Monte Carlo Mesh Tallies based on a Kernel Density Estimator Approach

By

Kerry L. Dunn

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Nuclear Engineering and Engineering Physics)

at the

UNIVERSITY OF WISCONSIN-MADISON

2014

Date of final oral examination: August 8th, 2014

The dissertation is approved by the following members of the Final Oral Committee:

Paul P. H. Wilson, Professor, Nuclear Engineering and Engineering Physics
Douglass L. Henderson, Professor, Nuclear Engineering and Engineering Physics
Gregory A. Moses, Professor, Nuclear Engineering and Engineering Physics
James P. Blanchard, Professor, Nuclear Engineering and Engineering Physics
Timothy J. Tautges, Adjunct Professor, Nuclear Engineering and Engineering Physics
Thomas M. Evans, Research Staff, Oak Ridge National Laboratory

ACKNOWLEDGEMENTS

Although completing a dissertation is never easy, the past seven years for me have been like being on a wild rollercoaster ride. Thankfully, however, there have been many people over the years who have managed to keep me on track. First and foremost, I would like to thank my advisor, Paul Wilson, for his patience and guidance. Without his support this dissertation would never have been completed. My sincerest thanks also goes to the other members of my doctoral committee for providing both their time and expertise.

I would also like to thank the fellowship program at the U. S. Nuclear Regulatory Commission and the Nuclear Energy University Program (NEUP) at the Department of Energy for their financial support. Without that financial support I would never have been able to pursue a graduate degree.

Completing this dissertation would also never have been possible without the love and support of my family and friends. To my mother, Rose, who always cheers me up with her snail mail letters. To Steve, the father who knows how to make me laugh (even though he never did support the "right" footy team). To my brother, Simon, who helped me purchase my very first computer when I knew nothing about RAM or CPU speeds. To everyone else I have not mentioned individually, who continue to support me in pursuing my personal and academic goals. Thank you.

Last of all, I wish to thank my best and special friend, Bertin. Not only did he make sure I was well fed, he also gave me the motivation to continue moving forward during the times when I needed it the most.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Nuclear systems such as fission reactors are based on complicated engineering designs consisting of many different materials and integrated components. An important part of reactor design is to determine how much radiation exists at any given location within these systems, which can then be used to approximate other physical quantities like heating in the materials, or the radiation dose that workers will receive during periods of maintenance. Types of radiation that can be present in a nuclear reactor include neutrons, photons, and electrons. The amount of neutrons, or neutron flux, can be determined by solving the neutron transport equation, which is an integrodifferential equation describing a balance between neutrons being added to and being removed from the system. One technique often used in practice for approximating the neutron flux is the Monte Carlo transport method. Monte Carlo methods are computational algorithms that use a stochastic approach for solving physical systems or mathematical problems that are typically difficult or impossible to solve analytically. The primary advantage of using the Monte Carlo transport method over more deterministic methods is that it can solve neutron transport problems on complex geometries – including recent advances to directly use detailed CAD models – without needing to explicitly define the neutron transport equation.

To obtain the neutron flux using the Monte Carlo transport method, first the path and energy of individual neutrons are tracked from their birth until their death through a series of randomly-determined collision events that are collectively defined as one particle history. If the neutron is scattered during one of these events, then its direction and energy changes and it

continues on to the next collision. If it is absorbed or escapes the system instead, then the neutron is no longer tracked and its corresponding particle history is terminated. After each particle history has been obtained, the next step in the Monte Carlo transport method is to assign a score to that history using an estimator. In general terms, an estimator is a mathematical rule that approximates some unknown statistical parameter – such as the average number of collisions, or the average distance that neutrons travel through some finite volume. Once each score has been assigned, it is then added to a Monte Carlo tally that accumulates all of the scores that occur inside the region of interest for which some physical quantity is desired. This entire process continues until a fixed number of particle histories have been completed. At the end of the simulation, the mean value for the region of interest and an error metric approximating the stochastic error are then computed from the tally results.

This dissertation explores the idea of using alternative estimators with the Monte Carlo transport method for solving fixed-source problems on arbitrary 3D input meshes. Compared to criticality problems, which require multiple generations of particle histories to converge to a steady state solution, fixed-source problems are based on a known neutron source that is non-multiplying. This means that only one set of particle histories is used to solve for the neutron flux distribution. In this introductory chapter, first an overview of two estimators that are commonly used to obtain the mean neutron flux are presented in Section 1.1. Then, the conventional approach for constructing mesh tallies using these estimators is described in Section 1.2. Section 1.3 discusses why this conventional approach can sometimes be ineffective, and then introduces two alternative estimators that can also be integrated with Monte Carlo

tallies. These alternatives are known as the functional expansion technique (FET) and the kernel density estimator (KDE). Finally, Section 1.4 presents an outline of the scope of this work.

1.1 Estimator Theory for Conventional Monte Carlo Tallies

The neutron flux $\varphi(\vec{r},E,t)$ is a quantity that can be expressed mathematically as the product of the neutron velocity *v* and the neutron density $N(\vec{r},E,t)$ at some position $\vec{r}$ and time *t* for neutrons with energy *E*:

$$\varphi(\vec{r},E,t) = vN(\vec{r},E,t), \tag{1.1}$$

measured in units of $cm^{-2}\cdot eV^{-1}\cdot s^{-1}$. If the functional form of the neutron flux is known, then the mean value for some finite volume *V* can be analytically determined by integrating over time, energy, and volume:

$$\overline{\varphi} = \frac{1}{V}\int_V\int_E\int_t \varphi(\vec{r},E,t)\,dt\,dE\,dV. \tag{1.2}$$

When no analytical solution exists, then the mean neutron flux must be approximated. The Monte Carlo transport method can be used for this purpose by simulating the behavior of *N* particle histories. However, before the simulation can begin, first an estimator must be chosen that will convert each of these particle histories into scores. While there are many different estimators available, Carter and Cashwell claim in their discussion on Monte Carlo particle transport methods that all estimators used in practice can be classified as collision estimators, last-event estimators, track length estimators and next-event (point-detector) estimators [1]. For approximating the mean neutron flux, however, the most commonly used are the collision and

track length estimators [2]. In the following sections, these two estimators are introduced. Note that the index i will refer to the i^{th} particle history, and the index c will refer to the sequence of collision events that occur for that history. The value of c begins at 0 when the neutron is born, and increases incrementally with each collision event up to a maximum of C_i when it is removed from the system.

1.1.1 The Collision Estimator

The collision estimator assigns a score for each particle history based on the number of collisions it experiences within some volume V. From these individual scores, the mean number of collisions per particle history $\bar{c}$ can be approximated by the following rule:

$$\bar{c} \approx \hat{c} = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{Ci} w_{ic}, \tag{1.3}$$

where w_{ic} represents the weight of the i^{th} particle before the c^{th} collision event is processed. While the particle weight is not always used in Monte Carlo transport simulations, it is necessary for adopting certain variance reduction techniques that can increase the fraction of histories that contribute to regions with low neutron density [2].

Once the mean number of collisions is obtained for V using Eq. 1.3, the mean neutron flux can then be obtained by associating this estimated quantity with the collision density $\Sigma_t\bar{\varphi}$:

$$\bar{c} \approx \Sigma_t \bar{\varphi} V, \tag{1.4}$$

where Σ_t is the total macroscopic cross section for V (measured in units of cm^{-1}), and the mean neutron flux is also normalized to the number of particle histories. By first solving Eq. 1.4 for

the mean neutron flux, and then substituting in Eq. 1.3 for the mean number of collisions, the Monte Carlo tally based on the collision estimator can be defined by:

$$\overline{\varphi} \approx \hat{\varphi} = \frac{1}{\Sigma_t V} \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{Ci} w_{ic}. \tag{1.5}$$

Note that the units in Eq. 1.5 are neutrons per cm^2 per particle history. To get an actual physical quantity, simply multiply by the source strength. For example, if the source is defined in terms of neutrons per second, then the mean neutron flux will be in units of $cm^{-2} \cdot s^{-1}$.

1.1.2 The Track Length Estimator

The collision estimator described in the previous section can only assign a score whenever a collision occurs inside the volume V. Therefore, if V is defined within a region where the collision density is low, or even non-existent, then this can lead to poor results with large variance. To improve the statistical errors in such cases, there are two different approaches that can be used. The first approach is to rely on variance reduction techniques, which includes the option of simply using more particle histories to produce more collision data. However, the second approach is to use a track length estimator instead of the collision estimator. Track length estimators assign a score for each particle history based on the total distance the neutron travels within V, which means that it can contribute to the tally even if the collision occurred outside the region of interest. From these individual scores, the mean track length per particle history $\bar{l}$ can be approximated by the following rule:

$$\bar{l} \approx \hat{l} = \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{Ci} w_{ic} d_{ic}, \tag{1.6}$$

where w_{ic} is once again the particle weight, and d_{ic} is the distance that the i^{th} particle traveled within V between collision events $c - 1$ and c.

To associate the mean track length per particle history with the mean neutron flux, note that substituting Eq. 1.1 into Eq. 1.2 produces a $v\,dt$ term, which represents the incremental distance dl that each neutron travels during time dt. By interpreting $N(\vec{r},E,t)\,dl$ as a track length density, the mean neutron flux can be estimated by averaging the total track length traveled by all of the neutrons:

$$\overline{\varphi} = \frac{1}{V}\bar{l}. \tag{1.7}$$

Combining Eq. 1.6 with Eq. 1.7 produces the following Monte Carlo tally based on the track length estimator:

$$\overline{\varphi} \approx \hat{\varphi} = \frac{1}{V}\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{Ci} w_{ic}d_{ic}. \tag{1.8}$$

Note that, like Eq. 1.5, the units in Eq. 1.8 are also neutrons per cm^2 per particle history.

1.2 Conventional Monte Carlo Mesh Tallies

Both the collision and track length estimators just described can be implemented as either a standard tally or a conventional mesh tally. The standard tally computes the mean neutron flux for a volume V that is conformal to one or more of the distinct volumes that already exist as part of the problem geometry. In contrast, the conventional mesh tally can accumulate scores anywhere by superimposing a separate input mesh over all or part of the problem geometry.

Each mesh cell represents its own volume V, which produces a piecewise constant approximation of the true neutron flux distribution throughout the domain defined by the input mesh. Traditionally, in Monte Carlo transport codes such as MCNP5 this input mesh could only be structured [3]. However, MCNP6, the most recent version of this code, also allows the use of unstructured meshes [4]. Figure 1.1 shows an example of both structured and unstructured mesh representations of a rectangular prism. Note that the structured mesh is based on orthogonal hexahedra, whereas the unstructured mesh consists of a set of tetrahedral elements.

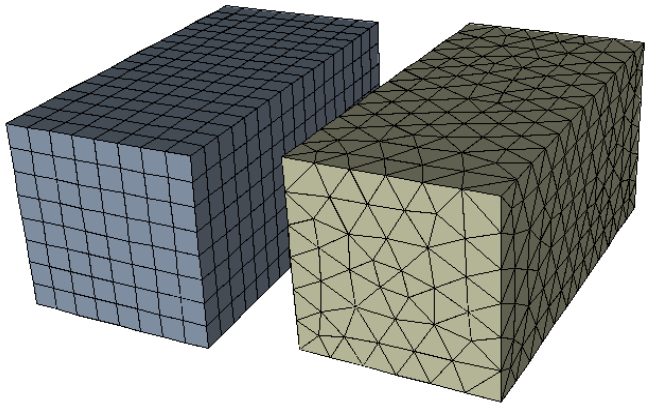

Figure 1.1 Structured (left) and unstructured (right) mesh representations.

The primary advantage of using unstructured meshes over structured meshes is that they conform better to non-orthogonal surfaces within the geometry. This conformal structure is especially useful when multiple materials are involved, since each tetrahedron within the mesh can be defined so that it only contains a single material. Figure 1.2 shows another example of an unstructured mesh for the flow insert in a detailed blanket module model of the International Thermonuclear Experimental Reactor (ITER). Given the curvature of the geometry for this flow insert, representing it accurately as a structured mesh would be significantly more challenging.

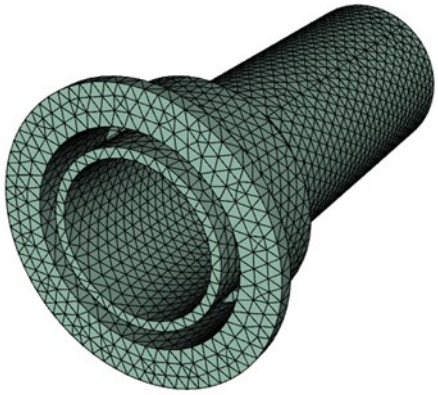

Figure 1.2 Unstructured mesh for the flow insert in a detailed blanket module model of ITER.

1.3 Alternative Monte Carlo Mesh Tallies

In general, conventional mesh tallies like the one implemented in MCNP5 use the track length estimator because it performs better in regions with low collision densities. However, these track length mesh tallies still only produce average values for the neutron flux within each mesh cell. While this piecewise constant approximation can reconstruct the neutron flux distribution in some cases, it can be ineffective in regions with strong gradients. The only way that the conventional approach can even attempt to capture these gradients is through mesh refinement. Unfortunately, refining the mesh to capture any fine details also reduces the size of the mesh cells. Using smaller mesh cells means that fewer histories contribute to the individual tallies distributed throughout the entire mesh, which leads to increased statistical uncertainty in the final results. One way to reduce this increased statistical uncertainty in regions with strong gradients is to simply track more particle histories. However, both refining the mesh and increasing the particle histories can substantially increase the time it takes to obtain a valid solution within the desired margin of error. Not only are more particles being used, but for

unstructured meshes each of those particles also needs to be processed through more internal surfaces as they travel through the refined mesh. Instead of tracking more particle histories, another option is to consider using the concept of density estimation methods.

1.3.1 Density Estimation Methods

Density estimation methods are often used by statisticians to reconstruct an unknown probability density function directly from a finite sample of randomly-determined observations. A probability density function $f(x)$ based on some continuous random variable X is defined as a function that assigns a probability to each possible value of X that can occur. In Monte Carlo neutron transport, for example, X could represent the number of collisions experienced by one neutron, or the total track length it has traveled within some region. For both of these examples, possible values of X includes any x that lies within the interval $[0, \infty)$. The probability that X will be between the values a and b can be determined from $f(x)$ as follows:

$$P(a \le X \le b) = \int_a^b f(x)dx. \tag{1.9}$$

Note that $f(x)$ can also be used to define the cumulative distribution function $F(x)$, which determines the probability of X being less than or equal to some value x:

$$P(X \le x) = F(x) = \int_{-\infty}^{x} f(x)dx. \tag{1.10}$$

Some important properties of $f(x)$ are that it must be non-negative and integrate to unity over its domain to represent valid probabilities. An example of a probability density function and its

corresponding cumulative distribution function is shown in Figure 1.3 for the standard normal distribution $N(0, 1)$. The general normal distribution $N(\mu, \sigma^2)$ is defined by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \tag{1.11}$$

where μ and σ^2 are statistical parameters known as the mean and variance respectively.

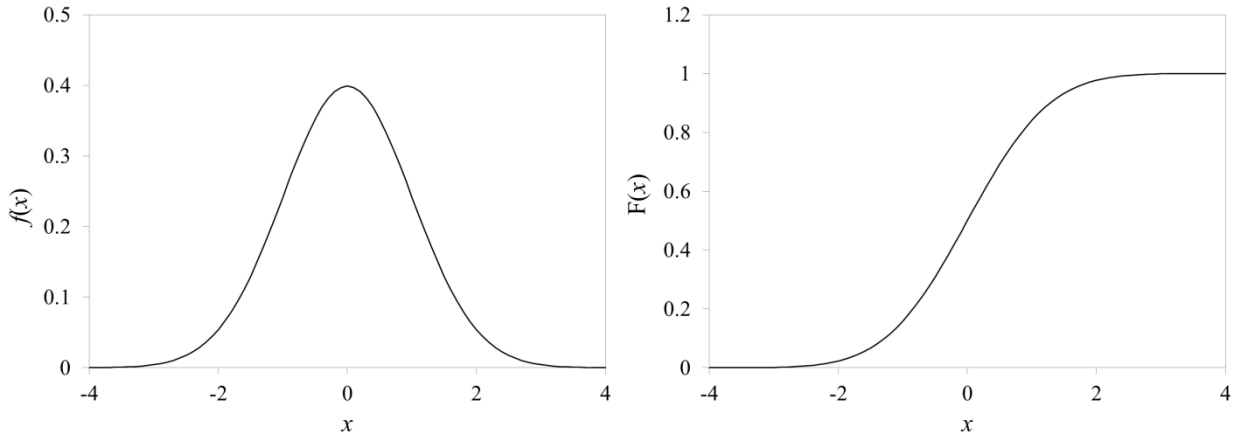

Figure 1.3 Probability density function (left) and cumulative distribution function (right) of the standard normal distribution with $\mu = 0$ and $\sigma^2 = 1$.

Density estimation methods for approximating probability densities like the one shown in Figure 1.3 can be separated into two distinct cases – parametric and non-parametric. Parametric density estimation methods assume that the unknown probability density function belongs to some known parametric family of distributions. Knowing the general shape of the distribution allows a theoretical function to be fit to the data by approximating its required parameters. As an example, consider a set of observations sampled from an unknown $N(\mu, \sigma^2)$ distribution. The parametric approach would be to estimate the mean μ and variance σ^2 from these observations so that Eq. 1.11 can then be used to reconstruct its shape. In many cases, however, the general

shape of the distribution cannot be known *a priori*. This is when non-parametric methods must be used. Non-parametric density estimation methods do not need to assume that the unknown probability density function belongs to a specific family of distributions. Instead, these methods only rely on the observations themselves for reconstructing the shape of the distribution.

The accuracy of any approximation obtained using a density estimator will usually depend on both bias and variance. Bias is defined as the difference between the expected value (i.e., mean value) of the approximated density $\hat{f}(x)$ and the true density $f(x)$:

$$bias\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x)\right] - f(x), \tag{1.12}$$

where the expected value for some function $g(x)$ of a continuous random variable X with probability density function $f(x)$ is defined by:

$$\mathrm{E}[g(x)] = \int_{-\infty}^{\infty} g(x) f(x) dx. \tag{1.13}$$

While bias determines how close the approximation is to the true solution, variance is a measure of its precision with respect to the expected value. Therefore, the variance of an approximation obtained using a density estimator can be expressed as:

$$var\left[\hat{f}(x)\right] = \mathrm{E}\left[\left(\hat{f}(x) - \mathrm{E}\left[\hat{f}(x)\right]\right)^2\right] = \mathrm{E}\left[\hat{f}(x)^2\right] - \left(\mathrm{E}\left[\hat{f}(x)\right]\right)^2. \tag{1.14}$$

1.3.2 Density Estimation Methods for Monte Carlo Neutron Transport

Since the neutron flux must be non-negative everywhere to be a valid physical quantity, it can always be treated as an un-normalized probability density function that does not belong to a specific family of distributions. Therefore, non-parametric density estimation methods can be

used, in theory, for approximating the neutron flux at any single point (x, y, z) within some arbitrary domain. The conventional mesh tally described in Section 1.2 is actually equivalent to constructing a simple histogram, which is one of the most widely used non-parametric density estimators [5]. Although this conventional approach produces unbiased results with respect to the average neutron flux over the volume being tallied, the mean value theorem only guarantees that at least one point inside that volume has that specific value. Attributing the average neutron flux to any other point can introduce a bias in the results.

Rather than relying on the histogram estimator, an alternative option is to consider other more sophisticated density estimators for use in Monte Carlo neutron transport. Two additional non-parametric methods have been developed within the past decade for use as alternative collision and track length tallies. The first non-parametric method is called the functional expansion technique, which was studied in depth by Griesheimer [6]. These FET tallies reconstruct the neutron flux distribution by representing it as a finite series expansion, using some orthogonal basis set such as the Legendre Polynomials. While the FET approach was shown to be effective for some 1D and 2D problems, there are a couple of issues that limit its usefulness for implementation as a 3D mesh tally. Since many basis sets are restricted to finite domains, this means that explicit boundaries need to be defined within the mesh structure in order to expand the approximation using the chosen basis functions. This would be challenging to do effectively for unstructured meshes, and is even more of an issue if the underlying distribution contains any material discontinuities. Without using a piecewise approximation or a large expansion order, the FET approach has been shown to have difficulties resolving these material discontinuities for criticality problems [6, 7].

The second non-parametric method considered for use as an alternative tally is called the kernel density estimator. The earliest reference to what would later become known as the naïve estimator appeared in an unpublished report by Fix and Hodges in 1951 [8]. It was not until 1956 that the first published work appeared, authored by Rosenblatt [9]. This work discussed both the naïve estimator and the more general univariate kernel density estimator. Then, in 1966 these univariate concepts were extended to the multivariate case by Cacoullos [10]. Over the past 50 years, the KDE approach has been shown to be an effective density estimation method for numerous practical applications, ranging from basic statistical analysis to nuclear engineering. For particle transport applications in particular, the first examples of its use were as alternative techniques for resampling from electron [11] and photon [12] distributions. Then, in 2009 Banerjee and Martin introduced KDE versions of conventional collision and track length estimators for use in Monte Carlo neutron transport [13]. Since 2009, these KDE tallies have been studied as a viable alternative for solving 1D and 2D criticality problems [7, 14], and can even be used instead of conventional point detector or surface flux tallies [7, 15]. Most recently, the KDE approach has also been considered for tallying on arbitrary 3D meshes for more general transport problems [16, 17], as well as for computing reaction rates [18].

KDE mesh tallies offer a significant improvement over both conventional and FET mesh tallies primarily because they can approximate the neutron flux at exact locations on the mesh independently of its underlying structure. Therefore, unlike the conventional histogram approach, KDE mesh tallies are not affected by the size, shape, origin, or orientation of the mesh cells. No matter where the mesh is located or how it is constructed, the final tally results will always be the same for a fixed point in the domain. Another advantage of being independent of

the mesh structure is that no knowledge of internal mesh boundaries is required – even when those boundaries separate different materials. This means that KDE mesh tallies can use the same implementation for both structured and unstructured meshes, and that no particles ever need to be tracked across any additional surfaces if the mesh resolution is refined.

1.4 Scope of Work

Previous work on the kernel density estimator in the context of nuclear engineering only focused on using standard KDE tallies for solving 1D or 2D criticality problems. The goal of this dissertation is to determine if the kernel density estimator is also a viable alternative for solving fixed-source problems on arbitrary 3D input meshes. To be a viable alternative for the conventional mesh tally, the kernel density estimator must be able to produce accurate results as efficiently as possible. Ease of use in obtaining and understanding the results should also be a consideration. A brief overview of the structure of this dissertation follows.

Chapter 2 begins by exploring the use of the multivariate kernel density estimator for mesh tallies. Since the original KDE collision and track length estimators can be less effective for general fixed-source problems, a new KDE track length estimator based on integrated particle tracks is proposed. The implementation of this new estimator as a KDE mesh tally within a production Monte Carlo transport code is also described.

The KDE integral-track mesh tally introduced in Chapter 2 is compared to other mesh tally implementations in Chapter 3 for verification purposes. Detailed quantitative comparisons are described and performed using a common set of five simple fixed-source problems. Both the original KDE track length estimator, which will be referred to as the KDE sub-track estimator,

and the conventional histogram estimator are considered in this analysis. After comparing the results for these five simple problems, a demonstration of the usage of the KDE integral-track mesh tally for a more complicated fixed-source problem is presented and compared to the conventional mesh tally.

Whereas Chapter 3 focuses on quantitatively comparing the KDE integral-track mesh tally to other mesh tally implementations, Chapter 4 explores topics related to its accuracy with respect to the true solution. These topics include the sensitivity of different parameter choices that can be used with KDE tallies, as well as an introduction to both bias reduction and boundary correction methods for 3D problems.

In Chapter 5 the focus shifts from accuracy to the efficiency of the KDE integral-track mesh tally. A performance comparison is presented using the same five test cases from Chapter 3. Also included in this chapter is a detailed discussion on key factors and problem characteristics that can affect the performance of the KDE integral-track mesh tally.

The final chapter of this dissertation provides a summary of each of the preceding chapters, as well as discussing what future work is needed before the KDE integral-track mesh tally can become a viable alternative to the conventional mesh tally for solving general fixed-source problems.

CHAPTER 2

Kernel Density Estimated Monte Carlo Mesh Tallies

Both the functional expansion technique and the kernel density estimator method introduced in Chapter 1 are capable of producing a continuous representation of the neutron flux, rather than the piecewise constant approximation obtained using conventional methods. In addition, each collision or track length that is tallied is allowed to contribute to more than one tally point in the domain. As such, both of these non-parametric density estimation methods can potentially be more effective than conventional estimators when detailed information about the neutron flux distribution is needed. For the purposes of solving fixed-source neutron transport problems on arbitrary 3D input meshes, however, the kernel density estimator is preferable to the functional expansion technique because it is always independent of the mesh structure. Other advantages of using KDE mesh tallies over FET mesh tallies are that in most cases it is guaranteed to produce non-negative results, and that it also performs better in regions with material discontinuities [7, 14].

In this chapter, the multivariate kernel density estimator is first introduced in Section 2.1. Then, the original KDE estimators that were developed as alternatives to conventional collision and track length estimators will be presented in Section 2.2. Section 2.3 discusses the application of these estimators to mesh tallies, and explains why they can be ineffective for fixed-source problems. In Section 2.4 a new KDE estimator based on integrated particle tracks is derived. Finally, Section 2.5 discusses the implementation of this KDE integral-track estimator as a mesh tally.

2.1 Multivariate Kernel Density Estimator

The multivariate kernel density estimator can approximate the value of some unknown probability density function $f(x_1, x_2, \ldots, x_d)$ at any given calculation point $(x_1, x_2, \ldots, x_d)$ using a fixed set of N observations sampled from that distribution. For the 3D case, this fixed set of observations can be denoted by $S = \{(X_i, Y_i, Z_i) : i = 1 \text{ to } N\}$. One approach for defining a 3D kernel density estimator based on S is to compute the average kernel contribution for a product of three 1D kernels:

$$\hat{f}(x, y, z) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h_x} K\left(\frac{x - X_i}{h_x} \right) \frac{1}{h_y} K\left(\frac{y - Y_i}{h_y} \right) \frac{1}{h_z} K\left(\frac{z - Z_i}{h_z} \right), \tag{2.1}$$

where (h_x, h_y, h_z) is known as the bandwidth vector, and the shape of the kernel function $K(u)$ determines the weight of each of the observations X_i, Y_i, or Z_i with respect to x, y, or z respectively. Visually, this can be imagined as a group of individual kernels placed around each of the N observations that are then added together to form an approximation of the true density $f(x, y, z)$. Note that the width of these kernels in each dimension is dictated by the size of the bandwidth that is used, which ultimately determines the amount of smoothing (i.e., statistical noise) that is present in the final results.

2.1.1 Bandwidth Vector

Theoretically, any non-zero bandwidth vector with positive components can be used with Eq. 2.1 to approximate the true density $f(x, y, z)$. However, choosing poor values for this smoothing parameter can significantly impact the accuracy of the results at *all* calculation points (x, y, z). A mathematical analysis concerning this issue has already been discussed in detail by

Banerjee for both univariate and multivariate kernel density estimators [7]. Based on this analysis, it was shown that the choice of bandwidth is essentially a trade-off between bias and variance. Using smaller values minimizes the bias at the cost of higher variance, whereas using larger values has the opposite effect. The optimal value falls somewhere in between these two extremes. One method for computing an optimal bandwidth for a *d*-dimensional kernel density estimator is to use the following definition in each dimension *i* [7, 14]:

$$h_i = \left\{ \frac{4}{(d+2)N} \right\}^{\frac{1}{d+4}} \sigma_{X_i}, \tag{2.2}$$

where for the 3D case σ_{X_i} would be the standard deviation of the X_i, Y_i or Z_i component of the N observations that were used to approximate $f(x, y, z)$.

