LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-222526

Sampling the Number of Neutrons Emitted per Fission

D. E. Cullen

June 30, 2006

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

UCRL-TR-222526

Sampling the Number of Neutrons Emitted per Fission

by
Dermott E. Cullen
University of California
Lawrence Livermore National Laboratory
P.O.Box 808/L-159
Livermore, CA 94550

Tele: 925-423-7359
E.Mail: cullen1@llnl.gov
Website: http://www.llnl.gov/cullen1

May 1, 2006

Introduction

I define in detail the model used to **sample the number of prompt neutrons emitted in fission**; this description is based on publications defining the model [1] as well as publications comparing the model to experimental measurements [2]. The model described in these publications is exactly what the TART [3] Monte Carlo transport code uses.

Based on comparisons between TART [3] and MCNPX [4], it is obvious that at the time this report was published these two computer codes are not using the same model, and the results significantly differ. It is my hope that this report will contribute toward better understanding of this model, and hopefully eventually to agreement between TART and MCNPX results.

Partial success has already been achieved in the sense that based upon reading a preliminary version of this report, John Hendrichs [5], one author of MCNPX, acknowledged that the sources of differences as described in this report demonstrate an error in MCNPX (John even offered me the traditional $20 reward for reporting an error in MCNPX; I declined to accept). John is presently updating MCNPX to eliminate these sources of differences; hopefully in the not too distant future this correction will be available in MCNPX, and we will obtain agreement between TART and MCNPX, which is the ultimate objective of this report.

Overview

In nucleus fission both prompt and delayed neutrons are emitted. Prompt neutrons are emitted instantaneously at the time of the fission, whereas delayed neutrons are emitted at later times due to the decay of the fission products. In any given fission event (either spontaneous or induce by a neutron) a variable number of neutrons are promptly emitted: 0, 1, 2, 3,…. For most applications we are only interested in the average number of neutrons emitted, what we call "nu bar" $< \nu(E) >$, which is defined by summing over many fission events.

However, for some applications we are interested in the complete probability distribution for the emission of 0, 1, 2, 3… prompt neutrons. There is a long history of publications stretching back 50 years or more, dealing with both the theoretical model for this distribution, and verification of the model compared to experimental measurements. By now the model is well established and verified, and in this report we do not question the model, nor do we make any contributions to improving the model. Here we are strictly interested in detailing how this model is used in neutron Monte Carlo transport calculations; specifically we provide details of how this model is used in the TART and MCNPX Monte Carlo neutron transport codes and we compare TART results to MCNPX results and explain the source of differences.

Why Look at this Model Now?

Recently we have been comparing results using the Monte Carlo neutron transport codes TART [3] and MCNPX [4]. This model is quite simple, well documented and known for many years, and yet we still find significant differences in the results produced by these codes. Therefore this report is interested to investigate why the results differ and hopefully eventually eliminate these differences.

The Model

The model is quite simple; it says that the distribution of the number of neutrons emitted is a Gaussian, the average value of which is the average number of neutrons emitted $< \nu(E) >$, and the width (W) is a constant that within the accuracy of the model, can be universally applied to any isotope,

$$P(z)dz = \mathrm{Exp}[-z^2/2]dz$$

$$z = [\nu - < \nu >]/W$$

Inverting to define ν,

$$\nu = W*z + < \nu >$$

Physically the sampled ν cannot be negative, i.e., fission can only emit some real number of neutrons. This means we are actually sampling from a "truncated" Gaussian, truncated in the sense z does not extend from $-\infty$ to $+\infty$, but rather only over the range where ν is non-negative, i.e., z extends from $-<\nu>/W$ to $+\infty$.

Below we will discuss how to define the width W, and additions to this model, but basically this is the entire model. The corrections that we will add to this model include,

1) **The Floor Correction:** Our distribution defines the number of neutrons emitted as a continuous variable, whereas physically each fission can only emit an integer number of neutrons, 0, 1, 2, 3…. For use in our Monte Carlo calculation it is actually the probability distribution for an integer number of numbers that we use.

2) **The Truncation Correction:** This insures that the distribution that we sample reproduces the average number of prompt fissions emitted per fission, $<\nu>$.

The Definition of the Width (W)

There is some confusion between papers and codes as to the definition of the width to be used in this model. The confusion arises because all papers and codes agree that "ν is distributed as a Gaussian with width W". However, this can be interpret to mean,

$P(z)dz = Exp[-z^2/2]dz$
or,
$P(z)dz = Exp[-z^2]dz$

The only difference being the factor of "2" in the definition. Either definition can be used as long of the width for each distribution is correctly defined. In both case we have,

$z = [\nu - <\nu>]/W$

Inverting to define ν,

$\nu = W*z + <\nu>$

Note, that this only depends on the product W*z. Using the first form with the "2" we have the most quoted value of **W = 1.08**. Using the second form without the "2" decreased the samples value of z by $\sqrt{2}$, and to compensate for this we merely increase the definition of the width by a factor of $\sqrt{2}$.

In summary either model can be used as long as W is defined appropriately,

$P(z)dz = Exp[-z^2/2]dz$: W = 1.08 (this is used by MCNPX)
or,

$P(z)dz = Exp[-z^2]dz$: $W = 1.08\sqrt{2} = 1.52735$ (this is used by TART)

Let me stress that the different definitions of width used by MCNPX and TART do not lead at any differences in the sampled results; in both codes W*z are identical and that is all that the final sampled distribution depends on.

The Floor Correction: Distribution for an Integer Number of Neutrons Emitted

As actually used in our Monte Carlo Calculations in order to simulate analog fission events we only emit an integer number of neutrons for each fission event. This means that rather than the continuous distribution of ν described above $P(\nu)$, $\nu = 0$ to $+\infty$, we are really only interested in the discrete distribution, $P(\nu)$, $\nu = 0, 1, 2, 3, 4\ldots$such that the average value of our sample is equal to our known average $<\nu>$,

$<\nu> = \dfrac{\sum \nu * P(\nu)}{\sum P(\nu)}$, where the sum over integer values of ν extends from 0 upwards.

