Fast and Accurate Computation of Binomial Probabilities

Catherine Loader
Lucent Technologies
Room 2C-279
600 Mountain Avenue
Murray Hill, NJ 07974, USA

February 5, 2002

Abstract

An algorithm commonly used to compute binomial probabilities is shown to be numerically inaccurate for large sample sizes. We develop an alternative algorithm based on a saddle point expansion, and show it obeys three common limit theorems, essentially up to machine precision. The speed of the new algorithm is similar to that of the commonly used algorithm. Extensions to other distributions and to cumulative probabilities are briefly discussed.

C code implementing the algorithms in this paper is available from the author's web page,
`http://cm.bell-labs.com/stat/catherine/research.html`.

Keywords: binomial probability; numerical accuracy; saddle point

1 Introduction

The binomial distribution is one of the most commonly used distributions in statistics, with a discrete mass function

$$p(x; n, p) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}; x = 0, \ldots, n. \tag{1}$$

In this paper, it is shown that an algorithm commonly used for computing $p(x; n, p)$ is not accurate for large n. An alternative fast algorithm with improved accuracy is presented in Section 2. Performance under certain large sample limit theorems is discussed in Section 3. Numerical results and timings are presented in Section 4.

2 Computational Algorithms

A direct implementation of $p(x;n,p)$ would multiply out all the factorials and powers appearing in (1). This is accurate even for moderately large n, provided that one is careful to avoid unnecessary underflow and overflow problems. But the computational time for this algorithm is $O(n)$, making it unsuitable for routine use.

An alternative computation is to write (1) as

$$\begin{aligned}\log(p(x;n,p)) &= \log(n!) - \log(x!) - \log((n-x)!) \\ &\quad + x\log(p) + (n-x)\log(1-p). \end{aligned} \tag{2}$$

Since $n! = \Gamma(n+1)$ and the log-gamma function is provided in most math libraries, this provides a convenient $O(1)$ algorithm for computing $p(x;n,p)$. This is the algorithm used by the `dbinom()` functions in S, S-Plus and R[1], and by the `binopdf()` function in Matlab's statistics toolbox. NAG uses an incomplete algorithm similar to that presented below, but misses some crucial parts of the expansion. As a result, NAG restricts arguments to $np(1-p) < 10^6$. This paper appears to provide the first implementation producing stable results for larger parameter values with standard finite-precision arithmetic.

The classic algorithm is numerically inaccurate for large n. To see this, suppose $n = 2 \times 10^6$, $x = 10^6$ and $p = 0.5$. Then $\log(n!) \approx 2.7 \times 10^7$, while $\log(p(x;n,p)) \approx -7.5$. This implies that cancellation in the subtractions in (2) will result in the loss of about seven significant figures of precision. This is quite severe, and gets worse for larger n.

The algorithm recommended in this note is based on a saddle point expansion:

$$p(x;n,p) = p(x;n,x/n)e^{-D(x;n,p)} \tag{3}$$

where the deviance $D(x;n,p)$ is defined as

$$\begin{aligned} D(x;n,p) &= \log(p(x;n,x/n)) - \log(p(x;n,p)) \\ &= x\log(\frac{x}{np}) + (n-x)\log(\frac{n-x}{n(1-p)}). \end{aligned}$$

To implement the saddle point algorithmm, we need accurate methods for computing $p(x;n,x/n)$ and $D(x;n,p)$. Evaluation of $p(x;n,x/n)$ uses the Stirling-De Moivre series:

$$\log(n!) = \frac{1}{2}\log(2\pi n) + n\log(n) - n + \delta(n) \tag{4}$$

where the remainder term $\delta(n)$ has the expansion

$$\delta(n) = \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5} + O(n^{-7}).$$

[1] Current versions of R have adopted methods from this paper

Using this expansion for the factorials in $p(x; n, x/n)$, significant cancellation occurs, and

$$p(x; n, x/n) = \sqrt{\frac{n}{2\pi x(n-x)}} e^{\delta(n)-\delta(x)-\delta(n-x)}. \tag{5}$$

We remark that the expansion (4) is routinely used in evaluating the log-gamma function. With simplifications, mathematical libraries including *Numerical Recipes* (Press, Teukolsky, Vetterling and Flannery 1992, Section 6.1) and the `dbinom.f` routine in SLATEC (1993), use this expansion for computing the binomial coefficient $n!/x!(n-x)!$. But simplifying the binomial coefficient alone is not sufficient for accurate computation of binomial probabilities; parts of the coefficient must be incorporated into the deviance.

