Implementation of Functional Expansion Tallies in the Monte Carlo Code SHIFT

Ryan H. Stewart, Leslie Kerby

Idaho State University, Department of Nuclear Engineering and Health Physics, Pocatello, ID 83209, stewryan@isu.edu

1 Abstract

Exascale computing allows for the creation of full scale nuclear reactor simulations. It is the hope of these simulations to provide in depth research for reactor safety and design via coupled neutronics and computational fluid dynamics. However, combining computational fluid dynamics and neutronics is not inherently easy. Typically, neutroncis codes produce results, such as neutron flux, energy deposition, etc, in terms of unit cells, where fluid dynamics codes are produced in terms of meshes for finite element analysis. Given the two types of results, it is not inherently easy to couple and transfer data between them. It was decided that a new methodology of representing results was necessary for ease and accuracy of transferring data. The first step in producing transferable data sets was implementing a new method into the neutronics code. The neutronics code Shift was chosen to perform reactor physics simulations, due to it's recent build for large scale computing. Shift currently utilizes cells for implementing collision tallies and a surface mesh for surface current tallies. To provide an easily transferable tally the Functional Expansion (FE) method was chosen to represent units such as the neutron flux and surface fluence.
A method to determine the applicability of Functional Expansion Tallies (FETs) in Shift was explored. Viability focused only on discrete estimators, in the form of surface current and cell collision tallies (resulting in a cell flux). The implementing basis set used were Legendre polynomials. An algorithm was created to solve for the FET coefficients and was verified to calculate basic functions correctly. The FET algorithm was unable to be implemented into Shift, but was able to extract data from multiple Shift simulations and produce comparable results.

2 Introduction

Shift is a new Monte Carlo (MC) transport code developed by Oak Ridge National Laboratory for large scale reactor analysis. It is optimized to perform MC transport calculations on current and near-future computing architectures, while retaining the ability to run on small clusters and personal laptops. The design of Shift allows for modular and easily extensible implementation of features such as physics and source implementations, hybrid capabilities with deterministic codes, and parallel decomposition [5]. Implementation of these features allows for full utilization with large computing clusters to examine whole core

reactor behavior with the hope of coupling Shift with computational fluid dynamics codes, such as Nek5000. Coupling neutronics and computational fluid dynamics codes allows the user to simulate feedback mechanism such as fuel behavior and temperature in near real time. Shift utilizes traditional MC bin tallying techniques to determine collision rates or flux values within the core. Although this method provides good results, it can lead to excessive particle requirements and run times depending on the amount of detail required.
Traditional MC tallies utilize two different classes of estimators, discrete and track length, to extract useful information during the MC process. The information is then separated into bins to create a histogram based solution for the underlying physically meaningful result such as flux, or reaction rate. Each bin requires many particles to provide meaningful results, and if a small bin is used a large variance may occur due to too few particles being tallied. To obtain reliable results the tally size would have to be increased, which degrades the quality fo the answer. Another approach is to increase the number of particle histories which would increase the run time. To avoid quality degradation and increased run times, a new methodology was needed.
An attempt to address these issues using Functional Expansion Tallies (FETs) was implemented by Griesheimer in the code MCNPX [1][2]. FETs use a set of basis functions, which form a complete set, to estimate the shape of the flux (or current) as a series expansion. This provides a continuous representation of the flux compared to the bin method of traditional MC tallies. FETs also have the ability to extract higher order information about the tally shape from particle histories, while retaining the lower order information, such as integral quantities. Along with this, FETs are not dependent on mesh or finite element calculations which provides a method to transfer data between multiple codes with differing geometric definitions.

3 Background

3.1 Surface Current FET

FETs rely on a series expansion to approximate the shape of an unknown distribution. The series used for expansion can vary depending on the boundary conditions, geometry, and prior information of the distribution. Often times, there is little knowledge of a priori information and it is best to select a basis set known to have fast convergence rates for nearly smooth functions. Along with this, for ease of implementation, it is often best to utilize a basis function with an orthogonal basis, such as Chebyshev or Legendre polynomials. For the work in Shift, Legendre Polynomials were selected for implementation.
A brief description of on how FETs can be constructed to describe physical quantities within a reactor is described below. For this description, a one dimensional derivation is shown below for ease of understanding. For an in depth derivation, readers should look at towards the many papers and dissertation by Griesheimer [1][3]. The first step in implementing FETs into Shift was the

implementation of surface current. The surface current in the x-direction is relatively easy to implement and can be expressed as a series expansion, $J(x)$. This can be generalized to multiple dimensions, collision types, energy levels, etc. The finite expansion of $J(x)$, using a basis function $\psi(x)$, can be described by

$$J(x) \approx \sum_{m=0}^{M} \bar{a}_m \psi_m(x) k_m \rho(x) \tag{1}$$

Where $\rho(x)$ is the associated weighing function for the basis function used, and k_m is the orthonormalization constant described by

$$k_m = \frac{1}{||\psi_m(x)||^2} \tag{2}$$

The individual coefficients to be described as

$$\bar{a}_m = \int |\vec{j}(x) * \vec{s}| \psi_m(x) \rho(x) dx \tag{3}$$

In this sense, $\bar{a}_m$ is the m^{th} coefficient of the m^{th} basis function, $\psi_m(x)$ (in this case, the Legendre polynomial, $P_m(x)$). In the case of Legendre polynomials, $\rho(x) = 1$. The ease of using a constant associated weighing function provides an inherent benefit over other basis functions which can have singularities or other complex functions associated with their weighing function. Monte Carlo does not utilize an infinite number of particles, and thus an estimate of $\bar{a}_m$ is required, namely

$$\hat{a}_m = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} w_{n,k} \psi_m(x_{n,k}) \tag{4}$$

Where in the context of a surface estimator; w_n is the weight of the n^{th} particle, and $\psi_m(x_n)$ is the contribution of the n^{th} particle to the m^{th} order Legendre polynomial. N is the total number of particles that contributed to the coefficient, and k is the number of times the n^{th} particle crosses the surface. For the derivation above, only one surface is described, if multiple surfaces were needed, then a coefficient vector would be required for each surface.

