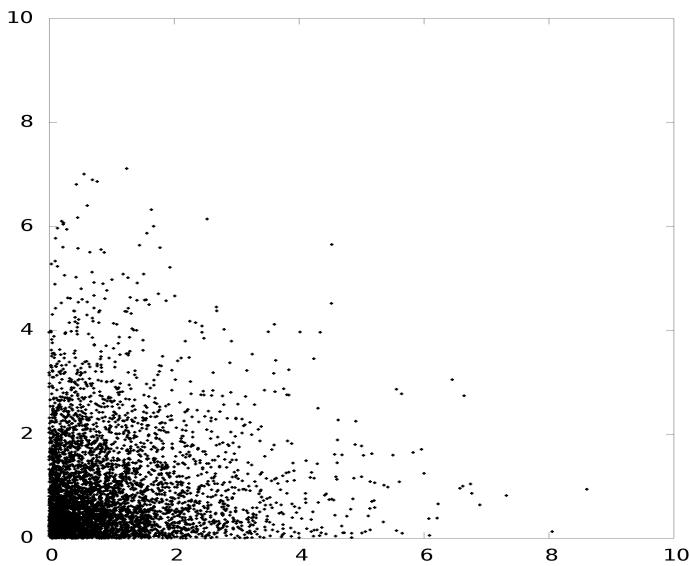
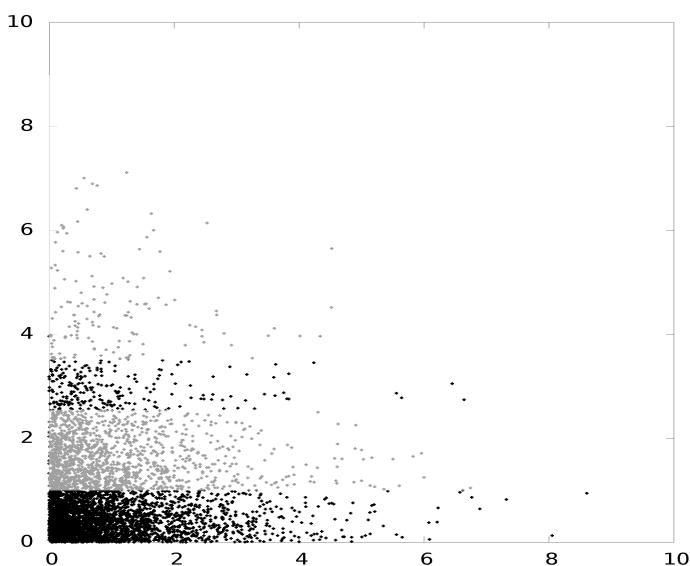


Figure 2.8: Does this graph show dependent or independent variables, take 2



One important use of independence is in testing for randomness in a random sequence generator. If the sequence is r_1, r_2, \dots, r_n , then graph every element r_i against its previous element r_{i-1} . If the sequence shows a pattern, then the sequence fails to be very random.

PROBLEMS: (Answers on page 276)

1. (Easy) If sets A and B are independent, then what is $\Pr(A \cup B)$?
2. (Easy) Would you say that the following sets tend to be independent or dependent:
 - (a) The set of people of a political party, and the set of people of an economic class.
 - (b) Whether a flipped penny lands heads, and whether a flipped nickel lands heads.
 - (c) How well a person does on a large-scale test of general knowledge, and the size of that person's foot.
3. (Medium) Some places claim that if two events are independent then they cannot be mutually exclusive (disjoint) and vice versa. Is this idea true? If so, prove it. If not, give a counterexample.
4. (Medium) An earlier version of this text said that A and B are independent if $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$. Is this definition the same as the one in the text?
5. (Medium) Examine the diagrams in graph 2.9 on page 72. Do they represent dependent or independent distributions?

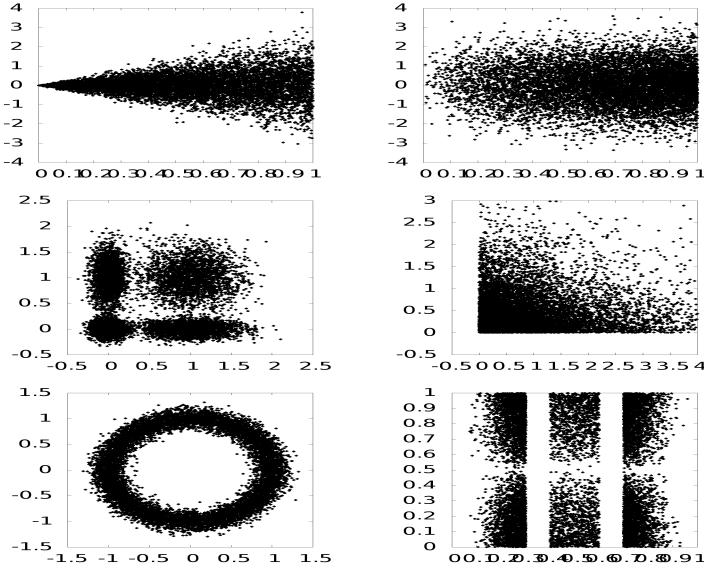
2.3.2 The Monty Hall Problem

Example 2.10 The Monty Hall Problem

In an old television game show, the host would show three doors. Behind one door was a major prize (often, a new car), and behind the other two were goats. The contestant would pick a door. The host, knowing where the prize was, would open an unchosen door, revealing a goat. The host would then allow the contestant to switch to the other unopened door. Is it better strategy to switch doors or to stay with the original door? (Assume that you want the car, not the goat.)⁷

⁷[Stafford and Webb, 2005, page 238] gives the reference for this question as "vos Savant, M. (1997) *The Power of Logical Thinking*. New York: St. Martin's Press". The true original reference for this problem is "Selvin, Steve (1975a). *A problem in probability* (letter to the editor). American Statistician 29(1):67 (February 1975)." Many ideas in this section came from *The Straight Dope* by Cecil Adams, dated 02-Nov-1990, 23-Nov-1990, and 15-Mar-1991, retrieved from <http://www.straightdope.com/classics/a3.189.html>.

Figure 2.9: Are these variables independent or dependent?



Most people reason: There are two doors left. Therefore, the chance of the car being behind either door is exactly $\frac{1}{2}$. Therefore, it makes no difference whether I change or not.

This analysis is wrong. It ignores that the host introduces new information.

The following two examples give an intuitive feel for why swapping doors makes sense.

Example 2.11 Let's Make a Second Deal!

In a television show in an alternate universe, the host would show 1,000 doors. Behind one door was a major prize (for example, a Pan Galactic Gargle Blaster), and behind the other nine hundred and ninety-nine were cheap digital watches. The contestant would pick a door. The host, knowing where the prize was, would then open nine hundred and ninety-eight doors, revealing cheap digital watches behind all of them. He would then allow the contestant to switch to the remaining unopened door. Is it better strategy to switch doors or to stay with the original door?

More people intuitively see that it's better to switch in this case. The original guess had only a $\frac{1}{1000}$ chance of being correct; swapping doors is the equivalent of choosing 999 doors at once.

Example 2.12 Let's Make a Third Deal!

In a television show in another alternate universe, the host would show two contestants three doors. Behind one door was the major prize, a textbook without dated jokes about long-dead authors and overplayed television shows. Behind the other two doors were dead parrots. One contestant, Alice, would pick a door, and the other contestant, Bob, would receive what's behind the other two doors. The host, knowing where the prize was, would then open one of Bob's doors, revealing a dead parrot. Which contestant would have the better chance of escaping long-dead jokes?

In example 2.12 on page 72, Bob gets two doors, and Alice only gets one. Since the host knows where the major prize is, he can always know which of Bob's doors to open to show a dead parrot.

PROBLEMS: (Answers on page 276)

1. (Easy) Write a program that accurately simulates one thousand runs of example 2.10 on page 71. Are the switch-don't switch probabilities closer to $\frac{1}{3}$ and $\frac{2}{3}$ or closer to $\frac{1}{2}$ and $\frac{1}{2}$?
2. (Medium) Give a mathematical proof that the contestant in the original Monty Hall problem is better off switching.
3. (Medium) Monty Hall is feeling generous, and he passes three envelopes to Alice, Bob, and you. In one of the envelopes is a coupon for a year's supply of chocolate. Bob opens his envelope, and it is empty. Alice offers to trade her envelope with you. Are you better off trading or not? If this problem's probabilities are different than the usual Monty Hall problem, then why are they different?

2.3.3 States and conditional probability

Here's an experiment to try:

Example 2.13 One hundred coins

1. Take a set of one hundred coins.
 2. Make a fair flip of the hundred coins.
 3. Set aside all coins that are heads. Do not touch them any longer.
 4. If no coins remain (are tails), then stop.
 5. Otherwise, make a fair flip of the remaining coins (those that were tails).
 6. Return to step 3.
-

I did this experiment 10,000,000 times and only 2 of them needed more than 30 flips. On average, it takes about 9 flips to flip all coins.

This experiment clearly shows the difference between conditional and unconditional probability. The unconditional probability that one hundred coins would all flip to heads within thirty flips is $\frac{30}{2^{100}}$, a vanishingly small number.

state However, because we maintain a **state** between the different flips, the probability is not unconditional.

PROBLEMS: (Answers on page 278)

1. (Easy) Write code to perform example 2.13 on page 73.

If the idea of state seems obvious, visit any website promoting Intelligent Design. Every one will talk about the tiny probability of the world that we live in, and most will give rough mathematical chances of, say, bringing together one hundred specific amino acids in a specific order.

The unconditional chance of a specific set of one hundred amino acids coming together is tiny. However, if portions of a set of amino acids are more stable than sequences that are not in that set, then the conditional chance of getting that set of amino acids increase. Evolution acts like the one hundred coin experiment.

Sidebar 2.1: The problem of intelligent design

2.3.4 Bayes' Formula

A subtle but extremely frequent misuse of statistics comes from reversing the condition and the result. For example, “99% of people without teeth eat apple sauce.” is different than “99% of people who eat apple sauce have no teeth.”

Bayes' Formula

Bayes' Formula allows us to compute the probability of these reversals. It comes from the definition of conditional probability.

Conditional probability was defined on page 65. Let's do a bit of math:

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)} \quad (2.1)$$

Therefore

$$\Pr(A \cap B) = \Pr(A|B) \Pr(B) \quad (2.2)$$

Since $A \cap B = B \cap A$

$$\Pr(A \cap B) = \Pr(B|A) \Pr(A) \quad (2.3)$$

By substituting (2.3) into (2.1), we get:

$$\Pr(A|B) = \frac{\Pr(B|A) \Pr(A)}{\Pr(B)} \quad (2.4)$$

Since $S = A \cup A^c$ is the whole sample space, $B = B \cap S = (B \cap A) \cup (B \cap A^c)$. Therefore,

$$\Pr(B) = \Pr(B \cap A) + \Pr(B \cap A^c) \quad (2.5)$$

Substituting (2.2) and (2.3) into (2.5),

$$\Pr(B) = \Pr(B|A) \Pr(A) + \Pr(B|A^c) \Pr(A^c) \quad (2.6)$$

Finally, substituting (2.6) into the (2.1), we get:

$$\Pr(A|B) = \frac{\Pr(B|A) \Pr(A)}{\Pr(B|A) \Pr(A) + \Pr(B|A^c) \Pr(A^c)} \quad (2.7)$$

This formula is the binary version of Bayes' Formula.

Example 2.14 Bayes' Formula: Disease detection

Imagine that a *caudal protrusion* exists in 1 in 100,000 people. Luckily, a test exists that is 99% accurate in detecting *caudal protrusions*. Zell tests positive for *caudal protrusion*. What is the chance that Zell actually has one?

Let A represent that Zell has a *caudal protrusion*, and let B be that Zell tests positive for *caudal protrusion*. Then:

$$\begin{aligned}\Pr(A) &= 0.00001 \\ \Pr(A^c) &= 0.99999 \\ \Pr(B|A) &= 0.99 \\ \Pr(B|A^c) &= 0.01\end{aligned}$$

Substitute these values into the binary version of Bayes' formula:

$$\Pr(A|B) = \frac{0.99 \times 0.00001}{0.99 \times 0.00001 + 0.01 \times 0.99999} = \frac{0.000099}{0.000099 + 0.0099999} = \frac{1}{102}$$

Therefore, even though Zell passes the test for *caudal protrusion*, he has only a 1 in 102 chance of having a tail.

Another way to think about the problem is by counting. Every million people can be divided into four groups based on whether they test positive or negative, and whether they have the *caudal protrusion*.

	<i>Caudal protrusion</i>	No <i>caudal protrusion</i>	Total
Tests Positive	99	9999	10,098
Tests Negative	1	989,901	989,902
Total	100	999,900	1,000,000

If there is no *caudal protrusion*, then 99% of the subjects will test false. If there is a *caudal protrusion*, then 99% of the subjects will test true. Even still, most positives are false positives.

PROBLEMS: (Answers on page 279)

1. (Very Easy) In a future world, the police have a record of the DNA of every one of the 10,000,000,000 people in the world. A murder occurs. Fortunately, the killer left a hair sample at the crime scene. The police run the DNA, find a match, and on the strength of that single piece of evidence, bring in a suspect. The suspect says that he's completely innocent, and was nowhere near the crime. At the trial, the police say that only one person in a million would match the DNA, therefore, he's obviously guilty. How should the defense respond?
2. (Easy) You have a mail filter to separate spam (unsolicited junk email) from real email. The filter has discovered that 60% of your mail is spam. The filter also knows that 70% of spam has rows of exclamation points, but only 10% of real mail have them. It receives an email with a row of exclamation points. What is the chance that the mail is spam?
3. (Easy) According to the Wall Street Journal⁸, a company plans to measure biometric responses in airline travelers.

The article reports that when passengers respond to certain questions, people with terrorist intent have different physiological responses than people without those intents.

The company states that its goal is to catch 90% of people with terrorist intent — that is, to have a 10% false-negative rate — while inconveniencing no more than 4% of innocent travelers.

According to the Bureau of Transportation Statistics⁹, 622 million passengers flew in the United States in 2001. Among those passengers were nineteen terrorists. Assume that the improved system existed in 2001. If the system works exactly as advertised, and it points out a person as a potential terrorist, what is the chance that the person is a terrorist?

4. (Easy) Find a website or news story that misuses Bayes' Formula.

2.3.5 Game Theory

Game theory happens when there are two (or more) players, each of whom have a choice between two (or more) actions, and a result occurs depending on both people's choices.

Some parts of graph theory require probability and the expected value. This example is heavily reworked from [Casti, 1992, pages 10-12]:

⁸Which Travelers Have 'Hostile Intent'? Biometric Device May Have the Answer by Jonathan Karp and Laura Meckler, August 14, 2006, Page B1. Retrieved from http://online.wsj.com/public/article/SB115551793796934752-2hgveyRtDDtssKozVPmg6mg6RAAa_w_20070813.html?mod=3dtff_main_tff_top.

⁹http://www.bts.gov/press_releases/2002/bts011_02.html

In soccer, a penalty kick is awarded against a team that commits an offense while the ball is in play.¹⁰ When a penalty kick happens, only two players are (usually) involved: the kicker and the goalkeeper; all other players must be at least ten yards away.¹¹

The kicker has two strategies: hit the ball high and center, which is faster; or hit the ball to either side, which is slower. Because a penalty shot is taken very close to the goal, the goalkeeper must decide before the shot whether she will block high and close to the center or to either side. To keep this problem simple, assume that those are the only choices for both sides.

In a hypothetical game¹², assume that the chances for making a goal are represented by table 2.3.5.

	Goalie left (g_l)	Goalie Center (g_c)	Goalie Right (g_r)
Kick to left (k_l)	0.5	0.8	0.9
Kick to center (k_c)	0.7	0.6	0.7
Kick to right (k_r)	0.8	0.7	0.4

Table 2.1: Hypothetical chances for scoring a goal

Without any prior probability, it is best for the kicker to kick to the left: he has, on average, a $\frac{2}{3}$ chance of scoring a goal. But if the kicker always goes to the left, then the goalie can also always go to the left, keeping the chance of scoring to 0.5. That's not the best strategy for the kicker. Probability and the expected value come into play.

If the goalie has an k_l chance of kicking to the left, a k_c chance of kicking to the center, and a k_r chance of kicking to the right, then we have three equations:

$$\begin{aligned} E[\text{kicker}] &= 0.5k_l + 0.7k_c + 0.8k_r \text{ if } g_l \\ E[\text{kicker}] &= 0.8k_l + 0.6k_c + 0.7k_r \text{ if } g_c \\ E[\text{kicker}] &= 0.9k_l + 0.7k_c + 0.4k_r \text{ if } g_r \end{aligned}$$

We can combine them into one expected value:

$$\begin{aligned} E[\text{kicker}] &= (0.5k_l + 0.7k_c + 0.8k_r) g_l + \\ &\quad (0.8k_l + 0.6k_c + 0.7k_r) g_c + \\ &\quad (0.9k_l + 0.7k_c + 0.4k_r) g_r \end{aligned}$$

The kicker wants to maximize this equation; the goalie wants to minimize it.

¹⁰<http://www.fifa.com/worldfootball/lawsofthegame.html>, page 40

¹¹ibid

¹²The real statistics for penalty kicks can be found at <http://www.scienceofsocceronline.com/2009/04/penalty-kicks-by-numbers.html>

This problem is best solved with “linear programming”, whose methods are beyond the scope of this book. If you’re curious, the linear programming setup is:

Maximize P subject to:

$$\begin{aligned} 0.5k_l + 0.8k_c + 0.9k_r - P & \geq 0 \\ 0.7k_l + 0.7k_c + 0.4k_r - P & \geq 0 \\ 0.8k_l + 0.7k_c + 0.4k_r - P & \geq 0 \\ k_l + k_c + k_r & = 1 \end{aligned}$$

The answers are:

$$\begin{array}{ll} k_l = 0.25 & g_l = 0.5 \\ k_c = 0.5 & g_c = 0.25 \\ k_r = 0.25 & g_r = 0.25 \end{array}$$

In this case, the kicker has a 67.5% chance of making a goal.

2.4 Philosophy, Probability, and Information Theory

Most sections in this book are important. This section isn’t one of them.

Probability is a very strange concept.

Imagine a fairly-weighted coin. Before it’s flipped, there’s a 50% chance that the coin will turn out heads and a 50% chance that it will turn out tails. Probability exists at this point.

After it’s flipped, the coin is either heads or tails. Probability no longer exists at this point.

What other process disappears because its results are observed?

I prefer to think of probability as an inverse of information.

Re-imagine the fairly-weighted coin. Before it’s flipped, we lack information about its final state. After it’s flipped, we gain a bit of information: whether the coin is heads or tails.

Information theory gives a way to convert from a set of probabilities to a measurement of surprise.¹³

Assume that \Pr is a probability function on the set S . Then the uncertainty for any $s \in S$ is defined as $u(s) = -\log(\Pr(s))$ and the average uncertainty for the set as a whole is $H = -\sum_{s \in S} \Pr(s) \log(\Pr(s))$. When the base for this logarithm is 2, the probability is called a **bit**.

In the coin flip example, there are two possible results:

$$u(\text{heads}) = -\log_2 \left(\frac{1}{2} \right) = 1$$

¹³This section’s information is based on <http://www.ccrnp.ncifcrf.gov/~toms/paper/primer/>.

$$u(\text{tails}) = -\log_2 \left(\frac{1}{2} \right) = 1$$

So discovering the result of a flip gains one bit of information.

Knowing this philosophical point won't matter much for the rest of the book.

Chapter 3

Distributions

3.1 Introduction

Every C++ programmer knows what is a random number and how to generate thousands of them¹:

```
std::srand48( (long int) std::time( NULL ) );  
  
for ( i=0; i<NUM_RANDOM; i++ )  
    random_numbers[ i ] = drand48();
```

For a programmer, a random number is chosen from a range, where every number is equally likely.

Statisticians have a different definition, given on page 54. Every random variable has a distribution. Most distributions have different likelihoods for different numbers. `drand48()` is just one distribution, called the uniform distribution. We've seen a non-uniform distribution in example 2.5 on page 60. Another non-uniform distribution is the number of heads that happen when one hundred coins are tossed.

Different distributions show different behaviors. When someone flips ten coins, two heads and eight tails show up less frequently than five heads and five tails. Adults may range from 81 centimeters tall² to 272 centimeters tall³, but people are not equally likely among all of those heights.

A distribution gives the likelihood of choosing any particular random number. In theory, many distributions could choose any number, no matter how large or how small.

This chapter lists some important distributions, when they are used, and how to create them.

¹Good programmers know that `drand48()` makes pseudo-random numbers: numbers that look random but aren't. Really good programmers know tests that measure how good or how bad a random number is. Patience. You'll learn about them in section ?? on page ??

²Verne Troyer

³Robert Wadlow

3.2 CDFs and PDFs

Cumulative Distribution Functions The probability for a distribution can be described in many ways. One way is through **Cumulative Distribution Functions**, or **CDFs**. The CDF is the probability that a random number will be less than a given limit. A CDF is defined as $F = \Pr(X < x)$, where X is a random number and x is a given limit. If F is a CDF, then it has the following properties:

1. $F(-\infty) = 0$
2. $F(\infty) = 1$
3. If $a \leq b$ then $F(a) \leq F(b)$

I will write CDFs with capital letters.

Example 3.1 Cumulative distribution function on the three-marbles example

Assume that we have the same distribution on the three marbles that we did on page 60: For each marble, you are $\frac{2}{3}$ likely to choose it and $\frac{1}{3}$ likely not to choose it. Let your random variable, X , be the number of balls in your sample. Then its CDF is the function F :

...

$$F(-1) = 0$$

$$F(0) = 0$$

$$F(1) = \frac{1}{27}$$

$$F(2) = \frac{7}{27}$$

$$F(3) = \frac{19}{27}$$

$$F(4) = 1$$

$$F(5) = 1$$

...

Example 3.2 Cumulative distribution on $[0, 1]$

Assume that each number in the range $[0, 1]$ is equally likely, and that your random variable, X , is the number chosen. Then its CDF is the function F :

$$\begin{aligned} F(x) &= 0 && \text{if } x \leq 0 \\ F(x) &= x && \text{if } 0 \leq x \leq 1 \\ F(x) &= 1 && \text{if } 1 \leq x \end{aligned}$$

For any given number x between 0 and 1 the chance that a chosen random number would be less than x is x .

The \Pr function defined in section 2.2.5 on page 59 for discrete sets is also called the **probability distribution function** or PDF⁴. For any distribution, if F is the CDF and f is the PDF, then $F(n) = \sum_{x < n} f(n)$.

probability distribution function

I will write PDFs as lower-case functions.

Probability density functions have one additional rule over the Kolmogorov functions:

$$\Pr(-\infty) = \Pr(\infty) = 0$$

To be confusing, probability density functions are also called PDFs. For any distribution, if F is the CDF and f is the PDF, then $F(n) = \int_{-\infty}^n f(x) dx$.

A bit of thinking in Calculus will confirm that the same definition won't work for continuous and discrete distributions. In a continuous distribution, the chance of any individual number is infinitesimal.⁵

On the other hand, in a continuous distribution, the probability that a random number will be within an interval $[a, b]$ can be measured. If f is a probability density function, then $\Pr(x \in [a, b]) = \int_a^b f(x) dx$.

Unlike a discrete PDF, a continuous PDF can have values greater than 1.

PROBLEMS: (Answers on page 281)

1. (Easy) If you are given a discrete CDF, F , how do you derive the related discrete PDF, f ?
2. (Medium) Show that if \Pr meets the Kolmogorov Rules for Discrete Sets, given on page 60, then $f(x) = \Pr(X = x)$ meets the above three rules.
3. (Easy) If you are given a continuous CDF, F how do you derive the related continuous PDF, f ?

⁴Not related to Adobe's Portable Document Format!

⁵What is the probability that your random number generator will come up with exactly 0.4077938799?

3.3 Distribution

Statistical questions require more than just the CDF of a distribution. Distributions have other important values.

The following functions will be defined for every distribution:

density The PDF for the random number. `operator()` is defined as another name for the `density` function.

cumulative The CDF for the random number. The probability that a random number from this source will be less than x .

mean The mean of this distribution.

quantile The quantile is the inverse of `density`: If $q = \text{density}(p)$, then $p = \text{quantile}(q)$. On average, q is the fraction of random numbers less than `quantile(q)`.

random Choose a random number according to this distribution.

standardDeviation The standard deviation of this distribution

negativeInfinity A number smaller than the smallest reasonable number that this distribution will reach.

positiveInfinity A number larger than the largest reasonable number that this distribution will reach.

```
typedef double probability;

template <typename TBase, typename TParameter>
class Distribution : public IFunctor<TBase>
{
public:
    virtual ~Distribution() {}
    virtual probability density(TBase x) const = 0;
    virtual probability operator()(TBase x) const { \
        return density(x); }
    virtual probability cumulative(TBase x) const;
    virtual TParameter mean() const;
    virtual TBase mode() const;
    virtual TBase quantile(probability y) const;
    virtual TBase random() const;
    virtual TParameter standardDeviation() const;

protected:
    virtual TBase negativeInfinity() const;
    virtual TBase positiveInfinity() const;
};
```

Notice that `density` is a pure-virtual method. It must be implemented by a child class.

This chapter has three generic implementations for `Distribution`: one for data distributions, one for discrete distributions, and one for continuous distributions.

3.4 Data distribution

Some times, the only information that you have about a distribution is the data that came from it. You know nothing about how the numbers were made. In this case, a **data distribution**⁶ is an appropriate distribution to use.

data distribution

Other times, you have complete information about a well-defined, naive distribution. The kinds of probabilities found in section 2.2.4 on page 56 are also excellent for this kind of distribution.

Example 3.3 Data Distribution

In a role-playing game, Yang The Distasteful aims his Wand Of Uncertain Power against the Undead Ducks attacking his group. The number of Undead Ducks hit depends on the number rolled on a twenty-sided die:

Number rolled	Number Killed
1-5	0
6-10	1
10-14	2
15-18	3
19	4
20	5

For this table, you would generate a `Data` class with the numbers 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 5.

A data distribution always fits the data given to it, and it generates similar data. If you know nothing else about data, it is useful.

It is a wrapper around `Data`, the class defined on page 26

```
DataDistribution::DataDistribution(const Data<double> <
    data) {
    srand48((long int) std::time(NULL));

    m_data = data;
}
```

⁶It's also often called a "parameterless distribution". [Kleijnen and Groenendaal, 1992, page 34] calls it a "discrete distribution". I use that term for something else.

```

double DataDistribution::density(double x) const {
    return (double) m_data.count(x) / (double) m_data.\
        size();
}

double DataDistribution::mean() const {
    return m_data.mean();
}

double DataDistribution::quantile(double q) const {
    int pis;
    if (q < 0.0 || q >= 1.0) {
        throw new std::out_of_range("Quantile must be \
            between_0_and_1(not_inclusive).");
    }
    pos = (int) floor(q * m_data.size());
    return m_data[pos];
}

double DataDistribution::random() const {
    int pos;

    pos = (int) floor(drand48() * m_data.size());
    return m_data[pos];
}

double DataDistribution::standardDeviation() const {
    return m_data.standardDeviation();
}

double DataDistribution::negativeInfinity() const {
    return m_data[0];
}

double DataDistribution::positiveInfinity() const {
    return m_data[m_data.size() - 1];
}

```

```
}
```

3.5 Discrete Distributions

Strictly, any distribution with a discrete range is a discrete distribution. This book will concentrate on discrete distributions with a range of `int`; these classes will all derive from `Distribution<int, double>`.

Distributions often have two sets of parameters: one set in the constructor, and the other set in the PDF or CDF call. Mathematically, a PDF for a distribution is written $f(a; b_1, b_2, \dots, b_n)$. a is the parameter in the function call, and b_1, b_2, \dots, b_n are parameters of the constructor. For these distributions, there will be only one parameter in the function call.

PDFs often are the most convenient way to express discrete distributions. They are the probability function of page 60.

3.5.1 Implementation

Before we give specific discrete distributions, let's create a generic discrete distribution. This generic discrete distribution will work with almost any PDF, even if it does so slowly. These generic algorithms will often be specialized later.

The infinities

Many formulas in this chapter depend on taking the sum of all possible values of a function. Summing an infinite number of values would take infinite time. However, most distributions are exceedingly unlikely to take values below or above certain values.

This method computes a value that might serve as 'negative infinity' for these functions.

This code makes the strong assumption that as soon as it finds an x such that $f(x) \leq \text{densityMinimum}$, then for all $y \leq x$, it is also true that $f(y) \ll \text{densityMinimum}$.

```
const int k_xSearchStartNegInf = 0;
const int k_xSearchNegInfAdd = -1;
const double k_densityMinimum = 1E-10;

template <>
int Distribution<int, double>::negativeInfinity() const
{
    int xNegativeInfinity = k_xSearchStartNegInf;

    while (k_densityMinimum <= density(xNegativeInfinity))
        xNegativeInfinity += k_xSearchNegInfAdd;
```

```

return xNegativeInfinity;
}

```

We make the same computation for positive infinity.

```

const int k_xStartSearchPosInf = 0;
const int k_xSearchPosInfAdd = 1;
template <>
int Distribution<int, double>::positiveInfinity() const
{
    int xPositiveInfinity = k_xStartSearchPosInf;

    while (k_densityMinimum <= density(xPositiveInfinity))
        xPositiveInfinity += k_xSearchPosInfAdd;

    return xPositiveInfinity;
}

```

Cumulative

The `cumulative` is the CDF. In other words, it is the average proportion of random numbers from the distribution that are below a given number.

If $f(x)$ is the PDF, then the CDF is $F(y) = \sum_{x=-\infty}^y f(x)$.

```

template <>
double Distribution<int, double>::cumulative(int x) const
{
    int xCurrent;
    double ySum = 0.0;

    for (xCurrent = negativeInfinity(); xCurrent < x; xCurrent++)
        ySum += density(xCurrent);

    return ySum;
}

```

Mean

If $f(x)$ is the PDF, then the mean is $\sum_{x=-\infty}^{\infty} xf(x)$.

```

template <>
double Distribution<int, double>::mean() const {
    int xCurrent;
    int xNegInf = negativeInfinity();
    int xPosInf = positiveInfinity();

```

```

double ySum = 0.0;

for (xCurrent = xNegInf; xCurrent <= xPosInf;
      xCurrent++)
    ySum += xCurrent * density(xCurrent);

return ySum;
}

```

Mode

Remember from page 28, if you want a single value to “best” represent a distribution, you have three choices: the mean, the median, and the mode. The mode is the least-frequently used of the three representations.

The mode has the maximum probability.

```

template <>
int Distribution<int, double>::mode() const {
    int xCurrent;
    int xMode;
    int xNegInf;
    int xPosInf;
    double yDensity;
    double yModeDensity;

    xNegInf = negativeInfinity();
    xPosInf = positiveInfinity();

    yModeDensity = 0.0;
    for (xCurrent = xNegInf; xCurrent <= xPosInf; \x
        xCurrent++) {
        yDensity = density(xCurrent);

        if (yModeDensity < yDensity) {
            xMode = xCurrent;
            yModeDensity = yDensity;
        }
    }

    return xMode;
}

```

Quantile

`quantile` is an inverse of `cumulative`. If $q = \text{cumulative}(p)$, then $p = \text{quantile}(q)$. On average, the q proportion of random numbers will be less than $\text{quantile}(q)$.

Because this function calls `cumulative` many times, this method is slow. However, it's easy to understand: it's an interpolated search.

```

const double k_epsilon = 0.000001;

template <>
int Distribution<int, double>::quantile(double y) const {
    {
        int xLowerBound;
        int xTest;
        int xUpperBound;
        double yLowerBound;
        double yTest;
        double yUpperBound;

        if (y < 0.0 || 1.0 < y)
            throw std::out_of_range("The quantile input must be between 0 and 1 (inclusive).");
        else if (y == 0.0)
            return negativeInfinity();
        else if (y == 1.0)
            return positiveInfinity();

        // Since computers are not perfect, we occasionally
        // need to give a touch of help...

        y += k_epsilon;

        xLowerBound = negativeInfinity();
        yLowerBound = 0.0;
        xUpperBound = positiveInfinity();
        yUpperBound = 1.0;

        while (1 < xUpperBound - xLowerBound) {
            xTest = (xUpperBound + xLowerBound) / 2;
            yTest = cumulative(xTest);

            if (y <= yTest) {
                xUpperBound = xTest;
                yUpperBound = yTest;
            } else {
                xLowerBound = xTest;
                yLowerBound = yTest;
            }
        }
    }
}
```

```

if (yTest <= y)
    return xTest;
else
    return xLowerBound;
}

```

Random

The `random` function chooses a random number from this distribution.

This implementation uses the **Quantile Method**.⁷

Quantile Method

The quantile of a uniformly-distributed random number⁸ has the same distribution as the distribution that the `quantile` came from.

Informally, the `quantile` function maps from a uniformly-distributed [0, 1] to the domain of the random numbers. So, the range from `quantile(0.35)` to `quantile(0.45)` will occur 10% of the time.

```

template <>
int Distribution<int, double>::random() const
{
    double dUniformRandom;

    dUniformRandom = (double) drand48();
    return quantile(dUniformRandom);
}

```

Standard deviation

If $f(x)$ is the density and μ is the mean, then the standard deviation is $\sqrt{\sum_{x=-\infty}^{\infty}(x - \mu)^2 f(x)}$.

```

template <>
double Distribution<int, double>::standardDeviation() \
    const
{
    double xMean;
    int x;
    int xNegInf;
    int xPosInf;
    double ySum;

    xNegInf = negativeInfinity();
    xPosInf = positiveInfinity();

    xMean = mean();
}

```

⁷This name comes from [Kundu and Basu, 2004, page 27]

⁸That's mathematical talk for something like `drand48()`

```

ySum = 0.0;
for (x = xNegInf; x <= xPosInf; x++)
    ySum += (x - xMean) * (x - xMean) * density(x);

return sqrt(ySum);
}

```

PROBLEMS: (Answers on page 281)

1. (Medium) `cumulative()` is slow. But it can be sped up by memoization: storing computed results in a class variable. The first time that `cumulative` is called, compute all cumulative values from `negativeInfinity()` to `positiveInfinity()`. Always return the saved value. Implement this idea.

3.5.2 Binomial distribution

Bernoulli trials

A Bernoulli trial is a single test with a known probability, p . It has two possible outcomes:

probability	value
p	1
$1 - p$	0

Example 3.4 Bernoulli trial examples

- Whether a weighted, flipped coin shows heads or tails.
 - Whether a voter will choose to vote for a particular candidate.
 - Whether a shopper will buy a particular product.
 - Whether a music player on random-play will pick a particular song next.
 - Whether a professor will use the word “conceptual” in a given five minutes.
 - Whether a school cafeteria will serve casserole during a given week.
-

By itself, a Bernoulli trial is very boring. But it forms the basis of interesting distributions:

- The sum of n Bernoulli trials is the binomial distribution.
- The number of Bernoulli trials until the first success is the geometric distribution.
- The number of Bernoulli trials until the n th success is the negative binomial distribution.

PROBLEMS: (Answers on page 282) For the following problems, the Bernoulli trial has probability p .

1. (Easy) What is the mean of this Bernoulli trial?
2. (Medium) What is the standard deviation of this Bernoulli trial?

Binomial distribution

Parameters	$1 \leq n$ is the number of trials (integer). $0 \leq p \leq 1$ is the probability (real)
Support	$0 \leq y \leq n$ are the number of successes
Density	$f(x; n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$
Mean	np
Standard Deviation	$\sqrt{np(1 - p)}$

Properties 3.1: Properties of the binomial distribution

The binomial distribution happens whenever:

1. There are n , a fixed number, of identical trials.
2. Each trial either succeeds or fails.
3. The chance of success for each trial is p .
4. Each trial is independent of every other trial.
5. The result is the number of successes.

A binomial distribution can be thought of as the sum of n Bernoulli trials, each of probability p .

Example 3.5 Examples of binomial distributions

- The number of students in a class of 30 that are more than 180 cm tall.
- Among three six-sided dice, the number of dice that rolled 5 or 6.
- Among a group of fifty programmers, the number who have used Lisp professionally.
- Among one thousand voters, the number who choose a particular candidate.
- Among one hundred photographs, the number that have faces.

- Among ten equally-weighted coins, the number that flipped heads.
 - Among 100 shoppers, the number that choose to buy a particular product.
 - The number of times a professor uses the word “conceptual” during a thousand word lecture.
 - The number of times a school cafeteria serves casserole during a given week.
-

If we have n trials and a probability p of success in each trial, the probability that we get x successes is $\text{binom}(n, p, x) = \binom{n}{x} p^x (1 - p)^{n-x}$, where $\binom{n}{x}$ is the combination function, as defined on page 24.

To translate binomials to code, we need a constructor:

```
Binomial::Binomial(int trials, double probability)
{
    srand48((long int) std::time(NULL));

    if (0.0 <= probability && probability <= 1.0)
        m_probability = probability;
    else
        throw std::out_of_range("Error: The probability must be between 0.0 and 1.0.");

    if (0 < trials)
        m_trials = trials;
    else
        throw std::out_of_range("Error: The number of trials must be strictly positive.");
}
```

The only other required function is the PDF:

```
double Binomial::density(int x) const
{
    if (x < 0 || m_trials < x)
        return 0.0;

    return combinations(m_trials, x) * std::pow(
        m_probability, x) *
        std::pow((1 - m_probability), (m_trials - x));
}
```

(The code for `combinations` can be found on page 24.)

Although the default functions work, the program will be much faster with specialized functions.

Figure 3.1: Binomial distributions: probability = 0.3, 0.5, 0.7; size = 20.

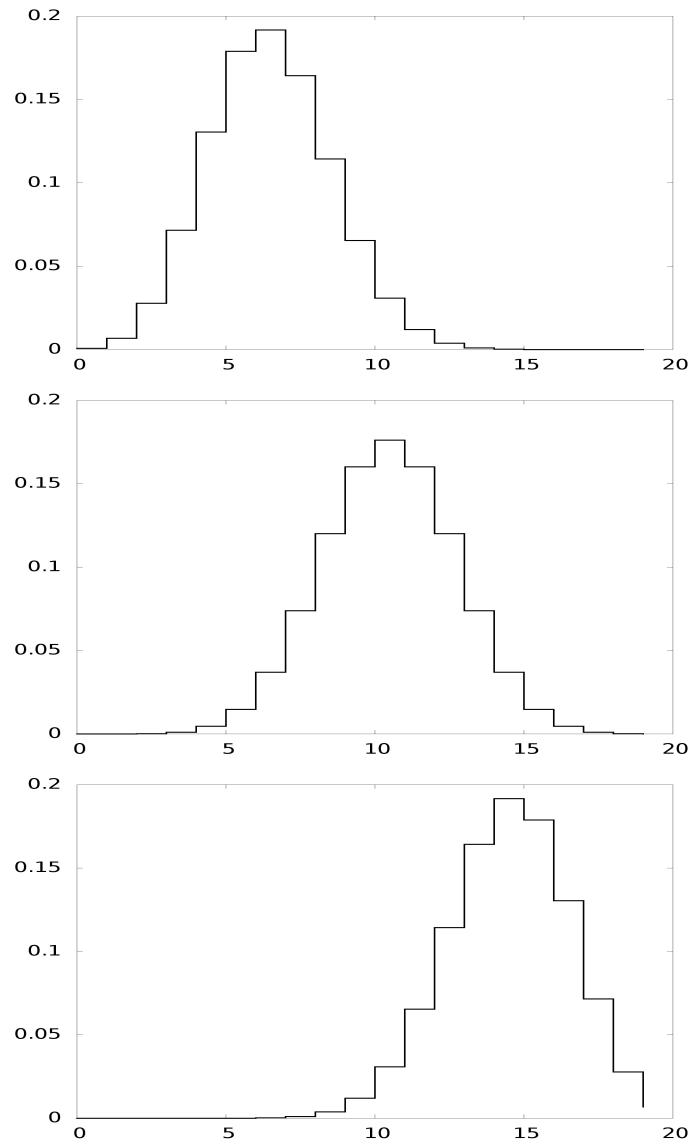
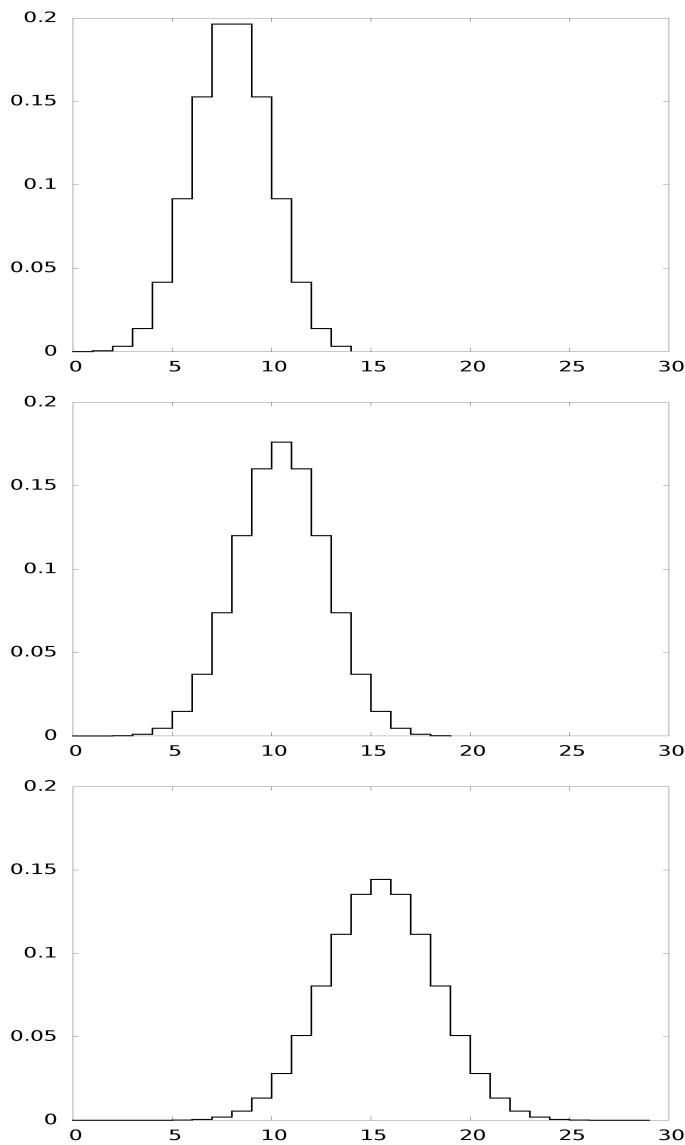


Figure 3.2: Binomial distributions: probability = 0.5; size = 15, 20, 30.



The mean and standard deviation functions can be created from the formula given on page 93.

```
double Binomial::mean() const
{
    return m_probability * m_trials;
}

double Binomial::standardDeviation() const
{
    return sqrt(m_probability * m_probability * m_trials\_
        );
}
```

I won't write out these functions again for discrete distributions.

The binomial can be thought of as the sum of many Bernoulli trials. That idea can be used to choose a random, binomial number:

```
int Binomial::random() const
{
    int trial;
    int sum = 0;

    for (trial = 0; trial < m_trials; trial++)
        if (drand48() < m_probability)
            sum++;

    return sum;
}
```

Finally, we know the largest and smallest values that will ever occur:

```
int Binomial::negativeInfinity() const
{
    return 0;
}

int Binomial::positiveInfinity() const
{
    return m_trials + 1;
}
```

Example 3.6 Roulette example

Becky has \$10 to bet on a roulette wheel. Roulette wheels have 38 equally likely slots. She's not very gambling-savvy, so with every spin, she bets one dollar on her lucky number, 17. If the ball lands on her number, she gets \$36 back as a payout. For her ten dollars, how much does she get back?

A naïve answer is $\$10 \times \frac{36}{38} \approx \9.47 . Although she'll get \$9.47 back on average, that's not an actual return that she could get. She can only get \$0, \$36, \$72, or a multiple of \$36 back.

`Binomial bin(10, 1.0 / 38.0);` constructs the roulette wheel, and `bin.random() * 36.0` returns how many dollars she won.

The exact probabilities of each win can be computed with:

```
for (i = bin.negativeInfinity(); i < bin.\  
    positiveInfinity(); i++)  
    chance[i * 36] = bin.distribution(i);
```

Her exact chances of each outcome are in figure 3.3.

Figure 3.3: The exact probabilities for Becky's wins

Value	Probability
0	0.765916
36	0.207004
72	0.025176
108	0.001815
144	0.000086
180	0.000003
216	0.000000
252	0.000000
288	0.000000
324	0.000000
360	0.000000

PROBLEMS: (Answers on page 282)

1. (Easy) People of Hollyweird marry other people at random. Each time they marry, there's a 10% chance that true love will blossom, and that the marriage will last. What's the chance that a person of Hollyweird will find true love on her third marriage?
2. (Easy) A used-car salesman knows that 25% of his cars are duds that will die within a month. He sells ten cars one day. What is the chance that two or fewer of the cars he sold that day are duds?

3.5.3 Geometric distribution

Parameters	$0 \leq p \leq 1$ is the probability (real)
Support	$1 \leq x$ are the number of trials until first success
Density	$f(x; p) = (1 - p)^{x-1} p$
Mean	$\frac{1-p}{p}$
Standard Deviation	$\sqrt{\frac{1-p}{p}}$

Properties 3.2: Properties of the geometric distribution

The Binomial distribution asks how many successes happen in a given set of trials. The geometric distribution answers a different question: How many trials do you need until you reach the first success?

Let X be a random number representing the number of trials until the first success. If p is the probability of a success, and x is the number of trials until a success, then $\Pr(X = x) = (1 - p)^{x-1} p$. If the first success took place at the x th trial, then there were $x - 1$ failures, each with probability $1 - p$, and one success with probability p .

Example 3.7 Examples of geometric distribution

- The number of rolls of a six-sided die until it shows a 5 or a 6.
- The number of flips of a weighted coin until it shows heads.
- The number of shoppers that pass until one buys a particular product.
- The number of minutes until a professor uses the word “conceptual”.
- The number of pages in a book until the phrase “crunchy bacon” is used.
- The number of trigger pulls in a game of Russian roulette.

- The number of days between servings of casserole in a school cafeteria.
-

The only parameter in the class is double `m_probability`.
The `density` function is easy to write:

```
double Geometric::density(int x) const
{
    if (x < 1)
        return 0.0;

    return std::pow(1.0 - m_probability, x - 1) * \
        m_probability;
}
```

The `random` function comes from the definition of the Geometric distribution.

```
int Geometric::random() const
{
    int tries = 0;
    double value;

    do {
        value = drand48();
        tries++;
    } while (value > m_probability);

    return tries;
}
```

The cumulative function can be derived from the density.