Inspection of the deviance $D(x; n, p)$ shows (dependent on the sign of $x - np$) that one of the log terms is positive and the other negative, creating the possibility of loss of significance. To avoid this problem, we write

$$D(x; n, p) = npD_0(\frac{x}{np}) + nqD_0(\frac{n-x}{nq}) \tag{6}$$

where $D_0(\epsilon) = \epsilon \log(\epsilon) + 1 - \epsilon$ and $q = 1 - p$. This function is non-negative for all ϵ. For ϵ close to 1, $D_0(\epsilon)$ can be evaluated through the series expansion

$$npD_0(\frac{x}{np}) = \frac{(x-np)^2}{x+np} + 2x \sum_{j=1}^{\infty} \frac{v^{2j+1}}{2j+1}$$

where $v = (x - np)/(x + np)$. In the author's implementation, this expansion is used for $|v| < 0.1$, or equivalently, $9/11 < \epsilon < 11/9$. For other values of ϵ, $D_0(\epsilon)$ is evaluated directly. The final saddle point algorithm is obtained by substituting (5) and (6) into (3).

Tail Probabilities. Frequently, one is interested in computing tail probabilities,

$$P_{n,p}(X \geq x) = \sum_{y=x}^{n} p(y; n, p).$$

A common implementation is to factor this sum as a product of the marginal probability and a correction factor:

$$P_{n,p}(X \geq x) = p(x; n, p) \sum_{y=x}^{n} \frac{x!(n-x)!}{y!(n-y)!} \left(\frac{p}{1-p}\right)^{y-x},$$

and then compute or approximate $p(x; n, p)$ and the sum separately. Using the methods in this paper improves accuracy in the computation of $p(x; n, p)$, and hence of the tail probability.

Other Distributions. Many common distributions have standard implementations similar to (2), and suffer similar cancellation problems for large parameter values. For example, the Poisson mass function $r(x; \lambda)$ may be computed as

$$\log(r(x; \lambda)) = x \log(\lambda) - \log(x!) - \lambda,$$

which is unstable for large values of λ and x. A more stable algorithm along the lines of (3) is

$$r(x;\lambda) = \frac{1}{\sqrt{2\pi x}} e^{-\delta(x)-\lambda D_0(x/\lambda)}. \tag{7}$$

This approach is easily adapted to distributions such as the hypergeometric, gamma and negative binomial.

Student's t. The Student's t density with ν degrees of freedom is

$$t_\nu(x) = \frac{1}{\sqrt{2\pi(1+x^2/\nu)}} \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\nu/2}} \frac{1}{(1+x^2/\nu)^{\nu/2}}$$

We require an algorithm that is stable both as $\nu \to \infty$ and $\nu \to 0$. This is evaluated using

$$\begin{aligned}
\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\nu/2}} &= \frac{\Gamma(\frac{\nu+3}{2})\sqrt{\nu/2}}{\Gamma(\frac{\nu+2}{2}\frac{\nu+1}{2}} \\
&= \exp(\frac{-D_0(\frac{n}{2},\frac{n+1}{2})}{+}\delta(\frac{n+1}{2}) - \delta(\frac{n}{2}) \\
\frac{1}{(1+x^2/\nu)^{\nu/2}} &= \exp(D_0(\frac{\nu}{2},\frac{\nu+x^2}{2}) - \frac{x^2}{2}).
\end{aligned}$$

The first expansion is used for all ν; the second whenever $x^2 < n$.

3 Limit Theorems

Three common limit theorems for the binomial distribution are

1. Poisson limit:
$$\lim_{n\to\infty} p(x;n,\lambda/n) = \frac{\lambda^x}{x!}e^{-\lambda}.$$

2. Central limit:
$$\lim_{n\to\infty} \sqrt{npq}\cdot p([np+c\sqrt{npq}];n,p) = \frac{1}{\sqrt{2\pi}}e^{-c^2/2}.$$

3. Large Deviation limit. For $\epsilon > 0$,
$$\lim_{n\to\infty} \frac{p(x;n,p)}{p^*(x;n,p)} = 1$$
uniformly for $n\epsilon < x < n(1-\epsilon)$, where
$$p^*(x;n,p) = \sqrt{\frac{n}{2\pi x(n-x)}}e^{-D(x;n,p)}.$$

Numerically, the classic algorithm (2) does not obey any of these limit theorems and ultimately shows divergence as n increases. The saddle point algorithm obeys all three, essentially up to the limits of machine precision.