Expanding the current to two dimensions is not trivial and requires some manipulation of $\hat{a}_m$. A detailed rework is not present however the results are given with a brief description [3]. To incorporate a second dimension, a second basis function, call it $\psi_i(y_{n,k})$, is introduced. This requires a slight rework of equation (4), where $\hat{a}_m$ becomes $\hat{a}_{mi}$ to incorporate the second dimension. This means $\hat{a}_{mi}$ is a M x I matrix holding all of the coefficients. Where equation (4) becomes

$$\hat{a}_{mi} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} w_{n,k} \psi_m(x_{n,k}) \psi_i(y_{n,k}) \tag{5}$$

3.2 Collision FET

The second step in implementing FETs into Shift was the development of cell (collision) tallies. Cell tallies are found in a similar way to surface tallies. In a MC simulation, if a particle undergoes a collision within the cell of interest it is sampled for the FET. This only requires a slight rework of section (3.1). First, the flux is used in place of the current, thus a finite expansion of the flux $\phi(x)$ can be written as

$$\phi(x) \approx \sum_{m=0}^{M} b_m \psi_m(x) k_m \rho(x) \tag{6}$$

Where again, the orthogonality constant can be described by (2), and b_m is the expansion coefficient. The expansion coefficient can be described by

$$\bar{b}_m = \int \phi(x)\psi_m(x)\rho(x)dx \tag{7}$$

The expansion coefficient is described in a slightly different manor, due to the underlying physics involved with a collision tally. Following the same logic from the surface tally, when a Monte Carlo method is used, an estimate of the coefficient yields

$$\hat{b}_m = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} \frac{w_{n,k}\psi_m(x_{n,k})}{\Sigma_t(x_{n,k})} \tag{8}$$

Again, when using the Legendre polynomials $\rho(x)$ is simply 1. The difference between equation (4) and equation (8) is the denominator. In the denominator for the cell tally is the macroscopic transport cross-section. This cross-section determines the probability of a collision within the cell, which acts essentially as a secondary weighing function. Where similarly, k is the frequency of the n^{th} particle colliding in the cell of interest.
To incorporate all three dimension for equation (8), a third basis function is required, call it $\psi_j(z_{n,k})$. Again, a slight rework of $\hat{b}_m$ becomes $\hat{b}_{mij}$, thus for a three dimensional FET, a three dimensional matrix M x I x J is required. Similar to the surface current, equation (8) becomes

$$\hat{b}_{mij} = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} \frac{w_{n,k}\psi_m(x_{n,k})\psi_i(y_{n,k})\psi_j(z_{n,k})}{\Sigma_t(x_{n,k}, y_{n,k}, z_{n,k})} \tag{9}$$

3.3 Monte Carlo Uncertainty in the FET

To effectively sample a parameter, the Monte Carlo technique incorporates a stochastic component, which causes an intrinsic statistical uncertainty associated with it. Functional expansion tallies are no exception to this and each coefficient will have a statistical uncertainty associated with it. Along with this, the final expression of the current will have an uncertainty. The variance for each coefficient, $\hat{a_m}$ is found using a derivative of the sum of squares law and is found to be

$$\hat{\sigma}^2_{\hat{a}_m} = \frac{\sum_{n=1}^{N}[\sum_{k=0}^{K} w_{n,k}\psi_m(x_{n,k})]^2 - \frac{1}{N}[\sum_{n=1}^{N}\sum_{k=0}^{K} w_{n,k}\psi_m(x_{n,k})]^2}{N(N-1)} \tag{10}$$

This uncertainty is only an estimator of the real uncertainty, where the real uncertainty would requires a prior knowledge of the probability density function. This knowledge is rarely known, and thus a statistical estimator provides a reasonable estimate of the uncertainty.
With the coefficients of the basis function found, along with their associated uncertainties, the Monte Carlo approximation of $J(x)$ can be written as

$$\hat{J}(x) = \sum_{m=0}^{M} \hat{a}_m k_m \psi_m(x) \tag{11}$$

As with the coefficients, Monte Carlo approximations are meaningless without the corresponding variance, which is given by the two-norm variances

$$\sigma^2_{\hat{J}(x)} = \sum_{m=0}^{M} \sigma^2_{\hat{a}_m} k_m \tag{12}$$

A similar description can be given for collision tallies. However, only the results are shown below for the variance in the coefficients, equation (13), the approximation of $\hat{\phi}(x)$, equation (14), and the total uncertainty, equation (15).

$$\begin{aligned}\sigma^2_{\hat{b}_m} &= \frac{1}{N(N-1)}\sum_{n=1}^{N}[\sum_{k=0}^{K}\frac{w_{n,k}}{\Sigma_t(x_{n,k})}\psi_m(x_{n,k})]^2 \\ &- \frac{1}{N-1}[\frac{1}{N}\sum_{n=1}^{N}\sum_{k=0}^{K}\frac{w_{n,k}}{\Sigma_t(x_{n,k})}\psi_m(x_{n,k})]^2\end{aligned} \tag{13}$$

$$\hat{\phi}(x) = \sum_{m=0}^{M} \hat{b}_m k_m \psi_m(x) \tag{14}$$

$$\sigma^2_{\hat{\phi}(x)} = \sum_{m=0}^{M} \sigma^2_{\hat{b}_m} k_m \tag{15}$$

3.4 Truncation Error & Optimization in FETs

The current in equation (1) is only exact if an infinite number of coefficients are calculated. An infinite number of coefficients is not possible when utilizing a Monte Carlo system, thus the coefficients must be truncated at some value, which introduces a truncation error. The truncation error, $E_m(x)$, introduced is equal to the value of all the expansion terms that are greater than truncated coefficient M.

$$E_m(x) = |J(x) - J_m(x)| = |\sum_{n=M+1}^{\infty} \hat{a}_m k_m \psi_m(x)| \tag{16}$$

For each coefficient, $\hat{a}_m$, added, the truncation error is decreased by a factor of $\hat{a}_m^2$. However, each time a new coefficient is added the statistical error is increased due to the new coefficients error being added. The increase in statistical error is given by $\hat{\sigma}^2_{\hat{a}_m} k_m$, or the variance associated with the $\hat{a}_m$ term. It is seen that the two terms are inversely related, and thus an optimization can be found to relate the two quantities and determine reliability. This optimization can be found in the ratio between the statistical error and the truncation error, given as

$$R_m^2 = \frac{\hat{\sigma}^2_{\hat{a}_m} k_m}{\hat{a}_m^2} \tag{17}$$

Equation (17) gives a relative cost-to-benefit ratio to including the n^{th} term of the series. This ratio can be used as a test to determine if the n^{th} expansion coefficient has converged on the true solution and should be included. Where an $R_n^2 >> 1$ indicates that the coefficient has not converged on the true solution and will yield poor results for the function approximation. $R_n^2 << 1$ indicates that the coefficient has converged on the solution and will yield good results for the function. $R_n^2 \approx 1$ indicates that the coefficient should be examined carefully before including the term for the function approximation. In this range, it is possible to run more particles to allow the coefficient to converge and be included in the approximation.