2.1.2 Kernel Function

In addition to choosing an appropriate bandwidth vector, it is also necessary to decide what kernel function $K(u)$ will be used with Eq. 2.1 to approximate $f(x, y, z)$. A kernel is defined as any function that satisfies the following property:

$$\int_{-\infty}^{\infty} K(u)du = 1. \tag{2.3}$$

If $K(u)$ is also non-negative throughout its domain, then it is itself a probability density function. This property is inherited by the kernel density estimate $\hat{f}(x, y, z)$, along with the continuity and differentiability properties of the kernel function [5]. Other properties of kernels that are commonly used in density estimation include:

$$\textbf{i}.\ K(u) = K(-u), \qquad \textbf{ii}.\ \int_{-\infty}^{\infty} uK(u)du = 0, \qquad \textbf{iii}.\ \int_{-\infty}^{\infty} u^2 K(u)du \neq 0, \tag{2.4}$$

which collectively state that the kernel function is typically symmetric, has a mean value of zero, and has non-zero variance.

All kernels satisfying the properties of Eq. 2.4 are considered to be of second-order.[1] One group of second-order kernels often used in density estimation is a subset of the general polynomial kernel family that was first introduced by Müller in 1984 [19]. These second-order kernels can be generated from the following common formula [20]:

$$K_s(u) = \frac{\left(\frac{1}{2}\right)_{s+1}}{s!}(1-u^2)^s, |u| \leq 1. \tag{2.5}$$

Note that the term in brackets with the subscript in the numerator of Eq. 2.5 is evaluated using the Pochhammer symbol:

$$(x)_n = x(x+1)(x+2)...(x+n-1), \tag{2.6}$$

where $(x)_0 = 1$ and $(x)_1 = x$.

The first few second-order polynomial kernels generated by Eq. 2.5 are shown in Table 2.1. Along with their common name and functional form $K_s(u)$, Table 2.1 also includes values for both their variance $\int u^2 K_s(u)du$ and roughness $\int K_s(u)^2 du$. All four second-order polynomial kernels defined in Table 2.1 are also plotted in Figure 2.1 on page 21. As the level of smoothness s of the function increases, the kernel applies more weight toward its middle and less

[1] The kernel order is different from the order of a polynomial function.

weight toward its tails. This means, for example, that the uniform kernel ($s = 0$) will add the same contribution for every observation. However, for the Epanechnikov kernel ($s = 1$), and all other second-order polynomial kernels, observations that are further away from the calculation point will add smaller contributions than those that are closer. The Epanechnikov kernel is considered to be the most efficient 1D kernel function [5], since it minimizes the mean integrated squared error (MISE). The MISE is often used to measure the effectiveness of a kernel density estimate with respect to the true density $f(x, y, z)$. It is defined as the expected value of the integral of the square of the difference between the actual values and the approximated values:

$$MISE\left\{\hat{f}(x,y,z)\right\} = E\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left(f(x,y,z)-\hat{f}(x,y,z)\right)^2 dxdydz. \tag{2.7}$$

Table 2.1 Second-order kernels from the polynomial kernel family up to $s = 3$.

Kernel Name	***s***	$\boldsymbol{K_s(u)}$	$\int \boldsymbol{u^2 K_s(u) du}$	$\int \boldsymbol{K_s(u)^2 du}$
Uniform	0	$\frac{1}{2}, \lvert u\rvert \le 1$	$\frac{1}{3} \approx 0.33$	$\frac{1}{2} = 0.50$
Epanechnikov	1	$\frac{3}{4}(1-u^2), \lvert u\rvert \le 1$	$\frac{1}{5} = 0.20$	$\frac{3}{5} = 0.60$
Biweight	2	$\frac{15}{16}(1-u^2)^2, \lvert u\rvert \le 1$	$\frac{1}{7} \approx 0.14$	$\frac{5}{7} \approx 0.71$
Triweight	3	$\frac{35}{32}(1-u^2)^3, \lvert u\rvert \le 1$	$\frac{1}{9} \approx 0.11$	$\frac{350}{429} \approx 0.82$

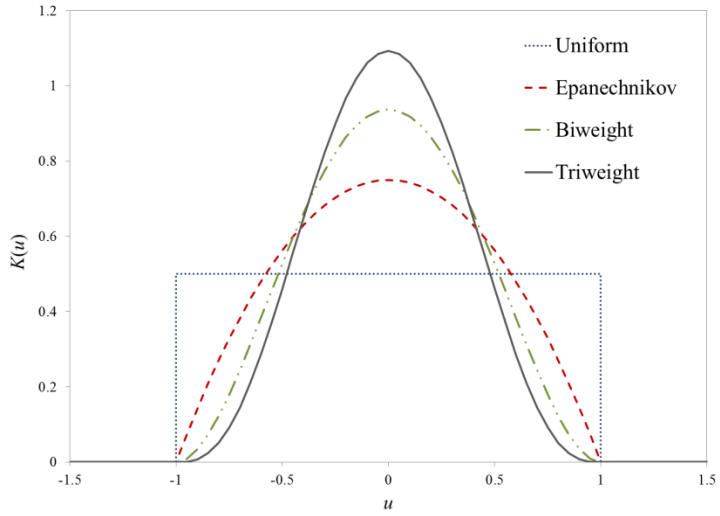

Figure 2.1 Second-order Polynomial Kernels.

2.2 KDE Collision and Sub-track Estimators

The 3D kernel density estimator defined by Eq. 2.1 was used by Banerjee to derive KDE versions of conventional collision and track length tallies [7, 14]. Both of these tallies are capable of approximating the neutron flux $\varphi(x, y, z)$ at *any* point within a 3D system, although they use slightly different estimators. The KDE collision tally is based on an estimator that uses the collision sites as the set of observations from which the kernel contributions are evaluated. Given some fixed calculation point (x, y, z), the contribution to the flux at that point can be computed for the i^{th} particle history using the following definition:

$$\hat{\varphi}_i = \sum_{c=1}^{C_i} \frac{w_{ic}}{\Sigma_t(X_{ic}, Y_{ic}, Z_{ic})} \frac{1}{h_x} K\left(\frac{x - X_{ic}}{h_x}\right) \frac{1}{h_y} K\left(\frac{y - Y_{ic}}{h_y}\right) \frac{1}{h_z} K\left(\frac{z - Z_{ic}}{h_z}\right), \tag{2.8}$$

where C_i is the number of collision events experienced by the particle, w_{ic} is its weight, and Σ_t is the macroscopic cross section of the material defined at the collision site (X_{ic}, Y_{ic}, Z_{ic}).

Like conventional collision tallies, the KDE collision tally suffers in regions with low collision density and cannot be used at all in void regions. So, to be able to handle general fixed-source problems, it is usually preferable to use an estimator based on track length. As an alternative to using the collision sites, the KDE track length tally developed by Banerjee is based on an estimator that chooses n pseudo-collision points (X_{icj}, Y_{icj}, Z_{icj}) along the track as the set of observations from which the individual kernel contributions K_{icj} are evaluated. These points are selected by first splitting up the total track length d_{ic} into a series of n evenly distributed sub-tracks, and then by randomly choosing one point within each region. The contribution to the flux for the i^{th} particle history at some fixed calculation point (x, y, z) is then computed using an average of these individual kernel contributions as follows:

$$\hat{\varphi}_i = \sum_{c=1}^{C_i} w_{ic} d_{ic} \frac{1}{n} \sum_{j=1}^{n} K_{icj}, \tag{2.9}$$

where K_{icj} is usually defined as the following product of three 1D kernel functions:

$$K_{icj} = \frac{1}{h_x} K\left(\frac{x - X_{icj}}{h_x}\right) \frac{1}{h_y} K\left(\frac{y - Y_{icj}}{h_y}\right) \frac{1}{h_z} K\left(\frac{z - Z_{icj}}{h_z}\right). \tag{2.10}$$

2.3 Applying Kernel Density Methods to Monte Carlo Mesh Tallies

Since the KDE collision and KDE sub-track estimators discussed in the previous section can approximate the neutron flux anywhere within some arbitrary geometrical domain, it is possible to implement them both as an alternative Monte Carlo mesh tally. The main consideration needed to expand their usage to mesh tallies is to determine the set of calculation points at which the kernel density estimator is to be evaluated. This can be done by using the set

of mesh nodes defined by an arbitrary input mesh, which makes the KDE approach well-suited to neutron flux tallies on both structured and unstructured meshes for three key reasons. First, the mesh nodes provide a useful set of calculation points for obtaining a good representation of the entire flux distribution throughout the region of interest. Second, as was mentioned in Chapter 1, KDE mesh tallies are not affected by the size or shape of the mesh cells. This means that refinement of the mesh to obtain higher fidelity results can be done by simply increasing the number of calculation points considered in the analysis, without having to worry about increased statistical error. Third, KDE mesh tallies also do not need to track particles across internal mesh surfaces like some conventional mesh tallies. This may provide some computational benefit in regions with a high density of surfaces.

Even though both KDE collision and KDE sub-track estimators can be used as a mesh tally, the accuracy of their contributions depends on the either the collision density or the number of sub-tracks. While the KDE sub-track estimator is preferable to the KDE collision estimator, too few sub-tracks used with longer particle tracks could produce less accurate results because only a small portion of the full track length is being considered. If this was the case, then a mesh tally based on the KDE sub-track estimator would not be much better than one based on the KDE collision estimator. While it is easy enough to simply increase the number of sub-tracks that are used, for a 3D mesh tally the neutron flux must be evaluated at numerous mesh nodes for each track segment. Using too many sub-tracks for these calculations could become expensive computationally, especially when considering large meshes that need to keep track of millions of particle histories. Therefore, a better approach is to consider an alternative KDE estimator that uses integrated particle tracks instead of pseudo-collisions.

2.4 KDE Integral-track Estimator

Beginning with the KDE sub-track estimator defined by Eq. 2.9 and 2.10, a new KDE track length estimator based on integrated particle tracks is now derived. The first step is to redefine the pseudo-collision points (X_{icj}, Y_{icj}, Z_{icj}) in terms of a common path length variable S_{icj}:

$$\textbf{i}.\ X_{icj} \to X_{\mathrm{o}} + uS_{icj}, \qquad \textbf{ii}.\ Y_{icj} \to Y_{\mathrm{o}} + vS_{icj}, \qquad \textbf{iii}.\ Z_{icj} \to Z_{\mathrm{o}} + wS_{icj}, \tag{2.11}$$

where (X_{o}, Y_{o}, Z_{o}) is the location of the previous collision event, and (u, v, w) is the unit direction vector pointing toward the next collision event. A graphical representation of the transition from randomly chosen points to their path length equivalents can be seen in Figure 2.2 for the X-component of an arbitrary pseudo-collision point.

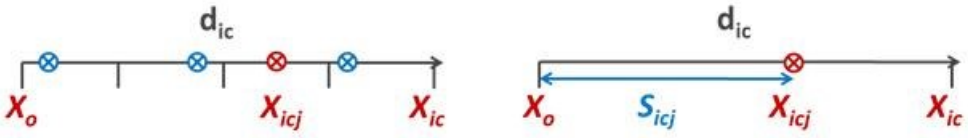

Figure 2.2 Graphical representation of the transition from a randomly chosen point X_{icj} within an individual sub-track (left) to its path length equivalent S_{icj} (right).

Substituting the three expressions of Eq. 2.11 into Eq. 2.10 and applying it to the second summation of Eq. 2.9 results in the following definition:

$$\frac{1}{n}\sum_{j=1}^{n} K_{icj} = \frac{1}{n}\sum_{j=1}^{n} \frac{1}{h_x} K\left(\frac{x - X_{\mathrm{o}} - uS_{icj}}{h_x}\right) \frac{1}{h_y} K\left(\frac{y - Y_{\mathrm{o}} - vS_{icj}}{h_y}\right) \frac{1}{h_z} K\left(\frac{z - Z_{\mathrm{o}} - wS_{icj}}{h_z}\right). \tag{2.12}$$

Note that Eq. 2.12 is the same summation as the one defined within the KDE sub-track estimator, with the exception that it now uses path length instead of pseudo-collision points as the random

variable. By taking the limit as the number of sub-tracks n approaches infinity, this average of individual kernel contributions can be converted into an integral over track length:

$$\lim_{n\to\infty}\left[\frac{1}{n}\sum_{j=1}^{n}K_{icj}\right]=\frac{1}{d_{ic}}\int_{0}^{d_{ic}}\frac{1}{h_x}K\left(\frac{x-X_{\mathrm{o}}-uS}{h_x}\right)\frac{1}{h_y}K\left(\frac{y-Y_{\mathrm{o}}-vS}{h_y}\right)\frac{1}{h_z}K\left(\frac{z-Z_{\mathrm{o}}-wS}{h_z}\right)dS. \quad (2.13)$$

Then, substituting Eq. 2.13 back into Eq. 2.9, and noting that the total track length term d_{ic} cancels, the final form of the KDE integral-track estimator for the i^{th} particle history becomes:

$$\hat{\varphi}_i=\sum_{c=1}^{C_i}w_{ic}\int_{0}^{d_{ic}}K_S\,dS, \quad (2.14)$$

where K_S is used to represent the following 3D kernel as a function of path length S:

$$K_S=\frac{1}{h_x}K\left(\frac{x-X_{\mathrm{o}}-uS}{h_x}\right)\frac{1}{h_y}K\left(\frac{y-Y_{\mathrm{o}}-vS}{h_y}\right)\frac{1}{h_z}K\left(\frac{z-Z_{\mathrm{o}}-wS}{h_z}\right). \quad (2.15)$$

Since the KDE integral-track estimator uses integrated particle tracks instead of pseudo-collisions, it is expected to produce more accurate results than the KDE sub-track estimator. Quantitative comparisons between these two KDE estimators and the conventional histogram estimator are discussed in detail in Chapter 3.

2.5 KDE Integral-track Mesh Tally Implementation

Extending the KDE integral-track estimator to a mesh tally can be done by computing an average contribution from the individual histories at all mesh nodes (x, y, z) from some arbitrary input mesh. This arbitrary input mesh can be either structured or unstructured, or even include components from both types. The formal definitions of the KDE integral-track mesh tally for computing the expected value of the neutron flux and its corresponding variance are as follows:

$$\hat{\varphi}(x,y,z) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C_i} w_{ic} \int_0^{d_{ic}} K_S dS, \tag{2.16}$$

$$\sigma^2_{\hat{\varphi}(x,y,z)} = \frac{1}{N-1}\left\{ \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{c=1}^{C_i} w_{ic} \int_0^{d_{ic}} K_S \, dS \right)^2 - \hat{\varphi}(x,y,z)^2 \right\}, \tag{2.17}$$

where N is the total number of particle histories used in the Monte Carlo simulation. Another way to envision how the KDE integral-track mesh tally works is to consider the set of mesh nodes for which the neutron flux is obtained as a series of fixed individual tally points or detectors, without the $1/R^2$ singularity that can cause issues with conventional point detector estimators [7, 15]. As each track length d_{ic} is processed during the random walk, all mesh nodes within some neighborhood region around this track will add a new score to their corresponding tally sums. Nodes further from the track typically add smaller contributions than those closer to the track, depending on the kernel function that is used in the approximation.

An initial implementation of the KDE integral-track mesh tally defined above has been added to a new C++ library called DAGTally (see APPENDIX A) as an optional part of the direct accelerated geometry Monte Carlo (DAGMC) workflow. The DAGMC workflow allows Monte Carlo simulations to be performed directly on CAD models [21]. With the addition of DAGTally, now both the geometry and tally behavior can be abstracted from the rest of the simulation – allowing a common implementation to be used by different Monte Carlo transport codes with minimal programming effort. For DAGTally in particular, this also means that new tally types can be added without having to change the general interface. Other tallies implemented within DAGTally that will be mentioned in this dissertation include both KDE

collision and KDE sub-track mesh tallies, as well as an unstructured version of the conventional mesh tally that shall be referred to as the TET track mesh tally. All results presented in the following chapters were obtained using DAG-MCNP5, which is the integration of DAGMC functionality within MCNP5. DAG-MCNP5 allows either native MCNP5 or CAD geometries to be defined, and also offers the option to use both native MCNP5 tallies and the advanced mesh tallies from DAGTally. Note that the native MCNP5 mesh tally will be referred to as simply the MCNP5 mesh tally from this point on.

The following four sections introduce some important features and limitations of the KDE integral-track mesh tally implemented within DAGTally – including the choice of bandwidth vector, the kernel function options that are available, the boundary bias effect, and the concept of the neighborhood region. The first three of these topics affect the accuracy of the tally results, whereas the last topic only affects the efficiency in obtaining those results. Accuracy and efficiency of the KDE integral-track mesh tally are discussed in more detail in Chapters 4 and 5 respectively.

2.5.1 Choice of Bandwidth Vector

Like all applications based on KDE methods, the accuracy of the KDE integral-track mesh tally is highly dependent on the choice of bandwidth vector. Using smaller values means that fewer track segments contribute to each mesh node, which increases the variance in the results. On the other hand, using larger values means that the domain around each mesh node for which contributions are added is so big that it over smooths the solution – causing increased bias and potentially hiding important features of the neutron flux distribution. For criticality

problems, an optimal bandwidth can always be computed using Eq. 2.2 for the current active cycle from data obtained during the previous active cycle [7, 14]. However, unlike criticality problems, fixed-source problems do not rely on an iterative solving process. This means that every Monte Carlo simulation would need to be run twice – once to find an optimal bandwidth, and again to produce the final tally results.[2] Therefore, it was decided to require that the user choose a global bandwidth vector *a priori* for all mesh nodes defined within the input mesh. The effect that this choice can have on the results will be explored further in Chapter 4.

2.5.2 Kernel Function Options

Once the bandwidth vector is chosen, its components will determine the width of the kernel function in each dimension. While the accuracy of the KDE integral-track mesh tally is primarily dependent on this kernel width, it can also be impacted by the general shape of the kernel function. Previously in Section 2.1, second-order polynomial kernels were introduced. This group of kernel functions, defined by Eq. 2.5, has been implemented within DAGTally. In addition, symmetric polynomial kernels of higher-order are also available. These higher-order kernels will be defined and explored as a bias reduction tool for the KDE integral-track mesh tally in Chapter 4. As a future expansion, it may also be worth adding the Gaussian kernel family [22] to DAGTally. The primary advantage of the Gaussian kernel is that it has derivatives of all orders, which can be useful if higher-order derivatives of the kernel density estimate are needed. For solving fixed-source problems, however, these kernels may need to be evaluated millions or even billions of times per Monte Carlo simulation. Since each evaluation

[2] Another option is to store all particle history data, then compute bandwidth and results during post-processing.

of the Gaussian kernel requires that an exponential series be computed, other low-*s* polynomial options such as the Epanechnikov kernel are preferable.

2.5.3 Boundary Bias Effect

In contrast to the bandwidth, which affects all tally results, another issue that needs to be addressed whenever KDE methods are used only affects mesh nodes that are less than one bandwidth from an external geometrical boundary. For these nodes the kernel function overlaps with the boundary, producing results that are underestimated [5]. This increased bias occurs partially because the kernel can no longer integrate to unity as required by Eq. 2.3, since the underlying probability density function being approximated is undefined outside these external boundaries. An example where this can occur in a 3D Monte Carlo transport application is shown in Figure 2.3, which compares results from both KDE integral-track and MCNP5 mesh tallies for a simple gradient problem with reflecting boundaries.

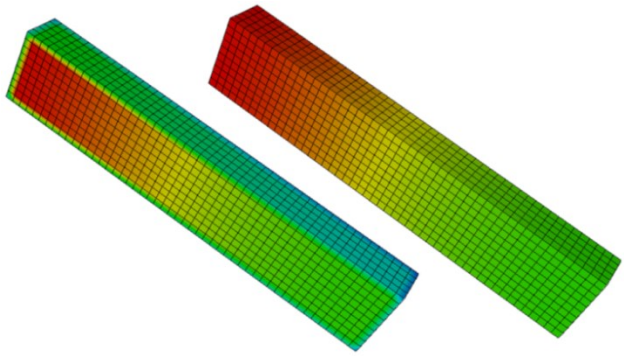

Figure 2.3 KDE integral-track mesh tally results obtained without using a boundary correction (left), compared to the reference MCNP5 solution (right) for a simple gradient problem with reflecting boundaries.

Because the reflecting boundary condition forces the neutron flux distribution to be undefined outside the prism, there is a noticeable difference in the results for mesh nodes located

on its surface. Note that a similar situation would also occur if vacuum boundaries were used, since the neutrons would no longer be tracked once they escaped the system. Fortunately, the boundary bias issue does not affect any of the internal mesh nodes where the neutron flux is well defined. This means that as long as the mesh boundaries are defined at a distance of at least one bandwidth from any external geometrical boundary, then the KDE integral-track mesh tally can still be compared to other mesh tallies for verification purposes. Note that there are several methods available within the statistical literature that can correct this boundary bias effect. This topic will be discussed in Chapter 4 in the context of the KDE integral-track mesh tally.

2.5.4 Neighborhood Region

The bandwidth vector, kernel function, and boundary bias all affect the accuracy of the KDE integral-track mesh tally. However, since typical input meshes can have thousands or even millions of nodes, the concept of the neighborhood region is only essential for efficiency purposes. This neighborhood region is defined as the local region in space around a single tally event for which the kernel function produces a non-trivial contribution for any mesh node inside that region. Although most kernels are confined to a finite domain that ensures no contribution is ever added for mesh nodes outside this region, checking too many trivial calculation points per tally event can add a significant penalty to the overall computing time. By limiting the number of nodes that are evaluated before any calculations are made, this penalty is kept to a minimum.

For a mesh tally based on the KDE collision estimator, defining a neighborhood region for each collision is straightforward. An axis-aligned box is simply placed around the collision site, with its size determined only by the bandwidth that was used. In contrast, attempting to

define the true neighborhood region for a KDE integral-track mesh tally becomes more complicated because its size is determined by more than just the bandwidth. To make things even more difficult, the shape of this region depends on the orientation of the track segment with respect to its starting location. The simplest case occurs when the particle is traveling parallel to any of the three coordinate axes, which results in a neighborhood region that is equivalent to an axis-aligned box around the track segment. For the most challenging case, however, the particle is traveling in a direction that is not aligned with any axis. In this case, the neighborhood region would look something like the inner shape with a hexagonal cross section shown in Figure 2.4.

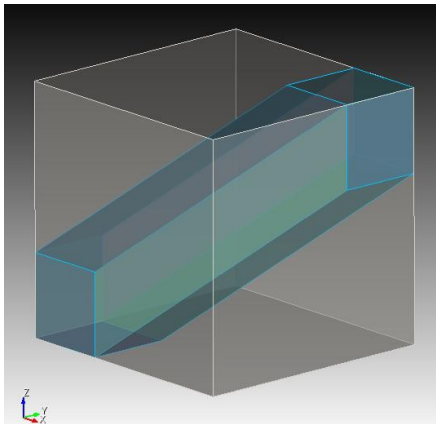

Figure 2.4 Sample neighborhood region for one track segment contributing to a KDE integral-track mesh tally.

Because these neighborhood regions can get so complicated to define for an arbitrary track segment, it is often necessary to construct an approximation instead. One approximation for the neighborhood region of a single particle track is to use the outer cubed region shown in Figure 2.4. This approximation is defined mathematically in Chapter 5, where a more detailed discussion of the neighborhood region and how it affects the efficiency of the KDE integral-track mesh tally will also be presented.

CHAPTER 3

Quantitative Mesh Tally Comparisons

Previous work that assessed the effectiveness of KDE [7, 14] and FET [6] tallies for criticality problems only considered 1D or 2D examples. For most of these cases, a qualitative graphical analysis was used to show that the results were visually equivalent to some reference MCNP5 solution. However, for 3D problems it is preferable to use a more quantitative approach because the entire domain cannot be represented by one graphic. Quantitatively comparing the KDE integral-track mesh tally to other KDE mesh tallies is simple because they all approximate the neutron flux at the mesh nodes. Unfortunately, things get more complicated when comparing these nodal-based results to the cell-averaged results of a conventional mesh tally. A similar issue arose when Griesheimer compared FET and MCNP5 results for a 2D numerical example based on a single pin cell within an infinite lattice [6]. For this example, the FET approximation was quantitatively compared to MCNP5 by first averaging the functional expansion over each mesh cell, and then by computing the relative difference between the two sets of results.

In this chapter, quantitative comparisons between the KDE integral-track mesh tally and other mesh tally implementations are explored. First, the notation and conventions that will be used are introduced in Section 3.1. Then, a series of verification test cases representing some simple fixed-source problems are described in Section 3.2. In Section 3.3 these test cases are used to perform a quantitative comparison between KDE integral-track and KDE sub-track mesh tallies. Sections 3.4 and 3.5 also use these test cases to perform a quantitative comparison between KDE integral-track and conventional mesh tallies. However, Section 3.4 uses a more

direct approach, whereas Section 3.5 relies on a data transfer method for converting the nodal-based results into cell-averaged values. Finally, Section 3.6 demonstrates the usage of the KDE integral-track mesh tally for a more complicated fixed-source problem.

3.1 Notation and Conventions

The node and cell indexing conventions used in this chapter to describe a quantitative tally comparison on 2D and 3D structured Cartesian meshes can be seen in Figure 3.1. For the 2D case, the node at the bottom-left of cell (i, j) is defined as node (i, j). Similarly, node (i, j, k) is defined as the node at the front-bottom-left of cell (i, j, k) for the 3D case. An equivalent indexing convention can also be defined for unstructured meshes. Based on these conventions, $\hat{\varphi}_{ijkg}$ will be used to represent either a KDE sub-track or KDE integral-track mesh tally result for node (i, j, k). To represent an MCNP5 or TET track mesh tally result, $\bar{\varphi}_{ijkg}$ will be used for cell (i, j, k). The subscript g in both of these cases refers to the energy group being considered, if applicable. Note that if a data transfer method is used to convert from node-to-cell or cell-to-node results, then the original accent is replaced by a tilde for all mesh tally types: $\tilde{\varphi}_{ijkg}$.

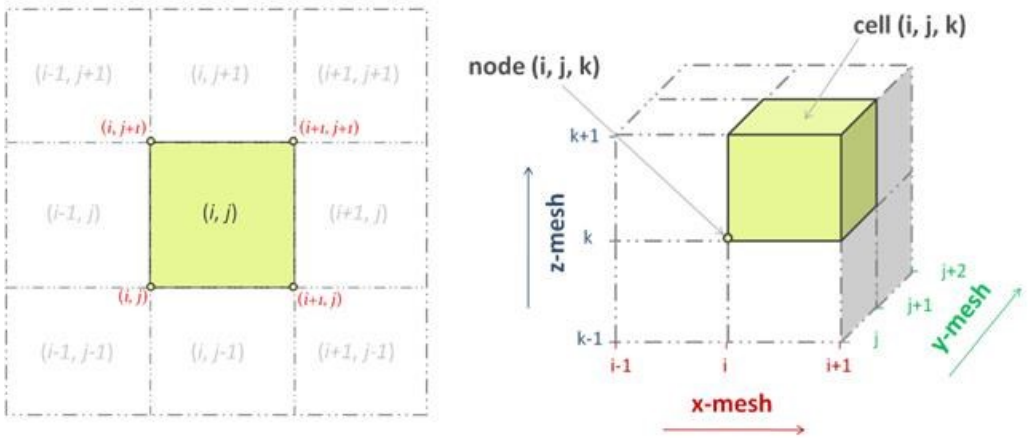

Figure 3.1 Mesh indexing conventions for 2D and 3D Cartesian representations.