Proofs:

1. Under the Poisson limit, the binomial algorithm (3) reduces to the Poisson algorithm (7), since $n/(n-x) \to 1$ and $\delta(n)$, $\delta(n-x)$ and the second term of the deviance (6) all converge to 0.

2. Under the central limit, $x/np = 1+c\sqrt{q/np} \to 1$ and $npD_0(x/np) \to qc^2/2$ using just the first term of the series expansion. Likewise, $nqD_0((n-x)/nq) \to pc^2/2$ and $D(x;n,p) \to c^2/2$.

   Note that under the central limit, underflow usually occurs at $n \approx 10^{32}$, since np and $np + c\sqrt{npq}$ will be numerically equal.

3. The large deviation limit is trivial, since $p^*(x;n,p)$ is the saddle point method without the error terms $\delta(n) - \delta(x) - \delta(n-x)$. The error terms converge uniformly to 0 for $n\epsilon < x < n(1-\epsilon)$.

4 Examples

For n odd, we evaluate the sum

$$S(n) = \sum_{x=0}^{[n/2]} p(x;n,0.5)$$

using direct multiplication, the classic algorithm (2) and the saddle point algorithm (3). The exact sum is $S(n) = 0.5$, and accuracy of a numerical evaluation $\hat{S}(n)$ is measured by

$$-\log_{10}|2\hat{S}(n) - 1|$$

which counts the number of decimal digits of accuracy. Figure 1 shows the results for a range of n. In particular, the classic algorithm is inaccurate for large n. The multiplication algorithm is best (usually exact) for $n < 100$. For larger n, the saddle point method is best, and the algorithm maintains high accuracy for all values of n.

The next two examples study the convergence of computed probabilities to theoretical limits. Using either the central limit or large deviation limit,

$$\sqrt{2\pi \times 0.21} \cdot p(0.3n;n,0.3) \to 1.$$

We compute $p(0.3n;n,0.3)$ using both the classic and saddle point algorithms and compute the measure

$$\log_{10}|\sqrt{2\pi \times 0.21} \cdot p(0.3n;n,0.3) - 1|.$$

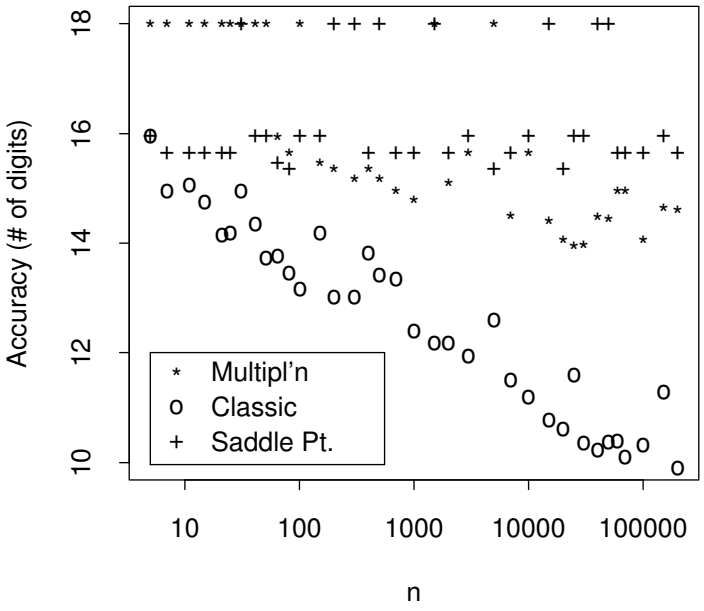

Figure 1: Accuracy of evaulations of $S(n) = \sum_{x=0}^{[n/2]} p(x; n, 0.5)$. '18' denotes accuracy to full machine precision.

This indicates the number of decimal digits agreement between the computed probability and the desired limit. In Figure 2, the saddle point algorithm converges to the desired limit, with about 15 digits agreement at $n = 10^{15}$. The classic algorithm initially shows convergence, but for $n \geq 10^8$ the round-off error dominates and the computed probability diverges.

Figure 3 studies the algorithms under the Poisson limit for $x = 3$ and $\lambda = 2$. The error measure is

$$-\log_{10} |\hat{p}/p_0 - 1|$$

where $p_0 = 4e^{-2}/3$ is the Poisson probability. The results are very similar to the central limit results in Figure 2, with the classic algorithm diverging for $n \geq 10^8$, while the saddle point algorithm improves essentially to machine precision.

