# HP Forum Archive 19

[ Return to Index | Top of Index ]

## Gamma (and Factorial) for 12C/12CP

*Message #1 Posted by **Les Wright** on 30 Jan 2009, 2:31 a.m.*

Inspired by Egan Ford's work for the 12C and 12CP (and now his "new" 10C) I thought I would offer an admittedly leaner but less well appointed variation.

If you visit this page of Peter Luschny and scroll down to the section entitled "Which Approximation to Choose?", you find in blue pseudocode the following version of the Stieltjes continued fraction estimate of the Gamma function:

```
Sqrt[ 2*Pi/y ] Exp[ y (Log[y] - 1) + 1/(12*y + 2/(5 y + 53/(42y))) ]
```

Now I have played around with this at some length in Mathematica and am convinced that simplifying the innermost coefficient of 53/42 to 5/4 doesn't compromise overall accuracy too much. So on making the substitution and doing a little rearranging that is friendly to the 12C/12CP, I get this.

```
Exp[ y (Log[y] - 1) + 1/(12*y + 2/(5 (y + 1/(4y)))) ] Sqrt[ 2*Pi ] / Sqrt[y]
```

This gives the following 28 steps for the 12C/CP:

```
STO 0
4
*
1/x
RCL 0
+
5
*
2
x<>y
/
RCL 0
g 12x
+
1/x
RCL 0
g LN
```

```
1
-
RCL 0
*
+
g e^x
RCL 0
g sqrt
/
RCL 1
*
```

The setup is to first store the constant Sqrt [2 Pi] in register 1, which can be done manually or by program code if there are steps left. I like to take advantage of all 12 available digits on the 12CP so this is what I key in: 2.506628274 STO 1 63 EEX 11 CHS STO + 1.

That done, assuming the program is the top one in memory, in run mode I hit f CLEAR PRGM, enter my positive argument, hit R/S, voila. E.g., on my 12CP:

```
5.5 R/S gives 52.34277784 (actual is 52.34277778)
2.25 R/S gives 1.133003995 (actual 1.133003096)
.127 R/S gives 8.540153601 (actual 7.409558064)
10 R/S gives 362880.0000 (exact to 10 digits)
15.8 R/S gives 7.567994848E11 (exact to 10 digits)
```

I don't have a 12C "classic" to test, but I would not expect the full 10 digit accuracy for the larger input values since values on that calculator carry only 10 and not 12 digits on the stack and there would be more rounding error accumulation.

Folks interested in gamma/factorial approximations know that for most (the exceptions being things like the Lanczos and Spouge formulae) accuracy goes down for smaller values but can be improved by repeatedly applying a "shift and divide" trick based on the identity Gamma[x+1] == x Gamma[x]. Egan's routines for 12C/CP and 10C include this but as of yet mine does not. Also I do not yet handle negative arguments, since this requires a way to compute the sine function as required by reflection formula. Egan's 99 step 12C routine builds this in.

Also keep in mind that the little routine computes Gamma straight up, not x!. To get the same behaviour as f x! on the 15C, simply add 1 to the input argument before running this routine.

I hope this little offering is of interest. Keep in mind though it is very similar to Egan's work, it is not exactly the same, computing Gamma rather than x! and using a slightly different formula that, at least in my foolings, seems to be a little more accurate, especially with the user accessible 12-digit precision of the 12CP. I hope this inspires some discussion by special function lovers.

Les

## Re: Gamma (and Factorial) for 12C/12CP

*Message #2 Posted by **Michael Andersson** on 30 Jan 2009, 11:14 a.m.,*
*in response to message #1 by Les Wright*

When it comes to implementing the gamma function on various calculators check http://www.rskey.org (for the HP12c: http://http://www.rskey.org/detail.asp?manufacturer=Hewlett-Packard&model=HP-12C.

/Michael

## Re: Gamma (and Factorial) for 12C/12CP

*Message #3 Posted by **Namir** on 30 Jan 2009, 6:08 p.m.,*
*in response to message #2 by Michael Andersson*

Very true. Viktor uses the gamma function evaluation as his test case for all kinds of calculators. So check out his web site for gamma functions.

Namir

## Re: Gamma (and Factorial) for 12C/12CP

*Message #4 Posted by **Les Wright** on 30 Jan 2009, 7:45 p.m.,*
*in response to message #2 by Michael Andersson*

The Lanczos approximation for the gamma function is ill-suited to the 12C or 12CP since every decimal constant needs to be pre-stored and this is labour intensive. The program itself is also somewhat longer. Going with an asymptotic series or continued fraction approximation with small constants is a lot "tighter". The drawback here is that a Pochhammer shift-and-divide is needed for the smaller arguments to improve accuracy there. Not necessary for Lanczos, which can have high accuracy for all nonnegative reals.

The Lanczos and similar Spouge approximations are actually better suited for high precision or arbitrary precision environments where pre-storing the decimal coefficients poses no huge hardship. For example, Thomas Okken (courtesy of Hugh Steers' library) uses a Lanczos approximation to compute the built-in Gamma and Factorial functions of Free42 to high accuracy.