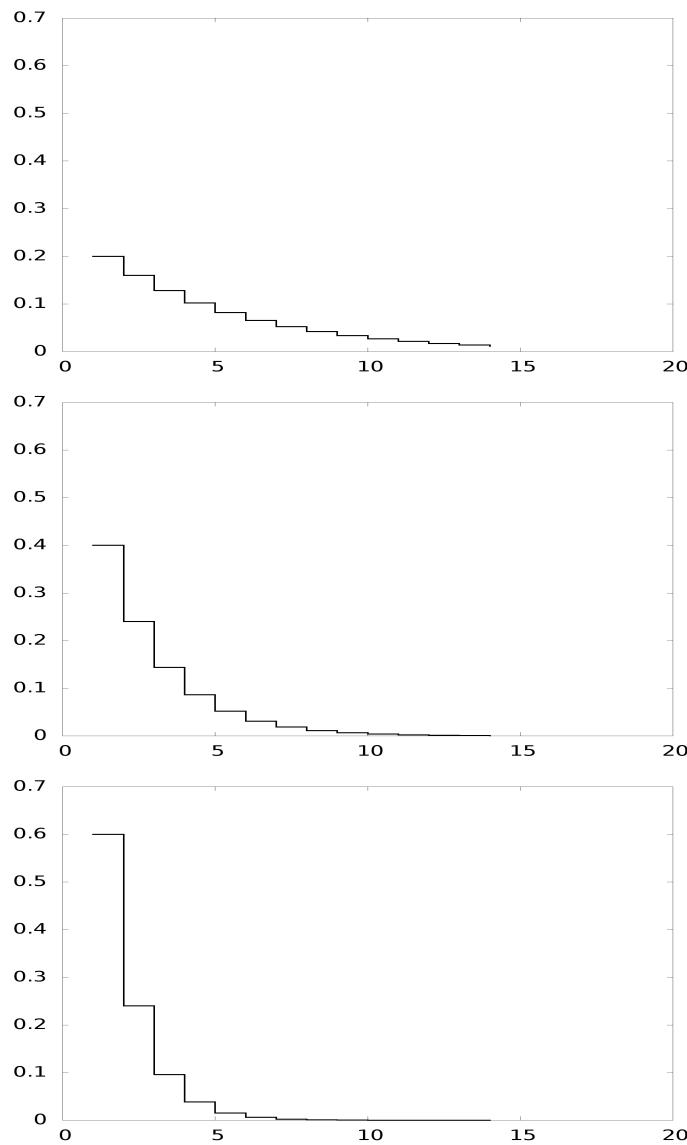
```
double Geometric::cumulative(int x) const
{
    if (x < 0)
        return 0.0;

    return 1.0 - std::pow(1 - m_probability, x-1);
}
```

PROBLEMS: (Answers on page 282)

1. (Medium; requires Calculus) The cumulative distribution formula can be derived directly. The density is $f(x; p) = (1 - p)^{x-1}p$. Prove that the cumulative formula is $(1 - p)^x$. (Hint: $\sum_{x=0}^n a^x = \frac{1-a^{x+1}}{1-a}$.)

Figure 3.4: Geometric distributions: probability = 0.2, 0.4, 0.6



2. (Medium) Some books define the Geometric distribution as the number of trials before, rather than until, the first success. What would be the density function, cumulative function, mean, and standard deviation of this alternate definition of this alternate geometric distribution?

3.5.4 Negative Binomial distribution

Parameters	$0 \leq p \leq 1$ is the probability (real) $0 \leq r$ is the number of successes (integer)
Support	$0 \leq x$ are the number of trials
Density	$f(x; p, r) = \binom{x-1}{r-1} p^r (1-p)^{y-r}$
Mean	$\frac{r}{p}$
Standard Deviation	$\sqrt{\frac{r(1-p)}{p}}$

Properties 3.3: Properties of the negative binomial distribution

The binomial distribution occurs when the experimenter knows the probability and the number of trials, and she wants to determine the expected number of successes. The negative binomial distribution occurs when the experimenter knows the probability and the expected number of successes, but she wants to know how many trials should happen to reach the expected number of successes.

Example 3.8 Examples of negative binomial distribution

- The number of rolls of a six-sided die until the third time that shows a 5 or a 6.
 - The number of flips of a weighted coin until the twelfth time that it shows heads.
 - The number of shoppers that pass until the second one that buys a particular product.
 - The number of minutes until the twentieth time a poorly-prepared speaker says “uhm”.
 - The number of pages in a book until the phrase “crunchy bacon” is used for the two hundred and seventy-sixth time.
-

The class has two private members: `double m_probability`, which holds the probability, and `int m_successes`, which hold the requested number of successes.

Creating a random number for the negative binomial is easy:

Figure 3.5: Negative Binomial: Probability = 0.3, 0.5, 0.7; Successes = 10

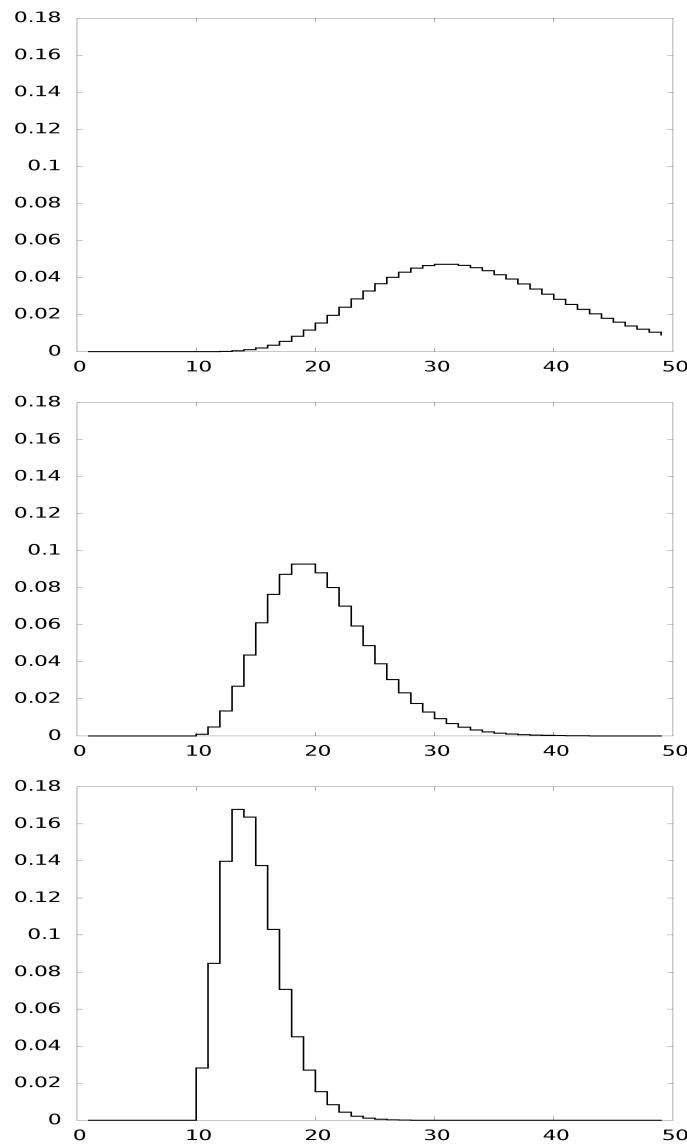
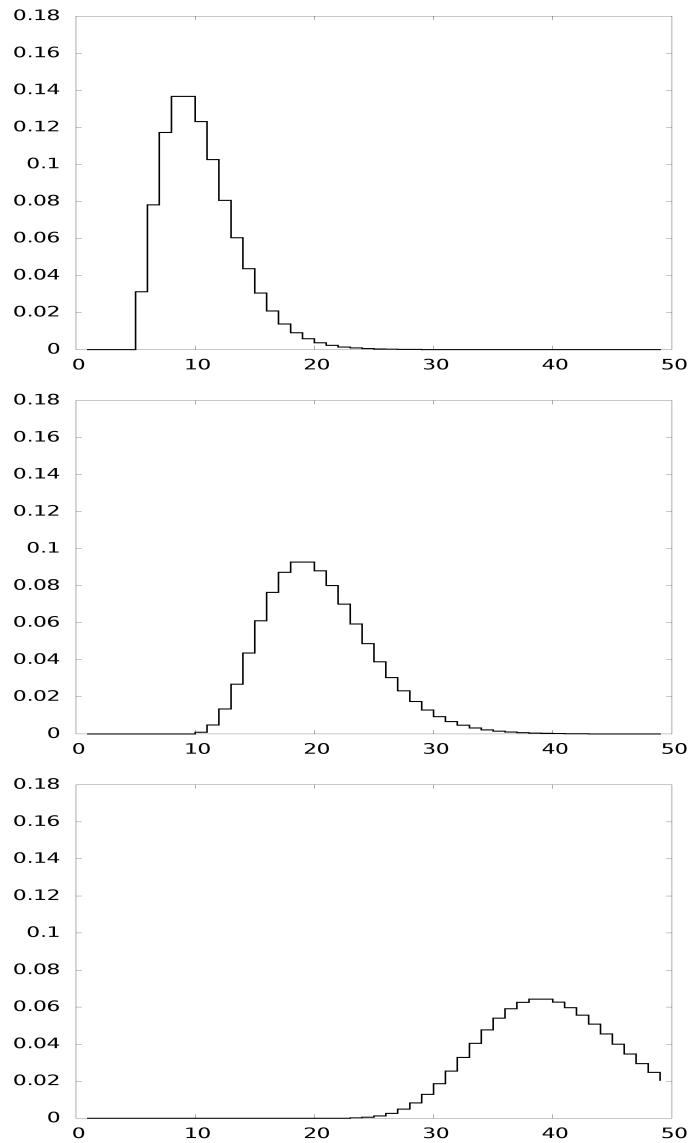


Figure 3.6: Negative Binomial: Probability = 0.5; Successes = 5, 10, 20



```

int NegativeBinomial::random() const
{
    int tries = 0;
    int successes = 0;

    while (successes < m_successes) {
        if (drand48() < m_probability)
            successes++;

        tries++;
    }

    return tries;
}

```

Code for the density comes from the definition of the negative binomial.

```

double NegativeBinomial::density(int x) const
{
    if (x < m_successes)
        return 0.0;

    return combinations(x - 1, m_successes - 1) *
        std::pow(m_probability, m_successes) *
        std::pow(1 - m_probability, x - m_successes);
}

```

Example 3.9 Video game example

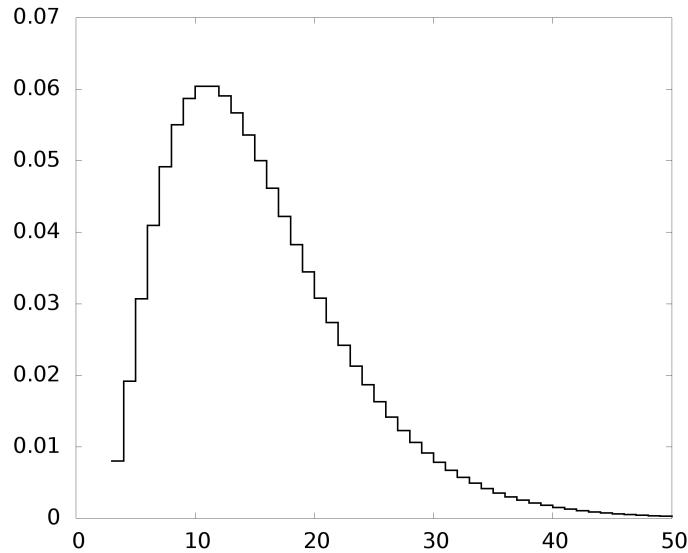
In a video game, the monstrous Gorlab can only be passed when she has been defeated three times. Each time that the player tries, she has only a 20% chance of defeating the omnivorous Gorlab. How many times must the player fight the ferocious Gorlab until she can pass?

The distribution of probabilities looks like figure 3.7 on page 106.

PROBLEMS: (Answers on page 283)

1. (Easy) What is the relationship between the geometric distribution and the negative binomial distribution?
2. (Easy) Write code to compute the distribution of example 3.9.
3. (Medium) Use the definition of the geometric distribution to prove that the probability of needing x trials to get r successes with probability p per success is $\binom{x-1}{r-1} p^r (1-p)^{x-r}$.

Figure 3.7: The distribution for the video game example



3.5.5 Hypergeometric

Parameters	$0 < N$ is the number of balls $0 \leq m \leq N$ is the number of white balls $0 \leq n \leq N$ is the number of balls chosen
Support	$0 \leq x \leq \min(n, m)$ are the number of successes
Density	$f(x; N, m, n) = \frac{\binom{m}{x} \binom{N-m}{n-x}}{\binom{N}{n}}$
Mean	$\frac{nm}{N}$
Standard Deviation	$\sqrt{n \frac{m}{N} \frac{N-m}{N} \frac{N-n}{N-1}}$

Properties 3.4: Properties of the Hypergeometric distribution

hypergeometric The **hypergeometric** distribution comes from the idea of balls in opaque jars.

Assume that a jar has N balls, of which m are white balls, and $N - m$ are black balls. Without looking, choose n balls from the jar. How many white balls will you get?

On average, you will get $n \times \frac{m}{N}$ white balls. The distribution for getting k white balls in your choice is $\frac{\binom{m}{k} \binom{N-m}{n-k}}{\binom{N}{n}}$.

There are three class variables: `m_balls`, `m_whiteBalls`, and `m_numChoose`.

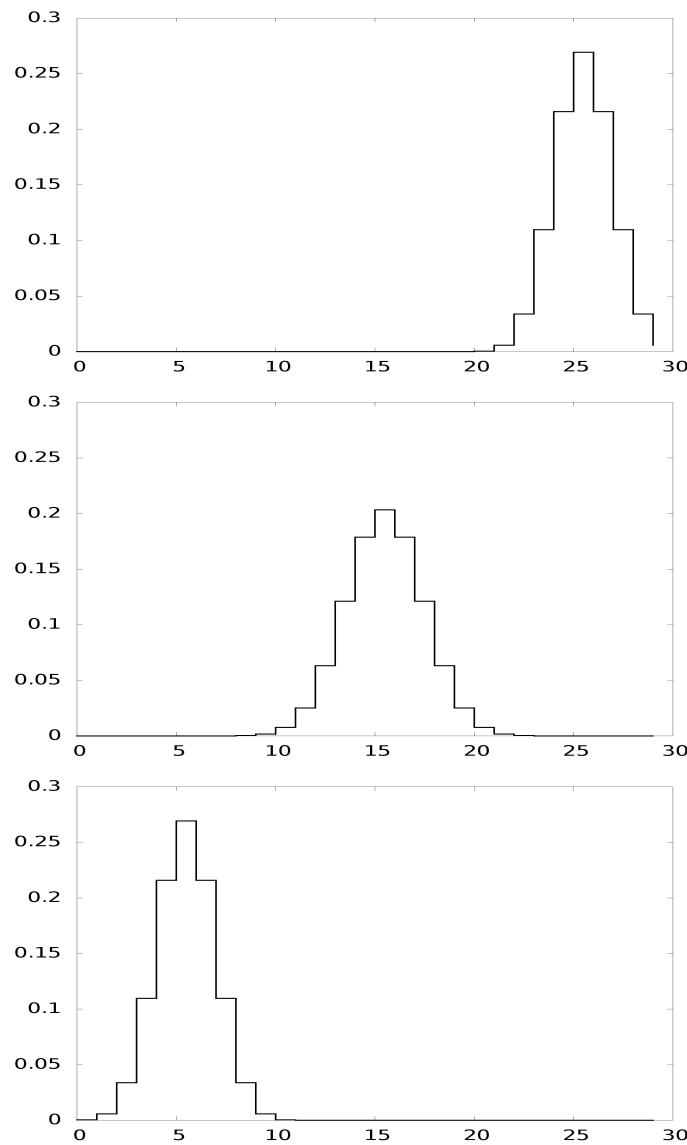
Figure 3.8: Hypergeometric: $N = 60$; $m = 10, 30, 50$; $n = 30$ 

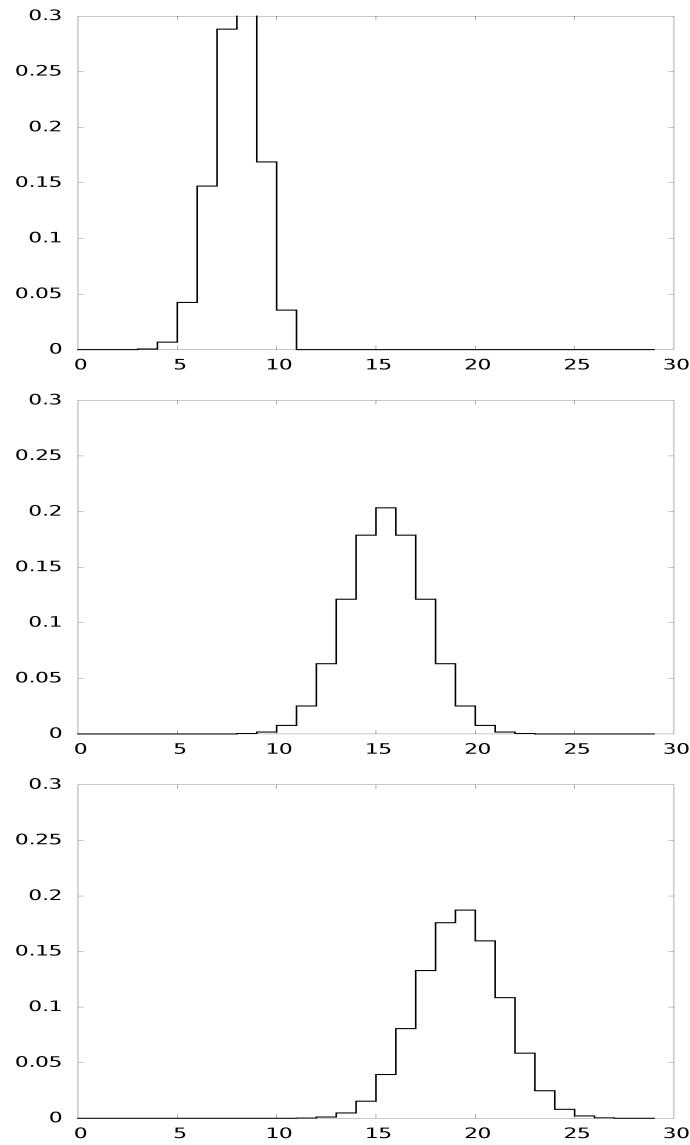
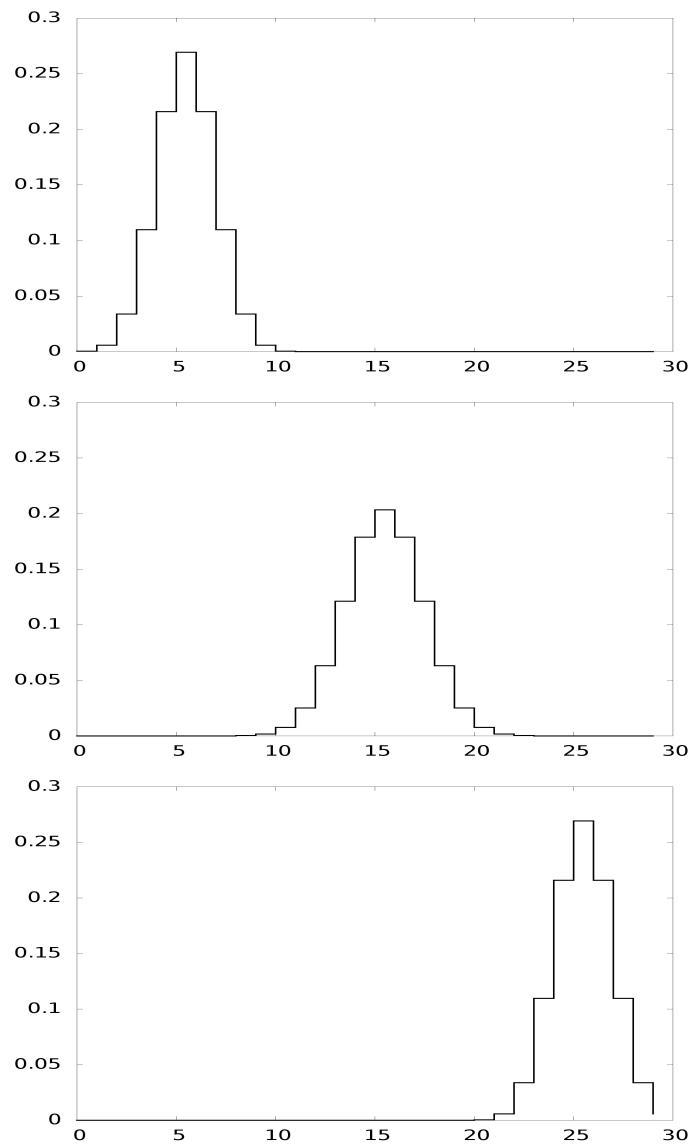
Figure 3.9: Hypergeometric: $N = 40, 60, 80$; $m = 30$; $n = 30$ 

Figure 3.10: Hypergeometric: $N = 60$; $m = 30$; $n = 10, 30, 50$ 

The code follows almost immediately:

```
double Hypergeometric::density(int x) const
{
    if (x < 0 || x > m_balls)
        throw std::out_of_range("Error: the parameter must be between 0 and the number of balls.");
    return combinations(m_whiteBalls, x) * combinations(m_balls - m_whiteBalls, m_numChoose - x) / combinations(m_balls, m_numChoose);
}

int Hypergeometric::random() const
{
    int choice;
    int chosenWhites = 0;
    int jarWhites = m_whiteBalls;
    int jarTotal = m_balls;

    for (choice = 0; choice < m_numChoose; choice++) {
        if (drand48() < ((double) jarWhites / (double) jarTotal)) {
            chosenWhites++;
            jarWhites--;
        }
        jarTotal--;
    }

    return chosenWhites;
}
```

Example 3.10 Mark and recapture

It's easy to count how many known and discovered bugs are in a program. But the more important question is: how many bugs, including unknown and undiscovered bugs, are in a program?

Here's one model that helps find that answer:

Have two independent QA engineers find as many bugs as possible. Call the (unknown) chance that the first engineer finds a bug, b_i , $\Pr_A(b_i)$ and the chance that the second engineer finds that bug, $\Pr_B(b_i)$.⁹

⁹This model assumes that all bugs are equally easy to find. I wish that were true in real life!

The number of bugs that either engineer finds would be a binomial distribution based on the total number of bugs and their probability. If $|A|$ is the total number of bugs that the first engineer finds, and $|N|$ is the total number of bugs in the program, then $|A|$ is an estimate of $\Pr_A(b_i) \times N$.

Combine the definition of conditional probability with the definition of independent sets:

$$\Pr_A(b_i) = \frac{\Pr_{A \cap B}(b_i)}{\Pr_B(b_i)}$$

Multiply both sides by $|N|$:

$$\Pr_A(b_i) \times |N| = \frac{\Pr_{A \cap B}(b_i)}{\Pr_B(b_i)} \times |N|$$

Substitute:

$$\begin{aligned} |A| &= \frac{|A \cap B|}{\frac{|B|}{|N|}} \\ &= \frac{|A \cap B||N|}{|B|} \\ |N| &= \frac{|A||B|}{|A \cap B|} \end{aligned}$$

It turns out that $|N|$ follows the hypergeometric distribution.

3.5.6 Poisson Distribution

Parameters	$0 < \lambda$ is the rate (real)
Support	$0 \leq x$
Density	$f(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$
Mean	λ
Standard Deviation	$\sqrt{\lambda}$

Properties 3.5: Properties of the Poisson distribution

The previous distributions were all related to the Bernoulli trial. This distribution isn't. It is related to the exponential distribution on 123, among the continuous distributions. However, since it's a discrete distribution, it's part of this section.

One pattern happens in many problems. If a situation is “continually dividable”, the **Poisson** distribution is a refinement on the Binomial distribution. **Poisson**

What do I mean by “continually dividable”? Assume that X happens randomly but instantaneously, 1200 times per day. X can be modeled by a Binomial distribution. However, if X is continually dividable, then that means X happens 50 times per hour, or $\frac{5}{6}$ times per minute, or $\frac{5}{360}$ times per second. In this case, we can also model the occurrences of X well with the Poisson distribution.

More formally, the Poisson distribution occurs when the following statements are true:

1. The goal is to enumerate a number of occurrences in a given space or time.
2. The time or space can be divided into “very small” regions.
3. The probability of having more than 1 occurrence in each “very small” region is extremely small.
4. Every occurrence is independent of every other occurrence.

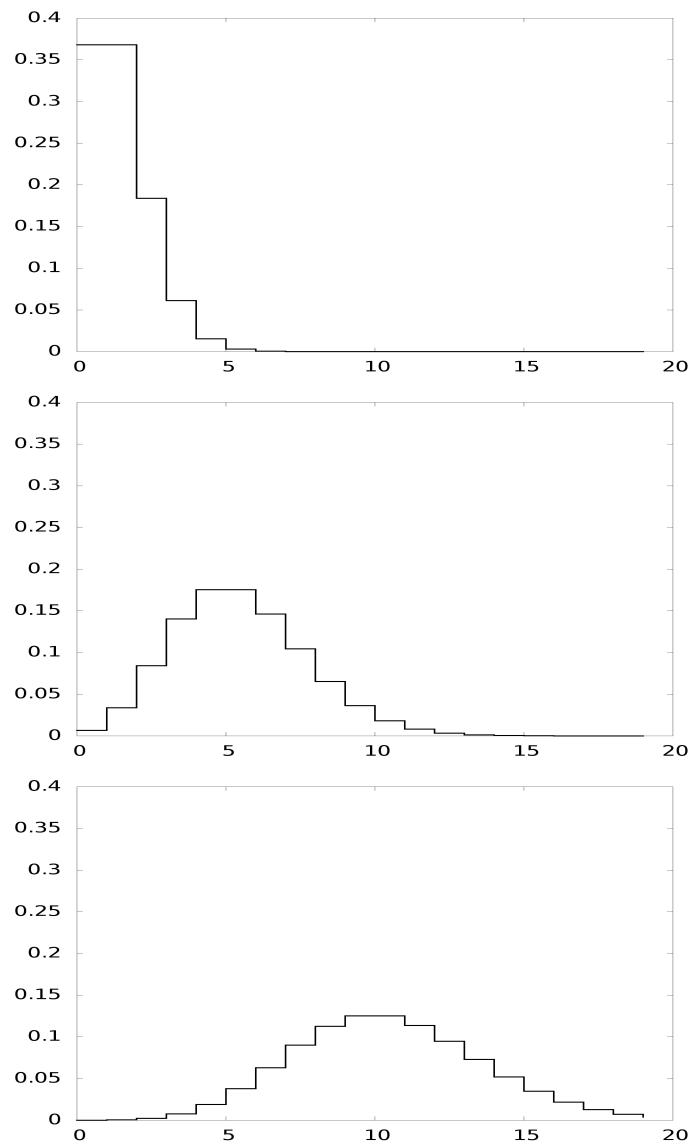
Poisson Process If these occurrences follow this kind of distribution, then it is called a **Poisson Process**.

This kind of distribution shows up often.

Example 3.11 Examples of the Poisson distribution

- The number of typos per page in a book.
 - The number of particles emitted by a radioactive source per minute.
 - The number of weeds found in a square meter of field.
 - The number of particles found per liter of liquid.
 - The number of accidents that occur in a factory per year.
 - The number of cartons of milk gone sour per large shipment.
 - The number of phone calls received per hour.
 - The number of fuses failed in a month.
 - The number of notes missed by a musician in a hour.
 - The number of bacteria in a $1 \text{ mm} \times 1 \text{ mm}$ square of a Petri dish.
-

Figure 3.11: Poisson: Lambda=1, 5, 10



The code has one private member: `double m_lambda`. This member represents the rate of the occurrences. `m_lambda` can be thought of as the product of `Binomial::m_trials` and `Binomial::m_probability`.

The constructor allows users to describe the parameter either as a scale or as a rate. This additional parameter allows the user to describe the Poisson distribution either as the number of occurrences per unit of time, or as the amount of time per occurrence.

To emphasize the relation between the Gamma distribution, the Exponential distribution, and the Poisson distribution, this book uses λ to represent the scale parameter. The scale and the rate parameters are easy to convert one to the other: $\text{scale} = \frac{1}{\text{rate}}$. This book uses λ to represent the scale parameter (when it occurs) and α to represent the rate parameter (when it occurs.)

```
Poisson::Poisson(double parameter, bool isScale)
{
    srand48((long int) std::time(NULL));

    // This implementation internally uses rates.
    if (parameter > 0.0) {
        if (isScale)
            m_rate = 1.0 / parameter;
        else
            m_rate = parameter;
    }
    else
        throw std::out_of_range("Rate must be greater than 0.0.");
}
```

The density function for Poisson is $f(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$. This function readily translates into code:

```
double Poisson::density(int x) const
{
    if (x < 0)
        return 0.0;

    return std::pow(m_rate, x) * std::exp(-m_rate) / \
           factorial(x);
}
```

An easy algorithm for generating Poisson random numbers comes from [Knuth, 1981, page 132]. Understanding this algorithm depends on understanding the exponential distribution, described on page 123.

The Poisson distribution can be described as the number of times that an exponential event occurs in a certain time. Exponential random numbers represent a length of time between these events.

Therefore, one can take sums of exponential random numbers until the sum is larger than the total time allowed, then count the number of exponential numbers that were chosen.

This algorithm transforms that idea slightly.

```
int Poisson::random() const
{
    int count;
    double compare;
    double product;

    compare = std::exp(-m_rate);

    count = 0;
    product = drand48();
    while (product > compare) {
        product *= drand48();
        count++;
    }

    return count;
}
```

Example 3.12 Poisson queue example

You want to simulate a very small shop. The shop is so small that only one customer can fit inside at a time; if another customer comes by while you're serving one, then she will just move on.

Customers come every c minutes. A customer, once inside the shop, takes s minutes to be served.

With these restrictions, make a simulation that shows how many customers come in the shop every hour and how many customers move on.

```
struct QueueResults
{
    int numServed;
    int numLost;
};

QueueResults simpleQueue(int custFreq, int \
    custTime, int timeTotal)
{
    bool isOccupied;
    int customerMinute;
    Poisson customers(custFreq, true);
```

```

QueueResults result;
int time;
int timeLeave = 0;

result.numServed = 0;
result.numLost = 0;

isOccupied = false;
for (time = 0; time < timeTotal; time++) {
    if (time == timeLeave)
        isOccupied = false;

    customerMinute = customers.random();

    if (customerMinute > 1)
        result.numLost += customerMinute - 1;

    if (customerMinute > 0) {
        if (isOccupied)
            result.numLost++;
        else { // !isOccupied
            result.numServed++;
            timeLeave = time + custTime;
            isOccupied = true;
        }
    }
}

return result;
}

```

PROBLEMS: (Answers on page 284)

1. (Medium) (Requires Calculus) Prove that if $\frac{n}{p} = \lambda$, then $\lim_{n \rightarrow \infty} \text{binom}(x; n, p) = \text{poisson}(x; \lambda)$
2. (Medium) The previous problem showed that at infinity, the Binomial distribution reaches the Poisson distribution. How well does the Poisson distribution approximate the Binomial distribution at smaller numbers? Compare $\text{binom}(x; n, p)$ against $\text{poisson}(x; n \times p)$ when $n \in \{10, 20, 50, 100, 200\}$ and $p \in \{0.2, 0.3, 0.4, 0.5\}$, where $x = \lfloor \frac{np}{2} \rfloor$. ($\lfloor x \rfloor$ is the largest integer smaller than x .)

3. (Medium) (Requires the exponential distribution) I gave some information about a way to generate Poisson numbers, then I gave a program. Show that the two ideas are the same. (Hint: use the exponential function.)

3.6 Continuous Distributions

3.6.1 Implementation

Like the discrete distributions, we will put all continuous distributions under the `Distribution` base class. However, continuous distributions will have the base class `Distribution<double, double>` instead of `Distribution<int, double>`.

The only method in a distribution that must be implemented is the `density` method, our probability density function.

All methods other than `density` can be created in terms of the `density` function. However, the generic implementation is usually much slower than its descendants.

The infinities

Like the summation function, our integration function does not work well with extremely large or small numbers. Luckily, most distributions shrink to very small probabilities at moderately positive or moderately negative numbers.

The infinity functions are identical to those on page 87, except that they return `double` instead of `int`.

Cumulative

The cumulative function is the probability that a random number chosen by this function will be equal or less than the given number. If $f(x)$ is the density function, then the cumulative function of a is $\int_{-\infty}^a f(x) dx$.

This method uses the fact that a `Distribution` is an `IFunctor`.

Since the cumulative function returns a probability, the result will always be a number between 0 and 1.

```
template<>
double Distribution<double, double>::cumulative(double x
    ) const
{
    double xNegInf = negativeInfinity();

    Integrate<double> intDensity(this, (x - xNegInf) / \
        10000.0);
    Interval interval(xNegInf, e_Closed, x, e_Closed);
```

```

    return intDensity.integrate(interval);
}

```

Mean

This function represents the mean value of the distribution. If $f(x)$ is the density function, then the mean is $\int_{-\infty}^{\infty} xf(x)dx$.

First, we need a class that returns $x \times f(x)$:

```

template <typename TParameter>
class MultiplyByX: public IFunctor<TParameter>
{
private:
    double m_mean;
    const IFunctor<TParameter> *m_pifunctor;

public:
    MultiplyByX(const IFunctor<TParameter> *pifunctor);
    TParameter operator() (TParameter x) const;
};

template <typename TParameter>
MultiplyByX<TParameter>::MultiplyByX(const IFunctor<TParameter> *pifunctor):
    m_pifunctor(pifunctor)
{
}

template <typename TParameter>
TParameter MultiplyByX<TParameter>::operator() (const TParameter x) const
{
    return x * (*m_pifunctor)(x);
}

```

Once this helper class has been created, the mean can be computed:

```

template<>
double Distribution<double, double>::mean() const
{
    double xNegInf = negativeInfinity();
    double xPosInf = positiveInfinity();

    MultiplyByX<double> mbx(this);
    Integrate<double> intMbx(&mbx, k_delta);
    Interval interval(xNegInf, e_Closed, xPosInf, e_Closed,
);

```

```

    return intMbx.integrate(interval);
}

```

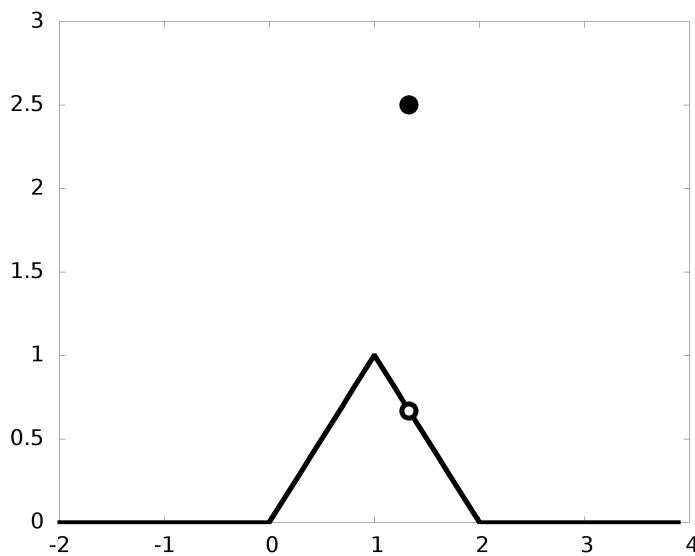
Mode

I chose not to create a general method for the mode.

Without intelligence about the distribution, guaranteeing that one has found the true maximum is impossible. Imagine trying to find the true maximum of the function in example 3.12 without seeing the distribution as a whole. Unless one knows that a discontinuity happens at 1.2345, finding the true mode is impossible.

Figure 3.12: Example where finding the mode is difficult

$$\begin{aligned}
 y &= 0 \text{ when } x < 0 \\
 y &= x \text{ when } 0 \leq x < 1 \\
 y &= 2 - x \text{ when } 1 \leq x < 2 \text{ and } x \neq 1.2345 \\
 y &= 2.5 \text{ when } x = 1.2345 \\
 y &= 0 \text{ when } x \geq 2
 \end{aligned}$$



Finding a mode is difficult when the distribution has many local maximums. Many methods have been written about this problem, all beyond the

scope of this book.¹⁰

Quantile

This function represents the inverse of the cumulative function. If $\text{cumulative}(x) = q$, then $\text{quantile}(q) = d$. The input to the quantile function is a probability, so the input is always between 0 and 1.

This method is slow; it requires multiple calls to `cumulative()`, each of which is fairly slow. When possible, this method should be overridden.

```
const double k_epsilon = 0.01;
const int k_maxRounds = 100;

template<double, double>::quantile(double y) const
{
    int countRounds;
    double negInf;
    double posInf;
    double xPrevious;
    double xCurrent;
    double xNext;
    double yPrevious;
    double yCurrent;
    double yNext;

    if (y < 0.0 || 1.0 < y)
        throw std::out_of_range("The quantile input must
            be between 0 and 1 (inclusive).");
    else if (y == 0.0)
        return negativeInfinity();
    else if (y == 1.0)
        return positiveInfinity();

    negInf = negativeInfinity();
    posInf = positiveInfinity();

    xPrevious = negInf + (posInf - negInf) / 3.0;
    yPrevious = cumulative(xPrevious) - y;

    xCurrent = xPrevious + (posInf - negInf) / 3.0;
    yCurrent = cumulative(xCurrent) - y;

    countRounds = 0;
```

¹⁰ Interested readers should look up algorithms like Simulated Annealing or Genetic Algorithms.

```

do {
    countRounds++;

    xNext = xCurrent - ((xCurrent - xPrevious) * \
        yCurrent) / (yCurrent - yPrevious);
    yNext = cumulative(xNext) - y;

    xPrevious = xCurrent;
    xCurrent = xNext;

    yPrevious = yCurrent;
    yCurrent = yNext;
} while ((std::abs(yCurrent - yPrevious) > k_delta) \
    && countRounds < k_maxRounds);

if (countRounds >= k_maxRounds)
    std::cerr << "WARNING: Reached the limit of " \
        countRounds_in_Distribution<double, double>:: \
        quantile.\n";

return xCurrent;
}

```

Random

This method returns a random number chosen from the distribution.

```

template<>
double Distribution<double, double>::random() const
{
    double dUniformRandom = (double) drand48();

    return quantile(dUniformRandom);
}

```

Standard deviation

This method returns the standard deviation of the random variable, as described on page 55. If $f(x)$ is the density function and μ is the mean, then the variance is $\int_{-\infty}^{\infty} x(f(x) - \mu)^2 dx$.

First, a class needs to return $x \times (f(x) - \mu)^2$ so that it can be integrated:

```

template <typename TParameter>
class VarianceFunction: public IFunctor<TParameter>
{
private:

```

```

double m_mean;
const IFunctor<TParameter> *m_pifunctor;

public:
    VarianceFunction(const IFunctor<TParameter> *pifunctor,
                    , TParameter mean);
    TParameter operator() (TParameter x) const;
};

template <typename TParameter>
VarianceFunction<TParameter>::VarianceFunction(const \
    IFunctor<TParameter> *pifunctor, TParameter mean) :
    m_pifunctor(pifunctor)
{
    m_mean = mean;
}

template <typename TParameter>
TParameter VarianceFunction<TParameter>::operator() (\
    TParameter x) const
{
    TParameter diff;

    diff = m_mean - (*m_pifunctor)(x);
    return x * diff * diff;
}

```

Once this helper class has been created, the standard deviation can be computed:

```

template<>
double Distribution<double, double>::standardDeviation() \
    const
{
    double xNegInf = negativeInfinity();
    double xPosInf = positiveInfinity();
    double xMean = mean();

    VarianceFunction<double> varfunc(this, xMean);
    Integrate<double> intVarFunc(&varfunc, k_delta);
    Interval interval(xNegInf, e_Closed, xPosInf, e_Closed \
        );
}

return std::sqrt(intVarFunc.integrate(interval));
}

```

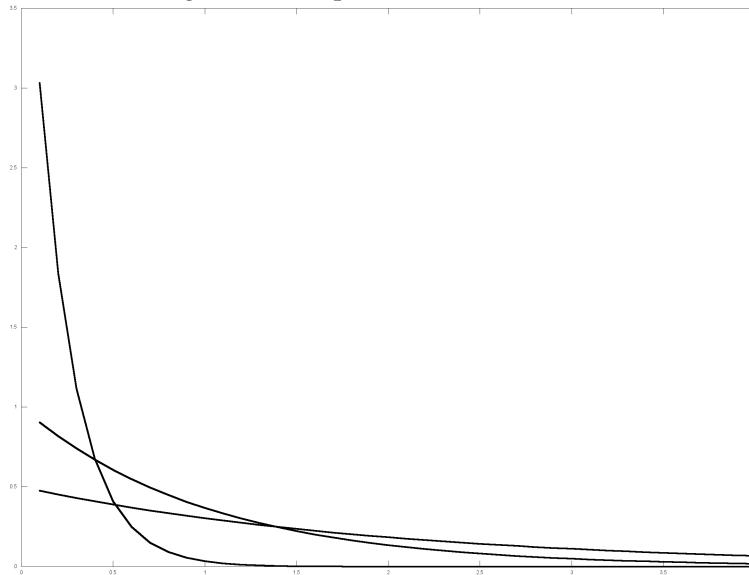
3.6.2 The Exponential distribution

Parameters	$\lambda > 0$ is the probability per unit time (real)
Support	$x > 0$ is the position
Density	$f(x; \lambda) = \frac{e^{-x/\lambda}}{\lambda}$
Mean	λ
Standard Deviation	λ

Properties 3.6: Properties of the exponential distribution

When an continuous-time event happens with a constant probability per unit time (or space), then the time (or space) until the event happens once can be described with the **exponential distribution**. For this book, λ represents the **expected number of units of time or space per occurrence**.¹¹

Figure 3.13: Exponential: $\lambda = 0.5, 1, 2$



¹¹Other books use the opposite: λ represents the number of occurrences per unit of time or space. This code allows either interpretation in the constructor.

Example 3.13 Examples of exponential distributions

- Hours until the first telephone calls received in a day.
 - Time between customers entering a shop.
 - Time until a fuse fails.
 - Time between bug hits on a Southern U.S. highway.
 - Time between requests to a web server.
 - Time between missed notes by a musician.
 - Time between clicks of a Geiger counter measuring radiation.
 - Distance until the next flaw in a copper wire.
 - The number of characters between typos in a book.
 - The time between accidents in a factory.
-

The exponential distribution is a continuous analogue to a Bernoulli trial. Both distributions represent the time (or distance) until a single event occurs.

memoryless

The exponential distribution is the only continuous, **memoryless** distribution. Mathematically, memoryless is written $\Pr(Y > a + b | Y > a) = \Pr(Y > b)$. Memoryless means that the chance that an event happens depends only on the length of time or amount of space involved, and not on how long the item otherwise has existed.

Like the Bernoulli trial, the exponential distribution is the basis for many other distributions:

- The sum of n exponential distributions is the Gamma distribution.
- The expected number of times an exponential distribution occurs in a given time or space is the Poisson distribution.
- If $\text{Poisson}(\lambda)$ represents the number of occurrences in an area, then $\text{Exponential}(\lambda)$ represents the distance between occurrences measured by the same area.

The source code is simple:

```
Exponential::Exponential(double parameter, bool isScale)
{
    srand48((long int) std::time(NULL));

    if (parameter > 0) {
        if (isScale)
            m_scale = parameter;
        else // It's frequency
            m_scale = 1.0 / parameter;
    }
    else
        m_scale = 1.0;
}
```

```

throw std::out_of_range("Lambda must be strictly
                        positive.");
}

```

Like the Poisson distribution, you can give the parameter as a scale or as a rate.

```

double Exponential::density(double x) const
{
    if (x > 0)
        return std::exp(-x / m_scale) / m_scale;
    else
        return 0;
}

```

The cumulative function can be derived from the density. Let f represent the density function and F represent the cumulative function:

$$\begin{aligned}
 F(x) &= \int_{-\infty}^x f(x) dx \\
 &= \int_0^x \frac{e^{-x/\lambda}}{\lambda} dx \\
 &= \left[-e^{-x/\lambda} \right]_0^x + C \\
 &= C - e^{-x/\lambda}
 \end{aligned}$$

We know that the result must be between 0 and 1. Since $0 < x$ and $0 < \lambda$, $-e^{-x/\lambda}$ is always between -1 and 0. Therefore, $C = 1$ and the cumulative function is $F(x) = 1 - e^{-x/\lambda}$.

```

double Exponential::cumulative(double x) const
{
    if (x > 0)
        return 1.0 - std::exp(-x / m_scale);
    else
        return 0;
}

```

Deriving the mean depends on knowing that $\int xe^{cx} dx = \frac{e^{cx}(cx-1)}{c^2} + C$:

$$\begin{aligned}
\text{mean} &= \int_{-\infty}^{\infty} xf(x) dx \\
&= \int_0^{\infty} \frac{x e^{-x/\lambda}}{\lambda} dx \\
&= \left[\frac{e^{-x/\lambda} (-\frac{x}{\lambda} - 1)}{\lambda \left(\frac{1}{-\lambda} \right)^2} \right]_{x=0}^{x=\infty} \\
&= \left[\frac{e^{-\infty/\lambda} (-\frac{\infty}{\lambda} - 1)}{-\frac{1}{\lambda}} \right] - \left[\frac{e^{0/\lambda} (-\frac{0}{\lambda} - 1)}{-\frac{1}{\lambda}} \right] = \lambda [0 - (1 \times -1)] \\
&= \lambda
\end{aligned}$$

```
double Exponential::mean() const
{
    return m_scale;
}
```

Deriving the variance depends on knowing that $\int x^2 e^{cx} dx = e^{cx} \left(\frac{x^2}{c} - \frac{2x}{c^2} + \frac{2}{c^3} \right)$

$$\begin{aligned}
\text{std dev} &= E[X^2] - (E[X])^2 \\
&= \int_{-\infty}^{\infty} x^2 f(x) dx - \lambda^2 \\
&= \int_0^{\infty} \frac{x^2 e^{-x/\lambda}}{\lambda} dx - \lambda^2 \\
&= \left[e^{-x/\lambda} \frac{-\lambda x^2 - 2\lambda^2 x - 2\lambda^3}{\lambda} \right]_{x=0}^{x=\infty} - \lambda^2 \\
&= \left[e^{-x/\lambda} (-x^2 - 2\lambda x - 2\lambda^2) \right]_{x=0}^{x=\infty} - \lambda^2 \\
&= [0 - (1 \times -2\lambda^2)] - \lambda^2 \\
&= 2\lambda^2 - \lambda^2 \\
&= \lambda^2
\end{aligned}$$

The standard deviation is the square root of the variance.

```
double Exponential::standardDeviation() const
{
    return m_scale;
}
```

If you can pick x , a uniformly-distributed random number between 0 and 1, you can pick a random number from any distribution by using the inverse of the cumulative function of x .

