Fast and Accurate Computation of Binomial Probabilities

Catherine Loader

July 9, 2000

1 Introduction

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

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

In this note, it is shown that an algorithm commonly used for computing $p(x; n, p)$ (including in the statistical packages S, S-Plus and R) 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. Appendix B provides an implementation. But the algorithm is $O(n)$, making it unsuitable for routine use.

An alternative computation is to write (1) as

$$\log(p(x; n, p)) = \log(n!) - \log(x!) - \log((n-x)!) + x \log(p) + (n-x) \log(1-p). \quad (2)$$

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 in S, S-Plus and R. It is also recommended in *Numerical Recipes* (Press, Teukolsky, Vetterling and Flannery 1992, Section 6.1) for computing the binomial coefficient $n!/x!(n-x)!$.

This 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:

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

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

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

Since both $\log(p(x;n,x/n))$ and $-D(x;n,p)$ are negative, there is no loss of precision due to the subtraction in (3).

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

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

Using this expansion for the factorials in $\log(p(x;n,x/n))$, significant cancellation occurs. In particular, the $n\log(n) - n$ terms all disappear. This gives

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

We remark that the expansion (4) is routinely used in evaluating the log-gamma function and with simplifications is used to compute the binomial coefficients by the `dbinom.f` routine in SLATEC (1993). 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

$$D(x;n,p) = npD_0(\frac{x}{np}) + nqD_0(\frac{n-x}{nq}) \tag{5}$$

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 $v = (x-np)/(x+np)$. For other values of ϵ, $D_0(\epsilon)$ is evaluated directly.

Poisson Probabilities. The Poisson distribution has mass function

$$P(X = x) = r(x;\lambda) = \frac{\lambda^x}{x!}e^{-\lambda}.$$

The algorithm similar to (2) has similar cancellations. A more stable algorithm along the lines of (3) is

$$r(x;\lambda) = \frac{1}{\sqrt{2\pi x}} e^{-\delta(x) - \lambda D_0(x/\lambda)}. \tag{6}$$

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 $-p < \epsilon < 1 - p$ fixed, and $x = n(p + \epsilon)$,
$$\lim_{n\to\infty} \frac{p(x; n, p)}{p^*(x; n, p)} = 1$$
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 (6), since $n/(n-x) \to 1$ and $\delta(n)$, $\delta(n-x)$ and the second term of the deviance (5) all converge to 0.

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

   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 to 0.

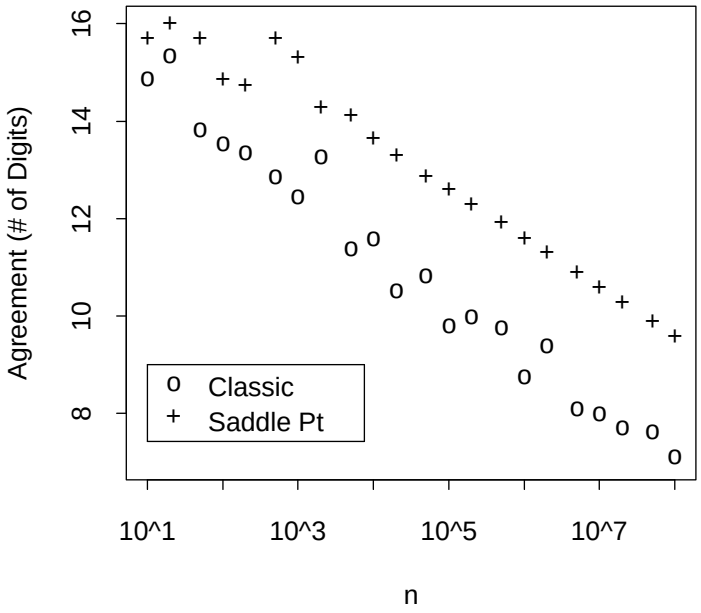

Figure 1: Comparison of computed $p(0.3n; n, 0.3)$ with multiplied-out results.

4 Examples

Our first example computes $p(0.3n; n, 0.3)$ for various values of n, using the multiplication, classic and saddle point algorithms. Letting p_0, p_1 and p_2 be the probabilities produced by the three algorithms, we compute the discrepancy

$$-\log_{10}|\frac{p_j}{p_0} - 1|; j = 1, 2.$$

This essentially counts the number of significant decimal digits the algorithms agree on. Figure 1 shows the results, for n ranging from 10 to 10^8. The classic algorithm shows discrepancy beginning at $n = 100$, while the Saddle point and multiplication algorithms agree up to $n = 1000$. At larger sample sizes, there are substantial discrepancies between all three algorithms, although we cannot tell from this figure which is correct.

Our next example evaluates

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

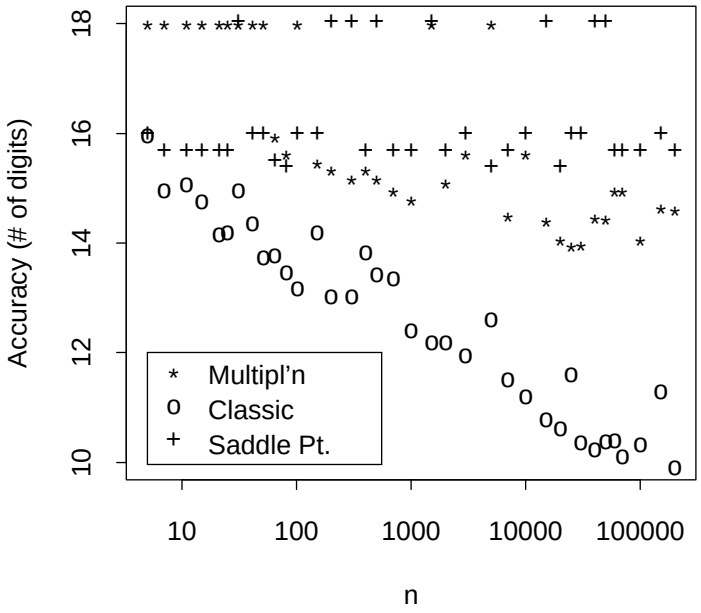

Figure 2: Significant digits of accuracy for a binomial sum. '18' denotes exact to full machine precision.