To quantitatively compare mesh tally results using the definitions just described, first an error metric must be chosen that can determine the extent of the difference between them. In the statistical literature, the most widely used method for measuring the global accuracy of a density estimator is the MISE defined by Eq. 2.7 [5]. However, the MISE is not necessarily the best option for quantitative mesh tally comparisons for a couple of reasons. First, it does not take the variance of the mesh tally being used as the reference solution into consideration. This requires that the reference solution be obtained with minimal variance so that it can be treated as the exact solution. Second, it may take substantial computing resources to approximate the expected values required by the MISE, in addition to the resources needed for obtaining the solutions. Therefore, it was decided to use relative discrepancy as the error metric instead. This relative discrepancy can be computed *a posteriori* once all of the tally data has been obtained, and it requires minimal computational resources. As an example, one way to compute the relative discrepancy δ_{ijk} between a KDE integral-track and MCNP5 or TET track mesh tally result is:

$$\delta_{ijk} = \frac{\widetilde{\varphi}_{ijk} - \overline{\varphi}_{ijk}}{\overline{\varphi}_{ijk}}, \tag{3.1}$$

where $\tilde{\varphi}_{ijk}$ represents the KDE integral-track result at node (i, j, k) that was converted into a cell-averaged value using some node-to-cell data transfer method. Another way is:

$$\delta_{ijk} = \frac{\hat{\varphi}_{ijk} - \widetilde{\varphi}_{ijk}}{\widetilde{\varphi}_{ijk}}, \tag{3.2}$$

where $\tilde{\varphi}_{ijk}$ now represents a nodal-based value for the MCNP5 or TET track result that was converted using some cell-to-node data transfer method.

3.2 Verification Test Cases

The notation and conventions introduced in the previous section were used as the basis for quantitatively comparing the KDE integral-track mesh tally to other mesh tally implementations for a series of five simple fixed-source problems. These five test cases were named Uniform Flux, Gradient Flux, Reflecting Boundaries, Material Discontinuity, and Uniform Volume Source. Each of these names reflect the specific feature of a fixed-source problem that was the primary focus of the test. Note that all five test cases were constructed so that the bounding surfaces of the mesh tally region(s) of interest were defined at a distance of at least one bandwidth from any external geometrical boundary. This was needed to avoid the boundary bias effect that was introduced in Chapter 2. A brief description of the geometry, materials, and source definition for each of the five test cases follows.

3.2.1 Uniform Flux

The Uniform Flux test case was designed to test the KDE integral-track mesh tally against a known uniform flux distribution. Figure 3.2 shows the geometry of this test case, which can be described as a rectangular prism with vacuum boundaries that is void of all materials. The dimensions of this rectangular prism are [-0.5, 5.5] cm x [-1, 1] cm x [-1, 1] cm, with the mesh tally region of interest defined as a smaller rectangular prism whose dimensions are [0, 5] cm x [-0.5, 0.5] cm x [-0.5, 0.5] cm. Note also that a 1 MeV planar surface source, uniformly distributed in both the y and z dimensions, is located at x = -0.5 cm. This surface source emits neutrons parallel to the x-axis. Since there are no materials in the prism, all of these neutrons travel exactly the same distance before escaping. Therefore, using the conventional

track length tally defined by Eq. 1.8, it can be shown that the exact solution is 0.25 neutrons per cm^2 per particle history at all points in the domain. This solution is equivalent to the reciprocal of the surface area of the source.

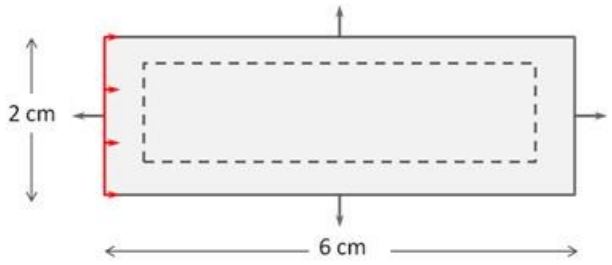

Figure 3.2 Geometrical cross-section of the Uniform Flux test case, with the dashed line representing the mesh tally region of interest.

3.2.2 Gradient Flux

The Gradient Flux test case was designed to test the KDE integral-track mesh tally against a gradient flux distribution. It is identical to the Uniform Flux test case with two notable exceptions. First, instead of containing no materials, the rectangular prism is now filled with water. In addition, the mono-directional planar surface source now emits 0.5 MeV neutrons instead of 1.0 MeV neutrons. This reduction in neutron energy was needed so that there was a sufficient gradient present within the results across the entire geometrical domain.

3.2.3 Reflecting Boundaries

The Reflecting Boundaries test case was designed to test the KDE integral-track mesh tally against a transport problem with reflecting boundaries. It is identical to the Gradient Flux test case, with the exception of the boundary condition that was used. One consequence of using reflecting boundaries is that this significantly increases the time it takes for neutrons to be

removed from the system in a highly scattering medium such as water. To minimize this increase in computing time, the energy of the neutrons emitted by the mono-directional planar surface source was reduced from 0.5 MeV to 0.05 MeV.

3.2.4 Material Discontinuity

The Material Discontinuity test case was designed to test the KDE integral-track mesh tally against a transport problem with a material discontinuity. Figure 3.3 shows the geometry of this test case, which can be described as a rectangular prism with reflecting boundaries that is split into equal water (left) and steel (right) regions. Note that the only difference between this test case and Reflecting Boundaries is the inclusion of the steel region. The composition of this steel region is a mixture containing 74% Iron (Fe), 18% chromium (Cr), and 8% nickel (Ni).

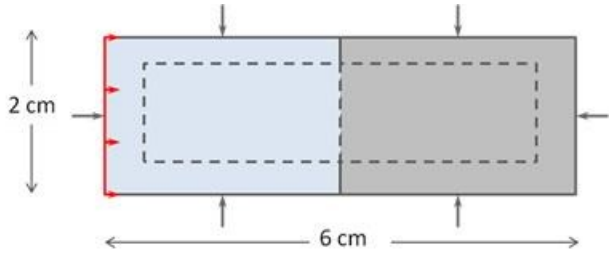

Figure 3.3 Geometrical cross-section of the Material Discontinuity test case, with the dashed lines representing the two mesh tally regions of interest.

3.2.5 Uniform Volume Source

The Uniform Volume Source test case was designed to test the KDE integral-track mesh tally against a neutron source that was uniformly distributed in volume in a highly absorbing medium. Figure 3.4 shows the geometry of this test case, which can be described as a sphere with vacuum boundaries that is filled with Boron-10. The sphere is centered at (0, 0, 0) and has

a radius of 0.12 cm, with the mesh tally region of interest defined as a smaller sphere of radius 0.10 cm. Whereas the other test cases all used planar surface sources, this test case was based on a volumetric neutron source. To determine where each neutron was born, first the radius was sampled using a power law (i.e., $p(r) \sim r^2$), and then its starting location was sampled uniformly on the surface defined by that radius. Note that the neutron energy was chosen to be 10 eV so that on average each one only experienced about one collision event before it was terminated.

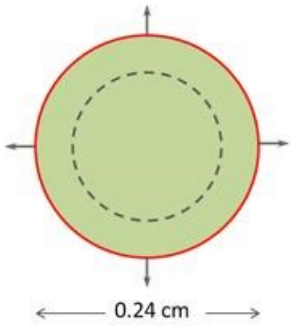

Figure 3.4 Geometrical cross-section at the center of the Uniform Volume Source test case, with the dashed line representing the mesh tally region of interest.

3.3 KDE Integral-track vs. KDE Sub-track Mesh Tallies

The five test cases described in the previous section were used to quantitatively compare KDE integral-track and KDE sub-track mesh tallies based on their corresponding DAGTally implementations. Because these tallies both approximate the neutron flux on the mesh nodes, the relative discrepancy between their results can be computed directly. Note that all tally results presented in this section were obtained using DAG-MCNP5 with native MCNP5 geometry and the Epanechnikov kernel. Recall from Chapter 2 that the KDE integral-track estimator was derived as the limiting case of the KDE sub-track estimator. Therefore, it is expected that both

KDE mesh tallies will always produce identical tally values and errors as the number of sub-tracks approaches infinity. This should be true even if the resulting neutron flux distributions are incorrect – as long as the same bandwidth vector and particle histories are used.

To verify that the KDE sub-track mesh tally produces the same results as the KDE integral-track mesh tally, a convergence analysis based on the number of sub-tracks was performed. A description of the input data needed to obtain these results can be found in Table 3.1. Note that the bandwidth vector for the first four cases was chosen to be roughly equivalent to the size of a mesh cell, whereas for the last case it was computed using the optimal bandwidth formula defined by Eq. 2.2 (with the collision sites as the set of observations).

Table 3.1 Input data used to obtain results for the five test cases considered in the KDE integral-track versus KDE sub-track mesh tally comparison.

Test Case	***N***	**Bandwidth Vector** (h_x, h_y, h_z)	**Mesh Type**	**Mesh Cells**	**Mesh Nodes**
Uniform Flux	1E+05	(0.1042, 0.0625, 0.0625)	Structured, Rectangular Prism	1536 (24 x 8 x 8)	2025
Gradient Flux	1E+05	(0.1042, 0.0833, 0.0833)	Structured, Rectangular Prism	864 (24 x 6 x 6)	1225
Reflecting Boundaries	1E+05	(0.1042, 0.0833, 0.0833)	Structured, Rectangular Prism	864 (24 x 6 x 6)	1225
Material Discontinuity	1E+05	(0.1042, 0.0833, 0.0833)	Structured, Rectangular Prism	864 (24 x 6 x 6)	1225
Uniform Volume Source	1E+06	(0.00722668, 0.00721741, 0.00722104)	Unstructured, Conformal Sphere	1010	246

An important clarification on the bandwidth vectors used in this convergence analysis needs to be made before the results can be presented. The values shown in Table 3.1 are not the ones typically used with the KDE sub-track mesh tally. Usually an optimal bandwidth is computed using Eq. 2.2 based on the set of pseudo-collisions extracted from all of the sub-tracks [14]. For the purposes of this analysis, however, the bandwidth vector was kept constant for each test case to isolate the effect that the number of sub-tracks had on the results. Using a variable bandwidth could introduce more fluctuation in the results for the KDE sub-track mesh tally, which would be inconsistent with the fixed bandwidth that was used with the KDE integral-track mesh tally. To quantify the overall difference between the results of these two KDE mesh tallies, the mean relative discrepancy was used:

$$\bar{\delta} = \frac{1}{M}\sum_{m=1}^{M}\left|\frac{T_n^m - T_\infty^m}{T_\infty^m}\right|, \tag{3.3}$$

where M is the total number of mesh nodes, and T_n^m and T_∞^m represent the values obtained for the m^{th} mesh node using the KDE sub-track and KDE integral-track estimators respectively.

The results of the convergence analysis performed on the KDE sub-track mesh tally are summarized in Figure 3.5 and Figure 3.6, which both plot the mean relative discrepancy as a function of the number of sub-tracks. Figure 3.5 represents the discrepancy in the tally values $\hat{\varphi}_{ijk}$, whereas Figure 3.6 represents the discrepancy in their corresponding tally errors. These tally errors refer to the error metric reported by DAGTally, which is formally known as the relative standard error. The relative standard error indicates the amount of variance present in the results, and is computed for each mesh cell by dividing the standard error by the tally value.

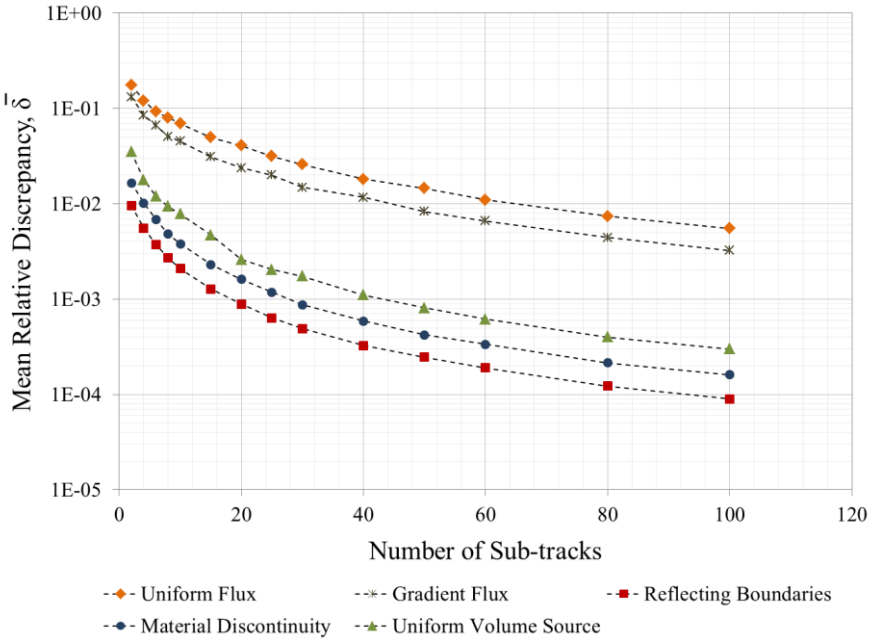

Figure 3.5 Discrepancy between the tally values computed using KDE integral-track and KDE sub-track mesh tallies versus the number of sub-tracks.

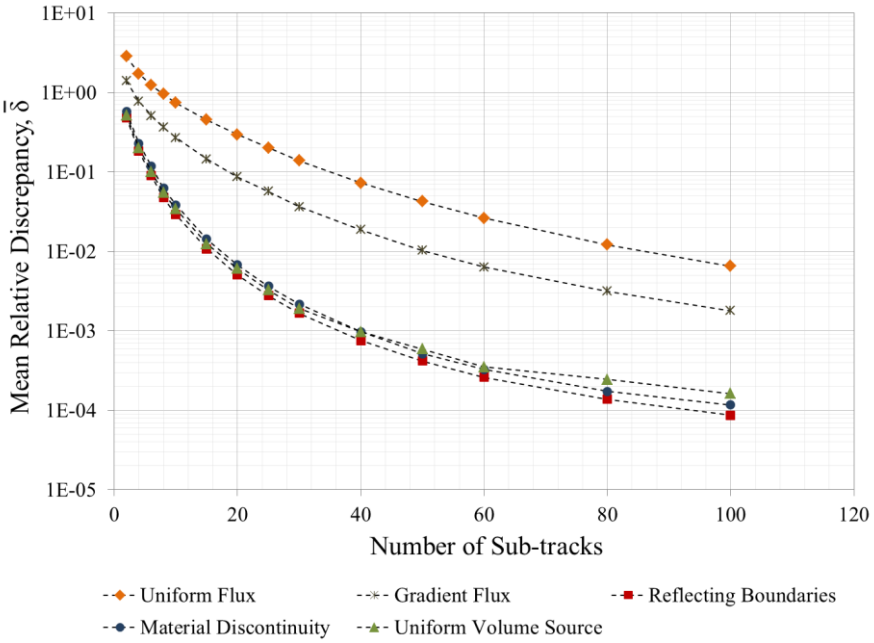

Figure 3.6 Discrepancy between the tally errors computed using KDE integral-track and KDE sub-track mesh tallies versus the number of sub-tracks.

For all five of the test cases considered, both Figure 3.5 and Figure 3.6 show clear convergence to the KDE integral-track mesh tally results as the number of sub-tracks used with the KDE sub-track mesh tally increases. The only interesting difference between the five test cases is that the magnitude of the initial discrepancy based on two sub-tracks seems to be dependent on the number of collisions that occurred. For example, the Uniform Flux test case has the worst initial discrepancy because no collisions can occur within void regions, so fewer tracks are tallied per particle history. This example is equivalent to using a poor collision density with a conventional collision-based mesh tally. In contrast, the initial discrepancy for the Reflecting Boundaries test case is much lower because neutrons can only escape via absorption and it is a highly scattering medium. Overall, however, these results show that for the same bandwidth vector the KDE integral-track estimator will generally be more accurate than the KDE sub-track estimator.

3.4 KDE Integral-track vs. Conventional Mesh Tallies: Direct Method

Results from the previous section showed that the KDE integral-track mesh tally is equivalent to a KDE sub-track mesh tally based on an infinite number of sub-tracks. While computing relative discrepancies for these KDE to KDE comparisons is straightforward, this is not the case when comparing any KDE mesh tally to the more conventional and widely-used MCNP5 mesh tally. Since KDE integral-track and MCNP5 mesh tallies both use full particle tracks to approximate the neutron flux, this common set of observations should make MCNP5 a good tool for verifying the accuracy of the KDE integral-track implementation – especially for test cases where analytical solutions cannot be defined. Unfortunately, even if both KDE

integral-track and MCNP5 mesh tallies use an identical set of particle histories to approximate the neutron flux, it can still be challenging to prove that their respective nodal-based and cell-averaged results are quantitatively equivalent [17].

One technique for producing a fair quantitative comparison between KDE integral-track and MCNP5 mesh tallies is to first use a data transfer method to convert the nodal-based KDE results into equivalent cell-averaged values (or vice versa with the MCNP5 results). This data transfer approach, which is useful for comparing tally results on identical mesh structures, will be introduced in Section 3.5. However, if the mesh structure can be manipulated so that the centers of the MCNP5 mesh cells align with the KDE mesh nodes, then it is possible to use a more direct approach. The theory and methodology of using this direct method for quantitatively comparing KDE integral-track and MCNP5 mesh tallies is introduced in this section.

3.4.1 Theory

Recall from Chapter 1 that the conventional mesh tally implemented in MCNP5 is based on a histogram, whereas the KDE mesh tallies all use the kernel density estimator. Both of these estimators use different approaches to approximate some unknown probability density function $f(x)$. Given a finite set of observations $\{X_i : i = 1 \text{ to } N\}$ randomly sampled from $f(x)$, the histogram approach requires that its entire domain first be split up into a series of regions called bins. Then, each observation is assigned to one and only one bin. Denoting n as the number of observations that fall within a single bin of width b, the histogram estimator is defined by:

$$\hat{f}(x) = \frac{1}{Nb} n, \tag{3.4}$$

where x lies within the same bin as the n observations. For the purposes of this dissertation, it is assumed that x is always assigned to the mid-point of the bin, and that no observations fall precisely on the boundaries of those bins. In contrast to the histogram approach, the kernel density estimator does not need to define any bin boundaries. Therefore, another approximation for $f(x)$ can be obtained using the same set of N observations with the following formula:

$$\hat{f}(x) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x - X_i}{h}\right), \tag{3.5}$$

where h is called the bandwidth or smoothing parameter, and $K(u)$ is some kernel function that determines the weight of each of the observations with respect to x. Note that Eq. 3.5 is the 1D version of the multivariate kernel density estimator described in Chapter 2.

While the histogram and kernel density estimator use different methods to approximate $f(x)$, it is possible to prove that these two approximations will be identical under appropriate conditions. Suppose an evaluation of $f(x)$ is needed at some arbitrary value x that lies at the mid-point of one of the histogram bins. This bin contains n observations and has a width of $b = 2h$, where h is the bandwidth used with the kernel density estimator for approximating $f(x)$. Now, assume that the uniform kernel defined in Table 2.1 was used to obtain this approximation. Since the uniform kernel only produces non-zero contributions on the interval $[x - h, x + h]$, its domain is exactly the same as the histogram bin. Therefore, only the n observations that fall within the histogram bin will contribute to the approximation at the mid-point x. In terms of Eq. 3.5, this means that:

$$\hat{f}(x) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x - X_i}{h}\right) = \frac{1}{Nh}\sum_{i=1}^{N} \frac{1}{2} = \frac{1}{N2h} n. \tag{3.6}$$

If the *2h* term in the denominator is replaced with *b*, then Eq. 3.6 produces exactly the same approximation for $f(x)$ as Eq. 3.4. This will be true as long as no observations ever fall on the boundaries of the histogram bin [5].

Based on a similar approach to the previous 1D example, it is possible to show that the KDE integral-track mesh tally is capable of producing identical neutron flux distributions to the MCNP5 mesh tally. This approach, called the direct method for quantitative mesh tally comparisons, will now be described. The first step of the direct method is to create an MCNP5 input mesh such that the centers of its cells align with the nodes of the KDE input mesh. An example of how these meshes should be aligned for a 2D problem is shown in Figure 3.7. Note that it is important that both of these input meshes be structured, and that their individual mesh cells have the same dimensions.

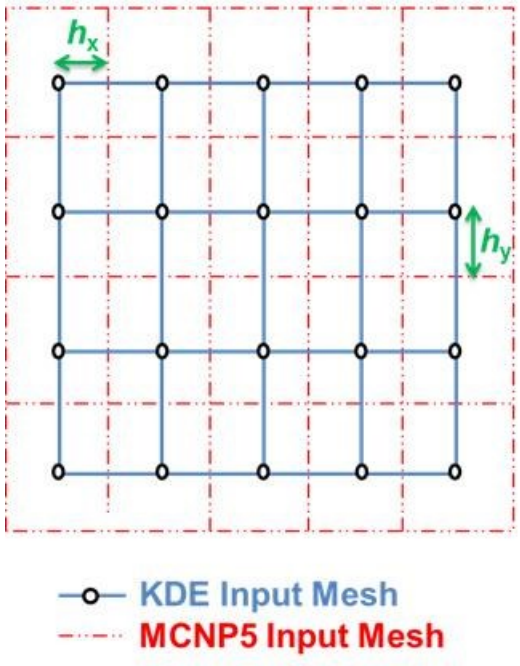

Figure 3.7 2D Graphical representation of the input mesh alignment needed to compare KDE integral-track and MCNP5 mesh tally results using the direct method.

The next step of the direct method is to define values for the bandwidth vector so that the neighborhood regions based on the uniform kernel around each mesh node are identical to the size of the mesh cells. To achieve this, the value used for the bandwidth in each dimension should be half the length of the mesh cell in that dimension, as shown in Figure 3.7 for h_x and h_y. The final step of the direct method is to make sure that both KDE integral-track and MCNP5 mesh tallies are guaranteed to use the same set of particle histories. This can be done by either putting both mesh tally definitions into one input file, or by using the same starting seed values with the same random number generator in separate input files. Ensuring that the same set of particle histories are used means that the only difference between the two mesh tallies is how they calculate their respective results.

3.4.2 Methodology and Results

The direct method for comparing KDE integral-track to MCNP5 mesh tallies that was just described was used on the first four test cases from Section 3.2: Uniform Flux, Gradient Flux, Reflecting Boundaries, and Material Discontinuity.[3] Note that all of these test cases used the same KDE input mesh, which was based on a much finer structure than the ones defined in Table 3.1 for the KDE integral-track to KDE sub-track comparison. This KDE input mesh had a total of 6912 mesh cells (48 x 12 x 12) and 8281 mesh nodes, resulting in a bandwidth vector of (0.05208333, 0.04166667, 0.04166667). After the KDE input mesh was constructed, results for all four test cases were obtained using DAG-MCNP5 with native MCNP5 geometry, the uniform kernel, and 1E+08 particle histories. Then, reference MCNP5 mesh tally solutions were obtained

[3] Uniform Volume Source could not be analyzed using the direct method because it uses an unstructured mesh.

using the same set of particle histories, but slightly different input meshes. These MCNP5 input meshes were translated by one bandwidth in each dimension so that the KDE mesh nodes would be aligned with the centers of the MCNP5 mesh cells.

According to the theory of the direct method, the KDE integral-track result $\hat{\varphi}_{ijk}$ at mesh node (i, j, k) should be identical to the MCNP5 result $\bar{\varphi}_{ijk}$ at the center of mesh cell (i, j, k). To show that this was indeed the case, the relative discrepancy between $\bar{\varphi}_{ijk}$ and $\hat{\varphi}_{ijk}$ for all of the mesh nodes and cells were computed using a similar expression to Eq. 3.1. These relative discrepancies are summarized for all four test cases in Table 3.2. The values on the left refer to the discrepancy in the tally values, whereas the values on the right refer to the discrepancy in their corresponding tally errors. In general, both the tally values and errors are identical to six significant figures. Since MCNP5 mesh tallies only report results up to six significant figures, these comparisons based on the direct method clearly show that the KDE integral-track mesh tally is capable of producing exactly the same flux distributions as the MCNP5 mesh tally.

Table 3.2 Relative discrepancies δ_{ijk} between KDE integral-track and MCNP5 mesh tally results for the four test cases compared using the direct method.

Test Case	TALLY VALUES			TALLY ERRORS		
	Min δ_{ijk}	**Max δ_{ijk}**	**Avg δ_{ijk}**	**Min δ_{ijk}**	**Max δ_{ijk}**	**Avg δ_{ijk}**
Uniform Flux	0.00E+00	6.42E-06	1.13E-06	1.71E-08	2.82E-06	1.09E-06
Gradient Flux	6.65E-10	4.30E-05	1.12E-06	0.00E+00	1.86E-05	8.52E-07
Reflecting Boundaries	3.94E-10	2.58E-06	9.98E-07	0.00E+00	1.01E-06	4.68E-07
Material Discontinuity	0.00E+00	5.17E-06	1.11E-06	0.00E+00	5.00E-06	8.81E-07

While most of the results were identical for both KDE integral-track and MCNP5 mesh tallies, there were 37 mesh nodes in the Gradient Flux test case that reported tally values that did not agree to six significant figures. Of these, only 11 also reported tally errors that were not identical to MCNP5. It is not known precisely why the results for these mesh nodes are slightly different, but these particular mesh nodes had the same z-coordinate and only two different y-coordinates. In theory, the only way that these results can differ is if one or more of the particle tracks that are tallied are coincident with a mesh boundary. Recall from Section 3.4 that for the direct method to be valid it is assumed that all observations fell within the bin boundaries. If this was not the case, then it is possible that the KDE results based on the uniform kernel could be slightly overestimated because it always includes non-zero contributions for these boundary observations. However, in practice these errors are most likely the result of rounding errors. MCNP5 mesh tallies use the particle tracks directly, whereas KDE integral-track mesh tallies perform both kernel evaluations and quadrature calculations on those particle tracks before tallying them. The Gradient Flux test case had a lower collision density than most of the other test cases, which may have made it more susceptible to rounding errors.

The equivalence shown in Table 3.2 between KDE integral-track and MCNP5 mesh tallies only occurred because the direct method was used for quantitatively comparing the results. If, however, a different kernel function was used, then the outcome of such a quantitative mesh tally comparison could potentially change. Recall from Chapter 2 that the accuracy of the KDE integral-track mesh tally can be affected by both the bandwidth and/or kernel function. Though this will be explored in greater detail in Chapter 4, the same procedure described at the beginning of this section was also used with the other three kernels defined in Table 2.1. The resulting

mean relative discrepancies between KDE integral-track and MCNP5 mesh tally results are plotted in Figure 3.8 as a function of the smoothness of the kernel. For all four test cases analyzed using the direct method, as the smoothness of the kernel function increased, so did the mean relative discrepancy for both the tally values and their corresponding tally errors. These results show that when the uniform kernel is not used with the kernel density estimator, then the neutron flux distribution that is obtained will no longer be identical to MCNP5.