To derive the inverse:

$$\begin{aligned}y &= 1 - e^{-x/\lambda} \\1 - y &= e^{-x/\lambda} \\\log(1 - y) &= -x/\lambda \\\lambda \log(1 - y) &= -x \\-\lambda \log(1 - y) &= x\end{aligned}$$

Since y is a uniform distribution in $[0, 1]$, so is $1 - y$. We can substitute y for $1 - y$. This change gives fast code to find an exponential random number:

```
double Exponential::random() const
{
    double uniform;

    uniform = drand48();

    return 0.0 - (std::log(uniform) * m_scale);
}
```

PROBLEMS: (Answers on page 286)

1. (Medium) The exponential distribution is the continuous **memoryless** **memoryless** distribution: $\Pr(X > a + b | X > a) = \Pr(X > b)$. Prove this property for the exponential distribution. (It is the only continuous memoryless distribution; you do not need to prove this fact.)
2. (Medium) Prove that if Poisson($x; \lambda$) represents the chance that x occurrences of an event happen in a given time, then Exponential($\frac{1}{\lambda}$) represents the time between occurrences.

3.6.3 Uniform distribution

The uniform distribution is the distribution that programmers are most familiar with. It has the following properties:

- The distribution has a minimum and a maximum value.
- All values within the distribution are equally probable.
- The probability of any interval is the size of the interval divided by the size of the distribution.

Parameters	θ_1 is the minimum $\theta_2 \geq \theta_1$ is the maximum
Support	$\theta_1 \leq x \leq \theta_2$ is the position
Density	$f(x; \theta_1, \theta_2) = \frac{1}{\theta_2 - \theta_1}$
Mean	$\frac{\theta_1 + \theta_2}{2}$
Standard Deviation	$\frac{(\theta_2 - \theta_1)}{\sqrt{12}}$

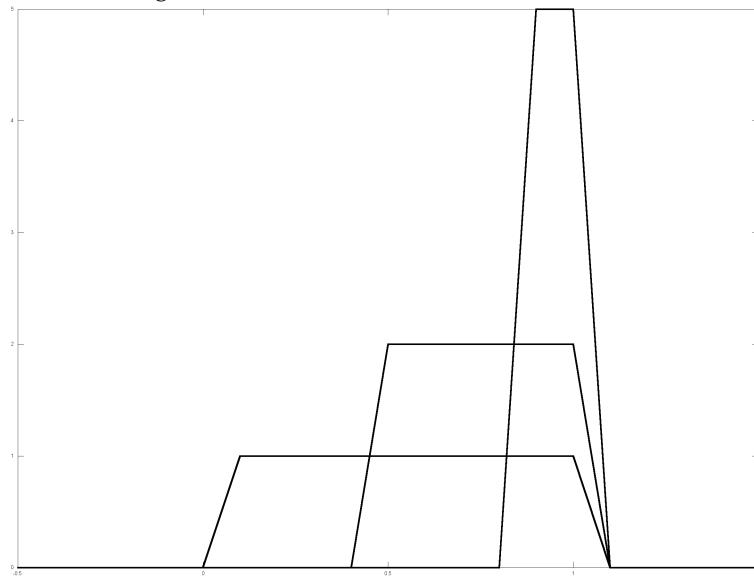
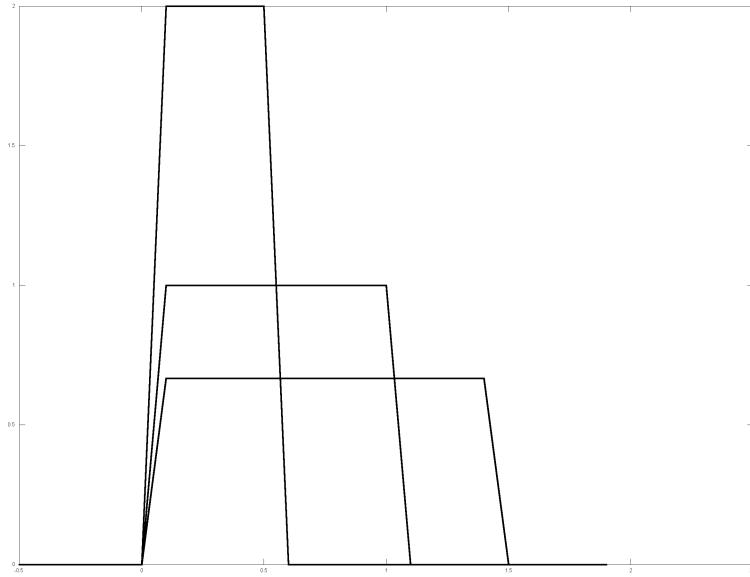
Properties 3.7: Properties of the Uniform distributionFigure 3.14: Uniform: $\theta_1 = 0, 0.5, 0.8; \theta_2 = 1$ 

Figure 3.15: Uniform: $\theta_1 = 0; \theta_2 = 0.5, 1, 1.5$ 

Example 3.14 Examples of the Uniform Distribution

The uniform distribution happens whenever any value in a range are equally likely, and no value outside that range are possible.¹² Continuous examples include:

- In the range $[0, 360]$, the angle that a dart aimed at the center will hit a dartboard.
- In the range $[0, 60)$, the second and fraction that a detector reacts to a particle from a very small, radioactive source.

By taking the largest integer smaller than a chosen random number, we get the discrete uniform distribution. Examples that use this distribution include:

- Choose a card from a deck of 52 cards.
 - Simulate a die roll.
 - Simulate a roulette wheel.
-

An implementation for the uniform distribution is:

¹²You have used this distribution, or its discrete equivalent, for ages.

```

Uniform::Uniform(double lower, double upper)
{
    srand48((long int) std::time(NULL));

    if (lower < upper) {
        m_lower = lower;
        m_upper = upper;
    } else {
        throw std::out_of_range("Error: the lower bound must be strictly less than the upper bound.");
    }
}

double Uniform::density(double x) const
{
    if (x < m_lower)
        return 0;
    else if (m_lower <= x && x <= m_upper)
        return 1 / (m_upper - m_lower);
    else
        return 0;
}

double Uniform::cumulative(double x) const
{
    if (x < m_lower)
        return 0;
    else if (m_lower <= x && x <= m_upper)
        return (x - m_lower) / (m_upper - m_lower);
    else
        return 1;
}

double Uniform::quantile(double q) const
{
    if (q < 0 || 1 < q)
        throw std::out_of_range("quantile(): The given quantile must be between 0 and 1.");

    return q * (m_upper - m_lower) + m_lower;
}

double Uniform::random() const
{
}

```

```

    return quantile(drand48());
}

```

PROBLEMS: (Answers on page 286)

1. (Easy) You want a uniform distribution with mean m and standard deviation s . What parameters do you use?

3.6.4 Normal Distribution

Parameters	μ is the mean (real) $0 \leq \sigma$ is the standard deviation (real)
Support	x is the position
Density	$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)\right]$
Mean	μ
Standard Deviation	σ

Properties 3.8: Properties of the normal distribution

The **normal distribution**, also known as the Gaussian distribution, is the most common distribution in nature. It occurs whenever data depends on many small choices.

The mean and standard deviation of the normal distribution come from its parameters.

Example 3.15 Examples of the normal distribution

- The height of a randomly-chosen set of women
 - The IQ of a random group of people
 - The number of apples harvested on equal-sized, equally-treated plots of land in an apple farm.
 - The number of people in the campus union at 5:30 on Thursday during the semester.
-

An implementation of the normal distribution is:

Figure 3.16: Normal: mean = -1, 0, 1; Standard deviation = 1

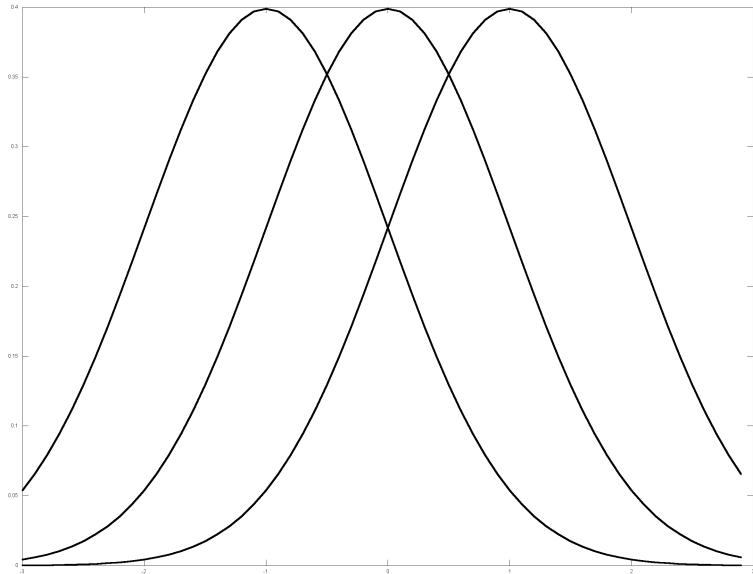
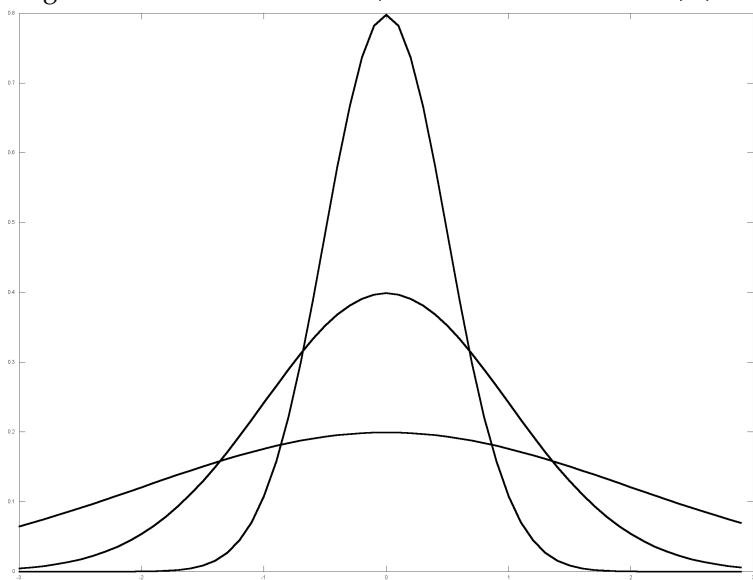


Figure 3.17: Normal: mean = 0; Standard deviation = 0.5, 1, 2



```

const double k_UnsetNextRandom = -999999;

Normal::Normal(double mean, double standardDeviation)
{
    srand48((long int) std::time(NULL));

    m_nextRandom = k_UnsetNextRandom;

    if (0 < standardDeviation) {
        m_mean = mean;
        m_standardDeviation = standardDeviation;
    } else
        throw std::out_of_range("Error: the standard deviation must be strictly positive.");
}

const double PI = 3.14159265358979;
double Normal::density(double x) const
{
    return (1 / (std::sqrt(2 * PI) * m_standardDeviation))
        * std::exp(- (x - m_mean) * (x - m_mean)
        / (2 * m_standardDeviation * m_standardDeviation));
}

```

The Box-Muller Transformation¹³ transforms u_1 and u_2 , two uniformly-distributed numbers in the range $[0,1]$ to z_1 and z_2 , two normally-distributed numbers with mean 0 and standard deviation 1 with the following two formulas:

$$z_1 = \cos(2\pi u_2) \sqrt{-2 \log_e(u_1)}$$

$$z_2 = \sin(2\pi u_2) \sqrt{-2 \log_e(u_1)}$$

The source code below gives the “polar form” of this transformation.

```

const double k_UnsetNextRandom = -999999;

double Normal::random() const
{
    double v1;
    double v2;

```

¹³The Box-Muller Transformation was originally published in Box, G. E. P. and Muller, M. E. “A Note on the Generation of Random Normal Deviates.” Ann. Math. Stat. 29, 610-611, 1958. My sources include [Basset, 2001, page 386], <http://www.taygeta.com/random/gaussian.html>, and <http://mathworld.wolfram.com/Box-MullerTransformation.html>.

```

double w;
double y;

if (m_nextRandom == k_UnsetNextRandom) {
    do {
        v1 = drand48() * 2.0 - 1.0;
        v2 = drand48() * 2.0 - 1.0;
        w = v1 * v1 + v2 * v2;
    } while (w >= 1.0);

    y = std::sqrt( std::log(w) * -2.0 / w );

    m_nextRandom = y * v2;
    return y * v1 * m_standardDeviation + m_mean;
} else {
    v1 = m_nextRandom;
    m_nextRandom = k_UnsetNextRandom;

    return v1 * m_standardDeviation + m_mean;
}
}

```

PROBLEMS: (Answers on page 287)

1. (Easy) *Dollar cost averaging* is recommended to smooth out investments. Dollar cost averaging is the idea that one should split one's investment money over the time that one invests. What happens with it?

Assume that the stock market follows a normal distribution, annually averages a 3% gain (that is, if the stock market starts at 100, after one year it averages 103), with a standard deviation at 6%. Then examine two strategies for investing \$10,000 over 10 years:

- (a) (Dollar-cost averaging) Invest \$1,000 at the beginning of every year for 10 years.
- (b) (Invest it all) Invest \$10,000 at the beginning, and let that grow for 10 years.

Run each strategy at least 1000 times. Which strategy usually gives a better result at the end of ten years? Which strategy has more variability?

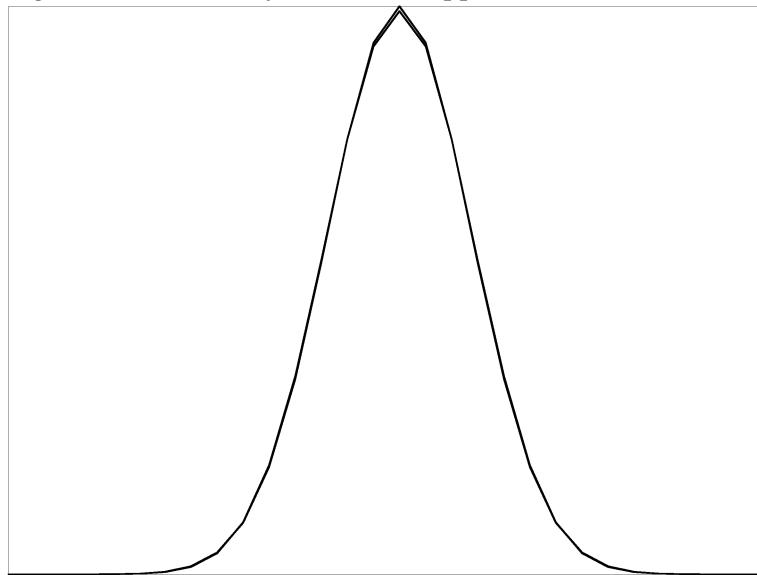
2. (Medium) The source code isn't the same as the description for Box-Muller description. Prove that the two give the same results.

Relationship between the binomial and the normal distributions

The normal distribution comes about when many small choices aggregate. The binomial distribution comes about when many copies of the same choice aggregate.

The normal distribution approximates the binomial distribution, and vice-versa. But the normal distribution is much easier to compute. When n , the number of trials for the binomial distribution, grows, then $\binom{n}{x}$ grows exponentially faster. Even a 64-bit integer can't hold $60!$.

Figure 3.18: The binary and normal approximation when $n = 30$



The binomial distribution has two parameters: n , the number of trials, and p , the probability of success. The equivalent parameters for the normal distribution uses np as the mean and $np(1 - p)$ as the variance.

```
BinomialApproximation::BinomialApproximation(int trials, 
    double probability)
{
    srand48((long int) std::time(NULL));

    m_pnorm = new Normal(trials * probability, sqrt(
        trials * probability * (1.0 - probability)));
}

BinomialApproximation::~BinomialApproximation()
```

```

{
    delete m_pnorm;
}

double BinomialApproximation::density(double x) const
{
    return m_pnorm->density(x);
}

```

(All other methods, like `density`, just call their counterpart in `m_pnorm`.)

Ranges and the cumulative function

Often, people are interested in ranges: If you flip one hundred coins, what is the chance that between 40 and 60 (inclusive) of them will turn out heads?

There are two natural ways to find a range under Binomial:

- Take the sum `binom.density(40) + binom.density(41) + ... + binom.density(60)`.
- Take the difference `binom.cumulative(60) - binom.cumulative(39)`.

Though both are approximated well using the normal distribution, statisticians have noticed that the approximations using the cumulative function work better when a **continuity correction** is used. Instead of using the maximum value, use the maximum value + 0.5. Instead of using the minimum value - 1, use the minimum value - 0.5. Therefore, the cumulative function would look like `norm.cumulative(60.5) - norm.cumulative(39.5)`.

PROBLEMS: (Answers on page 289)

1. (Easy) Test the continuity correction on a binomial distribution where $n = 30$ and $p = 0.5$. Make a chart with four columns:
 - x , where x ranges from 0 to 30.
 - The binary probability that $\Pr(X = x)$.
 - The approximation for `norm.density(x)`
 - The approximation for `norm.cumulative(x + 0.5) - norm.cumulative(x - 0.5)`.
2. (Medium) All storage systems degrade over time. Assume that you have a hard drive that, every day, flips one bit per million bits on the drive. (The flipped bits are randomly chosen uniformly across the drive.)

You have an important document, l bits long, on this drive: you must be able to read it bit-by-bit, correctly. You know that the drive is flaky, so you wrote the document three times on the drive. You read each bit

in a best-two-out-of-three fashion: whichever bit is agreed by two out of three copies is accepted.

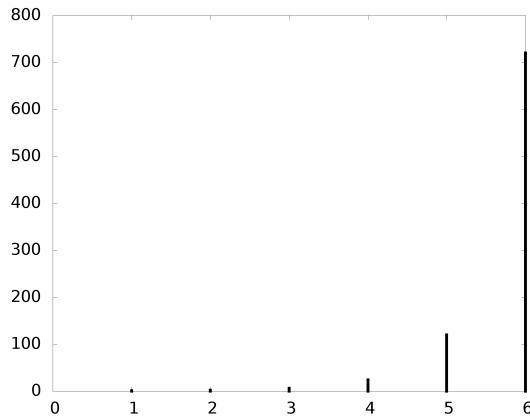
How long will it be until you're less than 95% certain that you can read the document correctly?

3.6.5 Gamma and its relations

The gamma function

Remember the factorial function of page 21? The function is graphed on graph 3.19.

Figure 3.19: The factorial function



This graph looks like you could draw a simple, curved line between the members of the factorial function. You can. Closely related to the continuous version of the factorial function is the **gamma function**.

gamma function

The gamma function is defined as:

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$$

On page 138 is a proof that $\Gamma(z + 1) = z!$.

To implement this function, create a `GammaFunctionBase` class that takes the parameter m_z in the constructor and one method:

```
double GammaFunctionBase::operator() (double parameter) const
{
    return std::pow(parameter, m_z - 1) * std::exp(-parameter);
}
```

If z is a positive integer, then $\Gamma(z + 1) = z!$

This proof comes in two parts:

1. $\Gamma(1) = 1$:

$$\begin{aligned}\Gamma(1) &= \int_0^\infty t^0 e^{-t} dt \\ &= -\left(e^{-\infty} - e^0\right) \\ &= e^0 \\ &= 1\end{aligned}$$

2. $\Gamma(z + 1) = z\Gamma(z)$: This proof requires integration by parts (rule 10 on page 47).

$$\Gamma(z + 1) = \int_0^\infty t^z e^{-t} dt$$

Let $u = t^z$ and $dv = e^{-t} dt$. Then $du = zt^{z-1} dt$ and $v = -e^{-t}$

$$\begin{aligned}\Gamma(z + 1) &= \int_0^\infty u dv \\ &= uv - \int_0^\infty v du \\ &= [t^z e^{-t}]_{t=0}^{t=\infty} + z \int_0^\infty t^{z-1} e^{-t} dt\end{aligned}$$

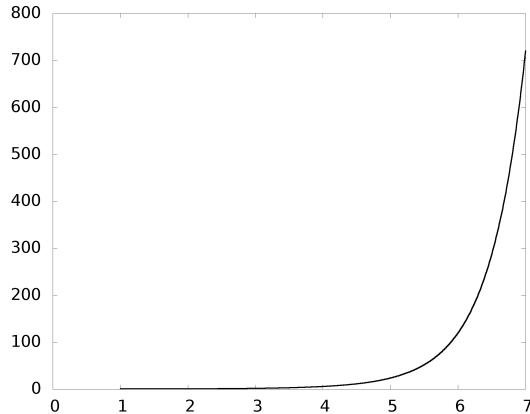
Now, if $t = 0$, then $0^z e^0 = 0 \times 1 = 0$. Further, when $t \rightarrow \infty$, $t^z e^{-t} \rightarrow 0$ no matter what value z is; e^{-t} very quickly outweighs any value of t^z .

Therefore,

$$\begin{aligned}\Gamma(z + 1) &= 0 + z \int_0^\infty t^{z-1} e^{-t} dt \\ &= z \times \Gamma(z)\end{aligned}$$

Theorem 3.1: The gamma function extends the factorial.

Figure 3.20: The gamma function



To integrate, create a `GammaFunction` class that takes no parameters in the constructor, and has a single method:

```
const double k_delta = 0.001;
const double k_start = 0.0;
const double k_end = 100.0;

double GammaFunction::operator() (double parameter) const
{
    GammaFunctionBase gfb(parameter);

    Integrate intGFB(&gfb, k_delta);

    Interval interval(k_start, e_Open, k_end, e_Open);

    return intGFB.integrate(interval);
}
```

This class gives excellent answers. However, it's slow. The gamma distribution (below) calls the gamma function thousands of times. There are faster (but less easy to understand) ways to compute the gamma function. The CD includes one based on the work of Viktor T. Toth from <http://www.rskey.org/gamma.htm>.

PROBLEMS: (Answers on page 291)

1. (Easy) What happens when the gamma function takes numbers less than 1? Make a graph.

The gamma distribution

Parameters	$\lambda > 0$ is the shape parameter (real) $\theta > 0$ is the scale parameter (real)
Support	$x > 0$ is the position
Density	$f(x; \lambda, \theta) = \frac{x^{\lambda-1} e^{-x/\theta}}{\Gamma(\lambda)\theta^\lambda}$
Mean	$\lambda\theta$
Standard Deviation	$\sqrt{\lambda}\theta$

Properties 3.9: Properties of the gamma distribution

gamma distribution The binomial distribution can be thought of as a sum of Bernoulli trials. In the same way, the **gamma distribution** can be thought of as a sum of exponential distributions. If an exponential distribution has parameter λ , then the sum of θ different exponential distributions is the gamma distribution(λ, θ).

The gamma distribution $\text{gamma}(\lambda, \theta)$ represents the sum of θ exponential random variables, each with a mean of λ .

Example 3.16 Examples of gamma distributions

The gamma distribution is the sum of exponential distributions. Examples similar to the exponential distribution work for the gamma distribution.

- Hours until the third telephone calls received in a day.
- Time until the fourth customer enters a shop.
- Time until the fifth fuse fails.
- Time the sixth bug hits on a Southern U.S. highway.
- Time until the seventh request to a web server.
- Time until the eighth missed note by a musician.
- Time until the ninth click of a Geiger counter measuring radiation.
- Distance until the tenth flaw in a copper wire.
- The number of characters until the eleventh typo in a book.
- The time until the twelfth accident in a factory.

Figure 3.21: Gamma: shape = 0.5, 1, 1.5, 2, 3; scale=2

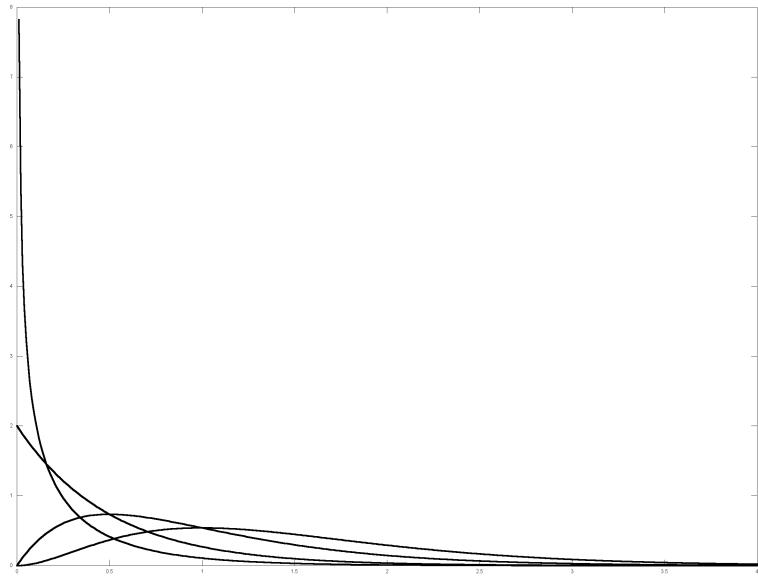
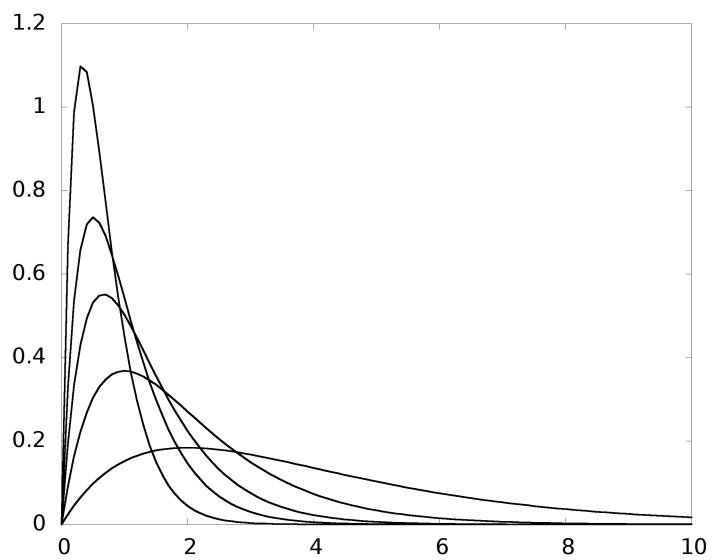


Figure 3.22: Gamma: shape = 2; scale=0.5, 1, 1.5, 2, 3



The code for the Gamma distribution follows:

The constructor can accept either scales or rates by using a third boolean parameter.

```
Gamma::Gamma(double shape, double scaleOrRate, bool \
    isScale)
{
    if (shape > 0.0 && scaleOrRate > 0.0) {
        m_shape = shape;

        if (isScale)
            m_scale = scaleOrRate;
        else // !isScale -- the second parameter \
            represents a rate parameter.
            m_scale = 1.0 / scaleOrRate;
    }
    else
        throw std::out_of_range("Both parameters must be \
            greater than 0.");
}
```

The density function defines the gamma distribution.

```
double Gamma::density(double x) const
{
    return std::pow(x, m_shape - 1) * std::exp(-x / \
        m_scale) /
        (m_gf(m_shape) * std::pow(m_scale, m_shape));
}

double Gamma::random() const
{
    int countShape;
    double total;
    Exponential exp(m_scale, true);

    // This routine only works for integer shapes.
    total = 0.0;
    for (countShape = 0; countShape < m_shape; \
        countShape++)
    {
        total += exp.random();
    }

    return total;
}
```

PROBLEMS: (Answers on page 291)

1. (Easy) The figures don't specify which curve matches which parameters. So, which curve of figure 3.21 on page 141 has shape 0.5? Which curve of figure 3.22 on page 141 has scale 0.5?
2. (Medium) You want a gamma curve with mean m and standard deviation s . What parameters would you use?

chi-squared distribution

One significant distribution derived from the gamma distribution is the **chi-squared distribution**. If the distribution is $\text{gamma}(\lambda = \nu/2, \theta = 2)$, then the distribution is often written as $\chi^2(\nu)$. This parameter, ν , is often called the number of **degrees of freedom** for the chi-squared distribution.

chi-squared distribution

degrees of freedom

We know that the gamma distribution is the sum of many exponential distributions. In the same way, if X_i are a set of k independent normally-distributed variables each with mean 0 and standard deviation 1, then $\sum_{1 \leq i \leq k} (X_i)^2$ is distributed according to the $\chi^2(k)$ distribution.

In other words, the following function generates random χ^2 variables:

```
double ChiSquare::random() const
{
    int i;
    Normal norm(0, 1);
    double sum = 0.0;
    double U;

    for (i=0; i<m_df; i++) {
        U = norm.random();
        sum += U*U;
    }

    return sum;
}
```

The most important use for the chi-squared distribution is Pearson's chi-squared test, listed on page 185.

PROBLEMS: (Answers on page 292)

1. (Easy) Write source code that implements a chi-squared distribution. Use the gamma distribution.
2. (Medium) Derive the distribution function, mean, and variance for the chi-squared distribution. Use the gamma distribution.

Parameters	$\alpha > 0$ is the first shape parameter (real) $\beta > 0$ is the second shape parameter (real)
Support	$0 \leq x \leq 1$ is the position
Density	$f(x; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}$
Mean	$\frac{\alpha}{\alpha+\beta}$
Standard Deviation	$\sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}$

Properties 3.10: Properties of the beta distribution

3.6.6 The Beta distribution

Among the basic distributions, the beta distribution is the most abstract.

Assume that two Poisson processes have the same parameter, λ . Imagine that you're interested in the ratio of the first process against the total time used by both processes. Let A represent the time until the α th success in the first process and B represent the time until the β th success in the second process. Then the $\text{Beta}(x; \alpha, \beta)$ distribution describes the ratio of $X = \frac{A}{A+B}$.

An easier way to understand the Beta distribution: Let U be a uniform distribution on $[0, 1]$. Choose n numbers from U , and let X_k be the k th order statistic among those n numbers. Then the probability distribution for X_k is a beta distribution with parameters k and $n - k + 1$.

Figure 3.23: Beta: side1 = 0.5, 1, 1.5, 2, 3; side2 = 2

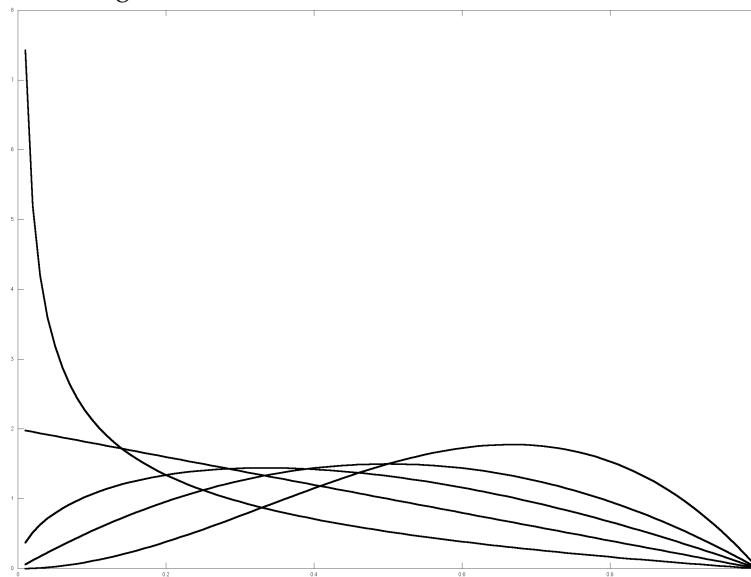
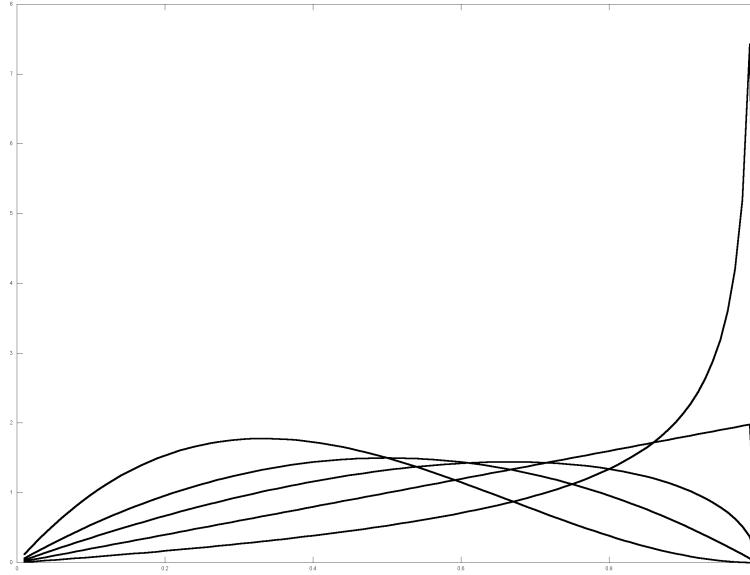


Figure 3.24: Beta: side1 = 2; side2 = 0.5, 1, 1.5, 2, 3



Example 3.17 Some uses for the beta distribution.

- If x_1, x_2, \dots, x_n are taken from the uniform distribution between 0 and 1, then the order statistic $x_{(k)}$ follows the beta distribution with $\alpha = k$ and $\beta = n + 1 - k$.
 - The beta distribution models some events that are constrained to an interval with a minimum and maximum value, and no more than one (maximum or minimum) value.
-

The source for the Beta distribution follows. The function `m_gf` is the gamma function.

```
Beta:::Beta (double shape1, double shape2)
{
    srand48((long int) std::time(NULL));

    if (shape1 < 0 || shape2 < 0)
        throw std::out_of_range("Error: the parameters
            for the Beta distribution must be non-
            negative.");

    m_shape1 = shape1;
    m_shape2 = shape2;
```

```

}

double Beta::density(double x) const
{
    if (x >= 0 && x <= 1)
        return m_gf(m_shape1 + m_shape2) / (m_gf(\n
            m_shape1) * m_gf(m_shape2)) *
            std::pow(x, m_shape1 - 1) * std::pow(1 - x, \n
                m_shape2 - 1);
    else
        return 0;
}

```

The random variable comes from the definition of the Beta distribution. This function only works with integer values of shape1 and shape2.

```

double Beta::random() const
{
    Gamma gamma1(m_shape1, 1.0, false);
    Gamma gamma2(m_shape2, 1.0, false);
    double rand1;
    double rand2;

    rand1 = gamma1.random();
    rand2 = gamma2.random();

    return rand1 / (rand1 + rand2);
}

```

3.7 Final Problems

Let's bring the previous material together.

PROBLEMS: (Answers on page 292)

1. (Easy) For which distributions is the mean always equal to the median?
2. (Easy) Which distribution is most appropriate to describe...
 - (a) The weight of a newborn baby?
 - (b) The distance that a golf ball travels?
 - (c) A gambler's number of wins or losses after 50 games?

Chapter 4

Introduction to Statistical Tests

4.1 Introduction

In the previous chapter, you were given parameters and a distribution and you were asked to generate data. Statistical tests are the opposite: you are given data and a distribution and asked to find what parameters match the data. This reversal loses information. Our best estimates for the parameters act like random numbers with a distribution.

This chapter covers material that is useful for performing statistical tests.

4.2 Certainty

What does a statistician mean by certainty? The same story has happened many times:

The boardroom's mahogany table has been polished by the nervous sweat of hundreds of vice-presidents over dozens of years. Sitting around the table are gray-haired executives, wearing suits that each cost enough to replace all computer servers in the company. The CEO stares at the young statistician. "I saw your numbers," she says. "How certain are you of the values?"

The statistician clears her throat. "I'm ninety percent sure of them," she says.

The stares from the assembled board members gives the statistician frostbite. "Come back when you *know* your numbers," says the CEO.

In normal speech, being ninety percent certain of something means being very uncertain of something. For a statistician, being ninety percent certain of

something means that, on average, nine times out of ten, the numbers are right.

This section will help you to understand what certainty means.

4.2.1 How to estimate

Distributions measure uncertainty. One example of uncertainty is our internal knowledge of facts and figures.

Take a question like: What is the longitude of Cairo, Egypt?

A first thought is: I should look that up.

What if, sitting here, right now, I needed to estimate that question without using any references. Could I give any answer?

I am 100% certain that the longitude is between $-\infty$ and $+\infty$. Since I know the definition of longitude, I am 100% certain that the longitude is between $180^\circ E$ and $180^\circ W$. Because I know that Cairo is in the Eastern hemisphere, I am certain that Cairo's longitude is between 0° and $180^\circ E$.

If I did not need to be 100% certain, could I give a better estimate? Yes. Egypt is on the eastern side of the Mediterranean. And the 0° longitude is not too far from the western edge of the Mediterranean.

I can make an estimate in a few ways:

1. I do not know how long the Mediterranean Sea is. But I know that the Continental United States is 3,000 miles wide. Comparing my mental maps of the United States and the width of the Mediterranean, they seem to be comparable. Therefore, I estimate that the Mediterranean is between 1,500 miles and 6,000 miles wide (half the width to twice the width). I know that a degree is about 70 miles long. Therefore, the easternmost point of the Mediterranean would be between $\frac{1500}{70} = 21.4^\circ$ and $\frac{6000}{70} = 85.7^\circ$ to the east of the easternmost point of the Mediterranean.
2. I can try to mentally guess how many Mediterranean Seas would fit between 0° and $180^\circ E$. I mentally guess that between 3 and 10 Mediterranean Seas could fit. Therefore, the easternmost point of the Mediterranean would be between $\frac{180}{3} = 60^\circ W$ and $\frac{180}{10} = 18^\circ W$ to the east of the westernmost point of the Mediterranean.

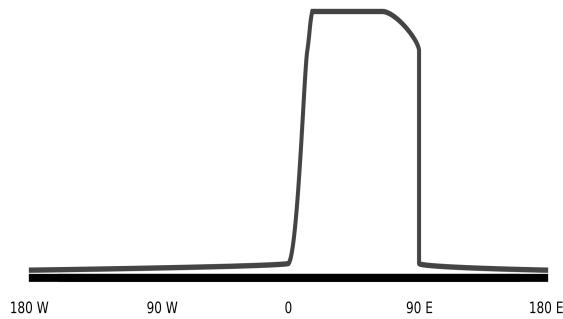
Now, I do not know the distance from zero-longitude to the western point of the Mediterranean, but I am reasonably certain that it is close. I further do not know the distance of Cairo's longitude to the longitude of the eastern point of the Mediterranean, but I know that Cairo is on the Nile, and I know that the Nile flows into the far west of the Mediterranean. I therefore estimate that I am no more than 5 degrees off on either side.

Putting together the information, I am very certain that the lower limit for the longitude of Cairo is $18^\circ E$ (the smaller of my lower bounds) - 5° (my probable error) = $13^\circ E$, and the upper limit for the longitude of Cairo is $85.7^\circ E$ (the larger of my two upper bounds) + $5^\circ E$ (my probable error) = $90.7^\circ E$. I am less

certain that the longitude of Cairo is between $21.4^\circ E$ (the larger of my lower bounds) - 5° (my probable error) = $16.4^\circ E$ and $60^\circ E + 5^\circ = 65^\circ E$.

I can express my knowledge and ignorance as a probability distribution in graph 4.1 on page 149.

Figure 4.1: My personal estimate of the longitude of Cairo (Not to scale!)



So... what is the actual longitude of Cairo? According to <http://www.infoplease.com/ipa/A0001769.html>, it is $31^\circ 21'$, or about 31.3° . It is within my estimate.
(*phew*)

Try this exercise.

Example 4.1 Certainty exam

You are not expected to know this material. This quiz tests your estimation skills, not your knowledge. Please do not use any resources for this quiz other than your memory and a blank sheet of paper.

These questions have numeric answers. For each question, give a lower bound and an upper bound where the answer can be found. Answer each question with 90% certainty: give a range so that your answer has a 90% chance of including the right answer.¹

1. What was the price of the Model T Ford in 1925?

¹All information in these questions come from [Wright, 2004].

2. On average, how many shares were traded daily on the New York Stock Exchange in 1830?
3. How many square kilometers is the land area of Antarctica?
4. How long, in kilometers, is the Yangtze River?
5. In what year did Tutankhamen's reign of Egypt start?
6. From the start of the first Punic War until the end of the third Punic War, how long did the three Punic Wars between Rome and Carthage last?
7. What year was the earliest known written Swahili epic, *Ubendi wa Tambuka*, created?
8. In 1920, Westinghouse made regularly scheduled evening broadcasts from KDKA in Pittsburgh. How many kilowatts of power did its transmitter have?
9. How many people died of cardiovascular diseases in 2001?
10. How many hours does it take for Jupiter to rotate on its axis?
11. How many kilograms of sulfuric acid are produced in the United States each year?
12. In the Paleocene epoch, early primates, early horses, and rodents evolved. How long ago did this epoch start?
13. How many major-league baseball games did Jackie Robinson take part in?
14. How many athletes took part in the XXVII Olympic Games in 2000 in Sydney, Australia?
15. The first Indianapolis 500 took place in 1911, and was won by Ray Harroun. What was his average speed in miles per hour?

The answers are on page 293. Give yourself one point for every range that included the correct answer, and no points for any range that did not include the correct answer (or that you gave no answer to.)

Please do not read beyond this line until you have tried the quiz.

Without the *New York Times Guide to Essential Knowledge* in front of me, I would not know the answers. Not one of them. The questions dealt with topics that I have no personal knowledge about. Thank heavens, the quiz asked for estimates, not exact values.

The quiz asked for a range of values that has a 90% chance of including the true value. That statement is the key. Unless you have specific information about any of those trivia questions, your guess must be wide enough to cover the real answer. This range is often much wider than what people intuitively consider "90% confident".

When given this quiz, most people make too small ranges.
Good estimates come from a few rules:

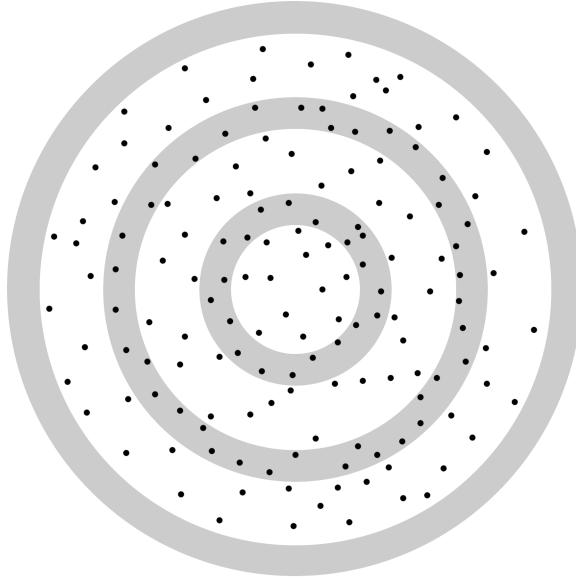
1. Estimates based on multiple, independent sources are better than estimates based on a single source.
2. Estimates based on information similar to the question are better than estimates from more widely-ranging topics.
3. The less certain that you are, the less accurate your estimate must be.
4. Sometimes, the only estimate that you can make is that a value is between $-\infty$ and ∞ .

If you read ahead, and did not try the quiz... please go back and try it. Even with this extra information, please experience the difference between what is emotionally considered 90% confidence and reality.

4.2.2 Accuracy and Precision

In statistics, tension happens between three goals: certainty, accuracy, precision, and small amount of data collected. If a test is **precise**, then it has small variability. If a test is **accurate**, then the variability covers the goal. Certainty is a measure of accuracy not precision. The most common illustration of these two ideas are on graph 4.2 and graph 4.3 on page 152.

Figure 4.2: An illustration of accurate but not precise

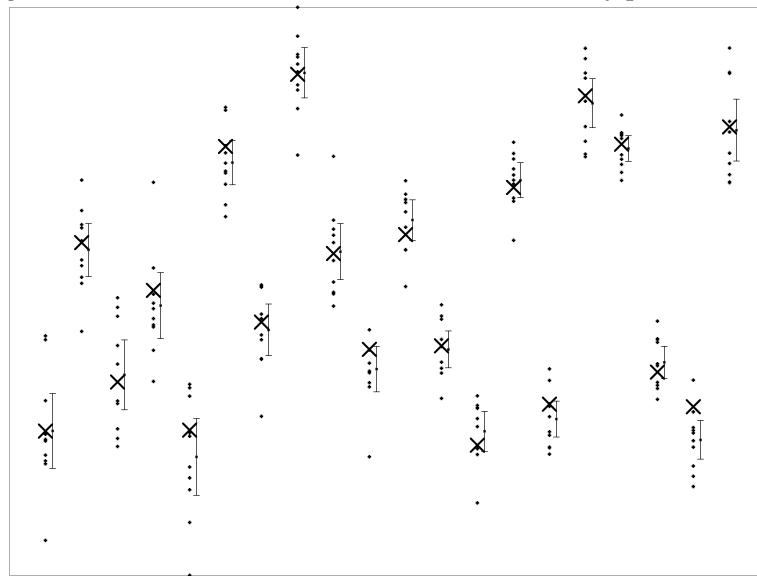


As an illustration, graph 4.4 on page 152 gives twenty experiments. For each experiment, ten data points were chosen from a normal distribution. Next

Figure 4.3: An illustration of precise but not accurate



Figure 4.4: The actual and estimated means for twenty pieces of data.



to the points is the 95% accurate approximation of where the mean is, from techniques that will be presented later. Finally, the X represents the actual mean of the distribution. Exactly as expected, nineteen of the twenty experiments had the actual mean inside the 95% accurate estimates of the mean.

Using certainty

There are two ways to use certainty:

- Given a certainty, determine a range.
- Given a range, determine a certainty.

Statisticians often start with a given certainty, called the **alpha value** or **p-value**. The most common certainty is 95%: that is, nineteen out of twenty times, the test will be correct. This choice is out of habit; in practice, a 95% certainty is usually a good balance between certainty and precision. However, the customer sometimes wants different certainties. For example, medical devices or spacecraft may require 99.9% certainty or higher, and a rough first screening for a topic may need just 80% certainty.

If c is a given certainty between 0 and 1 and d is a Distribution, then $d.quantile(0.5 - c/2)$ and $d.quantile(0.5 + c/2)$ represent a boundary of a range around m with the certainty c of happening.

Other times, statisticians have a range, and want to determine the certainty within that range.

