

Estimation of the Hypergeometric Number of Successes

Jerry Alan Veeh

October 27, 2005

Contents

1	Introduction	1
2	The HTML Interface	2
2.1	Introductory Comments	3
2.2	Interface Components	4
3	The JavaScript Code	6
4	Functions for Computation	10
4.1	The hyperprob Function	10
4.2	The hypertail Function	11
4.3	The hgrlower Function	12
4.4	The hgrupper Function	13

1 Introduction

The objective here is to provide an implementation of the theory of the article *Conservative Confidence Intervals for a Single Parameter* by Mark Finkelstein, Howard G. Tucker and Jerry Alan Veeh, which appeared in *Communications in Statistics: Theory and Methods* volume 29 #8 pages 1911-1928 (2000). The focus is on finding confidence intervals for the number R of successes in a finite

population of known size N . A sample of size n drawn without replacement has produced r successes and $n - r$ failures.

The results of the paper are now briefly summarized. Let M denote the number of successes in a sample of size n drawn without replacement from a population of total size N of which R items are successes and $N - R$ are failures. Denote by r the observed value of M .

Denote by U the largest value of R for which $P[M \leq r] > \alpha$. Then the interval $[0, U]$ is a one sided confidence interval for R with confidence level at least $1 - \alpha$. Similarly, if L is the smallest value of R for which $P[M \geq r] > \alpha$, the interval $[L, N]$ is a one sided confidence interval for R with confidence level at least $1 - \alpha$. A two sided confidence interval for R is found by finding one sided confidence intervals each having confidence level $1 - \alpha/2$.

The maximum likelihood estimator of R is known to be the greatest integer in $r(N + 1)/n$. If $r(N + 1)/n$ is an integer, then the maximum likelihood estimator is not unique, since $r(N + 1)/n - 1$ is also a maximum likelihood estimator of R , unless $r = 0$.

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

"Hyperr.html" 1a \equiv

⟨Introductory Comments 3⟩
⟨Interface Components 4⟩
⟨Closing Components 5a⟩
◇

"hyperr.js" 1b \equiv

⟨Compute Function 5b⟩
⟨Functions for Computation 9⟩
◇

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 Hypergeometric
Number of Successes</title>
<script src="hyperr.js"></script>
</head>
<BODY>
<h1>
Confidence Intervals for the Hypergeometric
Number of Successes
</h1>
<hr>
<p>
This JavaScript program computes the maximum likelihood
estimator of, and conservative confidence intervals for, the
number of successes in a population of known size when a
sample of known size is drawn from the population without
replacement. The population is assumed to consist of only
two types of objects: successes and failures. The number of
successes observed in the sample is known. The theory
underlying the computations can be found in the paper
<em>Conservative Confidence Intervals for a Single
Parameter</em> by Mark Finkelstein, Howard G. Tucker and
Jerry Alan Veeh, which appeared in <em>Communications in
Statistics: Theory and Methods</em> volume 29 #8 pages
1911-1928 (2000). The annotated source code is <a
href="HypergeometricR.pdf">available</a>
</p>
<hr>
◇
```

Macro referenced in scrap 1a.

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 4⟩ ≡

```
<form name="theform">
  <table>
    <tr>
      <td>Observed number of successes in the sample</td>
      <td><input type="text" name="successes" size="12"></td>
    </tr>
    <tr>
      <td>Sample Size</td>
      <td><input type="text" name="samsize" size="12"></td>
    </tr>
    <tr>
      <td>Population Size</td>
      <td><input type="text" name="popsize" 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 1a.

⟨Closing Components 5a⟩ ≡

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

Macro referenced in scrap 1a.

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 5b⟩ ≡

```
function compute(){
var outwin=document.theform.textout.value;
⟨Parse Input 6⟩
⟨Validate Input 7⟩
⟨Compute 8⟩
document.theform.textout.value=out+outwin;
}
◇
```

Macro referenced in scrap 1b.

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

⟨Parse Input 6⟩ ≡

```
var r=0, n=0, N=0, cl=95;  
r=parseInt(document.theform.successes.value);  
n=parseInt(document.theform.samsize.value);  
N=parseInt(document.theform.popsize.value);  
cl=parseFloat(document.theform.conf.value);  
◇
```

Macro referenced in scrap 5b.

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

⟨Validate Input 7⟩ ≡

```
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(r))
  {
    alert("The number of successes is not a number.");
    return;
  }
if ((r<0)||(r>n))
  {
    alert("The number of successes must be between zero "+
          "and the sample size, inclusive.");
    return;
  }
if (isNaN(N))
  {
    alert("The population size is not a number.");
    return;
  }
if (N<n)
  {
    alert("The population size must be "+
          "at least the sample size.");
    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 5b.

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

⟨Compute 8⟩ ≡