4 Algorithm

4.1 One Dimension

The algorithm for solving the FET's was written in C++ for ease of integration into SHIFT. To solve for the FETs, data had to be obtained from in SHIFT and transfered to the FET algorithm. The FET algorithm requires the weight, particle position, and the total macroscopic cross-section. With this information, an approximation can be made for current or flux shape with an FET.
The FET algorithm was simple in nature to allow for the most efficient method in calculating tallies. The first step was to obtain the data from SHIFT that would be required. This involved activating the algorithm each time a tally was interacted with. For example, if a surface current was desired, each time a particle passed a surface to be tallied the FET algorithm was required. When a tally interaction occurred, the particle weight, particle position and total macroscopic cross-section was transfered to the FET algorithm. For the position, each direction is bounded by a maxima and minima which is used to transfer the particles position from the reactor space to Legendre space, which has boundaries

at [-1,1]. This is done using the transform equation in (18). Where $\tilde{x}$ is the Legendre space transform of position x in reactor space.

$$\tilde{x} = 2\frac{x - x_{min}}{x_{max} - x_{min}} - 1 \tag{18}$$

The transformed $\tilde{x}$ is then plugged into the Legendre polynomial, $P_m(x)$ and solved. Each coefficient for the Legendre polynomial is solved for and a vector containing the coefficients is produced. This process can be repeated multiple times depending on the number of tally interactions occurring for a single particle. Each time a particle interaction occurs, the Legendre coefficients are summed to create a_n. Once a particle is killed, the a_n's are summed to create an estimate of $\hat{a}_m$, designated as A_n. In addition to this, for each a_n, a separate sum of a_n^2 is kept, and designated as A_m. The value of A_m is used in post processing for calculating the coefficient variance. Along with this, a running tally for the number of particle interactions for each tally of interest is kept for post processing.
Once the simulation is complete, and all of the histories/particles have been run, post processing is performed to find the values of $\hat{a}_m$, their associate variance, the current, and the overall variance through the surface. For the current in one dimension, to find $\hat{a}_m$, the final A_n is divided by the total number of particles crossing the surface of interest. This normalizes the coefficients according to the particle fluence through the surface.
To find the variance, equation (10) was used where $[\sum_{n=1}^{N}\sum_{k=0}^{K} w_{n,k}\psi_m(x)]^2$ is represented by A_m, and $\frac{1}{N}[\sum_{n=1}^{N}\sum_{k=0}^{K} w_{n,k}\psi_m(x)]^2$ is represented by A_n. Once the variance is found for each coefficient, the current can be written as

$$\tilde{J_m}(\tilde{x}) = \sum_{m=0}^{M} \hat{a}_m k_m P_m(\tilde{x}) \pm \sigma_{\hat{a}_m} \tag{19}$$

It should be noted that equation (19) is still contained in Legendre space, that is [-1,1]. If the user desired to transform the equation to original units, a simple transformation of $\tilde{x}$ is required, as in equation (20).

$$\tilde{x} = 2\frac{x - x_{min}}{x_{max} - x_{min}} - 1 \tag{20}$$

Where $\tilde{x}$ from equation (19) is replaced with equation (20). Along with this, the orthonormalization constant must be transformed to k'_m via equation (21).

$$k'_m = \frac{2m+1}{x_{max} - x_{min}} \tag{21}$$

With the transformations from equations (20) and (21) the current for the original tally domain is described as

$$\hat{J_m}(x) = \sum_{m=0}^{M} \hat{a}_m k'_m P_m(2\frac{x - x_{min}}{x_{max} - x_{min}} - 1) \pm \sigma_{\hat{a}_m} \tag{22}$$

4.2 Surface Tallies

The surface tally algorithm to solve for two dimensions is not trivial. If both an x, and y directions are required, then a coefficient matrix for the current is created. To record all of these, a I x J matrix is created, and labeled as A_{ij}. Where i is the number of coefficients in the x direction, and j is the number of coefficients in the y direction. Thus, the current can be written as

$$\hat{J}(\tilde{x}, \tilde{y}) = \sum_{i=0}^{I} \sum_{j=0}^{J} \hat{a}_{ij} k_{ij} \psi_i(\tilde{x}) \psi_j(\tilde{y}) \tag{23}$$

The uncertainty for the two dimensional surface tally is not yet implemented in the algorithm.

4.3 Cell Tallies

The cell tally algorithm follows the same general guidelines of the surface tally. In addition to needing the particle weight and position, the total-macroscopic cross-section is required. Again, similar to the surface tally, if an x, y, and z direction are required then a I x J x K matrix is required to store the coefficients. Where in this case, three variables are required which yields a matrix labeled as A_{ijk}. Where i is the number of coefficients in the x direction, j is the number of coefficients in the y direction, and k is the number of coefficients in the z direction. Thus, the flux can be written as

$$\hat{\phi}(\tilde{x}, \tilde{y}, \tilde{z}) = \sum_{i=0}^{I} \sum_{j=0}^{J} \sum_{k=0}^{K} \hat{a}_{ijk} k'_{ijk} \psi_i(\tilde{x}) \psi_j(\tilde{y}) \psi_k(\tilde{z}) \tag{24}$$

The uncertainty for the three dimensional flux tally is not yet implemented in the algorithm.

5 Verification & Validation

5.1 Verification of Functionality

The first phase of the verification process was to ensure the algorithm was performing the intended calculations and producing reliable results. To test this, a test function was developed. The test function used rejection sampling to sample the shape of a user defined distribution. For the sake of testing, multiple distributions were selected to determine accurate results, and to learn additional nuances of FET properties; such as time requirements, optimal coefficient selection, particle number, and statistical uncertainties.
For the first test, a distribution of $5x^3 - x^2 - 2x + 4$ was used. This initial test was to determine if the algorithm created a set of Legendre polynomials that matched the distribution function. The results were promising, for nearly any number of particles, as seen in Table 1. For the results in Table 1, seven

coefficients. These coefficients can be seen in Table 2. Along with these, the uncertainty in each coefficient can be seen in Table 3.

