

The Equiprobable Coupon Collector Problem

Jerry Alan Veeh

October 31, 2005

Contents

1	Introduction	1
2	The HTML Interface	3
2.1	Introductory Comments	3
2.2	Interface Components	5
3	The JavaScript Code	7
4	Functions for Computation	10
4.1	The <code>color_distribution</code> Function	11
4.2	The <code>couponmle</code> Function	13
4.3	The <code>couponlower</code> Function	14
4.4	The <code>couponupper</code> Function	15

1 Introduction

The objective here is to provide an implementation of the theory of the article *Confidence Intervals for the Number of Unseen Types* by Mark Finkelstein, Howard G. Tucker, and Jerry Alan Veeh, which appeared in *Statistics and Probability Letters* volume 37 pages 423-430 (1998).

This JavaScript program computes the maximum likelihood estimator of, and confidence intervals for, the number of unknown colors in the equiprobable coupon collector's problem. A population contains an unknown number of different types

of items, here referred to as colors. There are an equal number of items of each color. Items are drawn one at a time with replacement from the population. In these draws the number of different colors obtained is observed. The objective is to estimate, and find confidence intervals for, the number of different colors in the population. One application of this problem is the estimation of the size of a wildlife population.

The results of the paper are now briefly summarized. Denote by k the unknown number of different colors in the population and by C the number of different colors in a sample of size n drawn with replacement from the population. The observed value of C is c .

Denote by U the largest value of k for which $P[C \leq c] > \alpha$. If $c = n$, $U = \infty$. If $c < n$ this probability goes to 0 as k tends to infinity, so there will be a unique finite value of U . The interval $[1, U]$ is then a one sided $100(1 - \alpha)\%$ confidence interval for k . In the same way, if L is the smallest value of k for which $P[C \geq c] > \alpha$ then $[L, \infty)$ is a one sided $100(1 - \alpha)\%$ confidence interval for k . A two sided $100(1 - \alpha)\%$ confidence interval is found by finding one sided intervals using $\alpha/2$ in place of α .

The maximum likelihood estimator of k does not exist if $c = n$. If $c < n$ the maximum likelihood estimator of k is the smallest value of k for which the quantity $(k + 1 / (k + 1 - c)) (k / (k + 1))^n$ is less than 1.

The programming consists of an HTML file which contains the interface components, together with a JavaScript file containing the computational components.

```
"Coupon.html" 1 ≡
```

```
    <Introductory Comments 3>  
    <Interface Components 5>  
    <Closing Components 6a>  
    ◇
```

```
"coupon.js" 2 ≡
```

```
    <Compute Function 6b>  
    <Functions for Computation 10>  
    ◇
```

2 The HTML Interface

The HTML file contains a brief description of what is to be computed, followed by interface elements to collect the data from the user and display the results of the computation.

2.1 Introductory Comments

Here the basic structure of the HTML file is set up. The script containing the computational elements is included by means of the script tag.

⟨Introductory Comments 3⟩ ≡

```
<HTML>
<head>
<title>Confidence Intervals for the Equiprobable
Coupon Collector's Problem</title>
<script src="coupon.js"></script>
</head>
<BODY>
<h1>
Confidence Intervals for the Equiprobable
Coupon Collector's Problem
</h1>
<hr>
<p>
This JavaScript program computes the maximum likelihood
estimator of, and confidence intervals for, the number of
unknown colors in the equiprobable coupon collector's
problem. A population contains an unknown number of
different types of items, here referred to as colors. There
are an equal number of items of each color. Items are drawn
one at a time with replacement from the population. In these
draws the number of different colors obtained is observed.
The objective is to estimate, and find confidence intervals
for, the number of different colors in the population. One
application of this problem is the estimation of the size of
a wildlife population. The theory underlying the
computations can be found in the paper <em>Confidence
Intervals for the Number of Unseen Types</em> by Mark
Finkelstein, Howard G. Tucker, and Jerry Alan Veeh, which
appeared in <em>Statistics and Probability Letters</em>
volume 37 pages 423-430 (1998). The annotated source
code is <a href="CouponCollector.pdf">available</a>
</p>
<hr>
◇
```

Macro referenced in scrap 1.

2.2 Interface Components

The interface components consist of textboxes for input together with a textarea for output. These boxes are named so that the JavaScript program can access them easily. The layout of the items is controlled by an HTML table element.

⟨Interface Components 5⟩ ≡

```
<form name="theform">
  <table>
    <tr>
      <td>Observed number of colors in the sample</td>
      <td><input type="text" name="obscolors" size="12"></td>
    </tr>
    <tr>
      <td>Sample Size</td>
      <td><input type="text" name="samsize" size="12"></td>
    </tr>
    <tr>
      <td>Confidence Level (%)</td>
      <td><input type="text" value="95" name="conf"
size="12"></td>
    </tr>
    <tr>
      <td></td>
      <td> <input type="button" value="Compute"
onclick="compute();"></td>
    </tr>
    <tr>
      <td colspan="2">
        <textarea name="textout" rows="15" cols="60" wrap="physical">
          Copyright 2005 Jerry Alan Veeh. All rights reserved.

        </textarea>
      </td>
    </tr>
  </table>
</form>
◇
```

Macro referenced in scrap 1.