If a and b are the lower and upper limits of a range, and d is a Distribution, then $d.cumulative(b) - d.cumulative(a)$ is the certainty within that range.

PROBLEMS: (Answers on page 293)

1. (Easy) I wrote:

$d.quantile(0.5 - c/2)$ and $d.quantile(0.5 + c/2)$ represent a boundary of a range with the certainty c of happening.

Why is this a range, and not the range? Find another range around with certainty c .

4.2.3 Null and Alternate Hypotheses

Every statistical test is based around a hypothesis. A common framework for writing a hypothesis is the null hypothesis and the alternate hypothesis.

The **null hypothesis** is the “expected, standard” probability distribution for **null hypothesis** the data. It is often written as H_0 .

The **alternate hypothesis** is the negation of the null hypothesis, H_0^c . For example, if the null hypothesis is $X = 0$, then the alternate hypothesis is $X \neq 0$. If the null hypothesis is $Y < 10$, then the alternate hypothesis is $Y \geq 10$. The alternate hypothesis is often written as H_A or, more rarely, H_1 .

Example 4.2 Null Hypothesis

1. If we are testing whether a coin is biased, the null hypothesis would be $\Pr(\text{heads}) = \Pr(\text{tails}) = 0.5$.
 2. If we are testing whether an IQ test is properly normalized, the null hypothesis would be normal (μ, σ) where μ is the expected mean and σ is the expected standard deviation.
 3. If we are testing whether the number of characters between typos in a book follow an exponential distribution, the null hypothesis would be exponential (λ) , where λ is the expected number of characters between typos.
-

Statisticians test against the null hypothesis: find whether the data meets the expectation, or whether the data does not meet the expectation. If the data does not meet the expectation, statisticians say that “the data rejects the null hypothesis.” If the data meets the expectations, statisticians say that “the data does not reject the null hypothesis.”²

Type I and II errors

This framework leads to two kinds of possible errors:

Type I error A **Type I error** occurs when the statistical test rejects the null hypothesis even though the null hypothesis is true. In other words, the test says that the value is not typical when it is.

Type II error A **Type II error** occurs when the statistical test accepts the null hypothesis even though the null hypothesis is false. In other words, the test says that the value is typical when it is not.

This distinction is made clearer in graph 4.5 on page 155.

Both terms are used both for the occurrence of an error and for the chance that a Type I or Type II error occurs.

²Statisticians don't love double-negatives.

Figure 4.5: The difference between type I and type II errors

	Null Hypothesis True	Null Hypothesis False
Test accepts null hypothesis	True Positive	Type I Error
Test rejects null hypothesis	Type II Error	True Negative

Example 4.3 Type I and Type II errors, example 1

In a court of law, a defendant is assumed innocent until proven guilty. In other words, the null hypothesis is that the defendant is innocent.

A Type I error occurs in the courtroom when a jury finds the defendant guilty even though she is actually innocent.

A Type II error occurs in the courtroom when a jury finds the defendant innocent even though she is actually guilty.

Example 4.4 Type I and Type II errors, example 2

When testing a new drug, the null hypothesis is that the drug is no better than a placebo. In other words, the null hypothesis is that the drug is ineffective.

A Type I error occurs when the company considers the drug effective when it is ineffective.

A Type II error occurs when the company considers the drug ineffective when it is effective.

As above, let H_0 represent the null hypothesis, and let H_A represent the alternate hypothesis. Let E be the evidence for H_0 . Then the probability of a Type I error can be written $\Pr(H_A|E)$ and the probability of a Type II error can be written $\Pr(H_0|E^c)$.

As a general rule of thumb, given the same amount of data, most tests can trade one kind of error for the other.

Among medical devices, different terms are used:

The **sensitivity** of a device is, among those who have property H_0 , the percent that the device correctly identifies as having H_0 . If E represents the evidence, then sensitivity represents $\Pr(H_0|E)$. If p_1 is the chance of Type I error, then the sensitivity is $1 - p_1$.

The **specificity** of a device is, among those who do not have property H_0 , the percent that the device correctly identifies as not having H_0 . If E^c represents evidence against the null hypothesis, then specificity represents $\Pr(H_0^c|E^c)$. If p_2 is the Type II error, then the specificity is $1 - p_2$.

PROBLEMS: (Answers on page 293)

1. (Easy) Assume that you have two coins, indistinguishable except how they flip. One is fair: there is a 50% chance that it comes up tails. The other is weighted: there is a 70% chance that it comes up tails.

You flip the coin n times, and count the number of tails. Where you put the d , the split between assuming the coin is fair or weighted will determine your chance of a type I or type II error.

The null hypothesis is that the coin is fair. As d increases, are you trading Type I errors for Type II errors, or vice-versa?

2. (Easy) At the time I write this book, Wikipedia³ says that increasing specificity decreases the probability of Type I errors and increases the probability of Type II errors.

Is this statement always accurate?

ROC Curves

Tests can often be modified without changing the amount of data gathered. These modifications often lower Type I errors at the expense of raising Type II errors, or vice-versa. In many cases, allowing a slight increase in the chance of one type of error can yield a strong decrease in the chance of the other type of error. One way to demonstrate these changes is through Receiving Operating Characteristic (ROC) curves.

An ROC curve maps the sensitivity against the specificity for a set of tests. An ROC curve demonstrates the trade-offs that happen by varying parameters in a set of tests.

Example 4.5 ROC Curves

You work for a factory that makes trick coins. One set of coins is perfectly weighted: flipping one gives a 50% chance that heads comes up and a 50% chance that tails comes up. The other set is unbalanced: flipping one gives a 30% chance that heads comes up and a 70% chance that tails comes up. They are otherwise identical.

One day, the two sets fell into the same box and got mixed together. You now have one coin in your hand, and you want to determine which kind it is. How do you classify it?

One way to classify it is to flip the coin 100 times, then classify based on the number of times it turned up tails. If it turned up tails more than 70 times, you would be (fairly) certain that it is the weighted coin. If it turned up tails fewer than 50 times, you would be (fairly)

³http://en.wikipedia.org/w/index.php?title=Type_I_and_type_II_errors&oldid=327243860

certain that it is the unweighted coin. But the values between 50 and 70 are more iffy.

Let d be a fraction between 0.50 and 0.70 that you choose. If the coin flips tails more frequently than d on average, we identify the coin as unbalanced. Let n be the number of flips that we make. Finally, let the equally-weighted coin be the null hypothesis.

Sensitivity is, among the times that the coin actually is equally-weighted, the chance that we identified it as such. Specificity is, among the times that the coin is unbalanced, the chance that we identified it as such. Both depend on our choice of n and d .

What is the sensitivity and specificity? In the following computation, I use a nonstandard summation: instead of adding 1 with each computation, I add $\frac{1}{n}$.

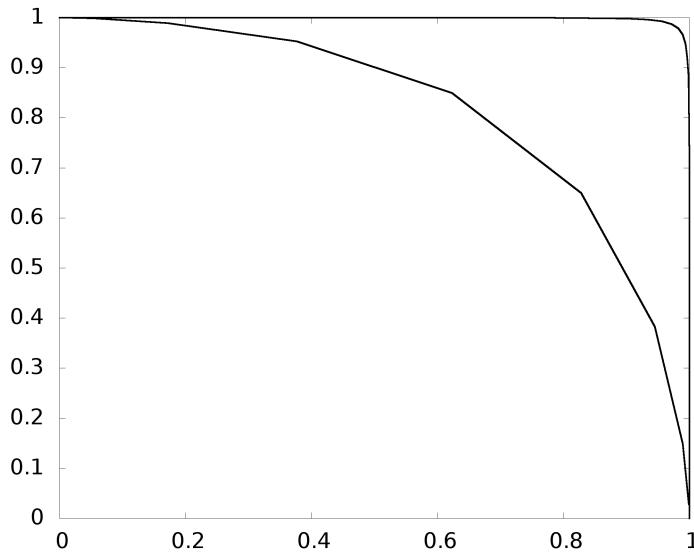
$$\text{sensitivity} = \sum_{\substack{i=0 \\ i=0 \text{ by } \frac{1}{n}}}^d \text{binom}(i \times n; n, 0.5)$$

$$\text{specificity} = \sum_{\substack{i=d+\frac{1}{n} \\ i=d+\frac{1}{n} \text{ by } \frac{1}{n}}}^1 \text{binom}(i \times n; n, 0.7)$$

The plot for $n = 10$ or $n = 100$, and d varying from 0 to 1 can be found on page 158.

Notice a few things about the ROC curve:

1. Most sets of tests will have a point where sensitivity equals 100% and specificity equals 0%, and the opposite point.
2. The best possible ROC curve has a point where sensitivity equals specificity equals 100%. The test perfectly separates the null hypothesis and the alternate hypothesis.
3. The worst realistic ROC curve follows the diagonal line from the lower-left corner to the upper right corner. This set of tests acts independently of the data: if s is the sensitivity, then no matter the data, it has an s probability of being labeled in the null hypothesis.

Figure 4.6: Two ROC curves when $n = 10$ or $n = 100$ and d varies from 0 to 1

PROBLEMS: (Answers on page 294)

1. (Easy) I said that the worst *realistic* ROC curve follows a diagonal line. Assume that you have a test that is always below the diagonal line (except at the (0%, 100%) corners.) What would you do with this test?
2. (Medium) I have dozens of decks of nine cards. Each deck contains either five red cards and four black cards, or it contains four red cards and five black cards. These decks occur with equal probability. I allow observers to choose a deck, and to draw cards successively at random from the deck with replacement.
Abigail draws six cards; each time, the card is red.
Becky draws six hundred cards. 303 times, the card is red; 297 times, the card is black.
Both people believe that they have a deck with five red cards and four black cards. Who has stronger empirical evidence for her prediction?

4.3 Outliers

Outliers are unexpected pieces of data. They include the kid with the 171 I.Q., the fellow who claims twelve dependents on his income tax form, or the person who will only speak Esperanto.

outlier An **outlier** is defined as any point that is more than 1.5 times the interquartile range⁴ from the mean. An **extreme outlier** is defined as any point more than 3 times the interquartile range from the mean.

extreme outlier

Many books will indirectly or directly tell you to throw out your extreme outliers from your data analysis. For example, many books recommend using a trimmed mean⁵ for analysis. They point out that outliers can dramatically change answers, and they cause wider uncertainty.

This book emphatically takes the opposite path. Data is hard to gather. Throwing it away is wasteful. We shall go on to the end. We shall fight for our data in France, we shall fight for our data on the seas and oceans, we shall fight with growing confidence and growing strength for our data in the air. We shall defend our island, whatever the cost may be. We shall fight on the beaches, we shall fight on the landing-grounds, we shall fight in the fields and in the streets, we shall fight in the hills. We shall never surrender! — err, have I gone overboard?⁶

Outliers are an important check of the data. Check them carefully. For example, the report of the 171 I.Q. may have been a mistype for 117: check the original examination. The fellow who claims twelve dependents may need an explanation that his cats do not count as dependents. And the person who only speaks Esperanto has no cure.

Outliers are often the most important part of the data. An extreme example comes from seismic data. According to the USGS⁷, cutting off just the top 0.1% of seismic activity would eliminate all earthquakes of magnitude 5 or greater.

If you want to lower the width of your uncertain range, the most honest way is to use (and report) a smaller p-value.

4.4 Fitting equations

The next sections will contain many questions of the form, “Given data, what parameters best fit an equation?”

Example 4.6 Fitting equations, example 1

x	y
2.0	4.0
5.0	10.0
12.0	24.0

You are asked to fit the above three pieces of data to the function $y = ax$. The best fit is obviously $a = 2$.

⁴See page 32.

⁵See page 37.

⁶Yes. Apologies to Winston Churchill.

⁷<http://neic.usgs.gov/neis/eqlists/eqstats.html>

Example 4.7 Fitting equations, example 2

x	y
2.0	6.0
5.0	10.0
12.0	12.0

You are asked to fit the above three pieces of data to the function $y = ax$. No obvious “best fit” exists.

Several questions are important before you fit equations:

- Do you have the correct equation? You will always be able to find a best set of parameters, but these techniques cannot tell you whether the equation is good.
- Are the data typical? Do they come from the whole dataset? Are they a true random sample from your data set? Do they cover all the range that you are interested in?
- If your function is $y = f(x)$, then are the y -values uncorrelated? If the y -values each depend on a factor z , then you might want to do your analysis based on that z .
- Are any measurement errors on the y values about the same? If the measurement errors vary, you might want to apply a function on the data so that the measurement errors are about the same.

The next two sections will cover fitting equations in two forms:

1. When you fit parameters to a formula, the least squares method is useful.
2. When you fit parameters to a distribution, the maximum likelihood method is useful.

4.4.1 Least Squares

Assume that you have a parametrized function, $y = f(x)$ – that is, the function has parameters that can change. You also have data x_1, x_2, \dots, x_n , and results y_1, y_2, \dots, y_n . You’re trying to find the best parameters to best match $f(x_i) = y_i$.

The least-squares distance is $\sum_{i=1}^n (f(x_i) - y_i)^2$. Fill in the equation (with parameters) instead of $f(x)$, then find the minimum least-squares distance using techniques like those of section 1.7.3 on page 44.

Example 4.8 Fitting equations, using least-squares

x	y
2.0	6.0
5.0	10.0
12.0	12.0

Fit the above three pieces of data to the function $y = ax$. The least-squares distance is

$$\begin{aligned} l &= (2a - 6)^2 + (5a - 10)^2 + (12a - 12)^2 \\ &= 4a^2 - 24a + 36 + 25a^2 - 100a + 100 + 144a^2 - 288a + 144 \\ &= 173a^2 - 412a + 280 \end{aligned}$$

To find the minimum, take the derivative of l and solve for zero:

$$\begin{aligned} 346a - 412 &= 0 \\ a &= \frac{412}{346} \end{aligned}$$

The best equation is $y = \frac{412}{346}x$. Although this answer is the best possible answer, it is not a very good answer. The line is shown in graph 4.7 on page 162.

This technique can be extended for equations with more one parameter.

4.4.2 Maximum Likelihood

Assume that you know that data comes from a distribution, D with just one parameter, θ . For example, the data could have come from the geometric, Poisson, or exponential distribution. Assume that we have n observations, x_1, x_2, \dots, x_n from $D(\theta)$, but we do not know the value for θ .

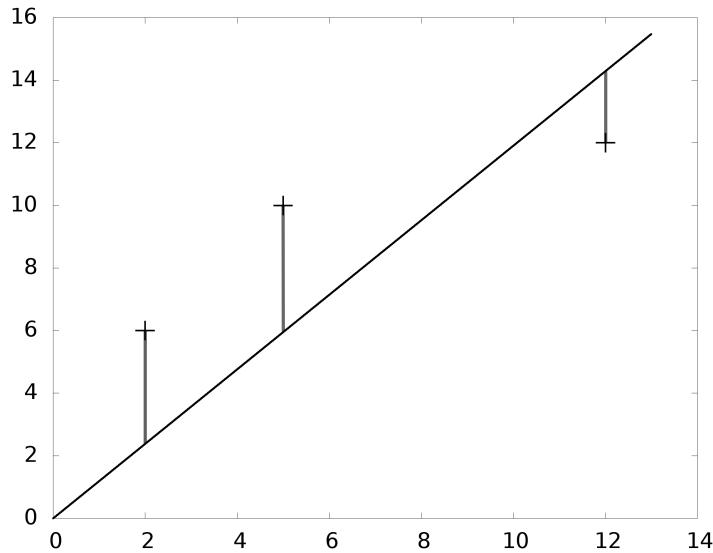
From each θ , you can compute a probability associated with getting the values observed. The **maximum likelihood** is the value of θ that gives the **maximum likelihood** highest likelihood.

In the previous chapter, a probability distribution inputs a parameter, then outputs data according to the distribution. In a maximum likelihood test, the probability distribution inputs the data, then outputs the parameter.

Most books write the likelihood of the data x_1, x_2, \dots, x_n and the parameter θ as $L(x_1, x_2, \dots, x_n; \theta)$. $L(x_1, x_2, \dots, x_n; \theta)$ means the same as $\Pr(\theta | x_1, x_2, \dots, x_n)$.

Two ideas make finding the maximum likelihood simpler:

Figure 4.7: The best-fit line, according to least squares



1. If the data is independent, then the likelihood of many pieces of data is the product of the likelihood of each individual piece of data. In symbols, if x_1, x_2, \dots, x_n are independent, then

$$\begin{aligned}
 L(x_1, x_2, \dots, x_n; \theta) &= \Pr(\theta | x_1 \cup x_2 \cup \dots \cup x_n) \\
 &= \Pr(\theta | x_1) \times \Pr(\theta | x_2) \times \dots \times \Pr(\theta | x_n) \\
 &= \prod_{i=1}^n \Pr(\theta | x_i) \\
 &= \prod_{i=1}^n L(x_i; \theta)
 \end{aligned}$$

2. Likelihood often deals with very small numbers. Often, examining the logarithm of the likelihood is nice:
 - (a) The logarithm preserves ordering: If $a < b$, then $\log(a) < \log(b)$ and vice-versa.
 - (b) Instead of taking the product of many tiny numbers (which can cause underflow), the logarithm takes the sum of many negative numbers.
 - (c) We are often interested in the most likely value for the maximum likelihood. It is easier to find the derivative for a sum of values than it is to find the derivative for a product.

In symbols, $\log(L(x_1, x_2, \dots, x_n; \theta))$ is written $l(x_1, x_2, \dots, x_n; \theta)$.

Example 4.9 Maximum Likelihood Example

You are counting the number of typos per page of a book, a process that follows the Poisson distribution. You take a sample of ten pages. On six of the pages, no typos are found. On three of the pages, one typo is found. On one page, two typos are found. What is the maximum likelihood value of λ ?

The formula for a Poisson distribution is $\frac{\lambda^x e^{-\lambda}}{x!}$. Since the typos on each page are independent of the typos on every other page, you can take the product of the Poisson processes:

$$\begin{aligned} L[\lambda|x] &= \left(\frac{\lambda^0 e^{-\lambda}}{0!}\right)^6 \left(\frac{\lambda^1 e^{-\lambda}}{1!}\right)^3 \left(\frac{\lambda^2 e^{-\lambda}}{2!}\right)^2 \\ &= \frac{\lambda^5 e^{-10\lambda}}{2} \\ l[\lambda|x] &= \log(L[\lambda|x]) \\ &= 5\log(\lambda) - 10\lambda - \log(2) \end{aligned}$$

Where does $l[\lambda|x]$ have its maximum? Take the derivative.

$$l'[\lambda|x] = \frac{5}{\lambda} - 10$$

$l'[\lambda|x] = 0$ when $\lambda = \frac{1}{2}$. Therefore, our best estimate for θ is $\frac{1}{2}$.

PROBLEMS: (Answers on page 295)

1. (Medium) In general, is the maximum likelihood a distribution, according to the Kolmogorov Axioms (see page 59.) If so, prove it. If not, explain how to change it to become a distribution.

Every computer scientist will be asked one specific question. This one question makes or breaks careers, gets or loses contracts, and causes more problems than almost any other question. This question is often asked as if it were casual, at the end of a discussion about a proposed project. The question is never casual. The question is **How long will the project take?** (If you are a contractor, the question is phrased as **How much will the project cost?** It's the same question.) Answering this question wrong will always hurt your career. If you estimate too high, another person or group might get the job. If you estimate too low, you will overrun time or cost. Some people suggest intentionally estimating low, so that you at least get the job, then accepting the overrun. If you follow that bad advice, you will spend extra time and effort explaining to Very Important People why your program is late and over budget. Such presentations hurt job security.

The key to estimation is information. The best predictor of how long a program will take to write is how long previous programs took to write. Keep records of how long it took you to write programs, classes, tests, and documentation; use those records to guide future estimates.

The best book that I have read on the topic is Steve McConnell's *Software Estimation: Demystifying the Black Art* ([McConnell, 2006]).

Sidebar 4.1: The most important estimate

For most scientific endeavors, a result is publishable if it can be shown to have a 95% chance of being correct. This rule of thumb leads to a problem: When an idea becomes popular, dozens of experiments may be done on the same field. Even if idea has no relation to reality, one in twenty experiments will show a false positive — and therefore, be publishable. This creates a feedback loop: more scientists become interested and perform tests. One in twenty of this larger group of scientists will have a false positive, and be able to publish it.

Only when scientists are able to publish rebuttals to this new idea can the system correct itself.

Sidebar 4.2: The problem of publishing

Chapter 5

Statistical Tests for one variable

5.1 What statistical tests do

Often, a source of data has two features:

- The source of data could potentially generate a huge number of measurements.
- Time or expense restricts you to gathering a (relatively) small number of measurements from this source.

Examples of these sources include consumer or business surveys, scientific tests, and quality assurance on real objects.

Statisticians rarely get every possible piece of data from a source. Instead, they usually get a sample of a few pieces of data from the wealth of possible data. From this sample, they predict parameters for the whole data. For example, political interviews never ask a question of every voter in a district. Instead, they take a sample of several thousand voters. From that representative sample, they estimate what the voters as a whole think.

It might be that the source has a true distribution with true parameters. Even if we know the actual distribution, observations will only let us estimate the parameters for the distribution. These estimated parameters act like random variables: they have their own means, standard deviations, and other statistics.

Books use different sets of letters to represent the difference between the real, unknown, parameters and the random variables that estimates the real parameters. Like most books, this book uses Greek letters (μ, σ) to represent the real parameter and Latin letters (\bar{x}, s) to represent the computed, estimated random variables.

The most common example of an estimated parameter is the mean. Given a sample of data, what is the estimate of the mean for the data as a whole? This question comes in many disguises:

- Will our candidate win the election? (Is her voting block really above 50%)?
- Does our new recipe taste better than the competition's recipe tastes? (Is the mean rating of our recipe higher or lower than that of our competition?)
- Will my swarm of intelligent locusts consume at least ten cities before government troops learn their weakness for Appalachian polkas?

Example 5.1 The difference between the mean of a distribution and the mean of a sample

Assume that we have nine samples from a distribution: 1, 2, 3, 2, 1, 3, 3, 1, 2. Then the estimated mean for the distribution is 2.

But these random numbers came from the following distribution:

x	$\Pr(x)$
1	0.33
2	0.33
3	0.33
1002	0.01

The true mean of the distribution is $0.33 \times 1 + 0.33 \times 2 + 0.33 \times 3 + 0.01 \times 1002 = 12.0$.

Although example 5.1 demonstrates that the true mean of the distribution might be any value, most distributions have few outliers.

5.2 Bootstrapping

Bootstrapping estimates some parameters using the data distribution from section 3.4 on page 85. It creates distributions from relatively small amounts of original data. Bootstrapping assumes:

1. Each piece of data is chosen independently from the same original distribution.
2. The data retrieved is a fair representation for the original distribution as a whole.

If the original distribution has n members, then the most common bootstrapped distribution chooses n elements with replacement from the original data. In software, this can be written as:

```

DataDistribution bootstrap(const Distribution& orig, int size)
{
    int count;
    std::multiset<double> boot;

    for (count = 0; count < size; count++)
        boot.insert(orig.random());

    DataDistribution result(boot);
    return result;
}

```

The `orig` distribution might be a `DataDistribution`. In this case, the bootstrap often contains repeats of some elements of the original set and ignores other elements of the original set.

5.2.1 Estimating a distribution's mean using bootstrapping

Even with a limited sample of data, it is easy to find the mean of the sampled data. But by taking the average of many bootstraps, you can generate a bootstrapped distribution for the mean.

Bootstrapped distributions for the mean estimate how far the actual mean might be from the estimated mean. If the variation in the bootstrapped distribution is wide, then the estimated mean might be far from the actual mean. The converse is not true: a narrow variation does not guarantee that the estimate for the mean is good. (See example 5.1 on page 166 for a counterexample.)

```

DataDistribution bootstrapMean(const Distribution& orig,
    int sizeBootstrap, int sizeMean)
{
    int count;
    DataDistribution ddBoot;
    std::multiset<double> bootMean;
    double mean;

    for (count = 0; count < sizeMean; count++) {
        ddBoot = bootstrap(orig, sizeBootstrap);
        mean = ddBoot.mean(); // ****

        bootMean.insert(mean);
    }

    DataDistribution result(bootMean);
    return result;
}

```

By replacing the starred line, you can also get other distributions for functions of the data.

5.2.2 The accuracy of bootstrapping

How accurate is bootstrapping? If `dd` is the result of `bootstrapMean`, then the data distribution claims to have a 95%-certain answer between `dd.quantile(0.025)` and `dd.quantile(0.975)`. However, the bootstrapped distribution's accuracy is not as good as it claims to be. The true accuracy depends on the original distribution and the number of elements in the sample.

I ran the following code many times:

```
const int numCycle = 2000;

int count;
int cycle;
DataDistribution ddMean;
std::multiset<double> *msSample;
Normal norm(0, 1);

for (count=3; count<100; count++) {
    int countIncluded95 = 0;

    for (cycle = 0; cycle < numCycle; cycle++) {
        msSample = new std::multiset<double>();

        for (member = 0; member < count; member++)
            msSample->insert(norm.random());

        DataDistribution ddSample(*msSample);
        ddMean = bootstrapMean(ddSample, count, count);

        if (ddMean.quantile(0.025) < 0.0 && ddMean.quantile(0.975) > 0.0)
            countIncluded95++;

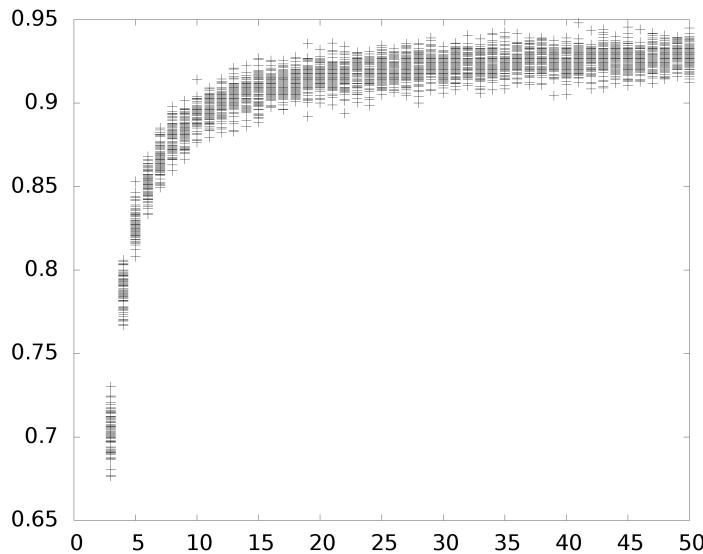
        delete msSample;
    }
}
```

A graph of the results can be found in figure 5.1 on page 169. The graph has a line at 95%, the accuracy that the representation claims.

Notice two things in figure 5.1 on page 169:

- Accuracy doesn't reach 95%, even when 100 samples are used.
- Accuracy grows with more samples.

Figure 5.1: Bootstrapping Accuracy



PROBLEMS: (Answers on page 296)

1. (Easy) The bootstrap estimate for the variance (or standard deviation) is usually poor. Its values are almost always smaller than the real variance. Why?
2. (Medium) How many of the original values appear in a bootstrap? Make a graph that, for n from 5 to 100, shows the distribution of the number of unique values in a bootstrap.

5.3 Estimating the mean of a normal distribution

This section covers many ways of answering: Given a set of data, what is the distribution for the estimate of the data's mean?

5.3.1 The normal approximation

When a source is affected by many independent variables, it tends to look like the normal distribution. The mean of more than 30 variables almost always acts like a normal distribution.

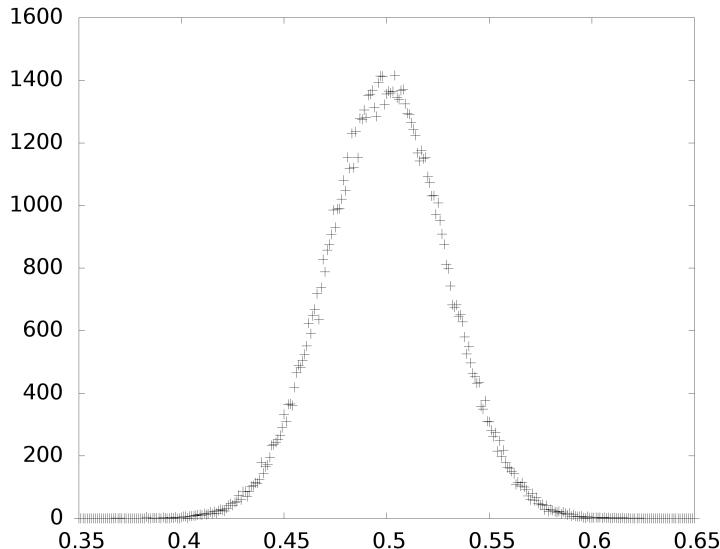
If $A = \{a_1, a_2, \dots, a_n\}$ is a sample, and if you know that σ is the standard deviation for the whole distribution¹, then the mean of n variables can be estimated by the normal distribution with mean $= \bar{a}$ and std dev $= \frac{\sigma}{\sqrt{n}}$.

Example 5.2 Mean of 100 uniform random numbers

The uniform distribution is very different than the normal distribution. How close is the mean of 100 uniform random numbers to a normal distribution? I ran the following code 10,000 times:

```
count = 100;
sum = 0.0;
for (i=0; i<count; i++)
    sum += drand48() / count;
```

10,000 runs of the above code made the following histogram:



This looks like a bell curve, which describes a normal distribution.

5.3.2 One-sided z-tests

Assume that even before you do any computation, you have a value, μ_0 , that is your estimate for a mean of a sample.

Also assume that you have lots of samples from a normal distribution: x_1, x_2, \dots, x_n . The mean of the samples is \bar{x} , and the standard deviation of the samples is s_x .

You think that the unknown mean of the distribution as a whole (written as **one-sided z-test** μ) is smaller than your estimate, μ_0 . This test is called a **one-sided z-test**.

¹You rarely get this information.

$$\begin{aligned} H_0 &: \mu \leq \mu_0 \\ H_A &: \mu > \mu_0 \end{aligned}$$

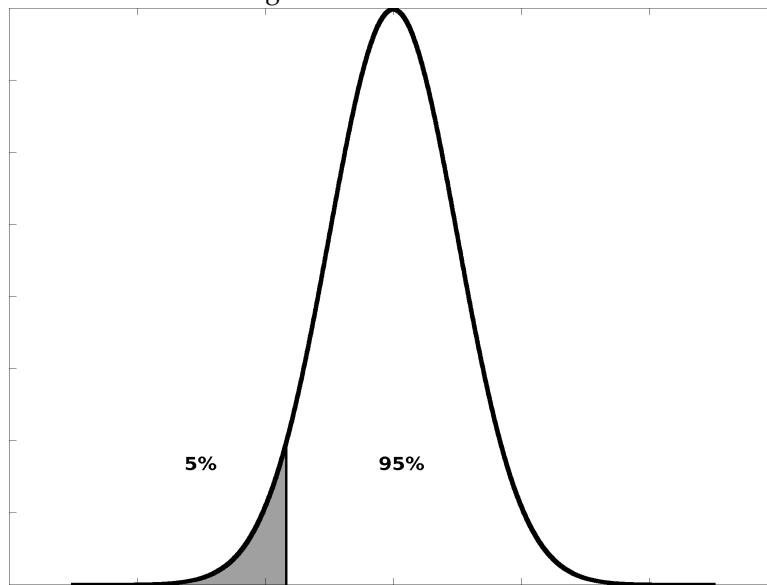
If you thought that μ was bigger than your estimate, then you have a slightly different one-sided z-test:

$$\begin{aligned} H_0 &: \mu \geq \mu_0 \\ H_A &: \mu < \mu_0 \end{aligned}$$

You choose which direction that the test happens.

If life were easy, we could just compare the average of our observed data, \bar{x} against our estimate, μ_0 . But we don't know whether the observed points of data happened to tend higher or lower than the distribution's mean, μ , or by how much. Therefore, we use the variance in the observations to help measure how far the observed average might be from the actual average.

Figure 5.2: One-sided test



The image graph 5.2 is the estimate of the mean of a normal distribution. If your value is in the gray area, then it fails the null hypothesis.

Therefore, to compute a one-sided z-test when $\mu \leq \mu_0$ is the following code:

```
Normal norm(meanX, sX/sqrt(n));
bool nullHypothesis1 = (val >= norm.quantile(1 - alpha))\n;
```

```
bool nullHypothesis2 = (val <= norm.quantile(alpha));
```

In the above code, `meanX` is the observed mean μ , `sX` is the observed standard deviation, `n` is the number of data points, `val` is the value being tested μ_0 , and `alpha` is the certainty of the null hypothesis, usually 0.95. You should use only one of the two `nullHypothesis` results, depending on which way you use the null hypothesis.

The estimate for the mean of a normal distribution acts like a normal distribution with mean at \bar{x} and standard deviation $\frac{s_x}{\sqrt{n}}$.

Example 5.3 Estimating an election

You work for an advocacy group, and a ballot initiative that you sponsored is coming up. You run a poll of 100 randomly-selected voters; 54 of the voters support the issue. If the poll accurately represents the mood of the voters, are you more than 95% certain that the ballot would pass?

Assume that each vote is a Bernoulli test. Then the sum of the votes would be a binomial distribution. This distribution tells you how to compute the standard deviation.

The estimate of the mean would be a normal distribution with $\mu_e = 0.54$ and $\sigma_e = \frac{0.54 \times 0.46}{\sqrt{100}} = 0.02484$. The following code gives the answer:

```
Normal norm(0.54, 0.02484);
bool nullHypothesis = (0.5 >= norm.quantile(
    0.05));
```

`norm.cumulative(0.05)` is about 0.499; therefore, we are not yet 95% certain that the ballot will pass.

Instead of asking whether a value fails or does not fail the null hypothesis, we often want to know the exact p-value for the z-test. That is equally easy to compute:

```
Normal norm(meanX, sX/sqrt(n));
double pValue1 = norm.cumulative(val);
double pValue2 = 1 - norm.cumulative(val);
```

Again, you should use only one of the `pValue` results.

Example 5.4 Data and a z-test

Scores on most IQ tests are designed so that their mean is 100 and their standard deviation is 15. Therefore, the chance that someone has an IQ above 120 can be computed with:

```
Normal n(100, 15);
bright = 1 - n.cumulative(120);
```

5.3.3 Two-sided z-tests

Take the same setup as the previous section.

A **two-sided z-test** asks: Is it possible that μ_0 might be the actual mean?

two-sided z-test

This is a different sense of equal. The chance that the mean is any particular value is infinitesimal. However, a two-sided z-test measures whether any particular mean can be ruled out.

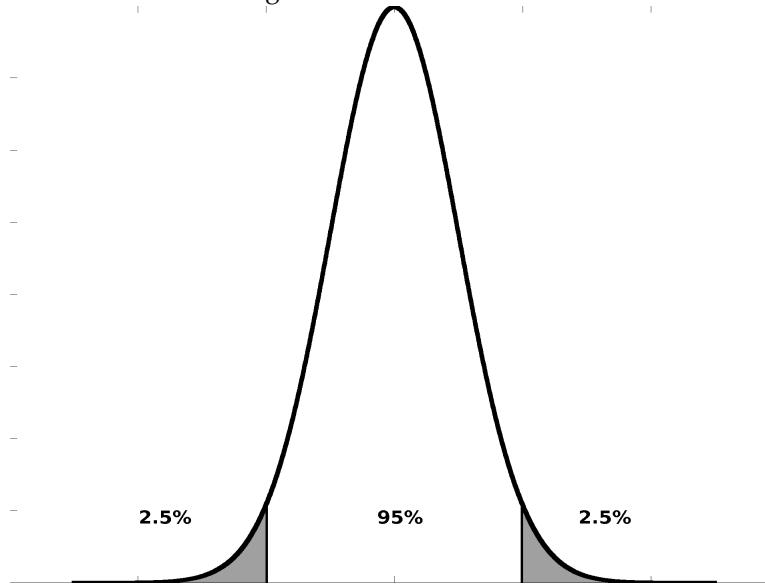
Another way to think about the two-sided z-test is to take the union of the two one-sided z-tests. If $H_0 : \mu \leq \mu_0$ and $\mu \geq \mu_0$, then $H_0 : \mu = \mu_0$.

Let μ_0 be the mean that you're testing against and μ be the (unknown) true mean of the distribution. Then a two-sided test would be:

$$\begin{aligned} H_0 &: \mu = \mu_0 \\ H_A &: \mu \neq \mu_0 \end{aligned}$$

This test asks whether we can be certain that the given mean is not the actual mean.

Figure 5.3: Two-sided test



The image graph 5.3 is the estimate of the mean of a normal distribution. If your value is in the gray area, then it fails the null hypothesis.

Notice that the area that fails the null hypothesis has been split in half. Half of the uncertainty goes to testing whether the mean is definitely below the known mean, and the other half of the uncertainty goes to testing whether the mean is definitely above the known mean.

To compute a two-sided z-test is almost as easy as a one-sided z-test:

```
Normal norm(meanX, sx/sqrt(n));
bool nullHypothesis = (val >= norm.quantile(0.5 - alpha/2))
&& (val <= norm.quantile(0.5 + alpha/2));
```

You should use two-sided tests instead of one-sided tests when you're measuring an effect that may cause a measurement to move in either direction.

Example 5.5 Two-sided versus one-sided, example 1

A doctor wants to know whether a new drug reduces the amount of ear wax produced by patients. She divides her subjects into two groups: a control group who gets a placebo, and an experimental group who gets the drug.

Even though she's only interested in medicines that reduce ear wax, she should use a two-sided test to compare the control and experimental groups. It's possible that her medication causes extra ear wax!

A two-sided test may fail to reject a test that a one-sided test of the same alpha does reject.

Example 5.6 Two-sided z-test

A scientist wants to learn whether chanting Homeric hymns at the television causes Ellen DeGeneres to blink more often. He sets up an experiment: For each show, he randomly chooses half of her show. He chants the thirty-three ancient Greek hymns during that random half, and refrains from praising those gods during the other half. After the show is over, he reviews the recordings to count how many times she blinked in each half.

After 60 experiments, the scientist takes the difference between the number of times she blinked with the Homeric hymns against the number of times she blinked without chanting. On average, she blinked 6 times more with the Homeric hymns than without; the standard deviation was 35. Is the scientist 95% sure that his Homeric hymns affected Ellen?

In this case, the “mean” is that of the null hypothesis, zero.

Try the following lines:

```
Normal n(0.0, 35.0/sqrt(60));
bool nullHypothesis = (6 >= n.quantile(0.025))
&& (6 <= n.quantile(0.975));
```

In this case, the null hypothesis is true; it might be random luck that Ellen blinks more often.

Instead of whether the null hypothesis fails at a given alpha, we often want to know what alpha we would need to fail the null hypothesis. This can be computed with:

```
Normal norm(meanX, sX/sqrt(n));
alpha1 = 1.0 - 2*norm.cumulative(value);
alpha2 = 2*norm.cumulative(value) - 1.0;
alpha = alpha1 > alpha2 ? alpha1 : alpha2;
```

PROBLEMS: (Answers on page 297)

1. (Easy) You have 100 samples from a normal distribution. Their mean is 98.0, and the known standard deviation for the data is 10.0. What is the chance that the mean for the distribution as a whole is 100.0 or greater?
2. (Easy) We have a distribution with standard deviation 10 units. We need to estimate the mean, with 95% certainty, to within 1 units. How many samples from the random distribution do we need to get this accurate of an estimate?
3. (Medium) Write a subclass of `ContinuousDistribution` that accepts a `DataDistribution` in its constructor. This distribution holds the z-test for the mean of that set.
4. (Medium) Write code that generates 100 samples from a normal distribution. Then run those samples through two different estimates of the mean:
 - A bootstrap test
 - A Z-test

Which estimate is more accurate (i.e.: which estimate actually contains the mean in a 95% interval more frequently?)

5. (Medium) A two-sided Z-test has the null hypothesis that the mean is zero. Sometimes, you want the opposite: you want a test with the null hypothesis of the mean is not-zero and the alternate hypothesis of the mean is zero. How would you construct this test?

5.3.4 Student's t-test

In practice, we rarely know the standard deviation for our source. Further, many times we cannot get 30 samples from a distribution.

Around the turn of the last century, William S. Gosset, a statistician for the Guinness brewery, noticed that substituting s (the standard deviation of the sample) for σ (the standard deviation for the whole distribution) did not quite work: he rejected the null hypothesis incorrectly more than his theory told him that he should. In 1908, he published *The Probable Error of a Mean* under his pseudonym, Student. This paper founded the theory of statistics for small sample sizes.²

Student's t-distribution degrees of freedom

The center of this theory is a new distribution, the **Student's t-distribution**. This distribution depends on the number of **degrees of freedom** in the data. For this distribution, if there are n pieces of data, then there are $\mu = n - 1$ degrees of freedom in the data.

Notice that sidebar 5.1 gives a distribution that has its mean at zero and variance that depends on the degrees of freedom. In order to get a t-distribution with mean m and variance a , use $\frac{a(\nu-2)}{\nu} f(x) + m$.

The t-test gives better estimates than the z-test when the sample is from a normal distribution and has fewer than 30 pieces of data.

You can have both one-sided and two-sided t-tests. They use the same code as one-sided and two-sided z-tests, except that they use the t-distribution.

Example 5.7 t-test

You help create BELM³ robotic pupkitbots, the toys with the slogan "So Cute, You'll Puke!" They look like they are spliced from one-fourth puppy, one-fourth kitten, one-fourth rabbit, and the rest is a combination of duckling, mongoose, wombat, rhinoceros,

²Information about William S. Gosset's life comes from http://www.swlearning.com/quant/kohler/stat/biographical_sketches/bio12.1.html, <http://www-history.mcs.st-andrews.ac.uk/Biographies/Gosset.html>, and [Ott and Longnecker, 2001, page 229]

³Your boss assures you that it stands for "Big Eyes, Little Mouth" rather than "Bug-Eyed, Loath-some Monsters"

Parameters	$\nu > 0$ is the number of degrees of freedom
Support	x is the position
Density	$f(x; \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})} - \left(1 + \frac{x^2}{\nu}\right)^{\frac{\nu+1}{2}}$
Mean	0
Variance	$\frac{\nu}{\nu-2}$ if $\nu > 2$

Sidebar 5.1: Student's t-distribution

mosquito, rabid vampire bat, coral snake, and tuba. They are guaranteed to have an average CR⁴ over 1000. As the Quality Assurance Manager, you must be at least 99% certain that a crate of newly manufactured pupkitbots meets this guarantee before they're shipped out. Naturally, the CR test destroys the pupkitbot, as you need to be sure that the pupkitbot is just as cute on the inside as it is on the outside. Therefore, you are limited to testing just five pupkitbots per batch.

In the most recent batch, five pupkitbots measured 990, 1000, 1005, 1020, and 1030 CR. Would you be 99% certain that the batch as a whole had an average CR above 1000?

Since we are only concerned with whether a measurement is above a given threshold, we can use a one-sided t-test.

```
int main(int argc, char* argv[]) {
    std::multiset<double> multisetCR;

    multisetCR.insert(990.0);
    multisetCR.insert(1000.0);
    multisetCR.insert(1005.0);
    multisetCR.insert(1020.0);
    multisetCR.insert(1030.0);

    Data<double> dataCR(multisetCR);
    StudentT tdist(dataCR.mean(),
                    dataCR.standardDeviation() / \
                    sqrt(dataCR.size()),
                    dataCR.size() - 1);
    bool nullHypothesis = (1000 >= tdist.quantile(
        0.01));

    if (nullHypothesis) {
        std::cout << "The null hypothesis is not \
disproven; we don't know for certain \
that the pupkitbots are cute enough.\n";
    } else {
        std::cout << "The null hypothesis is \
disproven; the pupkitbots are definitely \
not cute enough!\n";
    }
}
```

The null hypothesis is not disproved; we are not certain that this batch of pupkitbots are cute enough.

⁴Cuteness Rating

PROBLEMS: (Answers on page 299)

1. (Easy) Write the class for the Student's T-Distribution.
2. (Medium) Write a subclass of `ContinuousDistribution` that takes a set of doubles in its constructor, and outputs the t-test distribution of the estimate of its mean.
3. (Easy) Using the T-Test, what is the probability that the pupkitbots have a cuteness rating above 1000?
4. (Medium) Write code that generates n samples from a normal distribution. Then run those samples through two different estimates of the mean:
 - A Z-test
 - A t-test

For various n , which estimate is more accurate (i.e.: which estimate actually contains the mean more frequently?)

5.3.5 Wilcoxon Signed-Rank Test

The z-test and the t-test both assume that the data comes from a normal distribution. This assumption isn't always true.

Wilcoxon signed-rank test

The **Wilcoxon signed-rank test** is nonparametric: It does not assume that its data is normal. However, if the data were actually normal, tests won't be as accurate.

The setup for the Wilcoxon signed-rank test is same as the above tests: you have data $x_0 \dots x_n$, you have a value μ_0 that you're testing against the mean. The general algorithm for the test follows:

1. Create the set of y_i where $y_i = x_i - \mu_0$.
2. Remove all y_i that are zero.
3. Sort the set of $|y_i|$, and determine the ranks for each $|y_i|$.
4. Let T_+ be the sum of ranks for the positive y_i .
5. Let T_- be the sum of ranks for the negative y_i .
6. Let T_0 be the smaller of T_+ and T_- .

Algorithm 5.1: Wilcoxon Signed-Rank Test

If you're interested in one-sided tests, use T_+ or T_- . If you're interested in two-sided tests, use T_0 .

There are two ways to use T : compare it directly against known values, or use it like a normal distribution.

Comparing T against values

Most books and websites provide tables for the values of T for the Wilcoxon test. This one doesn't.

There are two ways to compute the values of T for the Wilcoxon test: an exact method and an estimating method. This book covers the estimating method: it is accurate, and it is much faster than the approximate method for larger sets of data.

