

Computations for Markov Chain Usage Models

S. J. Prowell

Technical Report UT-CS-03-505

Preface

This document summarizes the basic computations for Markov chain usage models, presents their derivations, and includes Scilab code to compute each of them.

The contents of this document are the result of years of work by many different people, and very few results are original. James Whittaker, Michael Thomason, and Jesse Poore did the original work on Markov chain usage models [17, 18]. Gwen Walton's research applied mathematical programming techniques to set model probabilities under testing constraints [16]. Jenny Morales [8] and Dave Pearson [9] investigated combining information across tests to improve reliability measurements. Kirk Sayre's research provided many new and useful analytical results, and provided a framework for simulation and partition testing [13]. Walter Gutjahr demonstrated how a Markov chain could be modified to bias test generation toward low-use critical function, and how the bias could be removed in the results [1].

This document only discusses computations. Carmen Trammell and Jesse Poore have written about experimental control during the testing process [15]; Rob Oshana and Dave Kelly have written about their experience applying Markov chain usage models to very large, embedded, real-time, distributed systems [5, 4].

Acknowledgments

This document began life as a summary of Markov computations to answer questions in an email discussion with Tjin Merry. The discussion of Markov chain usage modeling, of the related computations, and of Markov chain-based testing contributed greatly to my understanding of the subject. I thank Tjin Merry for prompting this document, and for patience with my explanations.

Many people have contributed to this manuscript. I was first introduced to the topic of Markov chain usage models by Jesse Poore and James Whittaker. Michael Corum provided considerable support and assistance by both reading and using this document. Special thanks are due Zizhong Chen and Chris Sellers, who carefully proofread a draft of this document and uncovered many careless mistakes; any errors which remain are mine alone.

Contents

1 Preliminary Definitions	15
1.1 Scilab	15
1.2 Notation	15
1.3 Expectation of random variables	17
1.4 Variance of random variables	18
2 Markov Computations	21
2.1 Number of occurrences of a state in a test case	24
2.2 Computing the long-run state probabilities	26
2.3 Sensitivity analysis	29
2.4 Other long run statistics	29
2.5 Probability of occurrence for states	33
2.6 Probability of occurrence for arcs	34
2.7 Probability of occurrence for stimuli	37
2.8 First passage times	37
2.9 Number of occurrences of a stimulus in a test case	40
3 Information Theory	45
3.1 Entropy	45
3.2 Statistically-typical sequences	47
3.3 Discrimination	49
4 Reliability Models	53
4.1 Sample reliability	54
4.2 Binomial distribution	54
4.3 Testing Markov chain	57
4.4 Beta distribution and the Miller model	63
4.5 Confidence	64
4.6 Choosing priors for the beta distribution	67
A Probability Density and Distribution Functions	71
A.1 The gamma and beta functions	71
A.2 Probability density and distribution functions	74
A.3 The beta distribution	76

B	Alternative Derivations of Information Theory Results	77
B.1	Statistically-typical sequences	77
B.2	Discrimination	78
B.3	Sayre discrimination	79

List of Figures

2.1	Example usage model	22
4.1	Reliability, confidence, and number of tests	57
4.2	Usage model with successful traversal counts	59
4.3	Usage model with failure states	60
4.4	Testing Markov chain modified to compute reliability	61
4.5	Confidence level versus priors	67

List of Tables

2.1	Expected occurrence and associated variance for each state	25
2.2	Long-run occupancies	28
2.3	Transition sensitivities	29
2.4	Probability of occurrence for each state	33
2.5	Probability of occurrence for each arc	37
2.6	Mean first passage times and associated variances (events) for each state	38
2.7	Mean first passage times and associated variances (test cases) for each state	41
2.8	Stimulus expectations	42
4.1	Trials to achieve estimated reliability and confidence levels	56
4.2	Expectation and variance of failure states	62
A.1	Common probability density functions (for $x > 0, a > 0, b > 0$)	75
A.2	Common probability distribution functions (for $x = 0, 1, 2, \dots$)	75

List of Algorithms

1	Row-normalize a matrix	16
2	Compute the binomial coefficient $\binom{n}{p}$	16
3	Compute the beta function $B(a, b)$	16
4	Compute the expected number of occurrences for each state	25
5	Compute the Perron eigenvector (long-run probabilities)	27
6	Compute an approximation to the Perron eigenvector via the power method	30
7	Compute the matrix of sensitivities	31
8	Compute the stimulus long-run occupancy	32
9	Compute the probability of occurrence for each state	34
10	Compute the probability of occurrence for each arc	36
11	Compute the mean first passage times and their variances	39
12	Compute mean first passage in terms of test cases	41
13	Compute the stimulus expectation matrix	43
14	Compute source entropies	48
15	Compute the perturbed discrimination	52
16	Compute the confidence for the Miller model	66

Chapter 1

Preliminary Definitions

This chapter summarizes some basic results used in the remainder of the document. See [2] for a discussion of Matrix analysis, and see [10] for a discussion of probability theory.

1.1 Scilab

The algorithms presented in this paper can be executed using Scilab. This is a freely-available software package which runs on several different platforms. While the Scilab input language is very similar to MATLAB's input language, there are differences and the algorithms presented here may require modification to run under MATLAB. Information about Scilab is available from Inria:

<http://www-rocq.inria.fr/scilab/>

It will be necessary to use several stochastic matrices, often constructed from frequency counts by row-normalization. A Scilab procedure to accomplish this is presented as algorithm 1.

Oddly enough, the current version of Scilab (2.5) does not provide the binomial coefficient $\binom{n}{p}$, or compute the beta function. Procedures for these are given as algorithms 2 and 3. These will be used when reliability models are introduced.

1.2 Notation

Every random variable X has an associated probability distribution $P : X \rightarrow [0, 1]$ such that $\int_X dP = 1$. For a random variable X with associated probability distribution P , the probability of a particular outcome $\Pr[X = x]$ will be denoted simply $p(x)$. Thus if one has two random variables X and Y , and writes $p(x)p(y)$, it should be understood that this represents $\Pr[X = x]\Pr[Y = y]$.

Algorithm 1 Row-normalize a matrix

```

// Row-normalize a matrix.
//
// P: a matrix
function [P]=r_n(P),
    [m,n]=size(P);

    // Get the row sums.
    E(1:m)=1; v=P*E;

    // Normalize.
    for i=1:m, P(i,:)=P(i,+)/v(i); end;
endfunction;

```

Algorithm 2 Compute the binomial coefficient $\binom{n}{p}$

```

// Compute the binomial coefficient.
//
// B: The binomial coefficient.
// n: The number of objects of both classes.
// p: The number of objects of one class.
function [B]=binomial_coeff(n,p),
    B=exp(sum(log(p+1:n))-sum(log(1:n-p)));
endfunction;

```

Algorithm 3 Compute the beta function $B(a,b)$

```

// Compute the beta function.
//
// B: The value of the beta function.
// a: The first parameter.
// b: The second parameter.
function [B]=beta_function(a,b),
    B=exp(gammln(a)+gammln(b)-gammln(a+b));
endfunction;

```

The *joint* probability $\Pr[X = x \wedge Y = y]$ will be denoted simply $p(x\&y)$. The conditional probability $\Pr[Y = y|X = x]$ will be denoted simply $p(y|x)$.

The Kronecker delta $\delta_{i,j}$ extended to the real numbers is defined for $i, j \in \mathbb{R}$ as:

$$\delta_{i,j} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j. \end{cases}$$

Matrices will be denoted by upper-case letters, and elements of matrices by lower-case letters. For example, the $m \times n$ matrix A consists of mn elements $a_{i,j}$, where i and j are the row and column indices of the element, respectively. To make the association clear, matrix A may be written $A = [a_{i,j}]$. It will often be useful to refer to a single column of a matrix; this will be done by subscripting the matrix, so A_j denotes the j th column vector of matrix A .

For any matrix $A = [a_{i,j}]$ let $A_d = [a_{i,j}\delta_{i,j}]$ denote the matrix consisting of the diagonal of A , with zeros everywhere else. Whenever necessary, let $U = [1]$ denote a matrix of ones of appropriate size.

1.3 Expectation of random variables

The *expectation* of a random variable X , denoted $E[X]$, is an unbiased estimator of the random variable's value. In the discrete case this expectation is a probability-weighted average of the random variable's values:

$$E[X] = \sum_{x \in X} x \cdot p(x).$$

Mathematical expectation exhibits several useful properties:

- *Linearity:* $E[\sum_{i=1}^n a_i X_i] = \sum_{i=1}^n a_i E[X_i]$
- *Positivity:* If $X \geq 0$ then $E[X] \geq 0$. If $X \geq Y$ then $E[X] \geq E[Y]$
- $E[|X|] \geq |E[X]|$
- *The Schwarz Inequality:* $|E[XY]|^2 \leq E[|X|^2]E[|Y|^2]$; equality holds if and only if there is a real constant λ such that $\lambda X + Y = 0$.
- *Jensen's Inequality:* Let $g : \mathbb{R} \rightarrow \mathbb{R}$ be a convex Borel function. Then $g(E[X]) \leq E[g(X)]$.

For two random variables X and Y , one has:

$$E[XY] = \sum_{x \in X, y \in Y} xy \cdot p(x\&y).$$

If and only if the two random variables are independent, then $p(x&y) = p(x)p(y)$. In this case, the following derivation holds:

$$\begin{aligned}
 E[XY] &= \sum_{x \in X, y \in Y} xy \cdot p(x&y) \\
 &= \sum_{x \in X, y \in Y} xy \cdot p(x)p(y) \\
 &= \sum_{x \in X, y \in Y} x \cdot p(x) \cdot y \cdot p(y) \\
 &= \left(\sum_{x \in X} x \cdot p(x) \right) \left(\sum_{y \in Y} y \cdot p(y) \right) \\
 &= E[X]E[Y].
 \end{aligned}$$

1.4 Variance of random variables

How far can one expect the value of a random variable X to differ from its expectation $E[X]$? This question is answered by the *variance* $Var[X]$ of the random variable, which is the expectation of the square of the difference between the observed value and the expected value:

$$Var[X] = E[(X - E[X])^2].$$

One can “scale” the variance back by taking the square root. This is called the *standard deviation* $\sigma[X] = \sqrt{Var[X]}$.

The variance of random variable X can be restated using linearity and the definition of expectation as follows:

$$\begin{aligned}
 Var[X] &= E[(X - E[X])^2] \\
 &= E[X^2 - 2XE[X] + E^2[X]] \\
 &= E[X^2] - 2E[X]E[X] + E^2[X] \\
 &= E[X^2] - E^2[X].
 \end{aligned} \tag{1.1}$$

Eq. 1.1 is the standard formula for variance. It can be applied to a simple sum of two

random variables X and Y as follows:

$$\begin{aligned}
\text{Var}[X+Y] &= E[(X+Y - E[X+Y])^2] \\
&= E[X^2 + Y^2 + E^2[X+Y] + 2XY \\
&\quad - 2XE[X+Y] - 2YE[X+Y]] \\
&= E[X^2 + Y^2 + (E[X] + E[Y])^2 + 2XY \\
&\quad - 2XE[X] - 2XE[Y] - 2YE[X] - 2YE[Y]] \\
&= E[X^2 + Y^2 + E^2[X] + 2E[X]E[Y] + E^2[Y] + 2XY \\
&\quad - 2XE[X] - 2XE[Y] - 2YE[X] - 2YE[Y]] \\
&= E[X^2] + E[Y^2] + E^2[X] + 2E[X]E[Y] + E^2[Y] + 2E[XY] \\
&\quad - 2E^2[X] - 4E[X]E[Y] - 2E^2[Y] \\
&= E[X^2] + E[Y^2] - E^2[X] - E^2[Y] + 2E[XY] - 2E[X]E[Y] \\
&= \text{Var}[X] + \text{Var}[Y] + 2(E[XY] - E[X]E[Y]). \tag{1.2}
\end{aligned}$$

The term $E[XY] - E[X]E[Y]$ in eq. 1.2 is referred to as the *covariance* of X and Y , denoted $\text{Cov}[X, Y]$. Thus one can write the above equation as:

$$\text{Var}[X+Y] = \text{Var}[X] + \text{Var}[Y] + 2\text{Cov}[X, Y].$$

If X and Y are independent random variables, one has $\text{Cov}[X, Y] = E[XY] - E[X]E[Y] = E[X]E[Y] - E[X]E[Y] = 0$, and $\text{Var}[X+Y] = \text{Var}[X] + \text{Var}[Y]$. If X and Y are not independent, one can use the Schwarz inequality to obtain an upper bound for $\text{Var}[X+Y]$ (in the covariance is not readily computable). Consider the Schwarz inequality for random variables V and W , and make the substitutions $V = X - E[X]$ and $W = Y - E[Y]$:

$$\begin{aligned}
E^2[VW] &\leq E[V^2]E[W^2] \\
E^2[(X - E[X])(Y - E[Y])] &\leq E[(X - E[X])^2]E[(Y - E[Y])^2] \\
E^2[XY - YE[X] - XE[Y] + E[X]E[Y]] &\leq \text{Var}[X]\text{Var}[Y] \\
(E[XY] - E[X]E[Y])^2 &\leq \text{Var}[X]\text{Var}[Y] \\
\text{Cov}^2[X, Y] &\leq \text{Var}[X]\text{Var}[Y] \\
\text{Cov}[X, Y] &\leq +\sqrt{\text{Var}[X]\text{Var}[Y]} \tag{1.3}
\end{aligned}$$

Since eq. 1.3 is an upper bound on the covariance, it follows that one has the inequality:

$$\text{Var}[X+Y] \leq \text{Var}[X] + \text{Var}[Y] + 2\sqrt{\text{Var}[X]\text{Var}[Y]}.$$

Note that generalizing the variance to an arbitrary sum requires all covariance pairs; thus if one has n terms in a sum one will require $\binom{n}{2} = (n^2 - n)/2$ covariances. The general expression can be written:

$$\text{Var}\left[\sum_{i=1}^n X_i\right] \leq \sum_{i=1}^n \text{Var}[X_i] + 2\sum_{i \neq j} \sqrt{\text{Var}[X_i]\text{Var}[X_j]}. \tag{1.4}$$

If X and Y are “completely dependent” ($X = Y$), such as for a case in which one state unconditionally follows another in a Markov chain, then one has:

$$\begin{aligned} \text{Cov}[X, Y] &= E[XY] - E[X]E[Y] \\ &= E[X^2] - E^2[X] \\ &= \text{Var}[X] = \text{Var}[Y]. \end{aligned}$$

While the variance doesn't have as many nice properties as the expectation, a more intuitive sense of what it means can be obtained from the Chebyshev inequality. This states that the probability that the observed value of the random variable X differs from its expectation $E[X]$ by at least k (for some positive k) is bounded above by $\text{Var}[X]/k^2$. That is:

$$\Pr[|X - E[X]| \geq k] \leq \frac{\text{Var}[X]}{k^2}. \quad (1.5)$$

Note that eq. 1.5 is a *statistically conservative* upper bound. It may be possible to do much better if more is known about the random variable, especially its distribution. As an example, consider a 95% confidence interval:

$$\begin{aligned} 0.95 &\leq \frac{\text{Var}[X]}{k^2} \\ k^2 &\leq \frac{\text{Var}[X]}{0.95} \\ k &\leq \sqrt{\frac{\text{Var}[X]}{0.95}}. \end{aligned}$$

Thus one can conclude that 95% of the time the observed value will be in the closed interval:

$$\left[E[X] - \sqrt{\frac{\text{Var}[X]}{0.95}}, E[X] + \sqrt{\frac{\text{Var}[X]}{0.95}} \right].$$

Chapter 2

Markov Computations

Many of the computations in this chapter are taken directly from [6].