⟨Closing Components 6a⟩ ≡

```
<hr>
<a href="statapps.html">Statistical Programs
in JavaScript</a><br>
<a href="http://javeeh.net">Jerry Veeh's Personal
Home Page</a><br>
<small>Copyright &copy; 2005 Jerry Alan Veeh.
All rights reserved.</small>
</p>
</BODY>
</HTML>
◇
```

Macro referenced in scrap 1.

3 The JavaScript Code

The JavaScript code first performs validation of the input values and then does the required computation.

The `compute` function verifies the input values, activating alert boxes to prompt for corrections as needed. The computational routines are then called.

The `outwin` variable is used to direct output to the textarea in the HTML page.

⟨Compute Function 6b⟩ ≡

```
function compute(){
var outwin=document.theform.textout.value;
⟨Parse Input 7⟩
⟨Validate Input 8⟩
⟨Compute 9⟩
document.theform.textout.value=out+outwin;
}
◇
```

Macro referenced in scrap 2.

The input is parsed using integer or floating point parsing functions.

⟨Parse Input 7⟩ ≡

```
var c=0, n=0, cl=95;  
c=parseInt(document.theform.obscolors.value);  
n=parseInt(document.theform.samsize.value);  
cl=parseFloat(document.theform.conf.value);  
◇
```

Macro referenced in scrap 6b.

Validation checks for correctly parsed values, and verifies that the inputs lie within the necessary ranges.

⟨Validate Input 8⟩ ≡

```
if (isNaN(n))
  {
    alert("The sample size is not a number.");
    return;
  }
if (n<=0)
  {
    alert("The sample size must be at least 1.");
    return;
  }
if (isNaN(c))
  {
    alert("The number of observed colors is not a number.");
    return;
  }
if ((c<0)||(c>n))
  {
    alert("The number of observed colors must be between zero "+
          "and the sample size, inclusive.");
    return;
  }
if (isNaN(cl))
  {
    alert("The confidence level is not a number.");
    return;
  }
if ((cl<=0)||(cl>=100))
  {
    alert("The confidence level must be between "+
          "zero and 100, exclusive.");
    return;
  }
◇
```

Macro referenced in scrap 6b.

Statistical functions are used for the computation. The results are incrementally added to the output variable out.

⟨Compute 9⟩ ≡

```
var out="";
out += "Here "+c+" colors were observed in a "+
      "sample of size "+n+" with replacement.\n";
if (c==n)
{
out+="The MLE does not exist.\n";
}
else
{
out += "The MLE of the total number of colors is "+
      couponmle(n, c)+".\n";
}
out += "The lower endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "total number of colors is "+
      couponlower(n, c, (100-cl)/100)+".\n";
out += "The upper endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "total number of colors is "+
      couponupper(n, c, (100-cl)/100)+".\n";
out += "The endpoints of a two sided "+
      cl+"% confidence interval for the ";
out += "total number of colors are "+
      couponlower(n, c, (100-cl)/200)+" and ";
out += ""+couponupper(n, c, (100-cl)/200)+".\n\n";
◇
```

Macro referenced in scrap 6b.

4 Functions for Computation

The functions used here compute probabilities and confidence interval endpoints.

⟨Functions for Computation 10⟩ ≡

⟨color_distribution 11a⟩

⟨couponmle 13⟩

⟨couponlower 14⟩

⟨couponupper 15⟩

◇

Macro referenced in scrap 2.

4.1 The color_distribution Function

The function call `color_distribution(n, k, c)` computes the probability of obtaining c or fewer different colors in a sample of size n with replacement from a population which contains k different colors in equal proportions.

The routine to compute the value of the distribution function of C is rather elaborate. Denote by C_m the number of distinct colors in a sample of size m drawn with replacement from a population containing k different colors. The recursive formula

$$P[C_{m+1} = j] = P[C_m = j](j/k) + P[C_m = j-1](k-j+1)/k$$

is used to compute the density function of C_n . This formula is easily derived by noting that on draw $m+1$ either a coupon of a previously drawn color is again observed, or a coupon of a new color is observed. From this density the required tail probabilities are computed by summation.

The density need only be computed for $0 \leq j \leq c$. The recursion is performed by using a single array, `density`, of length $2c+2$ which is divided into two halves. The halves take turns holding the successive values of the density. The indices `old_density` and `new_density` are used to point to the starting place of the two halves.

When $n = 1$ the density is degenerate. This provides a starting point for the recursion.

⟨color_distribution 11a⟩ ≡

```
function color_distribution(n, k, c){
  var j, old_density, new_density;
  var density=new Array(2*c+2);
  for(j=0;j<=2*c+1;j++) density[j]=0;
  old_density=0;
  new_density=c+1;
  density[1]=1;
  ⟨Compute Color Density 11b⟩
  ⟨Compute Color Distribution 12⟩
  return dist;
}
◇
```

Macro referenced in scrap 10.