Table 1: FET Algorithm Results

Position	Dist.	FE		Diff.		
		1e5	1e6	1e7	1e8	1e9
-1.0	0.00	-2.32e-2	1.91e-3	-1.41e-3	1.63e-3	3.41e-4
-0.6	3.76	-2.94e-2	3.04e-3	3.94e-3	4.68e-4	1.96e-5
-0.2	4.32	-1.74e-2	-5.40e-3	-3.09e-3	-6.71e-4	4.04e-5
0.2	3.60	1.98e-2	-4.01e-3	-1.26e-4	-1.66e-4	5.83e-5
0.6	3.52	1.04e-2	6.93e-3	-1.20e-3	-2.88e-4	-2.15e-4
1.0	6.00	-7.93e-2	-1.11e-2	1.90e-2	5.98e-3	2.254e-4

Table 2: FET Coefficient Values

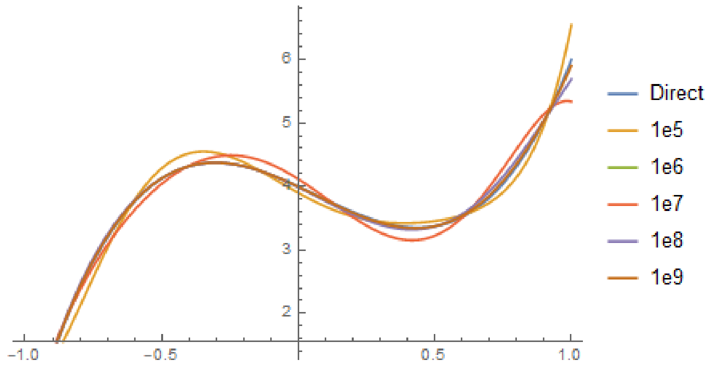
Coefficient			FE		
	1e5	1e6	1e7	1e8	1e9
$P_0(x)$	3.67	3.67	3.67	3.67	3.67
$P_1(x)$	0.99	1.00	1.00	1.00	1.00
$P_2(x)$	-0.66	-0.67	-0.67	-0.67	-0.67
$P_3(x)$	2.05	2.00	2.00	2.00	2.00
$P_4(x)$	3.10e-3	1.13e-2	2.35e-4	-7.11e-4	-2.31e-4
$P_5(x)$	-1.21e-2	1.92e-3	-7.83e-3	-1.36e-3	-1.31e-4
$P_6(x)$	3.74e-2	-4.45e-3	-6.51e-3	-2.44e-3	-2.14e-4

Table 3: FET Coefficient Uncertainty

Coefficient			FE		
	1e5	1e6	1e7	1e8	1e9
$P_0(x)$	3.14e-3	9.97e-4	3.15e-4	1.00e-4	3.15e-5
$P_1(x)$	6.71e-3	2.12e-3	6.70e-4	2.12e-4	6.70e-5
$P_2(x)$	5.46e-3	1.73e-3	5.45e-4	1.72e-4	5.45e-5
$P_3(x)$	4.56e-3	1.44e-3	4.54e-4	1.44e-4	4.55e-5
$P_4(x)$	4.07e-3	1.28e-3	4.06e-4	1.28e-4	4.06e-5
$P_5(x)$	3.69e-3	1.16e-3	3.66e-4	1.16e-4	3.66e-5
$P_6(x)$	3.39e-3	1.07e-3	3.37e-4	1.06e-4	3.37e-5

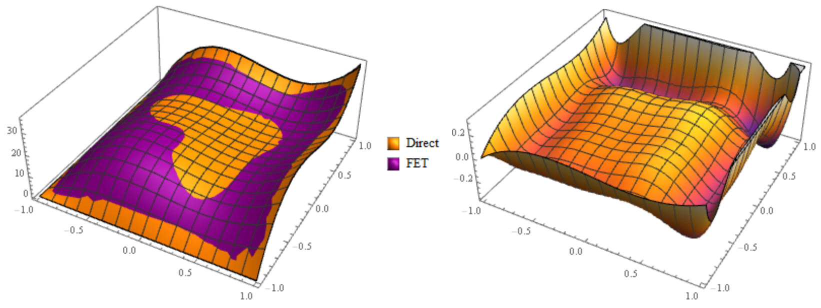
It can be seen from Tables 1 - 3 that increasing the number of particles decrease the difference between the real distribution and the functional expansion distribution. Along with this, an increase in the number of particles decreases the uncertainty for the individual coefficients. This provides confidence that the FET algorithm is performing its intended functions and calculating an accurate result with a test function. A graphical representation of this can be seen in Figure 1.

Figure 1: Comparison for number of particles FE to $5x^3 - x^2 - 2x + 4$



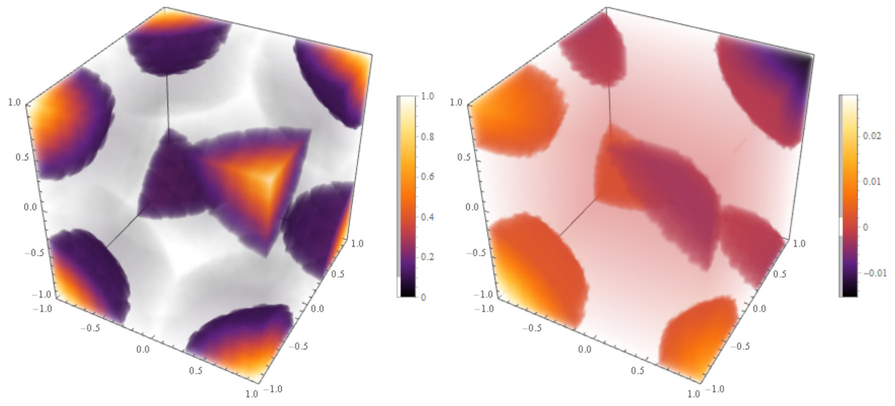
Both the surface, and collision tallies require multiple dimensions to express the current or the flux. To express the surface tallies, the original test function was utilized in both the x and y directions. On the left hand side of Figure 2 is both the direct calculation and the FET approximation fo the test function, where in the z-axis would be equivalent to the current as a function of x and y. Along with this, the right hand side of Figure 2 shows the difference between the FET and the direct calculation of the test function. Due to this test function being cubic in order, the higher order coefficients remain small and unsteady, but congregate around zero. Thus, the higher order terms show a lower importance to the solution, and could be neglected if desired. This is purely a consequence of using a contrived test function, and not indicative of true Monte Carlo simulations. In a simulation regarding a neutron population passing through a surface, each coefficient value would converge towards a true value.

Figure 2: 3D FET & Difference Map



As mentioned, the collision tallies are used to represent the flux in a three-dimensional cell. Figure 3 represents the neutron flux as a density plot throughout a cell for the FET representation and the difference between the FET and a direct calculation. For ease of differentiation, a simple quadratic, $x^2y^2z^2$ is used. Where the left hand side is representative of the FET solution and the right hand side is representative of the direct solution.