Consider the Markov chain usage model in fig. 2.1. TML is a language developed for representing Markov chain usage models [12]. This model is equivalent to the following TML.

```
model example
[Enter] ($ 1 $) "a" [A]
[A]      ($ 1/2 $) "b" [B]
         ($ 1/2 $) "c" [C]
[B]      ($ 1/2 $) "b" [B]
         ($ 1/4 $) "c" [C]
         ($ 1/4 $) "e" [Exit]
[C]      ($ 1/4 $) "a" [A]
         ($ 1/2 $) "e" [Exit]
         ($ 1/4 $) "f" [Exit]
end
```

There are two special states in this model: the *source* or enter state [Enter], and the *sink* or exit state [Exit]. The sink state represents the end of a test (or use) and thus no usage events are possible from this state.

This example model can be described by two matrices. For these matrices let the states be indexed in the order [Enter], [A], [B], [C], and [Exit], and let the stimuli be indexed in the order "a," "b," "c," "e" and "f."

The first matrix is the (*state*) *transition* matrix, $P = [p_{i,j}]$, for which $p_{i,j}$ is the probability that the next state is j given that the current state is i . For the example model the transition matrix is:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & 0 & \frac{3}{4} \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

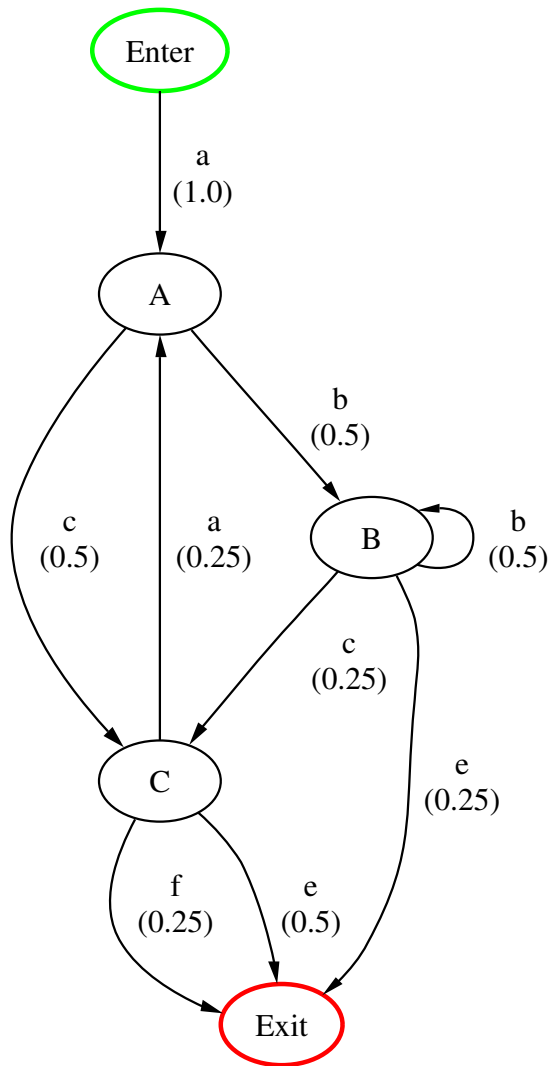


Figure 2.1: Example usage model

Note that the example matrix has been made *recurrent* by connecting [Exit] back to [Enter]. This will be important for several computations. Note that there are two transitions from state [C] to state [Exit]; these two transitions are mutually exclusive, so the total probability of a transition from [C] to [Exit] is the sum of the two: $\frac{1}{2} + \frac{1}{4} = \frac{3}{4}$. For many computations it will be useful to have a reduced matrix $Q = [q_{i,j}]$ in which the row and column for the model sink have been removed. For the example model the reduced matrix is:

$$Q = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & 0 \end{bmatrix}.$$

Both the P and Q matrices are square matrices.

The second matrix is the *stimulus (occurrence)* matrix $S = [s_{i,j}]$, for which $s_{i,j}$ is the probability that the next stimulus will be stimulus j given that the current state is i . For the example model, the stimulus matrix is:

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

The usage models used are deterministic; a stimulus can only label one outgoing transition from a state, so there is no need to sum transition probabilities in the stimulus matrix.

It will occasionally be convenient to discuss the probability of a transition from state i to state j on an arc labeled with a particular stimulus k . This probability will be denoted $p_{i,j,k}$ when needed. The *restriction* of a matrix A to only those elements corresponding to stimulus k will be denoted $A|_k$. Thus $P|_k = [p_{i,j,k}]$ for fixed k , and:

$$Q = \sum_k Q|_k. \quad (2.1)$$

Note that eq. 2.1 cannot be re-written for P because the recurrence loop is not labeled with a stimulus. For the example model, the restriction of P to stimulus e is the matrix:

$$P|_e = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Throughout the rest of this document, the source will be assumed to have index 1, while the sink will be assumed to have index n . The number of stimuli for a model will be s . Thus P is an $n \times n$ matrix, while S is a $(n-1) \times s$ matrix, since there can be no outgoing arcs (other than the recurrence loop) from the sink.

It is worth noting a few important facts about P . First, thanks to the recurrence loop the directed graph of the model is strongly-connected, meaning that from any node i there is a path to any other node j . It follows that P is an irreducible matrix (see Theorem 6.2.24 in [2]).

2.1 Number of occurrences of a state in a test case

The number of occurrences of a state in a test case can be computed using the reduced matrix Q . If the sink is made absorbing ($p_{n,n} = 1$ and $p_{n,1} = 0$), one can then compute the number of occurrences of a state prior to absorption. Let T denote the set of transient (non-absorbing) states, which will in this case be every state other than the sink. Let $n_{i,j}$ denote the number of occurrences of state j prior to absorption, given that one starts in state i . Using the “first passage” method, one obtains the relation:

$$E[n_{i,j}] = \delta_{i,j} + \sum_{k \in T} p_{i,k} E[n_{k,j}]. \quad (2.2)$$

Let $N = [E[n_{i,j}]]$. Eq. 2.2 can be re-written in matrix form as:

$$\begin{aligned} N &= I + QN \\ N - QN &= I \\ (I - Q)N &= I \\ N &= (I - Q)^{-1}. \end{aligned} \quad (2.3)$$

The matrix N defined by eq. 2.3 is called the *fundamental matrix* for absorbing chains. Many Markov chain results can be obtained from the fundamental matrix.

What about the variance associated with this expectation? While $E^2[n_{i,j}]$ is easy to compute, the other term $E[n_{i,j}^2]$ is a bit harder:

$$E[n_{i,j}^2] = n_{i,j}(2n_{j,j} - 1). \quad (2.4)$$

The derivation of eq. 2.4 can be found in [6]. Thus the variance $Var[n_{i,j}]$ is:

$$Var[n_{i,j}] = n_{i,j}(2n_{j,j} - 1) - n_{i,j}^2.$$

These results allow the computation of the expected number of occurrences for each state and the associated variance by the Scilab procedure in algorithm 4. Note that for a usage model one is primarily concerned only with the first row of these matrices, since one is concerned with the behavior when the model is started from the source. The results obtained from the algorithm for the example model are presented in table 2.1.

Let l denote the number of state transitions from the source to the sink in a test case. The expected length $E[l]$ of a test case generated from a Markov chain usage model can be quickly computed once N is known:

$$E[l] = E \left[\sum_{i=1}^{n-1} n_{1,i} \right] = \sum_{i=1}^{n-1} E[n_{1,i}]. \quad (2.5)$$

That is, the average length of a test case is the total average number of state transitions from the source to the sink. Note that $E[l]$ does not include the sink; the total average number of state visits (including the sink) is $E[l] + 1$, since the sink is always visited exactly once. In section 2.8 another way to obtain $E[l]$ will be presented, along with a means to obtain $Var[l]$.

Table 2.1: Expected occurrence and associated variance for each state

State	Occurrence (visits / test case)	Variance (visits / test case)
Enter	1.000	0.000
A	1.231	0.284 0
B	1.231	2.556
C	0.923 1	0.497 0
Exit	1.000	0.000

Algorithm 4 Compute the expected number of occurrences for each state

```

// Compute the non-terminal expectation
// and variance matrices for the stochastic
// matrix P.
//
// P: a stochastic matrix
// N: the expected occurrence of each state
// V: the associated variances
function [N,V]=get_nte(P),
    // Discard the last row and column to obtain
    // the reduced matrix.
    n=size(P,1)-1;
    Q=P(1:n,1:n);
    // Compute the expectation matrix.
    N=inv(eye(n,n)-Q);

    // Compute the variance matrix. Note that
    // diag(N) is a vector, whereas diag(diag(N))
    // is a matrix.
    V=N*(2*diag(diag(N))-eye(n,n))-(N.*N);
endfunction;

```

2.2 Computing the long-run state probabilities

Assume that many realizations (test cases) are generated from a usage model. For each state one can sum the number of occurrences of the state, and then divide this by the number of occurrences of all states. As the number of test cases becomes very large, this ratio will approach a fixed value called the *long-run occupancy* or *long-run probability* of the state. Since the chain is always in one of the states, the sum of all long-run occupancies is one.

Let $\Pi = [\pi_1, \pi_2, \dots, \pi_n]$ be the vector of long-run probabilities for the n states. This vector can be found for a given transition matrix P as the unique stochastic vector solution to the eigenvector equation:

$$\Pi = \Pi P. \quad (2.6)$$

The eigenvector Π is sometimes called the *Perron* eigenvector. Eq. 2.6 is equivalent to the system of equations:

$$\begin{aligned} \pi_1 &= \pi_1 p_{11} + \pi_2 p_{21} + \dots + \pi_n p_{n1} \\ \pi_2 &= \pi_1 p_{12} + \pi_2 p_{22} + \dots + \pi_n p_{n2} \\ &\vdots \\ \pi_n &= \pi_1 p_{1n} + \pi_2 p_{2n} + \dots + \pi_n p_{nn} \\ 1 &= \pi_1 + \pi_2 + \dots + \pi_n. \end{aligned}$$

Note that there are n unknowns (the n π_i values) and $n + 1$ equations. Incidentally, each row of $P^\infty = \lim_{n \rightarrow \infty} P^n$ is equal to the vector Π , unless P is periodic [6].

There are many ways to solve this system. For a simple solution by hand, back-substitution will work just fine using the above equations.

The fundamental matrix can be used to obtain Π . The average number of state visits in a test case is $E[l] + 1$. The fraction of time one spends in state i in the long run is π_i , so the average number of times state i will be visited in a single test case is $\pi_i(E[l] + 1)$. This gives:

$$\pi_i = \frac{E[n_{1,i}]}{E[l] + 1}. \quad (2.7)$$

Note that $E[n_{1,n}]$ is not computed via eq. 2.3, but that one can conclude that $E[n_{1,n}] = 1$ since the sink is always visited exactly once in a test case. Algorithm 5 uses this relationship to compute Π .

To use algorithm 5, define your P matrix and then call the function. The example model's transition matrix can be expressed in Scilab as follows:

```
p = [
0 1 0 0 0
0 0 1/2 1/2 0
0 0 1/2 1/4 1/4
0 1/4 0 0 3/4
1 0 0 0 0 ]
```

Algorithm 5 Compute the Perron eigenvector (long-run probabilities)

```
// Compute the Perron eigenvector for the
// stochastic matrix P and return it. The
// computation is performed by computation
// of the fundamental matrix.
//
// pi: the Perron eigenvector.
// P: a square stochastic matrix.
// N: the fundamental matrix, if available.
function [pi]=get_pi(P,N),
    // Get the matrix size.
    n=size(P,1);

    // Compute the fundamental matrix.
    if argn(2)<2 then
        [N,V]=get_nte(P);
    end;

    // Get the test case length, plus one.
    l=1.0;
    for i=1:n-1, l=l+N(1,i); end;
    // Compute the solution.
    for i=1:n-1, pi(1,i)=N(1,i)/l; end;
    pi(1,n)=1/l;
endfunction;
```

Table 2.2: Long-run occupancies

State	Long-run Occupancy
Enter	0.1857
A	0.2286
B	0.2286
C	0.1714
Exit	0.1857

The long-run occupancies can be computed using the following command:

```
get_pi(p)
```

The results from the analysis are presented in table 2.2. Note that if one already has the fundamental matrix, it can be passed as the optional second argument to avoid recomputing it:

```
get_pi(p, N)
```

An alternate method of getting the long run distribution is to use an iterative method called the *power method*. Given a guess at the long-run distribution Π_i , one computes a slightly better guess by $\Pi_{i+1} = \Pi_i P$. This method is guaranteed to converge to the Perron eigenvector if the matrix is *primitive* [2, p. 516]. Without going into too much detail, if there is a non-zero entry on the diagonal, the matrix is primitive (though the converse is not true). Thus if $p_{i,i} \neq 0$ for any i , then the power method will converge to the Perron eigenvector.

One way to guarantee primitivity is to introduce a “dummy” state d with a self-loop to ensure primitivity (that is, $p_{d,d} \neq 0$), apply the power method, then remove the dummy state and correct for its presence. This can be done by adding the dummy state on the recurrence loop from the sink to the source, so $p_{\text{sink},d} = 1$, $p_{d,d} = 1/2$, and $p_{d,\text{source}} = 1/2$; flow always passes straight through the state, so any of the probability mass absorbed by the state can be redistributed among the remaining states. This change can be made to the example model’s transition matrix P as follows:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{3}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}.$$

(Note that the matrix was primitive before this change, since $p_{3,3} \neq 0$.) This change guarantees primitivity by making $p_{d,d} \neq 0$. The power method can be applied to yield the long-run probabilities Π . The entry for the dummy state is dropped from Π , and

Table 2.3: Transition sensitivities

From	To	Enter	A	B	C	Exit
A	B	-0.047 25	-0.094 49	0.378 0	-0.189 0	-0.047 26
A	C	0.047 25	0.094 49	-0.378 0	0.189 0	0.047 26
B	B	-0.163 9	-0.201 7	0.680 8	-0.151 3	-0.163 9
B	C	0.003 151	0.037 82	-0.182 8	0.138 7	0.003 151
B	Exit	0.080 67	0.064 54	-0.161 3	-0.064 54	0.080 67
C	A	-0.132 7	0.096 50	0.096 50	0.072 37	-0.132 7
C	Exit	0.132 7	-0.096 50	-0.096 50	-0.072 37	0.132 7

the vector is re-normalized to obtain the correct long-run probabilities. A Scilab procedure to perform this computation is given as algorithm 6. For the example model convergence to within 1×10^{-12} requires 98 steps.

2.3 Sensitivity analysis

A change in a transition probability impacts the occupancies of all probabilities downstream. Consider the example model and the exit arcs from [C]. If one re-directs some probability to make the probability of “e” or “f” 0.9, this has a significant effect on the model. Alternately, changing the probability of “a” to 0.9 has a very different effect.

For each pair of states i and j in the model such that $p_{i,j} \neq 0$, set $p_{i,j}$ to 0.95 and compute the long-run occupancies $\Pi^{(p_{i,j}=0.95)} = [\pi_k^{(p_{i,j}=0.95)}]$. Next set $p_{i,j}$ to 0.05 and again compute the long-run occupancies $\Pi^{(p_{i,j}=0.05)} = [\pi_k^{(p_{i,j}=0.05)}]$. The sensitivity of state k with respect to transition from state i to state j is defined as:

$$z_{i,j,k} = \frac{\pi_k^{(p_{i,j}=0.95)} - \pi_k^{(p_{i,j}=0.05)}}{0.90}.$$

The matrix of sensitivities is computed by algorithm 7. The sensitivities for the example model are given in table 2.3. If one wishes to direct more of the model’s flow to state [C], the table indicates that the best way to do so is to increase the probability associated with transition from state [A] to state [C]. The best way to increase the average length of a test case is to decrease the long-run occupancy of [Exit], and the sensitivities indicate that the best way to do that is to increase the probability associated with transition from state [B] to state [B].

