# 140418 NTF Implementations of Entropic Index

## NOTE TO FILE:

Garvin H Boyle

18th April 2014, Revised 24th April 2914

## References:

1. 140209 NTF How to write a function in Excel
2. 140409 NTF Discussion with Dr Yakovenko.
3. 140412 NTF Entropy in a Histogram.
4. 140414 Stirling’s Approximation.xlsm (a spreadsheet).
5. 140414 NTF Stirling’s Approximation.
6. 140415 NTF Entropy Equations with Stirling’s Approximation.
7. Press, WH; Teukolsky, SA; Vetterling, WT; and Flannery, BP; Numerical Recipes in FORTRAN – The Art of Scientific Computing, Second Edition; Cambridge University Press, 1992.
8. 140418 Test of GammaLn() Function.xls (a spreadsheet).
9. 140419 Test of GammaLn For Entropic Index.xlsm (a spreadsheet).
10. Website “Rosetta Code”, where computational implementations of algorithms are given in multiple computer languages. URL( <http://rosettacode.org/wiki/Gamma_function#C> )
11. Website, “Wikipedia”, URL( [http://en.wikipedia.org/wiki/Stirling's\_approximation](http://en.wikipedia.org/wiki/Stirling%27s_approximation) )

## Background

This note covers some material also covered in other notes. The purpose of this note is to record my decision on why and how I plan to implement the computation of an entropic index in both MS Excel and in C++ to get the same results for my “theoretical” analysis in MS Excel and my “empirical” results when I run a real model. This is with the intent to study the computation of various types of entropy in agent-based models (ABMs). So, I must get around the restrictions found in both computational sites.

In short, I plan to write specific custom functions to be used in MS Excel, and similar custom functions in C++ for entropy (S), for the maximum possible entropy (Smax) and for the entropic index (Sindex=S/Smax).

## MS Excel – S, Smax and Sindex

As per Ref 6, the formula for calculating the entropy of a histogram related to an ABM is given by:

where:

* S is entropy,
* ai is the contents of bin i (typically a count of agents),
* A is the sum of the contents of the bins (typically the total number of agents), and K is the number of bins,

and A, K and ai are all positive integers.

Some relevant discoveries:

* MS Excel cannot handle A! for A > 170, so I can use Ln(Fact(A)) as long as A <= 170.
* When A > 170 I will have to write a LawnOfFactorial(A) routine that sums the Ln()s of the factors of A!.
* C++ does have a library function for LN(A), but C++ apparently does not have a library function for the factorial. I will have to write both a Fact(A) program, multiplying the factors, and a LawnOfFactorial(A) function, summing the Ln()s of the factors.

Let:

Then, as per Ref 6, the formula for calculating the maximum entropy of a histogram related to an ABM is given by:

Again, A and K are positive integers, but α is not, and that is a problem. In MS Excel Fact(A) must take an integer. If a non-integer is used as the argument, it is truncated to the largest integer smaller than the argument, WITHOUT ERROR MESSAGE. It took some messing around to figure that out. So, as long as A is a multiple of K, the above approach works, but it introduces distortions when A is not a multiple of K. I have avoided this source of distortions so far in my analysis by always working with integral α, but I want to be more general.

It happens that, in MS Excel, there is a built-in function called GammaLn(A) that computes the Ln() of the Gamma() function, with A as an argument. If A is an integer:

Ln(Fact(A)) = GammaLn(A+1)

And the GammaLn() function seems to be able to take arguments of a size well over a million. So, I think I can use this built-in function in all places within MS Excel where I need the Ln() of a factorial. It will be consistent with those calculations where A is a multiple of K, but will interpolate values otherwise. Good! I explore this option in some detail below.

C++ does not have a library function for GammaLn(). I could probably buy such a library somewhere. However, I found in my Ref 7 book a “recipe” for the GammaLn() function. It’s written in FORTRAN, so I will have to translate it.

This is derived from an approximation developed by Lanczos. From page 207 of Ref 7, here is the FORTRAN code for the required function:

FUNCTION gammln(xx)

REAL gammln, xx

 Returns the value of ln[Γ(xx)] for xx > 0

INTEGER j

DOUBLE PRECISION ser, stp, tmp, x, y, cof(6)

 Internal arithmetic will be done in double precision, a nicety that you can omit if five-figure

 accuracy is good enough.