Figure 3: Density Plot & Difference Map

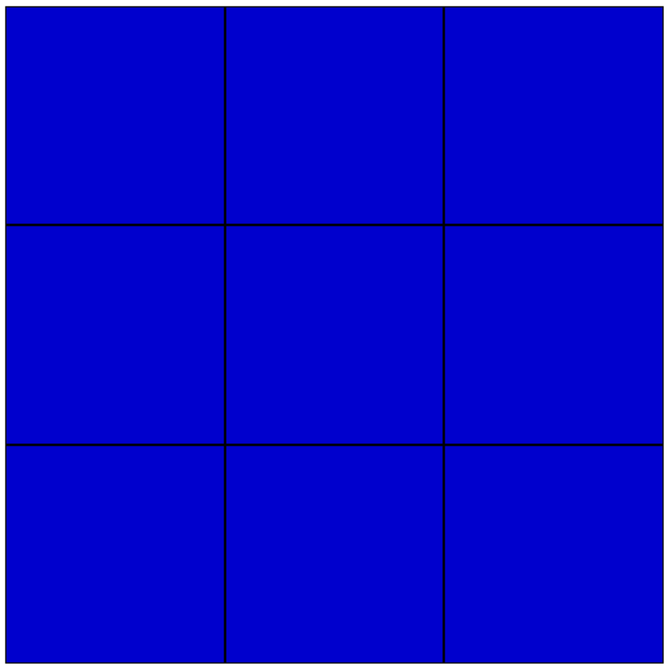


The opacity through out the cell is used for ease of view-ability, where at the center the flux would be zero. As with both the surface, and the two dimensional plots, the largest difference between the direct calculation and the FET, is the edges. This is a consequence of using FETs, which can lead to varying degrees of accuracy between the center and the boundaries of the problem.

5.2 Verification in Shift

The FET algorithm was unable to be implemented into Shift, and thus an integrated comparison between the FETs and histogram based tallies was unable to be made. Despite this, the information required for the FETs was able to be extracted during the process, and could be analyzed to determine if the FET algorithm could analyze the data and produce a corresponding flux value. For this verification, only the cell tally was examined. To determine applicability, a simple simulation was created and tested. This simulation included a 2 *cm* x 2 *cm* x 2 *cm* cube of graphite with a point source at the origin. The neutron energy was set at 0.0025 ev to allow for diffusion to occur within the cell. The cell was split into 27 different cells to create a mesh to compare with the FET, which analyzed the cube as a whole. The mesh had consisted of a 3 x 3 x 3 matrix, with each block measuring 0.666 *cm* x 0.666 *cm* x 0.666 *cm*. This provided a rough mesh to compare with the FET. The mesh lay out can be seen as a cross section, at $z = 0$, in Figure 4. Graphite was chosen to allow for multiple collision to occur with each particle generated before exiting the cell. A high collision rate would allow for good statistics and an ease in determining FET coefficients.

Figure 4: Scale Input Geometry



To utilize the FET algorithm, the particles weight, position, and the macroscopic transport cross-section were pulled from Shift for each collision that occurred. All of this information was then written to a text file. The file was edited to remove any superfluous information, and was then read in by a modified FET algorithm. Due to this, there were limitations on the number of particles that could be used in a simulation. It was found that approximately 10^6 particles was the maximum allowable particle number to allow for manipulation of the output file. Over this number of particles created output files on the order of 0.5 GB, and prevented manual manipulation of the files.
The FETs ability to determine flux shape was determined using three different method. These methods all had the same geometry, that is a 2 cm^3 graphite block, and used 10 coefficients in the FE unless otherwise stated. The first method to measure the FET algorithms ability was to vary the number of particles run for each simulation. This resulted in four simulations with 10^3, 10^4, 10^5, and 10^6 particles. Figure 5 shows the FET approximation for varying particle counts, in the y direction, where the x and z dimensions are held constant at zero.

Figure 5: Particle Count Impact on Flux Shape

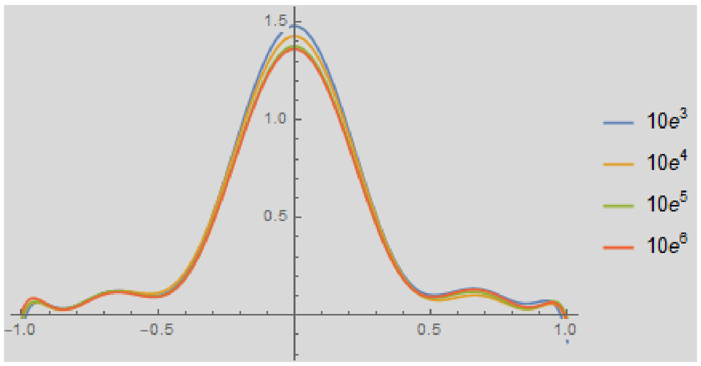


Figure 5 reveals that the shape of the flux only changes slightly with an increased particle count. This matches previous validation, as performed in Section 5.1. The difference between $10e^3$ and $10e^4$ varies 5%, the difference between $10e^5$ and $10e^6$ varies by less than 1%. This indicates that the FE is converging on the true solution. Along with this, the flux shape follows the expected pattern for diffusion within a scattering media. Where along the center is the highest neutron flux rate, and it quickly dies off as the neutrons diffuse towards the boundaries of the media.
The second method used in examining the FET algorithm was to examine the impact on the number of coefficients used. The number of coefficients was altered between 5, 10, and 15. Each simulation was run with 10^5 particles. Figure

6 shows the impact of expansion coefficient truncation.