The following source code finds the Wilcoxon value for any p :

```
// Create a random bit string of a given size.
std::vector<bool> randomSample(int sizeSample)
{
    int i;
    std::vector<bool> sampleRandom(sizeSample);

    for (i=0; i<sizeSample; i++) {
        sampleRandom[i] = (drand48() < 0.5);
    }

    return sampleRandom;
}

// Compute the Wilcoxon value of a sample.
int wilcoxon(std::vector<bool> theSample)
{
    unsigned int numBit;
    int TPos = 0;

    for (numBit = 0; numBit < theSample.size(); numBit++)
    {
        if (theSample[numBit])
            TPos += (numBit + 1);
    }
}

return TPos;
}

// Find the wilcoxon quantile.
int wilcoxonQuantile(int sizeSample, double dFraction,
                     int numRounds)
{
    int maxWilcox = sizeSample * (sizeSample + 1) / 2;
    std::vector<int> countWilcox(maxWilcox + 1);
    int fractionRounds;
    int round;
```

```

std::vector<bool> theSample;
int totalRounds;
int t;

// Initialize the Wilcox count.
for (round = 0; round < numRounds; round++) {
    theSample = randomSample(sizeSample);
    t = wilcoxon(theSample);
    countWilcox[t]++;
}

// Find the Wilcox value for dFraction.
fractionRounds = (int) (numRounds * dFraction);
totalRounds = 0;
for (t = 0; t < maxWilcox; t++) {
    totalRounds += countWilcox[t];
    if (totalRounds > fractionRounds) {
        return t;
    }
}

return maxWilcox;
}

```

Using T as a normal distribution

When there is enough pieces of data, the Wilcoxon T acts like a normal distribution. Let n be the number of pieces of data. Then the Wilcoxon T has mean $\frac{n(n+1)}{4}$ and standard deviation $\sqrt{\frac{n(n+1)(2n+1)}{24}}$.

How many pieces of data is enough? [Ott and Longnecker, 2001, page 309] recommends $n > 50$. <http://faculty.vassar.edu/lowry/ch12a.html> recommends $n > 10$. Those qualify as two extremes.

5.4 Estimating the variance of a normal distribution

In the previous section, we were concerned about finding a distribution for the mean of a normal distribution. In this section, we are concerned with finding a distribution for the variance (or the standard deviation).

5.4.1 Normal means and variance

The normal distribution is unusual: its mean is independent of its variance.

The four charts in figure 5.4 on page 181 were created with the following pseudo-code, substituting for the function:

```

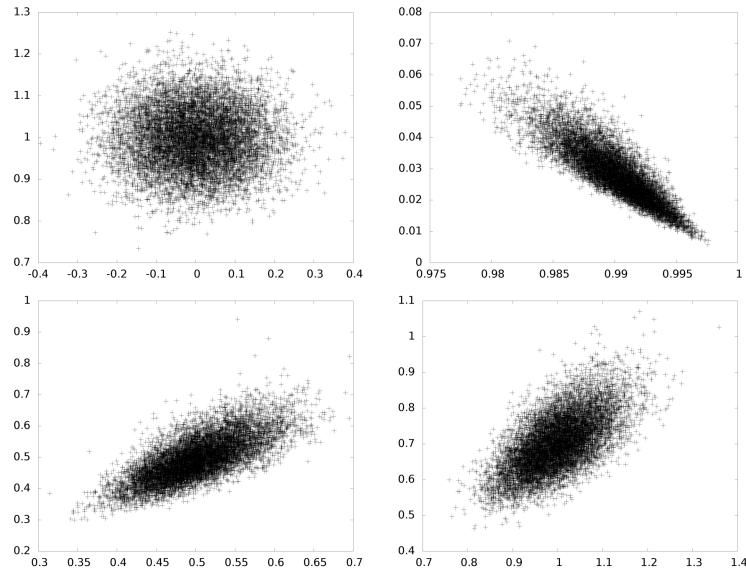
for (i=0; i<10000; i++) {
    for (j=0; j<100; j++) {
        array[j] = function();
    }

    mean[i] = array.mean();
    standardDeviation[i] = std::sqrt(array.variance());
}

```

They are the same kind of charts as seen around page 67.

Figure 5.4: The observed mean graphed against the observed standard deviation for normal, Beta, Exponential, and Gamma distributions



Then mean was graphed against standardDeviation. The four functions were

- Normal(0, 1) (upper left corner)
- Beta(10, 0.1) (upper right corner)
- Exponential(2) (lower left corner)
- Gamma(2, 2) (lower right corner)

The important factor of figure 5.4 is that, for most distributions, the observed variance depends on the observed mean. The only distribution with independent means and variances is the normal distribution.

5.4.2 Fast estimates for the standard deviation of a normal distribution

You will sometimes want a fast, easy way to estimate the standard deviation of a normal distribution. [Snedecor and Cochran, 1989, page 137] suggests that a fast estimate comes by multiplying the range⁵ by a constant.

Very few books contain this table⁶, but it can be found by experimentation.

PROBLEMS: (Answers on page 303)

1. (Medium) Write code to create a table of the 5% and 95% percentile of the ratio between the range of data and the standard deviation, for a normal distribution with between 2 and 30 elements.

5.4.3 Full-data estimates for the variance

Assume that your data comes from a normal distribution. What is the best estimate the variance of the data?

This section extensively uses the chi-squared, as defined in section 3.6.5 on page 143.

A bit of notation: I know of no good, mathematical way to express the quantile of the chi-squared notation. Most books write it as χ_α^2 , where α is the quantile. However, this notation expresses no relationship between the quantile and the original distribution.

For the sake of this section, I will write the density function as $\chi^2(\nu)$, the cumulative function as $\alpha = X^2(\nu)$, and the quantile function as $\nu = X^{-2}(\alpha)$. This notation is not even close to standard.

Let n be the number of elements in the sample, and S^2 be the estimate for the variance generated by the sample. What distribution describes the variance of our target distribution?

Other books (like [Wackerly et al., 2002, page 408]) will give a $100(1 - \alpha)\%$ confidence interval for σ^2 as $\left(\frac{(n-1)S^2}{X^{-2}(\alpha/2)}, \frac{(n-1)S^2}{X^{-2}(1-(\alpha/2))}\right)$.

How do we make this idea into a distribution? The function $\frac{(n-1)S^2}{X^{-2}(\alpha)}$ is a quantile function: given an α , it computes the x^2 such that α proportion of the random numbers of the distribution are (on average) below x^2 .

To compute the cumulative function, solve for α :

$$\begin{aligned} x^2 &= \frac{(n-1)S^2}{X^{-2}(\alpha)} \\ X^{-2}(\alpha) &= \frac{(n-1)S^2}{x^2} \\ \alpha &= X^2 \left(\frac{(n-1)S^2}{x^2} \right) \end{aligned}$$

⁵Defined on page 32

⁶Not even this one!

To compute the density function, take the derivative. Let $u = \frac{(n-1)S^2}{x^2}$. Then:

$$\begin{aligned} y &= \frac{d}{dx} X^2 \left(\frac{(n-1)S^2}{x^2} \right) \\ &= \frac{d}{dx} (X^2(u)) \\ &= \chi^2(u) \frac{du}{dx} \\ &= \chi^2 \left(\frac{(n-1)S^2}{x^2} \right) \frac{(n-1)S^2}{x^3} (-2) \end{aligned}$$

The sum for the density function over all possible inputs must be 1.0; this only happens when the given density function is multiplied by -1. The final density function for the estimate of the variance is:

$$2 \left(\frac{(n-1)S^2}{x^3} \right) \chi^2 \left(\frac{(n-1)S^2}{x^2} \right)$$

PROBLEMS: (Answers on page 305)

1. (Easy) Create a chart like figure 5.4 on page 181 with uniform numbers.
2. (Medium) Write a subclass of `ContinuousDistribution` that takes a list of doubles from a normal distribution in its constructor, and handles the variance as a distribution.

5.5 Estimating parameters for other distributions

5.5.1 Verifying the distribution

Often, you want to verify whether data matches a distribution. There is a graphical check to determine how well data matches a distribution.

If there are n points of data, a_1, a_2, \dots, a_n , and the distribution is d , then sort the data into their order statistics⁷, $a_{(1)}, a_{(2)}, \dots, a_{(n)}$. Graph these sorted values against `d.quantile(i / (n+1))`.

If the data is very linear, then it's likely to be the correct distribution. If the data looks very non-linear, then it's not likely to be the correct distribution.

⁷See page 33

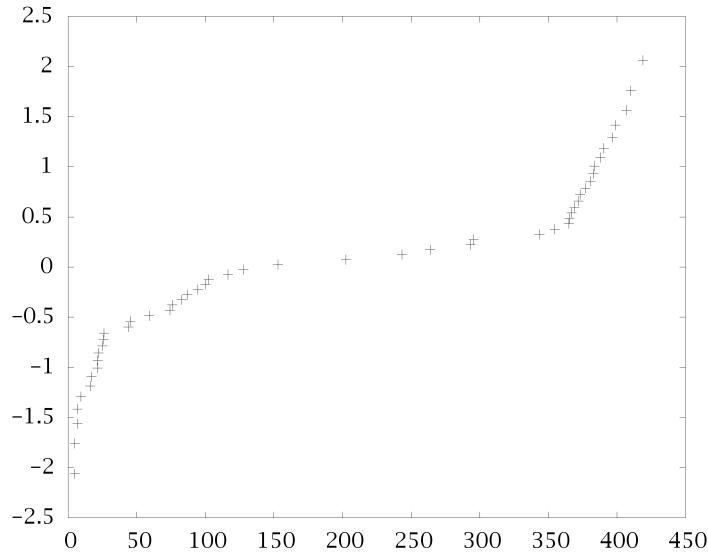
Example 5.8 Verifying the distribution

Imagine that you have the response times for fifty calls to your web service:

4.855496	4.970593	6.690846	6.848987	9.22763
16.04260	17.15662	21.29605	21.54219	22.08481
24.92836	25.63826	26.10288	43.81787	45.34917
59.11008	74.15157	76.01053	82.87135	86.86965
94.47901	100.2002	102.6042	116.6128	127.7183
152.9715	202.4939	243.4196	263.9509	293.3521
295.6814	343.5438	354.7969	365.0621	365.3715
366.8702	369.0556	372.0163	373.1268	377.1924
380.5952	382.8161	383.7155	387.7767	390.4872
396.6826	398.8868	406.7078	410.0899	419.0317

You wonder whether this web server might be normal. You compare this data against the $\text{Normal}(0, 1)$ distribution, and get graph 5.5.

Figure 5.5: Comparing observed response times against the normal distribution



Notice that this graph is not very linear. Therefore, it's unlikely that the underlying distribution would be normal.

PROBLEMS: (Answers on page 307)

1. (Easy) Create three sets of one hundred random numbers from the following distributions: Normal(0, 1), Normal(10, 5), Exponential(0.2). Graph each set against the quantiles of Normal(0, 1), Normal(10, 5), and Exponential(0.2). (You will have nine graphs.)

5.5.2 Pearson's χ^2 -test (goodness of fit)

Pearson's χ^2 test (chi-squared test) is a way to determine how well a distribution matches its expected values.

This test is useful when you want to determine whether:

- A die is unbiased.
- Students' heights are a normal distribution.
- The number of typos per page is a Poisson distribution.

The χ^2 test compares an observed histogram against an expected histogram. You need three things for the χ^2 test:

- Observed data
- A distribution with all parameters filled.

Divide your observed data into intervals so that no interval has fewer than 5 points of data. Call these intervals $I_1 \dots I_n$ – there are n of them.

Let $O(I_k)$ be the number of points of data observed in interval I_k . Let $E(I_k)$ be the number of points of data expected in interval I_k for the same number of pieces of data. Then the test value is

$$x = \sum_{i=1}^{i=n} \frac{(O(I_i) - E(I_i))^2}{E(I_i)}$$

What to compare the value of x against depends on how the distribution was created. Some times, you may have chosen the distribution without examining the data. Other times, you used the data to choose the distribution. For example, you knew that the distribution should be normal, but you did not know its mean and standard deviation. You estimated its mean and standard deviation from the data. In this case, two parameters came from the data.

If the distribution's parameters did not come at all from the data, then compare x value against the χ^2 distribution⁸ with $n - 1$ degrees of freedom. If p parameters came from the data, then compare x against the χ^2 distribution with $n - 1 - p$ degrees of freedom.

⁸See section 3.6.5 on page 143.

Example 5.9 A distribution of words in English

In every language, short words tend to be more common than long words. We expect that there are more four-letter words than six-letter words, for example. One distribution that describes this well is the exponential distribution. Do English words, after a minimum length, follow the exponential distribution?

I chose *Alice's Adventures in Wonderland* by Lewis Carroll⁹ and *Adventures of Huckleberry Finn* by Mark Twain¹⁰ as example texts in English. A full investigation should use more texts from more authors. I counted the number of letters in words with the following Perl code:

```
while (<>) {
    foreach $word (split(" ")) {
        $word =~ s/\W//;
        $len = length($word);
        if ($len >= 3 && $len <= 18)
            $count[$len]++;
    }
}
```

From the data, it's possible to estimate the parameter for the exponential distribution should be 4.55. I got the following tally:

letters	count	expected	$\frac{(O(I_i) - E(I_i))^2}{E(I_i)}$
3	36599	12019.93	50260.75
4	28739	9649.32	37777.84
5	16603	7746.25	10126.47
6	9632	6218.50	1873.91
7	6845	4992.07	687.76
8	3139	4007.52	188.22
9	1918	3217.14	524.61
10	994	2582.64	977.21
11	584	2073.29	1069.79
12	364	1664.39	1016.00
13	162	1336.13	1031.77
14	79	1072.61	920.43
15	50	861.07	763.97
16	30	691.25	632.55
17	16	554.92	523.38
18	8	445.47	429.61

The sum of the fourth column is 108804.27. We have 14 degrees of freedom: 16 intervals - 1 (from the formula) - 1 (parameter for

⁹<http://www.gutenberg.org/files/11/11.txt>

¹⁰<http://www.gutenberg.org/files/76/76.txt>

the exponential distribution). For 14 degrees of freedom, anything above 23.68 is above the 95% certainty level. Therefore, we can say with near-certainty that the distribution of English words does not follow the exponential distribution.

Chapter 6

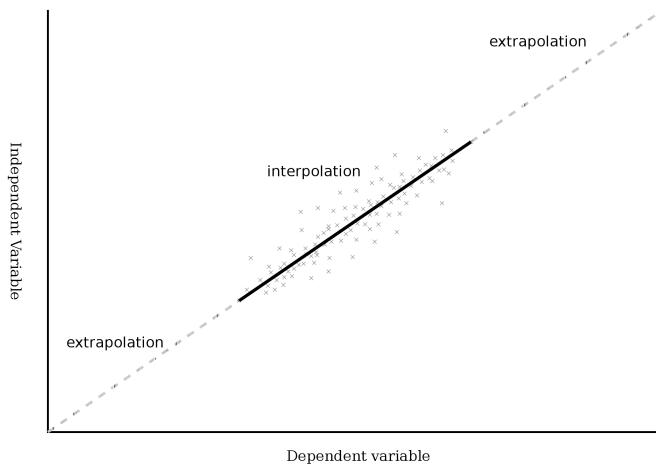
Correlation and Interpolation

Data often shows correlations: as one variable changes, the other changes with it. Creating a formula that fits the data, within the limits of the data, is called **interpolation**.

Interpolation is often compared against **extrapolation**. Interpolation occurs when someone finds values within the limits of the data. Extrapolation occurs when someone finds values outside the limits of the data. Extrapolation is always less certain than interpolation.

interpolation
extrapolation

Figure 6.1: The difference between interpolation and extrapolation



Example 6.1 Extrapolation: example 1

When Joni eats three pieces of pizza in one sitting she has 100% satisfaction. Forcing her to eat thirty pieces in one sitting does not cause her satisfaction to be 1000%.

Example 6.2 Extrapolation: example 2

A (not very good) scientist investigates the properties of a liquid at temperatures from -20°C to 10°C , then extrapolates the expected behavior at -40°C and 40°C . Later, saner scientists point out that the liquid freezes at -30°C and boils at 30°C .

Example 6.3 Extrapolation: example 3

A proud mother, Emma, hugs her new baby daughter. Only a bad statistician would say that, given that yesterday Emma had 0 daughters and today she has 1 daughter, therefore she should expect another daughter every day for the rest of eternity.

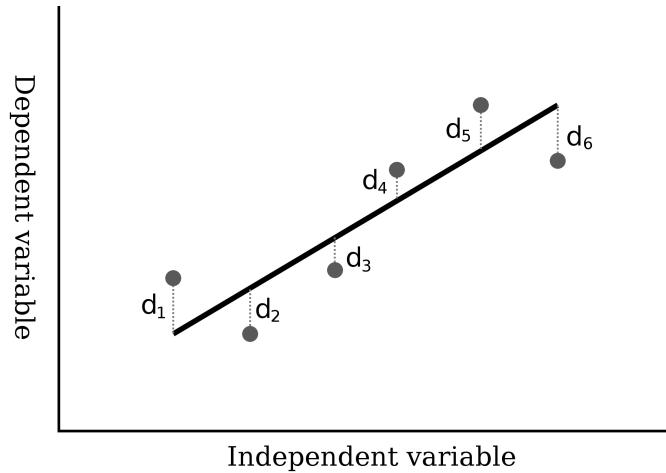
The examples of extrapolation should seem silly. But they're just as silly as saying, without supporting information, that a company's growth will continue at the same rate for the foreseeable future.

There are two kinds of variables: independent and dependent variables. Independent variables can change on their own. When independent variables change, changes to dependent variables are observed. Interpolation pretends that independent variables causes changes to dependent variables.

We often need to express the difference between observed values and predicted values. This book uses a "hat" to represent predicted, rather than observed, values. For example, if $y = f(x)$ represents the actual (unknown) relationship between x and y , then $\hat{y} = \hat{f}(x)$ is the predicted value of y at point x .

PROBLEMS: (Answers on page 308)

1. (Easy) This book talks about pretending that independent variables cause changes to dependent variables. When would interpolation be useful, even if independent variables did not cause changes to the dependent variables?

Figure 6.2: The best-fit measurement is $\sum_{i=1}^6 d_i^2$ 

6.1 Linear Regression

The simplest question for interpolation is: What is the best linear (straight line) relationship between two variables?

Look back at the definition of “least square” in section 4.4 on page 159.

Let x and y be vectors, both with n elements. In the equation $y = a + bx$, what are the least square best fits for a and b ?

This problem needs to minimize:

$$\begin{aligned} m &= \sum_{i=1}^n (f(x_i) - y_i)^2 \\ &= \sum_{i=1}^n (a + bx_i - y_i)^2 \end{aligned}$$

We want to minimize m with regard to a and b . Therefore, take two partial derivatives of m with regard to a and b , and compute.

$$\begin{aligned}
0 &= \frac{\partial m}{\partial a} \\
0 &= \sum_{i=1}^n (2(a + bx_i - y_i)(-1)) \\
0 &= -2 \sum_{i=1}^n (a + bx_i - y_i) \\
0 &= na + b \sum_{i=1}^n x_i - \sum_{i=1}^n y_i \\
a &= \frac{\sum_{i=1}^n y_i - b \sum_{i=1}^n x_i}{n} \\
a &= \bar{y} - b\bar{x}
\end{aligned}$$

$$\begin{aligned}
0 &= \frac{\partial m}{\partial b} \\
0 &= \sum_{i=1}^n (2(a + bx_i - y_i)(x_i)) \\
0 &= 2 \sum_{i=1}^n (ax_i + bx_i^2 - x_i y_i) \\
0 &= a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 - \sum_{i=1}^n x_i y_i \\
0 &= (\bar{y} - b\bar{x}) \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 - \sum_{i=1}^n x_i y_i \\
0 &= \bar{y} \sum_{i=1}^n x_i - b\bar{x} \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 - \sum_{i=1}^n x_i y_i \\
0 &= n\bar{x}\bar{y} - nb\bar{x}^2 + b \sum_{i=1}^n x_i^2 - \sum_{i=1}^n x_i y_i \\
0 &= b \left(\sum_{i=1}^n x_i^2 - n\bar{x}^2 \right) + n\bar{x}\bar{y} - \sum_{i=1}^n x_i y_i \\
b &= \frac{\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^n x_i^2 - n\bar{x}^2}
\end{aligned}$$

**** insert example here ****

PROBLEMS: (Answers on page 308)

1. (Easy) Write code that computes the optimal linear equation for pairs of data. The amount of memory used should be constant, no matter how much data is entered.

6.2 Distribution of linear regression

It's not enough to know the single best-fit measurement for data. It's also important to know the distribution of these estimates.

In probability, distributions yield two important questions:

1. What are the distributions for the slope and y-intercept of the optimal linear equation?
2. Given a new point, what is the likelihood that this point is part of the interpolation?

6.2.1 Distribution of the best-fit line

The first questions are about the distribution for the best-fit line for the data.

This is important for more than academic reasons. We are often curious whether there is a linear relationship between our variables. We have two hypotheses:

The null hypothesis: $b = 0$

The alternate hypothesis: $b \neq 0$

To perform this test, we need to have certain assumptions¹:

1. At any given value of x , the population of potential error term values has a mean equal to 0.
2. At any given value of x , the population of potential error term values has a variance that does not depend on x . That is, the different populations of potential error term values corresponding to different values of x have equal variances.
3. At any given value of x , the population of potential error term values has a normal distribution.
4. Any value of the error term is statistically independent of any other value of any other error term. That is, the value of the error term e_y corresponding to an observed value of y is statistically independent of the value of the error term corresponding to any other observed value of y .

Given that information, let's compute some values.

We define the **sum of squared residuals** as

sum of squared residuals

$$SSE = \sum (y_i - f(x_i))^2$$

This is another name for what we're trying to minimize with the linear square best fit.

The point estimate for the *****

6.2.2 Point likelihood

The second set of questions are about the likelihoods of points within the distribution. The question of points within the distribution is very different than the best-fit line: the variance for points is much bigger than the variation for the line.

6.3 Measuring the linear regression equation

The linear regression equation always finds a best-fit equation, no matter what data is given to it. Measuring how well a linear equation matches the data gives an estimate of how good are its predictions.

Although a "best linear correlation" always exists, the linear correlation might not be useful. In graph 6.3 on page 195 the linear correlation only somewhat matches the data.

6.3.1 covariance

Page 55 defined the variance of X as $E[X - E[X]]^2$.

covariance The **covariance** of two variables X and Y is defined as

$$\text{Cov}(X, Y) = E[X - E[X]] E[Y - E[Y]]$$

It's a measurement of how much variables vary based on each other. You can easily see that the covariance of one variable against itself is the same as its variance.

PROBLEMS: (Answers on page 309)

1. (Medium) Prove that $\text{Cov}(X, Y)$ can also be written as $E[XY] - E[X]E[Y]$.

Figure 6.3: An example of bad linear correlation

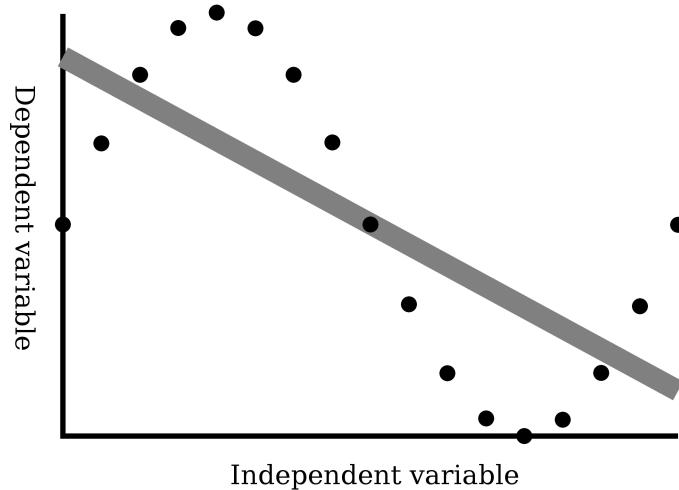


Figure 6.4: Best-fit correlation ignoring X axis

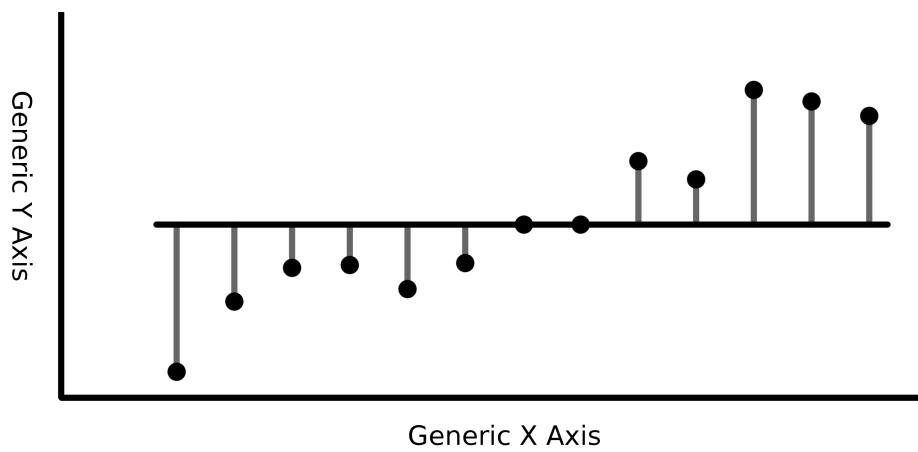
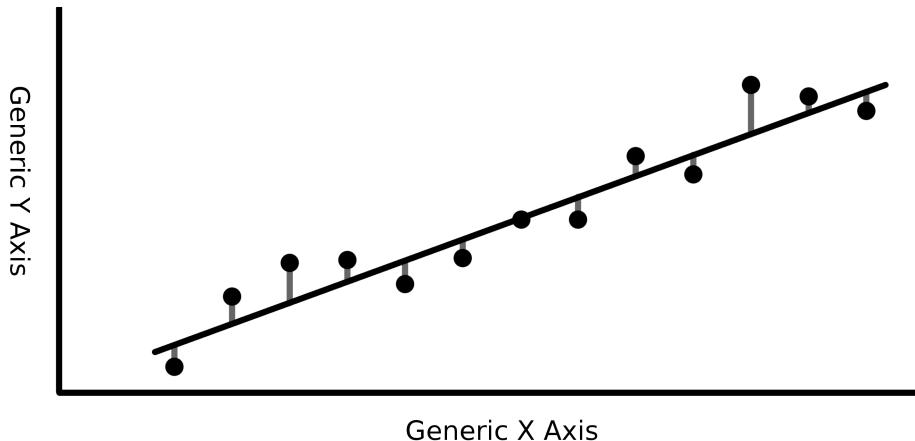


Figure 6.5: Best-fit correlation with X axis



6.3.2 The linear correlation coefficient

When we make a prediction, we try to limit unexplained variation. The linear correlation coefficient measures the ratio of explained variation against the variation without the explanation.

In graph 6.4 on page 195 are points, as well as the best-fit line of the points that makes no reference to the X values. The gray bars represent the variation without the explanation of the X axis.

In graph 6.5 are the same points, as well as the best-fit line using information from the X axis. The gray bars represent the variation that remains after the X axis has been considered.

The explained variation would be the difference between the first and the second figures. The linear correlation would be the ratio of the explained variation and the variation without the explanation of the X axis.

Let's compute the linear correlation directly.

x	y	$(y - \bar{y})^2$	$(y - f(x))^2$
1	2	16	0
2	5	1	1
3	6	0	0
4	7	1	1
5	10	16	0
sum		34	2

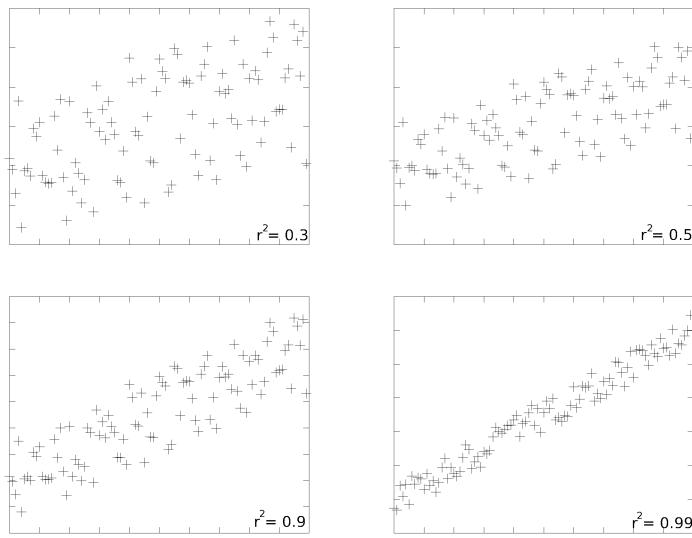
¹These assumptions are taken from [Bowerman et al., 2005, page 97]

The best fit for this data is $y = 2x$. When x can't be considered, the best fit for this data is $y = 6$.

The variance without using x as an explanation is 34. The variance using x is 2. Therefore, the function $y = 2x$ explains 32 units of variance. Therefore, the square of the linear correlation coefficient is $r^2 = \frac{32}{34}$.

In graph 6.6 are graphical examples of different r^2 values.

Figure 6.6: Examples of linear correlation, 2



The linear correlation coefficient is defined as:

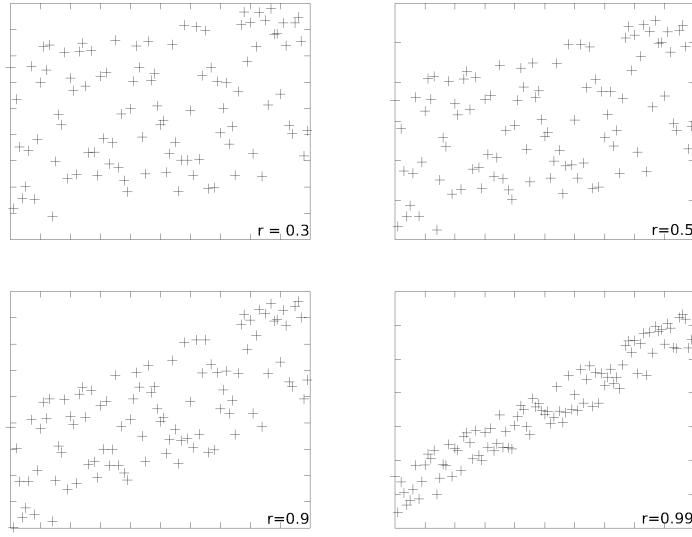
$$\begin{aligned} r &= \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y} \\ &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{j=1}^n (y_j - \bar{y})^2}} \end{aligned}$$

The linear correlation coefficient measures how linear is the relationship between X and Y . In graph 6.7 on page 198 are graphical examples of different r values.

The correlation coefficient is always between -1 and 1. Interpreting it takes some care:

- If $r > 0$, then a positive-slope line describes the data. The closer r is to 1, the more accurate this line is.

Figure 6.7: Examples of linear correlation, 1



- If $r < 0$, then a negative-slope line describes the data. The closer r is to -1, the more accurate this line is.
- If $r \approx 0$, then either of two things could be true:
 - The data has no correlation.
 - The data has a correlation, but not one that can be described by a linear relationship.

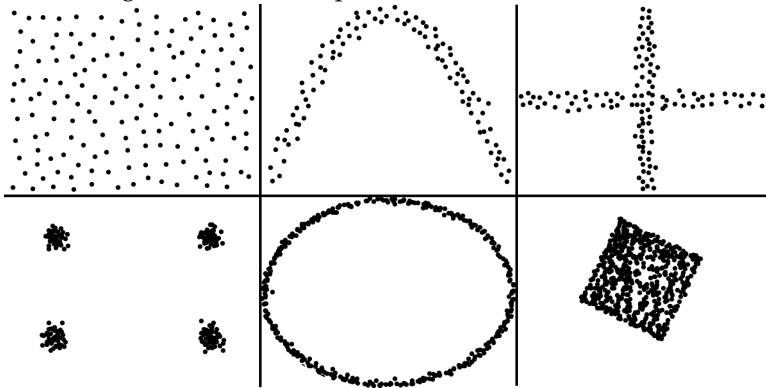
Remember that $r_{X,Y} = 0$ is not the same as saying there is no relationship between X and Y . In graph 6.8 on page 199 are six examples of groups with zero linear correlation; many are strongly correlated (just not under a linear correlation).

Note that the linear correlation coefficient is extremely sensitive to outliers.

PROBLEMS: (Answers on page 309)

1. (Easy) Expand the code that you wrote in “Linear Regression” to also compute the linear correlation coefficient.
2. (Medium) Prove that r is (almost) always in the range [-1, 1]. When is it outside that range?
3. (Medium) We have a definition for what the linear correlation coefficient is, and we have a mathematical definition. Prove that the two are the same.

Figure 6.8: Six examples of zero linear correlation



6.4 Abusing Regression and Correlation

6.4.1 Stock Market picks

I like to make money. Do you like to make money? Then we like to make money! Let's use our statistics knowledge to beat the stock market!

The variations in one stock often signal that other stocks will shift soon. For example, when oil prices go up, oil companies gain value, but shipping companies, car makers, fertilizer makers, and other companies that depend on oil lose value. Many other relationships are less obvious.

If we can find a relationship that tells us ahead of time when stocks will go up or down, we can make lots of money!²

Let's say that we have 30 days' worth of data on 50 stocks. All that we care about is whether the stock goes up or down on that date. Try correlating every stock against every other stock, shifted between one and seven days. Our stock market looks like table 6.1, where a + means a rise on that day, and a - means a fall on that day.

We use the following code to analyze the stock market:

```
// Do the analysis...
void Analysis::analyze_market() const
{
    int correlation;
    int daysAhead;
    int stockNumber1;
    int stockNumber2;
    std::vector<bool> stockValuesNow;
```

²The method given in this section only exists as an example of the misuse of statistics. Neither I nor my publisher take responsibility for any financial decisions made using this method. Unless, of course, you make money.

Stock name	Rise or fall
AELU	+++++-----++-+++-+--++-++-
BBNR	+++++-----++-+++-+--++-++-
BGYR	-----++-+++-+--++-++-
BNLF	+-----+----++-+++-+--++-
BQBC	-+---+----++-+++-+--++-
BQUP	-----+---++-+++-+--++-
CGIE	-----+---++-+++-+--++-
CNCA	+++++---++-+++-+--++-
CNYW	-----+---++-+++-+--++-
CQZQ	+++++---++-+++-+--++-
DBCB	+-----+---++-+++-+--++-
DCNT	-+---+---++-+++-+--++-
GGWS	-+---+---++-+++-+--++-
IPNS	-----+---++-+++-+--++-
IUUE	+-----+---++-+++-+--++-
KMZM	-+---+---++-+++-+--++-
KWCJ	+++---+---++-+++-+--++-
LGDK	+++++---+---++-+++-+--++-
MFAV	-+---+---++-+++-+--++-
MRYN	-+---+---++-+++-+--++-
MUDJ	+++---+---++-+++-+--++-
OCMC	+++---+---++-+++-+--++-
ONVA	-+---+---++-+++-+--++-
PDCW	-+---+---++-+++-+--++-
PDSC	-+---+---++-+++-+--++-
QDWA	+++++---+---++-+++-+--++-
QUIO	-+---+---++-+++-+--++-
QVLB	-+---+---++-+++-+--++-
RSNR	-+---+---++-+++-+--++-
RYHS	+---+---++-+++-+--++-
SHWW	+++++---++-+++-+--++-
SVBO	-+---+---++-+++-+--++-
TBVU	-+---+---++-+++-+--++-
TFIM	+++++---++-+++-+--++-
TRHN	+++---+---++-+++-+--++-
TRJV	-+---+---++-+++-+--++-
UEQF	+++---+---++-+++-+--++-
UQIO	+++++---+---++-+++-+--++-
VJTP	+---+---++-+++-+--++-
VLXA	-+---+---++-+++-+--++-
VUKV	+---+---++-+++-+--++-
VWWA	-+---+---++-+++-+--++-
WGMY	-+---+---++-+++-+--++-
YDWX	-+---+---++-+++-+--++-
YEHN	+---+---++-+++-+--++-
YEVA	-+---+---++-+++-+--++-
YHMY	-+---+---++-+++-+--++-
YOJO	+---+---++-+++-+--++-
ZEVW	-+---+---++-+++-+--++-
ZUJC	-+---+---++-+++-+--++-

Table 6.1: A very simplified stock market

where correlation returns the number of days that the current stock and the future stock either rose together or fell together.

To our happiness, the program returns plenty of stock tips:

Between YDWX and IPNS at a delay of 1 days, there is a correlation of 23
YDWX: ---+----+---+---+---+---+---+---+

```

IPNS: -----+-----+-----+-----+-----+
Between VWWA and LGDK at a delay of 2 days, there is a correlation of 23
VWWA: -+---+---+---+---+---+---+---+---+---+
LGDK: +---+---+---+---+---+---+---+---+---+
Between VWWA and IPNS at a delay of 2 days, there is a correlation of 23
VWWA: -+---+---+---+---+---+---+---+---+---+
IPNS: -----+-----+-----+-----+-----+-----+
Between QUIO and YEVA at a delay of 1 days, there is a correlation of 23
QUIO: -----+---+---+---+---+---+---+---+---+
YEVA: -----+---+---+---+---+---+---+---+---+
Between QUIO and YEVA at a delay of 2 days, there is a correlation of 24
QUIO: -----+---+---+---+---+---+---+---+---+
YEVA: -----+---+---+---+---+---+---+---+---+
Between QUIO and BQBC at a delay of 4 days, there is a correlation of 23
QUIO: -----+---+---+---+---+---+---+---+---+
BQBC: -+---+---+---+---+---+---+---+---+
Between BNLF and MRYN at a delay of 1 days, there is a correlation of 23
BNLF: +---+---+---+---+---+---+---+---+---+
MRYN: -+---+---+---+---+---+---+---+---+
Between RSNR and BGYR at a delay of 1 days, there is a correlation of 23
RSNR: -----+---+---+---+---+---+---+---+---+
BGYR: -----+---+---+---+---+---+---+---+---+

```

Let's examine one of these predictions: between QUIO and YEVA (delayed two days), there are 28 pieces of binary data to compare. Each piece has a fifty percent chance of matching. Referring back to the binomial distribution, section 3.5.2 on page 92, we know that the chance that 24 or more of these cases would come out the same is $\text{binom}(28, 0.5, 24) + \text{binom}(28, 0.5, 25) + \text{binom}(28, 0.5, 26) + \text{binom}(28, 0.5, 27) + \text{binom}(28, 0.5, 28) = 8.99956 * 10^{-5}$

Isn't that proof enough that, whenever QUIO goes up, we should buy YEVA, and whenever QUIO goes down, we should sell YEVA?

6.4.2 Problem with the stock market pick method

If you haven't guessed, all stocks in this market were created independently and randomly, and there was no real correlation between any two pairs of stocks.

What went wrong?

To find its suggestions, the program made $50 \times 49 \times 7 = 17150$ comparisons. If our level of significance (p-value) were 0.95, then we would expect 857.5 false positives for every true positive.

Chapter 7

Simple Classification

The questions in this chapter are of the form: Given texts t_1, t_2, \dots, t_n , how does a program classify these texts into m categories?

7.1 Supervised Learning

For supervised learning, you need two things:

1. The set of categories that texts will fit into.
2. An **oracle**, which can map from certain texts to their categories. **oracle**

7.1.1 Example

This chapter will use a long-running example: Is a given text written in English, Esperanto, or Estonian?¹

The training data used for these examples will be the preamble to the *United Nations Universal Declaration of Human Rights*. The English version appears on page 205; the Esperanto version appears on page 206; the Estonian version appears on page 207.

The first useful thing that you should notice² is that the three languages use slightly different alphabets. Going beyond the texts,

The English alphabet is: { A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z }.³

The Esperanto alphabet is: { A, B, C, Ĉ, D, E, F, G, Ĝ, H, I, J, Ĵ, K, L, M, N, O, P, R, S, Ĥ, T, U, Ě, V, Z }.⁴

¹Feel free to expand the example to languages that don't start with E.

²The first thing that most people will notice about Esperanto and Estonian is "I don't understand them!"

³<http://www.sesamestreet.org/parents/abc>

⁴<http://www.omniglot.com/writing/esperanto.htm>.

The purely-Estonian alphabet is: { A, B, D, E, G, H, I, J, K, L, M, N, O, P, R, S, Š, Z, Ž, T, U, V, Ö, Ä, Ö, Ü }⁵

We can break the letters into five categories:

English-only { Q, W, X, Y }

Esperanto-only { Ĉ, Ĝ, Ĵ, Ŝ, Ŭ }

Estonian-only { Ä, Ö, Ö, Š, Ü, Ž }

English and Esperanto only { C, F }

All languages { A, B, D, E, G, H, I, J, K, L, M, N, O, P, R, S, T, U, V, Z }

Recognizing languages is easy: a text with a 'Y' is almost definitely English, and a text with a 'Ĵ' is almost definitely Esperanto. For the purpose of making the problem more difficult, texts will go through the following transformation.

Original	Transformed	Original	Transformed
Ä	A	Š	S
C	K	Ŗ	S
Ĉ	K	Ü	U
F	V	Ü	U
Ĝ	G	W	U
Ĵ	J	X	Z
Ö	O	Y	I
Ö	O	Ž	Z
Q	K		

The only purpose of this transformation is to make the problem harder. In any real problem, you should happily accept anything that makes your solution easier.

⁵<http://www.estinst.ee/publications/language/alphabet.html>. Some other letters appear in foreign words.

Whereas recognition of the inherent dignity and of the equal and inalienable rights of all members of the human family is the foundation of freedom, justice and peace in the world,

Whereas disregard and contempt for human rights have resulted in barbarous acts which have outraged the conscience of mankind, and the advent of a world in which human beings shall enjoy freedom of speech and belief and freedom from fear and want has been proclaimed as the highest aspiration of the common people,

Whereas it is essential, if man is not to be compelled to have recourse, as a last resort, to rebellion against tyranny and oppression, that human rights should be protected by the rule of law,

Whereas it is essential to promote the development of friendly relations between nations,

Whereas the peoples of the United Nations have in the Charter reaffirmed their faith in fundamental human rights, in the dignity and worth of the human person and in the equal rights of men and women and have determined to promote social progress and better standards of life in larger freedom,

Whereas Member States have pledged themselves to achieve, in cooperation with the United Nations, the promotion of universal respect for and observance of human rights and fundamental freedoms,

Whereas a common understanding of these rights and freedoms is of the greatest importance for the full realization of this pledge,

<http://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=eng>

Sidebar 7.1: Preamble: English version

Pro tio, ke agnosko de la esenca digno kaj de la egalaj kaj nefordoneblaj rajtoj de ĉiuj membroj de la homara familio estas la fundamento de libero, justo kaj paco en la mondo,

Pro tio, ke malagnosko kaj malestimo de la homaj rajtoj rezultigis barbarajn agojn, kiuj forte ofendis la konsciencojn de la homaro, kaj ke la efektiviĝo de tia mondo, en kiu la homoj ĝuas liberecon de parolo kaj de kredo kaj liberiĝon el timo kaj bezono, estas proklamita kiel la plej alta aspiro de ordinara homoj.

Pro tio, ke nepre necesas, se la homoj ne estu devigitaj, sen alia elektebla vojo, ribeli kontraŭ tiranismo kaj subpremo, ke la homaj rajtoj estu protektataj de la lego,

Pro tio, ke nepre necesas evoluigi amikajn rilatojn inter la nacioj,

Pro tio, ke la popoloj de Unuiĝintaj Nacioj en la Ĉarto reasertis sian firman kredon je la fundamentaj homaj rajtoj, je la digno kaj valoro de la homa personeco kaj je la egalaj rajtoj de viroj kaj virinoj, kaj firme decidis antaŭenigi socian progreson kaj pli alnivelan vivon en pli granda libereco,

Pro tio, ke la Ŝtatoj-Membroj sin devigis atingi, en kunlaboro kun Unuiĝintaj Nacioj, la antaŭenigon de universala respekto al kaj observado de la homaj rajtoj kaj fundamentaj liberecoj,

Pro tio, ke komuna kompremo pri tiuj ĉi rajtoj kaj liberecoj estas esence grava por plena realigo de tiu sindavigo, _____

<http://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=1115>

Sidebar 7.2: Preamble: Esperanto

Pidades silmas, et inimkonna kõigi liikmete väärikuse, nende võrdsuse ning vőõrandamatute õiguste tunnustamine on vabaduse, õigluse ja üldise rahu alus; ja

pidades silmas, et inimõiguste põlastamine ja hülgamine on viinud barbaarsusteni, mis piinavad inimkonna südametunnistust, ja et sellise maailma loomine, kus inimestel on veendumuste ja sõnavabadus ning kus nad ei tarvitse tunda hirmu ega puudust, on inimeste üllaks püüdluseks kuulutatud; ja

pidades silmas vajadust, et inimõigusi kaitseks seaduse võim selleks, et inimene ei oleks sunnitud viimase abinõuna üles tõusma türannia ja rõhumise vastu; ja

pidades silmas, et on vaja kaasa aidata sõbralike suhete arendamisele rahvaste vahel ja;

pidades silmas, et ühinenud rahvaste perre kuuluvad rahavad on põhikirjas kinnitanud oma usku inimese põhiõigustesse, inimisiksuse väärikusse ja väärtsuse ning meeste ja naiste võrdõiguslikkusesse ning on otsustanud kaasa aidata sotsiaalsele progressile ja elutingimustele parandamisele suurema vabaduse juures; ja

pidades silmas, et liikmesriigid on kohustatud koosts Ühinenud Rahvaste Organisatsiooniga kaasa aitama inimõiguste ja põhivabaduste üldisele austamisele ja nendest kinnipidamisele; ja

pidades silmas, et üldisel arusaamisel nende õiguste ja vajaduste iseloomust on suur tähtsus selle kohustuste täielikuks täitmiseks,

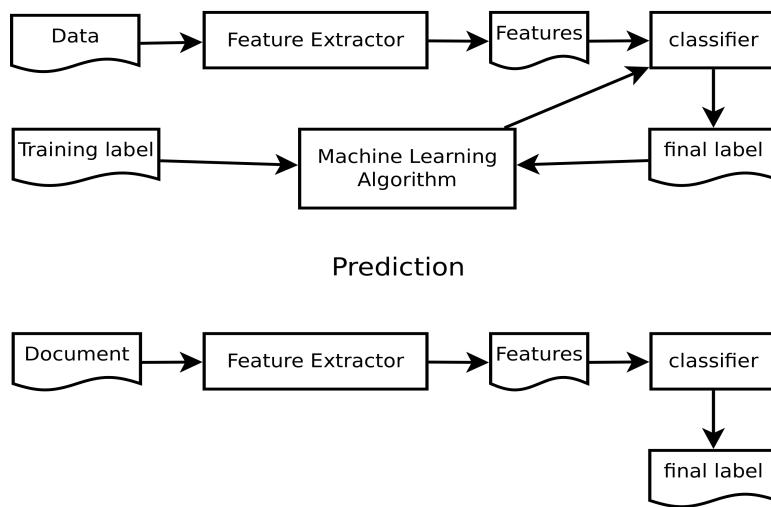
<http://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=est>

Sidebar 7.3: Preamble: Estonian

7.1.2 Algorithm

All cases of supervised learning have the same general algorithm:

Figure 7.1: Supervised Learning Algorithm
Training



Training

1. Given a text, the feature extractor extracts features from it.
2. Given the features, the classifier chooses a label.
3. Given the trained label and the chosen label, the machine learning algorithm adjusts the classifier.