The density of C_m is successively computed for values of m beginning with $m=2$. The index pointers `old_density` and `new_density` are swapped after each range of density values is computed. At the conclusion of the computation $P[C_n = j] = \text{density}[\text{old_density} + j]$, for $0 \leq j \leq c$.

⟨Compute Color Density 11b⟩ ≡

```
var m;
for(m=2;m<=n;m++)
  {
    for(j=1;j<=c;j++)
      {
        density[new_density+j]=( j*density[old_density+j] +
          (k-j+1)*density[old_density+j-1] )/k;
      }
    j=old_density;
    old_density=new_density;
    new_density=j;
  }
◇
```

Macro referenced in scrap 11a.

The density is summed to compute the needed tail probability $\text{dist} = P_k[C_n \leq c]$.

⟨Compute Color Distribution 12⟩ ≡

```
var dist;
dist=0;
for(j=1;j<=c;j++)
    {
        dist=dist+density[old_density+j];
    }
```

◇

Macro referenced in scrap 11a.

4.2 The couponmle Function

The function call `couponmle(n, c)` computes the maximum likelihood estimator of the number of colors in the population.

If $n = c$ the maximum likelihood estimator does not exist. Otherwise, the maximum likelihood estimator is the smallest value of k for which $(k + 1 / (k + 1 - c)) (k / (k + 1))^n$ is less than 1.

A bisection is initialize by finding two endpoints at which the quantity $(k + 1 / (k + 1 - c)) (k / (k + 1))^n$ is less than and bigger than 1. This is done by searching through powers of 2 for the endpoints. The variables `low` and `high` are initialized with the value `c` so that the case in which $c = 1$ will be handled correctly.

Bisection is then performed until the difference between `high` and `low` is 1 or less. At that point `high` is the maximum likelihood estimator.

⟨couponmle 13⟩ ≡

```
function couponmle(n, c){
  if(n==c) return(Number.POSITIVE_INFINITY);
  var low, high, test;
  low=c;
  high=c;
  while( (high+1)*Math.pow( (high)/(high+1), n) >(high+1-c) )
    {
      low=high;
      high=2*high;
    }
  while( high-low >1)
    {
      test=Math.floor((high+low)/2);
      if ((test+1)*Math.pow( (test)/(test+1), n) >(test+1-c) )
        {
          low=test;
        }
      else
        {
          high=test;
        }
    }
  return high;
}
◇
```

Macro referenced in scrap 10.

4.3 The couponlower Function

The couponlower function computes the lower endpoint of a one sided $100(1 - \alpha)\%$ confidence interval for the number of successes in the population.

The technique for each of the endpoints is the same as that used in finding the maximum likelihood estimator.

The lower endpoint is the smallest value of k for which $P[C \geq c] > \alpha$, i.e., the smallest k for which $P[C \leq c - 1] < 1 - \alpha$. Since the probability goes to 0 as k tends to infinity there is always a finite lower endpoint. In the following computations, $k = \text{low}$ always makes $P[C \leq c - 1] \geq 1 - \alpha$. After the first while

loop, $k = \text{high}$ makes $P[C \leq c - 1] < 1 - \alpha$.

$\langle \text{couponlower 14} \rangle \equiv$

```
function couponlower(n, c, alpha){
  var low, high, test;
  low=c-1;
  high=2*c;
  while( color_distribution(n, high ,c-1) >= 1-alpha )
    {
      low=high;
      high=2*high;
    }
  while( high-low >1)
    {
      test=Math.floor((high+low)/2);
      if ( color_distribution(n, test, c-1) < 1-alpha )
        {
          high=test;
        }
      else
        {
          low=test;
        }
    }
  return high;
}
◇
```

Macro referenced in scrap 10.

4.4 The couponupper Function

The couponupper function computes the upper endpoint of a one sided $100(1 - \alpha)\%$ confidence interval for the number of successes in the population.

This function finds the upper endpoint of the one sided $100(1 - \alpha)\%$ confidence interval for k in the coupon collectors problem. The upper endpoint is the largest value of the k for which $P[C \leq c] > \alpha$. If $n = c$ the upper endpoint is infinity. If $c < n$ this probability goes to 0 as k tends to infinity. At $k = \text{low}$, $P[C \leq c] > \alpha$, while after the first while loop at $k = \text{high}$, $P[C \leq c] \leq \alpha$.

⟨couponupper 15⟩ ≡

```
function couponupper(n, c, alpha){
  if(n==c) return (Number.POSITIVE_INFINITY);
  var low, high, test;
  low=c;
  high=2*c;
  while( color_distribution(n, high ,c) > alpha )
    {
      low=high;
      high=2*high;
    }
  while( high-low >1)
    {
      test=Math.floor((high+low)/2);
      if ( color_distribution(n, test, c) <=alpha )
        {
          high=test;
        }
      else
        {
          low=test;
        }
    }
  return low;
}
◇
```

Macro referenced in scrap 10.