

Estimation of the Binomial Number of Trials

Jerry Alan Veeh

October 26, 2005

Contents

1	Introduction	1
2	The HTML Interface	2
2.1	Introductory Comments	3
2.2	Interface Components	4
3	The JavaScript Code	5
4	Functions for Computation	8
4.1	The loggamma Function	9
4.2	The dfbeta Function	10
4.3	The bintail Function	15
4.4	The bnlower Function	16
4.5	The bnupper Function	17

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 n of trials in the binomial distribution which are guaranteed to have at least the nominal coverage probability no matter the size of the sample.

The results of the paper are now briefly summarized. Let X denote a binomial random variable with parameters n and p with p being known. Denote by k the observed number of successes in n trials. In this context the possible values of n are the non-negative integers. The convention that $P[X = 0] = 1$ if $n = 0$ is used.

Denote by U the largest value of n for which $P[X \leq k] > \alpha$. 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 $P[X \geq k] > \alpha$, 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 k/p . If k/p is an integer, then the maximum likelihood estimator is not unique, since $k/p - 1$ is also a maximum likelihood estimator of n , unless $k = 0$.

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

"Binn.html" 1a \equiv

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

"binn.js" 1b \equiv

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

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 Binomial
Number of Trials</title>
<script src="binn.js"></script>
</head>
<BODY>
<h1>
Confidence Intervals for the Binomial Number of Trials
</h1>
<hr>
<p>
This JavaScript program computes the maximum likelihood
estimator of, and conservative confidence intervals for, the
number of trials in a sequence of independent Bernoulli
trials. The success probability on each trial is assumed to
be known as is the observed number of successes in those
trials. The theory underlying the computations can be found
in the paper 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). Some of the computational methods use
algorithms from Applied Statistics Algorithms. The
annotated source code is available
</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</td>
      <td><input type="text" name="successes" size="12"></td>
    </tr>
    <tr>
      <td>Success probability on each trial</td>
      <td><input type="text" name="sucprob" 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 k=0,p=0.5, cl=95;  
k=parseInt(document.theform.successes.value);  
p=parseFloat(document.theform.sucprob.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(k))
  {
    alert("The number of successes is not a number.");
    return;
  }
if (k<0)
  {
    alert("The number of successes must be at least zero.");
    return;
  }

if (isNaN(p))
  {
    alert("The success probability is not a number.");
    return;
  }
if ((p<=0)|| (p>=1))
  {
    alert("The success probability must be between "+
          "zero and one, exclusive.");
    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.

Here the maximum likelihood estimator is the greatest integer in k/p unless k/p is an integer, in which case both k/p and $k/p - 1$ are maximum likelihood estimators.

⟨Compute 8⟩ ≡

```

var out="";
out += "Here "+k+" successes were observed, "+
      "each with success probability "+p+".\n";
if((Math.floor(k/p)==k/p)&&(k>0))
{
out += "The MLEs of the number of trials are "+
      "(k/p-1)+" and "+(k/p)+".\n";
}
else
{
out += "The MLE of the number of trials is "+
      Math.floor(k/p)+".\n";
}
out += "The lower endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "number of trials is "+bnlower(k,p,(100-cl)/100)+".\n";
out += "The upper endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "number of trials is "+bnupper(k,p,(100-cl)/100)+".\n";
out += "The endpoints of a two sided "+
      cl+"% confidence interval for the ";
out += "number of trials are "+bnlower(k,p,(100-cl)/200)+" and ";
out += ""+bnupper(k,p,(100-cl)/200)+".\n\n";
◇

```

Macro referenced in scrap 5b.

4 Functions for Computation

The functions used here compute the distribution function of the beta distribution. The connection between the beta distribution and the binomial distribution is used to find the required binomial probabilities.

⟨Functions for Computation 9a⟩ ≡

```
⟨loggamma 9b⟩
⟨dfbeta 12⟩
⟨bintail 16a⟩
⟨bnlower 16b⟩
⟨bnupper 18⟩
◇
```

Macro referenced in scrap 1b.