Figure 6: Coefficient Truncation Impact on Flux Shape

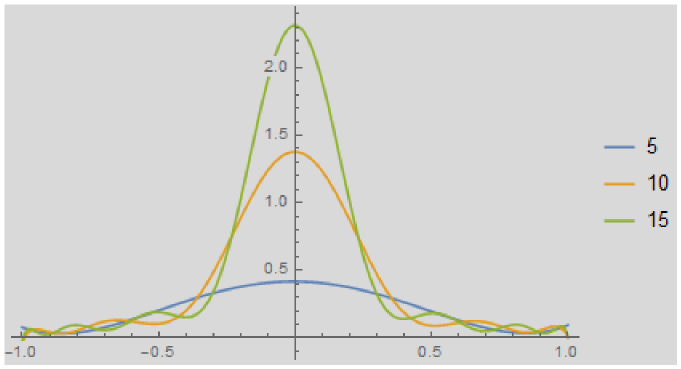


Figure 6 presents an important point; for a FET to provide valid results, the number of coefficients must be able to accurately represent the underlying physics occurring within the simulation. For example, when the number of coefficients is truncated from 15 to 5, the peak neutron flux drops by over 75%. Along with this, a drop of 40% occurs between 15 and 10 coefficients. This being said, it is important to consider the required number of coefficients to provide an accurate response for the problem at present. Where typically, a more conservative approach will yield better results. The results presented here are slightly misleading and the results will not typically vary this much due to number of coefficients. Much of this variation is due to the fact that a point source was used. When considering a point source, the flux near the center approaches infinity which is impossible for any type of expansion to replicate. This reality, can limit FETs ability to accurately describe the neutron flux in areas with drastic changes, and additional methods may be required for validation.
Although the flux peak is higher when using 15 coefficients, only 10 coefficients were used for the methods to determine the FET algorithm functionality. The reason for 10 coefficients was to decreases the time required for model verification, while still allowing general trends to be identified.
It is important to note that the x-direction had large oscillations as the flux moved from the center of the core. Figure 7 shows the oscillations in the x dimension, when y and z are held constant at 0. A similar nature can be seen in Figure 8, which only holds z constant at 0.

Figure 7: 2D Flux Shape with y=0 and z=0

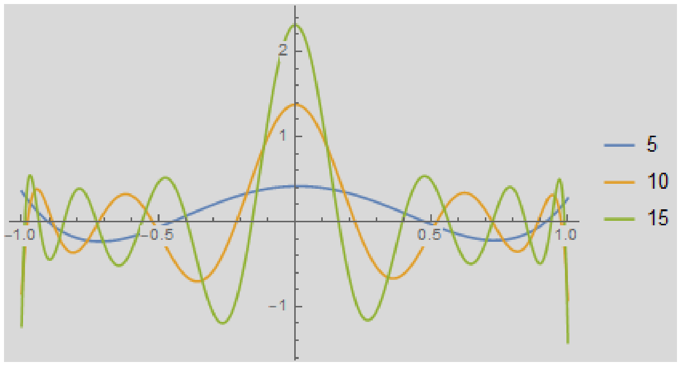
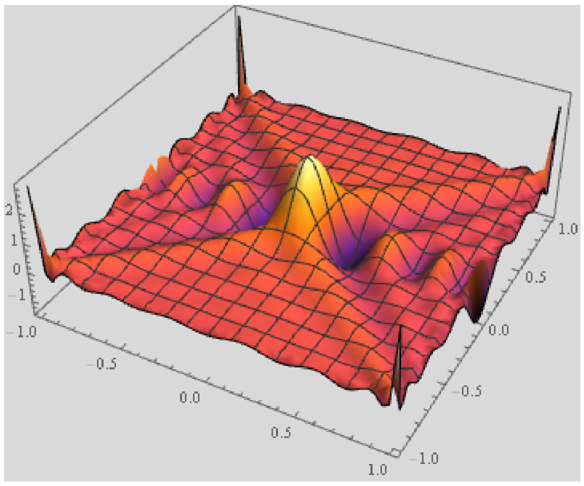


Figure 8: 3D Flux Shape with z=0



However, Figure 9 shows that when the x direction is held constant, there are no oscillatory effects. Another representation of this can be seen in Figure 10. Where the density plot represents the same data as Figure 9. This density plot representation of the data provides an easier to view manor for quantitative information.

Figure 9: 2D Flux Shape with x=0

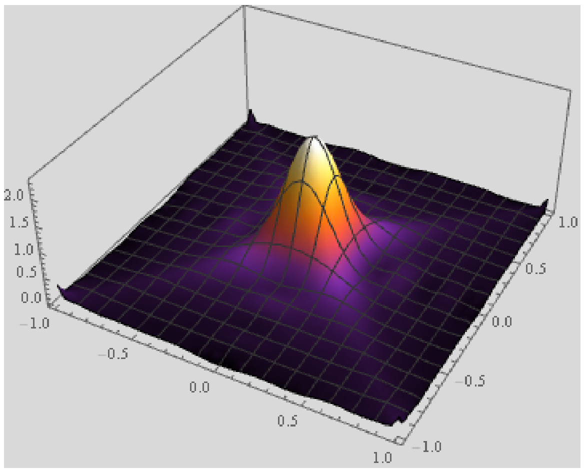
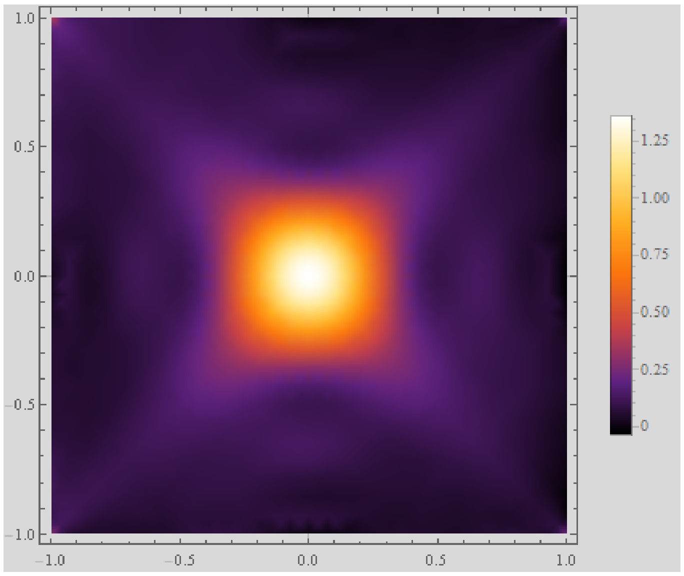


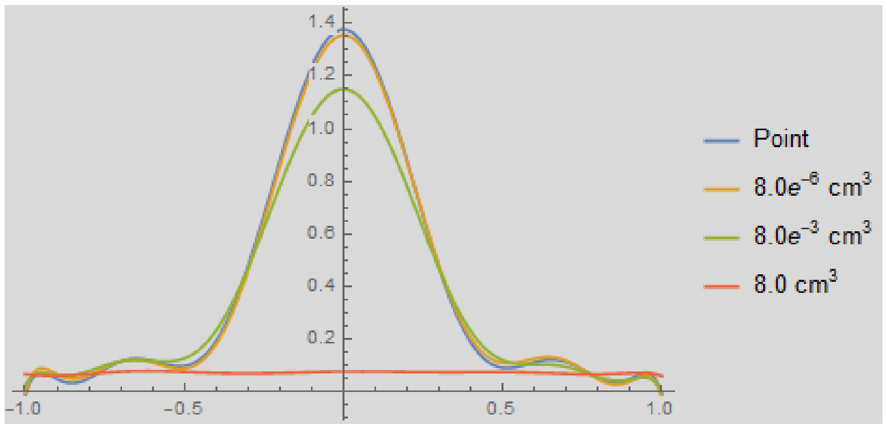
Figure 10: Density Plot with x=0



It is assumed that the y and z dimension coefficients are presenting the solutions correctly, but there is an error within the algorithm when calculating the x-dimension coefficients. This error is currently unknown, but will hopefully be found and corrected soon.

