Estimation of the Hypergeometric Population Size

Jerry Alan Veeh

October 27, 2005

Contents

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 population size N of a finite population.

A sample of size n drawn without replacement has produced r successes and $n-r$ failures. The number R of successes in the population is assumed to be known.

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

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

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

`"Hypern.html"` 1a ≡

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

`"hypern.js"` 1b ≡

⟨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
Population Size</title>
<script src="hypern.js"></script>
</head>
<BODY>
<h1>
Confidence Intervals for the Hypergeometric
Population Size
</h1>
<hr>
<p>
This JavaScript program computes the maximum likelihood
estimator of, and conservative confidence intervals for, the
population 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, as is the number of successes in the population.
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="HypergeometricN.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>Number of successes in the population</td>
<td><input type="text" name="popsuc" 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 Personal
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, R=0, cl=95;
r=parseInt(document.theform.successes.value);
n=parseInt(document.theform.samsize.value);
R=parseInt(document.theform.popsuc.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 observed successes is not a number.");
        return;
        }
if ((r<0)||(r>n))
        {
        alert("The number of observed successes must be between zero "+
                "and the sample size, inclusive.");
        return;
        }
if (isNaN(R))
        {
        alert("The number of population successes is not a number.");
        return;
        }
if (R<r)
        {
        alert("The number of population successes "+
                "must be at least the number of "+
                "successes observed.");
        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 with "+
        R+" successes, without replacement.\n";
if (r==0)
{
out+="The MLE does not exist.\n";
}
else if ((Math.floor(R*n/r)==R*n/r)&&(r>0))
{
out +="The MLEs of the population size are "+
        (R*n/r-1)+" and "+(R*n/r)+".\n";
}
else
{
out +="The MLE of the population size is "+
        Math.floor(R*n/r)+".\n";
}
out +="The lower endpoint of a one sided "+
        cl+"% confidence interval for the ";
out +="population size is "+
        hgnlower(R, n, r, (100-cl)/100)+".\n";
out +="The upper endpoint of a one sided "+
        cl+"% confidence interval for the ";
out +="population size is "+
        hgnupper(R, n, r, (100-cl)/100)+".\n";
out +="The endpoints of a two sided "+
        cl+"% confidence interval for the ";
out +="population size are "+
        hgnlower(R, n, r, (100-cl)/200)+" and ";
out +=""+hgnupper(R, 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⟩
> ⟨hgnlower 12⟩
> ⟨hgnupper 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 successess 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 `hgnlower` Function

The `hgnlower` function computes the lower endpoint of a one sided $100(1-\alpha)\%$ confidence interval for the population size.

Since M, the number of red balls in a sample, is stochastically monotone decreasing in N, and the parameter space is $N \geq 0$, the lower endpoint of the confidence interval is the smallest value of N for which $P[M \leq r] > \alpha$. Notice that as $N \to \infty$ the probability tends to one for any r. Also, the choice $N < R+n-r$ makes the probability zero. The sought after value of N is found by first bracketing the true value, and then using 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.

⟨hgnlower 12⟩ ≡

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

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

Macro referenced in scrap 9.

4.4 The `hgnupper` Function

The `hgnupper` function computes the upper endpoint of a one sided $100(1-\alpha)\%$ confidence interval for the population size.

Since M, the number of red balls in a sample, is stochastically monotone decreasing in N, and the parameter space is $N \geq 0$, the upper endpoint of the confidence interval is the largest value of N for which $P[M \geq r] > \alpha$. If $r = 0$ the largest value will be infinite. Notice that the choice $N = R + n - r$ makes the probability one for any r. The value of N is found by first bracketing the desired value and then using 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.

⟨hgnupper 13⟩ ≡

```
function hgnupper(R,  n,  r,  alpha)
{
if (r==0) return Number.POSITIVE_INFINITY;
var  high, low, test;
var testprob;
high=R+n-r;
testprob=1.0;
while(testprob>alpha)
        {
        high=2*high;
        testprob=hypertail(high,R,n,r);
        }
low=Math.floor(high/2);

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

Macro referenced in scrap 9.