When we use our model we will first sample a value of ν on a continuous basis (a floating point result), and then round the results down to an integer to define the integer number of neutrons emitted (the integer floor value). This means that compared to the continuous values sampled, on average we will decrease the sampled value by 1/2. To offset this reduction we will add 1/2 to the values sampled by our model,

$P(z)dz = Exp[-z^2/2]dz$

$z = [\nu - (<\nu> + 1/2)]/W$

Inverting to define ν,

$\nu = W*z + <\nu> + 1/2$

The Truncation Correction: Normalized Distribution with Correct Average Value

For sampling we need a normalized distribution, such that we conserve the known average number of fissions, $<\nu>$. For use in our Monte Carlo calculations we are interested in conserving $<\nu>$, based on rounding all sampled continuous values of ν downward to an integer (the floor). For simplicity in the following equations I will not include the factor of 1/2 until we have the final results. Starting from our basic model, the average value is by definition,

$$<\nu'> = \int \nu \ Exp[-\{(\nu - <\nu>)/W\}^2/2]\, d\nu \Big/ \int \ Exp[-\{(\nu - <\nu>)/W\}^2/2]\, d\nu$$

Where the limits of integration are $\nu = 0$ to $+\infty$. Changing variables to,

$z = [\nu - <\nu>]/W \quad : \quad \nu = W*z + <\nu> \; : \; d\nu = W*dz : z = -<\nu>/W \text{ to } +\infty$

$$<\nu'> = \int [W*z + <\nu>] \; \text{Exp}[-z^2/2]*W*dz \; / \int \text{Exp}[-z^2/2]*W*dz$$

$$<\nu'> = <\nu> + W \int z \; \text{Exp}[-z^2/2]*dz \; / \int \text{Exp}[-z^2/2]*dz$$

$$<\nu'> = <\nu> + W \int \text{Exp}[-z^2/2]*dz^2/2 \; / \int \text{Exp}[-z^2/2]*dz$$

$$<\nu'> = <\nu> + W*\text{Exp}[-z^2] \; / \; \{ \sqrt{\pi/2} \; [2 - \text{Erfc}(z)] \} \quad : z = <\nu>/W/\sqrt{2}$$

For large $<\nu>/W$ the second term on the right reduces to zero and we have equality between the sampled $<\nu'>$ and the known average value $<\nu>$, in which case no correction factor is required. However, for smaller values of $<\nu>/W$ the second term is significant. Since it is always positive, it will make a positive contribution to the average, and the sampled $<\nu'>$ will be larger than the known average value $<\nu>$. As mentioned above, the above derivation does not include the factor of 1/2 needed for the floor correction; from this point on I will include this factor in the following equations.

In order to insure that the sampled value is always equal to the known average value we must subtract the term,

$$C(z) = -W*\text{Exp}[-z^2] \; / \; \{ \sqrt{\pi/2} \; [2 - \text{Erfc}(z)] \} \quad : \; z = (<\nu> + 1/2)/W/\sqrt{2}$$

With this additional term our model becomes,

$$P(z)dz = \text{Exp}[-z^2/2]dz$$

$$z = [\nu - (<\nu> + 1/2 + C)]/W$$

Inverting to define ν,

ν **= W*z +** $<\nu>$ **+ 1/2 + C** : remembering that C is defined as a negative quantity.

This is the final form, including all corrections that we will actually use in our applications. **Furthermore it is important for the reader to understand that this is exactly the model proposed by Terrell in his paper [1]; nothing has been added or deleted.**

An Empirical Definition of the Truncation Corrections

Above I introduced the basic idea for why we must use a truncation correction term to obtain the correct average value, $<\nu>$. But life is not as simple as described above, because once we include this correction term C, we change the limits of the integral,

which are defined as where the sampled ν is non-negative. The limits of the integral before and after we add our correction factors are,

Before: $z = -(<\nu> +1/2)/W$ to $+\infty$

After: $z = -(<\nu> +1/2 + C)/W$ to $+\infty$

$C(z) = -W*Exp[-z^2] / \{ \sqrt{\pi/2}\ [2 - Erfc(z)] \}$: $z = [<\nu> + 1/2 + C]/W/\sqrt{2}$

Since C now appears in the definition of the limit we have a transcendental equation that is difficult to use in its exact form. In addition this correction factor C will now to used to correct for both truncating the Gaussian and for rounding downward to a integer. For use in our Monte Carlo transport codes where we sample billions of neutrons we need a simple, fast means to define an approximate correction factor. For this purpose the above derivation tells us the form of the correction as far as being a function of $(<\nu> +1/2)/W$. Starting from this we can empirically define the correction terms as,

$$A = Exp[- (\frac{<\nu> +1/2}{W})^2]$$

$C(<\nu>) = -\beta*A/[1 - A]$

Where β is defined to accurately reproduce the average value $<\nu>$ over our range of interest. Below we will illustrate that this analytical correction factor can be used over the entire range of interest of $<\nu>$.

Range and Accuracy of $<\nu>$

By examining all of the available $<\nu>$ data we find that the smallest values for neutron induced fission are slightly above 2.0, and for spontaneous fission slightly above 1.7. So that for use in our applications we want to define the "best" truncation correction factor for $<\nu>$ = 1.7 and larger.