4.1 The loggamma Function

The `loggamma` function computes the logarithm of the value of the gamma function. This function uses algorithm ACM 291 from the book *Applied Statistics Algorithms* edited by P. Griffiths and I. D. Hill. The algorithm is by M. C. Pike and I. D. Hill. Accuracy is asserted to at least 8 decimal digits.

The algorithm first tests for an incorrect value of x and returns NaN if found. If $x < 7$, use the recurrence $\Gamma(x+1) = x\Gamma(x)$ in reverse to increase x . Stirling's formula is then applied to the larger x with the increase subtracted out at the end.

⟨loggamma 9b⟩ ≡

```
function loggamma(x) {
var a1=Math.log(2*Math.PI)/2;
var a2= 1/1680;
var a3= 1/1260;
var a4= 1/360;
var a5= 1/12;
var f,xx,z;
f=1;
if (x <=0) return(Number.NaN);
for(xx=x;xx < 7;xx++) f=f*xx;
f=-Math.log(f);
z=1/(xx*xx);
return(f+(xx-0.5)*Math.log(xx)-xx+a1+(((a2*z+a3)*z-a4)*z+a5)/xx);
}
◇
```

Macro referenced in scrap 9a.

4.2 The dfbeta Function

The function `dfbeta(x, p, q)` computes the value of the beta distribution with parameters p and q at the point $0 < x < 1$. The algorithm is Algorithm AS 63 from *Applied Statistics Algorithms*. Modifications have been made to eliminate the `goto` statements in the original Fortran implementation.

The incomplete beta function, denoted by $I_x(a, b)$, is defined by the formula

$$I_x(a, b) = \frac{1}{B(a, b)} \int_0^x t^{a-1} (1-t)^{b-1} dt$$

for $0 \leq x \leq 1$. In this formula, a and b are positive real numbers and $B(a, b)$ is the complete beta function. Recall that

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

where $\Gamma(z)$ is the (complete) gamma function. The value of the incomplete beta integral is computed by the function call `dfbeta(x, p, q) = I_x(p, q)`.

There is an intimate connection between the incomplete beta function and several of the common distribution functions of statistics. Before developing these connections two useful recursions of the incomplete beta function will be derived.

Theorem 1 (Parts) *The relation*

$$I_x(a, b) = \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b)} x^a (1-x)^{b-1} + I_x(a+1, b-1)$$

holds.

proof: As the name suggests, this formula follows from an integration by parts. Taking $dv = t^{a-1} dt$ and $u = (1-t)^{b-1}$ in the definition of $I_x(a, b)$ gives

$$\begin{aligned} I_x(a, b) &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1} (1-t)^{b-1} dt \\ &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \left[\frac{1}{a} t^a (1-t)^{b-1} \Big|_0^x + \int_0^x \frac{1}{a} t^a (b-1) (1-t)^{b-2} dt \right] \\ &= \frac{\Gamma(a+b)}{a\Gamma(a)\Gamma(b)} x^a (1-x)^{b-1} + \frac{\Gamma(a+b)(b-1)}{a\Gamma(a)\Gamma(b)} \int_0^x t^a (1-t)^{b-2} dt \\ &= \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b)} x^a (1-x)^{b-1} + \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b-1)} \int_0^x t^a (1-t)^{b-2} dt \\ &= \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b)} x^a (1-x)^{b-1} + I_x(a+1, b-1) \end{aligned}$$

as desired.

Theorem 2 (Raising p) *The relation*

$$I_x(a, b) = \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b)} x^a (1-x)^b + I_x(a+1, b)$$

holds.

proof: First note that by factoring and expanding

$$\begin{aligned} I_x(a, b+1) &= \frac{\Gamma(a+b+1)}{\Gamma(a)\Gamma(b+1)} \int_0^x t^{a-1} (1-t)^{b-1} (1-t) dt \\ &= \frac{\Gamma(a+b+1)}{\Gamma(a)\Gamma(b+1)} \int_0^x t^{a-1} (1-t)^{b-1} dt - \frac{\Gamma(a+b+1)}{\Gamma(a)\Gamma(b+1)} \int_0^x t^a (1-t)^{b-1} dt \\ &= \frac{a+b}{b} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1} (1-t)^{b-1} dt - \frac{a}{b} \frac{\Gamma(a+b+1)}{\Gamma(a+1)\Gamma(b)} \int_0^x t^a (1-t)^{b-1} dt \\ &= \frac{a+b}{b} I_x(a, b) - \frac{a}{b} I_x(a+1, b). \end{aligned}$$