Table 1 reports computational time, in microseconds per call, for the algorithms. These are computed using the Linux `time` command, and are averaged

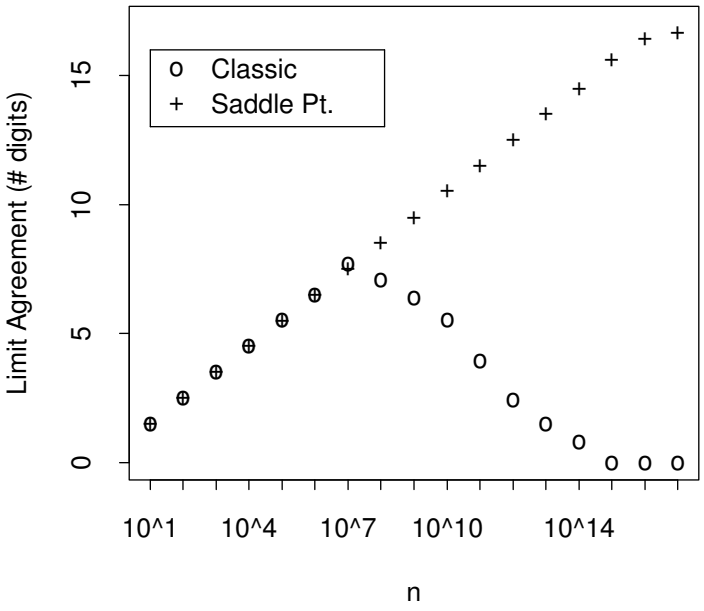

Figure 2: Convergence to the central limit for $p = 0.3$ and $x = np$.

over 10^6 calls. The multiplication algorithm is fastest at $n = 10$ but is not competitive for larger sample sizes. In most cases the classic algorithm and saddle point algorithms have similar timings, and the times change little as n increases.

All results presented in this section were computed on a 400 MHz. Pentium PC running Linux (RedHat 6.0). The computations were performed using double precision floating point arithmetic, which has an accuracy of about 16 decimal digits.

References

Char, B. W., K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan and S. M. Watt (1991). *Maple V Language Reference Manual*. New York: Springer-Verlag.

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1992). *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press.

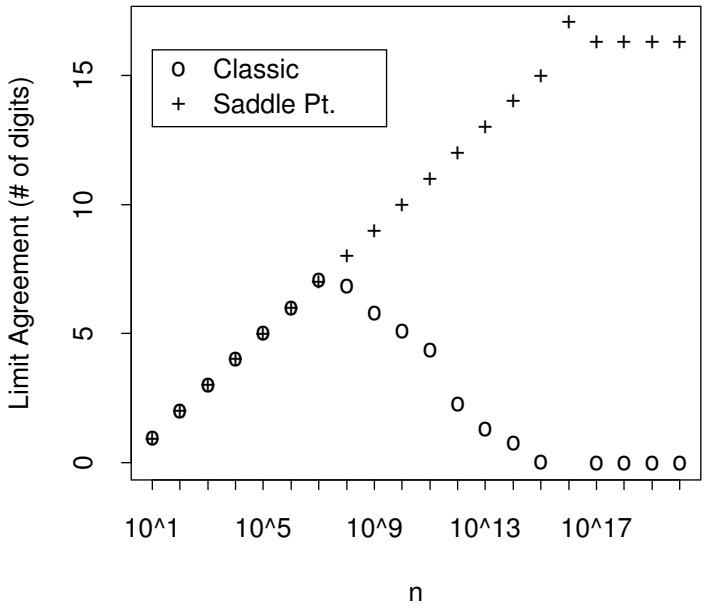

Figure 3: Convergence to the Poisson limit for $x = 3$ and $\lambda = 2$.

SLATEC (1993). Common mathematical library, version 4.1. Netlib Archive. `http://www.netlib.org/slatec`

n	Multiplic.	Classical	Saddle Pt.
10	1.75	4.11	2.97
100	16.53	4.03	4.58
1000	166.80	4.15	4.44
10000		4.53	4.63
100000		4.55	4.64
1000000		4.45	4.55

Table 1: Timings, in microseconds per call, for the multiplication, classical and saddle point algorithms at various sample sizes.

A Saddle Point Algorithm

The following code implements the saddle point algorithm. The entry point is `dbinom(x,n,p)`. Also provided is `dpois(x,lb)` for Poisson probabilities.