Another point to consider is how accurate do we need to reproduce the average values of $<\nu>$. For use in our applications we require the prompt neutrons per fission that we sample by this model to be much more accurate than the contribution of delayed neutrons, and much more accurate than the uncertainty in $<\nu>$. For example, in our applications it is important for us to distinguish between the prompt neutrons that are correlated, since they are all emitted at the same time, and delayed neutrons that are not correlated, since they are emitted at a later time, and also we want to reproduce criticality values for K-eff whether we are using $<\nu>$ directly or using the method described here to sample ν for each fission. The delayed neutron fraction is roughly 0.5% for uranium and 0.3% for plutonium. Today we can calculate K-eff to roughly an accuracy of +/- 0.1%. We require that the prompt average neutrons be accurate to at least an order of magnitude better than the delayed fraction. With the correction term defined here we can

reproduce the correct average prompt neutrons per fission to roughly 0.01% or better, which is well within the range of our requirements, e.g., see the table below.

How Important are the Correction Factors?

We can determine how important these correction factors are by calculating results over the entire range of interest of $< \nu >$, starting at the lowest value of interest, 1.7, where the correction factor will be most important, and increasing in small steps to larger values of $< \nu >$ where the correction factor becomes negligible. In the below table

$< \nu >$ - the average value we want to conserve in sampling
sample – the average of floating point ν sampled
floor - the average of integer ν sampled
$< \nu >$-floor – the difference between what we want and what we sample
%Difference - % difference between $< \nu >$ and floor
Correction – the correction factor C($< \nu >$) used for this value of ν
Seconds - the time to sample 100 million times.

The important points to note from this table include,

1) Due to our inclusion of the 1/2 in our model, the average floating point sample is always roughly 1/2 more than $< \nu >$; this is what we expect.
2) With our correction factor $< \nu >$ is reproduced by the floor (integer values) to within roughly 0.01% over the entire range of $< \nu >$.
3) The correction factor is important up to fairly large values of $< \nu >$, and as such must be included to obtain accurate results.
4) The timing indicates that on a 3.2 GHz PC we can calculate roughly 10 million samples per second.

$< \nu >$	sample	floor	$< \nu >$-floor	%Difference	Correction	Seconds
1.70	2.195661	1.699874	0.000126	0.00741 %	-0.067144	11.812
1.80	2.296712	1.800188	-0.000188	-0.01047 %	-0.054004	11.859
1.90	2.397242	1.899961	0.000039	0.00206 %	-0.043238	11.828
2.00	2.497977	2.000134	-0.000134	-0.00668 %	-0.034443	11.719
2.10	2.597978	2.099758	0.000242	0.01152 %	-0.027284	11.719
2.20	2.698536	2.199895	0.000105	0.00479 %	-0.021484	11.641
2.30	2.798663	2.299689	0.000311	0.01352 %	-0.016810	11.594
2.40	2.899067	2.399819	0.000181	0.00753 %	-0.013064	11.531
2.50	2.999337	2.499948	0.000052	0.00208 %	-0.010082	11.562
2.60	3.099471	2.599957	0.000043	0.00167 %	-0.007724	11.594
2.70	3.199421	2.699764	0.000236	0.00876 %	-0.005873	11.547
2.80	3.299716	2.799983	0.000017	0.00059 %	-0.004431	11.578
2.90	3.399686	2.899889	0.000111	0.00384 %	-0.003317	11.516
3.00	3.499694	2.999835	0.000165	0.00549 %	-0.002463	11.516
3.10	3.599827	3.099944	0.000056	0.00180 %	-0.001814	11.500
3.20	3.700017	3.200087	-0.000087	-0.00272 %	-0.001325	11.547
3.30	3.799915	3.299914	0.000086	0.00259 %	-0.000960	11.562
3.40	3.899945	3.399978	0.000022	0.00065 %	-0.000690	11.547
3.50	3.999938	3.499987	0.000013	0.00038 %	-0.000492	11.531
3.60	4.100451	3.600494	-0.000494	-0.01372 %	-0.000347	11.641
3.70	4.199814	3.699859	0.000141	0.00380 %	-0.000243	11.578
3.80	4.300082	3.800152	-0.000152	-0.00401 %	-0.000169	11.594
3.90	4.399995	3.899978	0.000022	0.00056 %	-0.000116	11.547
4.00	4.500089	4.000099	-0.000099	-0.00247 %	-0.000079	11.562
4.10	4.600272	4.100291	-0.000291	-0.00709 %	-0.000054	11.562
4.20	4.699972	4.199956	0.000044	0.00105 %	-0.000036	11.531
4.30	4.800015	4.299975	0.000025	0.00059 %	-0.000024	11.531

4.40	4.899895	4.399893	0.000107	0.00244 %	-0.000016	11.594
4.50	5.000014	4.500056	-0.000056	-0.00125 %	-0.000010	11.578
4.60	5.100040	4.600077	-0.000077	-0.00168 %	-0.000007	11.609
4.70	5.200035	4.700005	-0.000005	-0.00011 %	-0.000004	11.609
4.80	5.299977	4.799994	0.000006	0.00012 %	-0.000003	11.547
4.90	5.399910	4.899925	0.000075	0.00152 %	-0.000002	11.609
5.00	5.500073	5.000085	-0.000085	-0.00171 %	-0.000001	11.562
6.00	6.499982	5.999996	0.000004	0.00007 %	0.000000	11.625

Comparison between TART and MCNPX

There are significant differences between how TART and MCNPX sample ν, that are important for code users to understand.

1) MCNPX samples $\mathrm{Exp}[-z^2/2]dz$, W = 1.08; TART samples $\mathrm{Exp}[-z^2]dz$, W = 1.52735. As described above, this difference is compatible and they would both produce the same distribution if this were the only difference.

2) MCNPX uses a tabulated correction factor C, that does an excellent job of conserving $<\nu>$, comparable to the analytical expression used by TART. But be aware that this table is a fit based on using exactly the sampling method described here; any changes in the sampling method would require a different C table.

