

Estimation of the Binomial Success Probability

Jerry Alan Veeh

October 25, 2005

Contents

1	Introduction	1
2	The HTML Interface	2
2.1	Introductory Comments	2
2.2	Interface Components	3
3	The JavaScript Code	5
4	Functions for Computation	8
4.1	The <code>fix</code> Function	9
4.2	The <code>loggamma</code> Function	9
4.3	The <code>dfbeta</code> Function	10
4.4	The <code>ppbeta</code> Function	15
4.5	The <code>bplower</code> Function	20
4.6	The <code>bpupper</code> Function	20

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 success probability p 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 n being known. Denote by k the observed number of success in n trials. Denote by U the value of p for which $P[X \leq k] = \alpha$. Then the interval $[0, U]$ is a one sided confidence interval for p with confidence level at least $1 - \alpha$. Similarly, if L is the value of p for which $P[X \geq k] = \alpha$, the interval $[L, 1]$ is a one sided confidence interval for p with confidence level at least $1 - \alpha$. A two sided confidence interval for p is found by finding one sided confidence intervals each having confidence level $1 - \alpha/2$. The maximum likelihood estimator of p is known to be k/n .

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

"Binp.html" 2a \equiv

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

"binp.js" 2b \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
Success Probability</title>
<script src="binp.js"></script>
</head>
<BODY>
<h1>
Confidence Intervals for the Binomial
Success Probability
</h1>
<hr>
<p>
This JavaScript program computes the maximum likelihood
estimator of, and conservative confidence intervals for, the
success probability in a sequence of independent Bernoulli
trials. The number of trials 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
<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). Some of the computational methods use
algorithms from <em>Applied Statistics Algorithms</em>. The
annotated source code is <a href="BinomialP.pdf">available</a>
</p>
<hr>
◇
```

Macro referenced in scrap 2a.

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>Number of trials</td>
      <td><input type="text" name="trials" 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 2a.

⟨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 2a.

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 2b.

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

⟨Parse Input 6⟩ ≡

```
var k=0,n=0, c1=95;  
k=parseInt(document.theform.successes.value);  
n=parseInt(document.theform.trials.value);  
c1=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 number of trials is not a number.");
    return;
  }
if (n<=0)
  {
    alert("The number of trials must be a positive integer.");
    return;
  }

if (isNaN(k))
  {
    alert("The number of successes is not a number.");
    return;
  }
if ((k<0)||k>n)
  {
    alert("The number of successes must be between zero "+
          "and the number of trials, 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 5b.

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

⟨Compute 8⟩ ≡

```
var out="";
out += "Here "+k+" successes were observed in "+
      n+" trials.\n";
out += "The MLE of the success probability is "+
      fix(k/n)+".\n";
out += "The lower endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "success probability is "+
      fix(bplower(n,k,(100-cl)/100))+".\n";
out += "The upper endpoint of a one sided "+
      cl+"% confidence interval for the ";
out += "success probability is "+
      fix(bpupper(n,k,(100-cl)/100))+".\n";
out += "The endpoints of a two sided "+
      cl+"% confidence interval for the ";
out += "success probability are "+
      fix(bplower(n,k,(100-cl)/200))+ " and ";
out += ""+fix(bpupper(n,k,(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, as well as the percentage points 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⟩ ≡

```
⟨fix 9b⟩
⟨loggamma 10⟩
⟨dfbeta 13a⟩
⟨ppbeta 16a⟩
⟨bplower 20a⟩
⟨bpupper 20b⟩
◇
```

Macro referenced in scrap 2b.

4.1 The fix Function

The `fix` function provides formatting of the output, and is used instead of built-in JavaScript functions in order to provide compatibility with earlier versions of JavaScript.

⟨fix 9b⟩ ≡

```
function fix(x)
{
  return (Math.round(100000*x)/100000);
}
◇
```

Macro referenced in scrap 9a.

4.2 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 10⟩ ≡

```
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.

4.3 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 $P[X \geq k] = \text{dfbeta}(p, k, n+1-k)$.

⟨dfbeta 13a⟩ ≡

```
function dfbeta(x, p, q)
{
  ⟨dfbeta Constants and Input Validation 13b⟩
  ⟨dfbeta Change Tail 14⟩
  ⟨dfbeta Soper's Reduction 15⟩
  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 13b⟩ ≡

```
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 13a.

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 13a.

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 13a.

4.4 The ppbeta Function

The function `ppbeta(x, p, q)` computes the 100th percentile of the beta distribution with parameters p and q . The algorithm is Algorithm AS64/109 of *Applied Statistics Algorithms*. Some modifications have been made to eliminate the goto statements in the original Fortran implementation.

