Memorandum

To:	L3 Distribution
From:	F. Primini
Subject:	Avoiding Numerical Instabilities in Computing Aperture Fluxes
Date:	June 22, 2007

Introduction

In my June 6, 2007 memo, I described the calculation of background-marginalized posterior probability distributions for source intensity quantities (i.e., net counts, rates, photon and energy fluxes). The implementation has indicated that the calculation of the probabilities for the photon and energy fluxes are numerically instable. I describe here an alternate mathematical formulation which avoids these instabilities.

Difficulties in the Original Formulation

Recall the original formula for the probability (eq. 26 in Vinay's memo "Background marginalized X-ray source intensity"):

$$p(s|CB)ds = ds\frac{(rf-g)}{\Gamma(C+1)\Gamma(B+1)}\sum_{k=0}^{C}\sum_{j=0}^{B}\frac{f^k g^j r^{B-j} s^{k+j} e^{-s(f+g)}\Gamma(C+1)\Gamma(B+1)\Gamma(C+B-k-j+1)}{\Gamma(k+1)\Gamma(C-k+1)\Gamma(j+1)\Gamma(B-j+1)(1+r)^{C+B-k-j+1}}$$

If C or B, the counts in source and background apertures, are large, the above expression encounters difficulties in evaluating the Γ function since $\Gamma(n+1)=n!$. Additional difficulties occur if the aperture correction factors f and g are also large. This latter condition is almost always true in computing the probabilities for photon or energy fluxes, since f and g scale with exposure map values. In the test cases run so far, the problems encountered have indeed been in the calculation of the fluxes.

Alternative Formulation

The above difficulties can be addressed by re-writing and simplifying the expression for $p(s|CB)$:

$$p(s|CB)ds = ds(rf-g)\sum_{k=0}^{C}\sum_{j=0}^{B}\frac{(fs)^k e^{-fs}}{k!}\frac{(gs)^j e^{-gs}}{j!}\frac{r^{B-j}\Gamma(C+B-k-j+1)}{\Gamma(C-k+1)\Gamma(B-j+1)(1+r)^{C+B-k-j+1}}$$

Here, I have taken advantage of the above definition of $\Gamma(n+1)=n!$. The first two terms in the summation can now be recognized as Poisson probability functions, which I'll write as

$$P_{pois}(k|fs)=\frac{(fs)^k e^{-fs}}{k!} \quad ,$$

and similarly for $P_{pois}(j|gs)$. The probability distribution for s can now be written as

$$p(s|CB)ds = ds(rf-g)\sum_{k=0}^{C}\sum_{j=0}^{B}\frac{P_{pois}(k|fs)P_{pois}(j|gs)r^{B-j}\Gamma(C+B-k-j+1)}{\Gamma(C-k+1)\Gamma(B-j+1)(1+r)^{C+B-k-j+1}} \quad .$$

Of course, the Γ and Poisson functions can still encounter difficulties for large arguments, but there are well-known techniques for mitigating these. For example, many special functions libraries (including the GNU Scientific Library) contain routines for evaluating both the Γ and $\log\Gamma$ functions, the latter being used for large arguments. The above expression can then be written

$$p(s|CB)ds = ds(rf-g)\sum_{k=0}^{C}\sum_{j=0}^{B}P_{pois}(k|fs)P_{pois}(j|gs)e^{((B-j)\ln(r)+\ln\Gamma(C+B-k-j+1)-\ln\Gamma(C-k+1)-\ln\Gamma(B-j+1)-(C+B-k-j+1)\ln(1+r))}$$

Finally, algorithms exist for evaluating P_{pois} without evaluating $n!$ for large *n.* I include example S-Lang code for this, as well as example code for computing $P(s|CB)$ in the appendix. I've tested this code on cases which failed earlier and obtained good results.

Appendix – S-Lang Code that implements the revised computation of $P(s|CB)$

```
% pois(n,xmu) computes the Poisson probability of obtaining n for a mean xmu,
% without evaluating n! explicitly.

define pois(n,xmu){

  variable i;
  variable sum=1.0;
  variable prb=exp(-1.0*xmu);
  for(i=1;i<=n;i++) sum*=xmu/double(i);
  prb*=sum;
  return prb;

}

% s_pdf_alt1 computes P(s|CB) without evaluating powers of very large numbers

define s_pdf_alt1(C,B,r,f,g,dp){

  % Compute the posterior probability distribution for source counts or intensity     ;
  % marginalized over background, assuming non-informative gamma priors (alpha=1, beta=0) ;

  variable i,j,k;
  variable C1 = C+1;
  variable B1 = B+1;
  variable CB1= C+B+1;

  variable rf_minus_g = r*f-g;
  variable one_plus_r = 1.0+r;

  variable K = Double_Type[C1,B1];
  variable S = Double_Type[C1,B1];

  % Compute MLE value for s, to estimate reasonable range for pdf ;

  variable S_mle = (r*C-B)/rf_minus_g;
  if(S_mle<=0.0) S_mle=30;
  variable ds = S_mle*dp;

  variable s = [0.0:3*S_mle:ds];
  variable ps= Double_Type[length(s)];
  variable fs, gs, pois_fs, pois_gs, tmps;

  for(i=0;i<length(s);i++){
  ps[i]=0.0;
  fs = f*s[i];
  gs = g*s[i];

    for(k=0;k<=C;,k++){
      pois_fs=pois(k,fs);
      for(j=0;j<=B;j++){
          pois_gs=pois(j,gs);
          tmps  =(B-j)*log(r)+lngamma(CB1-k-j)-lngamma(C1-k)-lngamma(B1-j)-\
                    (CB1-k-j)*log(one_plus_r);
          ps[i]+= exp(tmps)*pois_fs*pois_gs;
```

```
      }
    }
    ps[i]*=(rf-g);
  }

  % Renormalize psf just in case;

  return s,ps/(ds*sum(ps)),ds;

} % end of s_pdf_alt1;
```