for n odd. The sum should be 0.5 and numerical accuracy is assessed through the measure

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

which counts the number of decimal digits of accuracy. Figure 2 shows the results for a range of n. In particular, the classic algorithm is quite inaccurate for large n. The multiplication algorithm is best (usually exact) for $n < 100$. For larger n, the saddle point method is best, and maintains its performance up to $n = 200001$. Note that multiplication by $p = 0.5$ can be performed with no loss of precision; performance of the multiplication algorithm may be worse for other values of p.

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

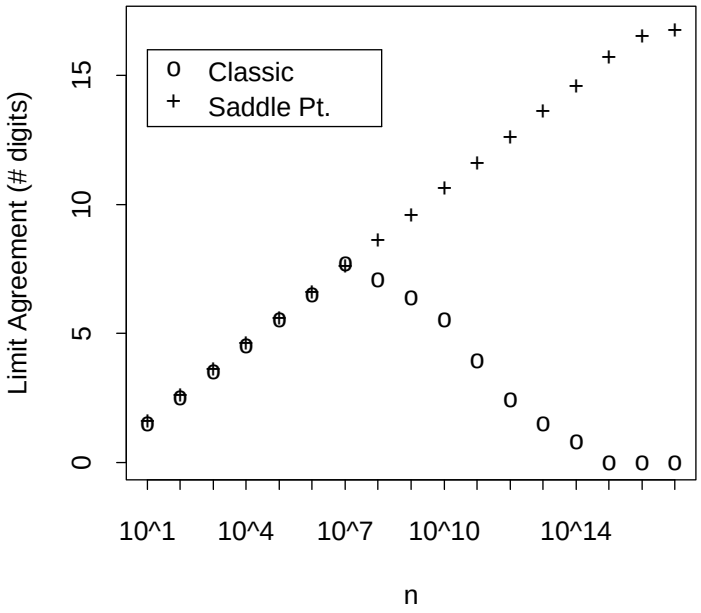

Figure 3: Convergence to the central limit for $p = 0.3$ and $x = np$.

and compute the measure

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

This indicates the number of decimal digits agreement between the computed probability and the desired limit. In Figure 3, 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 sets in, and the computed probability diverges.

Figure 4 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 3, with the classic algorithm diverging for $n \geq 10^8$, while the saddle point algorithm improves essentially to machine precision.

Table 1 reports timings for the algorithms at various sample sizes, averaged 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 is

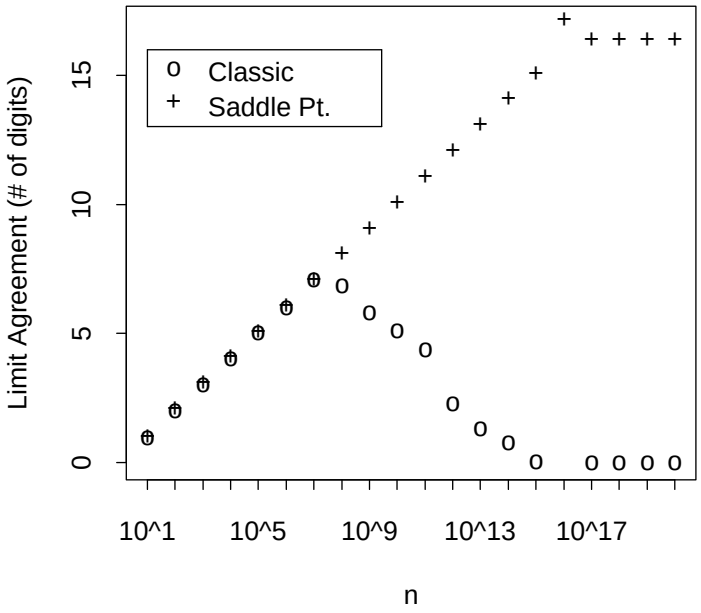

Figure 4: Convergence to the Poisson limit for $x = 3$ and $\lambda = 2$.

slightly faster than the saddle point method (S.P. wins at $n = 10$, since it uses stored values for $\delta(n), n \leq 15$).

All results presented in this section were computed on a 400 MHz. Pentium PC running Linux (RedHat 6.0).

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.
```

n	mult.	class.	S.P.
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 algorithms at various sample sizes.

```
   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,
  0.020790672103765093111522771, 0.016644691189821192163194865,
  0.013876128823070747998745727, 0.011896709945891770095055724,
  0.010411265261972096497478567, 0.009255462182712732917728637,
  0.008330563433362871256469318, 0.007573675487951840794972024,
  0.006942840107209529865664152, 0.006408994188004207068439631,
  0.005951370112758847735624416, 0.005554733551962801371038690
};

/* 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. 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.

```
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);
}
```

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.

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