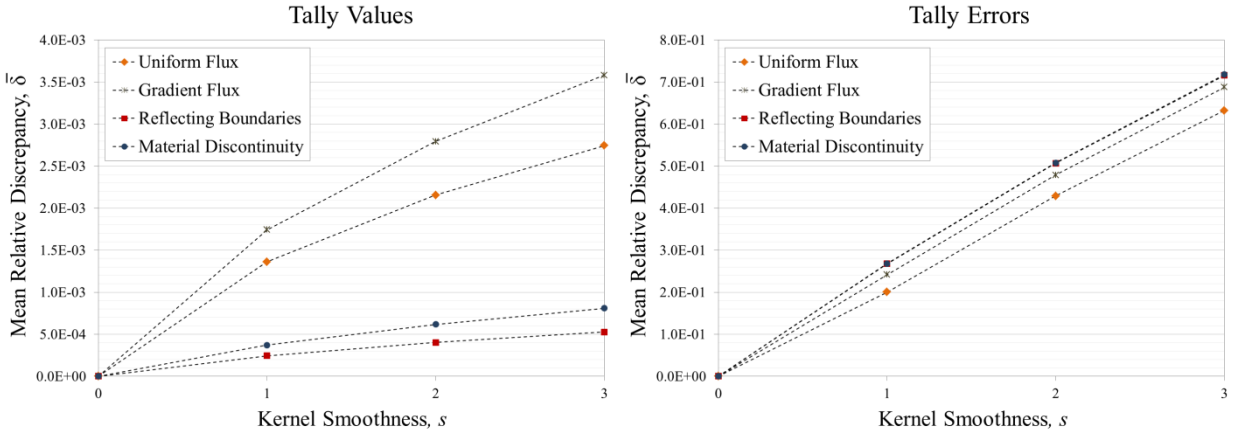

Figure 3.8 Discrepancy between nodal-based KDE integral-track and MCNP5 mesh tally results compared using the direct method with different kernel functions.

Though there is a noticeable difference between using the uniform kernel and the other kernel functions from Table 2.1, it is not clear if this difference is due to an improvement in bias or an increase in variance. To see how both bias and variance of the results are affected by using different kernels, ratios of the KDE integral-track to MCNP5 approximations were computed for the Gradient Flux test case. A summary of these ratios is shown in Table 3.3. Note that there is little variation in the tally values, but a significant variation in the tally errors. The same behavior was also observed for the other test cases, with average ratios that were similar to the

ones shown in Table 3.3. These results indicate that the mean relative discrepancy between KDE integral-track and MCNP5 mesh tally results is most likely dominated by the increased variance – which is not surprising considering that fine meshes were used and the bandwidth was kept fixed. Thus, while the direct method can show that KDE integral-track and MCNP5 mesh tally results are capable of being equivalent, it is important to remember that using different kernel functions may or may not produce significantly less biased results compared to MCNP5.

Table 3.3 Ratio of KDE integral-track to MCNP5 mesh tally results for the Gradient Flux test case, using the direct method with different kernel functions.

Kernel	TALLY VALUES			TALLY ERRORS		
	Min Ratio	**Max Ratio**	**Avg Ratio**	**Min Ratio**	**Max Ratio**	**Avg Ratio**
Uniform ($s = 0$)	0.999995	1.000043	1.000000	0.999981	1.000002	1.000000
Epanechnikov ($s = 1$)	0.987166	1.011114	1.000035	1.222900	1.251106	1.241744
Biweight ($s = 2$)	0.980305	1.017613	1.000056	1.455845	1.495598	1.479007
Triweight ($s = 3$)	0.976281	1.021609	1.000068	1.660925	1.710620	1.687843

3.5 KDE Integral-track vs. Conventional Mesh Tallies: Data Transfer Method

The direct method studied in Section 3.4 is useful for showing that the KDE integral-track mesh tally is capable of producing identical results to the MCNP5 mesh tally under appropriate conditions. However, if the fixed-source problem is defined using an unstructured mesh, then this direct method cannot be used for quantitatively comparing the KDE integral-track mesh tally to other conventional options such as MCNP6 or TET track. Even if it was

possible to align the KDE mesh nodes with the centers of the tetrahedral mesh cells, it is impossible to choose a bandwidth so that the neighborhood region based on the uniform kernel would be equivalent to the size of these mesh cells. Therefore, in this section the theory and methodology of using the more general data transfer method for quantitative mesh tally comparisons is introduced.

3.5.1 Theory

Data transfer methods are often used in multi-physics applications to convert results from one mesh to another when different mesh structures are used for the individual physics components. While this is usually done from node-to-node or cell-to-cell, these methods can also be adapted to quantitatively compare nodal-based results to cell-averaged values. One of the simplest methods that can be used for comparing different mesh tally results is a basic cell-to-node technique. This basic technique approximates the value at each mesh node (i, j, k) by averaging the neutron flux for all mesh cells adjacent to that node. For a structured 3D Cartesian mesh based on the notation and conventions defined in Section 3.1, this would be computed by:

$$\widetilde{\varphi}_{ijk} = \frac{1}{a_{ijk}} \sum_{l=i-1}^{i} \sum_{m=j-1}^{j} \sum_{n=k-1}^{k} \overline{\varphi}_{lmn}, \tag{3.7}$$

where a_{ijk} is the number of cells that was used in the approximation. If Eq. 3.7 is used to estimate the nodal-based flux at an internal mesh node, then eight mesh cells are included. However, if the same equation is used for a boundary node, then only up to four mesh cells are included. This can introduce significant errors into the relative discrepancy calculations, especially if those boundary nodes are at the corners of the mesh.

As an alternative to the basic cell-to-node technique, a more effective method is to use a node-to-cell technique that uses finite element theory. A finite element can be thought of as a geometrical construct that is uniquely defined by its shape and a fixed number of nodes. Some common finite element representations for both hexahedral and tetrahedral geometries are shown in Figure 3.9. Suppose that the flux $\hat{\varphi}_k$ is known at all n nodes of a mesh cell that can also be defined as a finite element. Given these values, the flux at any other point within the mesh cell can be approximated by the interpolation function [23]:

$$\hat{\varphi}(\xi_1, \xi_2, \xi_3) = \sum_{k=1}^{n} \hat{\varphi}_k N_k(\xi_1, \xi_2, \xi_3), \tag{3.8}$$

where N_k is the basis function at the k^{th} node of the element, expressed in terms of its natural coordinates (ξ_1, ξ_2, ξ_3). Natural coordinates represent a normalized set of coordinates that describe the location of any point within the finite element. Therefore, Eq. 3.8 can be used to interpolate the $\hat{\varphi}_k$ values of the mesh cell to the p cubature points of a Gaussian cubature rule. This Gaussian cubature rule can then provide an approximation for the cell-averaged flux, via a change of variables to the natural coordinates:

$$\bar{\varphi} = \frac{\int_V \varphi(x, y, z) dV}{\int_V dV} \approx \frac{1}{V} \sum_{l=1}^{p} w^l j^l \left(\sum_{k=1}^{n} \hat{\varphi}_k N_k^l \right), \tag{3.9}$$

where the superscript l refers to the natural coordinates of the l^{th} cubature point, w is the cubature weight, j is the determinant of the Jacobian matrix, and V is the volume of the mesh cell. For a more detailed example of how this node-to-cell data transfer method works, see APPENDIX B.

Once a cell-averaged approximation $\widetilde{\varphi}_{ijk}$ to a nodal-based KDE integral-track mesh tally result has been obtained for mesh cell (i, j, k) using Eq. 3.9, it can then be compared to the equivalent MCNP5 or TET track mesh tally result $\overline{\varphi}_{ijk}$ using the relative discrepancy defined by Eq. 3.1.

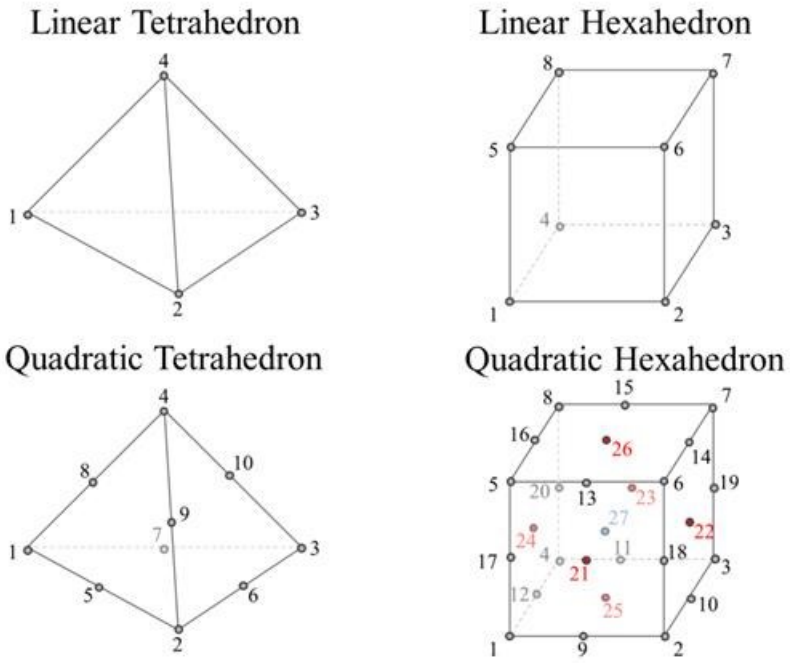

Figure 3.9 Some common 3D finite element representations.

If a data transfer method such as Eq. 3.7 or 3.9 is used to compare KDE integral-track and conventional mesh tallies, then there are a couple of key issues that need to be considered to ensure a fair comparison. The first issue is which data transfer method to use for converting the nodal-based results into cell-averaged values (or vice versa). For geometries with a lot of curvature, simple cell-to-node or node-to-cell methods may not be sufficient. As an example of how the choice of data transfer method can affect the outcome of a quantitative comparison, three different options were used to compare KDE integral-track and TET track mesh tallies for

the Uniform Volume Source test case described in Section 3.2. All results obtained for both of these tallies were based on 1E+06 particle histories and a fixed spherical mesh consisting of 1010 tetrahedral cells. The only thing that differed was the data transfer method used to compare the results. Figure 3.10 shows the distribution of relative discrepancies that were computed for each of the three methods considered in this example. Note that the first two methods used Eq. 3.9 with quadratic and linear finite elements respectively to convert the KDE integral-track mesh tally results into cell-averaged values. The third method used a simple cell-averaging method that was similar to Eq. 3.7 for converting the TET track results into nodal-based values.

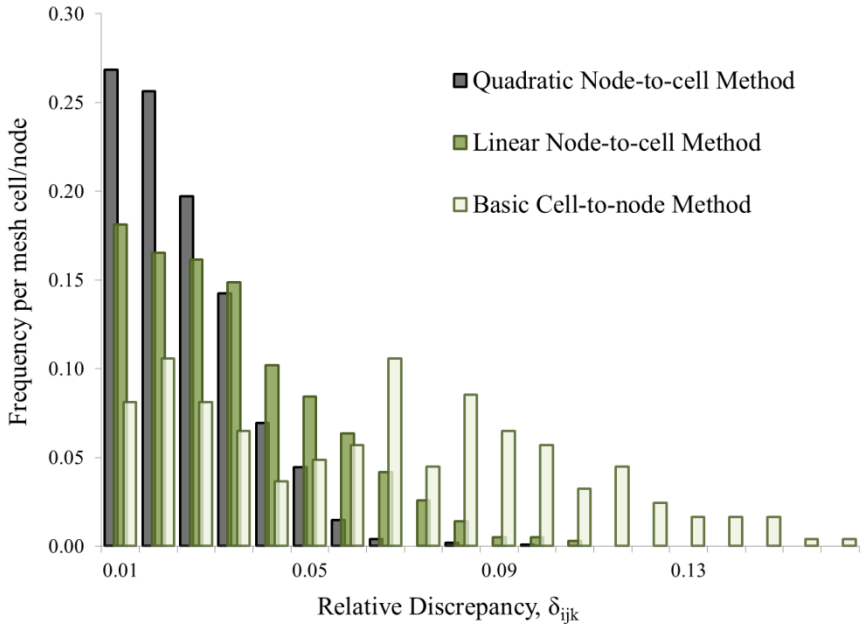

Figure 3.10 Distribution of the relative discrepancies between KDE integral-track and TET track results obtained using different data transfer methods on a spherical input mesh.

As expected, due to the curvature of the flux within the sphere, using the quadratic node-to-cell method produced more mesh cells with lower discrepancies on average than the linear node-to-cell method. However, both of these cases clearly performed better than the basic cell-

to-node method. This basic cell-to-node method produced a bi-modal distribution of relative discrepancies, with the comparison for nodes on the boundary of the mesh consistently reporting larger discrepancies than those within the sphere. If this was the only data transfer method considered for comparing KDE integral-track and TET track mesh tally results, then it would have been incorrectly concluded that these two tallies do not produce equivalent results for this particular problem. Therefore, it is essential to choose a data transfer method that is appropriate for the geometry being used to ensure a fair comparison.

Even if the nodal-based KDE integral-track mesh tally results were converted into cell-averaged values using an appropriate data transfer method, there is still no guarantee that these approximated values will compare favorably to equivalent results obtained using an MCNP5 or TET track mesh tally. If it is assumed that the cell-averaged KDE results are correct, and that the original nodal-based values were computed using an optimal bandwidth, then the only other factor that can affect the outcome of the quantitative comparison is the validity of the reference solution. Since the bias and variance of results obtained using conventional mesh tallies are highly dependent on the size of the mesh cells, this means that the resolution of the mesh that is used must be sufficient enough to not only get good statistics, but also to capture the shape of the underlying flux distribution correctly. In contrast, the bias and variance of results obtained using KDE mesh tallies primarily depend on the bandwidth, and are therefore not affected by the size of the mesh cells. This subtle difference can lead to incorrect conclusions about the KDE integral-track tally if a poor choice of mesh resolution is used for the conventional mesh tally. As an example, consider a non-void rectangular prism with a uniformly distributed surface source that emits neutrons parallel to the x-axis into the prism. The relative discrepancies

between cell-averaged KDE integral-track and MCNP5 mesh tally results for this example are shown in Figure 3.11 at regular intervals throughout the prism. Note that the cell-averaged values were obtained from the nodal-based KDE tally results using Eq. 3.9 with linear elements.

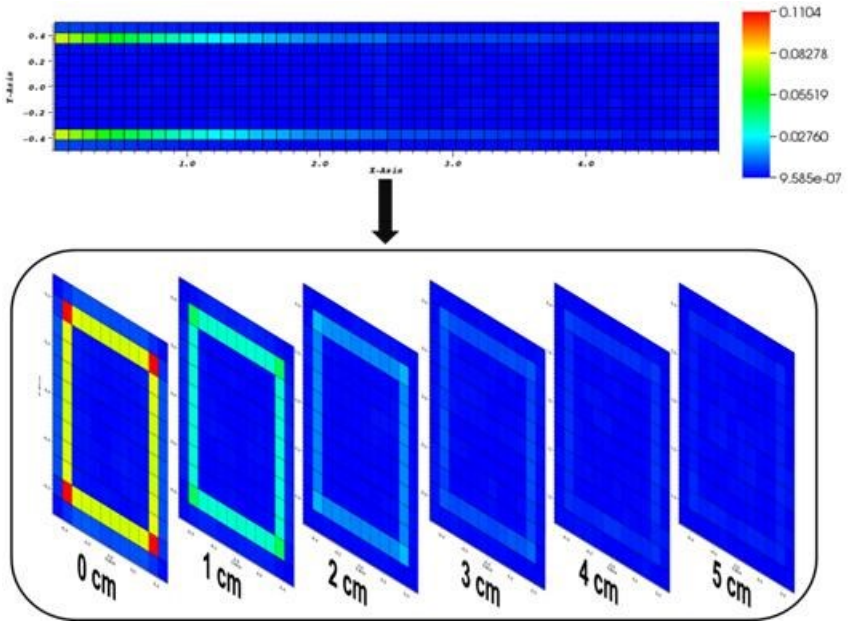

Figure 3.11 Relative discrepancy between cell-averaged KDE integral-track and MCNP5 mesh tally results for an example transport problem using a coarse mesh.

Figure 3.11 indicates that there is fairly good agreement between cell-averaged KDE integral-track and MCNP5 mesh tally results throughout most of the domain. The only exception seems to occur for mesh cells that lie parallel to the edges of the surface source. Neutron flux values reported for these mesh cells differ between the two tallies by up to a maximum of 11%. The reason for these larger discrepancies can be seen more clearly in Figure 3.12, which plots the flux distributions that were obtained using both KDE integral-track and MCNP5 tallies for the mesh nodes or cells along the *y*-axis closest to the center of the source. Note that for the MCNP5 results there is a noticeable difference between the coarse and fine

mesh representations near the edges of the source. This occurs because MCNP5 effectively averages the results of the fine mesh to create the results for the coarse mesh, which means that it does not correctly resolve the gradient of the flux distribution within these coarser mesh cells. In contrast, the KDE integral-track mesh tally is not dependent on the resolution of the mesh because the bandwidth remains fixed, and actual flux values at the mesh nodes are being approximated instead of computing cell-averaged values. This means that even though the mesh has been refined, the results for mesh nodes on the fine mesh that were also a part of the coarse mesh remain unchanged.

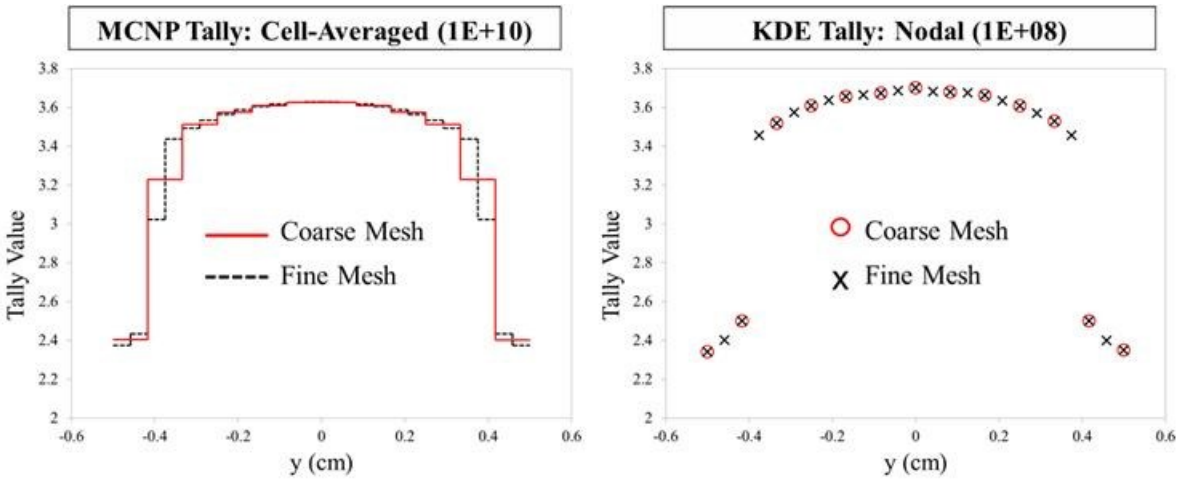

Figure 3.12 Comparison of cell-averaged MCNP5 and nodal-based KDE integral-track mesh tally results for coarse and fine mesh representations.

Now, consider what happens when the nodal-based KDE integral-track tally results are converted into their cell-averaged equivalents for both coarse and fine mesh representations. Figure 3.13 graphically compares the MCNP5 and cell-averaged KDE results that were obtained using this method for the $z = 0$ plane closest to the source. Note that the discrepancy between the two tallies that occurs in the coarser mesh near the edges of the source is noticeably reduced

simply by refining the mesh. When this finer mesh is used, both the MCNP5 and cell-averaged KDE results at least appear to be visually comparable. They are also more numerically comparable, since the maximum discrepancy is reduced from 11% to around 3% for the mesh cells in the corners closest to the source. These results show that even when a good data transfer method is used, in order to perform a fair quantitative comparison between KDE integral-track and MCNP5 mesh tallies it is also necessary to consider how well MCNP5 resolves the underlying flux distribution. Inconsistencies between KDE integral-track and MCNP5 results do not necessarily mean that the two tallies cannot be considered equivalent.

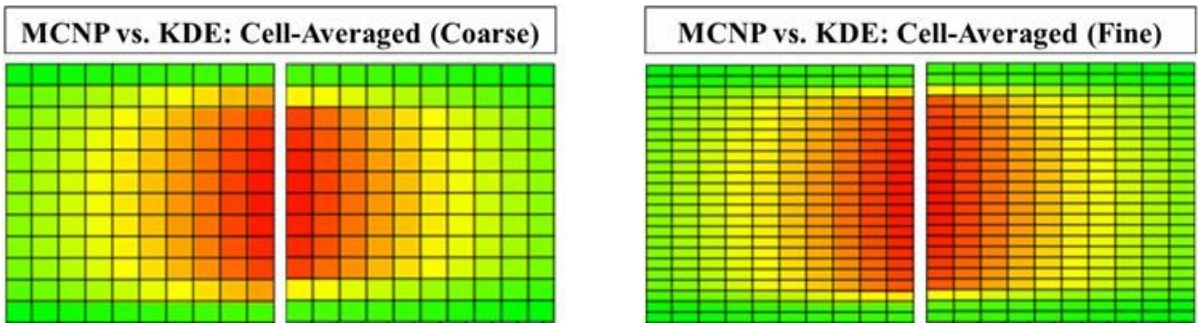

Figure 3.13 Comparison of cell-averaged MCNP5 and cell-averaged KDE integral-track mesh tally results for coarse and fine mesh representations.

3.5.2 Methodology and Results

Recall from Section 3.4 that the direct method required that the input meshes for the KDE integral-track and MCNP5 mesh tallies were structured. In contrast, the data transfer method can be used with any type of input mesh – including unstructured ones – as long as the mesh structures are identical for both mesh tallies. In this section, results from all five test cases described in Section 3.2 were compared using the node-to-cell data transfer method defined by Eq. 3.9. Note that for the first four test cases the KDE integral-track results were compared to

the MCNP5 mesh tally, whereas for the Uniform Volume Source test case they were compared to the TET track mesh tally. Since none of these quantitative comparisons were expected to produce identical results, a convergence analysis based on the number of particle histories N was performed to show that they were at least comparable. A description of the input data needed to obtain the results for all five test cases can be found in Table 3.4.

Table 3.4 Input data used to obtain results for the five test cases considered in the KDE integral-track versus conventional mesh tally comparison based on the data transfer method.

Test Case	***N***	**Bandwidth Vector (h_x, h_y, h_z)**	**Mesh Type**	**Mesh Cells**	**Mesh Nodes**
Uniform Flux	1E+03 to 1E+08	All particle histories used: (0.0521, 0.0417, 0.0417)	Linear Hexahedra	6912	8281
Gradient Flux	1E+02 1E+03 1E+04 1E+05 1E+06 1E+07 1E+08	(0.592466, 0.24703, 0.256785) (0.449338, 0.180352, 0.177625) (0.329554, 0.128115, 0.128349) (0.240632, 0.0926277, 0.092449) (0.172657, 0.0665775, 0.0665122) (0.124171, 0.0478787, 0.0478659) (0.0893531, 0.0344502, 0.0344511)	Linear Hexahedra	6912	8281
Reflecting Boundaries	1E+01 1E+02 1E+03 1E+04 1E+05 1E+06 1E+07 1E+08	(0.541427, 0.186427, 0.187013) (0.404541, 0.132818, 0.132997) (0.287487, 0.095486, 0.0959774) (0.206095, 0.0687937, 0.0688402) (0.148628, 0.0495376, 0.049546) (0.10694, 0.0356477, 0.0356532) (0.0769674, 0.025656, 0.0256555) (0.0553957, 0.018465, 0.0184648)	Linear Hexahedra	6912	8281

Test Case	N	Bandwidth Vector (h_x, h_y, h_z)	Mesh Type	Mesh Cells	Mesh Nodes
Material Discontinuity	1E+01	(0.510483, 0.210843, 0.205849)	Linear Hexahedra	6912	8281
	1E+02	(0.383036, 0.156653, 0.153628)			
	1E+03	(0.275468, 0.110795, 0.110699)			
	1E+04	(0.198643, 0.0797517, 0.0797534)			
	1E+05	(0.143026, 0.0574678, 0.0574548)			
	1E+06	(0.102903, 0.0413586, 0.0413604)			
	1E+07	(0.0740864, 0.0297638, 0.0297627)			
	1E+08	(0.0533155, 0.0214199, 0.0214197)			
Uniform Volume Source	1E+02	(0.0258763, 0.0261472, 0.0284612)	Quadratic Tetrahedra	47,039	67,331
	1E+03	(0.0200247, 0.0193893, 0.0193136)			
	1E+04	(0.0139877, 0.0139137, 0.0139136)			
	1E+05	(0.0100222, 0.0100461, 0.0100493)			
	1E+06	(0.00722668, 0.00721741, 0.00722104)			
	1E+07	(0.00519784, 0.00519823, 0.00519855)			
	1E+08	(0.00374131, 0.00374109, 0.00374135)			

The first four test cases described in Table 3.4 are based on input meshes that have the same structure as the ones used with the direct method. These four input meshes, along with the one used for the Uniform Volume Source test case, were created using a much finer resolution than the input meshes used for the KDE integral-track versus KDE sub-track comparison. This finer mesh resolution was needed so that there was no question as to whether or not the MCNP5 or TET track mesh tally results would represent valid reference solutions for the underlying flux distributions. Note also that instead of simply describing each input mesh as structured or unstructured, Table 3.4 lists the type of finite element that was used with Eq. 3.9 to obtain the

KDE integral-track mesh tally results. The first four test cases consist of mesh cells that represent linear hexahedra, which each have eight mesh nodes. The last test case is based on quadratic tetrahedra, which are mesh cells with ten mesh nodes. Quadratic tetrahedra were used instead of linear tetrahedra for the Uniform Volume Source test case to improve the accuracy of the node-to-cell conversion for nodes near the surface of the mesh where the curvature is higher.

In addition to using finer meshes, it was also necessary to choose appropriate bandwidth vectors to ensure that the KDE integral-track mesh tally produced valid solutions. For the KDE integral-track versus KDE sub-track comparison presented in Section 3.3, the bandwidth did not matter because both KDE mesh tallies suffer from the same bias and will always be comparable. For the KDE versus MCNP5 or TET track comparisons, however, this parameter plays a much bigger role in determining the comparability of the results. Recall that for the direct method the bandwidth had to be chosen so that the neighborhood region around each calculation point was equivalent to the size of the mesh cells. For the data transfer method, the bandwidth vectors that were used to obtain the KDE integral-track mesh tally results are shown in Table 3.4. Most of these bandwidth vectors were computed using the optimal bandwidth formula defined by Eq. 2.2, with the collision sites as the set of observations. Since this optimal bandwidth formula is inversely proportional to the number of observations that were used, this means that the KDE integral-track results should become more comparable to the MCNP5 or TET track results as more particle histories are added, and the corresponding bandwidth vectors get smaller. Only the Uniform Flux test case could not use Eq. 2.2 with the collision sites, since no collisions can occur in void regions. Therefore, for this case only, the bandwidth that was used was similar to the one used with the direct method.