2.4 Other long run statistics

The long-run occupancy for transition from state i to state j is obtained as $\pi_i p_{i,j}$. The long-run occupancy for a particular stimulus σ_k can then be obtained by summing the

Algorithm 6 Compute an approximation to the Perron eigenvector via the power method

```

// Compute the Perron eigenvector for the
// stochastic matrix P and return it. The
// computation is performed using the power
// method.
//
// y: an approximation of the Perron eigenvector
// step: the number of steps required
// P: a stochastic matrix
// g: (optional) an initial guess
function [y,step]=get_pi_approx(P,g),
    // Add a dummy state to ensure primitivity.
    n=size(P,1);
    d=n+1;
    P(n,d)=1; P(d,d)=1/2; P(d,1)=1/2; P(n,1)=0;

    // Create an initial guess unless one
    // is given.
    if isdef('g') then
        // The initial guess must be adjusted to
        // make it the correct size. The dummy
        // state's occupancy will be twice the
        // sink's.
        g(1,d)=2*g(n); yold=g/sum(g);
    else
        yold(1,1:d)=1/d;
    end;
    y=yold*P;

    // Count the number of steps and set the
    // error limit; the smaller the limit, the
    // better the approximation and the more
    // steps required.
    step=1; el=1e-12;

    // Run until the error limit is reached.
    while sum(abs(yold-y)) > el,
        step=step+1;
        yold=y; y=y*P;
    end;

    // Remove the dummy and normalize.
    y=y(1,1:n); y=y/sum(y);
endfunction;

```

Algorithm 7 Compute the matrix of sensitivities

```

// Compute the matrix of arc sensitivities.
// The first column of the returned matrix
// is the source state, the second column
// is the target state. The remaining
// columns are the changes in the
// occupancies.
//
// Z: the matrix of arc sensitivities.
// P: the state transition matrix.
// pi: the pi vector, if available.
function [Z]=get_sensitivities(P,pi),
    // Get the size of the matrices.
    n=size(P,1);
    // Get the baseline pi vector.
    if argn(2)<2 then
        pi=get_pi(P);
    end;
    // Loop over all the arcs.
    m=1;
    for i=1:n,
        for j=1:n,
            // If there is an uncertain
            // transition from state i to
            // state j, compute the
            // associated sensitivities.
            if P(i,j) <> 0 & P(i,j) <> 1,
                // Modify the transition.
                x=1-P(i,j); t=P(i,1:n);
                P(i,1:n)=t/x*0.05; P(i,j)=0.95;
                // Compute pi.
                ph=get_pi_approx(P,pi);
                // Modify the transition.
                P(i,1:n)=t/x*0.95; P(i,j)=0.05;
                // Compute pi.
                pl=get_pi_approx(P,pi);
                // Compute the sensitivities.
                Z(m,1:2)=[i,j];
                Z(m,3:n+2)=(ph-pl)/0.9; m=m+1;
                // Restore the transition.
                P(i,1:n)=t;
            end;
        end;
    end;
endfunction;

```

Algorithm 8 Compute the stimulus long-run occupancy

```

// Compute the stimulus long run occupancies.
//
// sigma: the stimulus long-run occupancies.
// S: the stimulus matrix.
// pi: the pi vector.
function [sigma]=get_sigma(S,pi),
    // Compute the normalization factor.
    n=size(S,1);
    m=1/(1-pi(n+1));
    p2=pi(1,1:n);

    // Compute the sum for each element.
    sigma=p2*S*m;
endfunction;

```

long run occupancies for all arcs labeled with the stimulus:

$$\sigma_k = \frac{1}{1 - \pi_n} \sum_{i=1}^{n-1} \pi_i s_{i,k}. \quad (2.8)$$

The factor $1/(1 - \pi_n)$ in eq. 2.8 is used to remove the sink from the computation and re-normalize the vector. This is needed because no stimulus labels the recurrence arc from the sink to the source. Algorithm 8 computes the stimulus occupancies given the state long-run occupancies and the stimulus matrix. The example model's stimulus matrix can be expressed in Scilab as follows:

```

s = [
    1  0  0  0  0
    0 1/2 1/2 0  0
    0 1/2 1/4 1/4 0
    1/4 0  0  1/2 1/4]

```

The stimulus occupancies can be computed using the following command:

```
get_sigma(s,get_pi(p))
```

The expected number of test cases, in the long run, between occurrences of state s_j is:

$$\frac{\pi_n}{\pi_j}.$$

The expected number of occurrences of state j in a single test case is:

$$E[n_{1,j}] = \frac{\pi_j}{\pi_n}. \quad (2.9)$$

Table 2.4: Probability of occurrence for each state

Enter	Probability of Occurrence
Enter	1.000
A	1.000
B	0.571 4
C	0.750 0
Exit	1.000

Note that eq. 2.9 is actually the same as eq. 2.7, once one notes that another way to compute the expected length of a test case is:

$$E[l] = \frac{1}{\pi_n} - 1.$$

The expected number of occurrences of a state transition from state i to state j during a test case is:

$$\frac{\pi_i p_{i,j}}{\pi_n}.$$

These results are all very easy to obtain from the Π vector.

2.5 Probability of occurrence for states

Let $y_{i,j}$ denote the probability that one visits state j prior to absorption, given that one starts in state i . Given one reaches state j , one re-visits it $E[n_{j,j}]$ times, on average. Thus $y_{i,j}E[n_{j,j}]$ is the average number of times one visits state j , given that one starts in state i . This gives the simple relation:

$$\begin{aligned} y_{i,j}E[n_{j,j}] &= E[n_{i,j}] \\ y_{i,j} &= \frac{E[n_{i,j}]}{E[n_{j,j}]} \end{aligned} \quad (2.10)$$

The matrix of node probabilities is computed by algorithm 2.10. The node probabilities for the example model, starting from the source, are given in table 2.4.

An alternate approach, which will be useful later, works as follows. Assume the sink and state j are made absorbing (that is, $p_{n,n} = p_{j,j} = 1$). The probability that one is absorbed in state j , given one starts in state i is $y_{i,j}$. Starting in state i , one may be absorbed into state j in one step with probability $p_{i,j}$, or after moving to a non-absorbing state k with probability $p_{i,k}y_{k,j}$. These are mutually-exclusive possibilities, so they may be summed to give:

$$y_{i,j} = p_{i,j} + \sum_{k \notin \{j,n\}} p_{i,k}y_{k,j}. \quad (2.11)$$

Algorithm 9 Compute the probability of occurrence for each state

```

// Compute probability of occurrence
// of a state in a realization.
//
// Y: the node probability matrix.
// P: the transition matrix.
// N: the fundamental matrix.
function [Y]=get_node_probability(P,N),
    // Compute the fundamental matrix.
    if argn(2)<2 then
        [N,V]=get_nfe(P);
    end;

    // Compute the matrix.
    Y=N*(1/diag(diag(N)));
endfunction;

```

2.6 Probability of occurrence for arcs

Consider the probability of occurrence of an arc in a usage model. This value can be computed by introducing a new, intermediate state on the arc of interest, thus splitting the arc into two arcs. The probability of occurrence of the new state can then be computed, and this will be equal to the probability of occurrence of the original arc. This approach has been proposed by Sayre [13], but no closed-form solution was given (other than re-applying the method of 2.1 to the modified model, thus requiring a matrix inversion for every arc). A computationally simpler approach will be derived here.

Assume the arc of interest x originates at state a and terminates at state b and let the arc's probability be p_x . Let the new intermediate state on arc x be state α . The modified transition matrix will be denoted $\hat{P} = [\hat{p}_{i,j}]$. For the modified matrix $\hat{p}_{a,\alpha} = p_x$ and $\hat{p}_{\alpha,b} = 1$. Because the new state and its arcs replace arc x , $\hat{p}_{a,b} = p_{a,b} - p_x$. Introduction of the new state will have no other effect on the transition matrix. The probability of occurrence for α can be computed using eq. 2.11:

$$y_{i,\alpha} = p_{i,\alpha} + \sum_{k \notin \{\alpha,n\}} \hat{p}_{i,k} y_{k,\alpha}. \quad (2.12)$$

Let \hat{Q} denote the usual Q matrix with the change that $\hat{q}_{a,b} = \hat{p}_{a,b} = p_{a,b} - p_x = q_{a,b} - p_x$. Note that $\hat{Q}Y_\alpha$ differs from QY_α by only one element:

$$\hat{q}_{a,b} y_{b,\alpha} = (q_{a,b} - p_x) y_{b,\alpha} = q_{a,b} y_{b,\alpha} - p_x y_{b,\alpha}.$$

It will be convenient to note that $\hat{Q}Y_\alpha = QY_\alpha - C$ for a column vector C defined by $c_a = p_x y_{b,\alpha}$ and $c_i = 0$ for all $i \neq a$. Finally, note that $p_{i,\alpha} \neq 0$ only when $i = a$; thus let $X = [x_i]$ be a column vector for which $x_a = p_x$ and $x_i = 0$ otherwise. Eq. 2.12 can

now be re-written in terms of the vectors Y_α and X and the matrix \hat{Q} . Observe that $(I - Q)^{-1} = N$:

$$\begin{aligned}
Y_\alpha &= X + \hat{Q}Y_\alpha \\
Y_\alpha &= X + QY_\alpha - C \\
Y_\alpha - QY_\alpha &= X - C \\
(I - Q)Y_\alpha &= X - C \\
Y_\alpha &= (I - Q)^{-1}(X - C) \\
Y_\alpha &= N(X - C).
\end{aligned} \tag{2.13}$$

While N and X are known, neither Y_α nor C is yet known. Re-expressing eq. 2.13 in terms of elements of the Y_α vector gives:

$$y_{i,\alpha} = \sum_{k \notin \{\alpha, n\}} E[n_{i,k}] (x_k - c_k). \tag{2.14}$$

The rightmost term in the sum is zero except when $k = a$. Since $x_k = p_x$, eq. 2.14 can be re-written as:

$$\begin{aligned}
y_{i,\alpha} &= E[n_{i,a}] (p_x - c_a) \\
&= E[n_{i,a}] (p_x - p_x y_{b,\alpha}).
\end{aligned} \tag{2.15}$$

Eq. 2.15 contains two unknowns: $y_{i,\alpha}$ and $y_{b,\alpha}$. Consider $y_{b,\alpha}$:

$$\begin{aligned}
y_{b,\alpha} &= E[n_{b,a}] (p_x - p_x y_{b,\alpha}) \\
y_{b,\alpha} &= E[n_{b,a}] p_x - E[n_{b,a}] p_x y_{b,\alpha} \\
y_{b,\alpha} + E[n_{b,a}] p_x y_{b,\alpha} &= E[n_{b,a}] p_x \\
y_{b,\alpha} (1 + E[n_{b,a}] p_x) &= E[n_{b,a}] p_x \\
y_{b,\alpha} &= \frac{E[n_{b,a}] p_x}{1 + E[n_{b,a}] p_x}.
\end{aligned} \tag{2.16}$$

Now eq. 2.16 for $y_{b,\alpha}$ can be substituted back into eq. 2.15 for $y_{i,\alpha}$:

$$\begin{aligned}
y_{i,\alpha} &= E[n_{i,a}] (p_x - p_x y_{b,\alpha}) \\
&= E[n_{i,a}] \left(p_x - p_x \frac{E[n_{b,a}] p_x}{1 + E[n_{b,a}] p_x} \right) \\
&= E[n_{i,a}] p_x \left(1 - \frac{E[n_{b,a}] p_x}{1 + E[n_{b,a}] p_x} \right) \\
&= E[n_{i,a}] p_x \left(\frac{1 + E[n_{b,a}] p_x}{1 + E[n_{b,a}] p_x} - \frac{E[n_{b,a}] p_x}{1 + E[n_{b,a}] p_x} \right) \\
&= E[n_{i,a}] p_x \left(\frac{1}{1 + E[n_{b,a}] p_x} \right) \\
&= \frac{E[n_{i,a}] p_x}{1 + E[n_{b,a}] p_x}.
\end{aligned} \tag{2.17}$$

Algorithm 10 Compute the probability of occurrence for each arc

```

// Compute the probability of occurrence for
// each arc.
//
// Z: arc probabilities.
// P: the transition matrix.
// N: the fundamental matrix, if available.
function [Z]=get_arc_prob(P,N),
    // Get the matrix size.
    n=size(P,1);

    // Compute the fundamental matrix.
    if argn(2)<2 then
        N=get_nfe(P);
    end;

    // Compute the matrix of arc probabilities.
    Z(1:n-1,1:n)=0;
    for i=1:n-1,
        for j=1:n,
            Z(i,j)=N(1,i)*P(i,j);
            if j<>n then
                Z(i,j)=Z(i,j)/(1+N(j,i)*P(i,j));
            end;
        end;
    end;
endfunction;

```

In eq. 2.17 all elements are known, and the probability of the arc can be computed using the original fundamental matrix and the transition matrix. Note that if b is the sink, then $E[n_{b,a}] = 0$ (since there can be no outgoing arcs from the sink), where $b \neq a$.

Consider the probability of the arc from i to j given that one starts in the source. Denote this probability by $z_{i,j}$:

$$z_{i,j} = \frac{E[n_{1,i}]p_{i,j}}{1 + E[n_{j,i}]p_{i,j}}. \quad (2.18)$$

Eq. 2.18 is used in algorithm 10 to generate the matrix of arc probabilities, given the transition matrix and possibly the fundamental matrix. The arc probabilities for the example model are given in table 2.5.

Table 2.5: Probability of occurrence for each arc

From	To	Probability
Enter	A	1.000
A	B	0.571 4
A	C	0.533 3
B	B	0.296 3
B	C	0.285 7
B	Exit	0.307 7
C	A	0.187 5
C	Exit	0.692 3

2.7 Probability of occurrence for stimuli

Let $Y = [y_{i,k}]$ be the matrix whose entry $y_{i,k}$ is the probability that, given one starts in state i , one visits an arc labeled with stimulus k prior to reaching the sink. This can be expressed as follows:

$$y_{i,k} = s_{i,k} + \sum_{j \neq n} \sum_{l \neq k} p_{i,j,l} y_{j,k}.$$

The inner summation is used to avoid traversing arcs labeled with k . Since the model is deterministic:

$$y_{i,k} = s_{i,k} + \sum_{j \neq n} (p_{i,j} - p_{i,j,k}) y_{j,k}.$$

The difference $p_{i,j} - p_{i,j,k}$ can be quickly obtained by taking $Q - Q|_k$. This is convenient with respect to the system of equations, since Q does not contain the sink.

Holding k constant over the above equation allows it to be re-written in terms of the vectors Y_k and S_k , which are just the k th columns of the corresponding matrices:

$$\begin{aligned} Y_k &= S_k + (Q - Q|_k) Y_k \\ Y_k - (Q - Q|_k) Y_k &= S_k \\ (I - (Q - Q|_k)) Y_k &= S_k \\ Y_k &= (I - Q + Q|_k)^{-1} S_k. \end{aligned} \tag{2.19}$$

It is now possible to solve for the probability with which each stimulus occurs, given that one starts in a given state. Note that solving for all stimulus probabilities requires s matrix inversions.