Supervised learning has two methods: training and prediction.

Prediction

1. Given a text, the feature extractor extracts features from it.
2. Given the features, the classifier chooses a label.

Three sub-algorithms are involved in supervised learning:

Feature Extractor

From a text, this algorithm extracts its most important features.

For each document:

```
void Classification::train(const Text& text, const Label&
    & labelOracle)
{
    Label labelClassifier;
    Set<Feature> setFeat;

    setFeat = m_extractor.extract(text);
    labelClassifier = m_classifier.getLabel(setFeat);
    m_machine_learning.update(setFeat,
        labelOracle,
        labelClassifier,
        m_classifier);
}
```

Algorithm 7.1: Supervised learning, training

For each document:

```
Label Classification::predict(const Text& text)
{
    Label labelClassifier;
    Set<Feature> setFeat;

    setFeat = m_extractor.extract(text);
    labelClassifier = m_classifier.getLabel(setFeat);

    return labelClassifier;
}
```

Algorithm 7.2: Supervised learning, prediction

```
class Extractor
{
public:
    Set<Feature> extract (const Text &text) const;
};
```

Example 7.1 Examples of extractors

Examples of extractors are often obvious. They can be easy or very difficult to implement.

- For a text document, an extractor might extract the words used.
 - For a game of chess, an extractor might extract the locations of the pieces, and which pieces are attacking or being attacked.
 - For a program analyzer, an extractor might extract all conditionals and all loops.
 - For a music recommendation service, an extractor might extract the songs and number of times each song has been played.
-

In the language recognition example, an easy feature would map each character to the percentage of the text that it represents.

Classifier

From the set of features, this algorithm chooses the most likely label. The classifier can also be updated by the Machine Learning algorithm.

```
class Classifier
{
public:
    Label getLabel (const Set<Feature>& features) const;
    void update (const void *pUpdate);
}
```

The division between the classifier and the extractor is often chosen for the programmer's convenience. If an extractor could be made perfect, then the classifier could be eliminated. If a classifier could be made perfect, then the features would be the original data.

A good place to separate the extractor from the classifier is with what changes with learning. If it never changes, then it belongs to the extractor. If it becomes more accurate with data, then it belongs to the classifier.

Machine Learning

From the output of the classifier and the correct label, this algorithm updates the classifier.

```
class MachineLearning
{
public:
    void update(const Set<Feature>& features, const \
        Label& labelOracle,
        const Label& labelClassifier,
        /* not const */ Classifier& classifier);
};
```

The machine learning step only occurs during training. It requires knowing the correct classification for the text.

Naive Bayesian Classifier

The simplest classifier is called the Naive Bayesian classifier. It's based on Bayes' Law (defined on page 75).

Assume that the categories are C_1, C_2, \dots, C_m . Assume that the feature sets are F_1, F_2, \dots, F_a , and that for every feature F_b , there are choices, $F_{b,1}, F_{b,2}, \dots, F_{b,c}$. In other words, each text has been transformed to a series of feature choices $(F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a})$.

We want to compute $\Pr(C_i | F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a})$ for each of the C_i . By using Bayes' Formula, we compute:

$$\begin{aligned} \Pr(C_i | F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a}) &= \frac{\Pr(C_i) \Pr(F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a} | C_i)}{\Pr(F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a})} \\ &= \frac{\Pr(C_i) \Pr(F_{1,f_1} | C_i) \Pr(F_{2,f_2}, \dots, F_{a,f_a} | C_i, F_{1,f_1})}{\Pr(F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a})} \\ &\quad \dots \\ \frac{\Pr(C_i) \Pr(F_{1,f_1} | C_i) \Pr(F_{2,f_2} | C_i, F_{1,f_1}) \dots \Pr(F_{a,f_a} | C_i, F_{1,f_1}, F_{2,f_2}, \dots, F_{a-1,f_{a-1}})}{\Pr(F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a})} \end{aligned}$$

The Naive Bayesian Classifier assumes that each feature set is independent of every other feature set. (That assumption is obviously not always true.) Therefore, the long equation above can be simplified to:

$$\frac{\Pr(C_i) \Pr(F_{1,f_1}|C_i) \Pr(F_{2,f_2}|C_i) \dots \Pr(F_{a,f_a}|C_i)}{\Pr(F_{1,f_1}, F_{2,f_2}, \dots, F_{a,f_a})}$$

So, how does one compute $\Pr(F_{i,f_i}|C_i)$?

A first answer comes from the definition of conditional probability:

$$\Pr(F_{i,f_i}|C_i) = \frac{|F_{i,f_i} \cap C_i|}{|C_i|}$$

It divides the number of times both the category and the feature occurred by the number of times the category occurred. If you're only interested in finding the most likely category for a category, and if you have plenty of data, then this method will usually work well.

This first answer immediately leads to a problem: What do you do about features that have never been observed with a category? Do we assume that the chance of this combination happening in the future is always zero?

Many solutions to this problem have been given. Assume that there are u features that have not been observed with one category. Two common rules-of-thumb are:

1. All unobserved features are part of one share of the probability. If a feature has been observed, it is given the probability $\frac{|F_{i,f_i} \cap C_i|}{|C_i|+1}$. If a feature has not been observed, it is given the probability $\frac{1}{u(|C_i|+1)}$.
2. Each unobserved feature is given one share of the probability. If a feature has been observed, it is given the probability $\frac{|F_{i,f_i} \cap C_i|}{|C_i|+u}$. If a feature has not been observed, it is given the probability $\frac{1}{|C_i|+u}$.

Decision Trees

A decision tree separates the training data through questions at each node of a decision tree.

At each node, a decision tree asks the question that most separates the categories.

According to information theory, the entropy⁶ of any set of data $X = \{x_1, x_2, \dots, x_n\}$ is defined as $H(X) = -\sum_{1 \leq i \leq n} \Pr(x_i) \log_2 \Pr(x_i)$. Assume that x_i is assigned to node n_i , which is category c_i , then the total entropy would be $-\sum_{1 \leq i \leq n} \Pr(c_i|n_i) \log_2 \Pr(c_i|n_i)$ across all data.

The best split has, among the smallest total entropy after the split, the more accurate answers.

⁶This formula measures entropy in bits. Other books may use other bases for the logarithm.

PROBLEMS: (Answers on page 311)

1. (Easy) Why does the text mention “among the smallest total entropy” after the split?

Maximum Entropy classifiers

Like all classifiers, a maximum entropy classifier defines a probability distribution.

A maximum entropy classifier uses restrictions on the probability distribution. It then chooses the classifier that has the maximum entropy among all classifiers that fit all restrictions.

The ‘maximum entropy’ classifier is one that best spreads out probabilities.

Example 7.2 Maximum Entropy examples

Assume that you have no information about the language of a text.

Which probability distribution seems most likely:

	English	Esperanto	Estonian
(a)	100%	0%	0%
(b)	0%	50%	50%
(c)	33.3%	33.3%	33.3%

Assume that you know that there’s a 50% chance that the text is in Estonian, but you know nothing more. Which probability distribution then seems most likely:

	English	Esperanto	Estonian
(a)	50%	0%	50%
(b)	10%	40%	50%
(c)	25%	25%	50%

With no other assumptions, all probability distributions are equally likely.

If there’s no information about a distribution, then assuming that the probabilities are equally spread seems to match many people’s sense of the most natural distribution. Therefore, for most people, (c) seems the most likely distribution.

One way to ensure that the probabilities are equally spread is to choose the distribution with the highest entropy.

The maximum entropy classifier has two important questions:

1. Given sample data, how do we choose restrictions?
2. Given restrictions, how do we find the distribution with the highest entropy?

How to choose restrictions

How to find the highest entropy

Hidden Markov Models

7.1.3 Testing and splitting data

There is an important question: How well does the algorithm work with data that hasn't been seen yet?

Before you perform any tests on any data, it's important to split the collection of data into two groups:

1. Data that will be used for training the algorithm.
2. Data that will only be used at the very end, to determine the strength of the algorithm. ("Test data")

The data used at the end should have the same properties as data that hasn't been seen yet. Which properties? Anything that can be measured and could affect the classification.⁷

As a programmer or statistician, you will often deal with "unknown unknowns"⁸: material that you neither know nor realize that you need to know. Find trusted resources about what you're trying to classify, and ask them about properties that could affect your classification. Find how often those properties occur in reality, and make sure that your test data matches how often those properties actually happen.

It's extremely tempting to use this test data as a rapid check: to try many different classifiers and many parameters for these classifiers, compare the results against this "set aside" data, then use the classifier with the best results.

If you do that, you ruin the ability of that data to answer the question "How well does the algorithm work with data that you haven't seen yet?". This is called overfitting the data. Overfitting means that you're more likely to match the trained data than what you would see in reality.

Comparing algorithms through test data is both good, useful, and easy. Divide data into three groups:

1. Data that will be used for training each algorithm. ("Training data")
2. Data that will be used to evaluate each algorithm. ("Evaluation data")
3. Data that will only be used at the very end, to determine the strength of the algorithm. ("Test data")

Use the evaluation data to decide which algorithm is best, and only use the test data at the very end to measure how well the algorithm works.

⁷An honest reaction to this paragraph is to throw the book against the wall. It's asking you to measure unknowable properties on uncollected data. Keep reading.

⁸Donald Rumsfeld

How large should the test data be?

The rule-of-thumb answer is that the test data should be 10% of the total data. Use that division, and few people will complain.

The real answer is that the size of the test data determines the accuracy of the measurement of how well the algorithm classifies data.

Testing the data can be thought of as a binomial distribution (see section 3.5.2 on page 93) with an unknown probability. This can be tested with a Z-test (or a T-test if your test data is small.)

7.2 Unsupervised Learning

There are some occasions when you know that your data fits into categories, but you don't know what those categories are.

Chapter 8

Statistical Tests for two variables

The previous chapters were about making predictions based on single points of data. This chapter is about pairs of data.

8.1 Pairs of variables

One situation happens regularly:

You have two measurements which are almost identical. One measurement, called the **control**, occurred before a change and the other measurement, called the **treatment**, occurred after the change. You are interested in how the **control** **treatment** change affected the measurements.

Almost always, the right thing to do in this case is to examine the differences¹ between pairs of data.

Paired data is more powerful than two separate variables. It will give definite accurate results than two-variables tests. However, paired data is only accurate when only one factor changes between the two measurements.

Creating experiments that control for every possible factor is difficult, and beyond the subject of this book. When reality has factors that may matter, but can't be gotten rid of, it's best to randomize these factors: to randomly choose parts of your experiments to have each version of the factor.

PROBLEMS: (Answers on page 312)

1. (Easy) Which of the following situations can be used as pairs of variables, and which cannot:

¹or the quotients, or ...

- (a) You're interested in the effects of peanut butter on the happiness of mice. Mice are separated into two groups: the control group gets no peanut butter, but the treatment group gets all the peanut butter it can eat. The mice's happiness is self-reported on a ten-point scale.
- (b) You create makeup, and are interested in whether a lip-gloss makes men more kissably attractive. Half the men in your sample wear the gloss on the left side of their face; the other half wear the gloss on the right side. You measure the number of times that the men get kissed on the side with the lip gloss against the number of times that the men get kissed on the other side.

8.2 Comparing the mean of two samples

When given two sets of data, the most important question is usually “Are the means different?”

8.3 Two-sample Z- and T-test

If both distributions are normal, and they have similar standard deviations, then you can perform a two-sample Z-test or T-test to compare their means.

Example 8.1 Web Site Differences

Imagine that your company does commerce through the Internet. You have two layouts for your website, and you are interested in how the choice of layout affects peoples' buying habits.

Therefore, you show different versions of the website for different people over different days, and you record how much in sales each version brings.

This experiment can't be done by pairing: Few people would say that they would buy a product if the website had the first layout, but would not buy the product with the second layout. Therefore, a two-sample test must be used.

8.3.1 Example: Go Handicaps

I love the game Go². This game is thousands of years old. Although it's a complete-knowledge game with no luck involved, it resists being computerized well.

Even if you've never played Go, you only need to know the following to understand the discussion:

- The game is a contest between two players, called *black* and *white*.
- Because black goes first, he has an advantage.
- If black is a weaker player than white, white may give black one or more *handicap stones*, which are extra moves at the start of the game.
- At the end of the game, each side counts the number of *points* on their side. To offset black's advantage, white adds a few points to her side, called *komi*.
- At the end, the only thing that matters is the difference between black's score and the sum of white plus komi.

This set of rules leads to two questions:

1. How much komi would give black and white an exactly equal game?
2. How much additional komi is a handicap stone worth?

No definitive answer to either question exists. For the first question, different groups advocate komi of 5.5, 6.5, and 7.5. For the second question, the best guesses are between 10 and 13.5 points. So, let's try the power of computer science and statistics against this question.

I had a computer Go program³ play against itself fifteen times, first with no handicap stones; then with one handicap stone. I recorded the amount

If I write a white win as a negative number, then the set of scores with no handicap stones in the fifteen games was: $g_1 = \{-2, 5, -8, 17, 11, -24, 4, 33, 59, -5, 19, -9, 18, 30, -50\}$. The set of scores with an additional handicap stone was: $g_2 = \{10, 31, 47, -14, 21, 10, 0, 36, 25, 28, 10, 9, 4, 6, 4\}$.

Let's analyze this data:

How much komi would give black and white an exactly equal game?

According to the data, the most likely answer is \bar{g}_1 , or about 6.53. But we can use the t-test to give the answer range with 95% probability.

$\sigma(g_1) \approx 25.65$. With 14 degrees of freedom, $t_{0.05} \approx 2.145$. Therefore, with 95% the range $(6.53 - 25.65 \times 2.145, 6.53 + 25.65 \times 2.145) \approx (-49.49, 61.55)$.

²Also written as wei-chi, Baduk, or Goe.

³Gnu Go version 3.7.10, available through "<http://www.gnu.org/software/gnugo/gnugo.html>"

How much komi is a handicap stone worth?

According to the data, the most likely answer is $\bar{g}_2 - \bar{g}_1$, or about 8.6. We can perform the same analysis on the second set of data:

$\bar{g}_2 \approx 15.13$, and $\sigma g_2 \approx 15.87$. Therefore, with 95% of data is in the range $(15.13 - 15.87 * 2.145, 15.13 + 15.87 * 2.145) \approx (-18.91, 49.17)$

Both of these ranges are very large. The next question is:

Can we even determine that the two sets have a different mean?

8.4 Pearson's chi-square test

Chapter 9

Statistical Tests — Multiple

9.1 Multiple sets of variables

9.1.1 Fisher's Least Significant Difference

9.1.2 Bonferroni Correction

9.2 ANOVA

In the chapter of hypothesis tests, we examined whether two sets had the same mean. What happens if there are more than two sets?

Chapter 10

Sampling

The previous chapter asked questions about data where we knew the distribution. Often, we even knew some of the parameters in the distribution. We don't always have that information.

Often, we want to know information, even when we do not know the distribution. Other times, we want to know information that isn't from the parameters of the distribution. Sampling gives us a way to do both.

Sampling is powerful and simple. To find the probability of an event happening in certain circumstances, sampling suggests making the circumstances happen many times, then count the proportion that the event happens.

Sampling can never answer as accurately as mathematical computation can. But sampling can answer questions even when no mathematical solution exists.

10.1 Basic concepts in sampling

Some concepts in sampling are not obvious. To get around this, I'll present a simple example: If you flip two coins, what is the chance that both coins will land heads? (Pretend that you don't know that the answer is $\frac{1}{4}$.)

Code for this simulation is simple:

```
double coinFlips(int numRuns)
{
    int count;
    int run;

    count = 0;
    for (run = 0; run < numRuns; run++) {
        if (random() % 2 == 0 && random() % 2 == 0)
            count++;
    }
}
```

```

    return (double) count / (double) numRuns;
}

```

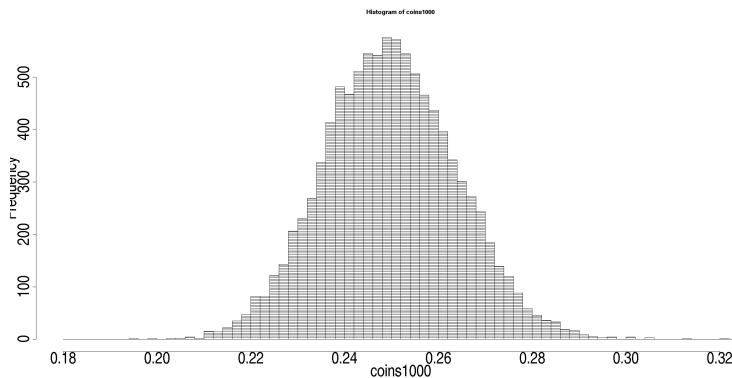
Running this function once with 1000 coin flips gives 0.253 as the answer. Can I therefore say that the chance that flipping two coins will give heads is exactly 25.3%? Hardly.

Running the same function ten times with 1000 coin flips each time gives ten different answers:

0.253	0.222	0.244	0.254	0.247
0.231	0.239	0.242	0.233	0.254

Running the function 10,000 times with 1000 coin flips each time gives a histogram:

Figure 10.1: Coin flip histogram



I can sort these 10,000 pieces of data, and get the 250th and 9,750th elements. This exercise estimates the 95% confidence interval: [0.223, 0.278].

Running the function 10,000 times with 10,000 coin flips each time gives another histogram, and a better 95% confidence interval: [0.2416, 0.2588]. We can compare the two histograms:

As you add more flips per trial, you gain accuracy for the guess. As you add more trials, you gain accuracy for the 95% confidence interval. The following chart shows the estimated probability as the number of flips per trial increases.

The more data used, the better the results.

PROBLEMS: (Answers on page 313)

1. (Easy) The coin flip example is a binomial distribution. By mathematics alone, what is the expected 95% confidence interval for the 1000-flip sample?

Figure 10.2: Coin flip comparison

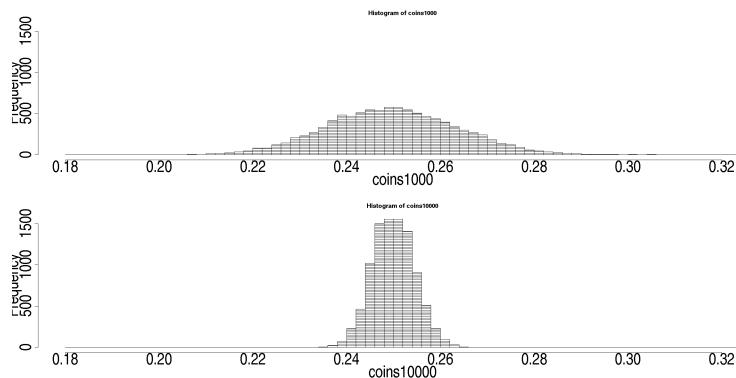
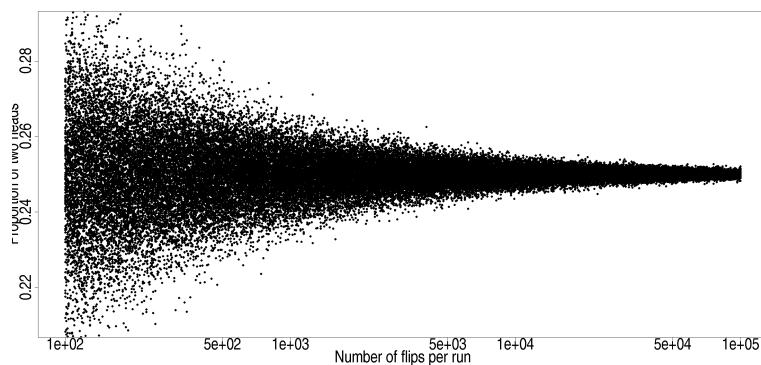


Figure 10.3: Coin flip accuracy



10.2 Monte Carlo Methods

Monte Carlo Method The most general kind of simulation is called a **Monte Carlo Method**. The algorithm for a Monte Carlo Method is algorithm 10.1 on page 226

Required: A distribution of some random variable v , a property, P , on the random variable, and some count, $n > 0$.

1. Generate n random variables v_1, v_2, \dots, v_n without bias according to the distribution.
2. Determine how many random variables have the desired property. Call this count c .
3. Return c / n , the proportion of random variables that have the property.

Algorithm 10.1: Algorithm for a Monte Carlo Method

According to <http://mathworld.wolfram.com/MonteCarloMethod.html>, “It was named by S. Ulam, who in 1946 became the first mathematician to dignify this approach with a name, in honor of a relative having a propensity to gamble.¹” After you’ve read this section, you no longer have to say that you’re “trying anything at random to see what works.” You can say that you’re “applying the Monte Carlo Method to solve my problem.”

Selecting random elements without bias is often very difficult.

Example 10.1 Monte Carlo: Images

The Monte Carlo simulation could estimate what percent of images on the Internet have people’s faces in them. Our random variable would be images in the Internet, and the desired property would be whether the image has a face.²

The difficulty is in finding a random selection of images. Most search strategies will find multiple copies of the most popular pictures, but few copies of unpopular or uncommon pictures. Further, search engines bias either against or for adult material, which often include faces.

The problem of biased data shows up in many circumstances: any time some data is easier to gather than other data, the researcher will be tempted to choose only the easiest data. These biases often change the results.

¹Hoffman, P. *The Man Who Loved Only Numbers: The Story of Paul Erdos and the Search for Mathematical Truth*. New York: Hyperion, pp. 238-239, 1998.

²Feel free to write this program!

Example 10.2 Percentage of points in a Mandelbrot set

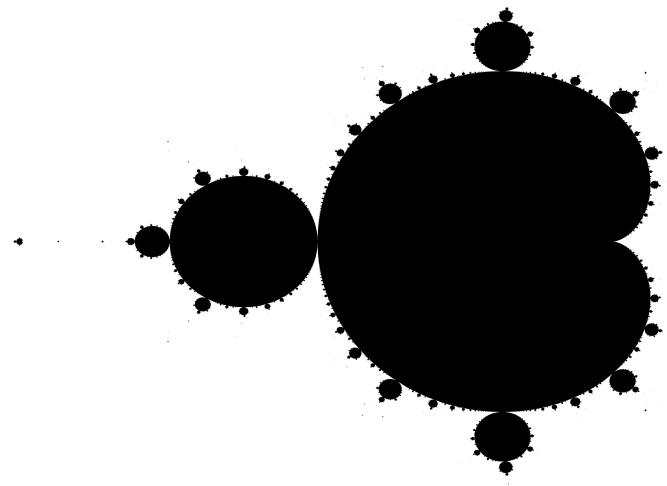
The **Mandelbrot set** is extremely complex but simple to model. A complex point, p , is in the Mandelbrot set if the following series of points does not diverge to infinity:

Mandelbrot set

$$\begin{aligned} p_0 &= 0 \\ p_1 &= p_0^2 + p \\ p_2 &= p_1^2 + p \\ p_3 &= p_2^2 + p \\ &\dots \\ p_n &= p_{n-1}^2 + p \end{aligned}$$

All points in the Mandelbrot set are in the range $[-2, 2] + [-2, 2] \times i$ where $i = \sqrt{-1}$. What proportion of points in the range $[-2, 2] + [-2, 2] \times i$ are in the Mandelbrot set?

Figure 10.4: The Mandelbrot Set



The sec

```

bool insideMandelbrot(std::complex<double> point)
{
    int i;
    std::complex<double> iter;

    iter = 0;
    for (i=0; i<1000; i++) {
        iter = iter * iter + point;
        if (std::abs(iter) > 2) {
            return false;
        }
    }

    return true;
}

double estimateMandelbrot(int repetitions)
{
    int count;
    int repNumber;
    std::complex<double> *point;
    double x;
    double y;

    count = 0;
    for (repNumber = 0; repNumber < repetitions; repNumber++)
    {
        x = ((double) random() * 4.0 / (double) RAND_MAX) - \
            2.0;
        y = ((double) random() * 4.0 / (double) RAND_MAX) - \
            2.0;

        point = new std::complex<double>(x, y);
        if (insideMandelbrot(*point)) {
            count++;
        }
        delete point;
    }

    return (double) count / (double) repetitions;
}

```

The function `insideMandelbrot` determines whether a complex point is within the Mandelbrot set, and `estimateMandelbrot` runs the Monte Carlo method.

When I ran the program, 10,000,000 repetitions gave the estimate 0.0944879

for the proportion.

PROBLEMS: (Answers on page 313)

1. (Easy) Use the Monte-Carlo method to estimate the percent of integers between 2 and RAND_MAX that are prime.
2. (Medium) **Monte Carlo Integration** is a different way to integrate a function $f(x)$ than the Riemann Rule of page 49. Instead of dividing the interval into strips δ wide, Monte Carlo integration chooses n (uniformly-distributed) random points, p_1, p_2, \dots, p_n from the interval and computes $\sum_{p \in p_i} \frac{f(p)}{n}$. Write a new function, `double Integrate::MonteCarloIntegrate(Interval interval)` that uses this method. How well does it seem to work?

10.3 Resampling

In the previous section, we were given a function that we could take many samples from. Often, instead, we have a set of data, and no further actual data will come. How can we simulate an infinite amount of data from a finite source?

10.3.1 Resampling without parameters

10.3.2 Resampling with parameters

10.4 Markov Chain

10.4.1 Breaking Caesar ciphers

10.5 MCMC

10.5.1 Improving compression codes

Even with vast increases in storage and transmission speed, no one ever gets enough storage or transmission speed. Everyone wants documents, pictures, music, and movies to take less space or to transmit more quickly. Data compression encodes the information so that information is usually represented in fewer bits.

Most data has a natural or standard representation. For text, the standard representation among computers is through Unicode. For pictures, the natural representation is to represent each pixel with its color values.

However, the most natural or standard representation rarely compact. People use a **compression code** when space or transmission time is at a premium. **compression code**

A compression code attempts to write data in fewer bits than the standard representation.

Many introductory textbooks teach a compression code: either the Huffman code, or run-length encoding. Conditional probability can improve many compression codes, including both of those. We'll first go through a more advanced compression code, then go through how conditional probability improves the code.

Arithmetic code

Example 10.3 A thought experiment for the arithmetic code

Think about a rod exactly one meter long. You would like to leave a message on it with a single mark. Here's one way to do it:

Mentally divide the rod in half; a mark anywhere in the top half of the rod can be considered a '1'; a mark anywhere in the lower half of the rod can be considered a '0'. The rod can be subdivided: a mark in the top quarter of the rod can be considered '11', in the second quarter '10', and so forth. If the rod could be infinitely subdivided, and if the mark can be placed with perfect precision, then it could hold a message of any length.³

The above example is a thought experiment. It's similar to saying that a real number can only be represented with an infinite number of bits, and therefore contains an infinite amount of data.

The natural representation for the rod for this experiment is to divide the rod into 2^n equal-sized segments, determine which segment the mark is in, and read that number as a binary number. Or, one can regard the mark as a binary decimal between 0 and 1, and by recursively dividing the rod, one can read the binary number from the rod. The natural representation of the rod's value gives an infinite, real number that starts with the given text.

So that we have a natural end point, we stop writing bits when all remaining bits are zero. So far, the rod has little to do with a compression code.

The final idea, which causes the rod to become a compression code: if ones and zeros did not occur in equal proportion, a person could divide the rod according to a weighted proportion. For example, if 0's occurred seven times more frequently than 1's, then the top 7/8 of the rod could be used for a 0 and the bottom 1/8 could be used for a 1. This idea could be used recursively.

If 0's occurred seven times more frequently than 1's, then you could compress by finding a point on the left side of the illustration with your bits, and reading that point using the values on the right side of the illustration. Decompressing happens when you read in the opposite order.

³Obviously, this rod only exists in the imagination. Real rods can only hold a mark to a certain precision; if nothing else, atoms don't stay still.

Figure 10.5: Dividing a rod recursively

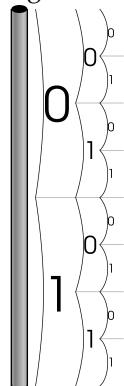
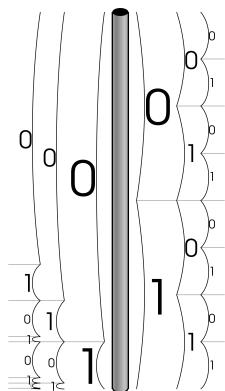


Figure 10.6: Two different divisions



(The bits are meant to be read from the inside out.)

Code

The central concept for the arithmetic code is in the rod. The rod holds a value (in our case, `m_bestInputValue`) that represents all bits input so far.

Input affects three values: two that determine the range that any future values can be in, `m_lowerInputBounds` and `m_upperInputBounds`, and a value for all bits input.

Output affects two more values: a range, similar to that of the input, `m_lowerOutputBounds` and `m_upperOutputBounds`.

They both use a division line, which has a function, `getDivider`, that returns a number within (0, 1). It is a function, and it may return different values with different calls. It also accepts, through its `input` method, the last bit that was seen.

The actual code for the rod follows:

```
template<class M>
void Rod<M>::input (bool bit)
{
    M divider;

    divider = m_pDivisionLineInput->getDivider();
    m_bestInputValue = m_lowerInputBounds + divider * (m_upperInputBounds - m_lowerInputBounds);

    if (!bit) {
        // m_lowerInputBounds remains the same.
        m_upperInputBounds = m_bestInputValue;
    }
    else {
        m_lowerInputBounds = m_bestInputValue;
        // m_upperInputBounds remains the same.
    }

    m_pDivisionLineInput->input (bit);
    m_lastBit = bit;
}

template<class M>
bool Rod<M>::output ()
{
    bool bit;
    M divider;
    M newLine;

    divider = m_pDivisionLineOutput->getDivider();
    newLine = m_lowerOutputBounds + divider * (m_upperOutputBounds - m_lowerOutputBounds);
```

```

if (newLine > m_bestInputValue) {
    bit = false;

    // m_lowerOutputBounds remains the same.
    m_upperOutputBounds = newLine;
}

else if (newLine < m_bestInputValue) {
    bit = true;

    m_lowerOutputBounds = newLine;
    // m_upperOutputBounds remains the same.
}

else {
    bit = m_lastBit;
}

m_pDivisionLineOutput->input(bit);

return bit;
}

```

Compression size

The compressed text would be written in $a_0 \times -\log_2(e_0) + a_1 \times -\log_2(e_1)$ bits, where a_0 and a_1 are the count of zero bits and one bits that are being compressed, and e_0 and e_1 are the fraction of zero and one bits that are expected.

For example, if our code is set up so that 0's occur seven times more frequently than 1's, then a message with seven 0's and one 1 could be compressed to $7 \times -\log_2(\frac{7}{8}) + 1 \times -\log_2(\frac{1}{8}) \approx 7 \times 0.1926 + 1 \times 3 = 4.3482$ bits. Because we cannot transmit less than a bit of information, the compressed message takes 5 bits.

A better compression code

The above system has a problem: for most real data, the number of 0's and the number of 1's will be about the same. Therefore, for most cases, this system gives little compression.

One idea makes the arithmetic compression code among the best compression models: the weighted proportions don't have to be the same. In fact, the proportion is not needed until the moment that a bit is encoded. Therefore, the proportions can adapt to anything in the environment, including bits that have been sent before.

You might call this idea by other names, but it's just another example of conditional probability.

Hundreds of ways exist to specify these predictions. I'll give two very simple ones.

The previous n bits

One very obvious way to predict a bit is by using the previous n bits; the bits that come before tend to force the next bit. The classic example of this idea is that the letter *q* almost always has the letter *u* after it.⁴

Even a small number of previous bits gives some compression. Using this chapter as an example, using just 3 bits of prediction has the following probabilities:

Predicting bits	Total Count	Count 0	% 0 after	Count 1	% 1 after
000	77036	43764	56.8%	33272	43.2%
001	67002	36315	54.2%	30687	45.8%
010	60699	33047	54.4%	27652	45.6%
011	65538	39627	60.5%	25911	39.5%
100	67001	33271	49.7%	33730	50.3%
101	59236	24385	41.2%	34851	58.8%
110	65538	33954	51.8%	31584	48.2%
111	33758	25911	76.8%	7847	23.2%

Position within a byte

As you well know, characters in ASCII or Unicode aren't used in equal proportion. Similar characters are grouped together. We can use this knowledge for an equally simple prediction: given a byte, we can estimate the percent of time each bit is 0 or 1.

Again using this chapter as sample text, we get the following predicted percentages:

Bit in byte	Total Count	Count 0	% 0 occurring	Count 1	% 1 occurring
0	62203	62203	100.0%	0	0.0%
1	62203	16164	26.0%	46039	74.0%
2	62203	5225	8.4%	56978	91.6%
3	62203	41868	67.3%	20335	32.7%
4	62203	38862	62.5%	23341	37.5%
5	62203	32078	51.6%	30125	48.4%
6	62203	40842	65.7%	21361	34.3%
7	62203	34142	54.9%	28061	45.1%

In other words, if you examine the 2-bit of every byte that makes up this chapter's electronic text, about 8.4% would be unset, and 91.6% would be set. That makes sense, because the vast majority of this chapter is text, the majority of the text is in lower-case, and in ASCII, lower-case letters have the 2-bit set. (They occur from 97-122 in ASCII.)

⁴Iraq, Compaq and Qing don't have a *u* after the *q*.

Compression size

The original text took 497624 bits to write.

When multiple variants of the compression code exist, then the total compression size is the sum of all of them: $\sum_i (a_{0,i} \times -\log_2(e_{0,i}) + a_{1,i} \times -\log_2(e_{1,i}))$.

For example, if this chapter were compressed according to the 3-bit prediction, we can compute the number of bits it would take:

matching	fraction	matching * $-\log_2(\text{fraction})$
43764	0.568	35713
33272	0.432	40289
36315	0.542	32089
30687	0.458	34571
33047	0.544	29026
27652	0.456	31327
39627	0.605	28729
25911	0.395	34723
33271	0.497	33569
33739	0.503	33448
24385	0.412	31195
34851	0.588	26799
33954	0.518	32222
31584	0.482	33255
25911	0.768	9867
7874	0.232	16597
Total		483419

In short, we get about a 2.9% compression with keeping track of the previous three bits. Keeping track of more bits gives better compression, but it requires more memory and longer tables in this textbook.

PROBLEMS: (Answers on page 313)

1. (Easy but long) Compute the number of bits that the “position within a byte” method requires.

PROJECTS:

Projects are much longer than problems. They don’t have answers in the back of the book. They may not even have answers. They’re meant to be springboards for you to read further.

1. Although the code and diagrams that I give in this text were accurate, they’re not complete. For example, to compress text of any length, this implementation requires infinite-precision rational numbers – doubles can only compress a few bits. In addition, real arithmetic code implementations output bits as soon as they know that the output will be within a range. Research how the arithmetic code is really implemented, and create a proper implementation. I recommend [Sayood, 2000, pages 77-116], but you can use any reference that you prefer.

Chapter 11

Bayesian methods

11.1 Bayesian Inference

11.1.1 Priors and likelihoods

11.2 Hierarchical Models

Chapter 12

Time Series

12.1 Time Series Decomposition

12.2 Time Series Regression

12.3 Nonseasonal Box-Jenkins Models

12.4 Seasonal Box-Jenkins Models

Chapter 13

Markov Chain - Monte Carlo

13.1 EM Algorithm

13.1.1 EM Variance Estimation

13.2 Markov Chains

13.3 Monte Carlo Integration

13.4 Markov Chain - Monte Carlo

13.4.1 Metropolis-Hastings Algorithm

13.4.2 Gibbs Sampler

13.5 Bootstrapping

13.5.1 Bootstrap-t confidence interval

13.5.2 BCA Intervals

Chapter A

Answers

Stuck on a problem? This section answers every problem in the text.

Most books contain a warning that the student should try her hardest to solve the problem before she reads the answers. All students ignore that warning. You know your best way to learn, and you know your goals.

The answers take up a large part of this book. I consider them important. Even if you've successfully solved a problem, the answers may give you a different way to think about a problem.

Many problems in this book come from mistakes. When I realized that a statement was wrong, I corrected the statement – then I often used it to create a problem.

A.1 Mathematical Fundamentals

PAGE 3:

1. My source code for this question is in the CD-ROM, in the files `Interval.cpp` and `Interval.h`.
2. The interval $[0, 10)$ contains all real numbers, like 3.14, but the sequence does not.

PAGE 5:

There are many possible answers to these two questions.

1. The most common answer is \mathbb{Q} , the rational number set.
2. Two easy answers:
 - (a) $\mathcal{R} \setminus (0, 1)$ has no numbers between 0 and 1. Other ideas in this vein include the Cantor ternary set.
 - (b) The power set of the natural numbers.

PAGE 7:

1. The code has two parameters: `lhs`, of type `T`, and `rhs`, of type `std::set<T>`. The `lhs` parameter is always exactly one element of the class: never less, never more. Since \emptyset has zero members, it can never be a member of type `T`.

PAGE 9:

1. Here's the code that I wrote:

```

template <typename T>
const std::set<T> finite_intersection(const std::set<T>&
                                         lhs, const std::set<T>&
                                         rhs)
{
    std::set<T> retval;

    typename std::set<T>::const_iterator iterLhs;
    typename std::set<T>::const_iterator iterRhs;

    iterLhs = lhs.begin();
    iterRhs = rhs.begin();

    while (iterLhs != lhs.end() && iterRhs != rhs.end())
    {
        if (*iterLhs == *iterRhs) {
            retval.insert(*iterLhs);

            iterLhs++;
            iterRhs++;
        } else if (*iterLhs < *iterRhs)
            iterLhs++;
        else if (*iterLhs > *iterRhs)
            iterRhs++;
        else {
            // This isn't possible with well-behaved
            // comparison functions, but not all
            // comparison functions are well-behaved.
            throw std::domain_error("Bad_comparison_"
                                   "operators.");
        }
    }

    return retval;
}

```

PAGE 11:

1. On first glance, this problem seems easy: $s_1 \cup s_2$ must have three green balls and seven red balls, right? That's only true if s_1 and s_2 are disjoint.

Let's distinguish the balls, and call them G_1, G_2, G_3 (for GREEN), and R_1, R_2, \dots, R_7 (for RED). Now, let the following be true:

$$\begin{aligned}s_1 &= \{G_1, R_1, R_2, R_3\} \\ s_2 &= \{G_1, G_2, R_1, R_2, R_3, R_4\}\end{aligned}$$

Then $s_1 \cup s_2 = \{G_1, G_2, R_1, R_2, R_3, R_4\}$, or just two greens and four reds. Until which elements s_1 and s_2 have in common is known, the question can't be answered.

PAGE 13:

1. Here's the true and false answers:
 - (a) *True*: Every measurement in A occurs in A .
 - (b) *True*: Every measurement in $A \cap B$ occurs in A .
 - (c) *Sometimes true, sometimes false*: If $B = \emptyset$, then $A \cup B \subseteq A$
 - (d) *True*: Since there are no measurements in \emptyset , every measurement in \emptyset occurs in A .
 - (e) *Sometime true, sometimes false*: If $A = \emptyset$, then $A \subseteq \emptyset$.
 - (f) *Sometimes true, sometimes false*: If $A \subseteq B$, then $A \subseteq A \cap B$.
 - (g) *True*: Every measurement in A occurs in $A \cup B$.

2. Here's my code:

```
template <class T>
bool subset(std::set<T> lhs, std::set<T> rhs) {
    return (finite_intersection(lhs, rhs) == lhs);
}
```

PAGE 15:

1. Here's my code:

```
template <typename T>
FiniteSet<T> FiniteSet<T>::difference(const \
    FiniteSet<T>& rhs) const {
    std::multiset<T> retval;
    typename std::multiset <T>::const_iterator \
        iterThis = m_set.begin();
    typename std::multiset <T>::const_iterator \
        iterRhs = rhs.m_set.begin();
```

```

while (iterThis != m_set.end() && iterRhs != rhs
    .m_set.end()) {
    if (*iterThis == *iterRhs) {
        iterThis++;
        iterRhs++;
    } else if (*iterThis < *iterRhs) {
        retval.insert(*iterThis);
        iterThis++;
    } else {
        // *iterThis > *iterRhs
        iterRhs++;
    }
}

while (iterThis != m_set.end()) {
    retval.insert(*iterThis);
    iterThis++;
}

return FiniteSet<T>(retval);
}

```

PAGE 17:

1. Here's one way to do it:

```

template <typename T>
const std::set<T> set_difference(const std::set<T>& \
    lhs, const std::set<T> rhs)
{
    std::set <T> negrhs; // Negation of the left-
                           hand side

    neglhs = set_negation(rhs);
    return intersection(lhs, negrhs);
}

```

2. This code is available on the included CD ROM.

PAGE 20:

```

1. double product()
{
    double result;
    double i;

    result = 1;
    for (i=a; i<=b; i++) {
        result *= f(i);
    }

    return result;
}

```

2. The easiest way to prove its value is to set the series to a variable, x .

$$\begin{aligned}x &= 0 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \\2x &= 2 + 1 + \frac{1}{2} + \frac{1}{4} + \dots\end{aligned}$$

Subtract the first equation from the second:

$$x = 2$$

PAGE 20:

1. Since the sets each have n_1, n_2, \dots, n_c elements, there are $\prod_{x=1}^c n_x$ arrays with one element from each set. This can be proven by induction:
 - (a) When only one set exists, and it has n_1 elements, then exactly n_1 arrays can be made from it.
 - (b) Assume that the expanded product rule is true when one uses $c - 1$ or fewer sets. Then consider using c different sets. The first $c - 1$ sets can be combined into one array with $\prod_{x=1}^{c-1} n_x$ elements. By the (binary) product rule, we can combine this large array with a set with c_n elements to create an array with $\prod_{x=1}^c n_x$ elements.

PAGE 22:

1. There are $6!$, or $1 \times 2 \times 3 \times 4 \times 5 \times 6$, or 720 ways for me to do my morning routine. Half of them have me walking the dog before I get dressed. I have very tolerant neighbors.

2.

$$\prod_{i=1}^{12} i$$

3. The question expects a subroutine like

```
double recursive_factorial(int n)
{
    if (n < 0) {
        // Error!
        return 0;
    }

    if ((n == 0) || (n == 1)) {
        return 1;
    } else {
        return n * recursive_factorial(n - 1);
    }
}
```

Every time that you recurse in C++, the computer has to store all of the variables in the current subroutine on the stack. Therefore, the space required for a recursive definition of $n!$ is $O(n)$, while the space required for the iterative definition is constant. Further, although both programs take linear time to solve the problem, the recursive procedure will take longer, because it goes through a function call for each product. Finally, the recursive solution is no simpler for other users to read.

Therefore, it has nothing to recommend it ... and plenty of reason to oppose it.

4. This is proved best by recursion:

- (a) If n is 1, then there is one way to order n items, and the product of all numbers from 1 to 1 is 1. Therefore, the two are the same.
- (b) Assume that the two are equivalent for all numbers equal to $n - 1$ or less. Prove that they are equivalent when you have n items. Imagine the list of n items: you have n choices for the first item in a list. After that choice, you have $n - 1$ items, and $n - 1$ places to put them in. By the assumption, there are $(n - 1) \times (n - 2) \times \dots \times 1$ ways to order those items. Since every different combination of an initial item and order of the rest of the items generates a different order to the n items, there are $n \times (n - 1)!$ different ways to order n items. Since $(n - 1)! = (n - 1) \times (n - 2) \times \dots \times 1$, there are $n \times (n - 1) \times (n - 2) \times \dots \times 1$ ways to order n items.

1. permutations (39, 5) is $\frac{39!}{(39-5)!}$ or $39 \times 38 \times 37 \times 36 \times 35$ or 69,090,840. Therefore, your chances are one in 69,090,840.
2. permutations (78, 10) is $\frac{78!}{(78-10)!} = 4\,566\,176\,969\,818\,464\,000$ different readings from a Tarot deck. Why do I always get generic readings like, "You sometimes have trouble making up your mind. You will be going through some kind of change in the near future. You may have money problems."?
3. $P(12, 5)$ can be written

$$\prod_{i=8}^{12} i$$

Be careful of the starting value!

4. If you had said that the code in the text was faster, give yourself a third of a point. That's true, but it misses the point.

As long as the parameters are small, the two methods give the same answer. If parameters are large, the method in the text can give answers even when the factorial method cannot. For example, the method in the text can easily compute permutations (300, 3) – but the factorial-based method runs out of space trying to compute factorial (300).

5. Many efficient algorithms compute a random permutation. Your answer may be different.

```
template <typename T>
const std::vector<T> random_permutation(const std::vector<T>& source)
{
    int lastElement;
    int randomElement;
    std::vector<T> result(source.size());

    copy(source.begin(), source.end(), result.begin());
    ;

    lastElement = source.size() - 1;

    for (lastElement = source.size() - 1; lastElement > 0; lastElement--) {
        randomElement = std::rand() % (lastElement + 1);

        if (randomElement != lastElement) {
            std::swap(result[randomElement], result[lastElement]);
        }
    }
}
```

```

        }
    }

    return result;
}

```

This function runs in linear time.

6. The following code replaces the randomness of the previous function with a way to compute the nth permutation:

```

template <typename T>
const std::vector< T > all_permutations(const std::vector< T >& source, long int permutation)
{
    long int factElement;
    int lastElement;
    int randomElement;
    std::vector< T > result(source.size());

    if (permutation < 0 || permutation > longFactorial< T >(source.size())) {
        throw std::out_of_range("The requested permutation is outside range.");
    }

    copy(source.begin(), source.end(), result.begin());
}

lastElement = source.size() - 1;

for (lastElement = source.size() - 1; lastElement > 0; lastElement--) {
    factElement = longFactorial(lastElement);
    randomElement = permutation / factElement;
    permutation %= factElement;

    if (randomElement != lastElement) {
        std::swap(result[randomElement], result[lastElement]);
    }
}

return result;
}

```

The function `long_factorial` is the factorial function of page 21, but returning `long int` instead of `double`.