Once all of the input files for the KDE integral-track mesh tally were set up using the data shown in Table 3.4, original nodal-based results were obtained for each test case using DAG-MCNP5 with native MCNP5 geometry and the Epanechnikov kernel. These nodal-based results were then converted into cell-averaged values using Eq. 3.9. After cell-averaged results were computed for all five test cases, they were then compared to reference solutions obtained using either MCNP5 or TET track mesh tallies with the same input meshes and 1E+10 particle histories. Note that all results were obtained using the same random seed number, which provided the same sequence of track segments as the number of particle histories was increased. The resulting mean relative discrepancies for all of the mesh cells are plotted as a function of particle histories in Figure 3.14. All five test cases considered show clear convergence to the reference solutions as more particle histories are used with the KDE integral-track mesh tally.

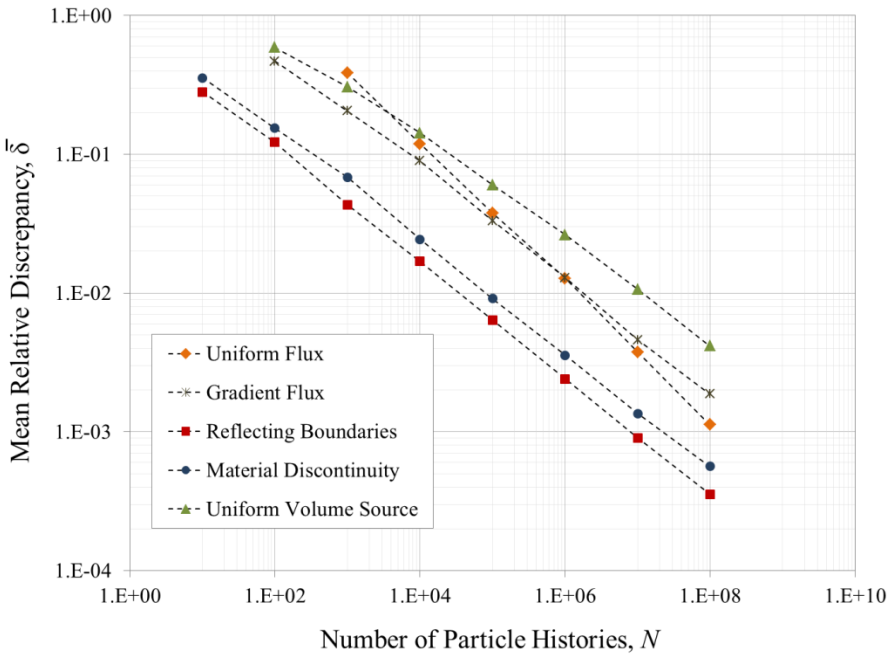

Figure 3.14 Discrepancy between the tally values computed using KDE integral-track and MCNP5 mesh tallies versus the number of particle histories.

While the mean relative discrepancy is useful for describing global convergence, it does not provide any information about the local convergence of the individual tally results. One way to observe this local convergence is to plot cumulative distribution functions for the relative discrepancies as a function of *N*. Figure 3.15 shows these cumulative distribution functions for four of the five test cases considered in the global convergence analysis.

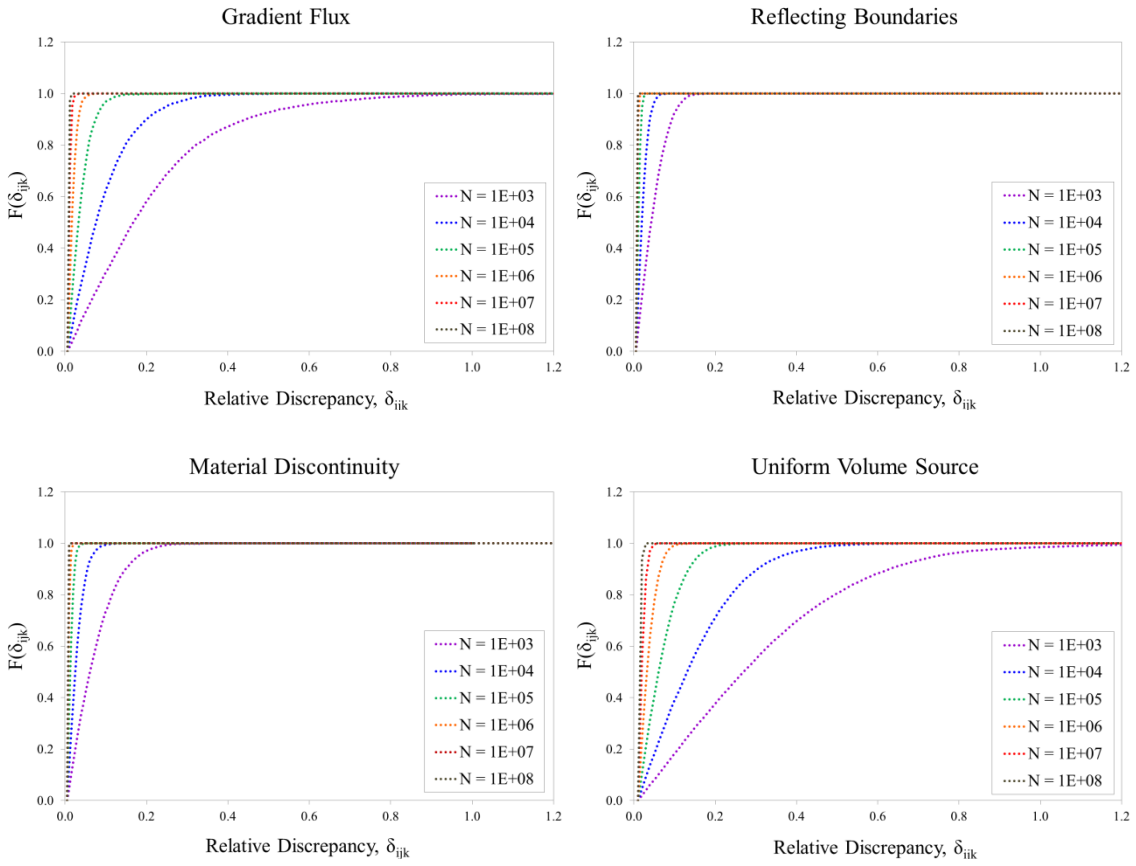

Figure 3.15 Series of cumulative distribution functions for the relative discrepancy between cell-averaged KDE integral-track and MCNP5 mesh tally results.

Although the Uniform Flux case is not included in Figure 3.15, note that the same general behavior was observed. As more particle histories are added, more and more mesh cells produce tally results that are closer to their corresponding reference solutions. These local convergence

results, along with the global convergence results shown in Figure 3.14, indicate that both KDE integral-track and MCNP5 or TET track mesh tallies are capable of producing comparable results for simple fixed-source problems.

Though both global and local convergence results support the hypothesis that the KDE integral-track mesh tally produces comparable results to conventional mesh tallies, neither of these analyses take the statistical errors of the tallies into consideration. Since the approximated neutron flux for any mesh node or cell is expected to be normally distributed, then the relative discrepancy defined by Eq. 3.1 should also be normally distributed. This means that there should be about 68%, 95% and 99.7% of mesh cells with relative discrepancies less than one, two and three standard deviations respectively. While precise values for these standard deviations are not known, it is possible to approximate them using error propagation theory. Based on the assumption that all tally results can be treated as independent random variables, the following approximation for the variance in the relative discrepancy was derived (see APPENDIX C):

$$\sigma_{\delta_{ijk}}^2 \approx \left(\frac{\sigma_{\widetilde{\varphi}_{ijk}}}{\overline{\varphi}_{ijk}} \right)^2 + \left(\frac{\widetilde{\varphi}_{ijk} \sigma_{\overline{\varphi}_{ijk}}}{\overline{\varphi}_{ijk}^2} \right)^2, \tag{3.10}$$

where $\sigma_{\widetilde{\varphi}_{ijk}}$ and $\sigma_{\overline{\varphi}_{ijk}}$ are the standard deviations of the cell-averaged KDE integral-track result $\widetilde{\varphi}_{ijk}$ and the MCNP5 or TET track result $\overline{\varphi}_{ijk}$ respectively.

Standard deviations for the relative discrepancy between KDE integral-track and MCNP5 or TET track mesh tally results were approximated using Eq. 3.10 for all five test cases. For the purposes of this analysis, note that the KDE results were based on 1E+08 particle histories,

whereas the reference results were based on 1E+10 particle histories. Therefore, since $\tilde{\varphi}_{ijk} \sim \bar{\varphi}_{ijk}$ and $\sigma_{\tilde{\varphi}_{ijk}} > \sigma_{\bar{\varphi}_{ijk}}$, the first term of Eq. 3.10 is the dominant factor. The resulting percentage of mesh cells that reported relative discrepancies less than their approximated statistical errors is summarized in Table 3.5. For four of the five test cases considered, the distribution of relative discrepancies for all of the mesh cells fell short of what was expected. Only the Reflecting Boundaries test case produced results that were even close to a normal distribution, due to the large number of collisions used in the approximation. One reason for this difference could be because the nodal-based KDE results were treated as independent random variables in order to derive Eq. 3.10. Making this assumption underestimates the statistical error, since the covariance terms for each of the mesh nodes involved in the computation of $\sigma_{\tilde{\varphi}_{ijk}}$ were omitted. If these covariance terms were included, then it should be possible to show that the relative discrepancies that were computed do in fact fall within statistical expectations.

Table 3.5 Distribution of relative discrepancies δ_{ijk} based on a comparison between KDE (1E+08) and MCNP5 (1E+10) mesh tally results.

Test Case	**% MESH CELLS WITHIN STATISTICAL ERROR**		
	$\delta_{ijk} < 1\sigma$	$\delta_{ijk} < 2\sigma$	$\delta_{ijk} < 3\sigma$
Uniform Flux	52.8%	88.9%	96.5%
Gradient Flux	50.8%	84.3%	96.5%
Reflecting Boundaries	61.6%	92.6%	99.3%
Material Discontinuity	59.3%	88.4%	97.3%
Uniform Volume Source	52.2%	84.2%	96.6%

Another way to show that the relative discrepancies between KDE integral-track and MCNP5 or TET track results are consistent with statistical expectations is to consider the Uniform Flux test case. Recall from Section 3.2 that an analytical solution exists because this test case is based on a void region with a mono-directional neutron source. Using this analytical solution as the reference for both KDE integral-track and MCNP5 mesh tallies, Table 3.6 shows the resulting percentages of mesh cells that reported relative discrepancies less than their approximated statistical errors. Note that when either nodal-based KDE or cell-averaged MCNP5 results are compared directly to the analytical solution, both sets of relative discrepancies fall within statistical expectations. It is only when the nodal-based KDE results are converted into their cell-averaged equivalents by assuming independence that the set of relative discrepancies becomes inconsistent with a normal distribution.

Table 3.6 Distribution of relative discrepancies δ_{ijk} based on a comparison to the analytical solution for the Uniform Flux test case.

Mesh Tally	***N***	**% MESH CELLS WITHIN STATISTICAL ERROR**		
		$\delta_{ijk} < 1\sigma$	$\delta_{ijk} < 2\sigma$	$\delta_{ijk} < 3\sigma$
KDE (Cell-averaged)	1E+08	50.7%	88.9%	96.5%
KDE (Nodal-based)	1E+08	63.3%	94.7%	100%
MCNP5 (Cell-averaged)	1E+10	68.8%	94.4%	100%

3.6 Demonstration of the KDE Integral-track Mesh Tally for a Real Problem

In addition to comparing the KDE integral-track mesh tally to the conventional mesh tally for the five simple fixed-source problems described in Section 3.2, a similar comparison

was performed on a real problem with more complex geometry. This geometry, shown in Figure 3.16, is a detailed blanket module model of ITER. For this particular problem the quantity of interest is the helium production in the flow insert, which was first introduced in Chapter 1 and is also shown in Figure 3.16. This flow insert has a maximum diameter of 8 cm and a length of 13 cm. The unstructured input mesh used to obtain the results had 14,671 mesh cells and 4735 mesh nodes, with an average edge length of around 0.37 cm. All of the results in this section were obtained using the parallel version of DAG-MCNP5 with CAD geometry.

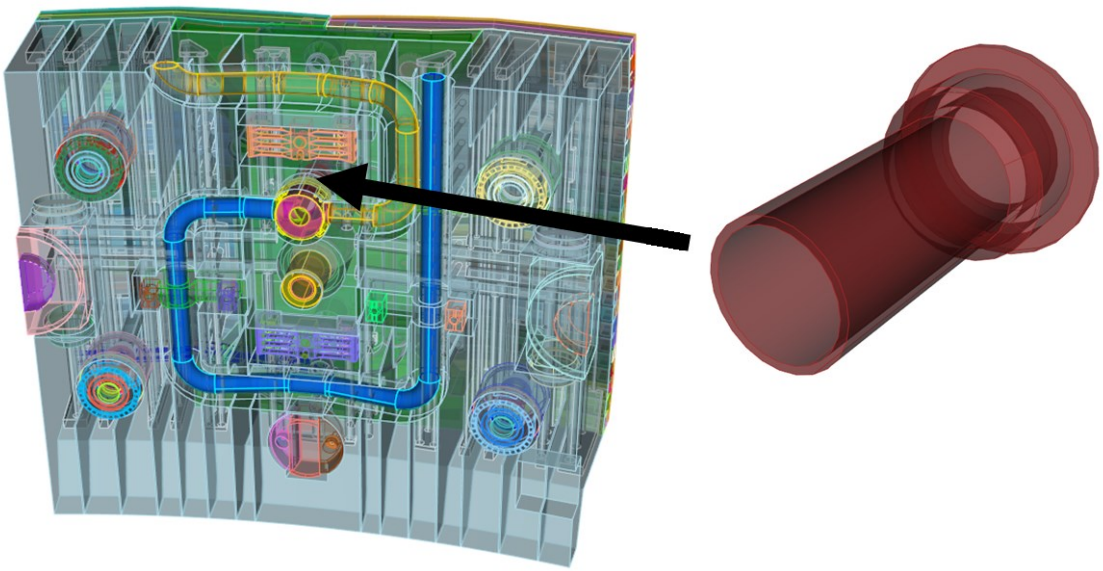

Figure 3.16 Detailed blanket module model of ITER highlighting the flow insert component.

Since an optimal bandwidth could not be computed using the parallel version of DAG-MCNP5, multiple values were considered for the KDE integral-track mesh tally. These bandwidth vectors were the same in all three dimensions, with values of h = 0.05, 0.1, 0.2, and 0.4. The kernel function that was used was the default second-order Epanechnikov kernel. Note also that since the flow insert is embedded within the rest of the geometry, boundary correction

was not needed. Results for the conventional mesh tally and three of the four KDE integral-track mesh tallies are shown in Figure 3.17, based on 5E+08 particle histories and a continuous tally multiplier to obtain the helium production in units of appm/sec. The nodal-based KDE results were converted into cell-averaged values using the data transfer method described in Section 3.5 with linear finite elements.

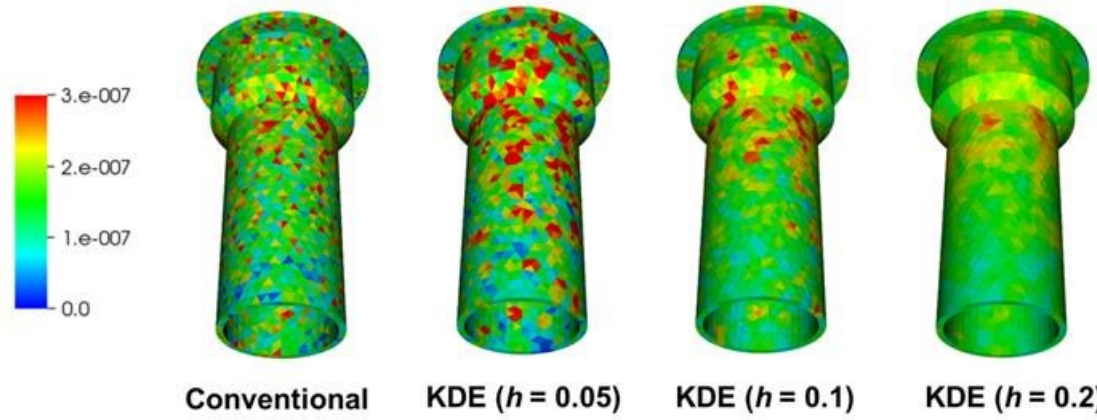

Figure 3.17 Cell-averaged results for the helium production (appm/sec) in the flow insert in the detailed blanket module model of ITER using the conventional mesh tally and the KDE integral-track mesh tally with different bandwidths.

From Figure 3.17, it appears as though the cell-averaged KDE integral-track mesh tally results using a bandwidth of $h = 0.2$ provides the best and most consistent solution for 5E+08 particle histories. This particular bandwidth was chosen based on the average edge length of the input mesh. As expected, reducing the size of this bandwidth introduces more variance in the results. The results for the conventional mesh tally also show significant variance. To perform a quantitative comparison on these results, a reference solution based on the conventional mesh tally with 5.55E+09 particle histories was obtained. The relative discrepancies for each mesh tally are shown in Figure 3.18 as cumulative distribution functions across all of the mesh cells.

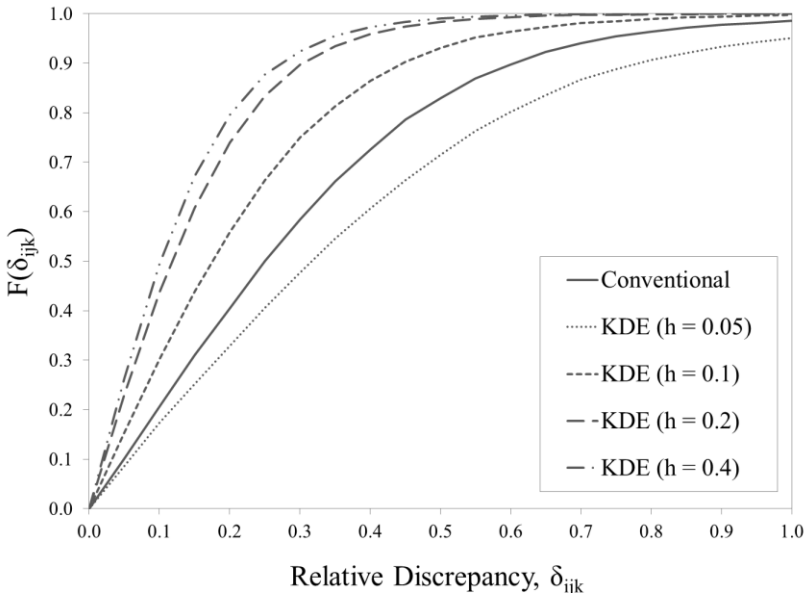

Figure 3.18 Cumulative distribution functions for the relative discrepancy between cell-averaged KDE integral-track and conventional mesh tally results for the helium production in the flow insert in the detailed blanket module model of ITER.

Figure 3.18 shows that the KDE integral-track mesh tally results become more accurate compared to the reference solution as the bandwidth is increased. As will be discussed in more detail in Chapter 4, these improved results occur with larger bandwidths for this particular problem because the helium production across the flow insert is fairly uniform. The bias in the results of the KDE integral-track mesh tally is much less sensitive to the bandwidth in regions that are uniform. Therefore, this is a problem that is well-suited to KDE mesh tallies. Note that, in comparison, the conventional mesh tally using 5E+08 particle histories actually produces results with greater discrepancies compared to the reference solution, even though the same input mesh was used in all cases. This example demonstrates that the KDE integral-track mesh tally is capable of producing more accurate results than the conventional mesh tally without needing to change the input mesh and/or use more particle histories.

CHAPTER 4

Accuracy of KDE Mesh Tallies

One of the most important metrics for assessing the feasibility of an estimator used to solve the neutron transport equation is accuracy. Accuracy is a measure of how close the approximation of the neutron flux distribution is to the true solution, which can have a significant impact on how nuclear systems are designed. For example, consider the decision of determining how much radiation shielding to use for some new nuclear technology. If the neutron flux is underestimated, this could pose a risk to workers as a result of insufficient shielding. On the other hand, if the neutron flux is overestimated this could increase cost by adding more shielding than was necessary. The optimal goal is to produce the best possible approximation that avoids both of these polarized scenarios. Recall from Chapter 1 that although the conventional mesh tally produces unbiased results with respect to the mean value of the neutron flux, its estimator is still biased with respect to the true solution at a specific point (x, y, z). Though the KDE integral-track mesh tally is also based on a biased estimator, this bias depends on the bandwidth and kernel function – not the mesh structure. Therefore, no matter how the input mesh is defined, the results will always be the same for a given bandwidth and kernel function.

In this chapter, several topics related to the accuracy of the KDE integral-track mesh tally are explored. Approximations for both bias and variance are first derived in Section 4.1 to provide a theoretical basis for exploring the sensitivity of the bandwidth. Then, in Section 4.2 these two approximations are used to analyze the results of a bandwidth sensitivity experiment that was performed. Section 4.3 expands on the conclusions drawn from Section 4.2 by

considering both global and regional methods for choosing an effective bandwidth. In Section 4.4 a different approach for improving the accuracy of the KDE integral-track mesh tally is introduced, based on using higher-order kernel functions as a bias reduction tool. Finally, Section 4.5 provides an overview of how the boundary bias effect can be corrected for 3D problems using the boundary kernel method.

4.1 Bandwidth Sensitivity

The concept of the bandwidth was first introduced in Chapter 2. This bandwidth can significantly impact the accuracy of any application that uses KDE methods at all calculation points. A visual example of how the KDE approximation of a standard normal distribution can change with extremely different bandwidth values is shown in Figure 4.1.

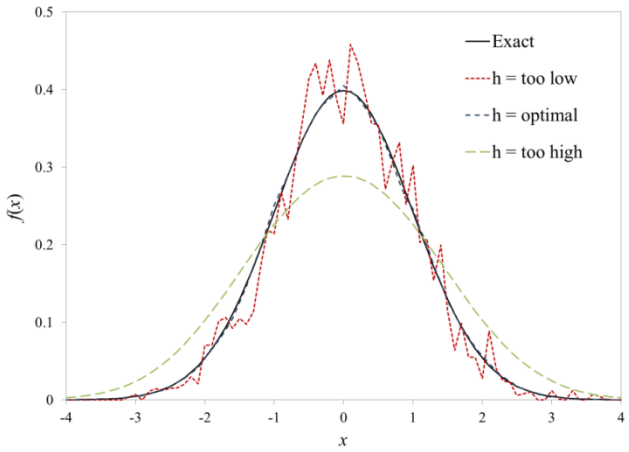

Figure 4.1 Effect of varying bandwidth on the accuracy of a 1D KDE approximation for the standard normal distribution.

When the bandwidth is too low, the general shape of the normal distribution is well resolved, but there is a substantial amount of variance. The opposite occurs when the bandwidth is too high – resulting in a much smoother approximation, but no longer correctly resolving the

shape. Finding the optimal bandwidth in practice is often done by minimizing the MISE for a specific distribution and kernel function. For example, Eq. 2.2 was obtained by using a multivariate normal distribution and a Gaussian kernel function [5, 7, 14]. Though this approach works well for the standard normal distribution, using a MISE-based bandwidth tends to produce the best balance between both bias and variance throughout the entire domain. However, what if it was more important to resolve the shape of the distribution for some local region? More observations can always be added to improve the variance, but the bias of any kernel density estimator is pre-determined by the bandwidth and kernel function.

In this section, approximations for both the bias and variance of the 3D kernel density estimator are derived. These approximations are needed to provide the theoretical basis for the experimental analysis that is performed in Sections 4.2 and 4.3. Rather than combining bias and variance terms together to form the MISE for assessing accuracy, each approximation will be considered on its own with respect to the bandwidth. Considering each of these statistical parameters separately allows the sensitivity of the bandwidth to be explored in greater detail.

4.1.1 Bias Approximation

Chapter 1 defined bias as the difference between the expected value of the approximated density and the true density. For a 3D density estimator, approximating the bias:

$$bias\left[\hat{f}(x,y,z)\right] = \mathrm{E}\left[\hat{f}(x,y,z)\right] - f(x,y,z), \tag{4.1}$$

is therefore equivalent to determining its expected value. The expected value of the 3D kernel density estimator can be expressed as an average of N terms:

$$\mathrm{E}\left[\hat{f}(x,y,z)\right]=\frac{1}{N}\sum_{i=1}^{N}\mathrm{E}\left[\frac{1}{h_x}K\left(\frac{x-X_i}{h_x}\right)\frac{1}{h_y}K\left(\frac{y-Y_i}{h_y}\right)\frac{1}{h_z}K\left(\frac{z-Z_i}{h_z}\right)\right]. \tag{4.2}$$

Each of the observations (X_i, Y_i, Z_i) is considered to be independent and randomly sampled from the same probability density function $f(X, Y, Z)$. This means that Eq. 4.2 is a summation of N identical terms and the expected value can be reduced into a single triple integral:

$$\mathrm{E}\left[\hat{f}(x,y,z)\right]=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{1}{h_x}K\left(\frac{x-X}{h_x}\right)\frac{1}{h_y}K\left(\frac{y-Y}{h_y}\right)\frac{1}{h_z}K\left(\frac{z-Z}{h_z}\right)f(X,Y,Z)\,dX\,dY\,dZ. \tag{4.3}$$

To evaluate this triple integral, first a change of variables is needed:

$$u=\frac{x-X}{h_x},\quad v=\frac{y-Y}{h_y},\quad w=\frac{z-Z}{h_z}, \tag{4.4}$$

which transforms Eq. 4.3 into the following form:[4]

$$\mathrm{E}\left[\hat{f}(x,y,z)\right]=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\prod_{l=x}^{z}K(u_l)f(x-uh_x,\,y-vh_y,\,z-wh_z)\,du\,dv\,dw. \tag{4.5}$$

The next step is to expand $f(x-uh_x,\,y-vh_y,\,z-wh_z)$ into a multivariate Taylor expansion about the point (x, y, z):[5]

$$f(x-uh_x,\,y-vh_y,\,z-wh_z)=f(x,y,z)-\sum_{j=x}^{z}h_j u_j\frac{\partial f}{\partial j}+\frac{1}{2}\sum_{j=x}^{z}h_j^2u_j^2\frac{\partial^2 f}{\partial j^2}-\ldots, \tag{4.6}$$

---

[4] The notation $\prod_{l=x}^{z}K(u_l)$ is shorthand for the separable 3D kernel function: $K(u,v,w)=K(u)K(v)K(w)$.