2.8 First passage times

Let $m_{i,j}$ be the number of state transitions, starting in state i , until the first occurrence of state j . Let $M = [E[m_{i,j}]]$ be the matrix whose entries are the mean number of state

Table 2.6: Mean first passage times and associated variances (events) for each state

State	Mean First Passage (events)	Variance (events)
Enter	5.385	4.095
A	1.000	0.000
B	5.500	25.25
C	4.333	13.33
Exit	4.385	4.095

transitions until state j occurs given that the process started in state i . Again using the method of first passage, these numbers are the solution to the system of equations:

$$\begin{aligned}
 E[m_{i,j}] &= 1 + \sum_{k \neq j} p_{i,k} E[m_{k,j}] \\
 E[m_{i,j}] &= 1 + \sum_{k=1}^n p_{i,k} E[m_{k,j}] - p_{i,j} E[m_{j,j}].
 \end{aligned} \tag{2.20}$$

Eq. 2.20 can be re-written in matrix vector form as follows:

$$M = U + PM - PM_d. \tag{2.21}$$

Let $A = [a_{i,j}]$ be the matrix for which $a_{i,j} = \pi_j$. That is, $\lim_{n \rightarrow \infty} P^n = A$ (provided P is not periodic). The expected number of events between occurrences of state j is $E[m_{jj}] = \frac{1}{\pi_j}$, so $M_d = 1/A_d$. Let matrix Z can be computed as:

$$Z = (I - (P - A))^{-1}.$$

In [6] the following equation is shown to solve eq. 2.21:

$$M = (I - Z + UZ_d)M_d$$

As in the previous section, $E^2[m_{i,j}]$ is easy to compute but $E[m_{i,j}^2]$ is more difficult. Let $M^{(2)} = [E[m_{i,j}^2]]$ be the matrix of second moments. [6] derives the following equation for $M^{(2)}$:

$$M^{(2)} = M(2Z_d M_d - I) + 2(ZM - E(ZM)_d)$$

These equations are implemented by the Scilab procedure in algorithm 11. The results for the example model are presented in table 2.6.

Note that $E[m_{1,n}]$ is the expected number of occurrences until the sink is visited, given that one starts in the source. Thus $E[m_{1,n}] = E[l]$, and further $\text{Var}[m_{1,n}] = \text{Var}[l]$. This allows one to compute the variance of the expected test case length.

The mean first passage times in terms of events from any starting state may not be the most useful metric for Markov chain usage models. A more useful statistic is the

Algorithm 11 Compute the mean first passage times and their variances

```

// Compute the mean first passage times
// and their variances for a stochastic
// matrix P.
//
// M: the mean first passage times.
// V: the variances of the first passage times.
// P: a stochastic matrix.
// pi: the long-run occupancies, if available.
function [M,V]=get_mfp_events(P,pi),
    // Compute the limit matrix A.
    if argn(2)<2 then
        pi=get_pi(P);
    end;
    n=size(P,1);
    for i=1:n, A(i,1:n)=pi; end;
    // Compute some "helper" matrices.
    I=eye(n,n); Ad=1/diag(diag(A)); E(1:n,1:n)=1;
    // Compute the fundamental matrix, then M.
    Z=inv(I-(P-A)); Zd=diag(diag(Z));
    M=(I-Z+E*Zd)*Ad;
    // Compute the variances.
    M2=M*(2*Zd*Ad-I)+2*(Z*M-E*diag(diag(Z*M)));
    V=M2-(M.*M);
endfunction;

```

number of test cases until a state first appears. The probability that state i occurs in a randomly-selected test case is $y_{1,i}$, which can be assumed to be greater than zero (since the model is connected). Then the probability that state i is visited for the first time in exactly $k \geq 1$ tests is:

$$(1 - y_{1,i})^{k-1} y_{1,i}.$$

Let m_i be the number of test cases until state i first appears. Applying the definition of expectation gives:

$$E[m_i] = \sum_{k=1}^{\infty} k(1 - y_{1,i})^{k-1} y_{1,i}. \quad (2.22)$$

This is a well-known series [3, series 3], and it converges when $1 - y_{1,i} < 1$. Since $0 < y_{1,i} \leq 1$ for every state i , the series converges and one obtains:

$$E[m_i] = \frac{1}{y_{1,i}}.$$

This gives a quick way to obtain a mean first passage metric from the fundamental matrix. The variance can be computed similarly. Applying the definition of variance gives another well-known series [3, series 1113]. Assume $0 < y_{1,i} < 1$:

$$\begin{aligned} \text{Var}[m_i] &= E[m_i^2] - E^2[m_i] \\ &= \sum_{k=1}^{\infty} k^2 (1 - y_{1,i})^{k-1} y_{1,i} - \frac{1}{y_{1,i}^2} \\ &= \frac{y_{1,i}}{1 - y_{1,i}} \sum_{k=1}^{\infty} k^2 (1 - y_{1,i})^k - \frac{1}{y_{1,i}^2} \\ &= \left(\frac{y_{1,i}}{1 - y_{1,i}} \right) \left(\frac{(1 - y_{1,i}) + (1 - y_{1,i})^2}{(1 - (1 - y_{1,i}))^3} \right) - \frac{1}{y_{1,i}^2} \\ &= (y_{1,i}) \left(\frac{1 + (1 - y_{1,i})}{y_{1,i}^3} \right) - \frac{1}{y_{1,i}^2} \\ &= \frac{1 + (1 - y_{1,i})}{y_{1,i}^2} - \frac{1}{y_{1,i}^2} \\ &= \frac{1 - y_{1,i}}{y_{1,i}^2}. \end{aligned}$$

Next, consider the restriction $y_{1,i} < 1$. If $y_{1,i} = 1$ then the variance computation gives zero, which happens to be correct, so the restriction (required by the derivation) can be removed and the above formula applied whenever $0 < y_{1,i} \leq 1$. These computations are implemented in algorithm 12. The results for the example model are presented in table 2.7.

2.9 Number of occurrences of a stimulus in a test case

Once one has the expected number of occurrences of each arc in a model, one can compute the expected number of occurrences of each stimulus.

Table 2.7: Mean first passage times and associated variances (test cases) for each state

State	Mean First Passage (test cases)	Variance (test cases)
Enter	1.000	0.000
A	1.000	0.000
B	1.750	1.313
C	1.333	0.444 4
Exit	1.000	0.000

Algorithm 12 Compute mean first passage in terms of test cases

```

// Compute the vector of mean first passage times
// and their variances in units of test cases.
//
// M: the mean first passage times.
// V: variances of the mean first passage times.
// P: the transition matrix.
// N: the fundamental matrix, if available.
function [M,V]=get_mfp(P,N),
    // Get the matrix size.
    n=size(P,1)-1;

    // Compute the fundamental matrix.
    if argn(2)<2 then
        N=get_nfe(P);
    end;

    // Get the mean first passage vector.
    for i=1:n,
        y=N(1,i)/N(i,i);
        M(i)=1/y;
        V(i)=(1-y)/y^2;
    end;
endfunction;

```

Table 2.8: Stimulus expectations

Stimulus	Expected Occurrence (visits / test case)
a	1.231
b	1.231
c	0.923 1
e	0.769 2
f	0.230 8

Let X_i be a random variable counting the number of occurrences of arc i in a test case. Let Y_j be a random variable counting the number of occurrences of a particular stimulus j in a test case, and let A_j be the set of arcs which are labeled with stimulus j . Then Y_j is defined in terms of the X_i as follows:

$$Y_j = \sum_{i \in A_j} X_i.$$

Based on this simple definition, the expectation for the new random variable is as follows:

$$E[Y_j] = E \left[\sum_{i \in A_j} X_i \right] = E[X_1] + E[X_2] + \cdots + E[X_k].$$

Thus the expected number of occurrences of the stimulus j is the sum of the expectations of the arcs labeled with j .

Given a state occurrence expectation matrix N and the stimulus matrix S , one can compute the stimulus expectation matrix T as follows:

$$t_{i,j} = \sum_{k=1}^n E[n_{i,k}] s_{k,j}.$$

In matrix terms this can be expressed simply as $T = NS$. The stimulus expectation matrix is computed by algorithm 13.

As with the state expectations, one is primarily concerned with the first row of the stimulus expectation matrix. The results for the example model are given in 2.8.

Computation of the variance for this expectation is complicated by the fact that it requires covariances. An upper bound can be obtained using eq. 1.4.

Algorithm 13 Compute the stimulus expectation matrix

```
// Compute the terminal (stimulus) expectation
// matrix given the state transition matrix
// and the stimulus matrix.
//
// T: the stimulus expectation matrix.
// P: the state transition matrix.
// S: the stimulus matrix.
// N: the fundamental matrix, if available.
function [T]=get_te(P,S,N),
    // Compute the fundamental matrix.
    if argn(2)<3 then
        N=get_nte(P);
    end;

    // Compute the terminal expectations.
    T=N*S;
endfunction;
```

Chapter 3

Information Theory

The field of information theory can be applied to yield additional results about Markov chain usage models. This chapter discusses three such results: the entropy, the number of statistically typical sequences, and the discrimination. Alternative derivations for some of the results presented here are included in appendix B.

In this chapter it will be useful to refer to logarithms of base 2, and the shorthand $\lg x$ will be used for $\log_2 x$. Note that:

$$\begin{aligned}\log_a x &= \log_a b^{\log_b x} \\ &= (\log_b x)(\log_a b)\end{aligned}$$

which results in the relationship:

$$\frac{\log_a x}{\log_a b} = \log_b x.$$

Therefore, to compute logarithms base 2, one could use $\lg x = \ln x / \ln 2$.

3.1 Entropy

Entropy is a measure of uncertainty. The greater the entropy of a process, the more uncertain the outcome. Entropy may also be thought of as the minimum average¹ number of “yes or no” questions required to determine the result of one observation of a random variable.

If one observes outcome $x \in X$ with probability $p(x)$, then $-\lg p(x)$ bits are required to encode this result. Since this result is seen with probability $p(x)$, the expected number of bits required to encode a single observation of X is:

¹The term *minimum average* may seem like an oxymoron, but consider playing a simple guessing game. Your opponent chooses a secret number between one and ten. For each guess you are told whether the secret number is higher, lower, or equal to your guess. You can guess the number with four guesses, and this is the best you can do on average; it is the minimum average.

$$H\{X\} = - \sum_{x \in X} p(x) \lg p(x)$$

which is called the *entropy* of X , and denoted $H\{X\}$.

The *joint* entropy of two random variables X and Y is the minimum average information needed to encode the result of the joint experiment of observing both an outcome of X and an outcome of Y . This joint entropy can be expressed using the previous definition as:

$$H\{X\&Y\} = - \sum_{x \in X} \sum_{y \in Y} p(x\&y) \lg(p(x\&y)).$$

Note that if X and Y are independent random variables, one has $p(x\&y) = p(x)p(y)$.

The *conditional* entropy of one random variable Y given another random variable X is the average amount of information needed to specify a particular observation of Y given that one already has an observation of X . This can be expressed as:

$$H\{Y|X\} = - \sum_{x \in X} \sum_{y \in Y} p(x\&y) \lg(p(y|x)).$$

The conditional probability can be re-expressed using the relationship:

$$\begin{aligned} p(y|x)p(x) &= p(x\&y) \\ p(y|x) &= \frac{p(x\&y)}{p(x)}. \end{aligned}$$

If and only if X and Y are independent, then one has $p(x\&y) = p(x)p(y)$, and $p(y|x) = p(y)$. In this case, the conditional entropy can be reduced to:

$$\begin{aligned} H\{Y|X\} &= - \sum_{x \in X} \sum_{y \in Y} p(x\&y) \lg(p(y|x)) \\ &= - \sum_{x \in X} \sum_{y \in Y} p(x)p(y) \lg(p(y)) \\ &= - \sum_{x \in X} p(x) \sum_{y \in Y} p(y) \lg(p(y)) \\ &= - \sum_{x \in X} p(x) (-H\{Y\}) \\ &= H\{Y\}. \end{aligned}$$

In other words if the two random variables are independent, observing X reveals no information about the outcome of Y .

Consider that one is in state i of a Markov chain. The probability associated with choosing next state j is $p_{i,j}$, and one could express the uncertainty of choosing the next state as:

$$H_i = - \sum_{j=1}^n p_{i,j} \lg p_{i,j}. \quad (3.1)$$

(The sum in eq. 3.1 must actually be over only those terms for which $p_{i,j} \neq 0$, since $\lg 0$ is undefined.) The quantity H_i is the *state entropy* for state i .

In the long run the probability of being in state i is π_i . Thus the expected value of the state entropy denoted simply H is:

$$\begin{aligned} H &= \sum_{i=1}^n \pi_i H_i \\ &= - \sum_{i=1}^n \pi_i \sum_{j=1}^n p_{i,j} \lg p_{i,j}. \end{aligned} \quad (3.2)$$

(Again, the sum in eq. 3.2 must be over only those terms which are non-zero.) This expected state entropy is the average number of bits one needs to encode the observation of the next state, and is called the Markov chain's (*transition*) *source entropy*, H .

The usual source entropy reveals the amount of uncertainty associated with choosing the next state. For a Markov chain usage model it may be more appropriate to determine the uncertainty associated with choosing the next state and stimulus pair. In the example model if one is in state [C] and the next state is [Exit], there is still some uncertainty associated with choosing one of the two arcs from the state [C] to state [Exit]. This uncertainty is not taken into account by the usual source entropy.

Given that one is in state i , the probability that one chooses next stimulus j is $s_{i,j}$. The entropy associated with choosing the next stimulus from state i is thus:

$$H_i = - \sum_{j=1}^n s_{i,j} \lg s_{i,j}.$$

Again one may compute the expected value of this uncertainty and obtain the *stimulus* (*source*) *entropy*:

$$H_S = - \sum_{i=1}^n \pi_i \sum_{j=1}^s s_{i,j} \lg s_{i,j}.$$

If there is at most one arc between each pair of states, then $n = s$, $P = S$, and thus $H_S = H$. Otherwise there will be more arcs and the $s_{i,j}$ will be smaller than the $p_{i,j}$. Thus there will be increased uncertainty with each choice, and one has $H_S \geq H$ with equality if and only if there is at most one arc between each pair of states. These two entropies are computed by algorithm 14.

The state transition source entropy for the example model is approximately 0.7105 bits, and the stimulus source entropy for the example model is approximately 0.8286 bits.

3.2 Statistically-typical sequences

Consider an “average” sequence. In such a sequence state i would be visited approximately π_i/π_n times, since this is the expected number of occurrences of the state. This means that one would visit the arc from state i to state j approximately $\pi_i p_{i,j}/\pi_n$ times.

Every time one observes the transition from state i to state j , $-\lg p_{i,j}$ bits are required to encode this observation. The expected number of bits required to encode a

Algorithm 14 Compute source entropies

```

// Compute the source entropies of the given
// Markov chain in bits.
//
// H: state transition source entropy.
// G: stimulus transition source entropy.
// P: the transition matrix.
// S: the stimulus matrix.
// pi: the occupancies, if available.
function [H,G]=get_entropies(P,S,pi),
    // Get the size of the matrices.
    n=size(P,1); s=size(S,2);
    // The pi vector is needed.
    if argn(2)<3 then
        pi=get_pi(P);
    end;
    // Convert everything to log base 2.
    l2=log(2);
    // Compute the entropies. There is no need
    // to loop for the sink.
    H=0; G=0;
    for i=1:n-1,
        // Compute the state contribution to
        // the state entropy.
        p=0;
        for j=1:n,
            if P(i,j)>0 then
                p=p+P(i,j)*log(P(i,j))/l2;
            end;
        end;
        H=H-p*pi(i);
        // Compute the state contribution to
        // the stimulus entropy.
        p=0;
        for j=1:s,
            if S(i,j)>0 then
                p=p+S(i,j)*log(S(i,j))/l2;
            end;
        end;
        G=G-p*pi(i);
    end;
endfunction;

```

statistically-typical sequence is called the *trajectory entropy* H' :

$$\begin{aligned}
 H' &= \sum_{i=1}^n \sum_{j=1}^n \frac{\pi_i p_{i,j}}{\pi_n} (-\lg p_{i,j}) \\
 &= - \sum_{i=1}^n \frac{\pi_i}{\pi_n} \sum_{j=1}^n p_{i,j} \lg p_{i,j} \\
 &= \frac{1}{\pi_n} \sum_{i=1}^n \pi_i H_i \\
 &= \frac{H}{\pi_n}.
 \end{aligned}$$

One could apply the same reasoning with respect to next state, stimulus pair and obtain $H'_S = H_S/\pi_n$, with the relation $H'_S \geq H'$ since $H_S \geq H$.