⟨ppbeta 16a⟩ ≡

```
function ppbeta(x, p, q)
{
  ⟨ppbeta Constants and Input Validation 16b⟩
  ⟨ppbeta Change Tail 17⟩
  ⟨ppbeta Initial Approximation 18⟩
  ⟨ppbeta Newton-Raphson 19⟩
  if (index==1) return(1-xinbta);
  return(xinbta);
}
◇
```

Macro referenced in scrap 9a.

If p or q is invalid, NaN is returned. Out of range values of x return reasonable percentiles.

⟨ppbeta Constants and Input Validation 16b⟩ ≡

```
var    acu=1.0E-14, lower=0.0001, upper=0.9999, const1=2.30753;
var    const2=0.27061, const3=0.99229, const4=0.04481;
var    beta=loggamma(p)+loggamma(q)-loggamma(p+q);
var    xinbta=x;
var    a, adj, g, gg, gx, h, pp, prev, qq, r, s, sq, t, tx, y, yprev, w ;
var    index;
if ( (p<0) || (q<0) ) return(Number.NaN);
if (x<=0) return(0.0);
if (x>=1) return(1.0);
◇
```

Macro referenced in scrap 16a.

Under certain conditions computations are easier using $1 - x$ rather than x . If so, the necessary adjustments are made here and again just before the function returns.

⟨ppbeta Change Tail 17⟩ ≡

```
if (x <=0.5)
  {
    a=x;
    pp=p;
    qq=q;
    index=0;
  }
else
  {
    a=1.0-x;
    pp=q;
    qq=p;
    index=1;
  }
```

◇

Macro referenced in scrap 16a.

An initial approximation to the percentage point is computed for use as a starting value for the iterative procedure. A for loop is used to replace some goto statements in the original Fortran implementation.

⟨ppbeta Initial Approximation 18⟩ ≡

```
for(;;){
r=Math.sqrt(-Math.log(a*a));
y=r-(const1+const2*r)/(1+(const3+const4*r)*r);
if ( (pp>1) && (qq>1) )
{
r=(y*y-3)/6;
s=1/(pp+pp-1);
t=1/(qq+qq-1);
h=2/(s+t);
w=y*Math.sqrt(h+r)/h-(t-s)*(r+5/6-2/(3*h));
xinbta=pp/(pp+qq*Math.exp(w+w));
break;
}
r=qq+qq;
t=1/(9*qq);
t=r*Math.pow(1-t+y*Math.sqrt(t),3);
if (t <= 0)
{
xinbta=1-Math.exp((Math.log((1-a)*qq)+beta)/qq);
break;
}

t=(4*pp+r-2)/t;
if (t<=1)
{
xinbta=Math.exp((Math.log(a*pp)+beta)/pp);
break;
}

xinbta=1-2/(t+1);
break;
}
◇
```

Macro referenced in scrap 16a.

The iterative procedure uses a modified Newton-Raphson method. Once again a for loop is used to replace goto statements.

⟨ppbeta Newton-Raphson 19⟩ ≡

```
r=1-pp;
t=1-qq;
yprev=0;
sq=1;
prev=1;
if (xinbta <lower) xinbta=lower;
if (xinbta>upper) xinbta=upper;

bigfor:
for(;;)
{
  y=dfbeta(xinbta, pp, qq);
  y=(y-a)*Math.exp(beta +r*Math.log(xinbta)+t*Math.log(1-xinbta));
  if (y*yprev <0) prev=sq;
  g=1;

  for(;;){
    adj=g*y;
    sq=adj*adj;
    if (sq >=prev)
      {
        g=g/3;
        continue;
      }
    tx=xinbta-adj;
    if ( (tx <0) || (tx >1) )
      {
        g=g/3;
        continue;
      }
    if (prev <= acu) break bigfor;
    if (y*y <= acu) break bigfor;
    if ( (tx ==0) || (tx ==1) )
      {
        g=g/3;
        continue;
      }
    if (tx==xinbta) break bigfor;
    xinbta=tx;
    yprev=y;
    break;
  }
}
```

◇

4.5 The `bplower` Function

The `bplower` function computes the lower endpoint of a one sided $100(1 - \alpha)\%$ confidence interval for the binomial success probability based on the observation of `suc` successes in `trials` trials.

`<bplower 20a>` \equiv

```
function bplower(trials, suc, alpha)
{
  if (suc==0) return 0.0;
  return ppbeta(alpha, suc, (trials-suc+1));
}
◇
```

Macro referenced in scrap 9a.

4.6 The `bpupper` Function

The `bpupper` function computes the upper endpoint of a one sided $100(1 - \alpha)\%$ confidence interval for the binomial success probability based on the observation of `suc` successes in `trials` trials.

`<bpupper 20b>` \equiv

```
function bpupper(trials, suc, alpha)
{
  if(suc==trials) return 1.0;
  return ppbeta(1-alpha, (suc+1), (trials-suc));
}
◇
```

Macro referenced in scrap 9a.