SAVE cof, stp

DATA cof, stp/76.18009172947146d0, -86.50532032941677d0, 24.01409824083091d0, -1.231739572450155d0, 0.1208650973866179d-2, -0.5395239384953d-5, 2.5066282746310005d0/

x=xx

y=x

tmp=x+5.5d0

tmp=(x+0.5d0)\*log(tmp)-tmp

ser=1.000000000190015d0

do ||

 j=1,6

 y=y+1d0

 ser=ser+cof(j)/y

enddo ||

gammln=tmp+log(stp\*ser/x)

return

END

My translation of that into C++ code is:

double DOMAIN::GammaLn( double xx )

{

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Orrery software; orrery@sympatico.ca; Garvin H Boyle.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Returns the value of ln[Γ(xx)] for xx > 0

// Based on “Numerical Recipes”, 1992, in FORTRAN.

// Note: this implementation is for positive reals only. It includes one call to

// a natural logarithm routine, but is otherwise self-contained.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

 // Declarations

double gammaln; // The value to return.

long j; // A counter.

double x, y; // Temporary values used to construct Prefix and Series.

double Prefix, Series; // Complex components of the answer.

double Root2Pi; // A constant.

double Coefficients[ 6 ]; // The Lanczos coefficients.

 // Ensure xx is in the valid domain.

ASSERT( xx >= 0 );

// Handle special case where the relative error is larger, and the answer may be

// negative.

if( xx == 0.0 ) return 0.0;

 // Load the Lanczos coefficients into the Coefficients [] matrix.

Coefficients [ 0 ] = 76.18009172947146E0;

Coefficients [ 1 ] = -86.50532032941677E0;

Coefficients [ 2 ] = 24.01409824083091E0;

Coefficients [ 3 ] = -1.231739572450155E0;

Coefficients [ 4 ] = 0.1208650973866179E-2;

Coefficients [ 5 ] = -0.5395239384953E-5;

// Load a constant.

Root2Pi = 2.5066282746310005E0;

// Construct the prefix number.

x = xx; // Make a local copy of the argument, non-changing.

Prefix = x + 5.5; // γ = 5.0, γ+1/2 = 5.5.

// Call to natural logarithm routine to compute Prefix.

Prefix = ( x + 0.5 ) \* log( Prefix ) - Prefix;

// Compute the value of the series.

Series = 1.000000000190015;

y = xx; // And another copy of xx, to-be-changed.

for( j = 0; j < 6; j++ ) // Arrays in C++ use zero-origin indexing.

{

 y = y + 1.0;

 Series = Series + Coefficients [ j ] / y;

}

// Put the two pieces together.

gammaln = Prefix + log( Root2Pi \* Series / x );

// Done!

return gammaln;

}

A question is, since this is an “approximation”, how good is it in the range in which I want to use it? Since α can be any real number > 0, I would want it to be accurate for the entire range of positive reals, especially the small ones.

I explored that issue at Ref 8 and the table is shown below. Epsilon is the absolute error between Ln(Fact(a)) and GammaLn(a). But there is a curious issue. The first chart was run on MS Excel 98. The second on MS Excel 2010. I note that the more recent version of the software has a more accurate computation of GammaLn().

This was produced in MS Excel 98.



But, when I load the SAME File into MS Excel 2010 I get this:



In any case, in MS Excel, within the domain in which I can check its accuracy, GammaLn() replicates the results of Ln(Fact()) very, very well. I will have to check the same thing for the C++ code.

So, I can summarize my options for computation of ln(A!) as follows:

|  |  |  |
| --- | --- | --- |
| 1 | Ln(Fact(A)) | MS Excel: Only good for A <= 170; good for A! and ai!; but NOT good for α! (truncates non-integers before computation). |
| C++: Not tested. |
| 2 | (A\*ln(A))-A | MS Excel : Very high relative error for A < 20 (of the order 102); which is where my study is focussed; reasonably low relative error (of the order of 10-5 or less) when A > =170 (of the order of 10-5 or less); tested by implication only when using Sum(pi\*ln(pi)) formula for entropy; not tested for α!, as the use of it removes the need for α!. |
| C++: Very high relative error for A < 20 (of the order 102); which is where my study is focussed; reasonably low relative error when A > =170 (of the order of 10-5 or less); tested by implication only when using Sum(pi\*ln(pi)) formula for entropy; not tested for α!, as the use of it removes the need for α!. |
| 3 | ((A+ξ)\*ln(A))+Ψ-A | MS Excel: Much lower relative error for A < 20, but still not good for A < 2; my own invention, so lacks credibility; not tested for α!. As it turns out, my ξ is close to ½, and my Ψ is close to Ln(Root of 2 Pi). This is a respectable variation on Stirling’s approximation. |
| C++: Not tested. |
| 4 | Ln(Fact(A)) and (A\*ln(A))-A | MS Excel: A hybrid, using formula 1 when A < 170 and formula 2 when A >= 170; this piecemeal function would have reasonable relative error (of the order of 10-5 over most of the domain of interest to me, and better for larger A); would have a minor discontinuity at A = 170; but NOT good for α! when α < 170 and a non-integer. |
| C++: Not tested. |
| 5 | GammaLn() | MS Excel: Good for A!, ai! and α!; excellent relative error (of the order of 10-11 or better); already implemented in MS Excel, so requires no custom functions; has some minor problem when the argument is zero in MS Excel 98, but solved in MS Excel 2010. |
| C++: Does not exist as a library function in MS Visual Studio 2010 C++ native library, which is the one I use, but I have found a source for FORTRAN code which I can translate, and may find an already written and tested version in C++ code. |