Assume that one has k equally-likely outcomes, each with probability $1/k$. This distribution has the following entropy:

$$\begin{aligned}
 H\{P\} &= - \sum_{i=1}^k \frac{1}{k} \lg \frac{1}{k} \\
 &= -k \frac{1}{k} \lg \frac{1}{k} \\
 &= -\lg \frac{1}{k} \\
 &= -(\lg 1 - \lg k) \\
 &= \lg k.
 \end{aligned}$$

One can reverse this reasoning and say that l bits could encode up to 2^l equally-likely outcomes. In the case above, one obtains $2^{H\{P\}} = 2^{\lg k} = k$, as expected.

One would expect statistically-typical test cases to have some “average” likelihood of being generated. It therefore follows that if one has a trajectory entropy of H' , one has:

$$2^{H'} = 2^{\frac{H}{\pi_n}}$$

statistically-typical trajectories. This is the number of (approximately) equally-likely test cases whose ensemble statistics match the expectations for the chain. For the example chain the trajectory entropy is $H'_S = H_S/\pi_n = 0.8286 \text{ bits}/0.1857 = 4.462 \text{ bits}$. Given this result, there are consequently $2^{4.462 \text{ bits}} = 22.04 \approx 23$ statistically-typical test cases for the example model.

3.3 Discrimination

Let X be a random variable governed by the true distribution p . A statistical model is constructed to approximate p , and this model uses the approximating distribution q . It is reasonable to ask how closely the model resembles the true distribution. One

way to measure this is to compute the *relative entropy* of distribution p with respect to distribution q . This number can be thought of as the number of bits which are wasted by encoding observations of X using the not-quite-right distribution q .

Whenever outcome $x \in X$ is observed, it is encoded using $-\lg q(x)$ bits. This outcome is observed with the true probability $p(x)$, so on average $-\sum_{x \in X} p(x) \lg q(x)$ bits are used to encode an outcome of X . If the statistical model precisely matched the true distribution, the minimum average of $H\{X\} = -\sum_{x \in X} p(x) \lg p(x)$ bits would be used to encode the outcome. The number of bits wasted is the difference between the encoding used and the minimum:

$$\begin{aligned} & \left(-\sum_{x \in X} p(x) \lg q(x) \right) - \left(-\sum_{x \in X} p(x) \lg p(x) \right) \\ &= \sum_{x \in X} p(x) \lg p(x) - \sum_{x \in X} p(x) \lg q(x) \\ &= \sum_{x \in X} p(x) (\lg p(x) - \lg q(x)) \\ &= \sum_{x \in X} p(x) \lg \frac{p(x)}{q(x)}. \end{aligned}$$

This quantity is known as the *Kullback-Leibler number* or sometimes just the *discrimination*, and is denoted $K[p, q]$. Note that the discrimination is not a true metric: it is not symmetric.

The discrimination can be used to compare two Markov chain usage models. For example suppose one has a usage chain $U = [u_{i,j}]$ which is believed to represent the “true” use of the system and let $T = [t_{i,j}]$ be the testing chain state transition matrix representing the testing experience. The expected number of bits wasted by encoding U with the approximate T is the discrimination between the two stochastic processes:

$$K[U, T] = \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg \frac{u_{i,j}}{t_{i,j}}.$$

As the testing experience comes to more closely represent the expected use, the discrimination approaches zero. (As usual, the logarithm is only taken when $u_{i,j} \neq 0$.)

The discrimination based on stimuli is computed by algorithm 15. Assume the following set of test trajectories are given:

- a b b e,
- a c e,
- a b b b e, and
- a c a b b c f.

Then the matrix of stimulus executions is as follows:

$$E_t = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 0 \\ 0 & 4 & 1 & 2 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

and the vector of state visitations is:

$$E_s = \begin{bmatrix} 4 \\ 5 \\ 7 \\ 3 \\ 4 \end{bmatrix}.$$

This matrix and vector can be represented in Scilab as follows:

```
Et=[  
4 0 0 0 0  
0 3 2 0 0  
0 4 1 2 0  
1 0 0 1 1]
```

```
Es=[4; 5; 7; 3; 4]
```

The discrimination between this testing experience and the usage chain is approximately 0.034 41 bits.

Algorithm 15 Compute the perturbed discrimination

```

// Compute the Kullback-Leibler number
// (the discrimination).
//
// K: the Kullback-Leibler number.
// P: the state transition matrix.
// S: the stimulus matrix.
// Es: the state visitation vector.
// Et: the stimulus execution matrix.
// pi: the occupancies, if available.
function [K]=get_discrimination(P,S,Es,Et,pi),
    // Get the pi vector for the usage matrix.
    if argn(2)<5 then
        pi=get_pi(P);
    end;
    // Matrices must be of the same size.
    m=size(S,1); n=size(S,2);
    // Compute the log of 2 for use in converting
    // to base two later on.
    l2=log(2);

    // This will accumulate the discrimination.
    K=0;
    for i=1:m,
        // This is the partial sum from the inner
        // summation.
        k=0;
        for j=1:n,
            if S(i,j) > 0,
                // Normalize the stimulus execution.
                t=Es(i)/Et(i,j);
                k=k+S(i,j)*log(S(i,j)*t)/l2;
            end;
        end;
        K=K+pi(i)*k;
    end;
endfunction;

```

Chapter 4

Reliability Models

This chapter is concerned with statistical inference; specifically, the point estimation of a software system's reliability. This chapter presents some reliability models used with Markov chain usage models. In the following, assume that t tests are conducted, and that every test can be classified as a success or a failure. Let k be the number of failures; there are thus $t - k$ successful tests.

The *single-use* reliability is the long run probability that the software will not fail on a randomly-selected use (as defined by the usage model). Let R denote the true (unknown) reliability of the system. The goal is to produce an estimator, \hat{R} , of the reliability based on observations of software tests. The observations of the tests are random variables, and \hat{R} is a function of these observations. It follows that \hat{R} is also a random variable.

Underlying the idea of a single-use reliability is the concept of a *usage profile*. This is, informally, a statement of how the software will be used once it is delivered to the field. Under different usage profiles, a single software system may have different reliabilities. Here it is assumed that the software system's usage profiles are specified using Markov chain usage models in the following way: the probability of generating a particular path from source to sink in the usage model corresponds closely to the relative frequency with which the same sequence of events will occur in actual use.

Mean time to failure (MTTF) is the average number of uses between two failures. This quantity is related to the reliability as follows:

$$MTTF = \frac{1}{1 - R} \text{uses.}$$

The denominator is the “unreliability,” and the expectation is its reciprocal. This MTTF is in units of “uses.” If MTTF is desired in terms of “events,” which is more useful for some systems, multiply by the average number of events per test case. If the expected test case length is $E[l]$, then the following should work nicely:

$$MTTF_l = \frac{1 \text{ use}}{1 - R} \times E[l] \frac{\text{events}}{\text{use}} = \frac{E[l]}{1 - R} \text{events.}$$

Since \hat{R} is a random variable, it makes sense to discuss its expectation and variance.

Note that the variance for \hat{R} may not be as desirable as a different measure, called the *confidence* in the estimate, denoted $C[\hat{R}]$. The confidence is the probability that the true reliability R is at least the estimated reliability \hat{R} :

$$C[\hat{R}] = \Pr[R \geq \hat{R}].$$

Thus as the estimated reliability grows closer to the true reliability, and as the variance of the estimated reliability decreases, the confidence grows. If the distribution of the reliability is known, one can compute the confidence as the integral:

$$C[\hat{R}] = \int_{\hat{R}}^1 \Pr[R = r] dr.$$

Since the total area under the curve from zero to one must be one, an alternate computation is:

$$1 - C[\hat{R}] = \int_0^{\hat{R}} \Pr[R = r] dr. \quad (4.1)$$

Eq. 4.1 is often more useful since one of the limits of integration is zero.

4.1 Sample reliability

For the sample reliability, compute \hat{R} as:

$$\hat{R} = \frac{t-k}{t}.$$

Thus if one test is run, and it passes, the sample reliability is one. Likewise, if ten tests are run and all pass, then the reliability is again one. In the long run as randomly-selected tests are executed, the sample reliability \hat{R} will approach the true reliability R ; it is an unbiased estimator. Further, the sample variance will approach zero as the number of trials increases, and thus the estimator is consistent. The sample variance can be computed as:

$$\frac{(t-k)^2}{t} - \frac{(t-k)^2}{t^2} = \frac{(t-1)(t-k)^2}{t^2}.$$

The difference between one successful test and ten successful tests is the confidence in the process which produced the reliability estimate. The sample reliability does not take this into account; a better model is needed. Note that the Chebyshev inequality may be applied to bound the area under the reliability curve without knowledge of the distribution, but one can do much better if the distribution of the true reliability is known.

4.2 Binomial distribution

One can consider each test a Bernoulli trial; that is, an experiment with two possible outcomes (the software performs correctly, or the software fails) in which trials are

independent and identically distributed. If the probability of success is R , then the probability of failure is $1 - R$.

Having confidence in the testing process essentially means that one believes that if the software is unreliable, this unreliability will be detected by observing failures during testing. The goal is to limit the probability that the software is not as reliable as thought, but few failures are seen. Letting $C[\hat{R}]$ denote confidence gives the following relationship for confidence:

$$\Pr \left[\begin{array}{c} \text{observe } \leq k \text{ failures} \\ \text{in the } t \text{ trials} \end{array} \middle| R < \hat{R} \right] = (1 - C[\hat{R}]).$$

That is, one has confidence $C[\hat{R}]$ that the reliability is at least \hat{R} given n trials and no more than k failures observed. This is the probability that the reliability is at least R [11], but includes additional information about failures observed.

Since the trials are independent and identically distributed, the probabilities of the outcomes of the t trials can be multiplied to obtain the estimated probability of the particular sequence of outcomes: in this case $\hat{R}^{t-k}(1 - \hat{R})^k$. One can distribute k failures among t trials in:

$$\binom{t}{k}$$

ways (which is just the same as distributing $t - k$ successes over t trials). These different ways of distributing the failures are all mutually exclusive (only one can be observed), so they may be summed. The probability that one sees k failures in t trials is thus:

$$\binom{t}{k} \hat{R}^{t-k} (1 - \hat{R})^k.$$

The probability that one sees k or fewer failures (given estimated reliability \hat{R}) is thus:

$$(1 - C[\hat{R}]) = \sum_{i=0}^k \binom{t}{i} \hat{R}^{t-i} (1 - \hat{R})^i \quad (4.2)$$

since again only one of the mutually-exclusive outcomes can be observed. One can solve this numerically for t , and get the number of trials to certify with estimated reliability \hat{R} and confidence $C[\hat{R}]$. The above distribution is known as the *binomial* distribution.

With a simplifying assumption of zero failures, the equation reduces dramatically. Letting $k = 0$ in the above equation, gives:

$$(1 - C[\hat{R}]) = \hat{R}^t. \quad (4.3)$$

(Notice that the relationship between reliability and confidence is not linear.) This simple equation can be solved for the number of test cases. Taking \ln^1 of both sides

¹The logarithm base makes no difference:

$$\frac{\log_a(1 - C[\hat{R}])}{\log_a \hat{R}} = \frac{\log_a(1 - C[\hat{R}]) \log_b a}{\log_a \hat{R} \log_b a} = \frac{\log_a(1 - C[\hat{R}]) \log_b a}{\log_b a \log_a \hat{R}} = \frac{\log_b(1 - C[\hat{R}])}{\log_b \hat{R}}.$$

Table 4.1: Trials to achieve estimated reliability and confidence levels

Confidence Level	Estimated Reliability					
	0.90	0.95	0.99	0.999	0.999 9	0.999 99
0.90	22	29	44	66	88	110
0.95	45	59	90	135	180	225
0.99	230	299	459	688	917	1,146
0.999	2,302	2,995	4,603	6,905	9,206	11,508
0.999 9	23,025	29,956	46,050	69,075	92,099	115,124
0.999 99	230,258	299,572	460,515	690,773	921,030	1,151,287

gives:

$$\begin{aligned}\ln(1 - C[\hat{R}]) &= \ln \hat{R}^t \\ \ln(1 - C[\hat{R}]) &= t \ln \hat{R} \\ \frac{\ln(1 - C[\hat{R}])}{\ln \hat{R}} &= t\end{aligned}$$

Thus the minimum number of trials required to obtain the desired estimated reliability and confidence is given by the ceiling of the left hand side of the above equation (the smallest integer greater than the quantity).

$$t = \left\lceil \frac{\ln(1 - C[\hat{R}])}{\ln \hat{R}} \right\rceil.$$

Using this approach, it is possible to explore the relationship between estimated reliability and confidence. This can be done (for example) with the following Scilab commands:

```
def(' [t]=rc(r,c)', 't=ceil(log(1-c)/log(r))' )
x=[0.9,0.95,0.99,0.999,0.9999,0.99999]
for i=1:6, for j=1:6, t(i,j)=rc(x(i),x(j)); end; end
```

This will build the matrix for table 4.1. This table makes it clear that reliability and confidence grow differently. To see this in more detail, one can execute the following Scilab commands:

```
t=0; x=0; y=0
def(' [t]=rc(r,c)', 't=ceil(log(1-c)/log(r))' )
for i=1:9, for j=1:9,
    t(i,j)=rc(i/10,j/10);
end; end
for i=1:9, x(i)=i/10; y(i)=i/10; end
xbasc()
```

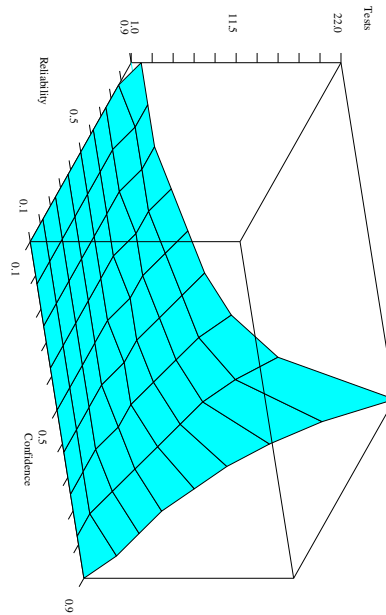


Figure 4.1: Reliability, confidence, and number of tests

```
plot3d(x,y,t,-117,5,...
"Confidence@Reliability@Tests")
```

These commands should generate a plot similar to figure 4.1.

The binomial model gives equal weight to each test, so 59 randomly-generated tests executed without failure certify a system at an estimated reliability of 0.95 and a confidence of 0.95, regardless of whether the 59 tests represent millions of events and days of testing, or only a few events per test and almost no testing time. The binomial model is *statistically conservative* in that it uses very little statistical information about each test.

4.3 Testing Markov chain

One can incorporate information about each test using the Markov chain. Consider again the tests given in section 3.3:

- a b b e,
- a c e,
- a b b b e, and

- a c a b b c f.