PAGE 25:

1. As the text says, combinations (10, 3) is $10!/(3! \times 7!)$, or 120.
2. Each choice has n possibilities, and there are r choices. Therefore, the mathematical formula would be n^r . I leave to the most creative readers the problem of finding the number of solutions where you choose items with replacement and when order does not matter.

$$\begin{aligned}3. \quad (a) \quad & \binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{P(n,r)}{r!} \\(b) \quad & \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!} = \frac{n!}{(n-r)!(n-(n-r))!} = \frac{P(n,n-r)}{(n-r)!}\end{aligned}$$

4. Here's the source code that I wrote:

```
long int longFactorial(int n)
{
    long int result;
    int i;

    result = 1;
    for (i=2; i<=n; i++)
        result *= i;

    return result;
}

template <typename T>
const std::vector< T > all_combinations(const std::vector<T>& source,
                                            long int combination)
{
    long int factElement;
    int lastElement;
    int randomElement;
    std::vector<T> result(source.size());

    copy(source.begin(), source.end(), result.begin());
    lastElement = source.size() - 1;

    for (lastElement = source.size() - 1; lastElement > 0; lastElement--) {
        factElement = longFactorial(lastElement);
        result[lastElement] = combination;
    }
}
```

```

    randomElement = combination / factElement;
    combination %= factElement;

    if (randomElement != lastElement)
        std::swap(result[randomElement], result[\
            lastElement]);
    }

    return result;
}

```

This returns a single combination. Using the code is simple:

```

maxComb = longFactorial(10);
for (comb = 0; comb < maxComb; comb++) {
    randomized = all_combinations(numbers, comb);
    . . .
}

```

PAGE 30:

1. Not all data sets that have orderings are numeric. For example, Edmond's ranking of restaurants can have an even number of elements, and there may not exist a restaurant between the two middlemost restaurants.
2. (a) No, it does not matter. Give the \$1000 to the richest or the poorest person in town; the mean would always go up by $\$ \frac{1000}{n}$.
(b) Yes, it does matter. Only by giving to someone from the bottom half of the town would you raise the median wage.
3. The median might not be unique. For example, in the set $\{3, 4, 4, 5\}$, the median is 4, but it's less than or equal to $\frac{3}{4}$ of the values and greater than or equal to $\frac{3}{4}$ of the values.
4. (a) Males score higher than females in engineering.
(b) Males score higher than females in literature.
(c) The average score for males is $\frac{70}{90} \times 72 + \frac{20}{90} \times 94 \approx 76.888$. The average score for females is $\frac{30}{110} \times 68 + \frac{80}{110} \times 91 \approx 84.727$. Therefore, females have the higher overall score.
5. Assume that the set of data has n bytes. Then the speed of the first set was $\frac{n}{3}$ bytes/minute, and the speed of the second set is $\frac{n}{5}$ bytes/minute. Therefore, the average speed is $(\frac{5n}{15} + \frac{3n}{15}) / 2 = \frac{4n}{15}$ bytes / minute. This will take $\frac{15}{4}$ minutes on average.

The average of speeds is counter-intuitive.

PAGE 33:

1. Here's the source code I wrote:

```
template <class T>
T Data<T>::percentile(double percentile) const
{
    typename std::multiset<T>::const_iterator iter;
    typename std::multiset<T>::size_type \
        positionPercentile;
    typename std::multiset<T>::size_type position;
    typename std::multiset<T>::size_type size;

    size = size();
    positionPercentile = (typename std::multiset<T>::size_type) std::floor((size-1) * \
        percentile / 100.0);

    iter = FiniteSet<T>::m_set.begin();
    for (position = 0; position < positionPercentile \
        ; position++)
        iter++;

    return *iter;
}
```

PAGE 34:

1. **template <class T>**
T Data<T>::order(**int** orderPosition) **const**
{
 typename std::multiset<T>::const_iterator iter;
 typename std::multiset<T>::size_type position;

 iter = FiniteSet<T>::m_set.begin();
 for (position = 0; position < orderPosition; \
 position++)
 iter++;

 return *iter;
}

2. Here's the source code I wrote:

```

const int k_numExperiment = 100000;

int main(void)
{
    int countNumbers;
    int experiment;
    int i;
    double minRandom;
    double nextRandom;
    double totalMin;

    srand48((long int) std::time(NULL));

    for (countNumbers = 2; countNumbers < 10; \
          countNumbers++) {
        totalMin = 0.0;

        for (experiment = 0; experiment < \
              k_numExperiment; experiment++) {
            minRandom = drand48();

            for (i = 1; i < countNumbers; i++) {
                nextRandom = drand48();

                if (nextRandom < minRandom)
                    minRandom = nextRandom;
            }

            totalMin += minRandom;
        }

        std::cout << "The minimum of " << \
                  countNumbers << " uniformly distributed " \
                  "numbers averages to " << totalMin / \
                  k_numExperiment << std::endl;
    }

    return 0;
}

```

Here's the output that I received:

```

The minimum of 2 uniformly distributed numbers averages to 0.333317
The minimum of 3 uniformly distributed numbers averages to 0.250493
The minimum of 4 uniformly distributed numbers averages to 0.199424

```

The minimum of 5 uniformly distributed numbers averages to 0.166336
 The minimum of 6 uniformly distributed numbers averages to 0.142561
 The minimum of 7 uniformly distributed numbers averages to 0.125208
 The minimum of 8 uniformly distributed numbers averages to 0.110758
 The minimum of 9 uniformly distributed numbers averages to 0.100082

The minimum of the a set of n random numbers seems to be about $\frac{1}{n+1}$.

PAGE 36:

1. The distance is always 0 when all data have the same value, no matter how many elements are added.
2. Only three. Your program needs to remember `sum_of_squares`, `square_of_sum`, and `my_size`. These three variables can be accumulated with every new piece of data.
3. The program uses the equation

$$\frac{\sum_{i=1}^n (a_i)^2 - \frac{(\sum_{j=1}^n a_j)^2}{n}}{n - 1}$$

Let's compute:

$$\begin{aligned} \text{source} &= \frac{\sum_{i=1}^n (a_i)^2 - \frac{(\sum_{j=1}^n a_j)^2}{n}}{n - 1} \\ &= \frac{\frac{(\sum_{j=1}^n a_j)^2}{n} - 2 \frac{(\sum_{j=1}^n a_j)}{n} + \sum_{i=1}^n (a_i)^2}{n - 1} \\ &= \frac{\frac{n(\sum_{j=1}^n a_j)^2}{n^2} - 2 \frac{(\sum_{i=1}^n a_i)(\sum_{j=1}^n a_j)}{n} + \sum_{i=1}^n (a_i)^2}{n - 1} \\ &= \frac{\sum_{i=1}^n \left(\frac{(\sum_{j=1}^n a_j)^2}{n^2} - 2a_i \left(\frac{\sum_{j=1}^n a_j}{n} \right) + (a_i)^2 \right)}{n - 1} \\ &= \frac{\sum_{i=1}^n \left(\frac{\sum_{j=1}^n a_j}{n} - a_i \right)^2}{n - 1} \\ &= \frac{\sum_{i=1}^n (\bar{a} - a_i)^2}{n - 1} \end{aligned}$$

4. Let's try this problem mathematically:

$$\begin{aligned}
 \frac{\sum_{i=1}^n (a_i - \bar{a})}{n} &= \frac{\sum_{i=1}^n a_i}{n} - \frac{\sum_{i=1}^n \bar{a}}{n} \\
 &= \frac{\sum_{i=1}^n a_i}{n} - \frac{\bar{a} \times n}{n} \\
 &= \bar{a} - \bar{a} \\
 &= 0
 \end{aligned}$$

This distance would always be zero.

5. Here's the code that I used:

```

const int k_maxExperiments = 100;
const int k_maxIterations = 1000;
const int k_maxValue = 100;

int main(void)
{
    double lowerBound;
    double mean;
    double stdDev;
    double sumOfSquares;
    double upperBound;
    int data[k_maxIterations];
    int i;
    int iter;
    int sum;
    int withinBounds;

    for (iter = 0; iter < k_maxExperiments; iter++)
    {
        for (i = 0; i < k_maxIterations; i++)
            data[i] = lrand48() % k_maxValue;

        // Get the mean value
        sum = 0;
        for (i = 0; i < k_maxIterations; i++)
            sum += data[i];
        mean = (double) sum / (double)
            k_maxIterations;

        // Get the standard deviation
        sumOfSquares = 0;
        for (i = 0; i < k_maxIterations; i++)
    }
}

```

```

        sumOfSquares += data[i] * data[i];
stdDev = std::sqrt(sumOfSquares/\_
k_maxIterations - mean*mean);

// How many within bounds
withinBounds = 0;
lowerBound = mean - (2 * stdDev);
upperBound = mean + (2 * stdDev);
for (i = 0; i < k_maxIterations; i++)
    if (lowerBound <= data[i] && data[i] <= \
        upperBound)
        withinBounds++;

// Output
std::cout << "The mean was " << mean << ", the standard deviation was "
            << stdDev << " and the number within bounds was "
            << withinBounds << "/" << \
            k_maxIterations << std::endl;
}
}

```

The above program gives the following results:

```

The mean was 49.391, the standard deviation was 29.1036
and the number within bounds was 1000/1000
The mean was 48.647, the standard deviation was 28.42
and the number within bounds was 1000/1000
The mean was 50.927, the standard deviation was 29.1494
and the number within bounds was 1000/1000
The mean was 48.086, the standard deviation was 29.0664
and the number within bounds was 1000/1000
The mean was 50.135, the standard deviation was 28.4307
and the number within bounds was 1000/1000
The mean was 49.187, the standard deviation was 28.3847
and the number within bounds was 1000/1000
The mean was 50.13, the standard deviation was 29.2683
and the number within bounds was 1000/1000
. . .

```

Why doesn't the program match the expected percentage? The random numbers were generated according to a uniform percentage. The estimate assumes that random numbers come from a normal distribution. The distributions aren't the same.

In fact, the standard deviation for the uniform probability between Θ_1 and Θ_2 is $\frac{(\Theta_2 - \Theta_1)}{\sqrt{12}}$. In our example, the standard deviation should be around 28.8675. $4 \times \sigma > \Theta_2 - \Theta_1$, so it covers the whole of the uniform distribution.

6. I used the same code as above, except that I changed the call to `data[i] = lrand48() coinFlips();`, and I added the new function `coinflips`:

```
int coinFlips(void)
{
    int flips;
    int headsCount;

    headsCount = 0;
    for (flips = 1; flips <= k_maxValue; flips++)
        headsCount += (lrand48() % 2);

    return headsCount;
}
```

The program gives the following output:

```
The mean was 50.048, the standard deviation was 4.98856
and the number within bounds was 954/1000
The mean was 49.928, the standard deviation was 4.96053
and the number within bounds was 953/1000
The mean was 50.042, the standard deviation was 4.96752
and the number within bounds was 947/1000
The mean was 49.934, the standard deviation was 4.95859
and the number within bounds was 937/1000
The mean was 49.9, the standard deviation was 4.97896
and the number within bounds was 957/1000
The mean was 50.129, the standard deviation was 5.02716
and the number within bounds was 949/1000
. . .
```

These results are close to the expected 95%.

1. Here's the source code that I used:

```
int exponential(double probability) {
    int count = 0;

    do {
        count++;
    } while (drand48() >= probability);

    return count;
}

double trimmedMean(double probability, int numElements, double trim)
{
    std::multiset<int> allElements;
    int count;
    int countElements;
    std::multiset<int>::iterator iter;
    int numTrim;
    double sumValues;

    // Generate the elements
    for (count = 0; count < numElements; count++)
    {
        allElements.insert(exponential(probability));
    }

    // Trim the multiset
    numTrim = (int) (numElements * trim);
    if (numTrim > 0)
    {
        for (iter = allElements.begin(), count = 0;
             count < numTrim;
             iter++, count++) {
            // Empty.
        }

        allElements.erase(allElements.begin(), iter);
    }

    for (iter = allElements.end(), count = 0;
         count < numTrim;
         iter--, count++) {
        // Empty.
    }

    allElements.erase(iter, allElements.end());
}
```

```

    }

    // Find the mean of the remaining elements.
    sumValues = 0.0;
    for (iter = allElements.begin(), countElements = 0;
         iter != allElements.end();
         iter++, countElements++) {
        sumValues += *iter;
    }

    return sumValues / countElements;
}

const int k_numExperiments = 10000;

int main(int argc, char *argv[]) {
    int countExperiments;
    double mean;
    double squareOfSum;
    double sum;
    double sumOfSquares;
    double trim;

    srand48((long int) std::time(NULL));

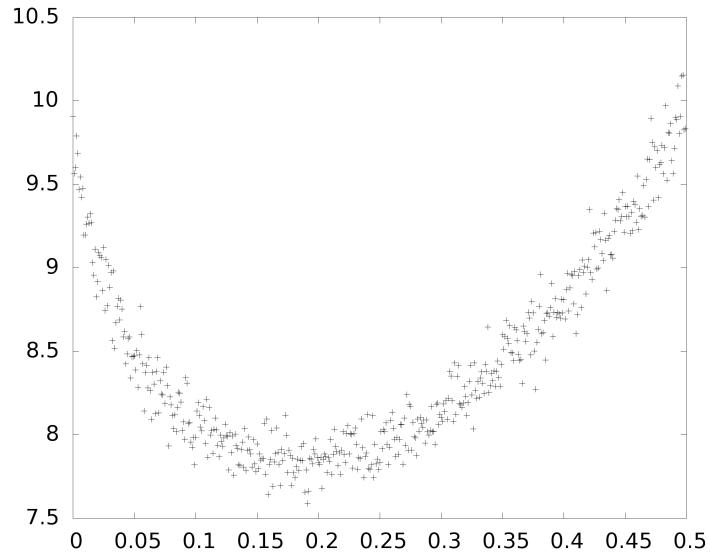
    for (trim = 0.0; trim <= 0.4991; trim += 0.001) {
        sum = 0.0;
        sumOfSquares = 0.0;
        for (countExperiments = 0; countExperiments < k_numExperiments; countExperiments++) {
            mean = trimmedMean(0.01, 1000, trim);
            sum += mean;
            sumOfSquares += mean * mean;
        }
        squareOfSum = sum * sum;

        std::cout << trim << "\t" << (sumOfSquares - (squareOfSum / k_numExperiments)) / (k_numExperiments - 1) << "\n";
    }
}

```

Here's the graph of the results:

Figure A.1: Comparing trimmedMean variances



PAGE 42:

1. Here's the code that I wrote:

```
void computeE()
{
    double x;

    for (x = 1; x + 1 > 1; x /= 2)
        std::cout << x << "\t" << std::pow(1.0+x, 1.0/\
            x) << std::endl;
}
```

PAGE 45:

There are many answers to these questions. I chose these because they're simple.

1. $y = -x^4$
2. $y = x^4$

PAGE 47:

- Let x_n be the current estimate (and x_{n+1} be the next estimate), $f(x)$ be the function, and $f'(x)$ be the derivative (slope) at x .

$f(x_n)$ is related to how far the current estimate is from zero, and $f'(x_n)$ is related to the slope at the current estimate.

The slope is “rise over run” or $\frac{\delta y}{\delta x}$ – the number of units change vertically per one unit horizontally. But $f'(x_n)$ measures how much units change horizontally per one unit vertically. Therefore, $f'(x_n)$ is the inverse of the slope.

Therefore, $\frac{f(x_n)}{f'(x_n)}$ is a good estimate for how far the x axis is incorrect.

PAGE 49:

- The integration function has a limit: it doesn’t handle infinities well. You could use an arbitrarily large negative number as the lower limit, but the more negative the number, the longer the program takes to complete. The two features listed in the problem give a better idea. Let $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

$$\begin{aligned} \text{if } Z > 0 \text{ then } \int_{-\infty}^Z \phi(x) \, dx &= \int_{-\infty}^0 \phi(x) \, dx + \int_0^Z \phi(x) \, dx \\ &= \Phi(0) + \int_0^Z \phi(x) \, dx \\ &= 0.5 + \int_0^Z \phi(x) \, dx \end{aligned}$$

$$\text{if } Z < 0 \text{ then } \Phi(Z) = 1 - \Phi(-Z)$$

The new integral uses no ∞ , so the answer can be more accurate for much less work.

Here’s the code that I wrote:

```
class StandardNormal: public IFunctor
{
public:
    StandardNormal() {}
    virtual ~StandardNormal() {}
}
```

```
    double operator() (double parameter);
};

double StandardNormal::operator() (double parameter)
{
    return std::sqrt((1 / (2*PI))) * std::exp( -\
        parameter*parameter/2);
}

/* A table of the standard normal curve */
const double k_standard_delta = 1E-6;

void standardNormalTable()
{
    double integrated;
    double z;
    Interval *pinterval;

    StandardNormal stdnorm;
    Integrate integStandardNormal(&stdnorm, \
        k_standard_delta);

    std::cout << "Cumulative_normal_Frequency" << std\
        ::endl;

    for (z = -5.0; z < 0; z += 0.01) {
        pinterval = new Interval(z, Closed, 0, Closed) \
            ;
        integrated = integStandardNormal.integrate(*\
            pinterval);
        delete pinterval;

        std::cout << z << "\t" << 0.5 - integrated << \
            std::endl;
    }

    for (z = 0; z < 5.005; z += 1) {
        pinterval = new Interval(0, Closed, z, Closed) \
            ;
        integrated = integStandardNormal.integrate(*\
            pinterval);
        delete pinterval;

        std::cout << z << "\t" << 0.5 + integrated << \
            std::endl;
    }
}
```

```

        std::endl;
    }
}

```

The output starts:

```

Cumulative normal Frequency
-5      2.86508e-07
-4.99   3.01753e-07
-4.98   3.17779e-07
-4.97   3.34622e-07
-4.96   3.52324e-07
-4.95   3.70926e-07
-4.94   3.90472e-07
-4.93   4.11008e-07
-4.92   4.32582e-07
-4.91   4.55243e-07
-4.9    4.79045e-07

```

...

2. The only thing that changes from the previous problem is the `for` loop in `Integrate::integrate(Interval interval)`.

```

integralResult = 0;
for (slice = lower;
      slice < upper;
      slice += m_dDelta)
{
    integralResult += ( ((*m_pifunctor)(slice) + \
                        (*m_pifunctor)(slice + m_dDelta)) / 2) * \
                           m_dDelta;
}

```

For most common functions, this change slightly *decreases* the accuracy of the output. For example, we know that $\int x^2 dx = \frac{x^3}{3}$. If `delta` is 0.1, then the two method's outputs for computing $\int x^2 dx$ give the results:

lower	upper	rectangle	quadrilateral	real
0	1	0.3325	0.335	0.3333333
0	3	8.9975	9.005	9
0	6	71.995	72.01	72
0	9	242.992	243.015	243

where *rectangle* is the usual method, *quadrilateral* is the new method, and *real* is the actual answer.

Most simple functions are either *converse* or *concave* for large areas: that is, their second derivatives are positive or negative over large areas. Therefore, most quadrilaterals tend to be completely under or completely over

its function. These small errors add up, giving a worse overall sum than the simpler rectangle method.

A.2 Probability basics

PAGE 54:

1. There are two conditions to prove:

- (a) Because the s_i are the samples in S , $\bigcup_i \{s_i\} = S$.
- (b) Since the samples are unique, if s_i and s_j are different, then $\{s_i\} \cap \{s_j\} = \emptyset$.

Therefore, the $\{s_i\}$ are a partition.

2. There are two conditions to prove:

- (a) By definition, $\bigcup\{S\} = S$.
- (b) Since only one partition exists, there is no second partition to be different from.

Therefore, $\{S\}$, by itself, is a partition.

PAGE 55:

1. Since the values are equiprobable, each a_i has a probability of $\frac{1}{n}$. The computation follows:

$$\begin{aligned} E[A] &= \sum_{i=1}^n \Pr(a_i) a_i \\ &= \sum_{i=1}^n \frac{1}{n} a_i \\ &= \frac{\sum_{i=1}^n a_i}{n} \\ &= \text{mean} \end{aligned}$$

2. The proof is simple:

$$\begin{aligned} E[f(X) + g(X)] &= \sum_{x \in X} (f(x) + g(x)) \Pr(x) \\ &= \sum_{x \in X} f(x) \Pr(x) + \sum_{x \in X} g(x) \Pr(x) \\ &= E[f(X)] + E[g(X)] \end{aligned}$$

3. The proof is equally simple.

$$\begin{aligned} E[k \times f(X)] &= \sum_{x \in X} kf(x) \Pr(x) \\ &= k \times \sum_{x \in X} f(x) \Pr(x) \\ &= k \times E[X] \end{aligned}$$

4. $E[X]$ is a constant.

5. The proof follows:

$$\begin{aligned} E[(A - E[A])^2] &= E[A^2 - 2AE[A] + E[A]^2] \\ &= E[A^2] - 2E[AE[A]] + E[E[A]^2] \end{aligned}$$

But $E[A]$ doesn't change with A . Therefore, we can continue:

$$\begin{aligned} E[(A - E[A])^2] &= E[A^2] - 2E[A]E[A] + E[A]^2 \\ &= E[A^2] - E[A]^2 \end{aligned}$$

6. When variables are equiprobable, then $\Pr(a_i) = \frac{1}{n}$ for any i , and $E[A] = \bar{A}$. Therefore:

$$\begin{aligned} E[(A - E[A])^2] &= \sum_{i=1}^n \Pr(a_i) (a_i - E[A])^2 \\ &= \sum_{i=1}^n \frac{1}{n} (a_i - \bar{A})^2 \\ &= \sum_{i=1}^n \frac{(a_i - \bar{A})^2}{n} \end{aligned}$$

Let $\alpha_i = (a_i - \bar{A})^2$. Then:

$$\begin{aligned} s^2 &= \sum_{i=1}^n \frac{\alpha_i}{n-1} \\ \sigma^2 &= \sum_{i=1}^n \frac{\alpha_i}{n} \end{aligned}$$

These are both estimates of the true variance. It turns out that s is unbiased (that is, many s s on different data sets from the same distribution would be average to the true variance), but σ tends to be more accurate (that is, for a given set of data, σ is more likely to be close to the true value.)

For now, just use s when you're given data, and use σ when you're given a distribution.

PAGE 57:

1. The chance of selecting the first card and it not be a queen is $\frac{5}{7}$. The chance of the second card is $\frac{4}{6}$. The chance of the third card is $\frac{3}{5}$. Therefore, the overall chance is $\frac{5}{7} \times \frac{4}{6} \times \frac{3}{5} = \frac{2}{7}$.
2. The first trick to solving this problem is to figure out how many ways exist that two people in a room don't share a birthday. This trick was used in the poker example.

A first thought is that, if $n \geq 2$ people are in the room, then there are 365^n possible sets of birthdays and combinations ($365, n$) ways that no two people in the room have the same birthday. (Obviously, order matters in this count.) So, a first program to compute the birthday problem would be:

```
int main(void)
{
    int count;
    double numPermutations;
    double numPossibilities;
    double probability;

    for (count = 1; count < 100; count++) {
        numPermutations = permutations(k_numDaysInYear, \
            count);
        numPossibilities = std::pow((float) \
            k_numDaysInYear, count);

        probability = 1.0 - numPermutations / \
            numPossibilities;

        std::cout << count << "\t" << probability << std\
            ::endl;
    }

    return 0;
}
```

This gives the output:

```

1      0
2      0.00273973
3      0.00820415
4      0.0163559
5      0.0271356
6      0.0404624
7      0.0562357
8      0.0743353
9      0.0946238
10     0.116948
11     0.141141
12     0.167025
13     0.19441
14     0.223103
15     0.252901
16     1
17     1
18     1
19     1
20     1
21     1
...

```

What happened? Simply put, 365^{16} is a larger number than a 32-bit double can hold. The function underflows.

What can be done? Although one could switch to 64-bit numbers, it's better to re-examine the function. With the i -th new person in the group, the probability that no two people share the same birthday is multiplied by $\frac{366-i}{365}$. This insight gives a different, more accurate program:

```

probability = 1.0;
for (count = 1; count < 50; count++) {
    probability *= (366.0-count) / 365.0;
    std::cout << count << "\t" << 1 - probability << \
        std::endl;
}

```

This program finishes the output and gives the answer:

```

15      0.252901
16      0.283604
17      0.315008
18      0.346911
19      0.379119
20      0.411438

```

```

21      0.443688
22      0.475695
23      0.507297
24      0.538344
...

```

The first answer with a probability above 0.5 is 23. Therefore, if you have 23 people in a room, you have a better-than 50% chance that two will share the same birthday.

3. Here's some code that implements the algorithm:

```

int bet(int wager)
{
    if (drand48() * 38.0 < 18.0)
        return wager * 2;
    else
        return 0;
}

bool martingale(int initialBankroll, int maxBankroll,
                 , int minBet)
{
    int bankroll;
    int currentBet;
    int result;

    bankroll = initialBankroll;
    currentBet = minBet;
    while (0 < bankroll && bankroll < maxBankroll) {
        bankroll -= currentBet;
        result = bet(currentBet);
        bankroll += result;

        if (result > 0) {
            currentBet = minBet;
        } else if (bankroll >= currentBet * 2) {
            currentBet *= 2;
        } else {
            currentBet = bankroll; // Our one, last shot.
        }
    }

    return (bankroll >= maxBankroll);
}

```

I did 1,000,000 runs of this strategy using $p = \frac{18}{38}$, the fraction that a person would win in an “even” bet on the roulette wheel. I varied the starting bankroll and the amount to gain. Here are the results I got; your mileage will vary:

	Gain \$10	Gain \$100	Gain \$1000	Gain \$10000
Bankroll \$100	86.66%	34.61%	1.945%	0.0208%
Bankroll \$1000	98.34%	84.93%	27.40%	0.804%
Bankroll \$10,000	99.77%	97.78%	80.48%	20.81%
Bankroll \$100,000	99.97%	99.71%	97.21%	76.17%

4. The code for a minimum-bet gambler is easy:

```
bool minimumBet(int initialBankroll, int maxBankroll,
                 , int minBet)
{
    int bankroll;

    bankroll = initialBankroll;
    while (0 < bankroll && bankroll < maxBankroll) {
        bankroll -= minBet;
        bankroll += bet(minBet);
    }

    return (bankroll >= maxBankroll);
}
```

However, this strategy takes much longer to run. I performed 15,000 runs of this strategy with the same p , varying the starting bankroll and the amount to gain. Here are the results I got:

	Gain \$10	Gain \$100	Gain \$1000	Gain \$10000
Bankroll \$100	35.05%	0.0067%	0%	0%
Bankroll \$1000	35.51%	0%	0%	0%
Bankroll \$10,000	35.58%	0.0067%	0%	0%
Bankroll \$100,000	33.76%	0%	0%	0%

The minimum-bet strategy is a poor way to make even small amounts of money.

5. The code for a maximum bet follows:

```
bool maximum(int initialBankroll, int maxBankroll,
             int minBet)
{
    int bankroll;
    int currentBet;
    int result;

    bankroll = initialBankroll;
```

```

while (0 < bankroll && bankroll < maxBankroll) {
    if (2 * initialBankroll <= maxBankroll)
        currentBet = bankroll;
    else
        currentBet = maxBankroll - bankroll;

    bankroll -= currentBet;
    result = bet(currentBet);
    bankroll += result;
}

return (bankroll >= maxBankroll);
}

```

It runs quickly. I ran it with 10,000,000 runs with the same p . Here are my results:

	Gain \$10	Gain \$100	Gain \$1000	Gain \$10000
Bankroll \$100	92.33%	47.38%	5.03%	0.54%
Bankroll \$1000	98.88%	92.33%	47.36%	5.04%
Bankroll \$10,000	99.83%	98.88%	92.32%	47.38%
Bankroll \$100,000	99.98%	99.84%	98.88%	92.32%

6. The code for the HowStuffWorks gambler is easy to write:

```

bool howstuffworks(int initialBankroll, int maxBankroll, int minBet)
{
    int bankroll = initialBankroll;
    int currentBet = minBet;
    int result;
    int winningRun = 0;

    while (0 < bankroll && bankroll < maxBankroll) {
        bankroll -= currentBet;
        result = bet(currentBet, 0.4736);
        bankroll += result;

        if (result > 0) {
            winningRun++;
            currentBet = minBet * (floor(winningRun / 2) + 1);
        } else {
            currentBet = minBet;
            winningRun = 0;
        }
    }
}

```

```

    }

return (bankroll >= maxBankroll);
}

```

However, this strategy is slow to run. I ran it overnight with 100,000 runs. Here are my results:

	Gain \$10	Gain \$100	Gain \$1000	Gain \$10000
Bankroll \$100	50.061%	0.314%	0.00%	0.00%
Bankroll \$1000	50.159%	0.258%	0.00%	0.00%
Bankroll \$10,000	50.011%	0.302%	0.00%	0.00%
Bankroll \$100,000	50.041%	0.280%	0.00%	0.00%

It's better than the minimum-bet gambler, but not by much.

7. The best among the four gambling strategies is the extremely bold one.

Each time that you place a bet when $p < 0.5$, the expected value of your total bankroll goes down. Therefore, to maximize the value of your bankroll, you should bet each dollar as few times as possible.

8. There are two parts to compute:

- (a) Given two cards, how many ways can we find a pair of cards? There are 13 ranks, and $\binom{4}{2} = 6$ ways to combine two suits. Therefore, there are $13 \times 6 = 78$ ways to get a pair of cards.
- (b) Given that the other two cards are a pair, how many ways can we find three cards that neither match each other nor the pair? One of the 13 ranks is taken by the pair, leaving 12. Just considering ranks, there are $\binom{12}{3} = 220$ ways to combine three ranks that don't match each other. Since there are 4 suits per rank, there are $220 \times 4^3 = 14080$ ways to combine three cards that don't share a rank with each other, or with the already-made pair.

Therefore, in all, there are $78 \times 14080 = 1098240$ hands of poker with exactly one pair.

The probability that a hand has exactly one pair is $\frac{1098240}{2598960} \approx 42.3\%$.

9. Let g be the unknown number of green balls; there are $60 - g$ blue balls.

The chance of winning l_1 is $\frac{30}{90}$. The chance of winning l_2 is $\frac{g}{90}$. The chance of winning l_3 is $\frac{60}{90}$. The chance of winning l_4 is $\frac{90-g}{90}$.

If $g < 30$, then the expected value for l_1 is better than the expected value for l_2 and the expected value for l_4 is better than the expected value for l_3 .

If $g = 30$, then the expected value for each pair of lotteries is the same.

If $g > 30$, then the expected value for l_2 is better than the expected value for l_1 and the expected value for l_3 is better than the expected value for l_4 .

But we do not know whether $g < 30$, $g = 30$, or $g > 30$. If you assume that $\Pr(g < 30) = \Pr(g > 30)$, then there is no reason to prefer any strategy.

PAGE 62:

1. The naïve probability matches the probability rules for discrete sets. Here are the three rules::

- (a) The \Pr function maps a sample A in a sample space S to $\frac{|A|}{|S|}$. Since every element of A is an element of S , and since no sample can have fewer than 0 members, all probabilities of the naïve probability are between 0 and 1.
- (b) There are $|S|$ elements in the sample space, and every element, e has a probability of $\Pr(e) = \frac{1}{|S|}$. Therefore, $\sum_{e \in S} \Pr(e) = |S| \times \frac{1}{|S|} = 1$.
- (c) If $s_1 \subseteq S$, $s_2 \subseteq S$, and $s_1 \cap s_2 = \emptyset$, then $\Pr\{s_1 \cup s_2\} = \frac{|s_1| + |s_2|}{|S|} = \Pr(s_1) + \Pr(s_2)$.

2. A continuous distribution only requires the total area under the probability function to be 1. Therefore, it's quite easy for a function value to be greater than 1. For example, if the probability function only has positive probability for values between 0 and $\frac{1}{2}$, then the average value in that area will be 2.

PAGE 66:

1. If $\Pr(A|B) = \Pr(A)$, then $\frac{\Pr(A \cap B)}{\Pr(B)} = \Pr A$, then $\frac{\Pr(B \cap A)}{\Pr(A)} = \Pr(B)$, so $\Pr(B|A) = \Pr(B)$.
- 2.

$$\begin{aligned}\Pr(A) &= \Pr(A \cap B) + \Pr(A \cap B^c) \\ &= \frac{\Pr(A \cap B) \Pr(B)}{\Pr(B)} + \frac{\Pr(A \cap B^c) \Pr(B^c)}{\Pr(B^c)} \\ &= \Pr(A|B) \Pr(B) + \Pr(A|B^c) \Pr(B^c)\end{aligned}$$

The two are equal only if $\Pr(B) = \Pr(B^c) = 1$, which is impossible.

PAGE 71:

1. $\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A) \times \Pr(B)$
 2. (a) Usually (though not always) dependent.
 (b) Independent.
 (c) Dependent. If someone has relatively small feet, that someone is also more likely to be young. In this situation, the two sets have no causation in either direction, but another factor (age) affects both of them.
 3. It is not true. Consider \emptyset as one set. We know that $\Pr(\emptyset) = 0$. For any other event, B , it's both that $\emptyset \cap B = \emptyset$ and $\Pr(\emptyset \cap B) = \Pr(\emptyset) = 0 = \Pr(\emptyset) \times \Pr(B)$. Therefore, \emptyset and B are both independent and mutually exclusive.
 4. The definition is a good one. Two calculations prove that the definition means the same thing:
 - (a) Prove that if $\Pr(A|B) = \Pr(A)$, then $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$:
 If $\Pr(A|B) = \Pr(A)$, then $\frac{\Pr(A \cap B)}{\Pr(B)} = \Pr(A)$, so $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$.
 - (b) Prove that if $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$, then $\Pr(A|B) = \Pr(A)$:
 If $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$, then $\frac{\Pr(A \cap B)}{\Pr(B)} = \Pr(A)$, so $\Pr(A|B) = \Pr(A)$.
 5. The following table gives the answers:
- | | |
|-------------|-------------|
| Dependent | Independent |
| Independent | Independent |
| Dependent | Independent |

PAGE 73:

1. Here's the code that I wrote:

```
const int k_iterations = 1000;

int main(int argc, char *argv[])
{
    int countAlternateRight = 0;
    int countChoiceRight = 0;
    int choice; // Which door did the contestant choose?
```

```

int door; // Which door holds the car?
int iter;
int opened; // Which door is opened?

for (iter = 0; iter < k_iterations; iter++) {
    // Since we're in C++, we number the doors \
    0, 1, and 2.
    door = lrand48() % 3;
    choice = lrand48() % 3;
    do
        opened = lrand48() % 3;
    while (opened != door && opened != choice);

    if (choice == door)
        countChoiceRight++;
    else
        countAlternateRight++;
}

std::cout << "The_choice_was_right_" << \
    countChoiceRight << "_times." << std::endl;
std::cout << "The_alternate_was_right_" << \
    countAlternateRight << "_times." << std::endl\
;
}

```

2. When the contestant chooses, there are two possibilities:

- There is a $\frac{2}{3}$ chance that the contestant will first choose the a door hiding a goat. The host will always reveal the other goat. In this case, if the contestant switches, it is certain that the car will be revealed.
- There is a $\frac{1}{3}$ chance that the contestant will first choose the door hiding the car. The host will choose either door, revealing a goat. In this case, if the contestant switches, it is certain that the other goat will be revealed.

Therefore, the chance of getting the car if the contestant switches is $\frac{2}{3} \times 1 + \frac{1}{3} \times 0 = \frac{2}{3}$

3. This problem assumes that Bob did not know that he was choosing an empty envelope. The Monty Hall problem assumes that Monty knew where the car was. This difference changes the problem in an important way: if we were doing this as an experiment, we would have to throw out $\frac{1}{3}$ of all trial runs, because Bob would have gotten the envelope with the chocolate.

Each envelope has a $\frac{1}{3}$ chance of holding the chocolate coupon. Bob had no previous knowledge – he could not cause himself to open an empty envelope.

Because we don't have someone with perfect knowledge changing the probabilities, we can use the conditional probability formula. Let C be that you have chocolate, that A that Alice has chocolate, and B that Bob has chocolate. Then $\Pr(C \cap B^c) = 1/3$, since if you have chocolate, then Bob has none unless you share. And $\Pr(B^c) = 2/3$. Therefore:

$$\begin{aligned}\Pr(C|B^c) &= \frac{\Pr(C \cap B^c)}{\Pr(B^c)} \\ &= \frac{1/3}{2/3} \\ &= \frac{1}{2}\end{aligned}$$

PAGE 74:

1. If the chance of choosing an individual amino acid is $\frac{1}{20}$, or 5×10^{-2} , then the chance of choosing one hundred amino acids is $(5 \times 10^{-2})^{100}$. This number is approximately 7.8886×10^{-131} .
2. Here's the sourcecode that I wrote:

```
#define EXPERIMENTS 1000000L
#define COINS 100
#define MAXSIZE 50

int main(int argc, char *argv[])
{
    int ariCoins[COINS];
    int ariRepeats[MAXSIZE];
    int bContinue;
    int i;
    int iRepeats;
    long int liExperiment;

    srand48((long int) time(NULL));

    for (i=0; i<MAXSIZE; i++) {
        ariRepeats[i] = 0;
    }
}
```

```

for (liExperiment = 0; liExperiment < EXPERIMENTS; \
liExperiment++) {

    for (i=0; i<COINS; i++) {
        ariCoins[i] = 0;
    }

    iRepeats = 0;
    do {
        bContinue = 0; /* FALSE */
        iRepeats++;

        for (i=0; i<COINS; i++) {
            if (ariCoins[i] == 0) {
                bContinue = 1; /* TRUE */
                ariCoins[i] = rand()/16 % 2;
            }
        }
    }
    while (bContinue);

    ariRepeats[iRepeats]++;
}

for (i=0; i<MAXSIZE; i++) {
    printf("%d\t\t\t\t\t%d\n", i, ariRepeats[i]);
}

return 0;
}

```

PAGE 77:

1. We don't need Bayes' formula. The problem says that only one person in a million would match the DNA. Since there are 10,000,000,000 in this world, there are 10,000 suspects in the world, not just one. Since we have no other proof that this person is guilty, we can't hold him.
2. This problem requires Bayes' formula. Let $\Pr(S)$ represent the probability that the email is spam, let $\Pr(S|!!!)$ be the probability that the email is spam, given the row of !!!'s.

$$\begin{aligned}\Pr(S|!!!) &= \frac{\Pr(!!!|S)\Pr(S)}{\Pr(!!!|S)\Pr(S) + \Pr(!!!|S^c)\Pr(S^c)} \\ &= \frac{0.70 \times 0.60}{0.70 \times 0.60 + 0.10 \times 0.40} \\ &\approx 91.3\%\end{aligned}$$

3. Again, we don't need Bayes' formula. We would inconvenience just 4% of travellers. 4% of 622 million passengers is 24 million passengers. There were 19 terrorists. Therefore, the chance that a person is a terrorist, given that the passenger failed the test, is 19 in 24 million – still less than 1 in a million.
4. This problem remains far too easy.

A.3 Distributions

PAGE 83:

1. $f(x) = F(x) - F(x - 1)$
 2. The Kolmogorov rules have three items to prove.
 - (a) $f(-\infty) = f(\infty) = 0$
I said that the range of the discrete distribution does not include the infinities.
 - (b) For all x , $0 \leq f(x) \leq 1$
According to the first Kolmogorov rule, \Pr maps from S to $[0, 1]$.
 - (c) $\sum_{x=-\infty}^{\infty} f(a) = 1$
According to the second Kolmogorov rule, $\sum_{s_i \in S} \Pr(s_i) = 1$.
 3. $f(x) = \frac{dF(x)}{dx}$
-

PAGE 92:

1. First, RandomBase.h needs the following member, preferably as a private member:

```
mutable std::vector<double> m_results;
```

(**mutable** means that the variable can change even in a **const** method like **cumulative**.)

Second, use the following implementation of **cumulative**:

```
template <>
double RandomBase<int, double>::cumulative(int x) \
const
{
    int negInfinity;
    int xCurrent;
    double ySum;

    negInfinity = negativeInfinity();

    if (m_results.size() == 0) {
        ySum = 0.0;
    }
    else {
        int i;
        double sum = 0.0;
        for (i = 0; i < m_results.size(); i++) {
            sum += m_results[i];
            if (x <= i)
                break;
        }
        ySum = sum;
    }
}
```

```

for (xCurrent = negInfinity; xCurrent <= \
    positiveInfinity(); xCurrent++) {
    ySum += density(xCurrent);
    m_results.push_back(ySum);
}
}

if (x <= negInfinity)
    return 0.0;
else if (x <= positiveInfinity())
    return m_results[x - negInfinity - 1];
else
    return 1.0;
}

```

PAGE 93:

1. The mean is $0 \times (1 - p) + 1 \times p = p$.
2. The standard deviation is

$$(0 - p)^2 \times (1 - p) + (1 - p)^2 \times p = (p^2 - p^3) + (p - 2p^2 + p^3) \\ = p - p^2$$

PAGE 99:

1. Notice that finding love in her third marriage means that she did not find love in her first two marriages, but did find it in her third marriage. The chance is $\text{binom}(2, 0.1, 0) * \text{binom}(1, 0.1, 1) = 0.81 * 0.1 = 0.081$. The answer $\text{binom}(3, 0.1, 1)$ is the probability that she finds true love exactly once in her first three marriages, which isn't the problem.
 2. There are three possibilities: that he sold no duds, that he sold one dud, or that he sold two duds. Therefore, the chance is $\text{binom}(10, 0.25, 0) + \text{binom}(10, 0.25, 1) + \text{binom}(10, 0.25, 2) \approx .0563 + .1877 + .2816 = .5256$. Notice that you had to take the sum of several binomials.
-

PAGE 100:

1. The cumulative distribution formula is $F(x) = \sum_{y=-\infty}^x f(y)$. Compute:

$$\begin{aligned}
 F(x) &= \sum_{y=-\infty}^x f(y) \\
 &= \sum_{y=1}^x p(1-p)^{y-1} \\
 &= p \sum_{y=1}^x (1-p)^{y-1} \\
 &= p \sum_{y=0}^{x-1} (1-p)^y \\
 &= p \frac{1 - (1-p)^x}{p} \\
 &= 1 - (1-p)^x
 \end{aligned}$$

2. Use the same parameter, p . The support, x , is the number of trials before the first success.

Parameters	$0 \leq p \leq 1$ is the probability (real)
Support	$0 \leq x$ are the number of trials before first success
Density	$f(x; p) = (1-p)^x p$
Mean	$\frac{1-p}{p} - 1$
Standard Deviation	$\sqrt{\frac{1-p}{p}}$

Properties A.1: Properties of the alternate geometric distribution

PAGE 105:

- The geometric distribution is the negative binomial distribution, when $n = 1$.
- Here's the source code that I used:

```

const int NUM_DEFEATS=3;
const int MAX_ATTEMPTS=50;
const double SUCCESS_PROBABILITY=0.2;

int main(int argc, char* argv[])
{
    int attempt;
  
```

```

    std::cout << "Successes\tProbability" << std::endl \
;

for (attempt=NUM_DEFEATS; attempt<=MAX_ATTEMPTS; \
attempt++) {
    std::cout << attempt << "\t" <<
    negative_binomial_probability(attempt, \
        SUCCESS_PROBABILITY, NUM_DEFEATS) <<
    std::endl;
}

return 0;
}

```

3. The geometric distribution happens when there is just one success. Therefore, it is $\text{negbin}(n, p, 1)$, which is

$$\begin{aligned}
\text{negbin}(n, p, 1) &= \binom{n-1}{1-1} p^1 (1-p)^{n-1} \\
&= \frac{n-1!}{0!(n-1)!} p (1-p)^{n-1} \\
&= p(1-p)^{n-1}
\end{aligned}$$

PAGE 116:

1. First, realize that $p = \frac{n}{\lambda}$. Do some calculations:

$$\begin{aligned}
\lim_{n \rightarrow \infty} \text{binom}\left(x; n, \frac{n}{\lambda}\right) &= \lim_{n \rightarrow \infty} \binom{n}{x} \frac{\lambda^x}{n^x} \left(1 - \frac{n}{\lambda}\right)^{n-x} \\
&= \lim_{n \rightarrow \infty} \frac{n! \lambda^x}{x! (n-x)! n^x} \left(1 - \frac{n}{\lambda}\right)^{n-x}
\end{aligned}$$

According to Calculus, $\lim_{n \rightarrow \infty} \left(1 - \frac{\lambda}{n}\right)^n = e^{-\lambda}$. Since x is held constant while n increases, we can use the substitution.

$$\lim_{n \rightarrow \infty} \text{binom}\left(x; n, \frac{n}{\lambda}\right) = \lim_{n \rightarrow \infty} \frac{n! \lambda^x e^{-\lambda}}{x! (n-x)! n^x}$$

It remains to prove that $\lim_{n \rightarrow \infty} \frac{n!}{(n-x)! n^x} = 1$.

$$\lim_{n \rightarrow \infty} \frac{n!}{n^x (n-x)!} = \lim_{n \rightarrow \infty} \left(\frac{n}{n}\right) \left(\frac{n-1}{n}\right) \cdots \left(\frac{n-x+1}{n}\right) \times \left(\frac{(n-x)!}{(n-x)!}\right)$$

As $n \rightarrow \infty$, the first $n - x$ elements each become closer to 1.

Therefore, $\lim_{n \rightarrow \infty} \text{binom}(x; n, p) = \text{poisson}(x; \lambda)$.

2. Here's a table:

<i>n</i>	<i>p</i>	<i>x</i>	Binomial	Poisson
10	0.2	1	0.2684	0.2707
10	0.3	1	0.1211	0.1494
10	0.4	2	0.1209	0.1465
10	0.5	2	0.0439	0.0842
20	0.2	2	0.1369	0.1465
20	0.3	3	0.0716	0.0892
20	0.4	4	0.0350	0.0573
20	0.5	5	0.0148	0.0378
50	0.2	5	0.0295	0.0378
50	0.3	7	0.00477	0.0104
50	0.4	10	0.00144	0.00582
50	0.5	12	0.000108	0.00173
100	0.2	10	0.00336	0.00581
100	0.3	15	0.000248	0.00103
100	0.4	20	0.0000105	0.000192
100	0.5	25	0.000000191	0.0000371
200	0.2	20	0.0000609	0.000192
200	0.3	30	0.000000391	0.00000730
200	0.4	40	0.00000000791	0.000000294
200	0.5	50	0.0000000000282	0.0000000122

3. A direct translation of the text would create a function like:

```
int Poisson::random() const {
    int count;
    double compare;
    double sum;
    Exponential exp;

    compare = m_rate;
    count = 0;
    sum = exp.random();

    while (sum < compare) {
```

```

    sum += exp.random();
    count++;
}
return count;
}

```

For both `compare` and `sum`, use the function $y = 1 - e^{-x}$.