Then I can make a formula for the entropic index which is:

* good for both computational engines (MS Excel and C++),
* good for the entire domain of interest [0, ∞] and
* good for both
	+ integer parameters ( A! and ai! ) and
	+ real parameters ( α! ).

This formula will suffer from overflow on most computers:

But this formula should be free from overflow, and should give the same result, IFF the GammaLn(A) implementation accurately replicates the output of ln(A!) for all A ∈ [0, ∞]:

## More Code For GammaLn()

I have found an alternate source for the Lanczos approximation, but I have discovered there are at least three different versions of GammaLn accepted in the “computational science” community: Stirling’s approximation, Lanczos’ approximation, and Sprouge’s approximation.

Drat! This hole just seems to get deeper and deeper. But, aha!. While Stirling’s approximation seems to fail badly in the domain of interest, it seems like Sprouge’s and Lanczos’ techniques produce equally precise and accurate results in the part of the real domain in which I have an interest. So, it will not hurt if I stick with Lanczos’ approximation and ignore Sprouge.

This is from URL(http://rosettacode.org/wiki/Gamma\_function#C ). It is said to be written in BBC BASIC for Windows.

\*FLOAT64

 INSTALL @lib$+"FNUSING"

 FOR x = 0.1 TO 2.05 STEP 0.1

 PRINT FNusing("#.#",x), FNusing("##.############", FNgamma(x))

 NEXT

 END

 DEF FNgamma(z) = EXP(FNlngamma(z))

 DEF FNlngamma(z)

 LOCAL a, b, i%, lz()

 DIM lz(6)

 lz() = 1.000000000190015, 76.18009172947146, -86.50532032941677, \

 \ 24.01409824083091, -1.231739572450155, 0.0012086509738662, -0.000005395239385

 IF z < 0.5 THEN = LN(PI / SIN(PI \* z)) - FNlngamma(1.0 - z)

 z -= 1.0

 b = z + 5.5

 a = lz(0)

 FOR i% = 1 TO 6

 a += lz(i%) / (z + i%)

 NEXT

 = (LNSQR(2\*PI) + LN(a) - b) + LN(b) \* (z+0.5)

The code of interest to me is highlighted. I presume LNSQR() is the natural log of the square root. I don’t understand the “THEN = ” construct. Does it mean “THEN z = ” ? Nor do I understand why it applies the reflection technique when z < 0.5, when the Lanczos approximation is good for z > 0 without the reflection formula. I wonder what BBC BASIC was used for. I recall GW BASIC, and IBM BASIC from way back, and QUICK BASIC and Visual BASIC. This one is new to me.

# Lanczos’ Approximation of Ln(Gamma(z))

So, having decided I will use a C++ version of Lanczos’ approximation of GammaLn() to emulate GammaLn() in MS Excel, I should explore exactly how it works. First, note the following identity, true by definition of the Gamma function.

* z! = Γ(z+1) where z ∈ N.

This equation for Lanczos’ approximation is taken from the Ref 7 “Numerical Recipes” (1992, Press, et. al.), and is described there as a variation on Stirling’s approximation.

where F is an infinite series, truncated, with an unspecified residual, as follows:

* F is an infinite series, here truncated to six terms.
* ε is the residual error.
* ci are coefficients that were determined in some odd fashion not explained.
* γ is a parameter which is set to 5, but may have a range of values. It seems the value of 5 is a best choice of some kind, converging most quickly.

This definition works, with adjustments, it is claimed, for almost all complex numbers z in the complex plane (excepting only zero and negative integers), but I am only interested in real z >= 0. So, I note I may need to impute a value when z = 0.