[5] The notation $\sum_{j=x}^{z}g(u_j)$ represents a summation over x, y, and z with $u_x = u$, $u_y = v$, and $u_z = w$.

which can then be substituted back into Eq. 4.5:

$$\mathrm{E}\left[\hat{f}(x,y,z)\right]=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\prod_{l=x}^{z}K(u_l)\left\{f(x,y,z)-\sum_{j=x}^{z}h_j u_j\frac{\partial f}{\partial j}+\frac{1}{2}\sum_{j=x}^{z}h_j^2u_j^2\frac{\partial^2 f}{\partial j^2}-\ldots\right\}du\,dv\,dw. \quad (4.7)$$

Note that the first term in Eq. 4.7 is multiplied by an integral of $K(u, v, w)$ over its entire domain. Since this integral must be unity, by definition, this first term reduces to the true density $f(x, y, z)$. Now consider the second term. All three components of this second term are multiplied by one integral of the form:

$$\mathrm{E}\left[K(u_l)\right]=\int_{-\infty}^{\infty}u_l K(u_l)du_l. \quad (4.8)$$

Assuming that the kernel functions $K(u_l)$ are all symmetric, Eq. 4.8 will always evaluate to zero. Therefore, the second term is eliminated and only $f(x, y, z)$ and the third term remains. If it is also assumed that the bandwidth vector is small enough to neglect all higher-order terms, then Eq. 4.7 can be rearranged into the following form:

$$bias\left[\hat{f}(x,y,z)\right]=\mathrm{E}\left[\hat{f}(x,y,z)\right]-f(x,y,z)\approx\frac{1}{2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\prod_{l=x}^{z}K(u_l)\left\{\sum_{j=x}^{z}h_j^2u_j^2\frac{\partial^2 f}{\partial j^2}\right\}du\,dv\,dw. \quad (4.9)$$

Eq. 4.9 approximates the bias of the 3D kernel density estimator for different bandwidths and kernels in each dimension. To simplify this equation even further, suppose that the bandwidth h_j and kernel $K(u_l)$ are the same in all three dimensions. Then, the bias approximation becomes:

$$bias\left[\hat{f}(x,y,z)\right]\approx\frac{h^2}{2}\int_{-\infty}^{\infty}u^2K(u)du\left(\frac{\partial^2 f(x,y,z)}{\partial x^2}+\frac{\partial^2 f(x,y,z)}{\partial y^2}+\frac{\partial^2 f(x,y,z)}{\partial z^2}\right). \quad (4.10)$$

This simplified bias approximation consists of a product of three distinct factors. The first factor shows that the bias is expected to decrease proportionally to $O(h^2)$ as the bandwidth h gets smaller, whereas the second represents the variance of the kernel function $K(u)$. Since both of these factors are always non-zero for second-order kernels, the only way that the 3D kernel density estimator can produce zero bias is if the third factor is zero. This third factor is a summation of second derivatives of the true density $f(x, y, z)$, which provides a measure of the amount of curvature. Therefore, points in regions with greater curvature are expected to produce results that are more biased than points in regions with less curvature.

4.1.2 Variance Approximation

The variance of the 3D kernel density estimator can be approximated using a similar approach to the derivation of the bias approximation. First note that the $1/N$ term in Eq. 2.1 is considered constant, which means that:

$$var\left[\hat{f}(x,y,z)\right] = \frac{1}{N^2} var\left[\sum_{i=1}^{N} \frac{1}{h_x} K\left(\frac{x-X_i}{h_x}\right) \frac{1}{h_y} K\left(\frac{y-Y_i}{h_y}\right) \frac{1}{h_z} K\left(\frac{z-Z_i}{h_z}\right)\right]. \tag{4.11}$$

Then, since each observation (X_i, Y_i, Z_i) was sampled from the same distribution $f(X, Y, Z)$, the variance of the sum of kernel contributions can be expressed as the sum of N identical variances. As such, one of the N terms cancels and Eq. 4.11 becomes:

$$var\left[\hat{f}(x,y,z)\right] = \frac{1}{N} var\left[\frac{1}{h_x} K\left(\frac{x-X}{h_x}\right) \frac{1}{h_y} K\left(\frac{y-Y}{h_y}\right) \frac{1}{h_z} K\left(\frac{z-Z}{h_z}\right)\right] = \frac{1}{N} var\left[K_{X,Y,Z}\right], \tag{4.12}$$

where $K_{X,Y,Z}$ is used as shorthand for the kernel contribution. To evaluate the variance of $K_{X,Y,Z}$, note that Eq. 4.12 can be rewritten in terms of expected values:

$$var\left[\hat{f}(x,y,z)\right] = \frac{1}{N}\left\{ \mathrm{E}\left[K_{X,Y,Z}^2\right] - \mathrm{E}\left[K_{X,Y,Z}\right]^2 \right\} \tag{4.13}$$

Recall from the previous derivation that $\mathrm{E}[K_{X,Y,Z}]$ can be approximated by $f(x, y, z) + \mathrm{O}(h^2)$. The first term of Eq. 4.13 can be approximated by expressing it as the following triple integral:

$$\mathrm{E}\left[K_{X,Y,Z}^2\right] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left\{\frac{1}{h_x}K\left(\frac{x-X}{h_x}\right)\frac{1}{h_y}K\left(\frac{y-Y}{h_y}\right)\frac{1}{h_z}K\left(\frac{z-Z}{h_z}\right)\right\}^2 f(X,Y,Z)\,dX\,dY\,dZ, \tag{4.14}$$

and then by using the same change of variables and multivariate Taylor expansion as before:

$$\mathrm{E}\left[K_{X,Y,Z}^2\right] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\prod_{l=x}^{z}\frac{K(u_l)^2}{h_l}\left\{f(x,y,z) - \sum_{j=x}^{z}h_j u_j\frac{\partial f}{\partial j} + \frac{1}{2}\sum_{j=x}^{z}h_j^2 u_j^2\frac{\partial^2 f}{\partial j^2} - \ldots\right\}du\,dv\,dw. \tag{4.15}$$

The only term in Eq. 4.15 that is divided by all three components of the bandwidth vector is the first term. If each of these components is assumed to be small, then this first term will always be the largest and most dominant of the series. As a result, all higher-order terms can be ignored. Dividing Eq. 4.15 by N will also not approach zero like the second term of Eq. 4.13 as $N \to \infty$ and $h_l \to 0$. This means that Eq. 4.13 can be approximated by the following:

$$var\left[\hat{f}(x,y,z)\right] \approx \frac{1}{Nh_x h_y h_z} f(x,y,z)\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\prod_{l=x}^{z}K(u_l)^2\,du\,dv\,dw. \tag{4.16}$$

Eq. 4.16 approximates the variance of the 3D kernel density estimator for different bandwidths and kernels in each dimension. If it is assumed that the bandwidth h_j and kernel $K(u_l)$ are all the same, then the variance approximation reduces to:

$$var\left[\hat{f}(x,y,z)\right] \approx \frac{1}{Nh^3} f(x,y,z)\left(\int_{-\infty}^{\infty}K(u)^2\,du\right)^3. \tag{4.17}$$

Like the bias approximation defined by Eq. 4.10, this simplified variance approximation also consists of a product of three distinct factors. The first factor of Eq. 4.17 shows that the variance is inversely proportional to $O(h^3)$, which contrasts the behavior of the bias. Therefore, these two approximations demonstrate mathematically why there is always a trade-off between bias and variance with respect to the choice of bandwidth. The second and third factors of Eq. 4.17 show that the variance also depends on the value of the true density $f(x, y, z)$, as well as the roughness of the kernel function $K(u)$.

4.2 Bandwidth Sensitivity Experiment

To see if the KDE integral-track mesh tally is consistent with the bias and variance approximations that were just derived for the 3D kernel density estimator, a bandwidth sensitivity experiment was performed using the DAGTally implementation. A description of the methodology that was used and a detailed analysis of the results follows.

4.2.1 Methodology

The test case used in the bandwidth sensitivity experiment was a slightly larger version of the Uniform Volume Source problem described in Chapter 3. Increasing the outer radius of the sphere from 0.12 cm to 0.14 cm (~3.3 mean free paths) was necessary so that the experiment could be performed with a reasonable range of bandwidth values, without having to worry about correcting for the boundary bias effect described in Chapter 2. Once the geometry of the test case was defined using native MCNP5 geometry, an unstructured spherical mesh was created with a radius of 0.10 cm and 1010 tetrahedral cells. Then, KDE integral-track mesh tallies based on the Epanechnikov kernel in all three dimensions and 1E+06 particle histories were added in

separate input files for a series of 20 bandwidth values (ranging from 0.001 to 0.020). Note that due to the angular symmetry in the problem, all three components of the bandwidth vector were set to the same values in each of the 20 different input files that were created.

After setting up all of the input files, each one was run 200 times using a fixed set of 200 random seed numbers. This was done to estimate the expected value and sample variance of the kernel density estimated fluxes at each mesh node. When this first stage of the experiment was completed, the entire procedure was then repeated using the uniform kernel to observe how the choice of kernel function affects bandwidth sensitivity. The uniform kernel was chosen for this analysis because it has one of the lowest efficiency ratings compared to the Epanechnikov kernel [5]. Therefore, any changes in bandwidth sensitivity that occurred as a result of using the two different kernels would be at a maximum.

Once results were obtained for both Epanechnikov and uniform kernels, a finer mesh was then created with 26,124 tetrahedral cells to provide a more detailed reference solution. This reference solution, shown in Figure 4.2 as a function of radius, was obtained using the TET track mesh tally with 1E+08 particle histories and the same fixed set of 200 random seed numbers. Though the TET track mesh tally only produces cell-averaged results, it could still be reduced to a 1D function because of the angular symmetry. This was done by computing a least-squares fit to the cell-averaged data using a sixth-order polynomial, with the radius at the center of the mesh cells defined as the independent variable. Using this 1D reference solution enabled the neutron flux to be compared at any radius within the sphere, without needing to use any of the data transfer methods discussed in Chapter 3.

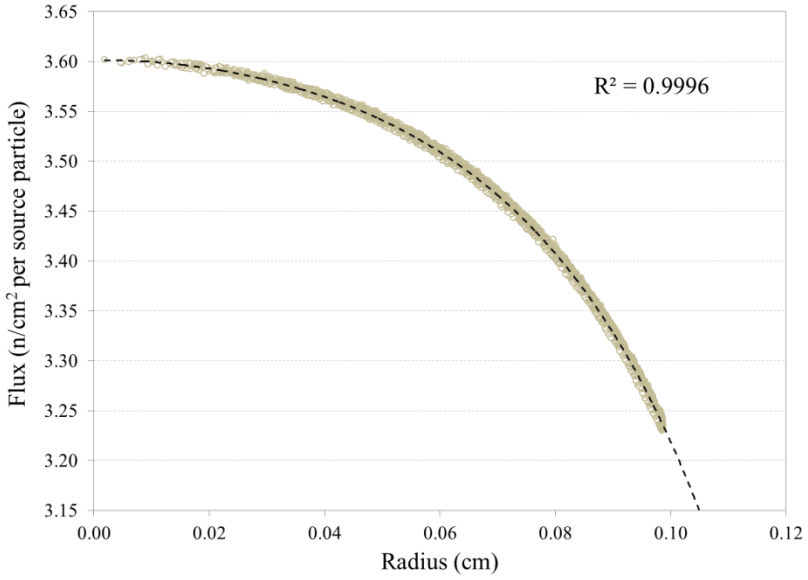

Figure 4.2 1D reference solution (dashed line) based on a least-squares fit to the cell-averaged results of the TET track mesh tally (data points).

4.2.2 Results: Bias vs. Bandwidth

The first metric used to analyze the tally data was the bias between the estimated expected value and the reference solution shown in Figure 4.2. Figure 4.3 plots this bias as a function of bandwidth for both a boundary point and an interior point on the mesh. Note that the error bars in the left plot reflect the approximate statistical error in the bias measurements:

$$\sigma_{bias} \approx \sqrt{\left(\frac{s_h}{\sqrt{200}}\right)^2 + \sigma_{ref}^2}\,, \tag{4.18}$$

where σ_{ref} is the standard error in the least-squares fit, and s_h is the sample standard deviation of the 200 different KDE integral-track mesh tally results obtained per bandwidth value h. Eq. 4.18 assumes that both the estimated expected value and the reference solution can be treated as independent random variables. However, since σ_{ref} is only on the order of 10^{-3}, the statistical

error is generally dominated by the variance in the KDE approximations. This explains why the error bars are much larger for lower bandwidths, due to the corresponding increase in variance.

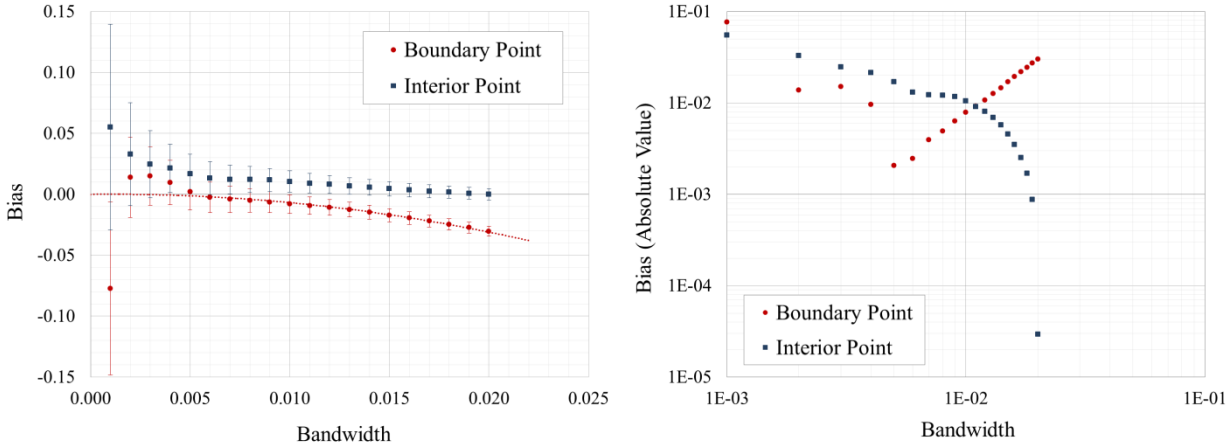

Figure 4.3 Bias in the KDE integral-track mesh tally results versus bandwidth.

The left plot of Figure 4.3 shows that the bias is approximately zero for the interior point.[6] However, the bias in the boundary point is proportional to $O(h^2)$ and clearly gets worse as the bandwidth is increased. This $O(h^2)$ relationship can be seen more clearly for the larger bandwidth values in the right plot of Figure 4.3, which shows the absolute value of the bias on a log-log scale. The differences between these two points can be explained by comparing the bias measurements to the theoretical bias approximation that was derived in Section 4.1. Recall that the only factor of Eq. 4.10 that can cause zero bias is the sum of derivatives evaluated at a specific point in the neutron flux distribution. This seems to be the case for the interior point, since it appears to be invariant to the bandwidth. For all other cases, like the boundary point, the bias in the KDE integral-track mesh tally will always be proportional to $O(h^2)$.

[6] Bias at very small bandwidths for different interior points fluctuates due to statistical variance.

4.2.3 Results: Variance vs. Bandwidth

The second metric used to analyze the tally data was the variance of the KDE integral-track mesh tally results. Figure 4.4 plots the sample variance as a function of bandwidth for the same two mesh points that were considered in the previous analysis. Though the bias for the two mesh points behaved differently, Figure 4.4 shows that the variance in both cases decreases at around the same rate. The only difference is that the variance at the boundary point is slightly lower in magnitude than the variance at the interior point, which can be explained by comparing the experimental data to the theoretical variance approximation that was derived in Section 4.1. Given that everything else is the same for both mesh points, the only factor in Eq. 4.17 that can differ is the magnitude of the true neutron flux $\varphi(x, y, z)$. Therefore, it is not surprising that the interior point has slightly more variance because the flux is higher at the center of the sphere than it is on the surface.

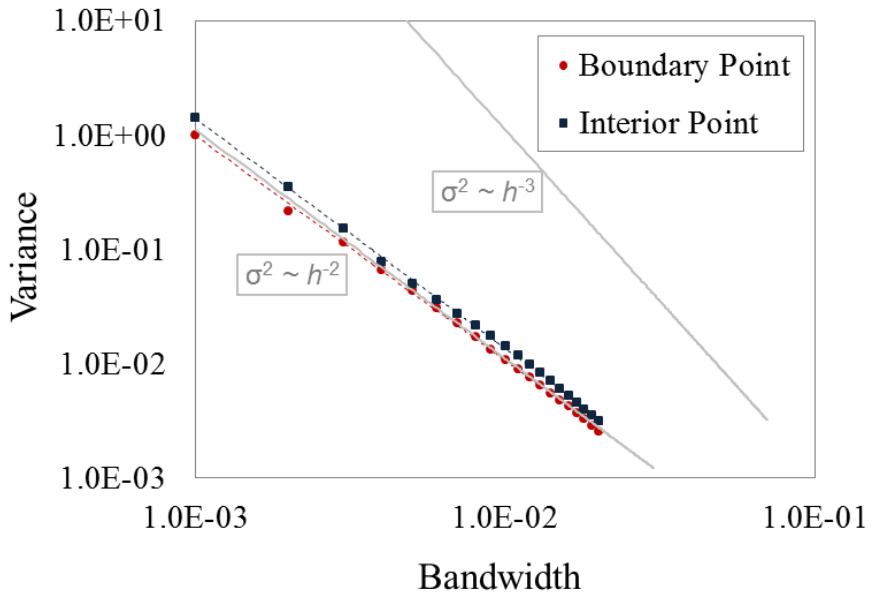

Figure 4.4 Variance in the KDE integral-track mesh tally results versus bandwidth.

While the magnitude of the sample variance is consistent with theoretical expectations, the rate at which it decreases with respect to the bandwidth is not. According to Eq. 4.17, the variance should be inversely proportional to $O(h^3)$. However, based on the results shown in Figure 4.4, the sample variance is roughly inversely proportional to $O(h^2)$ instead. Table 4.1 shows the behavior of sample variances that were computed for both the boundary point and the interior point for different KDE mesh tallies. Note that KDE collision mesh tally results decrease by the expected factor of around h^3, whereas KDE sub-track mesh tally results decrease by a factor between h^3 and h^2 depending on the number of sub-tracks that were used. These results seem to indicate that Eq. 4.17 does not accurately predict the variance for KDE mesh tallies based on particle tracks instead of collisions.

Table 4.1 Behavior of sample variance versus bandwidth for different KDE mesh tallies.

Mesh Tally Type	**Boundary Point $O(h^{-x})$**	**Interior Point $O(h^{-x})$**
KDE Collision	2.91	3.08
KDE Sub-track (n = 2)	2.67	2.76
KDE Sub-track (n = 4)	2.49	2.69
KDE Sub-track (n = 6)	2.32	2.48
KDE Sub-track (n = 8)	2.21	2.41
KDE Integral-track	2.03	2.03

4.2.4 Results: Neutron Flux vs. Bandwidth

The third metric used to analyze the tally data was the neutron flux distribution itself, represented as a 1D function of radius. Figure 4.5 shows the results that were obtained using the

KDE integral-track mesh tally with three different bandwidth values: too low (h = 0.004), too high (h = 0.020), and optimal (h = 0.008). The optimal value was approximated using Eq. 2.2 based on collision sites. All three cases produce a reasonable representation of the neutron flux distribution considering that only 1E+06 particle histories were used. The most accurate results overall, however, seem to be from using the optimal bandwidth. When the bandwidth is reduced from this optimal value to h = 0.004, there is significantly greater variance. In contrast, when the bandwidth is increased to h = 0.020 there is less variance, but a noticeable increase in bias. These two extremes are most noticeable near the surface of the mesh, which provides a great example of the trade-off that exists between bias and variance whenever KDE methods are used.

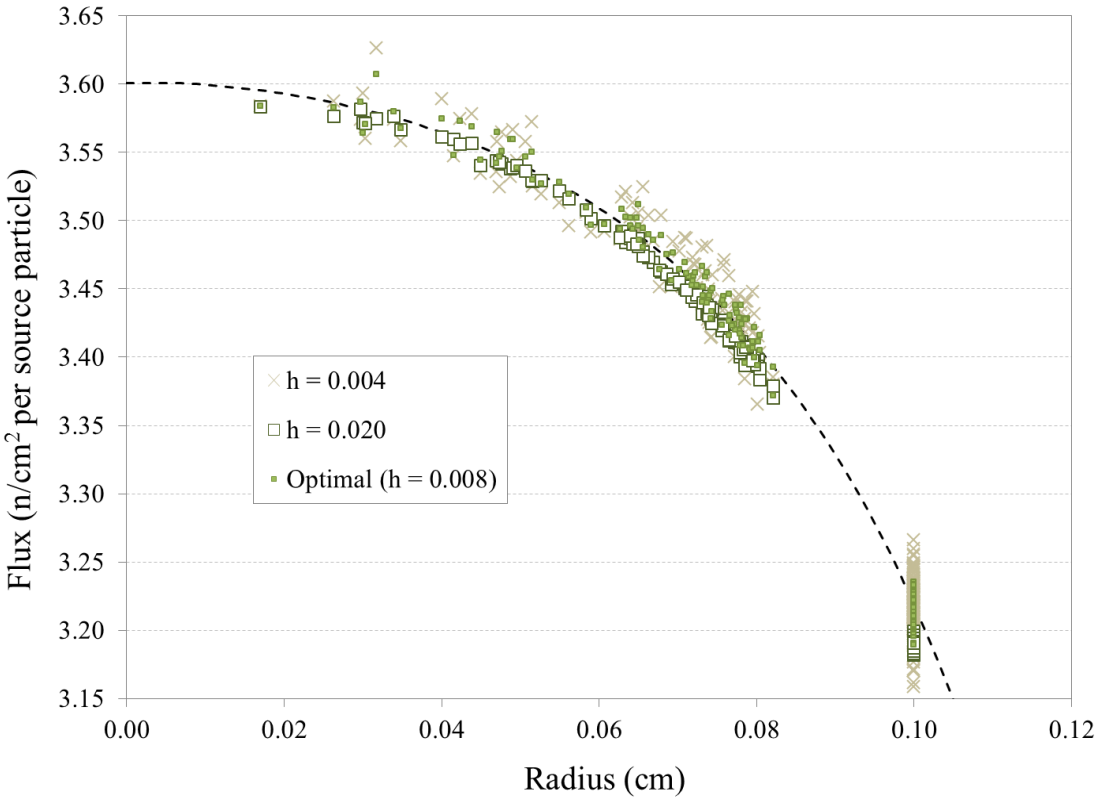

Figure 4.5 Neutron flux as a function of radius for three KDE integral-track mesh tally solutions using different bandwidth values.

To see the trade-off between bias and variance at the surface of the mesh more clearly, all 136 mesh points tallied at a radius of 0.10 cm are plotted as a line graph in Figure 4.6 for the same three bandwidth values used to construct Figure 4.5. This set of 136 mesh points can be considered collectively as a random sample that approximates the true neutron flux at this radius. Note that the solid horizontal line represents the reference solution. Figure 4.6 shows that as the bandwidth increases from $h = 0.004$ to $h = 0.020$, the bias between the estimated expected values and the reference solution for all 136 mesh points also increases on average. This indicates that the bandwidth used with the KDE integral-track mesh tally should be as low as possible to reduce the bias in the approximation near the surface of the mesh. However, Figure 4.6 also shows that the opposite is true with respect to the variance, since when the bandwidth is reduced from $h = 0.020$ to $h = 0.004$ the variance in the approximated values for the random sample increases. Therefore, as shown in both Figure 4.5 and Figure 4.6, the optimal bandwidth provides the best solution overall for this problem as it minimizes both bias and variance.

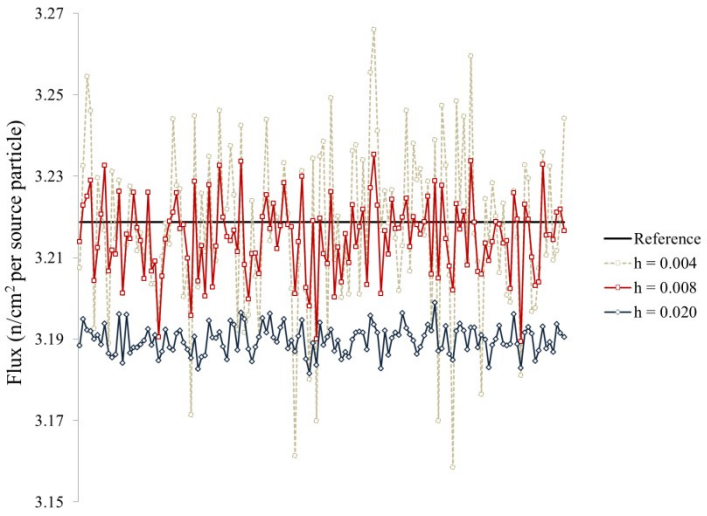

Figure 4.6 Neutron flux results obtained using different bandwidth values for all mesh points at a radius of 0.10 cm.

4.2.5 Results: Kernel Function vs. Bandwidth

The fourth and final metric used to analyze the tally data was the choice of kernel function with respect to both bias and variance. Figure 4.7 and Figure 4.8 compare the bias at the boundary point and interior point respectively for the Epanechnikov and uniform kernels.

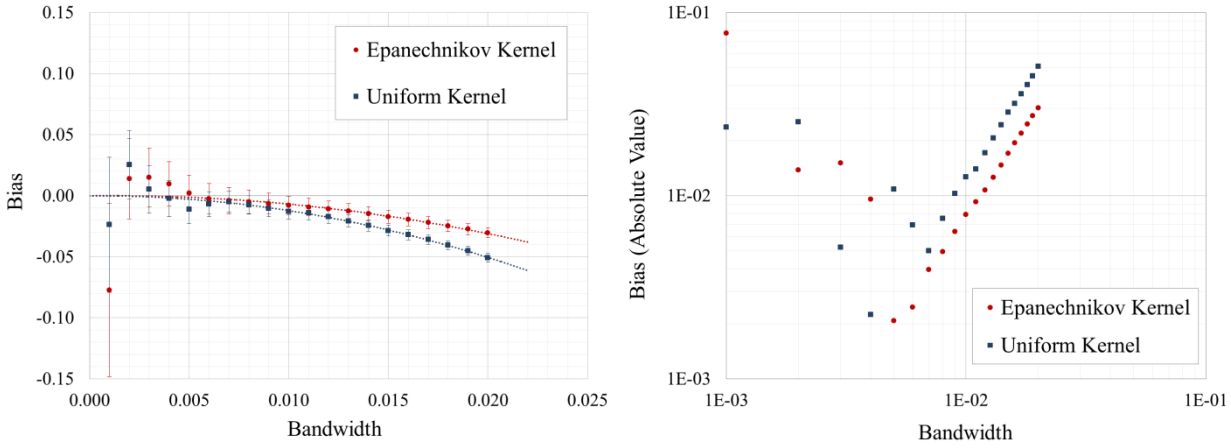

Figure 4.7 Effect of kernel on bias for boundary point plotted as a function of bandwidth.

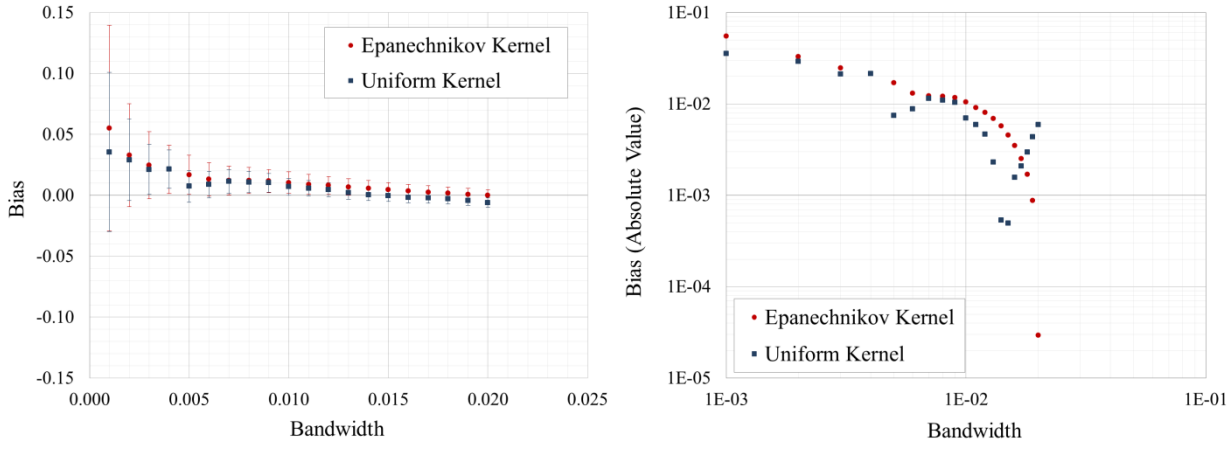

Figure 4.8 Effect of kernel on bias for interior point plotted as a function of bandwidth.