PAGE 127:

1. Let's compute:

$$\begin{aligned}
 \Pr(X > a + b | X > a) &= \frac{\Pr(X > a + b) \cap \Pr(X > a)}{\Pr(X > a)} \\
 &= \frac{\Pr(X > a + b)}{\Pr(X > a)} \\
 &= \frac{1 - \Pr(X \leq a + b)}{1 - \Pr(X \leq a)} \\
 &= \frac{1 - (1 - e^{-(a+b)/\lambda})}{1 - (1 - e^{-a/\lambda})} \\
 &= \frac{e^{-(a+b)/\lambda}}{e^{-a/\lambda}} \\
 &= e^{b/\lambda}
 \end{aligned}$$

2. A Poisson process has $\frac{\lambda^x e^{-\lambda}}{x!}$ chance of x occurrences in one unit of time. If we want the chance of x occurrences in t time, the formula becomes $\frac{(\lambda t)^x e^{-\lambda t}}{x!}$.

Therefore, the chance that no occurrences happen in time t is $\frac{(\lambda t)^0 e^{-\lambda t}}{0!} = e^{-\lambda t}$. The opposite — the chance that one or more occurrences happen in time t — is $1 - e^{-\lambda t}$. This function is cumulative; it contains any number of occurrences. To find the probability distribution function, differentiate with respect to t , which is $\lambda e^{-\lambda t}$. That function is identical to $\text{Exponential}(\frac{1}{\lambda})$.

PAGE 131:

1. Let m be the mean and s be the standard deviation. Then:

$$\begin{aligned}
 m &= \frac{1}{2}\theta_1 + \frac{1}{2}\theta_2 \\
 2m &= \theta_1 + \theta_2 \\
 s &= \frac{1}{2\sqrt{3}}\theta_2 - \frac{1}{2\sqrt{3}}\theta_2 \\
 2\sqrt{3}s &= \theta_2 - \theta_1 \\
 2m + 2\sqrt{3}s &= 2\theta_2 \\
 m + \sqrt{3}s &= \theta_2 \\
 m - \sqrt{3}s &= \theta_1
 \end{aligned}$$

PAGE 134:

1. Here's the source that I used:

```

Normal norm(1.03, 0.06);

const int k_numExperiments = 10000000;

void meanAndStdDev(double sum, double sumOfSquare) {
    double mean;
    double standardDev;

    mean = sum / k_numExperiments;
    standardDev = sqrt( (sumOfSquare - mean * mean * \
        k_numExperiments) / k_numExperiments );

    std::cout << "Mean = " << mean << " Standard Dev = "
        << standardDev << std::endl;
}

int main(int argc, char* argv[]) {
    double dBankroll;
    int i;
    double sum;
    double sumOfSquare;
    int year;
}

```

```

// Invest $10000 at the start.

sum = 0.0;
sumOfSquare = 0.0;
for (i = 0; i < k_numExperiments; i++) {
    dBankroll = 10000.0;
    for (year = 1; year <= 10; year++)
        dBankroll *= norm.random();

    sum += dBankroll;
    sumOfSquare += dBankroll * dBankroll;
}

std::cout << "Investing $10000 at the start gets: "
           << std::endl;
meanAndStdDev(sum, sumOfSquare);

// Invest $1000 per year.

sum = 0.0;
sumOfSquare = 0.0;
for (i = 0; i < k_numExperiments; i++) {
    dBankroll = 0.0;
    for (year = 1; year <= 10; year++) {
        dBankroll += 1000.0;
        dBankroll *= norm.random();
    }

    sum += dBankroll;
    sumOfSquare += dBankroll * dBankroll;
}

std::cout << "Investing $1000 per year gets: "
           << std::endl;
meanAndStdDev(sum, sumOfSquare);

return 0;
}

```

I performed 10,000,000 runs.

Investing \$10,000 at the start averaged \$13,439.50 at the end of 10 years, with a standard deviation of \$2,494.67.

Investing \$1,000 at the start of every year for 10 years averaged \$11,807.20 at the end of 10 years, with a standard deviation of \$1,397.94.

2. The “polar form” is sometimes better because it uses no cos or sin functions.

`v1` and `v2` represent two uniform numbers within a unit circle, where `w` is the square of the distance from (0,0). The `while` statement ensures that `w` is always between 0 and 1.

To simplify the answer, this examination assumes that the standard deviation is 1 and the mean is 0.

Let r_1 and r_2 represent the returned values.

The two values returned by the code are:

$$\begin{aligned} r_1 &= yv_1 \\ &= v_1 \sqrt{\frac{-2 \log(w)}{w}} \\ &= \frac{v_1 \sqrt{-2 \log(w)}}{\sqrt{w}} \end{aligned}$$

$$\begin{aligned} r_2 &= yv_2 \\ &= v_2 \sqrt{\frac{-2 \log(w)}{w}} \\ &= \frac{v_2 \sqrt{-2 \log(w)}}{\sqrt{w}} \end{aligned}$$

It's obvious that $\sqrt{-2 \log(w)}$ acts the same as $\sqrt{-2 \log(e)}$.

The pair $(\frac{v_1}{\sqrt{w}}, \frac{v_2}{\sqrt{w}})$ defines a random point on a unit circle. An alternate way to write a random point on the unit circle is $(\cos(2\pi u_2), \sin(2\pi u_2))$.

PAGE 136:

1. The code should be obvious. Here are the results I got:

x	$\Pr_{bin}(X = x)$	$\Pr_{norm}(X = x)$	$\Pr_{cum}(X = x)$
0	9.31323e-10	4.45617e-08	5.21469e-08
1	2.79397e-08	3.08034e-07	3.53058e-07
2	4.05125e-07	1.8635e-06	2.09485e-06
3	3.78117e-06	9.86626e-06	1.08933e-05
4	2.55229e-05	4.57163e-05	4.96448e-05
5	0.000132719	0.000185389	0.000198289
6	0.000552996	0.000657944	0.000694121
7	0.00189599	0.00204357	0.00212955
8	0.00545096	0.005555	0.00572603
9	0.0133246	0.0132152	0.0134938
10	0.0279816	0.0275141	0.0278693
11	0.0508756	0.050134	0.0504473
12	0.0805531	0.0799471	0.0800338
13	0.111535	0.111575	0.111286
14	0.135435	0.136278	0.135625
15	0.144464	0.145673	0.144868
16	0.135435	0.136278	0.135625
17	0.111535	0.111575	0.111286
18	0.0805531	0.0799471	0.0800338
19	0.0508756	0.050134	0.0504473
20	0.0279816	0.0275141	0.0278693
21	0.0133246	0.0132152	0.0134938
22	0.00545096	0.005555	0.00572603
23	0.00189599	0.00204357	0.00212955
24	0.000552996	0.000657944	0.000694121
25	0.000132719	0.000185389	0.000198289
26	2.55229e-05	4.57163e-05	4.96448e-05
27	3.78117e-06	9.86626e-06	1.08933e-05
28	4.05125e-07	1.8635e-06	2.09485e-06
29	2.79397e-08	3.08034e-07	3.53058e-07
30	9.31323e-10	4.45617e-08	5.21469e-08

2. The file remains correctly readable as long as two out of three corresponding bits remain unflipped. First, assume that $l = 1$: you have one bit to preserve. How long would you be 95% certain of this one bit?

```
int main(int argc, char *argv[]) {
    NegativeBinomial nbinom(2, 1e-06);

    std::cout << nbinom.quantile(0.05) << std::endl;
}
```

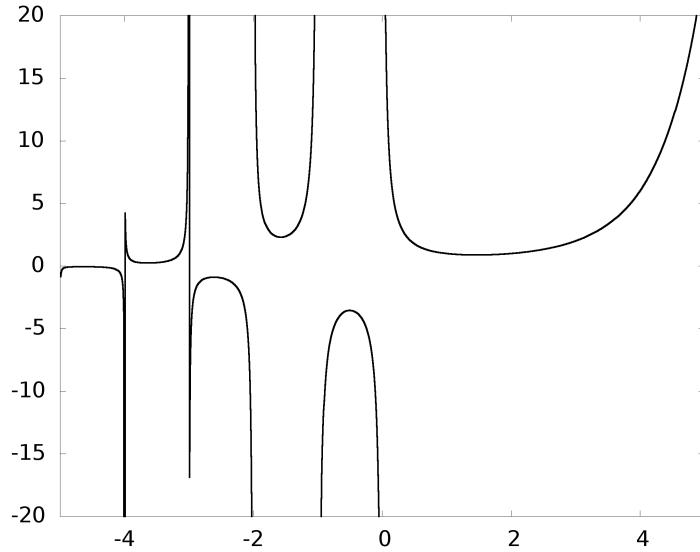
This function returns 355363, the number of days before any particular bit is at least 5% likely to be flipped (that is, have two of its three bits hit). Since every bit is independent of every other bit, the time until the document is unreadable averages $\frac{l}{355363}$. I leave to you to find the distribution

of this circumstance.

PAGE 140:

1. Here's the full graph:

Figure A.2: The gamma function from -5 to 5



PAGE 143:

1. In the top graph, the tallest graph has shape 0.5. In the bottom graph, the shortest graph has scale 0.5.
2. Compute from the properties:

$$\begin{aligned} m &= \lambda\theta \\ s &= \sqrt{\lambda\theta^2} \\ s &= \theta\sqrt{\lambda} \\ \lambda &= \frac{m^2}{s^2} \\ \theta &= \frac{s^2}{m} \end{aligned}$$

PAGE 143:

1. The easiest way to make a chi-squared distribution with parameter ν is
Gamma chisq(nu/2, 2, true);.
2. Here's the table that wasn't at the head of the chapter:

Parameters	$\nu > 0$ is the number of degrees of freedom
Support	$x > 0$ is the position
Density	$f(x; \nu) = \frac{x^{\frac{\nu}{2}-1} e^{-x/2}}{\Gamma(\frac{\nu}{2}) 2^{\frac{\nu}{2}}}$
Mean	ν
Variance	2ν

Properties A.2: Properties of the chi-squared distribution

PAGE 146:

1. The mean is equal to the median in symmetric distributions. The binomial, hypergeometric, normal, and uniform distributions are symmetric.
2. (a) A newborn baby's weight can never be negative, and it is a continuous value. The best distribution would be a gamma distribution. Give yourself partial credit if you chose the normal distribution.
(b) A golf ball can never travel a negative distance, and it is a continuous value. The best distribution would be a gamma distribution.
(c) The binomial distribution.

A.4 Introduction to Statistical Tests

PAGE 149:

1. **\$290**: [Wright, 2004, page 145] *New York Times Guide* did not have the year, but according to <http://www.britannica.com/eb/article-9053139/Model-T>, the price of the touring car was lowered from \$850 in 1908 to less than \$300 in 1925.
2. **8,500 shares**: [Wright, 2004, page 165]
3. **14,000,000 square kilometers**. [Wright, 2004, page 173]
4. **5,989 km**: [Wright, 2004, page 198]
5. **1361 B.C.**: [Wright, 2004, page 214]
6. **118 years**: (264 B.C. - 146 B.C.) [Wright, 2004, page 253]
7. **1728 A.D.**: [Wright, 2004, page 347]
8. **0.1 kilowatts (100 watts)**: [Wright, 2004, page 405]
9. **16,585,000 deaths**. They were 29.3% of all deaths worldwide. [Wright, 2004, page 443].
10. **9 hours, 56 minutes**. [Wright, 2004, page 538]
11. **39.62×10^9 or 39,620,000,000 kilograms**. [Wright, 2004, page 579]
12. **65.8 million years ago**. [Wright, 2004, page 596]
13. **1,382 games**. [Wright, 2004, page 680]
14. **10,651 athletes**: 6,582 men and 4,069 women. [Wright, 2004, page 749]
15. **74.602 MPH**. [Wright, 2004, page 766]

PAGE 153:

1. Any range from `d.quantile(a)` to `d.quantile(b)` where $b - a = c$ fits this requirement.

PAGE 156:

1. A Type I error happens when we reject the null hypothesis — assume that the coin is not fair — when the coin is fair.

A Type II error happens when we accept the null hypothesis — assume the coin is fair — when the coin is weighted.

As we raise d , we make it more likely that we accept the null hypothesis even when it's not true. Therefore, we are lowering the chance of Type I errors, and raising the chance of Type II errors.

2. No. It's possible to increase the specificity of a test AND lower the probability of type II errors at the same time. A very common way to do this is by gathering more data.

PAGE 158:

1. In this case, the sensitivity plus the specificity is less than 100%. According to the definitions on page 155,

$$\Pr(H_0|E) + \Pr(H_0^c|E^c) < 1$$

Since $\Pr(H_0|E) + \Pr(H_0^c|E) = 1$ and $\Pr(H_0|E^c) + \Pr(H_0^c|E^c) = 1$, it follows that

$$\Pr(H_0^c|E) + \Pr(H_0|E^c) > 1.$$

Using the opposite of the test will create a new test above the diagonal line.

2. If the deck were five red cards and four black cards, then the probability that Abigail drew six red cards is

$${6 \choose 0} \left(\frac{5}{9}\right)^6 \left(\frac{4}{9}\right)^0 = \left(\frac{5}{9}\right)^6$$

If the deck were four red cards and five black cards, then the probability that Abigail drew six red cards is

$${6 \choose 0} \left(\frac{5}{9}\right)^0 \left(\frac{4}{9}\right)^6 = \left(\frac{4}{9}\right)^6$$

Using Bayes' Theorem, the chance that Abigail was correct about her deck is

$$\begin{aligned} \Pr(D = r | \langle rrrrrr \rangle) &= \frac{\Pr(\langle rrrrrr \rangle | D = r) \Pr(D = r)}{\Pr(\langle rrrrrr \rangle | D = r) \Pr(D = r) + \Pr(\langle rrrrrr \rangle | D = b) \Pr(D = b)} \\ &= \frac{\Pr(\langle rrrrrr \rangle | D = r)}{\Pr(\langle rrrrrr \rangle | D = r) + \Pr(\langle rrrrrr \rangle | D = b)} \\ &= \frac{(5/9)^6}{(5/9)^6 + (4/9)^6} \\ &\approx .7923 \end{aligned}$$

If the deck were five red cards and four black cards, then the probability that Becky drew 303 red cards and 297 black cards is

$$\binom{600}{303} \left(\frac{5}{9}\right)^{303} \left(\frac{4}{9}\right)^{297}$$

If the deck were four red cards and five black cards, then the probability that Becky drew 303 red cards and 297 black cards is

$$\binom{600}{303} \left(\frac{5}{9}\right)^{297} \left(\frac{4}{9}\right)^{303}$$

Reusing Bayes' Theorem, we find:

$$\begin{aligned} \frac{\Pr(<rrrrrr> | D = r)}{\Pr(<rrrrrr> | D = r) + \Pr(<rrrrrr> | D = b)} &= \frac{\binom{600}{303} \left(\frac{5}{9}\right)^{303} \left(\frac{4}{9}\right)^{297}}{\binom{600}{303} \left(\frac{5}{9}\right)^{303} \left(\frac{4}{9}\right)^{297} + \binom{600}{303} \left(\frac{5}{9}\right)^{297} \left(\frac{4}{9}\right)^{303}} \\ &= \frac{(5/9)^6}{(5/9)^6 + (4/9)^6} \\ &\approx .7923 \end{aligned}$$

The two probabilities are exactly the same!

PAGE 163:

1. The likelihood almost follows the Kolmogorov Rules for continuous sets, except that the sum of likelihoods is usually not one.

Rescale it so that the sum of likelihoods is defined to be one: let $\hat{L}(\lambda) = \frac{L[\lambda|x]}{\int_{-\infty}^{\infty} L[\lambda|x] dx}$.

If λ can take only a finite number of elements, then we only need to prove that $\hat{L}(S_1 \cup S_2) = \hat{L}(S_1) + \hat{L}(S_2)$. Let's compute:

$$\begin{aligned} \hat{L}(S_1 \cup S_2) &= \frac{L[S_1 \cup S_2|x]}{\int_{-\infty}^{\infty} L[\lambda|x] dx} \\ &= \frac{\Pr(x|S_1 \cup S_2)}{\int_{-\infty}^{\infty} \Pr(x|\lambda) dx} \\ &= \frac{\Pr(x|S_1) + \Pr(x|S_2)}{\int_{-\infty}^{\infty} \Pr(x|\lambda) dx} \\ &= \frac{L(S_1|x) + L(S_2|x)}{\int_{-\infty}^{\infty} L[\lambda|x] dx} \\ &= \hat{L}(S_1) + \hat{L}(S_2) \end{aligned}$$

A.5 Statistical Tests for one variable

PAGE 169:

1. A bootstrapped sample almost always contains a smaller set of different values than the original set. This lowers the amount of variation in the bootstrapped sample.
2. Here's the source code that I used:

```
Bootstrap::Bootstrap(int numElements) {
    m_numElements = numElements;
}

Bootstrap::~Bootstrap() {
    // This method intentionally left blank.
}

// Run a bootstrap with m_numElements elements.
int Bootstrap::runBootstrap() {
    int countTrue;
    int i;
    int posRand;
    std::vector<bool> veboVisited(m_numElements)`;
    ;

    veboVisited.assign(m_numElements, false);

    for (i=0; i<m_numElements; i++) {
        posRand = (int) (drand48() * `m_numElements);
        veboVisited[posRand] = true;
    }

    countTrue = count(veboVisited.begin(), `veboVisited.end(), true);

    return countTrue;
}

int main(int argc, char* argv[]) {
    Bootstrap *pboot;
    int countExperiments;
    int countTrue;
```

```
int numElements;

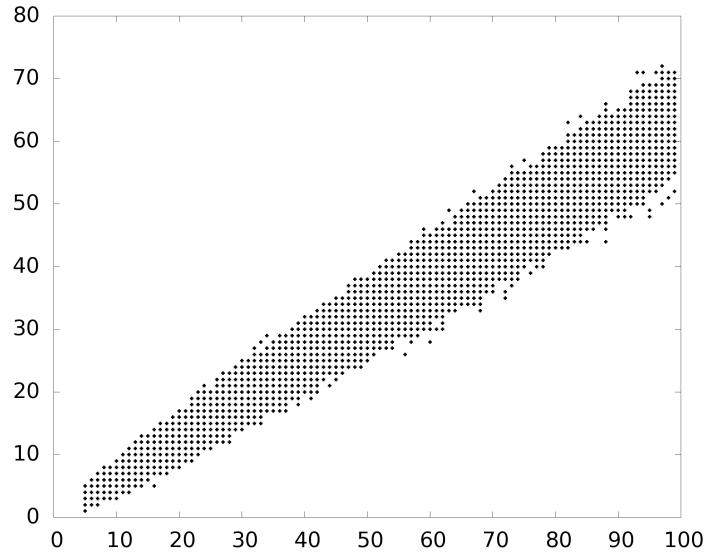
for (numElements = 5; numElements < 100; \
     numElements++) {
    pboot = new Bootstrap(numElements);

    for (countExperiments = 0; \
         countExperiments < 1000;
         countExperiments++) {
        countTrue = pboot-> \
                    runBootstrap();
        std::cout << numElements << \
                    "    " << countTrue <<
        std::endl;
    }

    delete pboot;
}
}
```

The final graph is figure A.3.

Figure A.3: Number of Unique elements in a bootstrap



1. 42.07%.
2. The distribution for a z-test is $\text{Normal}(\text{meanX}, \text{sX}/\sqrt{n})$. We are only concerned about the standard deviation for this distribution. Solve $1 = \frac{10}{\sqrt{n}}$ for n : $n = 100$.
3. The easiest way is to have a private Normal variable in your class:

```
class ZTest : public Distribution<double, double>
{
private:
    Normal *m_pZDist;

public:
    ZTest(const DataDistribution &datadist);
    virtual ~ZTest() {}

    virtual probability density(double x) const;
    . . .
}
```

In the constructor, fill in `m_pZDist`:

```
ZTest::ZTest(const DataDistribution &datadist) {
    double mean;
    double size;
    double stddev;

    mean = datadist.mean();
    size = datadist.size();
    stddev = datadist.standardDeviation();

    m_pzDist = new Normal(mean, stddev / size);
}
```

Finally, each method should just call the equivalent method in `m_pZDist`:

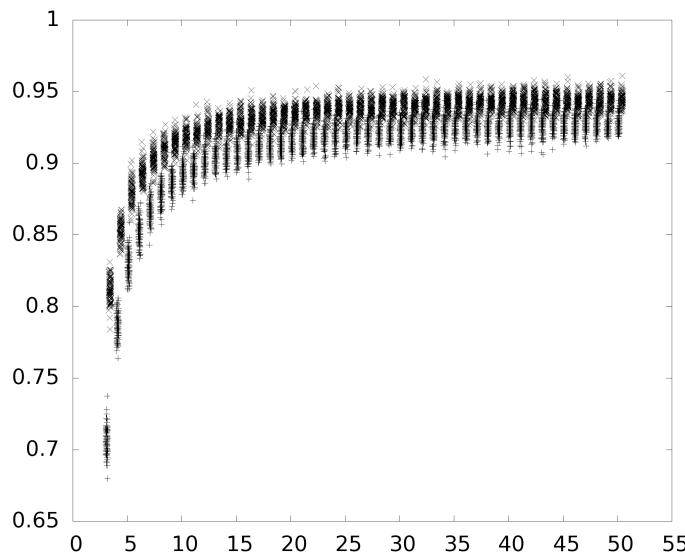
```
probability ZTest::density(double x) const {
    return m_pZDist->density(x);
}

probability ZTest::cumulative(double x) const {
    return m_pZDist->cumulative(x);
}

. . .
```

4. In the following chart, '+' represents the bootstrap, and 'x' represents the Z-Test.

Figure A.4: Comparison between Z Test and Bootstrap



Notice that, although the Z-Test is better than bootstrap tests, their 95% certainty is still not wide enough. The T-Test overcomes this problem.

5. The most common way is called **TOST**, or Two One-Sided Tests. First, **TOST** choose some δ that represents how close to 0 the tests are allowed to go. Then, run two one-sided tests:
 - (a) Where H_0 is that $\mu \leq -\delta$
 - (b) Where H_0 is that $\mu \geq \delta$

The test as a whole fails the null hypothesis when both tests fail their null hypotheses.

PAGE 178:

1. Here's the source code that I used:

```
StudentT::StudentT(double mean, double \
    standardDeviation, double degreesOfFreedom)
{
    m_mean = mean;
```

```

if (0 < standardDeviation)
    m_standardDeviation = standardDeviation;
else // standardDeviation <= 0
    throw std::out_of_range("Error: the standard\
                            deviation must be strictly
positive.");

if (0 < degreesOfFreedom)
    m_degreesOfFreedom = degreesOfFreedom;
else // degreesOfFreedom <= 0
    throw std::out_of_range("Error: the degrees\
                            of freedom must be strictly
positive.");
}

double StudentT::density(double x) const
{
    double numerator;
    double denominator;

    double fixedX;

    // This method is not very accurate for very \
    // small or very large
    // values of x.

    if (x <= negativeInfinity() || x >= \
        positiveInfinity())
        return 0.0;

    fixedX = (x - m_mean) / m_standardDeviation;

    numerator = m_gf((m_degreesOfFreedom + 1) / 2) *
        std::pow(1 + (fixedX * fixedX / \
                      m_degreesOfFreedom), -(m_degreesOfFreedom
+ 1) / 2);
    denominator = std::sqrt(m_degreesOfFreedom * PI) \
        *
        m_gf(m_degreesOfFreedom / 2);

    return numerator / denominator / \
        m_standardDeviation;
}

// Speeding up computation...

```

```

double StudentT::negativeInfinity() const
{
    return -10.0 * m_standardDeviation + m_mean;
}

double StudentT::positiveInfinity() const
{
    return 10.0 * m_standardDeviation + m_mean;
}

```

2. The source code is just like the one for ZTest, but with a different constructor:

```

class TTest : public Distribution<double, double>
{
private:
    StudentT *m_pTDist;

public:
    TTest(const DataDistribution &datadist);
    virtual ~TTest() {}

    virtual probability density(double x) const;
    virtual probability cumulative(double x) const;
    . . .
};

TTest::TTest(const DataDistribution &datadist) {
    double mean;
    double size;
    double stddev;

    mean = datadist.mean();
    size = datadist.size();
    stddev = datadist.standardDeviation();

    m_pTDist = new StudentT(mean, stddev / sqrt(size),
        size-1);
}

probability TTest::density(double x) const {
    return m_pTDist->density(x);
}

probability TTest::cumulative(double x) const {
    return m_pTDist->cumulative(x);
}

```

```

}
```

. . .

3. There's a 69.84% chance that it wasn't found at random.

4. Here's the source code that I used:

```

const int maxCount = 5;
const int numBootstrap = 30;
const int numRepeat = 10;
const int numCycle = 200;

int main(int argc, char* argv[]) {
    double dRandNum;
    int count;
    int countBootstrapIncluded95;
    int countZTestIncluded95;
    int cycle;
    int repeat;
    int member;
    std::multiset<double> *msSample;
    DataDistribution ddMean;
    Normal norm(0, 1);
    ZTest *pztest;

    srand48((long int) std::time(NULL));

    for (count = 3; count <= maxCount; count++) {
        for (repeat = 0; repeat < numRepeat; repeat++)
            {
                countBootstrapIncluded95 = 0;
                countZTestIncluded95 = 0;

                for (cycle = 0; cycle < numCycle; cycle++)
                    {
                        msSample = new std::multiset<double>()
                        >();

                        for (member = 0; member < count; \
                            member++)
                            {
                                dRandNum = norm.random();
                                msSample->insert(dRandNum);
                            }
                    }

                    DataDistribution ddSample(*msSample)
                    ;
    }
}

```

```

ddMean = bootstrapMean(ddSample, \
    count, numBootstrap);
if (ddMean.quantile(0.025) < 0.0 && \
    ddMean.quantile(0.975) > 0.0) {
    countBootstrapIncluded95++;
}

pztest = new ZTest(ddSample);
if (pztest->quantile(0.025) < 0.0 && \
    pztest->quantile(0.975) > 0.0) {
    countZTestIncluded95++;
}
delete pztest;

msSample->erase(msSample->begin(), \
    msSample->end());
delete msSample;
}

std::cout << count << " " <<
    ((double) countBootstrapIncluded95 / ( \
        double) numCycle ) << " " <<
    ((double) countZTestIncluded95 / ( \
        double) numCycle ) <<
    std::endl;
}
}

return 0;
}

```

PAGE 182:

1. Here's the code I used:

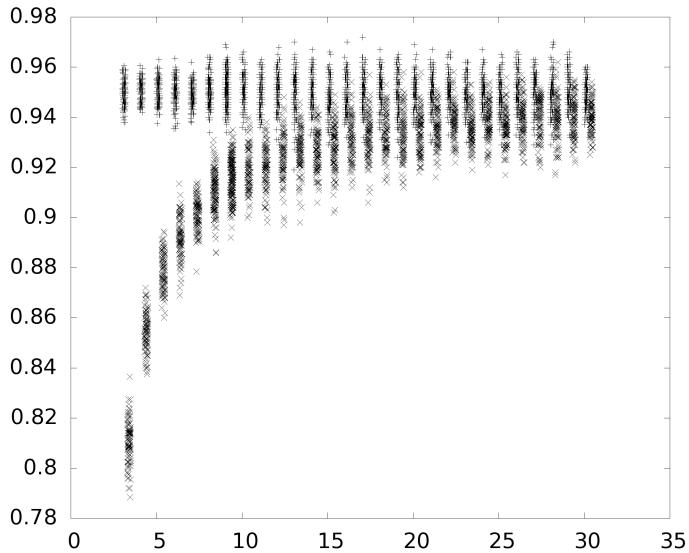
```

const int k_maxElements = 30;
const int k_maxRepeats = 1000000;

int main() {
    std::multiset<double> armulti[k_maxElements + \
        1];
    double dMax;
    double dMin;

```

Figure A.5: Comparison between Z Test and T Test



```

double dRand;
double element;
double estimate;
int numElements;
double range;
int repeat;

srand48((long int) std::time(NULL));

Normal norm(0.0, 1.0);
for (numElements = 2; numElements <= \
      k_maxElements; numElements++) {
    for (repeat = 0; repeat < k_maxRepeats; \
          repeat++) {
        dMin = 99999.9;
        dMax = -99999.9;

        for (element = 0; element < numElements; \
              element++) {
            dRand = norm.random();

            if (dRand < dMin)
                dMin = dRand;
    }
}

```

```

        if (dRand > dMax)
            dMax = dRand;
    }

    range = dMax - dMin;
    estimate = 1.0 / range;
    armulti[numElements].insert(estimate);
}
}

for (numElements = 2; numElements <= √
    k_maxElements; numElements++) {
    Data<double> data(armulti[numElements]);
    DataDistribution dd(data);

    std::cerr << numElements << " & " << dd.√
        quantile(0.05) << " & " << dd.quantile√
        (0.5) << " & " << dd.quantile(0.95) << " √
        \\\\" << std::endl;
}
}

```

PAGE 183:

1. Here's the output that I got from the average of 20 uniform random numbers:
2. The source code is simple:

```

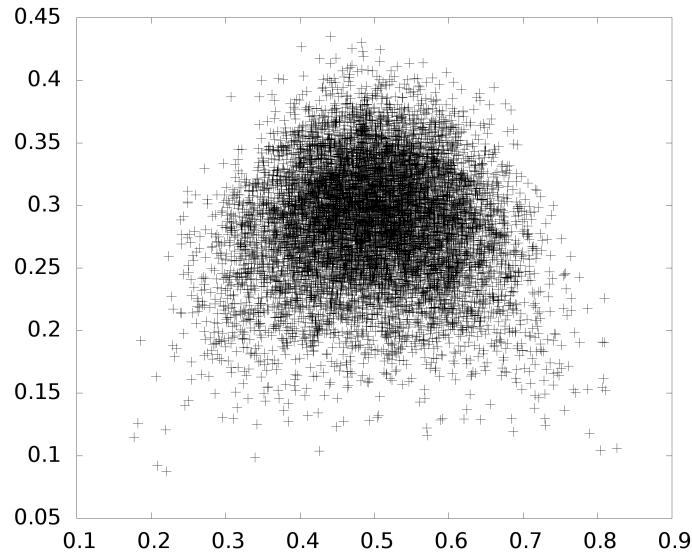
class EstimateSigma : public Distribution<double, √
    double>
{
private:
    double m_dMean;
    double m_dSize;
    double m_dStdDev;
    ChiSquare *m_pchisq;

public:
    EstimateSigma(const DataDistribution &datadist);
    virtual ~EstimateSigma();

    virtual probability density(double x) const;
};

```

Figure A.6: The mean and variance of 20 uniform random numbers



```

EstimateSigma::EstimateSigma(const DataDistribution <
    &datadist) {
    m_dMean = datadist.mean();
    m_dSize = datadist.size();
    m_dStdDev = datadist.standardDeviation();

    m_pchisq = new ChiSquare(m_dSize - 1);
}

EstimateSigma::~EstimateSigma() {
    delete m_pchisq;
}

probability EstimateSigma::density(double x) const {
    if (x > 0.0) {
        return (*m_pchisq)((m_dSize - 1) * m_dStdDev * \
            m_dStdDev / (x * x)) \
        * 2 * (m_dSize - 1) * m_dStdDev * m_dStdDev / \
            (x * x * x);
}

```

```

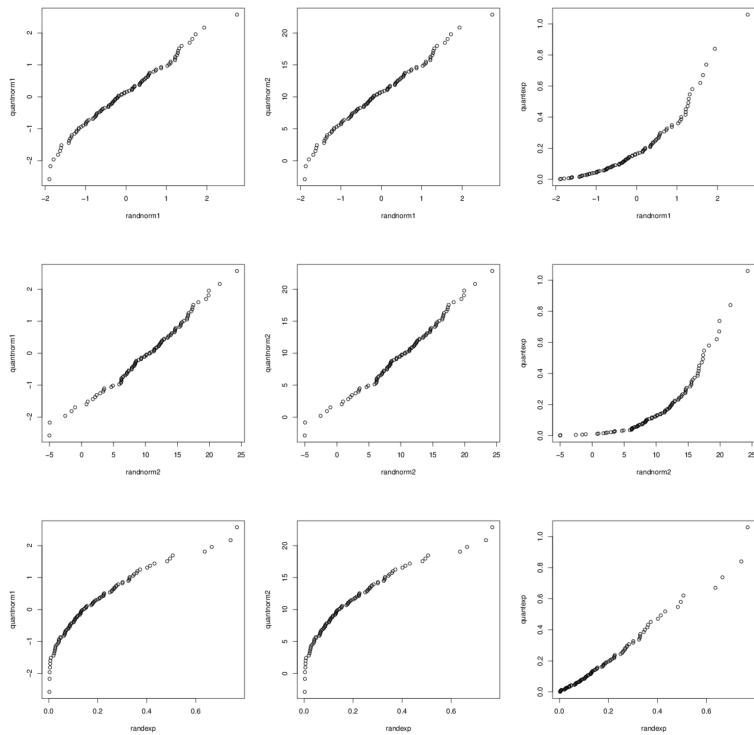
    } else {
        return 0.0;
    }
}

```

PAGE 185:

1. Here are the graphs that I get:

Figure A.7: The graph of normal and exponential against quantiles



A.6 Correlation and Interpolation

PAGE 190:

1. Many times! Here's a few possible answers:
 - (a) The independent and dependent variables depend on a third parent variable.
 - (b) The dependent variables depend on factors other than just the independent variables.
 - (c) There's an intermediate factor between the dependent and independent variables.
-

PAGE 193:

1. Here are the most important routines:

```
LinearRegression::LinearRegression() {
    m_n = 0;
    m_sumX = 0.0;
    m_sumXSquared = 0.0;
    m_sumXY = 0.0;
    m_sumY = 0.0;
}

LinearRegression::~LinearRegression() {
}

void LinearRegression::addPair(DataPair dp)
{
    m_n++;
    m_sumX += dp.getX();
    m_sumXSquared += dp.getX() * dp.getX();
    m_sumXY += dp.getX() * dp.getY();
    m_sumY += dp.getY();
    m_sumYSquared += dp.getY() * dp.getY();
}

LinearFunctor LinearRegression::getFunctor() const
{
    double a;
    double b;
    double avgX;
```

```

double avgY;

avgX = m_sumX / m_n;
avgY = m_sumY / m_n;

b = (m_sumXY - m_n * avgX * avgY) / (
    m_sumXSquared - m_n * avgX * avgX);
a = avgY - b * avgX;

return LinearFunctor(a, b);
}

```

PAGE 194:

1. By definition, $\text{Cov}(X, Y) = E[X - E[X]]E[Y - E[Y]]$. Since $X - E[X]$ is completely independent of $Y - E[Y]$, they act as constants to each other. Therefore:

$$\begin{aligned} E[X - E[X]]E[Y - E[Y]] &= E[(X - E[X])(Y - E[Y])] \\ &= E[XY - YE[X] - X[Y] + E[X]E[Y]] \end{aligned}$$

Now, since $E[Y]$ is independent of X , $E[XE[Y]] = E[X]E[Y]$. For the same reason, $E[YE[X]] = E[X]E[Y]$.

Therefore,

$$\begin{aligned} E[XY - YE[X] - X[Y] + E[X]E[Y]] &= E[XY - 2E[X]E[Y] + E[X]E[Y]] \\ &= E[XY - E[X]E[Y]] \\ &= E[XY] - E[X]E[Y] \end{aligned}$$

PAGE 198:

1. Here's the source:

```

double LinearRegression::getCorrelationCoefficient () const
{
    return (m_n * m_sumXY - m_sumX * m_sumY) /
        sqrt((m_n * m_sumXSquared - m_sumX * m_sumX) * (
            m_n * m_sumYSquared - m_sumY * m_sumY));
}

```

2. Linear correlation is undefined when the denominator is 0. This happens when there is only one element, all x values are the same, or all y values are the same.

3. *****

A.7 Simple Classification

PAGE 213:

1. Each node of the tree is a binary random variable, with two possibilities:

$$x_i \in c_i \text{ with probability } \Pr(x_i) \quad x_i \notin c_i \text{ with probability } 1 - \Pr(x_i)$$

Then the entropy of any node is:

$$\begin{aligned} H(x_i) &= - \sum_i \Pr(x_i) \log_2 \Pr(x_i) \\ &= -\Pr(x_i) \log_2 \Pr(x_i) + \Pr(1-x_i) \log_2 \Pr(1-x_i) \\ &= - \sum_i \Pr(1-x_i) \log_2 \Pr(1-x_i) \\ &= H(1-x_i) \end{aligned}$$

A.8 Statistical Tests for two variables

PAGE 217:

1. Pairs of variables:
 - (a) You cannot use these as pairs of variables. The mice are different.
 - (b) You can use these as pairs of variables.

A.9 Sampling

PAGE 224:

1. Flipping two coins and having both turn up heads is a binomial distribution, so the program estimates $\text{binom}(1000, \frac{1}{4}, y)$. The standard deviation for this binomial distribution is $\sqrt{\frac{1}{4} \times \frac{3}{4} \times 1000} \approx 13.69$. Therefore, the expected 95% confidence interval is expected to be about $500 \pm 1.96 \times 13.69 \approx [223.16, 276.83]$.
-

PAGE 229:

1. ****
 2. ****
-

PAGE 235:

1. Make the same table:

matching	fraction	matching * - $\log_2(\text{fraction})$
62203	1	0
0	0	0
16164	0.260	31413
46039	0.740	19999
5225	0.084	18671
56978	0.916	7217
41868	0.673	23920
30225	0.327	48742
38862	0.625	26351
23341	0.375	33028
32078	0.516	30620
31025	0.484	32481
40842	0.657	24752
21361	0.343	32975
34142	0.549	29537
28061	0.451	32236
Total		391942

This gives us a 21.2% compression, quite a lot better than the 3-bit prediction code.

This compression is mostly due to the 0-bit and the 2-bit. In those two cases, we had certainty or near-certainty about what the bit would be. The more certain that predictions can make, the better the arithmetic code works.

Bibliography

- Didier H. Basset. *Object-Oriented Implementation of Numerical Methods*. Morgan Kaufmann Publishers, 2001.
- Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly, first edition, 2009. Available at <http://www.nltk.org/book>.
- William M. Bolstad. *Introduction to Bayesian Statistics*. Wiley Interscience, first edition, 2004.
- Bruce L. Bowerman, Richard T. O'Connell, and Anne B. Koehler. *Forecasting, Time Series, and Regression*. Thomson, fourth edition, 2005. ISBN 0-534-40977-6.
- Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Thomson Brooks/Cole, eighth edition, 2005.
- Roger L. Burford. *Statistics: A Computer Approach*. Charles E. Merrill Publishing Company, first edition, 1968.
- Peter J. Cameron. *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, first edition, 1994.
- John L. Casti. *Reality Rules II: Picturing the World in Mathematics*. Wiley Interscience, first edition, 1992. ISBN 0-471-57798-7.
- Cuthbert Daniel and Fred S. Wood. *Fitting Equations to Data*. John Wiley & Sons, second edition, 1980.
- Ivar Ekeland. *The Broken Dice and Other Mathematical Tales of Chance*. The University of Chicago Press, first edition, 1991. ISBN 0-226-19991-6.
- Geof H. Givens and Jennifer A. Hoeting. *Computational Statistics*. Wiley-Interscience, first edition, 2005. ISBN 978-0471461241.
- Graham Glass and Brett Schuchert. *The STL Primer*. Prentice Hall PTR, first edition, 1996.
- Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. AMS, second revised edition, 1997. Available at http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html.

- William L. Hays. *Statistics*. Holt, Rinehart and Winston, Inc., fourth edition, 1988.
- Frank A. Height. *Handbook of the Poisson Distribution*. John Wiley & Sons, Inc., first edition, 1967.
- James J. Higgins and Sallie Keller-McNulty. *Concepts in Probability and Stochastic Modelling*. Duxbury Press, 1995.
- Schuyler W. Huck. *Statistical Misconceptions*. Routledge, Taylor, & Francis Group, first edition, 2009. ISBN 978-0-8058-5904-1.
- James L. Johnson. *Probability and Statistics for Computer Science*. Wiley-Interscience, first edition, 2003. ISBN 0-471-32672-0.
- Normal L. Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume two. John Wiley & Sons, Inc., second edition, 1995.
- Normal L. Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*, volume one. John Wiley & Sons, Inc., second edition, 1994.
- Nicolai M. Josuttis. *The C++ Standard Library*. Addison-Wesley, 1999.
- Jack Kleijnen and Willem Van Groenendaal. *Simulation: A Statistical Perspective*. John Wiley & Sons, Inc., first english edition edition, 1992.
- Daniel Kleppner and Norman Ramsey. *Quick Calculus: A Self-Teaching Guide*. John Wiley and Sons, Inc., second edition, 1985.
- Donald Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume two. Addison-Wesley Publishing Company, second edition, 1981.
- Debasis Kundu and Ayanendranath Basu. *Statistical Computing: Existing Methods and Recent Developments*. Alpha Science International Ltd., first edition, 2004.
- Leslie Lamport. *L^AT_EX A Document Preparation System*. Addison-Wesley, first edition, 1986.
- Richard J. Larsen and Morris L. Marx. *An Introduction to Mathematical Statistics and Its Applications*. Pearson Prentice-Hall, fourth edition, 2006.
- Seymour Lipschutz and Jr. John J. Schiller. *Schaum's Outlines: Introduction to Probability and Statistics*. McGraw-Hill, second edition, 1998.
- Simon Lovell. *How to Cheat at Everything*. Thunder's Mouth Press, first edition, 2003.
- Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, first edition, 2006.
- Scott Meyers. *Effective C++*. Addison-Wesley, third edition, 2005.

- Frank Mittelbach and Michel Goossens. *The L^AT_EX Companion*. Addison-Wesley, second edition, 2004.
- David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics*. W.H. Freeman & Company, fifth edition, 2005. Taken from http://bcs.whfreeman.com/ips5e/content/cat_080/pdf/moore14.pdf.
- Paul J. Nahin. *Duelling Idiots and Other Probability Puzzlers*. Princeton University Press, first edition, 2000.
- R. Lyman Ott and Michael Longnecker. *An Introduction to Statistical Methods and Data Analysis*. Duxbury Press, fifth edition, 2001.
- John W. Pratt, Howard Raiffa, and Robert Schlaifer. *Introduction to Statistical Decision Theory*. The MIT Press, first edition, 1995. ISBN 978-0-262-66206-2.
- Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, second edition, 2004.
- Sheldon Ross. *A First Course in Probability*. Pearson Prentice-Hall, seventh edition, 2006.
- George G. Roussas. *A Course in Mathematical Statistics*. Academic Press, second edition, 1997.
- Sam L. Savage. *The Flaw of Averages*. John Wiley & Sons, first edition, 2009. ISBN 978-0-471-38197-6.
- Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers, second edition, 2000.
- Wolfgang Schwarz. *40 Puzzles and Problems in Probability and Mathematical Statistics*. Springer, first edition, 2008. ISBN 978-0-387-73511-5.
- David J. Sheskin. *Handbook of Nonparametric and Statistical Procedures*. Chapman & Hall / CRC, third edition, 2004.
- George W. Snedecor and William G. Cochran. *Statistical Methods*. Iowa State Press, eighth edition, 1989.
- Tom Stafford and Matt Webb. *Mind Hacks: Tips & Tools for Using Your Brain*. O'Reilly, first edition, 2005.
- Inc. StatSoft. *Electronic Statistics Textbook*. StatSoft, electronic edition, 2010. Available at <http://www.statsoft.com/textbook/>.
- Dennis D. Wackerly, William Mendenhall III, and Richard L. Scheaffer. *Mathematical Statistics with Applications*. Duxbury Press, sixth edition, 2002.
- Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kauffman, third edition, 2011. ISBN 978-0-12-374856-0.

John W. Wright, editor. *The New York Times Guide to Essential Knowledge*. St. Martin's Press, first edition, 2004.

Index

- χ^2 test, 185
- accurate, 151
- alpha value, 153
- alternate hypothesis, 154
- Bayes' Formula, 75
- bit, 79
- Calculus, 40
- cardinality, 16
- CDF, 82
- central tendency, 27
- Chebyshev distance, 38
- chi-squared distribution, 143, 182
- city block distance, 38
- closed end point, 1
- closed interval, 1
- combinatorics, 18
- complement, 15
- compression code, 229
- concave function, 63
- conditional probability, 65
- confidence interval, 32
- continuity correction, 136
- continuous, 5
- control, 217
- convex function, 63
- covariance, 194
- Cumulative Distribution Functions, 82
- data distribution, 85
- definite integral, 48
- degrees of freedom, 143, 176
- differential, 42
- discrete, 5
- disjoint, 8
- dispersion, 31
- Euclidean distance, 34
- excess kurtosis, 39
- expected value, 55
- experiment, 4
- exponential distribution, 123
- extrapolation, 189
- extreme outlier, 159
- factorial, 21
- functor, 40
- gamma distribution, 140
- gamma function, 137
- half-open interval, 2
- hypergeometric, 106
- indefinite integration, 47
- independent, 66
- infinite sum, 18
- integration, 47
- inter-quartile range, 32
- interpolation, 189
- intersection, 7
- interval, 1
- Jensen's Inequality, 63
- join, 9
- Kolmogorov Axioms, 60
- kurtosis, 39
- limit, 41
- Mandelbrot set, 227
- Manhattan distance, 38

- Martingale system, 57
- maximum absolute deviation, 38
- maximum likelihood, 161
- mean, 29, 55
- mean absolute deviation, 38
- measurement, 4
- median, 28, 32
- memoryless, 124, 127
- midrange, 37
- mode, 28
- moments of a distribution, 38
- Monte Carlo Integration, 229
- Monte Carlo Method, 226
- normal distribution, 131
- null hypothesis, 154
- one-sided z-test, 170
- open end point, 1
- open interval, 1
- oracle, 203
- order statistics, 33
- outlier, 159
- overfit, 214
- p-value, 153
- partition, 53
- percentile, 32
- Poisson, 112
- Poisson Process, 112
- Poisson queue, 115
- precise, 151
- probability density function, 60
- probability distribution function, 59, 83
- product notation, 19
- product rule, 20
- products, 18
- Quantile Method, 91
- quartile, 32
- random variable, 54
- range, 32
- Riemann Rule, 49
- sample, 4
- sample space, 4
- sensitivity, 155
- set, 3
- set negation, 15
- skewness, 39
- slope, 42
- specificity, 155
- standard deviation, 35
- Standard Normal Curve, 49
- state, 74
- statistic, 26
- Student's t-distribution, 176
- subset, 11
- sum of squared residuals, 194
- summation sign, 18
- summations, 18
- taxicab metric, 38
- TOST, 299
- treatment, 217
- trimmed mean, 37
- two-sided z-test, 173
- Type I error, 154
- Type II error, 154
- unconditional probabilities, 65
- uncorrelated, 66
- union, 9
- unrestricted sample, 25
- unweighted mean, 29
- variance, 34, 55
- variation, 31
- Wilcoxon signed-rank test, 178
- winsorized mean, 37