Assume these tests are executed and that the first two tests execute without failure. The third test fails on the first “b” event, but testing is able to proceed to completion. The last test fails on the first “b,” and testing is not able to proceed. One has the following matrix of *successful* executions:

$$E_s = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 \\ 1 & 0 & 0 & 3 & 0 \end{bmatrix}$$

and the following matrix of *failed* executions:

$$E_f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

One can duplicate the structure of the original usage model, and then keep track of the transitions on each arc. This new model is called the *testing Markov chain* [18]. Every time an arc is successfully traversed in testing, a counter on the arc is incremented. This is shown in figure 4.2.

If a failure which has not been encountered previously is encountered, one classifies the failure as a *stop* failure, meaning that one cannot execute the remainder of the test after the failure, or as a *continue* failure, meaning that execution of the test continues after the failure. The first failure (in the third example test case) is a continue failure, since the rest of the test is executed. The second failure (in the fourth example test case) is a stop failure, since the rest of the test is not executed.

For each unique failure encountered, a new *failure state* is introduced and the failed arc is duplicated to point to the new failure state. If the new failure was a continue failure, an arc is created from the failure state to the appropriate next state. If the failure was a stop failure, an arc is created from the failure state to the sink. In either case, the frequency count on the arcs is incremented. In figure 4.3 the continue failure is [f1], and the stop failure is [f2]. If a previously-encountered stop or continue failure is encountered again during testing, the frequency counts associated with the existing failure state are incremented.

After the testing experience has been captured in the testing Markov chain, one can normalize the frequency counts to obtain probabilities. For the example in figure 4.3, the following probability matrix results:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & \frac{2}{5} & \frac{1}{5} & \frac{1}{5} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

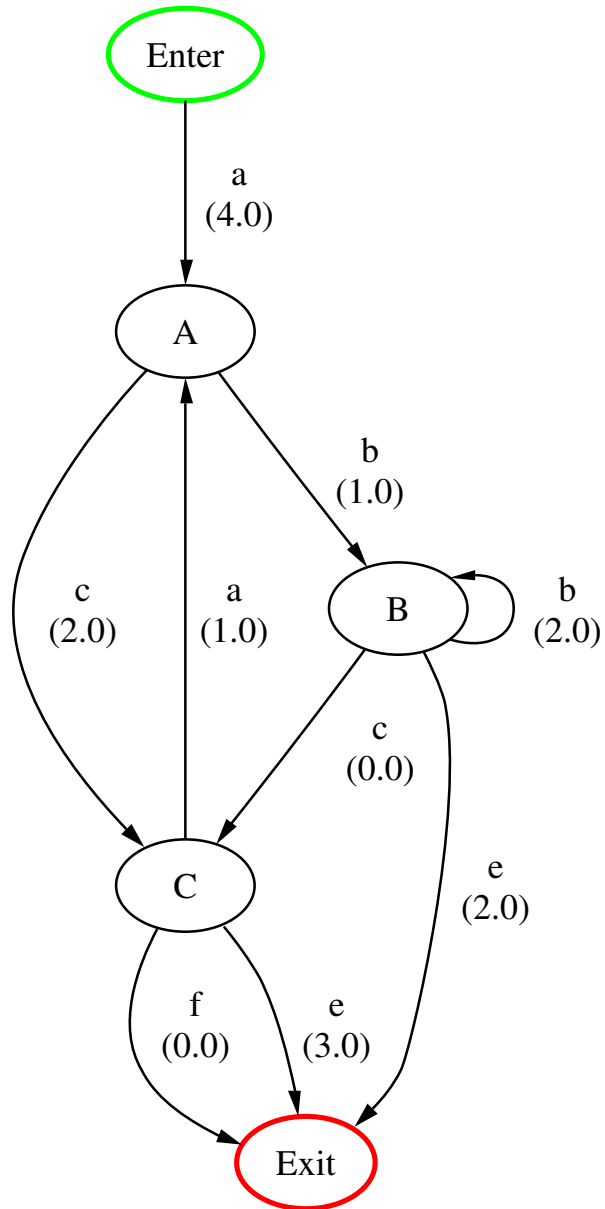


Figure 4.2: Usage model with successful traversal counts

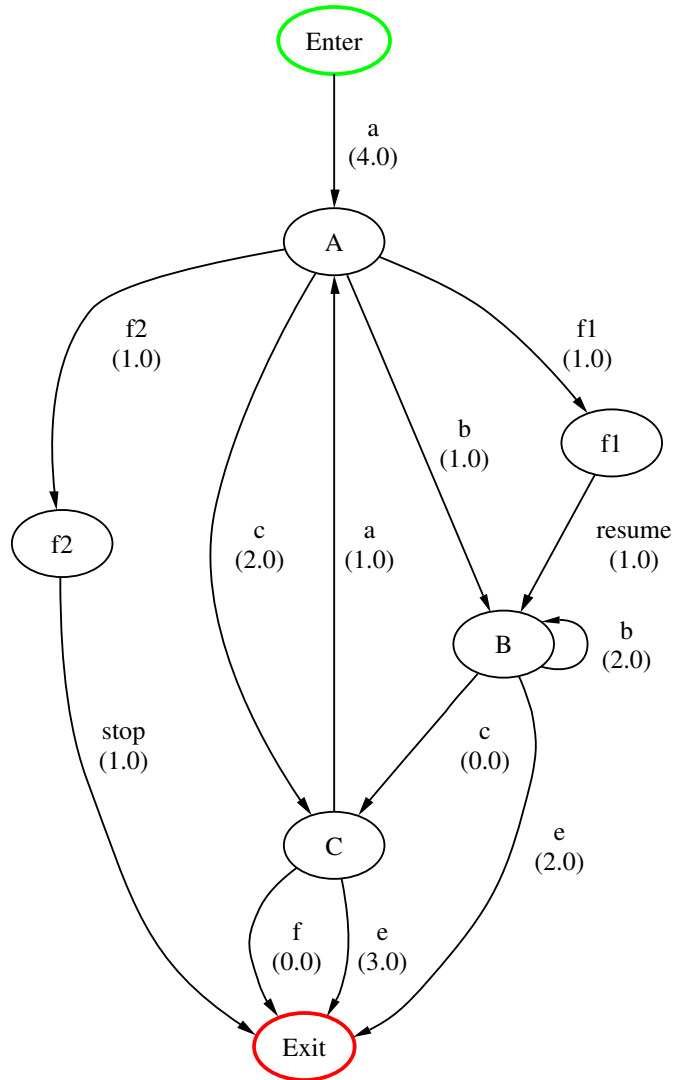


Figure 4.3: Usage model with failure states

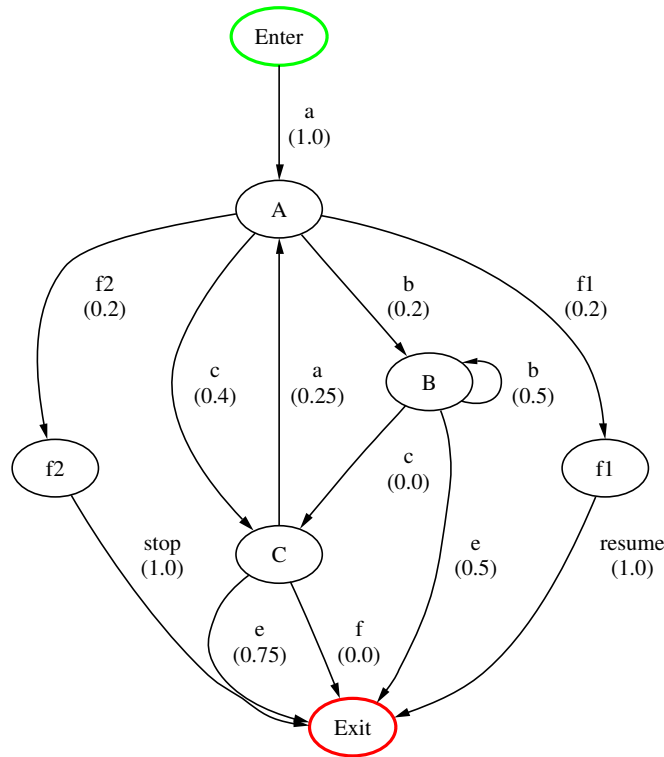


Figure 4.4: Testing Markov chain modified to compute reliability

Note that the failure states have been inserted *before* the sink state to keep it the last state in the matrix; many of the algorithms given here depend on this. This probability matrix can be analyzed to obtain the probability of occurrence of the failure states. Unfortunately, the failure states do not occur independently. For example, the trajectory a, b (with continue failure), c, a, b (with stop failure), visits both failure states.

The estimated probability that the software does not fail (encounter a failure state) on a randomly-selected use can be computed if occurrence of the failure states can be made independent. This can be done by redirecting the outgoing arcs from continue failure states to point to the model sink, as has been done in figure 4.4. After these changes are made, one can compute the probability of occurrence of each state via algorithm 9. Let there be m failure states, and let y_i be the probability of occurrence of failure state i . One now encounters the failure states independently, and the estimated software reliability can be computed as:

$$\hat{R} = 1 - \sum_{i=1}^m y_i$$

The matrix for the testing Markov chain can be expressed in Scilab as:

Table 4.2: Expectation and variance of failure states

State	Expectation (visits)	Variance (visits)
<i>f1</i>	0.222 2	0.172 8
<i>f2</i>	0.222 2	0.172 8

```
t=[
0 1 0 0 0 0 0
0 0 1/5 2/5 1/5 1/5 0
0 0 1/2 0 0 0 1/2
0 1/4 0 0 0 0 3/4
0 0 0 0 0 0 1
0 0 0 0 0 0 1
1 0 0 0 0 0 0 ]
```

For the example just given one obtains probabilities of occurrence of the failure states of $2/9$ each, giving a total probability of failure of $4/9$, and thus a reliability of $1 - 4/9 = 5/9$. If a confidence is desired, one can compute it using the binomial distribution of eq. 4.2. In this case, one has $\hat{R} = 5/9$, $t = 4$, and $k = 2$, giving a confidence of 0.765 9. The following Scilab commands can be used:

```
R=5/9; t=4; k=2;
bin=binomial((1-R),t); sum(bin(1:k+1))
```

The testing Markov chain model takes much more information into account than the binomial model, but it suffers from its own weaknesses. First, it does not immediately provide a means to compute a confidence. Second, it cannot render a judgment when testing reveals no failures. Consider the example, and assume that no failures were observed in the four test cases. One would then introduce no failure states, and the equation for reliability would yield $\hat{R} = 1$.

As an alternate means to compute the reliability using the testing Markov chain, assume that one takes the chain as it appears in figure 4.4 and computes the expected number of occurrences of each failure state in a test case. Since each failure state immediately passes to the sink, this value will also be the probability that the state is encountered in a test case. With the expected occurrence one can also compute a variance. These results are summarized for the example model in table 4.2.

Since the random variables (occurrence of [*f1*] and [*f2*]) are now independent in the modified testing chain, one can sum them and subtract from one to obtain the reliability of $5/9$. One can also sum the variances to obtain a composite variance of 0.345 7.

4.4 Beta distribution and the Miller model

One can often get a much more accurate estimator if the distribution is known. One possibility is the binomial distribution used previously. A very flexible approach is to use a distribution like the beta distribution, which uses two parameters to define a family of distributions. Further, the parameters of the beta distribution can be used to represent prior information, providing a way to use past testing or development history in certification. The reliability estimator based on the beta distribution will be referred to here as the *Miller model* [7].

Suppose a random variable X represents the number of successes in a binomial experiment (independent, identically-distributed trials with outcome either success or failure) with n trials and probability of success R . Consider trying to find an estimator \hat{R} of R based on X and given some prior information about R . Let this prior information be given by the beta distribution with parameters α and β , and the probability distribution function for $0 \leq R \leq 1$:

$$\Pr[R = r] = \mathcal{B}(r; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1}.$$

The expectation and variance for estimator \hat{R} can be computed as shown in appendix A:

$$E[\hat{R}] = \frac{\alpha}{\alpha + \beta},$$

$$\text{Var}[\hat{R}] = \frac{\alpha\beta}{(\alpha + 1 + \beta)(\alpha + \beta)^2}.$$

The parameters of the distribution can be split into prior information and observations. Take s and f to be the numbers of observed successes and observed failures, respectively. Then take $a - 1$ and $b - 1$ to be the numbers of prior successes and prior failures, respectively. These choices are such that, if there is no prior testing information, one has the “no information” priors $a = b = 1$. Then letting $\alpha = s + a$ and $\beta = f + b$ gives the expected reliability and variance:

$$E[\hat{R}] = \frac{s + a}{s + a + f + b}, \quad (4.4)$$

$$\text{Var}[\hat{R}] = \frac{(s + a)(f + b)}{(s + a + 1 + f + b)(s + a + f + b)^2}. \quad (4.5)$$

Eq. 4.4 and eq. 4.5 can be applied to each arc of the model using the success and failure counts for the arc. Let $r_{i,j,k}$ the reliability for the arc from state i to state j labeled with stimulus k . Then the expected single-event reliability \hat{R}_e is the long run, average probability of success for a randomly-selected event:

$$E[\hat{R}_e] = E \left[\sum_{i,j,k} \pi_i p_{i,j,k} r_{i,j,k} \right]$$

$$= \sum_{i,j,k} \pi_i p_{i,j,k} E[r_{i,j,k}].$$

The variance of the estimator is easy to compute given the expectations and variances of the arc reliabilities:

$$\begin{aligned}
 \text{Var}[\hat{R}_e] &= E \left[\sum_{i,j,k} \pi_i p_{i,j,k} r_{i,j,k}^2 \right] - E^2[\hat{R}_e] \\
 &= \sum_{i,j,k} \pi_i p_{i,j,k} E[r_{i,j,k}^2] - E^2[\hat{R}_e] \\
 &= \sum_{i,j,k} \pi_i p_{i,j,k} (\text{Var}[r_{i,j,k}] + E^2[r_{i,j,k}]) - E^2[\hat{R}_e] \\
 &= \sum_{i,j,k} \pi_i p_{i,j,k} \text{Var}[r_{i,j,k}] + \sum_{i,j,k} \pi_i p_{i,j,k} E^2[r_{i,j,k}] - E^2[\hat{R}_e].
 \end{aligned}$$

The derivation of a single-use reliability estimator based on estimates of the arc reliabilities is modestly complicated, and beyond the scope of this paper.

4.5 Confidence

The confidence $C[\hat{R}]$ in reliability estimate \hat{R} was defined by eq. 4.1. If the reliability is assumed to be governed by the beta distribution, as in section 4.4, then:

$$\begin{aligned}
 1 - C[\hat{R}] &= \int_0^{\hat{R}} \Pr[R = r] dr \\
 &= \int_0^{\hat{R}} \frac{r^{\alpha-1} (1-r)^{\beta-1}}{\text{B}(\alpha, \beta)} dr \\
 &= \frac{1}{\text{B}(\alpha, \beta)} \int_0^{\hat{R}} r^{\alpha-1} (1-r)^{\beta-1} dr
 \end{aligned} \tag{4.6}$$

At this point, eq. 4.6 can be simplified by assuming that β (the failure count) is one (the no-information case). This gives:

$$\begin{aligned}
 1 - C[\hat{R}] &= \frac{1}{\text{B}(\alpha, 1)} \int_0^{\hat{R}} r^{\alpha-1} dr \\
 &= \frac{\hat{R}^\alpha}{\alpha \text{B}(\alpha, 1)} \\
 &= \frac{\hat{R}^\alpha \Gamma(\alpha + 1)}{\alpha \Gamma(\alpha)} \\
 &= \frac{\hat{R}^\alpha \alpha \Gamma(\alpha)}{\alpha \Gamma(\alpha)} \\
 &= \hat{R}^\alpha.
 \end{aligned} \tag{4.7}$$

Note that eq. 4.7 is the same as eq. 4.3, obtained with the binomial distribution.