I expect that once I include some shift-and-divide steps, my little routine will provide as good or better performance as the Toth Lanczos routine, only with lower set up time. I also note that Viktor's treatment of Gamma provides excellent coverage of Lanczos, but I would recommend that one check out Peter Luschny's extensive list of other approximations when seeking options better suited to the step- and register- limited calculators like the 12C, 10C, and 33C/E. The modified Stieltjes CF that I use just happens to be my favourite.

Les

## Enhanced version optimized for 12cp

*Message #5 Posted by **Les Wright** on 31 Jan 2009, 4:25 a.m.,*
*in response to message #1 by Les Wright*

Here is my enhanced routine, where the initial 12 steps carry out the the Pochhammer shift-and-divide correction. Basically, the argument is incremented until it exceeds the integer threshold stored in register 2 (I have been experimenting with 7 thru 10, and seem to like 9 so far), computes the gamma of the incremented argument, and divides out the accumulated Pochhammer product in register 3 at the end. If anyone is familar with Jean-Marc Baillard's routine GAM+ in the HP41 library, I have simply adapted his approach to the Pochhammer shift to the limitations of the 12CP, but, unlike him, I have stuck with the Stieltjes continued fraction for the main computation (whereas he uses a traditional Stirling series):

```
STO 0
1
STO 3
RCL 0
RCL 2
g x<=y?
GTO 013
x<>y
ST* 3
1
ST+ 0
GTO 004
RCL 0
4
*
1/x
RCL 0
+
5
*
2
x<>y
/
RCL 0
g 12x
+
1/x
RCL 0
g LN
1
-
RCL 0
*
```

```
+
g e^x
RCL 0
g sqrt
/
RCL 1
*
RCL 3
/
```

The usage is similar to that reported in my initial post. I store the 12-digit version of sqrt(2Pi) in register 1 and the desired threshold in register 2 (in my case, this means 9 STO 2). And then I compute gamma for positive values exactly as described earlier. Some examples, once I have stored sqrt(2Pi) in R1, 9 in R2, and reset the program pointer with f CLEAR PRGM (in run mode):

```
.25 R/S returns 3.625609908 (all 10 digits correct)
5 R/S returns 24.00000000 (all correct)
3.222 R/S returns 2.478039436 (actual is 2.478039437)
5.5 R/S returns 52.34277778 (all correct)
.5 R/S returns 1.772453851 (this is sqrt(Pi), all digits correct)
.0001 R/S returns 9999.422884 (actual is 9999.422883)
12.771 R/S returns 269174672.9 (actual is 269174673.0)
```

Speed is really good too, though the smaller arguments take a little longer since the Pochhammer shift loop takes a few more trips in its augmentation of the argument. Of course, larger arguments that don't need shift lead to quicker results.

The routine WILL handle negative arguments since the Pochhammer shift steps will move the argument into positive territory, but for negative arguments with larger absolute value this can be rather inefficient. For example, -65.126 R/S will return the correct result of 5.831779435E-91, but it is much slower than for positive values since the Pochhammer shift loop has to go around seventyish some odd times to get the argument above 9. (That said, we are talking only a dozen or so seconds as opposed to a couple.) Programming the sine function to use the reflection formula, as Egan did, is much more efficient.

Also, keep in mind that the impressive accuracy is an advantage of the 12CP over the 12C--stack and register computations use 12-digit numbers all the way through, even though only 10 are displayed. A 12C version of this would NOT give as impressive results since rounding errors affect visible digits, not "hidden" ones.

Folks who insist that the routine return the factorial just like the f x! key sequence on the 15C need only increase the argument by 1, either manually or right at the beginning of the program, since of course Gamma[x+1] = x!

I really love programming the oft maligned 12cp, since I find the 12-digit accuracy and speed of the calculator are strong advantages (though the stubborn keyboard is not). I hope folks enjoy this and will see it as a supplement or alternative to Egan's excellent contribution.

Les

*Edited: 31 Jan 2009, 4:44 a.m.*

## Re: Enhanced version optimized for 12cp

*Message #6 Posted by **Rodger Rosenbaum** on 31 Jan 2009, 6:39 a.m.,*
*in response to message #5 by Les Wright*

Very nifty, Les.

When you get tired of this, I suggest as your next project to code some functions that just aren't available as built in functions on any calculator, AFAIK, such as Bessel functions and Jacobi Elliptic functions.

I did some work on the Jacobi Elliptic functions years ago, but I haven't looked at them lately.

## Re: Enhanced version optimized for 12cp

*Message #7 Posted by **George Bailey (Bedford Falls)** on 31 Jan 2009, 7:51 a.m.,*
*in response to message #6 by Rodger Rosenbaum*

> Quote:
>
> Bessel functions and Jacobi Elliptic functions

For me: the Next-Lottery-Drawing-Function, pleeeeeease! ;-)

## Re: Enhanced version optimized for 12cp

*Message #8 Posted by **Les Wright** on 2 Feb 2009, 1:06 a.m.,*
*in response to message #6 by Rodger Rosenbaum*

Programming Bessel functions and the like is really challenging on our calculators, but I think J-M Baillard has some routines in the HP41C software library.