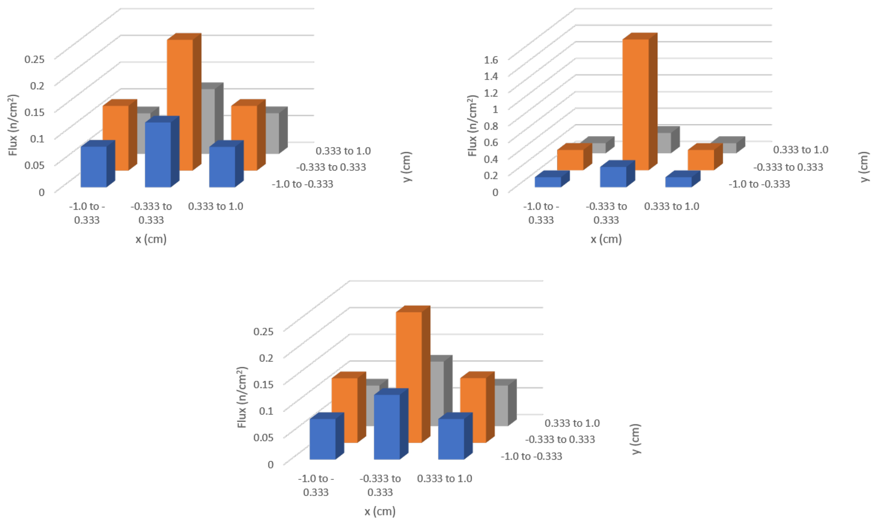
The third method to examine how the FET algorithm functioned was to solve for varying source sizes. To examine this, four difference source sizes were used; a point source, a $8.0e^{-6}$ cm^3 cube, a $8.0e^{-3}$ cm^3 cube, and a 8.0 cm^3 cube (which encompassed the entire graphite cube). Where it is expected, as the source size increases, the magnitude of the peak will decrease. Figure 11 shows the comparison between the multiple source types.

Figure 11: 2D Source Comparison



As noted, the neutron flux at the center cube decreases as the source is spread out further over the cube. As the source gets larger, the flux at the center continually drops, until a near flat source is found when the source size equals the entirety of the problem. Both cases help solidify the functionality of the FET, as each case represents the change in physics occurring during the diffusion process.
Due to the error in the x-dimension, a direct comparison between the mesh tally and the functional expansion tally was not able to be performed. However, some qualitative remarks can be made by comparing the mesh to FET data. The data drawn from Shift was consisted of the 3 x 3 x 3 mesh, to relate these, the x and y dimensions are presented. Each z mesh was given a figure, where the lower mesh ran from -1.0 cm to -0.333 cm, the middle mesh ran from -0.333 cm to 0.333 cm, and the upper mesh ran from 0.333 cm to 1.0 cm. Figure 12 shows the flux for the mesh tallies.

Figure 12: Flux Values for the Lower, Middle and Upper Mesh



From Figure 12, the same trend can be seen of basic diffusion within the cell. In the middle mesh, the center is an order of magnitude greater than the surrounding area, which corresponds to the point source at the center. Along with this, it is noted that the order of magnitude between the mesh tally and the various FETs are congruent, which indicates the FETs are calculating the flux correctly, even with the x-dimension having oscillations. Overall, while the FET is not fully functional, it does provide enough information to warrant further research and implementation into Shift.

6 Optimization

Creating the algorithm to solve for FETs underwent multiple iterations to create a semi-optimized function which performs the calculations efficiently and accurately. The first iteration of the Legendre solver involved using a recursion formula to only Legendre polynomial in question. To solve for this, the following recursion formula was used

$$P_n(x) = \frac{(2*n-1)xP_{n-1} - (n-1)P_{n-2}}{n} \tag{25}$$

Where $P_0(x) = 1$ and $P_1(x) = x$. Using the recursion relation requires the relation to be called $\sum_{n=1}^{N} x$ times to solve for all the coefficients. Most FETs require 20 or fewer coefficients to converge on the solution, and thus a direct calculation approach can save both time and the number of calculations required.

If 20 coefficients are used, then 2310 calculations are required per particle, using the recursion method. To minimize the number of calculations required, the first twelve solutions are solved for explicitly, where the remained are solved via equation (25). Using the explicit solutions for the first twelve coefficients reduces the number of calculations by 21%, down to 1817 calculations. Directly solving for the first twelve coefficients is not initially intuitive, but additional optimization provides a basis for choosing twelve coefficients.
In addition to directly solving the Legendre coefficients for the first twelve coefficients, the Legendre solver function was optimized by vectorizing the coefficients. This process involved solving for each coefficient required during one pass. Initially, the Legendre solver function was called each time a coefficient was required. Thus, if 20 coefficients were required, then the Legendre solver function is called twenty times to obtain each coefficient. This prevented saving any of the data obtained from performing the function previously, which meant that every time the function was called, it had to recalculate all the coefficients, even if they had already been calculated.
The next iteration involved vectorization of the coefficients. Vectorization created a vector of coefficients, which allowed the Legendre solver function to be called once, rather than multiple times. The vectorization process also allowed each coefficient that had previously been calculated to be saved, and used for any higher order functions. For example, if the 20^{th} coefficient was being calculated using (25), then P_{19} and P_{18} had already been calculated and those values could be plugged in and $P_{20}(x)$ could be solved for quickly. This process dramatically decreased the number of calculations required to solve for the coefficients. If the example of 20 coefficients is used again, and no direct calculations are present then the number of calculations required is still 2310. If vectorization is utilized, and the values for solved Legendre functions are saved then only 210 calculations are required, which decreases the number of calculations by 90%. If direct calculations are included, then only 159 calculations are required, which decreases the number of calculations by 93%. This reduction in the number of computations required per particle helps reduce the time to solve for the FETs in general. Returning to the number of direct calculations required, it was found that the number of calculations required for the direct method was the coefficient number plus one. Thus around the tenth coefficient, it took nearly as many calculations to perform a direct calculation as it does to calculate the recursion form. The reduction in calculation time required to solve fo the functional expansion can be seen in Table 4.