If $\beta = 1$ is not required, then the term $(1-r)^{\beta-1}$ must be dealt with. Replacing this term with its binomial expansion² gives:

$$\begin{aligned}
C[\hat{R}] &= 1 - \frac{1}{\mathbf{B}(\alpha, \beta)} \int_0^{\hat{R}} r^{\alpha-1} (1-r)^{\beta-1} dr \\
&= 1 - \frac{1}{\mathbf{B}(\alpha, \beta)} \int_0^{\hat{R}} r^{\alpha-1} \sum_{i=0}^{\beta-1} \binom{\beta-1}{i} (-1)^i r^i dr \\
&= 1 - \frac{1}{\mathbf{B}(\alpha, \beta)} \int_0^{\hat{R}} \sum_{i=0}^{\beta-1} \binom{\beta-1}{i} (-1)^i r^{\alpha-1+i} dr \\
&= 1 - \frac{1}{\mathbf{B}(\alpha, \beta)} \sum_{i=0}^{\beta-1} \binom{\beta-1}{i} (-1)^i \int_0^{\hat{R}} r^{\alpha-1+i} dr \\
&= 1 - \frac{1}{\mathbf{B}(\alpha, \beta)} \sum_{i=0}^{\beta-1} \binom{\beta-1}{i} (-1)^i \frac{\hat{R}^{\alpha+i}}{\alpha+i}. \tag{4.8}
\end{aligned}$$

As a modest check on eq. 4.8, note that setting $\beta = 1$ again results in eq. 4.7.

Consider the following term from eq. 4.8 with positive integer α and β :

$$\frac{1}{\mathbf{B}(\alpha, \beta)} \binom{\beta-1}{i} = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \left(\frac{(\beta-1)!}{i!(\beta-1-i)!} \right) = \frac{(\alpha+\beta-1)!}{(\alpha-1)!i!(\beta-1-i)!}.$$

The individual terms can become quite large. For example, Scilab can only accurately compute the gamma function up to $\Gamma(172)$. Scilab provides an alternate function, `gammaLn`, which computes $\ln\Gamma(n)$, and can thus use much greater values. For integer a the value of $a!$ can be computed as:

$$\ln a! = \ln \prod_{i=1}^a i = \sum_{i=1}^a \ln i.$$

Since the value of β will generally be small (indicating few failures with respect to total tests run), it is possible to compute the confidence by simply summing up the terms. A Scilab procedure to do so is given as algorithm 16.

Confidence levels versus the priors for fixed reliability 0.95 can be graphed as a surface in Scilab with the following commands:

```

for i=1:10, for j=1:10,
  c(i, j)=get_confidence(0.95, i, j);

```

²The binomial expansion of $(x+y)^n$ is:

$$(x+y)^n = \sum_{i=0}^n \binom{n}{i} x^i y^{n-i}.$$

When the second term is negative, this can be written as:

$$(x-y)^n = \sum_{i=0}^n \binom{n}{i} x^i (-1)^{n-i} y^{n-i}.$$

Algorithm 16 Compute the confidence for the Miller model

```

// Compute the confidence associated with
// priors a,b and reliability estimate R.
//
// C: the computed confidence.
// R: estimated reliability.
// a: prior successes, plus one.
// b: prior failures, plus one.
function [C]=get_confidence(R,a,b),
    // Compute the beta function coefficient.
    bfc=beta_function(a,b);

    // Compute the sum from zero to b.
    s=0.0;
    for i=0:b-1,
        // Compute the binomial coefficient for
        // this term of the expansion.
        bc=binomial_coeff(b-1,i);

        // Compute the main term.
        rc=R^(a+i) / (a+i);

        // Compute whether the term is negative and
        // accumulate the partial sum.
        if modulo(i,2)==1 then
            s=s-rc*bc;
        else
            s=s+rc*bc;
        end;
    end;
    C=1.0-s/bfc;
endfunction;

```

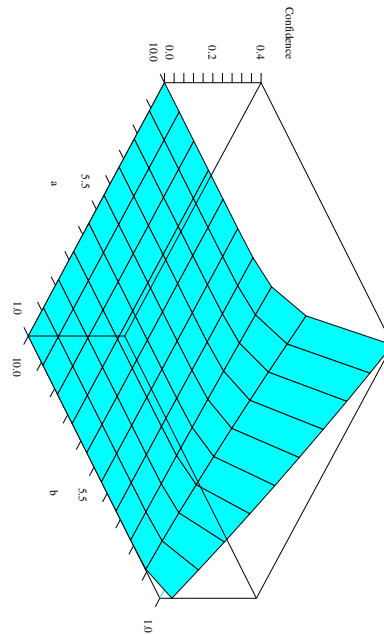


Figure 4.5: Confidence level versus priors

```

end; end;
xbasc();
[x,y,z]=genfac3d(1:10,1:10,c);
plot3d(x,y,list(z,4*ones(1,81)),...
-224,85,"a@b@Confidence");

```

These commands generate the graph shown in figure 4.5.

4.6 Choosing priors for the beta distribution

The Miller model (eq. 4.4 and eq. 4.5) requires values for the priors a and b . If no prior assumptions are warranted, one should set $a = b = 1$. Alternately a and b may be set to one plus the prior successes, and one plus the prior failures, respectively.

More realistically, one may have information about an assumed reliability μ and its variance σ^2 from previous releases. One may treat μ and σ^2 as the prior information,

and compute an a and b as follows. Applying the beta distribution for μ gives:

$$\begin{aligned}
 \mu &= \frac{a}{a+b} \\
 \mu(a+b) &= a \\
 \mu a + \mu b &= a \\
 \mu b &= a - \mu a \\
 b &= \frac{a(1-\mu)}{\mu}.
 \end{aligned} \tag{4.9}$$

For σ^2 one obtains:

$$\begin{aligned}
 \sigma^2 &= \frac{b\mu}{(a+1+b)(a+b)} \\
 \sigma^2 &= \frac{a(1-\mu)}{\left(a+1+\frac{a(1-\mu)}{\mu}\right)\left(a+\frac{a(1-\mu)}{\mu}\right)} \\
 \sigma^2\left(a+1+\frac{a(1-\mu)}{\mu}\right)\left(a+\frac{a(1-\mu)}{\mu}\right) &= a(1-\mu) \\
 \sigma^2\left(\frac{\mu+a}{\mu}\right)\left(\frac{a}{\mu}\right) &= a(1-\mu) \\
 \sigma^2(\mu+a)a &= a\mu^2(1-\mu) \\
 \sigma^2(\mu+a)a - a\mu^2(1-\mu) &= 0 \\
 a(\sigma^2(\mu+a) - \mu^2(1-\mu)) &= 0 \\
 \sigma^2\mu + \sigma^2a - \mu^2 + \mu^3 &= 0 \\
 \sigma^2a &= \mu^2 - \mu^3 - \sigma^2\mu \\
 a &= \frac{\mu^2(1-\mu) - \sigma^2\mu}{\sigma^2}.
 \end{aligned} \tag{4.10}$$

Substituting eq. 4.10 into eq. 4.9 gives the formula for b in terms of μ and σ^2 :

$$b = \frac{(1-\mu)(\mu^2(1-\mu) - \sigma^2\mu)}{\mu\sigma^2}. \tag{4.11}$$

Eq. 4.10 and eq. 4.11 allow choosing priors based on prior assumptions about reliability.

Bibliography

- [1] Walter Gutjahr. Importance sampling of test cases in markovian software usage models. *Probability in the Engineering and Information Sciences*, 11:19–36, 1997.
- [2] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, England, 1985.
- [3] L. B. W. Jolley. *Summation of Series*. Dover Publications, Inc., New York, NY, second revised edition, 1961.
- [4] David P. Kelly and Robert Oshana. Improving software quality using statistical testing techniques. *Information and Software Technology*, 42(12):801–807, 2000.
- [5] David P. Kelly and Jesse H. Poore. From good to great: Lifecycle improvements can make the difference. *Cutter IT Journal*, 13(2), February 2000.
- [6] J. G. Kemmeny and J. L. Snell. *Finite Markov Chains*. Springer-Verlag, New York, NY, 1976.
- [7] Keith W. Miller, Larry J. Morell, Robert E. Noonan, Stephen K. Park, David M. Nicol, Branson W. Murriel, and Jeffrey M. Voas. Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering*, 18(1):33–44, January 1992.
- [8] Jenny-Ann Morales. Test planning for software reuse. Master’s thesis, The University of Tennessee, Knoxville, TN, 1997.
- [9] David E. Pearson. Generalized testing chains. Master’s thesis, The University of Tennessee, Knoxville, TN, 2000.
- [10] P. E. Pfeiffer. *Concepts of Probability Theory*. Dover Publications, Inc., New York, NY, second revised edition, 1978.
- [11] Jesse H. Poore, Harlan D. Mills, and David Mutchler. Planning and certifying software system reliability. *IEEE Software*, 10(1):88–99, 1993.
- [12] S. J. Prowell. TML: A description language for markov chain usage models. *Information and Software Technology*, 42(12):835–844, September 2000.

- [13] Kirk D. Sayre. *Improved Techniques for Software Testing Based on Markov Chain Usage Models*. PhD thesis, The University of Tennessee, Knoxville, TN, December 1999.
- [14] Kirk D. Sayre and Jesse H. Poore. Stopping criteria for statistical testing. *Information and Software Technology*, 42(12):851–857, September 2000.
- [15] Carmen Trammell and Jesse H. Poore. Experimental control in software reliability certification. In *Proceedings of the Nineteenth Annual Software Engineering Workshop*. NASA/GSFC Software Engineering Laboratory, 1994.
- [16] Gwendolyn H. Walton and Jesse H. Poore. Generating transition probabilities to support model-based software testing. *Software—Practice and Experience*, 30(10):1095–1106, August 2000.
- [17] James A. Whittaker and Jesse H. Poore. Markov analysis of software specifications. *acm Transactions of Software Engineering and Methodology*, 2(1):93–106, January 1993.
- [18] James A. Whittaker and Michael G. Thomason. A markov chain model for statistical software testing. *IEEE Transactions on Software Engineering*, 20(10):812–824, October 1994.

Appendix A

Probability Density and Distribution Functions

A fundamental concept of continuous probability theory is that of the *Borel sets*, which can be informally defined as the sets composed from finite unions and complementations of open and closed intervals from the real number set. That is (informally), it is the collection of piecewise continuous real number sets. Let the set of Borel sets be denoted \mathbb{B} .

A.1 The gamma and beta functions

Some of the distributions to be discussed make use of the gamma function $\Gamma : \mathbb{R} \rightarrow \mathbb{R}$, which is defined as:

$$\Gamma(\alpha) = \int_0^{\infty} e^{-x} x^{\alpha-1} dx.$$

An interesting property of the gamma function is the following. Consider $\Gamma(\alpha + 1)$:

$$\Gamma(\alpha + 1) = \int_0^{\infty} e^{-x} x^{\alpha} dx. \tag{A.1}$$

The right-hand side of eq. A.1 can be approached using integration by parts. The standard form for this is $\int u dv = uv - \int v du + C$, obtained from the product rule for derivatives. Letting $u = x^{\alpha}$ and $dv = e^{-x} dx$ gives the following quick derivations. For du :

$$\begin{aligned} \frac{d}{dx} u &= \frac{d}{dx} x^{\alpha} \\ \frac{d}{dx} u &= \alpha x^{\alpha-1} \\ du &= \alpha x^{\alpha-1} dx, \end{aligned} \tag{A.2}$$

7 APPENDIX A. PROBABILITY DENSITY AND DISTRIBUTION FUNCTIONS

and for v :

$$\begin{aligned}
 v &= \int dv \\
 &= \int e^{-x} dx \\
 &= -e^{-x} + C.
 \end{aligned} \tag{A.3}$$

Substituting eq. A.2 and eq. A.3 into the integration by parts formula gives:

$$\begin{aligned}
 \Gamma(\alpha + 1) &= \int_0^{\infty} e^{-x} x^{\alpha} dx \\
 &= \int_0^{\infty} u dv \\
 &= \left[uv - \int v du \right]_0^{\infty} \\
 &= \left[(-e^{-x} + C)x^{\alpha} - \int (-e^{-x} + C)\alpha x^{\alpha-1} dx \right]_0^{\infty} \\
 &= \left[(-e^{-x} + C)x^{\alpha} \right]_0^{\infty} - \int_0^{\infty} (-e^{-x} + C)\alpha x^{\alpha-1} dx \\
 &= \left[-e^{-x} x^{\alpha} + Cx^{\alpha} \right]_0^{\infty} + \int_0^{\infty} e^{-x} \alpha x^{\alpha-1} dx - \int_0^{\infty} C\alpha x^{\alpha-1} dx \\
 &= \left[-e^{-x} x^{\alpha} \right]_0^{\infty} + \left[Cx^{\alpha} \right]_0^{\infty} + \alpha \Gamma(\alpha) - \left[Cx^{\alpha} \right]_0^{\infty} \\
 &= \alpha \Gamma(\alpha).
 \end{aligned} \tag{A.4}$$

Consider $\Gamma(1)$:

$$\begin{aligned}
 \Gamma(1) &= \int_0^{\infty} e^{-x} x^0 dx \\
 &= \int_0^{\infty} e^{-x} dx \\
 &= - \int_0^{\infty} e^{-x} d(-x) \\
 &= \left[-e^{-x} \right]_0^{\infty} \\
 &= \lim_{n \rightarrow \infty} (-e)^{-n} + e^0 \\
 &= 1.
 \end{aligned} \tag{A.5}$$

Eq. A.5 and eq. A.4 give the relationship:

$$\begin{aligned}
 \Gamma(1) &= 1 \\
 \Gamma(\alpha + 1) &= \alpha \Gamma(\alpha).
 \end{aligned}$$

Thus for all $\alpha \in \mathbb{Z}^+$, $\Gamma(\alpha) = (\alpha - 1)!$, and the gamma function can be viewed as a continuous version of the factorial function.