3) Instead of adding 1/2, as TART does, MCNPX adds a random number
   TART: ν = W*z + [$<\nu>$ +1/2 + C]
   MCNPX: ν = W*z + [$<\nu>$ +random + C]
   This significantly changes the distribution from a simple Gaussian sample,
   TART: $G(\nu) = \mathrm{Exp}\{-[\nu - (<\nu> +1/2 +C)/W]^2\}$
   MCNPX: $G(\nu) = \int \mathrm{Exp}\{-[\nu - (<\nu> +x +C)/W]^2\}dx$, x = 0 to 1
   As we can see this is not a simple Gaussian; it is a difference between ERF functions. The resulting distribution is significantly wider than the simple Gaussian, as shown in the below figures.

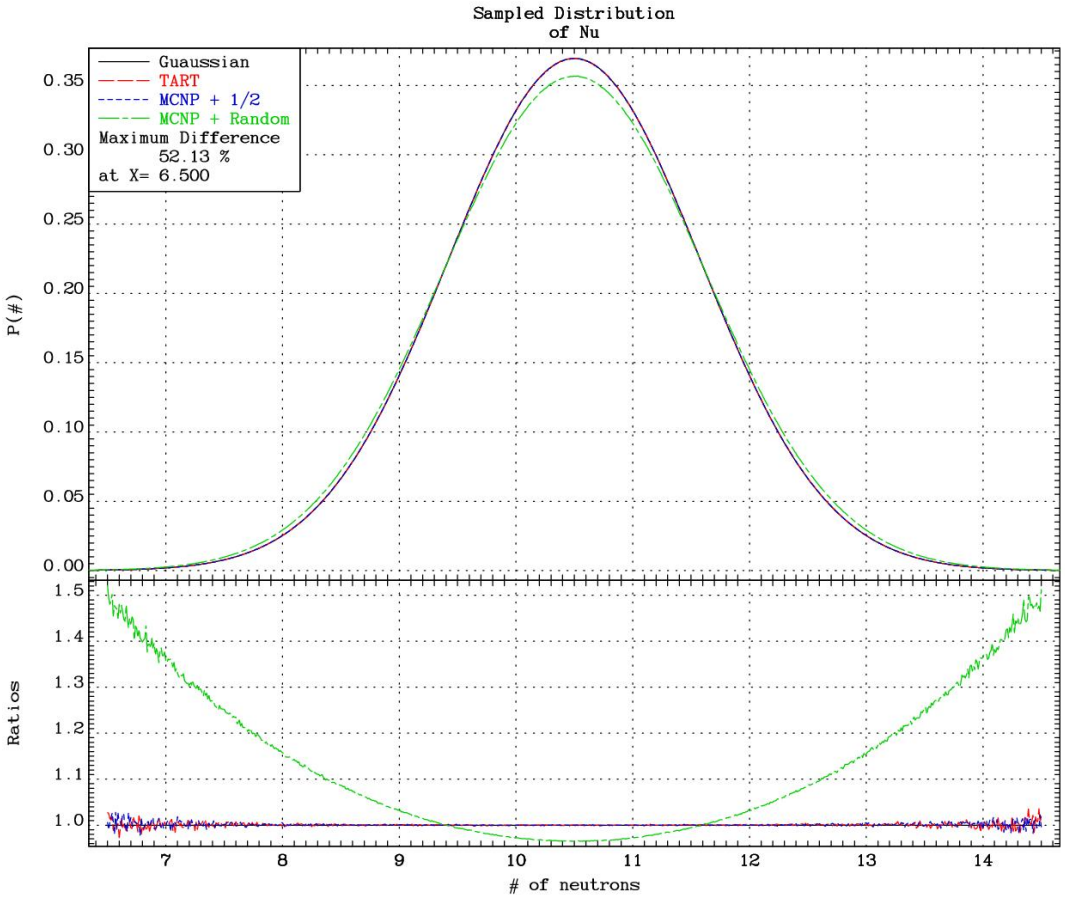
4) If the sampled ν is less than 0, MCNPX returns 0, whereas TART rejects and samples again. This further changes the sampled distribution,
   TART: $G(\nu)$, the Gaussian, as defined above
   MCNPX: c1*$G(\nu)$ + (1 – c1)*$\delta(\nu)$
   The result of returning 0 is to overestimate the probability of emitting 0 neutrons, and since this is a normalized distribution, underestimate the probability of emitting more than 0 neutrons.

5) The MCNPX routine takes 60% more time per sample than the TART routine, e.g., for 100 million samples, TART 11.5 seconds, MCNPX 18.7 seconds.

To better appreciate the effect that these differences have on the results below I present a few comparisons of the continuous sampled distributions.

Floor Correction of 1/2 (TART) or random (MCNPX)

The first plot compares the continuous sampled distribution to an analytical Gaussian. In this case all of the results correspond to $< \nu >$ = 10; for this high value of $< \nu >$ there is virtually no affect of the truncation correction factor C. For the first three curves 1/2 is used as the floor correction, and all three curves are in excellent agreement (Gaussian, TART, MCNP +1/2). For the last curve we use the MCNPX procedure to add a random number rather than 1/2; this is the only difference between the sampling methods, and we can see that the MCNPX procedure significantly broadens the distribution. Compared to the analytical Gaussian, the broader MCNPX results increase the probability of emitting 8 neutrons is about 15% and 7 neutrons by 35%; even more important it decreases the most probable number of neutrons emitted, defined by the peak of the curve, by 3.5%. **The important point to remember here is that the analytical Gaussian is exactly what Terrell proposed in his model, and this can only be reproduced by correcting for floor values by adding 1/2 (what TART does) as opposed to a random number (what MCNPX does).**