For the interior point there is little difference between the two kernel functions because the bias is already approximately zero at all bandwidth values within the statistical error.

Although the right plot of Figure 4.8 starts to show some linearity on a log-log scale for the uniform kernel, these results are inconclusive. In contrast, there is a noticeable difference at the boundary point – where the Epanechnikov kernel clearly produces results with less bias than the uniform kernel. As the variance is reduced with the larger bandwidth values, there is a strong $O(h^2)$ relationship between bias and bandwidth for both kernel functions.

Referring back to Eq. 4.10, the only factor in the bias approximation that is affected by the choice of kernel function is the kernel variance. From Table 2.1, this kernel variance is 0.20 for the Epanechnikov kernel, and 0.33 for the uniform kernel. Therefore, it is expected that the Epanechnikov kernel will produce results that are 40% less biased than the uniform kernel. Theoretically, this means that the ratio of the bias computed using the Epanechnikov kernel over the uniform kernel should always be around 0.6. Figure 4.9 shows that this is indeed the case on average for the majority of bandwidth values considered in the experiment. Note that the error bars in Figure 4.9 were obtained by propagating the errors in the bias measurements that were computed using Eq. 4.18 for both the Epanechnikov and uniform kernels. The larger error bars for the smaller values occur due to the variance being more dominant than the bias.

In addition to affecting the bias, kernel choice can also impact the variance. Figure 4.10 plots the ratio of the sample variance computed using the Epanechnikov kernel over the uniform kernel for both mesh points. Since the variance of these sample variances was not measured as part of the bandwidth sensitivity experiment, error bars for this plot are omitted. According to Eq. 4.17, the only factor that is affected by the kernel choice is the one that includes the roughness integral. From Table 2.1, this roughness integral is 0.6 for the Epanechnikov kernel,

and 0.5 for the uniform kernel. Therefore, it is expected that the variance in the results obtained using the Epanechnikov kernel should be around 1.7 times greater than when using the uniform kernel. While there is some deviation shown in Figure 4.10, on average the experimental results were consistent with this theoretical value of 1.7.

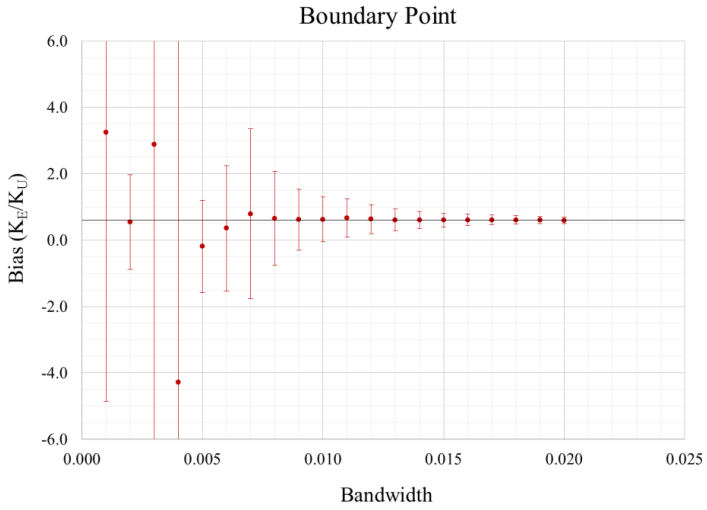

Figure 4.9 Ratio of the Epanechnikov bias (K_E) to the uniform bias (K_U) for the boundary point.

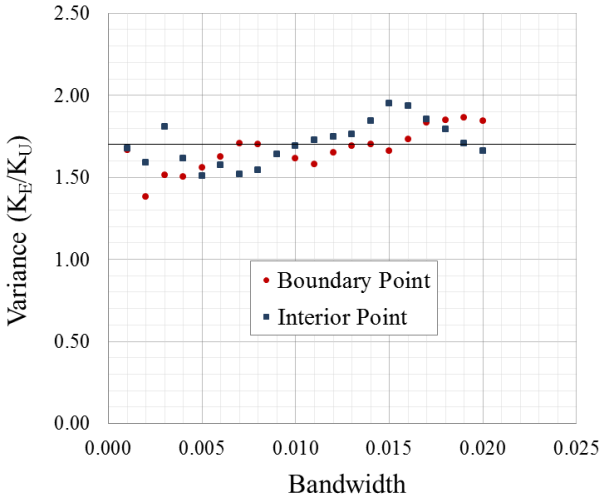

Figure 4.10 Ratio of the Epanechnikov variance (K_E) to the uniform variance (K_U) for both the boundary point and the interior point.

4.3 Choosing an Effective Bandwidth

Results from the bandwidth sensitivity experiment have shown that the bandwidth used with the KDE integral-track mesh tally can have a significant impact on both bias and variance. The trade-off between these two statistical parameters can make it difficult to choose an appropriate bandwidth at all calculation points. Many different bandwidth selection methods are available in the statistical literature [5]. However, as Banerjee has suggested in previous work, most of these are difficult to implement, or would be computationally expensive for Monte Carlo transport purposes [7]. Recall from Chapter 2 that any bandwidth method used for solving fixed-source problems will generally require that the Monte Carlo simulation is run twice. Therefore, it is extremely important that this method be as efficient as possible. In this section, two simple methods for choosing bandwidth values will be compared against a 1D reference distribution with highly contrasting levels of curvature. The first method computes an optimal global value for the entire domain, whereas the second uses a more regional approach.

4.3.1 1D Reference Distribution

To assess the effectiveness of global versus regional bandwidths, the following 1D probability density function is used:

$$f(x) = 0.5\,U(0,1) + 0.5\,N(0.5, 0.001), \tag{4.19}$$

where $U(0, 1)$ is the standard uniform distribution, and $N(0.5, 0.001)$ is a sharply-peaked normal distribution defined by Eq. 1.11 with $\mu = 0.5$ and $\sigma^2 = 0.001$. The reason for analyzing this particular 1D mixture distribution, which is plotted in Figure 4.11, is because it accentuates the difference between regions of no curvature and regions of high curvature.

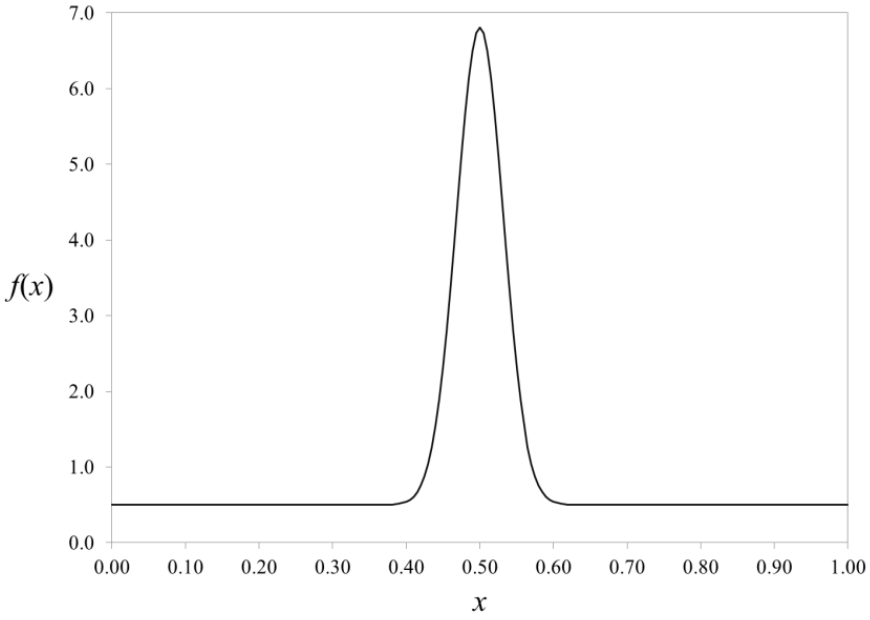

Figure 4.11 A 1D probability density function representing a mixture distribution that consists of 50% $U(0, 1)$ and 50% $N(0.5, 0.001)$.

Before the 1D kernel density estimator can be used to approximate the distribution shown in Figure 4.11, first a random sample of N observations X_i must be obtained. This can be done using the following procedure:

1. Sample X from $U(0, 1)$
2. Check whether X belongs to $U(0, 1)$ or $N(0.5, 0.001)$
    i. If $X < 0.5$, sample X_i from $U(0, 1)$
    ii. Otherwise, sample X_i from $N(0.5, 0.001)$
3. Repeat for all N observations

A random sample of 100,000 observations was obtained using the above procedure with the R programming language [24]. R is a programming language often used by statisticians because it

includes default functions that allow sampling from uniform (i.e., *runif*) and normal (i.e., *rnorm*) densities. This fixed set of 100,000 observations was used to analyze the effectiveness of both the global and regional bandwidth approach discussed in the following sections. Note also that all kernel contributions were computed using the Epanechnikov kernel and the 1D boundary kernel method [25] for correcting the boundary bias at both the left and right boundaries. Without this boundary correction, the results for calculation points less than one bandwidth from each end of the 1D mixture distribution would be significantly underestimated.

4.3.2 Global Bandwidth

The global bandwidth approach is the simplest method for choosing a bandwidth, since it uses the same value for computing all of the kernel contributions at all calculation points. However, while using a global bandwidth can be effective for general cases, it can also introduce greater bias for regions in the probability density function where there is greater curvature. To show how the choice of global bandwidth can affect the accuracy of a KDE approximation in regions of significantly varying curvature, consider the 1D mixture distribution described in the previous section. KDE approximations were computed for $f(x)$ using three distinct global bandwidth values. These three global bandwidth values are summarized in Table 4.2. The optimal value was computed by converting Eq. 2.2 to the 1D case:

$$h_{\mathrm{OPT}} = 1.06\sigma N^{-1/5}, \tag{4.20}$$

where σ is the sample standard deviation of the 100,000 observations that were used. The uniform-skewed and normal-skewed values were computed using different random samples of 100,000 observations taken from separate $U(0, 1)$ and $N(0.5, 0.001)$ densities respectively.

Table 4.2 Bandwidth values used to approximate a 1D mixture distribution.

Global Bandwidth	Notation	Value
Optimal	h_{OPT}	0.02180754
Uniform-skewed	h_U	0.03057827
Normal-skewed	h_N	0.00335376

Figure 4.12 compares the three approximations that were computed for $f(x)$ using the different global bandwidths near the peak region and across all regions. These results show that h_N is clearly the best option for approximating the peak region, but the worst option for the uniform regions due to significantly increased variance. Conversely, h_U is the best option for approximating the uniform regions, but the worst option for the peak region. The optimal bandwidth h_{OPT} attempts to provide the best solution throughout the entire domain. However, like h_U, this optimal value falls short near the peak where there is substantial curvature.

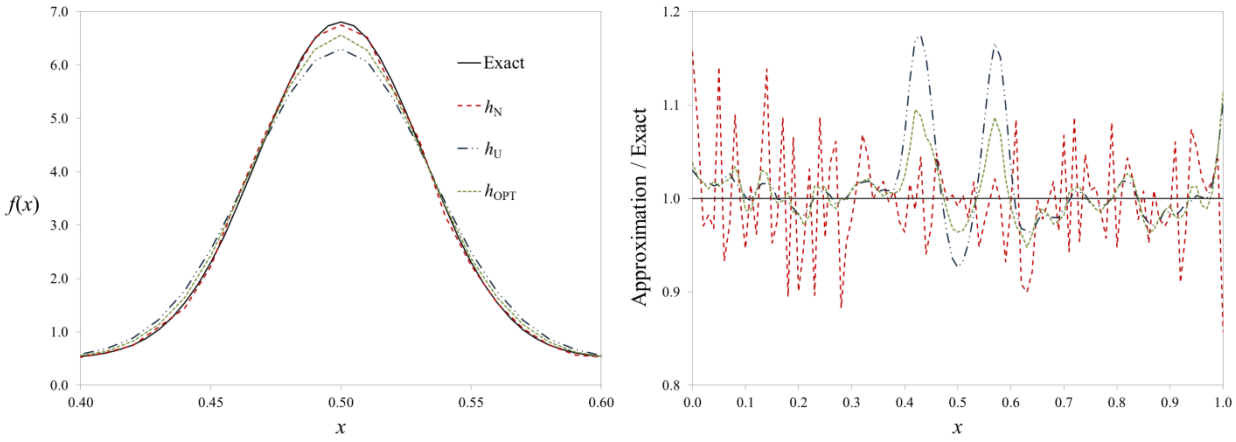

Figure 4.12 KDE approximations compared to the exact solution of a 1D mixture distribution for the peak region (left) and all regions (right), based on using different global bandwidths.

If the primary goal of a Monte Carlo simulation is to accurately capture the peak of a neutron flux distribution, then Figure 4.12 suggests that it would be better to use a smaller global bandwidth rather than the "optimal" value.[8] This is confirmed numerically in Table 4.3, which summarizes the mean relative discrepancies that were computed for each of the approximations with respect to the true density. Note that each calculation point was assigned to only one region for computing these discrepancies. The left and right regions include all calculation points within the open intervals [0.0, 0.4) and (0.6, 1.0] respectively, whereas the peak region includes all calculation points within the closed interval [0.4, 0.6]. By using h_N instead of h_{OPT}, the mean relative discrepancy in the peak region is reduced from 4.12% to 2.41%. The results are even better at the highest point in this peak, with the discrepancy for $x = 0.5$ being reduced from 3.60% to 0.86%. Though this is only a 1D example, for 3D cases the probability densities could potentially include regions with even greater fluctuations in curvature. Therefore, using a smaller global bandwidth rather than the optimal one could offer even greater benefit.

Table 4.3 Mean relative discrepancy in the KDE approximations of a 1D mixture distribution, based on using different global bandwidths.

Global Bandwidth	**MEAN RELATIVE DISCREPANCY**				
	Peak ($x = 0.5$)	**Left Region**	**Peak Region**	**Right Region**	**All Regions**
h_U	7.40%	1.20%	8.79%	1.70%	2.98%
h_N	0.86%	5.40%	2.41%	4.33%	4.35%
h_{OPT}	3.60%	1.40%	4.12%	2.05%	2.22%

[8] This would also be true for capturing valleys, which tend to be overestimated by the optimal global bandwidth.

4.3.3 Regional Bandwidth

One alternative to using a global bandwidth without introducing excessive pre-processing requirements is to use regional bandwidths. This regional bandwidth approach was first used in Monte Carlo transport for approximating the neutron flux of a 1D criticality problem based on an array of six fuel pins surrounded by water [7, 14]. In this work, regional bandwidths were obtained by first dividing the domain into R separate regions, and then by computing local optimal bandwidths h_R using only the observations that fell within R:

$$h_R = 1.06\sigma_R N_R^{-1/5}. \tag{4.21}$$

Once h_R has been determined for each region, it can then be used with the 1D kernel density estimator to approximate $f(x)$. Regions with high curvature can now use smaller bandwidths than regions with low curvature, which reduces the bias in the results that might otherwise occur trying to fit one "optimal" bandwidth to all regions. However, an important distinction must be made as to whether these regional bandwidths are associated with the observation points or the calculation points. For the former case, the 1D kernel density estimator is defined by:

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h_{R,i}}K\left(\frac{x - X_i}{h_{R,i}}\right), \tag{4.22}$$

where $h_{R,i}$ is the bandwidth of the region in which X_i is located.

The 1D mixture distribution that was analyzed using the global bandwidth method was also used to show how effective regional bandwidths can be for approximating distributions with significantly varying curvature. Eight different regional configurations were considered, with each configuration splitting the domain into R regions of equal width. R3, for example, splits the

domain into left [0, 0.333), middle [0.333, 0.667), and right [0.667, 1.00] regions. Once each local region is defined, a local optimal bandwidth can then be determined within that region using Eq. 4.21. Figure 4.13 plots the mean relative discrepancies that were computed for each configuration, based on the same distribution of calculation points used in Table 4.3.

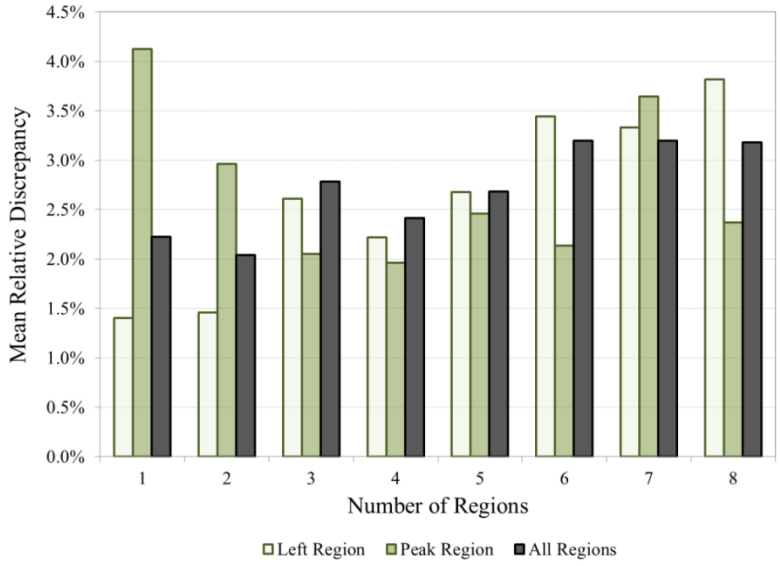

Figure 4.13 Mean relative discrepancy in the KDE approximations of a 1D mixture distribution, based on using different regional bandwidths.

Beginning with the R1 case, note that the results shown in Figure 4.13 are identical to the results shown in Table 4.3 for the optimal global bandwidth. As the number of fixed-width regions increases, the optimal local bandwidth changes depending on the distribution of observations throughout the individual regions. For most of these cases, the bandwidth used in the middle region is roughly half the bandwidth that is used in the outer regions. Using the smaller bandwidth in the middle region clearly produces more accurate results near the peak, since going from R1 to R4 reduces the mean relative discrepancy in this region from 4.12% to 1.96%. While this improvement still comes at the cost of increased variance in the uniform

regions, this cost is much less than if a smaller global bandwidth were used. Beyond R4 the improvement in the peak starts to worsen again, which is caused by the bandwidth becoming so small that increased variance starts to dominate over the improvement in the bias.

Though using more regions produced better approximations for the peak region up to R4, Figure 4.13 also shows that the R3 case performed worse than both R2 and R4. This behavior was unexpected because it was originally thought that three fixed-width regions would be optimal for separating the peak from the uniform regions. Some insight as to why R3 performed worse than expected can be seen in Figure 4.14, which plots some of the approximations that were obtained for the left uniform region and the peak region. Though R3 produces accurate results for the majority of the calculation points, note that there is a large spike near the regional boundary at $x = 0.333$. A similar spike also occurs near the other regional boundary at $x = 0.667$, which is not shown in Figure 4.14. These calculation points produced approximations with relative discrepancies around 17-20%.

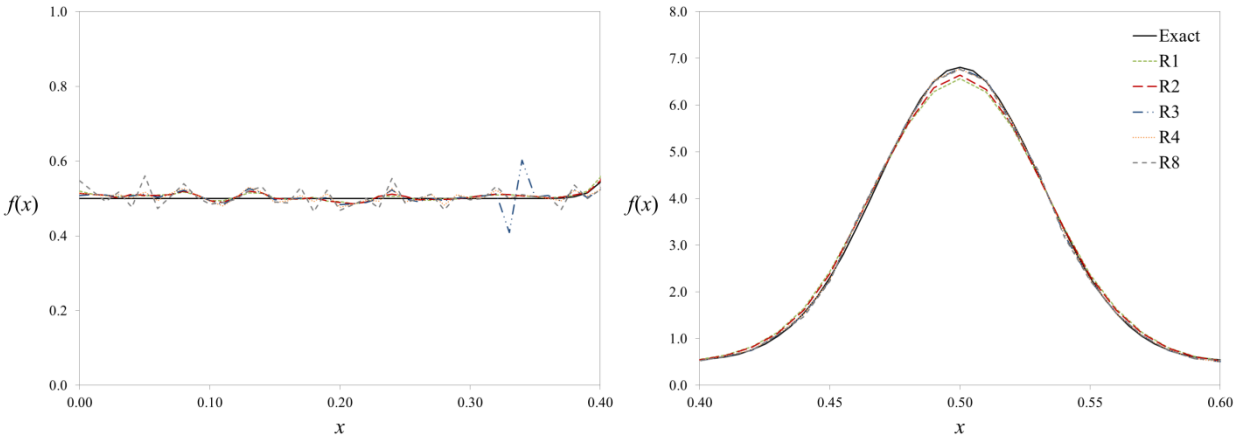

Figure 4.14 KDE approximations for one of the uniform regions (left) and near the peak (right) of a 1D mixture distribution, based on using different regional bandwidths.

The large spike shown in Figure 4.14 for R3 is not caused by increased variance due to using a bandwidth that was too small. If this were true, then the fluctuation would occur throughout the left uniform region like it does with the R8 case. In addition, since the bandwidth used for R3 was twice as big as the one used for R8, those fluctuations would be smaller in magnitude. The reason that these large spikes are only noticeable for R3 and not for any of the other cases is because there is a greater difference in the bandwidths used in adjacent regions. To see the impact that this difference has on results near the regional boundaries, a bandwidth sensitivity analysis was performed on the middle region. Keeping the bandwidth for the outer regions fixed at $h_1 = h_3 = 0.015$, the bandwidth for the middle region was varied from $h_2 = 0.001$ to 0.030. Approximations for $f(x)$ were then obtained using each of these bandwidths at key calculation points – including $x = 0.20$, 0.33,0.34, 0.50, 0.66, 0.67, and 0.80. Of these seven calculation points, only $x = 0.20$ and $x = 0.80$ were invariant to the bandwidth in the middle region. Both of these cases consistently produced results with relative discrepancies of 3.1% and 0.49% respectively. Figure 4.15 plots the bias for the other five calculation points versus bandwidth in the middle region.

Based on the bias approximation from Section 4.1, it was expected that $x = 0.33$, 0.34, 0.66, and 0.67 would be invariant to the bandwidth, whereas $x = 0.5$ would be proportional to $O(h^2)$. However, $x = 0.5$ was the only one that was consistent with theoretical expectations. For all other cases the bias got worse as the difference in bandwidth between the two regions got larger. Calculation points in the uniform regions (i.e., $x = 0.33$ and $x = 0.67$) underestimated $f(x)$ when the bandwidth in the middle region is lower, but overestimated $f(x)$ when it is higher. The opposite occurs for calculation points in the middle region (i.e., $x = 0.34$ and $x = 0.66$). Only

when the bandwidth approaches $h = 0.015$ for both regions is there minimal bias in the results. This is equivalent to using a global bandwidth, but requires more effort to compute the optimal values used in the different regions.

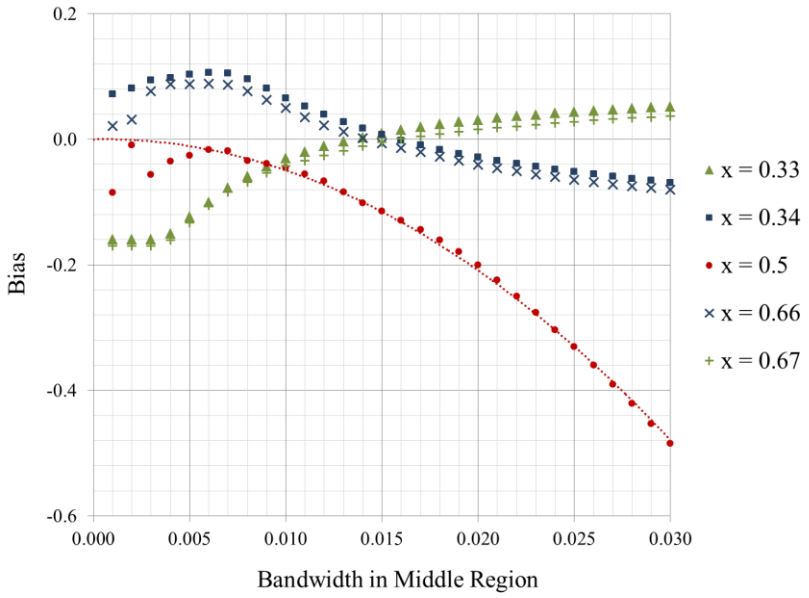

Figure 4.15 Bias in the KDE approximations for some key calculation points of a 1D mixture distribution, based on a regional bandwidth associated with the observation points.

The reason why the bandwidth in the middle region has such a significant impact on calculation points near regional boundaries is because their approximations depend on two bandwidth values instead of one. Figure 4.16 plots a comparison of the contributions that were computed for $x = 0.33$ and $x = 0.34$ for all of the observations that occurred near the regional boundary at $x = 0.333$. The light grey values reflect contributions that were computed using the same bandwidth in the two adjacent regions, whereas the dark grey values reflect contributions that were computed using different bandwidths. Both the kernel evaluations $K(u)$ and the scaled kernel contributions $K(u) / h$ are shown. On the left of the boundary, all observations add exactly

the same contribution that they would if the same bandwidth were used in all regions. However, on the right of the boundary, using a much lower bandwidth in the middle region substantially changes the sum of the scaled kernel contributions. For $x = 0.33$, the results are underestimated because this sum is much less than the sum of contributions based on the same bandwidth. For $x = 0.34$, the results are overestimated because this sum based on different bandwidths is much greater. The difference between the two cases is mostly due to the $1/h$ factor, since all kernel contributions for the Epanechnikov kernel are less than or equal to 0.75. Therefore, it seems as though associating regional bandwidths with the observation points is not effective for points near regional boundaries. In fact, the behavior that occurs is similar to the boundary bias effect described in Chapter 2, which indicates that some form of boundary correction may be needed. This could add a significant computational cost if too many regions are used.

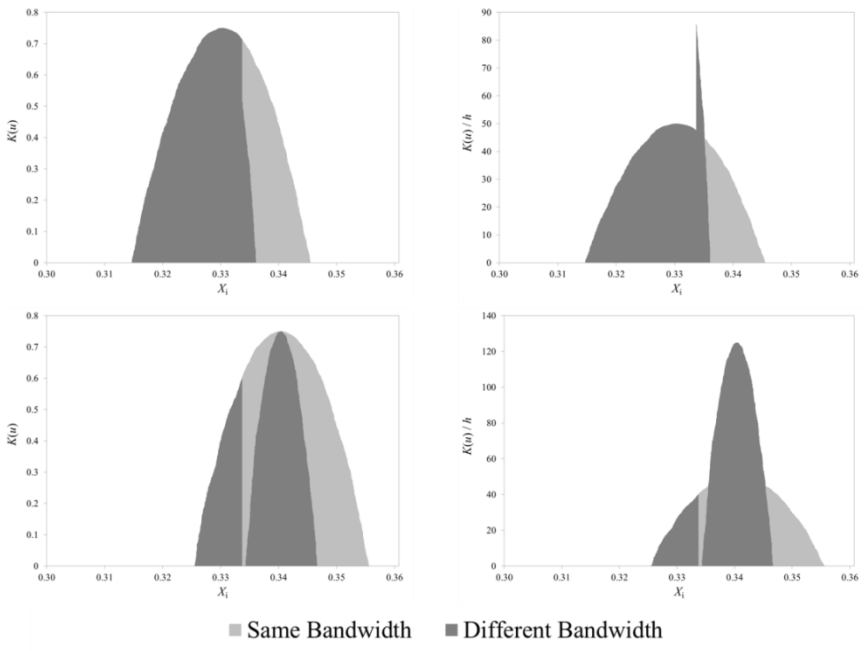

Figure 4.16 Comparison of the contributions added for all X_i to the KDE approximations for calculation points $x = 0.33$ (top) and $x = 0.34$ (bottom) near a regional boundary at $x = 0.333$.

