Trouble With Incomplete Gamma In Numerical Algorithms in C

LHickey

The incomplete gamma function $P(a,x)$ is defined as follows

$$\gamma(a,x) = \int_0^x e^{-t}t^{a-1}dt \qquad \text{(a>0)} \tag{1}$$

$$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)} \tag{2}$$

There is a routine to compute this in the second edition of Numerical Algorithms in C. Suppose $x = 3$ and $a = 2$. Then the provided routine uses the continued fraction form and is incorrect.

What is the answer supposed to be:

$$\gamma(a,x) = \int_0^x e^{-t}t^{a-1}dt \tag{3}$$

$$= -te^{-t} - e^{-t}\Big|_0^x \tag{4}$$

$$= (-xe^{-x} - e^{-x}) - (-1) \tag{5}$$

$$= 1 - e^{-x}(1+x) \tag{6}$$

and if x = 3, then the answer is $1 - 4e^{-3}$

```
echo ’scale=9; 1 - 4 * exp(-3)’|bc -l
0.800851728
```

I used my own simpson rule integrator as we vary x with $\alpha = 2$. you can see the correct values are developed. The 2nd Edition book algorithm book works ok if the series representation part is used, as is the case for $x < 6$, but out past that, it’s broken.

```
    x       P(a=2,x)   Using Simpson
----------- ---------- ------------
  5.977448   0.799161   0.799161
  5.977957   0.799199   0.799199
  5.978676   0.799254   0.799254
  6.001752   1.000000   0.800983
```

```
6.016358   1.000000   0.802070
6.030296   1.000000   0.803103
6.040990   1.000000   0.803892
6.047807   1.000000   0.804394
```

I converted the float to double but left the constants `ITMAX, EPS ,FPMIN` as I found them in the book, but I fiddled around with them and thats not the problem. I think function `gser` is busted.

```
#define ITMAX 100
#define EPS 3.0e-7
#define FPMIN 1.0e-30
double gammln(double xx);
double gamp( double a, double x)
{
   void gcf(double *gammcf, double a, double x, double *gln);
   void gser(double *gamser, double a, double x, double *gln);
   void nerror(char error_text[] );
   double gln;
   if ( x < 0.0 || a <= 0.0) nerror("invalid args in fammp");
   if ( x < (a+1.0) )
   {
      /* use the series rep */
      double gamser = 0.0;
      gser( &gamser, a,x,&gln);
      return (gamser);
   }
   else
   {
      /* use the continued fraction rep */
      double gammcf = 0.0;
      gcf(&gammcf, a,x,&gln);
      return (1.0 - gammcf);
   }
}

void gser( double *gamser, double a, double x, double *gln)
{
   /* returns P(a,x) using series rep,, also rtn ln $\Gamma(a)$ as *gln */
   void nerror(char error_text[] );
   int n;
   double sum,del, ap;
   *gln = gammln(a);
   if ( x <= 0.0)
   {
      if ( x < 0.0) nerror("x < 0 in rtn gser");
      *gamser = 0.0;
      return;
   }
   else
   {
      ap = a;
      del = sum = 1.0/a;
      for ( n=1;n<ITMAX;n++)
      {
         ++ap;
         del *= x/ap;
         sum += del;
         if ( fabs(del) < fabs(sum)*EPS)
         {
            *gamser = sum * exp( -x + a * log(x) - (*gln) );
            return;
         }
      }
      nerror("a too large. ITMAX too small in gser()");
      return;
   }
}
void gcf(double *gammcf, double a, double x, double *gln)
{
   /* returns P(a,x) using continue fracs, also rtn ln $\Gamma(a)$ as *gln */
   void nerror(char error_text[] );
   int i;
   double an,b,c,d,del,h;
   *gln = gammln(a);
   b = x + 1.0 - a;
   c=1.0/FPMIN;
   d=1.0/b;
   h = d;
   for ( i=0;i<ITMAX;i++)
   {
      an = -i * (i-a);
      b += 2.0;
      d = an+ d + b;
      if ( fabs(d) < FPMIN) d=FPMIN;
      c = b + an/c;
      if ( fabs(c) < FPMIN) c=FPMIN;
      d = 1.0/d;
      del = d * c;
      h *= del;
      if ( fabs(del-1.0) < EPS) break;
   }
   if ( i > ITMAX)
      nerror("a too large  ITMAX too small in gcf");
   *gammcf = exp( -x + a * log(x)  -(*gln)) * h; /*  put factors in front */
}

double gammln(double xx)
{
   /* returns log of $\Gamma(xx)$ */
    double x,y,tmp,ser;
    static double cof[6]={76.18009172947146,    -86.50532032941677,
                          24.01409824083091,    -1.231739572450155,
                          0.1208650973866179e-2,-0.5395239384953e-5};
    int j;
    y=x=xx;
    tmp=x+5.5;
    tmp -= (x+0.5)*log(tmp);
    ser=1.000000000190015;
    for (j=0;j<=5;j++) ser += cof[j]/++y;
    return -tmp+log(2.5066282746310005*ser/x);
}
```