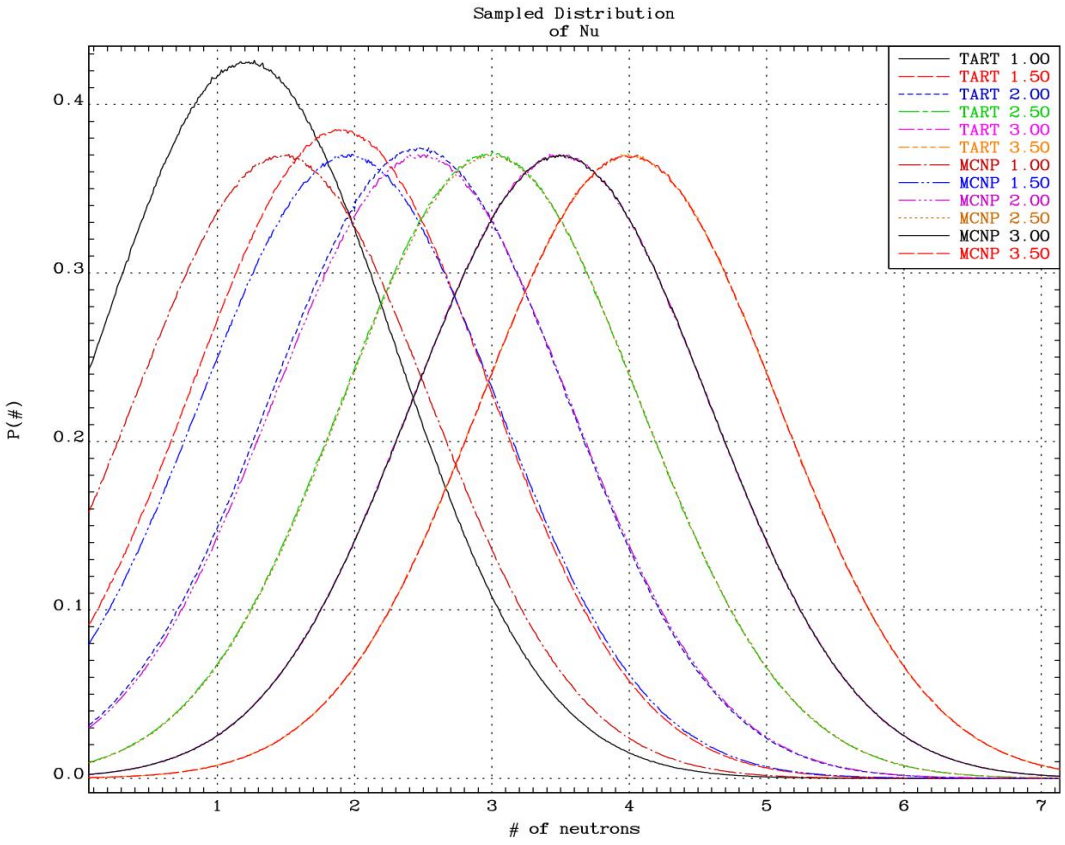
Reject or Return $\nu = 0$

First I will modify the MCNPX sampling routine to add 1/2 rather than a random number when sampling. When I do this the only difference between the two sampling methods is how each code handles a sample of $\nu < 0$: TART rejects and re-samples, MCNPX returns $\nu = 0$.

The plot below compares TART and MCNPX results for $\nu = 1$ through 3.5; I exclude the range near $\nu = 0$ from this plot. The points to note from this plot include,

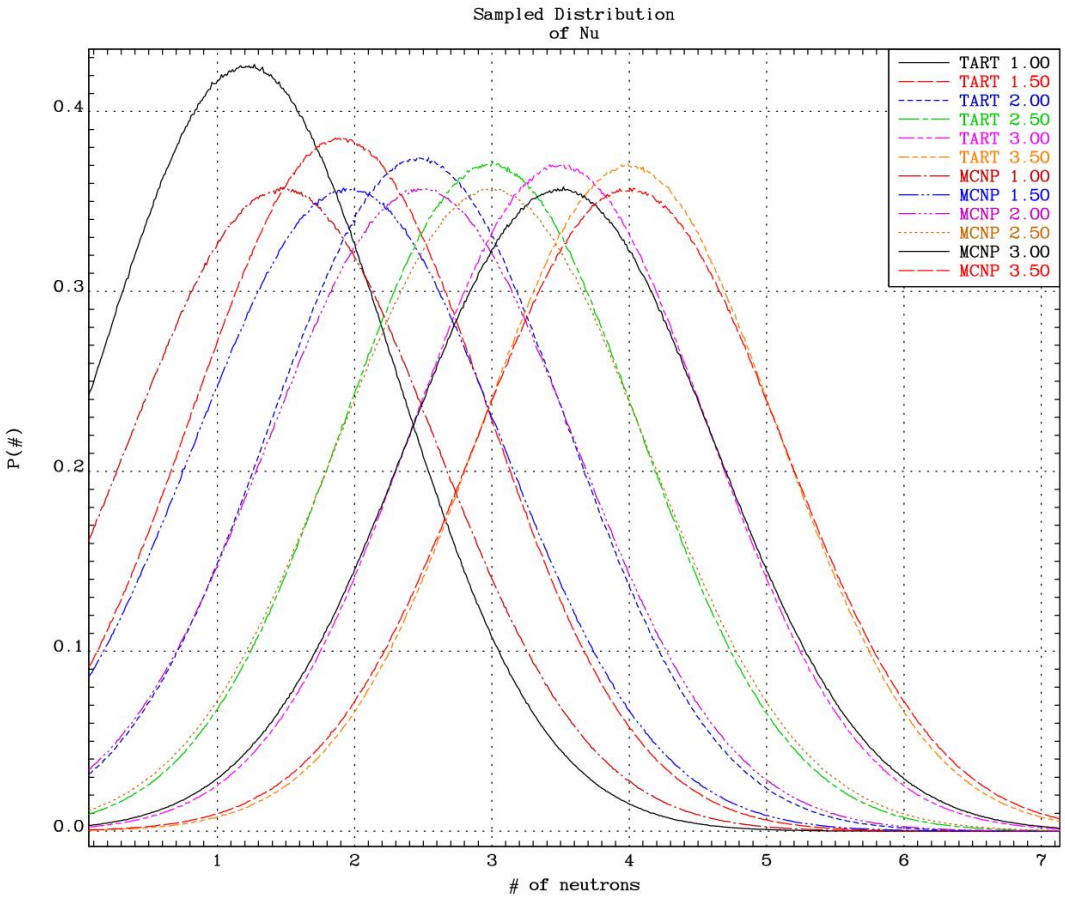
1) All of the MCNPX distributions are alike. This is because it always samples the same distribution, and if the sample $\nu < 0$ it returns $\nu = 0$. In contrast we can see that the TART distribution increases as ν decreases; this is because if the sample $\nu < 0$ it rejects the re-samples; the re-sampling increases the height of the sampled distribution.