On the other hand, using ‘parts’ gives

$$I_x(a, b+1) = \frac{\Gamma(a+b+1)}{\Gamma(a+1)\Gamma(b+1)} x^a (1-x)^b + I_x(a+1, b).$$

Equating these two expressions gives

$$\frac{\Gamma(a+b+1)}{\Gamma(a+1)\Gamma(b+1)} x^a (1-x)^b + I_x(a+1, b) = \frac{a+b}{b} I_x(a, b) - \frac{a}{b} I_x(a+1, b).$$

Re-arranging to solve for $I_x(a, b)$ gives the ‘raising p ’ relation above.

As an example of the use of these properties a useful connection between the incomplete beta integral and the binomial distribution is established.

Theorem 3 *For integers $0 \leq a \leq n$ and real $0 \leq p \leq 1$ the relation*

$$I_p(a, n-a+1) = \sum_{j=a}^n \binom{n}{j} p^j (1-p)^{n-j}$$

holds.

proof: By direct computation

$$I_x(1, n) = 1 - (1 - x)^n.$$

From the ‘parts’ relation

$$\begin{aligned} I_x(1, n) &= \binom{n}{1} x^1 (1-x)^{n-1} + I_x(2, n-1) \\ &= \dots \\ &= \sum_{j=1}^{a-1} \binom{n}{j} x^j (1-x)^{n-j} + I_x(a, n-a+1). \end{aligned}$$

Re-arranging this equation gives the desired result.

This theorem shows that if X has a binomial distribution with parameters n and p then $P[X \geq k] = \text{dfbeta}(p, k, n + 1 - k)$.

$\langle \text{dfbeta } 12 \rangle \equiv$

```
function dfbeta(x, p, q)
{
   $\langle \text{dfbeta Constants and Input Validation } 13 \rangle$ 
   $\langle \text{dfbeta Change Tail } 14 \rangle$ 
   $\langle \text{dfbeta Soper's Reduction } 15 \rangle$ 
  beta=loggamma(p)+loggamma(q)-loggamma(p+q);
  betain=betain*Math.exp( pp*Math.log(xx) +(qq-one)*Math.log(cx) -beta)/pp;
  if (index==1) betain=one-betain;
  return(betain);
}
◇
```

Macro referenced in scrap 9a.

The constant acu determines the accuracy of the final result. Invalid input causes the NaN value to be returned, while inputs of $x \leq 0$ or $x \geq 1$ return the obvious values of the distribution function.

⟨dfbeta Constants and Input Validation 13⟩ ≡

```
var one=1.0;
var zero=0.0;
var acu=0.0000001;
if ( (p<=zero) || (q<=zero) ) return(Number.NaN);
if (x<=zero) return(0.0);
if (x>=one) return(1.0);
◇
```

Macro referenced in scrap 12.

Under certain conditions the value of the distribution function at $1 - x$ is easier to compute than the value at x . This section of code makes that change if appropriate. The corresponding adjustments are made just before the function returns.

⟨dfbeta Change Tail 14⟩ ≡

```
var ai, beta, betain, psq, cx, ns, pp, qq, rx, temp, term, xx ;
var index;
betain=x;
psq=p+q;
cx=one-x;
if (p>= psq*x)
  {
    xx=x;
    pp=p;
    qq=q;
    index=0;
  }
else
  {
    xx=cx;
    cx=x;
    pp=q;
    qq=p;
    index=1;
  }
term=one;
ai=one;
betain=one;
ns=qq+cx*psq;
◇
```

Macro referenced in scrap 12.

The reduction formulas of Soper are used for the bulk of the iterative computation.