## Comparison With Stirling’s Approximation

Compare this with other variations on Stirling’s approximation.

1. The version used to simplify the equation for entropy is:

or

1. The version that I came up with empirically is:

or

where ξ is approximately ½ and Ψ is approximately ln(root(2PI)). My empirical value for ξ was just less than ½. I wonder if it can also be interpreted as a derivative of PI, or e.

1. And if you check on Wikipedia you find this version:

URL( [http://en.wikipedia.org/wiki/Stirling's\_approximation](http://en.wikipedia.org/wiki/Stirling%27s_approximation) )

or

## Returning to Lanczos’ Approximation:

So, Lanczos’ approximation seems to be a purpose-built variation of Stirling’s approximation which has a variety of corrective additions, the most obvious being the series F with the oh-so-random coefficients.

I note, before starting, that the following identities hold, by definition of factorial and gamma functions:

* ln( z! ) = ln( Γ( z + 1 ) )
* z! = z( z – 1 )!
* ln( z! ) = ln( z ) + ln( ( z – 1 )! )
* Γ( z + 1 ) = z Γ( z ) and
* ln( Γ( z + 1 ) ) = ln( z ) + ln( Γ( z ) )

To compute the natural logarithm of the above equation for Γ( z + 1 ), the following form is reconstructed **from the Ref 7 code**, rather than from the Ref 7 equation.

I don’t understand where the highlighted “/z” comes from. It’s in the code, but I cannot see it in the equation. This is equivalent to subtracting ln(z) from the value; or dividing Γ(z+1) by z, giving Γ(z). It does not appear to come from the equation, and can probably be explained in reference to the above-listed identities, though I cannot see it. **But it does seem to work, as shown below.** This raises the question “IS THE EQUATION INCORRECT?” I have heard that the Ref 7 book was criticized as having some errors. Was this one of them, or am I just missing something?

So, I took the Ref 7 code for GammaLN() and created a column in an Excel spreadsheet for each variable, and each change of value in each variable within a loop, so I could see how the code produces the answer, then I compared the output of this routine with the built-in functions. The results are in the Ref 9 spreadsheet, and copied below in three tables.

First table – unpacking the argument – calculating the variable “tmp”:



Series calculation – computation of the value of the variable ser:



This is the important table. In the following table:

* gammaln is computed with the routine shown in the Ref 7 “Numerical Recipes” (NR), but via an MS Excel spreadsheet
* x is the input
* GammaLn(x) is calculated using the built-in MS Excel function, so I can compare the NR routine’s output with the output of the built-in function.
* Fact(x) is the built-in factorial function.
* Ln(Fact(x)) is a composite of two built-in functions.
* RelError is ( GammaLn(x) – gammaln ) / gammaln. Note that the relative error is always very small, so the routine from NR seems to replicate the action of the built-in GammaLn(x) very well.
* Gamma(x) is computed as exp(gammaln), exp() being a built-in funvtion, just as a check. Indeed, it seems that Fact(x) = Gamma( x + 1 ).



## Conclusions:

* The built-in GammaLn() function of MS Excel 2010 replicates the output of the composite of the built-in functions Ln(Fact()) exactly, in the range of primary interest, so I am confident I can use the built-in GammaLn();
* The Ref 7 NR routine with the unexplainable “/z” insertion in the code seems to replicate the built-in GammaLn() function of MS Excel 2010 with relative error of the order of 10-13 over the same range. In fact, not shown above, but I tested it for values well in excess of 1 million. Good enough for me!

So, I will use the GammaLn() version of the formula for Sindex.

In MS Excel I will write custom functions, using the Ref 1 procedures, for S, Smax, and Sindex for K = {2, 3, 4, 5, … }. Each of these will be based on use of the built-in GammaLn() function and my revised formulae shown at the beginning of this note. I can put these into a Visual BASIC module that can be exported from the original source and imported into any spreadsheet. In this way, I will ensure that my computations are the same across all analytical spreadsheets.

In C++ I will write C++ code for GammaLn() based on Lanczos’ approximation, as discussed above, based on the NR FORTRAN code as translated into C++, and I will alter the code for all histograms (and bins) currently in place to use in all computations. I will test the output against baseline data generated by MS Excel. All current formulae and calculations of S, Smax and Sindex based on Stirling’s approximation will be replaced with the formulae shown at the beginning of this note.

This should be sufficient to remove any bias that may be creeping in due to the poor accuracy of Stirling’s approximation for small arguments.