2) For small ν this effect is obvious and important. By roughly $\nu = 3$ and higher, this effect is quite small and the MCNPX and TART sampled distributions converge together. Remember this only includes the effect of rejection vs. return 0, and not the effect of adding 1/2 vs. a random number, as such these are artificial results that do not show the true distribution generated by MCNPX.



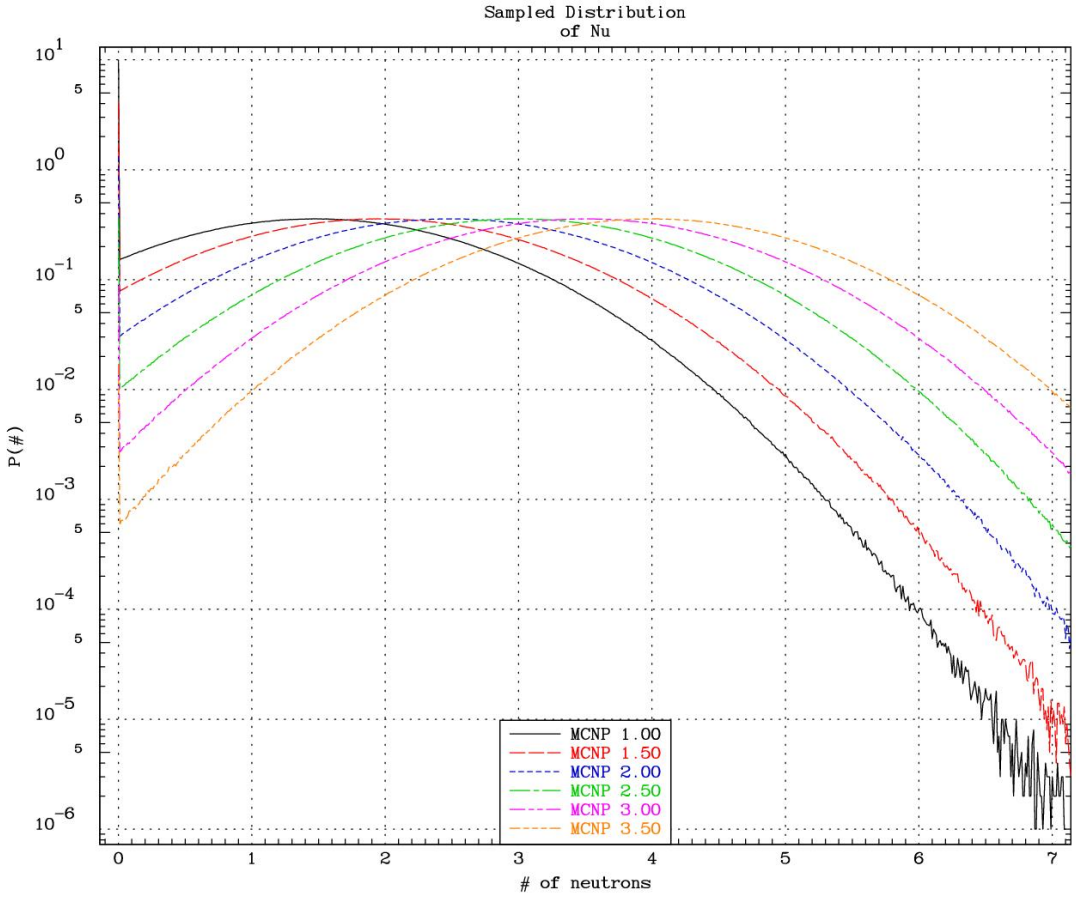
3) To see the actual distribution generated by MCNPX we next use the original MCNPX method to add a random number rather than 1/2. As shown above, the MCNPX distributions are somewhat wider than the TART distribution (the effect of using 1/2 vs. random). On the below figure note that all of the MCNPX distributions are the same shape; this is because any sampled $\nu < 0$ is returned as 0, so that all of positive sampled distributions are identical, regardless of ν. In contrast we can see that the TART distribution increases as ν decreases; this is because if the sample $\nu < 0$ it rejects the re-samples; the re-sampling increases the height of the sampled distribution.

The important point to note here is that even for large values of ν the distributions do not converge together. This result for high ν is solely due to the difference between adding 1/2 (TART) or a random number (MCNPX) to the sampled ν values, and as we have shown above in order to reproduce Terrell's model we must add 1/2, not a random number.



When we include $\nu = 0$ and plot only the MCNPX distributions we can clearly see the effect of returning $\nu = 0$, rather than rejecting. In this case the entire area under the curve corresponding to $\nu < 0$ is replaced by a spike exactly at $\nu = 0$; needless to say for smaller values of ν this area is quite large. For example, try to visualize the extension of Gaussian below $\nu = 0$ where they are cutoff, and you will have a feel for just how large this area is.