Due to my interest in statistical distributions I have routines for the incomplete gamma and incomplete beta functions (from which one can derive the error function and the cumulative normal, t, chisquare, and F distributions) for the 41C, 42, and 35S, and with respect to the prolific M. Baillard I actually prefer them over his versions. I have simply adapted the Numerical Recipes routines and I particularly like using more modern techniques for computing the continued fractions, since the centuries old Wallis technique is prone to overflow errors that need to be trapped. Indeed, using the "left" and "right" incomplete gamma functions to compute the complete gamma is a theoretically feasible yet fairly inefficient strategy. But it does work. I have SysRPL versions of my incomplete gamma stuff, but haven't looked at the beta stuff, which is about three times as long.

All of this of course is for amusement. Anyone in their right mind who really needs any of these functions to high accuracy in real life is going to count on something like Maple or Mathematica. Indeed, I use the latter as my gold standard when testing my calculator programs.

Les

## Some further comments

*Message #9 Posted by **Les Wright** on 2 Feb 2009, 12:51 a.m.,*
*in response to message #5 by Les Wright*

I did port this to the Crimson Research emulation of the 12C classic (I don't have a real one) and to a real 33C, and found the accuracy is not as impressive as on the 12cp, which by carrying those two extra digits off the display substantially mitigates rounding error. It still gives about 8 to 9 good digits fairly consistently on the 33C, and fits very well in the meagre 49 steps of the calculator. My gross experience is that it is not quite as good as with the Baillard 41C version, which requires 70 steps, uses the Stirling series taken out to several terms, and gives 9 digits usually and fully 10 digits often. Losing a digit of accuracy seems to be the trade off for a much simpler program that is almost half the size. I would like to see Egan try it out on his 10C and see if he is as happy with it as his own version.

Some time back Valentin posted a challenge asking how one would compute the sine function on a 15C with a missing or malfunctioning SIN key. This lead to a discussion of the very elegant and fascinating factorial (or Gamma) reflection formula. It turns out that if the angle x is in DEGREES (not radians),

```
sin(x) = Pi / Gamma(x/180) / Gamma(1-x/180) = Gamma(.5)^2 / Gamma(x/180) / Gamma(1-x/180)
```

Unfortunately, there is no subroutine return capability on the 12cp, so my gamma routine can't be programmatically called from another program. But it works pretty well when the formula is applied manually, except for angles close to multiples of 90 where rounding error starts creeping in. It also can't be used directly for angles that are multiples of 180, where Gamma(1-x/180) is a pole. But I thought it was cool to point out the relationship.

## Re: Some further comments

*Message #10 Posted by **Egan Ford** on 2 Feb 2009, 6:18 p.m.,*
*in response to message #9 by Les Wright*

> Quote:
>
> Egan try it out on his 10C and see if he is as happy with it as his own version.

I only have one 10C and no emulator. Since my 10C and 12C (classic) gamma versions are the same I should get the same results. I put your version on my 10C and compared to my version on my 12C.

Results:

| n | 50g | 10c Les | 12c Egan (n - 1) |
|---|---|---|---|
| 1.1010 | .950948396 | .950948404 (+.000000008) | .950948397 (+.000000001) |
| 2.2020 | 1.103003815 | 1.103003818 (+.000000003) | 1.103003815 |
| 5.5679 | 58.42037200 | 58.42037204 (+.00000004) | 58.42037190 (-.00000010) |
| 7.7777 | 3230.547120 | 3226.060157 (+.000037) | 3226.060131 (+.000011) |
| 8.8888 | 31802.28101 | 31802.28129 (+.00027) | 31802.12908 (-.15193) |
| 9.1234 | 52556.73243 | 52556.73292 (+.00049) | 52556.73244 (+.00001) |
| 10.0001 | 362961.7210 | 362961.7214 (+.0004) | 362961.7209 (+.0001) |

Both versions vary little except for the 8.8888 test (7.8888 for the 12C). This number is very close to my shift boundary of 8. Hmmm... I'll have to see if 7 or 9 do better.

10C code:

```
01  STO 0      14  RCL 0      27  2          40  SQRT
02  STO 2      15  4          28  *          41  /
03  1          16  *          29  +          42  2
04  STO 1      17  1/X        30  1/X        43  PI
05  RCL 0      18  RCL 0      31  RCL 0      44  *
06  9          19  +          32  LN         45  SQRT
07  X<=Y?      20  5          33  1          46  *
08  GTO 14     21  *          34  -          47  RCL 1
09  X<>Y       22  2          35  RCL 0      48  /
10  STO* 1     23  X<>Y       36  *          49  GTO 00
11  1          24  /          37  +
12  STO+ 0     25  RCL 0      38  E^X
13  GTO 05     26  1          39  RCL 0
```

[ Return to Index | Top of Index ]

Go back to the main exhibit hall