Table 4: Vectorization & Direct Calculation Optimization

Method	Time (s)				
Number of Particles	1e5	1e6	1e7	1e8	1e9
Recursion	1.64	16.3	163	1640	16400
Direct + Vectorization	0.213	2.13	21.6	213	2160

From Table 4, when both the direct calculation and vectorization are implemented the calculation time is reduced by 85%. This reduction in time will be extremely valuable when a Monte Carlo problem requires multiple histories, each with millions or billions of particles. If, the original recursion time was required, the FET would potentially be useless in the application of Exascale computing.
Other miscellaneous optimization techniques steamed from the choice data storage. This included using multiple 2D matrix arrays to store different surfaces that required tallies. Multiple 2D matrices were found to be easier to manipulate and required less manipulations to be performed when accessing storage locations, than using one large 3D matrix.

7 Proposed Implementation into SHIFT

Shift provides users with multiple ways to score tallies for a Monte Carlo simulation. Each tally type is accessed through a central class in Shift called Tallier. This class registers each type of tally that is going to be used in the simulation. As a first attempt to implement the functional expansion tally, it was placed inside the tally which counted surface crossings. Along with this, a separate template class was created to allow the FET to be a stand alone tally within Shift, however, due to time constraints this template class was not fully implemented. This implementation was attempted but not achieved for multiple reasons, the largest being a time constraint. However, a reasonable amount of insight was gained into altering the Shift source code which will be presented to help future implementation into Shift.
Shift is split into multiple directories, each related to their function for performing MC neutron transport. For implementing a new tally type into Shift, there are three directories within Shift that are relevant. The first is ‘mc_physics‘, where in the header file ‘SC.Physics.t.hh‘, a multitude of information can be found for particle interactions. This is important for collision tallies due to the fact that the total transport cross-section can be found for each collision that occurs. The next directory is ‘mc_transport‘, where the header file ‘Domain_Transporter.t.hh‘ describes particle movement within the geometry. Finally, the most important directory is ‘mc_tallies‘, which contains the ‘Tallier.t.hh‘ and ‘Tallier.hh‘ files.
It is assumed that the easiest method to implement the FET algorithm is to include all the header and program files within the ‘mc_tallies‘ directory. From there, it would be appropriate to initialize any classes within ‘Tallier.hh‘ for use in ‘Tallier.t.hh‘. Building a distinct template within ‘Tallier.t.hh‘ will allow the user to access the particles information via its class, where the particle weight and position can be accessed via ‘p.wt()‘ and ‘p.pos()‘ respectively. It should be noted that the x,y,z coordinates are given in a single cell, which will require some manipulation to access.

8 Conclusions

The functional expansion tallies have the ability to produce quantities such as the neutron flux and current in codes, such as Shift, as functional expansions rather than the historical bin method. Functional expansions provides new opportunities in the ability to couple multiples codes, often using multiple geometric descriptions, with relative ease. It is for this reason, that functional expansion tallies were attempted to be implemented into Shift. Although FETs were not fully implemented into Shift, they algorithm performing the tallies was able to be used on generated output from Shift. This allowed a baseline evaluation to be performed and determine the functionality and reliability of the FET algorithm.

9 Further Research

There is still much work to be done for implementing FETs into Shift. Much of the work relies on adapting the current code to account for different scenarios. Currently, the only FETs that are allowed utilize Legendre polynomials for Cartesian coordinates. This is useless when using Shift to describe cylindrical or spherical geometries. To aid in future research it would be beneficial to include Associated Legendre polynomials, which are the canonical solutions to the general Legendre equation. Along with this, additional polynomials can and should be included to for stability in using FETs. Multiple types of basis sets broadens their uses within the code. For example, Zernike polynomials could also be used for cylindrical functional expansions, as is being used in Serpent [6][4]. Along with this a plethora of additional basis sets can be included and used to analyze models where specific boundaries are present. For example, Bessel functions or Laguerre polynomials can be used as infinite or semi-infinite basis sets. With multiple basis sets, code could then be written to analyze the geometry, boundary conditions, and any a priori information to choose the most efficient set of FET for the given problem. This capability would allow FETs to be versatile enough to complete with the histogram bin approach that is typical with MC.
Another approach that will need to be considered for implementing FETs into Shift, are the inclusion of track length estimators. Track length estimators are typically superior to collision estimators at tallying volumetric fluxes within a cell. In addition to track length estimators, it has been suggested to use FETs to create the response matrix, which is valuable in linking Monte Carlo and deterministic codes.
For the current implementation, only one cell/surface is allowed to be sampled. In a large reactor simulation, this would be useless, as multiple cells/surfaces will be required for sampling. This will require minor modifications to the code to allow for multiple FETs to be incorporated at once. Currently, the FET algorithm is called from inside the base collision tally. A separate tally template was built in Shift, but not utilized. To allow the user to only use FETs

this will need to be set up, and built into the Shift input file reader to allow for user definition for the number of coefficients to be used. It will also need to be incorporated into the output file and H5 file creator to allow for the user to easily access the coefficients provided. Along with this, additional work can be done with the current code to clean it up and attempt to both generalize it, for use with multiple types of basis functions, and optimize it. Some work has already been done in the optimization section but additional work can be done to decrease the number of calculations or minimize the number of variables used.
Finally, the surface tally was not tested or implemented into Shift. This was due to an error in the Shift code which prevent building surface tallies. It is the hope that when the surface tallies are implemented into Shift, the FET for the surface will be implemented and tested as well.

References

[1] David Griesheimer. Function expansion tallies for monte carlo simulations.

[2] David Griesheimer and William Martin. Estimating the global scalar flux distribution with orthogonal function expansion.

[3] David Griesheimer and William Martin. Two dimensional functional expansion tallies for monte carlo simulations.

[4] Leslie Kerby, Aaron Tumulak, Jaakko Leppanen, and Ville Valtavirta. Preliminary serpent-moose coupling and implementation of functional expansion tallies in serpent. In *M&C 2017*.

[5] Tara Pandya, Seth Johnson, Gregory Davidson, Thomas Evans, and Steven Hamilton. Shift: A massively parallel monte carlo radiation transport package.

[6] Brycen Wendt, Leslie Kerby, Aaron Tumulak, Jaakko Leppanen, and Mark DeHart. Advancement of functional expansion tallies capabilities in serpent 2.