Note that in the worst case shown on the figure below ($\nu = 1$) the spike is roughly 30 times higher than the peak of the Gaussian-like remainder of the distribution, i.e., it is not at all insignificant. Note that this effect (the spike at $\nu = 0$) persists to fairly high values of ν, e.g., we can still see it even for the highest value of $\nu = 3.5$ shown on this figure.



As a reminder, unlike Terrell's model to sample a truncated Gaussian, MCNPX is actually sampling a "Gaussian-like" shape (a difference of ERF functions), and adds a spike at $\nu = 0$, by returning $\nu = 0$, whenever $\nu < 0$, is sampled.

How much do these Differences Affect the Integer Distributions?

Since we only use the integer distributions, p(ν), $\nu = 0, 1, 2, 3\ldots$the real bottom line is how much do these differences affect the integer distributions? Below we compare TART and MCNPX results, to illustrate the magnitude of the differences, due to MCNPX not

using exactly Terrell's model. In addition Zucker and Holden [2] provide integer distributions for a number of isotopes, that we can compare to TART and MCNPX results.

To illustrate results using rejection (TART) versus returning $\nu = 0$ (MCNPX), the first results below are for $<\nu> = 1.7$, where the effect of rejection versus return $\nu = 0$ is the largest. Two MCNPX results are included, the first the standard MCNPX method returning $\nu = 0$, and the second using rejection; since this is the only difference between these two we can clearly see the magnitude of the effect of rejection versus returning $\nu = 0$. Comparing the two MCNPX results we see over a 10% difference in P(0). The MCNPX+reject P(0) is still larger than the TART result because of the MCNPX effectively wider width.

ν	TART	MCNPX	MCNPX+reject
0	0.12601	0.14744	0.13348
1	0.31145	0.29164	0.30798
2	0.34629	0.33148	0.33664
3	0.17330	0.17848	0.17429
4	0.03889	0.04529	0.04251
5	0.00389	0.00536	0.00484
6	0.00017	0.00029	0.00025

The table below compares MCNPX and TART results to Zucker and Holden's results for (U235 + n) using $\nu = 2.414$ (E0 = 0).

ν	Zucker&Holden	TART	MCNPX	MCNPX+reject
0	0.0317223	0.03567	0.04384	0.04024
1	0.1717071	0.16341	0.16432	0.16716
2	0.3361991	0.33561	0.32422	0.32678
3	0.3039695	0.31022	0.30294	0.30284
4	0.1269459	0.12900	0.13399	0.13283
5	0.0266793	0.02402	0.02788	0.02741
6	0.0026322	0.00199	0.00269	0.00263
7	0.0001449	0.00007	0.00012	0.00011

The table below compares MCNPX and TART results to Zucker and Holden's results for (Pu239 + n) using $\nu = 2.876$ (E0 = 0).

ν	Zucker&Holden	TART	MCNPX	MCNPX+reject
0	0.0108601	0.01313	0.01679	0.01567
1	0.0993044	0.08796	0.09269	0.09321
2	0.2748737	0.26343	0.25934	0.26018
3	0.3270500	0.35463	0.34307	0.34340
4	0.2047660	0.21488	0.21508	0.21481
5	0.0727720	0.05847	0.06363	0.06341
6	0.0097430	0.00710	0.00881	0.00876

7 0.0006310 0.00038 0.00056 0.00055

Compared to the Zucker&Holden results it is difficult to say that one set of our model results are better than another. For smaller values of ν the TART results seem better, and for larger values of ν the MCNPX results seem better, but none of our results are best over the entire range of ν. Even if I claim one set is "better" than another there are still rather large differences between the Zucker&Holden results and our model results.

Conclusions

I defined in detail the model used to **sample the number of prompt neutrons emitted in fission**; this description is based on publications defining the model [1] as well as publications comparing the model to experimental measurements [2]. The model described in these publications is exactly what the TART [3] Monte Carlo transport code uses.

Based on comparisons between TART [3] and MCNPX [4], it is obvious that at the time this report was published these two computer codes are not using the same model, and the results significantly differ. It is my hope that this report will contribute toward better understanding of this model, and hopefully eventually to agreement between TART and MCNPX results.

Partial success has already been achieved in the sense that based upon reading a preliminary version of this report, John Hendrichs [5], one author of MCNPX, acknowledged that the sources of differences as described in this report demonstrate an error in MCNPX (John even offered me the traditional $20 reward for reporting an error in MCNPX; I declined to accept). John is presently updating MCNPX to eliminate these sources of differences; hopefully in the not too distant future this correction will be available in MCNPX, and we will obtain agreement between TART and MCNPX, which is the ultimate objective of this report.

References

[1] "Distributions of Fission Neutron Numbers", Phys. Rev. 108, 783 (1957), by John Terrell

[2] "Energy Dependence of the Neutron Multiplicity Pν in Fast Neutron Induced Fission of 235, 238 U and 239 Pu", BNL-38491, Brookhaven National Laboratory (1986), by M.S. Zucker an N.E. Holden

[3] **TART05:** A Coupled Neutron-Photon 3-D, Combinatorial Geometry Time Dependent Monte Carlo Transport Code, UCRL-SM-218009, Lawrence Livermore National Laboratory, by Dermott E. Cullen, November 2005, available online at **http://www.llnl.gov/cullen1/mc.htm**

[4] **MCNPX:** MCNP: A General Monte Carlo N-Particle Transport Code, Version 5, Volume I: Overview and Theory, X-5 Monte Carlo Team, Los Alamos National Laboratory report LA-UR-03-1987, April 24, 2003. Portions of the MCNP manual on-line at http://www xdiv.lanl.gov/x5/MCNP/themanual.html

[5] Private Communication, John Hendrichs (April 2006).