⟨dfbeta Soper's Reduction 15⟩ ≡

```
rx=xx/cx;
temp=qq-ai;
if (Math.floor(ns)==0) rx=xx;
for(;;)
    {
    term=term*temp*rx/(pp+ai);
    betain=betain+term;
    temp=Math.abs(term);
    if ( (temp <=acu) && (temp<=acu*betain) ) break;
    ai=ai+one;
    ns=ns-1;
    if (Math.floor(ns) >=0)
        {
        temp=qq-ai;
        if (Math.floor(ns) ==0) rx=xx;
        }
    else
        {
        temp=psq;
        psq=psq+one;
        }
    }
```

◇

Macro referenced in scrap 12.

4.3 The bintail Function

The function `bintail(n, p, k)` computes the probability that a binomial random variable with parameters n and p takes the value k or more. The computation uses the connection between this binomial probability and the incomplete beta integral discussed earlier.

⟨bintail 16a⟩ ≡

```
function bintail(n, p, k){
  if (k>n) return 0.0;
  if (k<=0) return 1.0;
  return dfbeta(p, k, (n-k+1));
}
◇
```

Macro referenced in scrap 9a.

4.4 The bnlower Function

The bnlower function computes the lower endpoint of a one sided $100(1 - \alpha)\%$ confidence interval for the binomial number of trials.

The lower endpoint is the smallest value of n for which $P[X \geq k] > \alpha$, where k is the observed number of successes. This probability is a decreasing function of n . The computational strategy is to bracket the desired value of n , and then proceed by bisection.

Notice that $P[X \geq k] = 0$ for $n < k$. If this probability exceeds α when $n = k$, then k is the lower endpoint of the confidence interval. When $k = 0$ the probability $P[X \geq 0] = 1$ even when $n = 0$. This puts the maximum likelihood estimator inside the confidence interval in this case.

⟨bnlower 16b⟩ ≡

```
function bnlower(k, p, alpha)
{
  var high, low, test;
  var testprob;
  high=k;
  testprob=bintail(high, p, k);

  if(testprob>alpha) return high;
  ⟨bnlower bracket 17a⟩
  ⟨bnlower bisection 17b⟩
  return high;
}
◇
```

Macro referenced in scrap 9a.

To bracket the final value of n , begin with a test value `high` set equal to k and double it until the probability first exceeds α . The final value of n must then lie between `high` and `low`.

`<bnlower bracket 17a> ≡`

```
while( testprob <=alpha)
  {
    high=2*high;
    testprob=bintail(high, p, k);
  }
low=Math.floor(high/2);
◇
```

Macro referenced in scrap 16b.

Now use bisection to compute the lower endpoint. The tail probability at `high` always exceeds α , while the tail probability at `low` is α or less. Thus when the difference between `high` and `low` is one, `high` is the desired value.

`<bnlower bisection 17b> ≡`

```
while(high-low>1)
  {
    test=Math.floor((high+low)/2);
    testprob=bintail(test, p, k);
    if(testprob>alpha) high=test;
    if(testprob<=alpha) low=test;
  }
◇
```

Macro referenced in scrap 16b.

4.5 The `bnupper` Function

The `bnupper` function computes the upper endpoint of a one sided $100(1 - \alpha)\%$ confidence interval for the binomial number of trials.

The upper endpoint is the largest value of n so that $P[X \leq k] > \alpha$. The computational method is bisection, as in the case of `bnlower`.

The tail probability is 1 if $n = k$ (even if $k = 0 = n$), so the desired value of n is bracketed by doubling the test value high until the tail probability falls below α .

Since the tail probability exceeds α at low and is less than α at high, when the difference between these two is 1, the value of low is the upper endpoint.

`<bnupper 18>` \equiv

```
function bnupper(k, p, alpha)
{
  var high, low, test;
  var testprob;
  high=k;
  testprob=1.0-bintail(high, p, k+1);
  while( testprob >alpha)
    {
      if (high==0) high=1;
      else
        high=2*high;
      testprob=1.0-bintail(high, p, k+1);
    }
  low=Math.floor(high/2);
  while(high-low>1)
    {
      test=Math.floor((high+low)/2);
      testprob=1.0-bintail(test, p, k+1);
      if(testprob>alpha) low=test;
      if(testprob<=alpha) high=test;
    }
  return low;
}
◇
```

Macro referenced in scrap 9a.