Another useful function is the beta function $B : \mathbb{R} \rightarrow \mathbb{R}$, defined:

$$B(a, b) = \int_0^1 x^{a-1} (1-x)^{b-1} dx.$$

APPENDIX A. PROBABILITY DENSITY AND DISTRIBUTION FUNCTIONS 3

This function has many interesting properties. Consider $B(a, b)$ and the transformation $y = 1 - x$ and thus $dy = -dx$:

$$\begin{aligned}
 B(a, b) &= \int_0^1 x^{a-1}(1-x)^{b-1} dx \\
 &= -\int_{1-0}^{1-1} (1-y)^{a-1}y^{b-1} dy \\
 &= -\int_1^0 (1-y)^{a-1}y^{b-1} dy \\
 &= \int_0^1 y^{b-1}(1-y)^{a-1} dy \\
 &= B(b, a).
 \end{aligned}$$

The gamma and beta functions are linked by the following relationship:

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}.$$

To see that this relationship is true, consider $\Gamma(a+b)B(a, b)$:

$$\begin{aligned}
 \Gamma(a+b)B(a, b) &= \int_0^\infty e^{-x}x^{a+b-1} dx \int_0^1 y^{a-1}(1-y)^{b-1} dy \\
 &= \int_0^\infty \int_0^1 e^{-x}x^{a+b-1}y^{a-1}(1-y)^{b-1} dy dx \\
 &= \int_0^\infty \int_0^1 e^{-x}x^{a-1}x^{b-1}xy^{a-1}(1-y)^{b-1} dy dx \\
 &= \int_0^\infty \int_0^1 e^{-x}x(xy)^{a-1}(x-xy)^{b-1} dy dx.
 \end{aligned}$$

Let $u = xy$ and $v = x - xy$. Then $x = u + v$ and $y = u/(u + v)$, and the region $R : 0 \leq x < \infty, 0 \leq y \leq 1$ is mapped to the new region $R' : 0 \leq u < \infty, 0 \leq v < \infty$. Consider the

Jacobian¹:

$$\begin{aligned}
 J(u, v) &= \left| \begin{array}{cc} \frac{\partial}{\partial u}(u+v) & \frac{\partial}{\partial v}(u+v) \\ \frac{\partial}{\partial u} \frac{u}{u+v} & \frac{\partial}{\partial v} \frac{u}{u+v} \end{array} \right| \\
 &= \left| \begin{array}{cc} 1 & 1 \\ \frac{v}{(u+v)^2} & \frac{-u}{(u+v)^2} \end{array} \right| \\
 &= -\frac{u}{(u+v)^2} - \frac{v}{(u+v)^2} \\
 &= -\frac{u+v}{(u+v)^2}.
 \end{aligned}$$

Note that u and v are never negative, so $\left| -\frac{u+v}{(u+v)^2} \right| = \frac{u+v}{(u+v)^2}$. The modulus of the Jacobian can be used with the change of variable in the double integral to yield the transformed integral:

$$\begin{aligned}
 \Gamma(a+b)\mathbf{B}(a, b) &= \int_0^\infty \int_0^\infty e^{-(u+v)} u^{a-1} v^{b-1} (u+v) \frac{u+v}{(u+v)^2} du dv \\
 &= \int_0^\infty \int_0^\infty e^{-u} e^{-v} u^{a-1} v^{b-1} du dv \\
 &= \int_0^\infty e^{-u} u^{a-1} du \int_0^\infty e^{-v} v^{b-1} dv \\
 &= \Gamma(a)\Gamma(b).
 \end{aligned} \tag{A.6}$$

A.2 Probability density and distribution functions

One can describe the probability associated with outcomes of a random variable X using a function $F_X : \mathbb{R} \rightarrow [0, 1]$ which maps the random variable's outcome to a real number equal to the probability mass of the outcome. In the discrete case such a mapping is a *probability distribution function* of the form:

$$\forall t \in \mathbb{R}, F_X(t) = \Pr[X = t].$$

In the continuous case such a mapping is called a *probability density function* or sometimes a *continuous distribution function* $f_X : \mathbb{R} \rightarrow [0, 1]$, and has the form:

$$\forall M \in \mathbb{B}, \int_M F_X(u) du = \Pr[X \in M].$$

¹For a double integral, let $x = x(u, v)$ and $y = y(u, v)$ be a change of variable which maps region R to transformed region R' . The *Jacobian* $J(u, v)$ describes the relationship between an element of the area $du dv$ and an element of the area $dx dy$:

$$J(u, v) = \left| \begin{array}{cc} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{array} \right| = \frac{\partial x}{\partial u} \frac{\partial y}{\partial v} - \frac{\partial x}{\partial v} \frac{\partial y}{\partial u} = \frac{\partial(x, y)}{\partial(u, v)}.$$

The transformed integral uses the modulus of the Jacobian:

$$\int \int_R f(x, y) dx dy = \int \int_{R'} f(x(u, v), y(u, v)) |J(u, v)| du dv.$$

APPENDIX A. PROBABILITY DENSITY AND DISTRIBUTION FUNCTIONS 5

Table A.1: Common probability density functions (for $x > 0, a > 0, b > 0$)

Distribution	Definition
Beta	$\mathcal{B}(x; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1}$
Exponential	$\mathcal{E}(x, a) = ae^{-ax}$
Gamma	$\mathcal{G}(x; a, b) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}$
Normal	$\mathcal{N}(x; \mu, \sigma) = \left(\frac{1}{\sigma\sqrt{2\pi}} \right) e^{\left(\frac{-(x-\mu)^2}{2\sigma^2} \right)}$
Uniform	$\mathcal{U}(x; a, b) = \frac{1}{b-a}$

Table A.2: Common probability distribution functions (for $x = 0, 1, 2, \dots$)

Distribution	Definition
Binomial	$B(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$
Geometric	$G(x; \theta) = \theta(1-\theta)^{x-1}$
Hypergeometric	$H(x; n, N, k) = \frac{\binom{k}{x} \binom{N-k}{n-x}}{\binom{N}{n}}$
Pascal (for $x = k, k+1, k+2, \dots$)	$P(x; p, k) = \binom{x-1}{k-1} p^k (1-p)^{x-k}$
Poisson	$\mathcal{P}(x; \theta) = \frac{e^{-\theta} \theta^x}{x!}$

There are many probability density functions; some of the better-known are the beta, exponential, gamma, normal, and uniform distributions. These are defined in table A.1. Some common probability distributions are the binomial, geometric, hypergeometric, Pascal, Poisson, which are defined in table A.2. Many continuous density functions have discrete versions, including the beta distribution.

The normal distribution has two parameters: the mean μ and standard deviation σ . The beta distribution has two parameters, a and b , which choose among a family of distributions. The uniform distribution is independent of x ; every point has the same density, hence its distribution. One uses these to compute the density of an interval $[i, j]$ via integrating over the interval; for example, the following is the probability density of the interval $[0, 1/2]$ given that the underlying random variable obeys the beta distribution:

$$\Pr[0 \leq x \leq \frac{1}{2}] = \int_0^{\frac{1}{2}} \mathcal{B}(x; a, b) dx.$$

A.3 The beta distribution

Let X be a parameter on $[0, 1]$ whose value is to be estimated, and assume X is governed by the beta distribution with parameters a and b :

$$\Pr[X = x] = \mathcal{B}(x; a, b).$$

Thus $E[X] = \int_0^1 \Pr[X = x]x dx$. Consider the more general case of $E[X^n]$:

$$\begin{aligned} E[X^n] &= \int_0^1 \Pr[X = x]x^n dx \\ &= \int_0^1 \mathcal{B}(x; a, b)x^n dx \\ &= \int_0^1 \frac{x^{a-1}(1-x)^{b-1}}{\mathcal{B}(a, b)}x^n dx \\ &= \frac{1}{\mathcal{B}(a, b)} \int_0^1 x^{a+n-1}(1-x)^{b-1} dx \\ &= \frac{\mathcal{B}(a+n, b)}{\mathcal{B}(a, b)}. \end{aligned} \tag{A.7}$$

Using eq. A.6, eq. A.4, and eq. A.7, it is easy to evaluate $E[X]$ and $\text{Var}[X]$:

$$\begin{aligned} E[X] &= \frac{\mathcal{B}(a+1, b)}{\mathcal{B}(a, b)} \\ &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(a+1)\Gamma(b)}{\Gamma(a+1+b)} \\ &= \frac{\Gamma(a+b)}{\Gamma(a)} \frac{a\Gamma(a)}{(a+b)\Gamma(a+b)} \\ &= \frac{a}{a+b}. \end{aligned}$$

$$\begin{aligned} \text{Var}[X] &= E[X^2] - E^2[X] \\ &= \frac{\mathcal{B}(a+2, b)}{\mathcal{B}(a, b)} - \left[\frac{a}{a+b} \right]^2 \\ &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(a+2)\Gamma(b)}{\Gamma(a+2+b)} - \left[\frac{a}{a+b} \right]^2 \\ &= \frac{\Gamma(a+b)}{\Gamma(a)} \frac{(a+1)a\Gamma(a)}{(a+1+b)(a+b)\Gamma(a+b)} - \left[\frac{a}{a+b} \right]^2 \\ &= \frac{(a+1)a}{(a+1+b)(a+b)} - \left[\frac{a}{a+b} \right]^2 \\ &= \frac{(a+1)a(a+b)}{(a+1+b)(a+b)^2} - \frac{(a+1+b)a^2}{(a+1+b)(a+b)^2} \\ &= \frac{ab}{(a+1+b)(a+b)^2}. \end{aligned}$$

Appendix B

Alternative Derivations of Information Theory Results

There are many different ways of interpreting the entropy and associated measures. This appendix presents some alternate derivations for these results.

B.1 Statistically-typical sequences

Another way of describing “statistically-typical” is to take the recurrent Markov chain and instead of focusing on single test cases, focus on very large realizations (i.e., sequences of several test cases).

Consider a series of test cases whose total length is N , where N is large enough that large number laws take effect and the sequence statistics closely match (within some ϵ) their expectations. For such a sequence state i is expected to appear $\pi_i N$ times, and the transition from state i to state j will appear $\pi_i N p_{i,j}$ times. To obtain the probability with which such a series is generated, one takes the product of the probabilities of each transition. Let \mathcal{R} denote such a series of test cases. Then:

$$\begin{aligned}\Pr[\mathcal{R}] &= \prod_{i,j} p_{i,j}^{\pi_i N p_{i,j}} \\ \lg \Pr[\mathcal{R}] &= \lg \prod_{i,j} p_{i,j}^{\pi_i N p_{i,j}} \\ &= \sum_{i,j} \lg p_{i,j}^{\pi_i N p_{i,j}} \\ &= \sum_{i,j} \pi_i N p_{i,j} \lg p_{i,j} \\ &= N \sum_{i,j} \pi_i p_{i,j} \lg p_{i,j} \\ &= -NH \\ \Pr[\mathcal{R}] &= 2^{-NH}.\end{aligned}$$

So such series have probability 2^{-NH} . Given this, there are:

$$\frac{1}{\Pr[\mathcal{R}]} = 2^{NH}$$

equally-likely statistically-typical series of test cases.

A sequence of length N corresponds to (on average) $\pi_n N$ test cases, since this is the expected number of times one observes the sink in a trajectory of length N . The average number of events per test case is thus (counting the recurrence loop):

$$\frac{N}{\pi_n N} = \frac{1}{\pi_n}.$$

If the same reasoning is applied as before but on the basis of test cases, this ends up dividing the occurrence counts $\pi_i N p_{i,j}$ by $\pi_n N$ to obtain $\pi_i p_{i,j} / \pi_n$. Letting \mathcal{T} denote a “statistically typical” test case gives:

$$\begin{aligned} \Pr[\mathcal{T}] &= \prod_{i,j} p_{i,j}^{\pi_i N p_{i,j} / \pi_n} \\ \lg \Pr[\mathcal{T}] &= \lg \prod_{i,j} p_{i,j}^{\pi_i p_{i,j} / \pi_n} \\ &= \sum_{i,j} \lg p_{i,j}^{\pi_i p_{i,j} / \pi_n} \\ &= \sum_{i,j} \frac{\pi_i}{\pi_n} N p_{i,j} \lg p_{i,j} \\ &= \frac{1}{\pi_n} \sum_{i,j} \pi_i p_{i,j} \lg p_{i,j} \\ &= -\frac{H}{\pi_n} \\ \Pr[\mathcal{T}] &= 2^{-H/\pi_n}, \end{aligned}$$

which is the same result as obtained previously.

B.2 Discrimination

Assume there are two distributions: p which is the true distribution, and q which is an approximating distribution. Observations are taken from the true distribution, and then explained, recorded, or encoded using the approximating distribution. Let \mathcal{R} be a collection of sequences of total length N generated from the true distribution, and let N be large enough that large number laws take effect. The probability of such a sequence given probability distribution r is denoted $\Pr[\mathcal{R}|r]$. Consider the limit in the average

number of bits wasted by using the approximating distribution q as N becomes large:

$$\begin{aligned}
 & \lim_{N \rightarrow \infty} \frac{1}{N} \left[- \sum_{\mathcal{R}} \Pr[\mathcal{R}|p] \lg \Pr[\mathcal{R}|q] + \sum_{\mathcal{R}} \Pr[\mathcal{R}|p] \lg \Pr[\mathcal{R}|p] \right] \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \left[\sum_{\mathcal{R}} \Pr[\mathcal{R}|p] \lg \frac{\Pr[\mathcal{R}|p]}{\Pr[\mathcal{R}|q]} \right] \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \left[\sum_{\mathcal{R}} \Pr[\mathcal{R}|p] \lg \frac{\prod_{i,j} \pi_i^N p_{i,j}}{\prod_{i,j} q_{i,j}} \right] \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \left[\sum_{\mathcal{R}} \Pr[\mathcal{R}|p] \sum_{i,j} \pi_i^N p_{i,j} \lg \frac{p_{i,j}}{q_{i,j}} \right] \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \left[\sum_{i,j} \pi_i^N p_{i,j} \lg \frac{p_{i,j}}{q_{i,j}} \sum_{\mathcal{R}} \Pr[\mathcal{R}|p] \right] \\
 &= \lim_{N \rightarrow \infty} \frac{1}{N} \left[\sum_{i,j} \pi_i^N p_{i,j} \lg \frac{p_{i,j}}{q_{i,j}} \right] \\
 &= \lim_{N \rightarrow \infty} \sum_{i,j} \pi_i p_{i,j} \lg \frac{p_{i,j}}{q_{i,j}} \\
 &= \sum_{i,j} \pi_i p_{i,j} \lg \frac{p_{i,j}}{q_{i,j}} \\
 &= K[U, T].
 \end{aligned}$$

B.3 Sayre discrimination

The Sayre discrimination [14], denoted $K^S[U, T]$, is computed using the Kronecker delta:

$$K^S[U, T] = \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg \frac{u_{i,j}}{\varepsilon \delta_{t_{i,j}0} + t_{i,j}}.$$

This version of the discrimination is discontinuous; one of the properties which entropy measures are intended to satisfy is continuity. An advantage to this measure is that once testing has covered all arcs one has $t_{i,j} \neq 0$ whenever $u_{i,j} \neq 0$ and thus $K^S[U, T] = K[U, T]$ immediately.

The Sayre discrimination does, however, introduce a different approach to computing a perturbed discrimination. Instead of adding the perturbation ε to the arc frequency count, one might choose to add the perturbation ε to the testing chain probability, obtaining:

$$\hat{K}[U, T] = \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg \frac{u_{i,j}}{\varepsilon + t_{i,j}}.$$

Investigating this further reveals:

$$\begin{aligned}
 K[U, T] - \hat{K}[U, T] &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg \frac{u_{i,j}}{t_{i,j}} - \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg \frac{u_{i,j}}{\varepsilon + t_{i,j}} \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \left[\lg \frac{u_{i,j}}{t_{i,j}} - \lg \frac{u_{i,j}}{\varepsilon + t_{i,j}} \right] \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \left[\lg \frac{\varepsilon + t_{i,j}}{t_{i,j}} \right].
 \end{aligned}$$

As the number of sequences executed grows, $t_{i,j}$ approaches $u_{i,j}$. This gives:

$$\begin{aligned}
 \lim_{T \rightarrow U} [K[U, T] - \hat{K}[U, T]] &= \lim_{T \rightarrow U} \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \left[\lg \frac{\varepsilon + t_{i,j}}{t_{i,j}} \right] \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \left[\lim_{T \rightarrow U} \lg \frac{\varepsilon + t_{i,j}}{t_{i,j}} \right] \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \left[\lg \frac{\varepsilon + u_{i,j}}{u_{i,j}} \right] \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} [\lg(\varepsilon + u_{i,j}) - \lg u_{i,j}] \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg(\varepsilon + u_{i,j}) - \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg u_{i,j} \\
 &= \sum_{i=1}^n \pi_i \sum_{j=1}^n u_{i,j} \lg(\varepsilon + u_{i,j}) - H.
 \end{aligned}$$

That is, as testing experience grows the perturbation does *not* wash out, but converges to the discrimination between the true distribution and the perturbed distribution, as one would expect.