TART's Sampling Routine

For inclusion here some of the comments from this routine have been deleted in order to present the entire routine on a single page; for the complete routine see the TART source code.

```
      FUNCTION RANDNU(FNUTAB)
C=====================================================================
C     PURPOSE
C     =======
C     RANDOMLY SAMPLE NU ASSUMING A TRUNCATED GAUSSIAN DISTRIBUTION,
C     A*EXP(-Z^2)
C     Z = {NU - (<NU>+1/2+C)]/WIDTH
C     INVERTING, NU = WIDTH*Z + <NU> + 1/2 + C
C     WHERE THE SAMPLED NU MUST BE 0 TO INFINITY.
C
C     PARAMETERS
C     ==========
C     FNUBAR   = NU-BAR          (INPUT)
C     RANDNU   = GAUSSIAN SAMPLE (OUTPUT)
C=====================================================================
      IMPLICIT REAL*8 (A-H,O-Z)
      SAVE
C-----COUNTER FOR PASSES - FIRST TIME DEFINE 2*PI
      DATA IPASS/-1/
C-----WIDTH OF GAUSSIAN
c-----sqrt(2)*1.08=1.52735, where 1.08 is the recommended value in many
c-----publications, due to sampling EXP(-z^2/2), rather than EXP(-z^2).
      DATA WIDTH/1.52735D+00/
C-----OFFSET FOR FLOOR OF SAMPLE
      DATA HALF/0.5000D+00/
C-----TRUNCATION PARAMETER DEFINITION
      DATA BSHIFT/-0.4675000D+00/
C-----DEFINE SHIFT ONLY ONCE (AVOID MORE WITH REJECTION)
      TEMP1  = FNUTAB + HALF
      EXPO   = DEXP(-(TEMP1/WIDTH)**2)
      ASHIFT = TEMP1 + BSHIFT*EXPO/(1.0D+00 - EXPO)
C-----SELECT PATH
      IF(IPASS) 10,20,30
C-----DEFINE 2*PI ONCE
   10 TWOPI = 2.0D+0*DACOS(-1.0D+0)
C
C     EVERY EVEN PASS GET 2 INDEPENDENT SAMPLES
C
   20 IPASS     = 1
      RW        = WIDTH*DSQRT(-DLOG(RANF()))
      THETA     = TWOPI*RANF()
      SAMPLEG   = RW*DCOS(THETA) + ASHIFT
C-----ONLY ACCEPT NU=0 TO INFINITY - OTHERWISE REJECT, TRY OTHER SAMPLE.
      IF(SAMPLEG.LT.0.0D+00) GO TO 30
      RANDNU  = SAMPLEG
      RETURN
C
C     EVERY ODD PASS USE SECOND INDEPENDENT SAMPLE
C
   30 IPASS     = 0
      SAMPLEG   = RW*DSIN(THETA) + ASHIFT
C-----ONLY ACCEPT NU=0 TO INFINITY - OTHERWISE REJECT, TRY OTHER SAMPLE.
      IF(SAMPLEG.LT.0.0D+00) GO TO 20
      RANDNU  = SAMPLEG
      RETURN
      END
```

MCNPX's Sampling Routine

For comparison I include here a slightly modified version of the routine used by MCNPX; it is modified only to make it compatible with the above TART routine to allow comparable testing. The only significant modification is that the MCNPX routine only returns the integer (floor) sampled value, whereas here I return the floating point sampled value, so that we can examine in detail the continuous and integer probability distributions sampled, as shown in the above figures. The only significant difference between what TART and MCNPX do is the 2 lines near the end of the routine.

```
      FUNCTION RANDNU(an)  ! an = <nu>
      IMPLICIT REAL*8 (A-H,O-Z)
c=======================================================================
c
c     Sample Gaussian = Exp[-r^2/2)
c     r = [nu - (<nu>+g)]/width
c     inverting
c     nu = width*r + (<nu>+g)
C
c=======================================================================
      DIMENSION C1(31)
c-----width
      DATA WD/1.0800d+0/
      DATA C1/
     1  0.100528d0, 8.02249d-2, 6.38562d-2, 5.06562d-2, 4.00222d-2,
     2  3.14746d-2, 2.46252d-2, 1.91590d-2, 1.48167d-2, 1.13856d-2,
     3  8.69035d-3, 6.58690d-3, 4.95634d-3, 3.70147d-3, 2.74301d-3,
     4  2.01670d-3, 1.47075d-3, 1.06379d-3, 7.63034d-4, 5.42685d-4,
     5  3.82671d-4, 2.67508d-4, 1.85374d-4, 1.27331d-4, 8.66889d-5,

     6  5.84945d-5, 3.91172d-5, 2.59241d-5, 1.70256d-5, 1.10803d-5,
     7  7.14548d-6/
      DATA ONE /1.0d0/
      DATA TWO /2.0d0/
      DATA TEN /10.d0/
c-----sample random Gaussian = exp[-r^2/2]
   10 x1=two*RANF()-one
      x2=x1*x1+RANF()**2
      if (x2.gt.one) go to 10
      fw=dsqrt(-two*dlog(x2)/x2)
      dl=WD*x1*fw
c-----Here    tn = WD*r + an , an = <nu> average value
      tn=an+dl
c-----decrease all values of nu (tn) by c1 values.
      x=an/WD
      j=int(ten*x)-9
      if(j.le.30) then
      c=c1(j)+(c1(j+1)-c1(j))*(ten*x-j-9)
      tn=tn-c*WD
      endif
c-----Here tn = WD*r + an - c*WD
c***** DEBUG
c-----MCNPX method
      tn = tn+RANF()
      if(tn.lt.0.0) tn = 0.0
c***** DEBUG
c-----TART method
c     tn = tn+ 0.5d+00
c     if(tn.lt.0.0) go to 10
c***** DEBUG
      RANDNU = tn               ! = WD*r + an – c*WD
      RETURN
      END
```