```
var out="";
out += "Here "+r+" successes were observed in a "+
      "sample of size "+n+
      " drawn from a population of size "+
      N+" without replacement.\n";
if((Math.floor(r*(N+1)/n)==r*(N+1)/n)&&(r>0))
{
out += "The MLEs of the number of successes are "+
      (r*(N+1)/n-1)+" and "+(r*(N+1)/n)+".\n";
}
else
{
out += "The MLE of the number of successes is "+
      Math.floor(r*(N+1)/n)+".\n";
}
out += "The lower endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "number of successes is "+
      hgrlower(N, n, r, (100-cl)/100)+".\n";
out += "The upper endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "number of successes is "+
      hgrupper(N, n, r, (100-cl)/100)+".\n";
out += "The endpoints of a two sided "+
      cl+"% confidence interval for the ";
out += "number of successes are "+
      hgrlower(N, n, r, (100-cl)/200)+" and ";
out += ""+hgrupper(N, n, r, (100-cl)/200)+".\n\n";
◇
```

Macro referenced in scrap 5b.

4 Functions for Computation

The functions used here compute probabilities related to the hypergeometric distribution by expansion of the binomial coefficients.

⟨Functions for Computation 9⟩ ≡

⟨hyperprob 10⟩

⟨hypertail 11⟩

⟨hgrlower 12⟩

⟨hgrupper 13⟩

◇

Macro referenced in scrap 1b.

4.1 The hyperprob Function

The function call `hyperprob(N, R, n, m)` function computes the probability of obtaining m successes in a sample of size n drawn without replacement from a population of total size N of which R items are successes and $N - R$ items are failures. This probability, $\binom{R}{r} \binom{N-R}{n-r} / \binom{N}{n}$, is computed by expanding the binomial coefficients and multiplying the factors together in an organized way. No error checking is done, since this function is designed to be called by the function `hypertail` to compute a tail probability for the hypergeometric distribution.

⟨hyperprob 10⟩ ≡

```
function hyperprob(N, R, n, m) {
  var frac=1.0;
  var i=0;
  for(i=0;i<=m-1;i++)
  {
    frac=frac*(R-i)*(n-i)/( (N-i)*(m-i) );
  }
  for(i=m;i<=n-1;i++)
  {
    frac=frac*(n-i)*(N-R-i+m)/ ( (N-i)*(n-m-i+m) );
  }
  return frac;
}
◇
```

Macro referenced in scrap 9.

4.2 The hypertail Function

The function call `hypertail(N, R, n, m)` computes the probability of observing at least m successes in a sample of size n drawn without replacement from a population of size N containing R successes and $N - R$ failures. Appropriate values are returned in extreme cases.

⟨hypertail 11⟩ ≡

```
function hypertail(N, R, n, m)
{
  if (m>Math.min(R,n)) return 0.0;
  if ( m<Math.max(0,n-(N-R)) ) return 1.0;
  var prob=0.0;
  var i=0;
  var end=Math.min(R,n);
  for(i=m;i<=end;i++)
  {
    prob=prob+hyperprob(N,R,n,i);
  }
  return prob;
}
◇
```

Macro referenced in scrap 9.

4.3 The hgrlower Function

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

Since M , the number of red balls in a sample, is stochastically monotone increasing in R , and the parameter space is $0 \leq R \leq N$, the lower endpoint of the confidence interval is the smallest value of R for which $P[M \geq r] > \alpha$. Notice that the choice $R = N$ makes the probability one for any r . Also, the choice $R = r - 1$ makes the probability zero. The sought after value of R lies between these two bounds, and is found by bisection. The tail probability at high always exceeds α , while that at low is α or less. When the difference between high and low is 1, the value of high is the lower endpoint of the confidence interval.

`<hgrlower 12> ≡`

```
function hgrlower(N, n, r, alpha)
{
var high, low, test;
var testprob;
high=N;
low=r;
testprob=hypertail(N,low,n,r);
if (testprob>alpha) return low;
while(high-low>1)
    {
    test=Math.floor((high+low)/2);
    testprob=hypertail(N,test,n,r);
    if(testprob<=alpha) low=test;
    if(testprob>alpha) high=test;
    }
return high;
}
◇
```

Macro referenced in scrap 9.

4.4 The hgrupper Function

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

Since M , the number of red balls in a sample, is stochastically monotone increasing in R , and the parameter space is $0 \leq R \leq N$, the lower endpoint of the confidence interval is the largest value of R for which $P[M \leq r] > \alpha$. Notice that the choice $R = r$ makes the probability one for any r . Also, the choice $R = N$ makes the probability zero, unless $r = N$. The sought after value of R lies between these two bounds, and is found by bisection. The tail probability at high is always α or less, while that at low exceeds α . When the difference between high and low is 1, the value of low is the upper endpoint of the confidence interval.

⟨hgrupper 13⟩ ≡

```
function hgrupper(N, n, r, alpha)
{
var high, low, test;
var testprob;
high=N;
low=r;
testprob=1.0-hypertail(N,high,n,r+1);
if (testprob>alpha) return high;
while(high-low>1)
{
test=Math.floor((high+low)/2);
testprob=1.0-hypertail(N,test,n,r+1);
if(testprob>alpha) low=test;
if(testprob<=alpha) high=test;
}
return low;
}
◇
```

Macro referenced in scrap 9.