The program uses stored values of $\delta(n); n = 1, \ldots, 15$. These were computed in Maple (Char, Geddes, Gonnet, Leong, Monagan and Watt 1991) using the formula

$$\delta(n) = \log(\frac{n!e^n}{n^n\sqrt{2\pi n}}).$$

```
#include <math.h>

/* NTYPE is the type used for the n and x arguments.
   For 32-bit integers, the maximum n is 2^31-1=2147483647.
   If larger n is required, NTYPE must be double.
*/
typedef int NTYPE;

#define PI2 6.28318530717958647692528676

#define S0 0.083333333333333333333       /* 1/12 */
#define S1 0.00277777777777777777778     /* 1/360 */
#define S2 0.00079365079365079365079365  /* 1/1260 */
#define S3 0.00059523809523809523809524 /* 1/1680 */
#define S4 0.00084175084175084175084175 /* 1/1188 */

static double sfe[16] = {
  0, 0.081061466795327258219670264,
  0.041340695955409294093822081, 0.027677925684998339148789292 7,
  0.020790672103765093111522771, 0.016644691189821192163194865 3,
  0.013876128823070747998745727, 0.011896709945891770095055724 1,
  0.010411265261972096497478567, 0.009255462182712732917728636 6,
  0.008330563433362871256469318, 0.007573675487951840794972024 2,
  0.006942840107209529865664152, 0.006408994188004207068439631 0,
  0.005951370112758847735624416, 0.005554733551962801371038689 9
```

```
};

/* stirlerr(n) = log(n!) - log( sqrt(2*pi*n)*(n/e)^n ) */
double stirlerr(n)
NTYPE n;
{ double nn;
  if (n<16) return(sfe[(int)n]);
  nn = (double)n;
  nn = nn*nn;
  if (n>500) return((S0-S1/nn)/n);
  if (n>80) return((S0-(S1-S2/nn)/nn)/n);
  if (n>35) return((S0-(S1-(S2-S3/nn)/nn)/nn)/n);
  return((S0-(S1-(S2-(S3-S4/nn)/nn)/nn)/nn)/n);
}

/* Evaluate the deviance term
   bd0(x,np) = x log(x/np) + np - x
*/
double bd0(x,np)
NTYPE x;
double np;
{ double ej, s, s1, v;
  int j;
  if (fabs(x-np)<0.1*(x+np))
  { s = (x-np)*(x-np)/(x+np);
    v = (x-np)/(x+np);
    ej = 2*x*v;
    for (j=1; ;j++)
    { ej *= v*v;
      s1 = s+ej/(2*j+1);
      if (s1==s) return(s1);
      s = s1;
    }
  }
  return(x*log(x/np)+np-x);
}

double dbinom(x,n,p)
NTYPE x, n;
double p;
{ double lc;
  if (p==0.0) return( (x==0) ? 1.0 : 0.0);
  if (p==1.0) return( (x==n) ? 1.0 : 0.0);
  if (x==0) return(exp(n*log(1-p)));
  if (x==n) return(exp(n*log(p)));
  lc = stirlerr(n) - stirlerr(x) - stirlerr(n-x)
```

```
        - bd0(x,n*p) - bd0(n-x,n*(1.0-p));
  return(exp(lc)*sqrt(n/(PI2*x*(n-x))));
}

double dpois(x,lb)
NTYPE x;
double lb;
{ if (lb==0) return( (x==0) ? 1.0 : 0.0);
  if (x==0) return(exp(-lb));
  return(exp(-stirlerr(x)-bd0(x,lb))/sqrt(PI2*x));
}
```

B Multiplication Algorithm

The routine `dbinom_mult(x,n,p)` evaluates binomial probabilities using the multiplication algorithm, in a method that avoids unnecessary overflow and underflow. For $x \le n/2$, the probability is factorized as

$$p(x;n,p) = \prod_{i=1}^{x} \frac{n-x+i}{i} \prod_{i=1}^{x} p \prod_{i=1}^{n-x} (1-p).$$

Terms from the three products are used in an order to keep the accumulated product as close to 1 as possible, until the first product is exhausted. For $x > n/2$, use $p(x;n,p) = p(n-x;n,1-p)$.

```
double dbinom_mult(x,n,p)
int x, n;
double p;
{ double f;
  int j0, j1, j2;
  if (2*x>n) return(dbinom_mult(n-x,n,1-p));
  j0 = j1 = j2 = 0;
  f = 1.0;
  while ((j0<x) | (j1<x) | (j2<n-x))
  { if ((j0<x) && (f<1))
    { j0++;
      f *= (double)(n-x+j0)/(double)j0;
    }
    else
    { if (j1<x) { j1++; f *= p; }
           else { j2++; f *= 1-p; }
    }
  }
  return(f);
}
```