4.3.4 Regional Bandwidth at Calculation Points

Instead of associating regional bandwidths with the observation points, it is also possible to associate them with the calculation points. This alternate version of the 1D kernel density estimator can be defined by:

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h_{R,x}}K\left(\frac{x - X_i}{h_{R,x}}\right), \tag{4.23}$$

where $h_{R,x}$ is the optimal bandwidth value for the region in which the calculation point x is located. These regional bandwidths are still computed using Eq. 4.21 based on the observation points. However, Eq. 4.23 is expected to eliminate the large spikes in bias that occurred near the regional boundaries because each calculation point uses the same bandwidth for all of its kernel evaluations. To show that this hypothesis is indeed true, the sensitivity of the regional bandwidth used in the middle region of the R3 configuration was once again analyzed using the same procedure described in the previous section. The only difference between this experiment and the previous one was that the regional bandwidths were associated with the calculation points rather than the observation points.

Figure 4.17 shows how the bias of the approximations at calculation points x = 0.33, 0.34, 0.5, 0.66, and 0.67 is affected by changing the bandwidth in the middle region. Compared to Figure 4.15, these results are much more consistent with theoretical expectations. Once again, the calculation point x = 0.5 produced approximations with bias values that were proportional to $O(h^2)$. In fact, note that the results shown in Figure 4.17 for x = 0.5 are exactly the same as the ones shown in Figure 4.15. The opposite is true for the rest of the calculation points, which now

appear to be invariant to the bandwidth in the middle region. Unlike the results shown in Figure 4.15, this is the expected outcome. Changing the bandwidth should only affect the variance of the KDE approximations at these calculation points, not the bias.

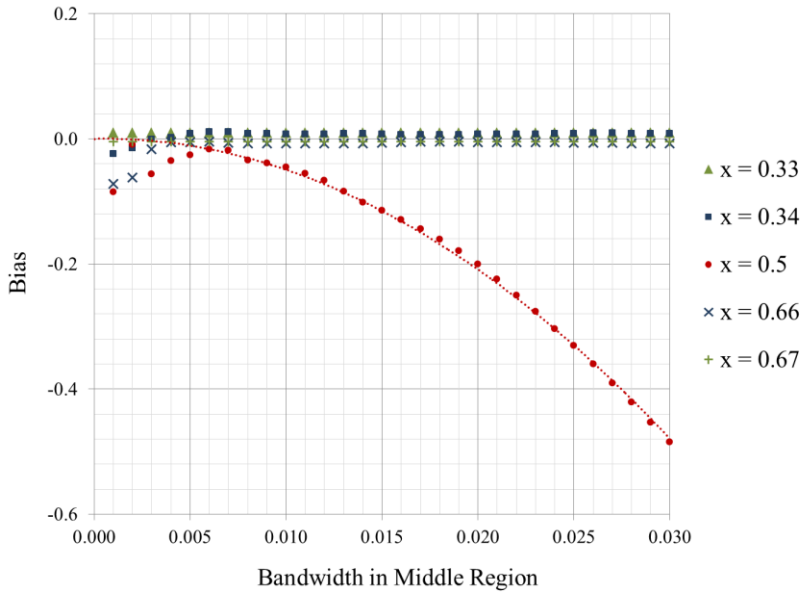

Figure 4.17 Bias in the KDE approximations for some key calculation points of a 1D mixture distribution, based on a regional bandwidth associated with the calculation points.

The advantage of associating regional bandwidths with the calculation points is that it produces better results near the regional boundaries, while still maintaining the same accuracy for the rest of the calculation points. Table 4.4 presents a summary of the mean relative discrepancies that were computed for approximations obtained using four different bandwidth selection methods. These methods include both global and regional bandwidths for the R3 configuration, as well as an "Optimal" approach. The "Optimal" approach was a best guess based on all of the results that have been presented thus far, which used the uniform-skewed bandwidth h_U for calculation points in the left and right regions, and the regional bandwidth for

R3 in the middle region. Actual bandwidth values that were used to obtain the results shown in Table 4.4 are summarized in Table 4.5 on page 102. Note also that instead of using the R3 configuration to assign calculation points to regions for the "Optimal" approach, a more visually optimal configuration was used. This configuration is equivalent to the regions that were used to compute the mean relative discrepancies for both Table 4.3 and Table 4.4.

Table 4.4 Mean relative discrepancy in the KDE approximations of a 1D mixture distribution, based on using different methods for choosing the bandwidth.

Bandwidth Method	**MEAN RELATIVE DISCREPANCY**				
	Peak (x = 0.5)	**Left Region**	**Peak Region**	**Right Region**	**All Regions**
Global	3.60%	1.40%	4.12%	2.05%	2.22%
Regional, Observation Points	0.23%	2.61%	2.05%	3.33%	2.78%
Regional, Calculation Points	0.23%	1.75%	2.05%	2.46%	2.10%
"Optimal"	0.23%	1.20%	1.91%	1.70%	1.55%

Though both regional bandwidth methods used identical bandwidth values for each region, Table 4.4 confirms that associating those values with the calculation points instead of the observation points produces more accurate results overall. Associating regional bandwidths with the calculation points reduced the relative discrepancies near the regional boundaries from 17-20% to 1-3%. This corresponded to a reduction from 2.78% to 2.10% for the mean relative discrepancy across all regions. Compared to the global bandwidth method, associating regional

bandwidths with the calculation points also produced results with approximately the same overall mean relative discrepancy. However, it did a much better job in the peak region – reducing the relative discrepancy of the maximum value at $x = 0.5$ from 3.60% to 0.23%. The "Optimal" approach takes things one step further, by increasing the bandwidth used in the uniform regions to reduce the variance. There is also a slight difference for the discrepancy in the peak region, which is due to assigning the calculation point $x = 0.6$ to this peak region rather than the right region. In general, however, without knowing anything about the distribution *a priori*, this experiment has shown that using regional bandwidths associated with the calculation points can be more effective than the global bandwidth approach.

Table 4.5 Bandwidth values used to compute the KDE approximations of a 1D mixture distribution, based on using different methods for choosing the bandwidth.

Bandwidth Method	**BANDWIDTH IN REGION *R***		
	h_1	h_2	h_3
Global	0.021807540	0.021807540	0.021807540
Regional, Observation Points	0.014620494	0.006328849	0.014631543
Regional, Calculation Points	0.014620494	0.006328849	0.014631543
"Optimal"	0.030578270	0.006328849	0.030578270

4.4 Bias Reduction

Choosing a more effective bandwidth is only one way to improve the accuracy of the KDE integral-track mesh tally. The downside to this approach is that there is no practical method for determining how accurate the approximation is with respect to the true solution without knowing what that solution looks like. Using a smaller bandwidth with a second-order kernel function will generally reduce the bias, but this improvement in bias always comes at the cost of increased variance. Although variance can be reduced by adding more particle histories, or by using variance reduction techniques, changing the bandwidth to improve accuracy is essentially equivalent to refining the mesh for a conventional mesh tally. Unlike the conventional mesh tally, however, the KDE integral-track mesh tally can also use different kernel functions to improve accuracy. The following sections describe how higher-order kernels can be used as a bias reduction tool.

4.4.1 Higher-order Kernels

The order of a kernel function $K(u)$ is a positive integer v representing its first non-zero moment, where the j^{th} moment is defined by:

$$\kappa_j = \int_{-\infty}^{\infty} u^j K(u) du. \tag{4.24}$$

Polynomial kernels such as the Epanechnikov or uniform kernel that were first introduced in Chapter 2 are of second-order ($v = 2$). In terms of Eq. 4.24, this means that κ_2 is their first non-zero moment. Higher-order kernels are defined as a kernel function whose first non-zero moment is greater than κ_2. For example, a kernel of fourth-order ($v = 4$) would be defined such

that the first three moments were equal to zero (i.e., $\kappa_1 = \kappa_2 = \kappa_3 = 0$), whereas the fourth moment κ_4 would be non-zero. Similarly, a kernel of sixth-order ($v = 6$) would require that $\kappa_1 = \kappa_2 = \kappa_3 = \kappa_4 = \kappa_5 = 0$ and $\kappa_6 \neq 0$.

To construct functions that satisfy the necessary conditions of a higher-order kernel, recall from Chapter 2 that all second-order polynomial kernels $K_s(u)$ can be obtained using a common formula (see Eq. 2.5). Extending this formula to a symmetric kernel of order $v = 2r$ can be done by multiplying $K_s(u)$ by an additional polynomial term $B_{r,s}(u)$ as follows [20]:

$$K_{2r,s}(u) = B_{r,s}(u) K_s(u),$$

where:

$$B_{r,s}(u) = \frac{\left(\frac{3}{2}\right)_{r-1}\left(\frac{3}{2}+s\right)_{r-1}}{(s+1)_{r-1}} \sum_{k=0}^{r-1} \frac{(-1)^k \left(\frac{1}{2}+s+r\right)_k u^{2k}}{k!\,(r-1-k)!\left(\frac{3}{2}\right)_k}, \tag{4.25}$$

and all of the $(x)_n$ terms are evaluated using the Pochhammer symbol defined by Eq. 2.6. While there are other formulae available for constructing higher-order polynomial kernels, Eq. 4.25 is considered to be numerically reliable for computational purposes [20].

For a better understanding of what higher-order kernels look like compared to second-order ones, three Epanechnikov kernel functions of different orders are plotted in Figure 4.18. All three cases integrate to unity, per the definition of a kernel function. However, note that while the second-order Epanechnikov kernel is non-negative throughout its domain, the higher-order Epanechnikov kernels include negative weights in their tails to account for the larger

weights at their peaks. This means that, though they still integrate to unity, higher-order kernels are no longer considered probability density functions. Though higher-order kernels can produce negative values when used with the kernel density estimator, these negative weights can also be an advantage. Previous work in the statistical literature has shown that using kernels with negative weights typically results in a faster asymptotic rate of convergence with respect to the MISE as a function of N, as well as reducing the bias in the results [5, 26]. This property makes higher-order kernels an attractive bias reduction tool for the KDE integral-track mesh tally.

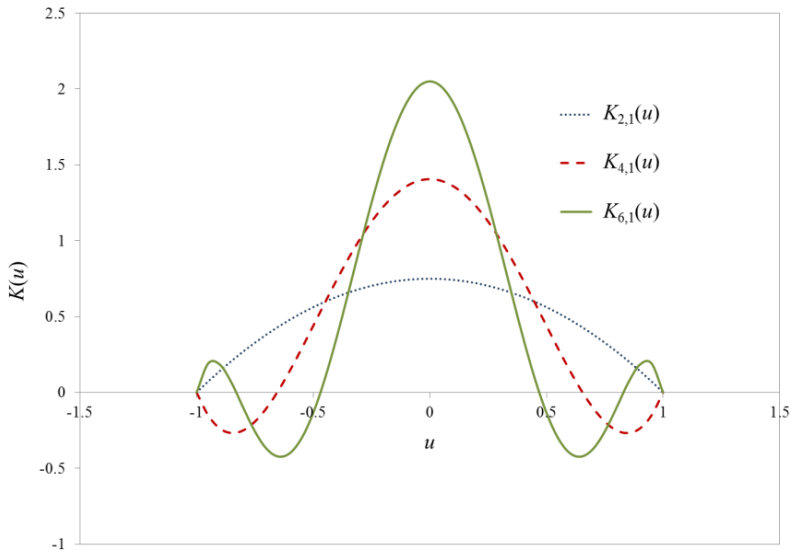

Figure 4.18 Epanechnikov kernels (s = 1) of different orders.

4.4.2 Bias Approximation for Higher-Order Kernels

Recall that the bias approximation for the 3D kernel density estimator derived in Section 4.1 assumed that only second-order kernels would be used. To extend this approximation to its more general form, note that the restriction on kernel order can be relaxed by including higher-order terms from the multivariate Taylor expansion:

$$f(x-uh_x,\, y-vh_y,\, z-wh_z) = f(x,y,z) - \sum_{j=x}^{z} h_j u_j \frac{\partial f}{\partial j} + \ldots + (-1)^{\nu} \frac{1}{\nu!} \sum_{j=x}^{z} h_j^{\nu} u_j^{\nu} \frac{\partial^{\nu} f}{\partial j^{\nu}} + \ldots \qquad (4.26)$$

Substituting Eq. 4.26 into Eq. 4.5 produces a series of terms that are multiplied by different kernel moments up to and including κ_ν. By the definition of higher-order kernels, all moments less than κ_ν will evaluate to zero and the general bias approximation becomes:

$$bias\left[\hat{f}(x,y,z)\right] \approx \frac{1}{\nu!} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \prod_{l=x}^{z} K(u_l) \left\{ \sum_{j=x}^{z} h_j^{\nu} u_j^{\nu} \frac{\partial^{\nu} f}{\partial j^{\nu}} \right\} du\, dv\, dw. \qquad (4.27)$$

As before, if the bandwidth h_j and the kernel function $K(u_l)$ are the same in all three dimensions, then Eq. 4.27 can be reduced to the following form:

$$bias\left[\hat{f}(x,y,z)\right] \approx \frac{h^{\nu}}{\nu!} \int_{-\infty}^{\infty} u^{\nu} K(u) du \left(\frac{\partial^{\nu} f(x,y,z)}{\partial x^{\nu}} + \frac{\partial^{\nu} f(x,y,z)}{\partial y^{\nu}} + \frac{\partial^{\nu} f(x,y,z)}{\partial z^{\nu}} \right). \qquad (4.28)$$

Eq. 4.28 shows that as the bandwidth h is increased, the bias in the results is expected to increase proportionally to $O(h^{\nu})$. Since h is typically small, this means that higher-order kernels should theoretically produce results that are less biased than second-order kernels.

4.4.3 Effect of Higher-Order Polynomial Kernels on Bandwidth Sensitivity

To explore the potential of using higher-order kernels as a bias reduction tool, the bandwidth sensitivity experiment described in Section 4.2 was repeated for the three different Epanechnikov kernels shown in Table 4.6. Note that the number of particle histories was increased from 1E+06 to 1E+07 to reduce the statistical errors in the results at lower bandwidth values. A discussion of the significant findings of this modified bandwidth sensitivity experiment follows.

Table 4.6 Higher-order Epanechnikov kernels (s = 1) up to r = 3.

Kernel Order	r	$\boldsymbol{K_{2r,1}(u)}$	$\int \boldsymbol{u^{2r}K_{2r,1}(u)du}$	$\int \boldsymbol{K_{2r,1}(u)^2du}$
2	1	$K_{2,1}(u) = \frac{3}{4}(1-u^2), \lvert u\rvert \le 1$	$\frac{1}{5} = 0.20$	$\frac{3}{5} = 0.60$
4	2	$K_{4,1}(u) = \frac{15}{8}(1-\frac{7}{3}u^2)K_{2,1}(u)$	$-\frac{1}{21} \approx -0.05$	$\frac{5}{4} = 1.25$
6	3	$K_{6,1}(u) = \frac{175}{64}(1-6u^2+\frac{33}{5}u^4)K_{2,1}(u)$	$\frac{5}{429} \approx 0.01$	$\frac{1575}{832} \approx 1.89$

Bias vs. Kernel Order. The first metric used to analyze the new tally data was the bias between the estimated expected value and the reference solution shown in Figure 4.2. Figure 4.19 and Figure 4.20 plot this bias as a function of bandwidth for the same two mesh points from the original experiment. Before considering the two higher-order cases, first compare the bias measurements for the second-order Epanechnikov kernel shown in Figure 4.19 and Figure 4.20 to Figure 4.3. There are two interesting conclusions that can be drawn from such a comparison. Since the number of particle histories was increased from 1E+06 to 1E+07, the error bars representing the variance are significantly smaller at lower bandwidths. As a result, the bias at the boundary point for these values was much closer to the expected O(h^2) behavior. The other interesting conclusion is that there also seems to be a slight increase in the bias for the interior point. The right plot of Figure 4.20 shows that this increase is roughly proportional to O(h^2), which indicates that although the bias at the interior point is less sensitive to the bandwidth than the boundary point, it is not always as close to zero as was initially observed in Figure 4.3.

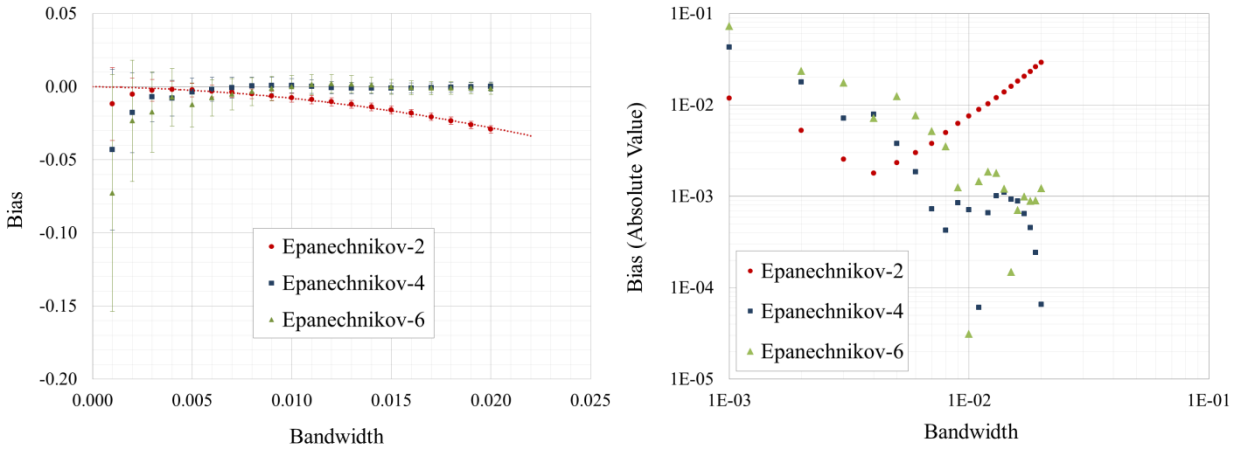

Figure 4.19 Bias in the KDE integral-track mesh tally results obtained for the boundary point using Epanechnikov kernels of different orders as a function of bandwidth.

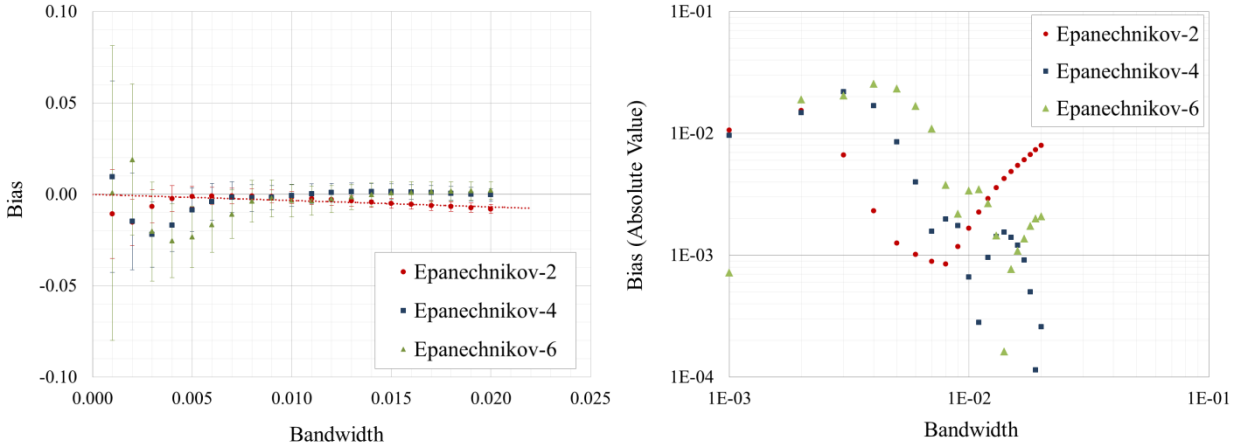

Figure 4.20 Bias in the KDE integral-track mesh tally results obtained for the interior point using Epanechnikov kernels of different orders as a function of bandwidth.

While the second-order kernel showed consistency with theoretical expectations, the two higher-order kernels did not. According to Eq. 4.28, the magnitude of the bias should increase proportionally to $O(h^v)$ for a kernel of order v. However, Figure 4.19 and Figure 4.20 show that this is not the case for either the fourth-order or the sixth-order Epanechnikov kernel. With bias measurements that are approximately zero (within statistical errors), these two kernels did not

produce results obeying the expected O(h^4) and O(h^6) behavior at all. Even though this was unexpected, it can be partially explained by considering the three distinct factors of Eq. 4.28 for kernels of different orders. Table 4.7 shows values for the two factors in this approximation that are independent of the calculation point (x, y, z). Note that the product of these factors is also shown explicitly for both the minimum and maximum bandwidths considered in the experiment.

Table 4.7 Bias approximation factors for Epanechnikov kernels of different orders.

Kernel Order	$\boldsymbol{B_1(h) = h^\nu/\nu!}$	$\boldsymbol{B_2 = \int u^\nu K_{\nu,1}(u)du}$	$\boldsymbol{B_1(h)B_2}$	
			$\boldsymbol{h = 0.001}$	$\boldsymbol{h = 0.020}$
2	$h^2/2$	$\frac{1}{5}$	1.0E-07	4.0E-05
4	$h^4/24$	$-\frac{1}{21}$	-2.0E-15	-3.2E-10
6	$h^6/720$	$\frac{5}{429}$	1.6E-23	1.0E-15

Table 4.7 shows that as the order of the kernel is increased, the product of B_1 and B_2 gets substantially smaller in magnitude. Therefore, the only way to get more bias with a higher-order kernel is if its sum of derivatives offsets this decrease in magnitude. For the test case used in the experiment, the initial bias measurements for the second-order kernel were within ±0.05 at all bandwidth values. These measurements are already quite small, being around 1.4% of the expected tally result at the center of the sphere. Multiplying these values by a factor of at least 10^{-5} makes them even smaller – to the point where their behavior with respect to the bandwidth is not visible within the statistical errors shown in Figure 4.19 and Figure 4.20. Therefore, a test case with greater initial bias is most likely needed to observe the expected O(h^ν) behavior.

Variance vs. Kernel Order. The second and final metric used to analyze the new tally data was the sample variance, which is plotted in Figure 4.21 as a function of bandwidth for the same two mesh points represented in Figure 4.19 and Figure 4.20. Note that while the higher-order kernels both produced less biased results compared to the second-order kernel, Figure 4.21 shows that this improvement in the bias still comes at the cost of increased variance. This cost is greatest for the sixth-order Epanechnikov kernel, which did not show much improvement in the bias compared to the fourth-order Epanechnikov kernel.

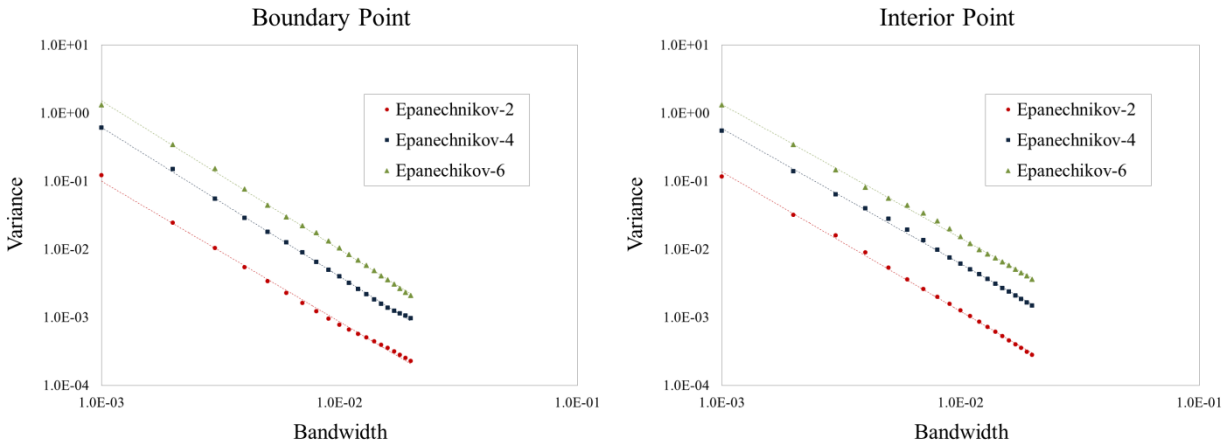

Figure 4.21 Variance in the KDE integral-track mesh tally results obtained using Epanechnikov kernels of different orders as a function of bandwidth.

While the magnitude of the variance was shown to increase with the kernel order, the rate at which this variance decreases with respect to the bandwidth is around the same for all three kernels considered in the experiment (i.e., $\sim 1/h^2$). As a result, this means that a larger bandwidth could theoretically be used to offset some of the cost of increased variance when using higher-order kernels. Since the variance can always be measured using Eq. 2.17, this approach could prove to be a useful bias reduction tool in practice for the KDE integral-track mesh tally.

4.5 Correcting the Boundary Bias Effect

The previous four sections explored the accuracy of the KDE integral-track mesh tally with respect to both bandwidth and kernel function. These two choices can affect the results at all calculation points, with the bandwidth magnitude and kernel order being the most influential factors. Recall from Chapter 2 that there is another issue that must also be addressed whenever KDE methods are used, namely the boundary bias effect. Numerous methods are available in the statistical literature for correcting this boundary bias in 1D [5, 25, 27-31], which is only needed for calculation points that are less than one bandwidth from an external geometrical boundary. However, Banerjee suggested that for Monte Carlo transport purposes the 1D boundary kernel method is preferred due to its generality, efficiency, and ease of implementation [7, 14]. A summary of this method and its extension to the 3D case is described in the following sections.

4.5.1 1D Boundary Kernel Method

In Section 4.1 it was shown that the bias of the 3D kernel density estimator should be proportional to $O(h^2)$ for second-order kernels. For calculation points near external geometrical boundaries, however, this $O(h^2)$ bias is no longer achievable because the kernel function and its moments cannot be integrated over their entire domains. The 1D boundary kernel method attempts to correct this issue by finding an alternative kernel function $K_b(u)$ that forces $O(h^2)$ bias for all calculation points. One approach is to take a linear combination of $K(u)$ and $uK(u)$, where $K(u)$ is some standard kernel function such as the Epanechnikov kernel [25]:

$$K_b(u) = (\alpha_0 + \alpha_1 u)K(u). \tag{4.29}$$

To determine the unknown coefficients α_0 and α_1, recall that the expected value of the 3D kernel density estimator can be approximated by Eq. 4.7. The 1D form of Eq. 4.7 in terms of $K_b(u)$ is:

$$\mathrm{E}\left[\hat{f}(x)\right] \approx f(x)\int K_b(u)du - h\frac{\partial f}{\partial x}\int uK_b(u)du + \frac{h^2}{2}\frac{\partial^2 f}{\partial x^2}\int u^2 K_b(u)du. \tag{4.30}$$

Therefore, for Eq. 4.30 to result in $O(h^2)$ bias, the following two constraints must be satisfied:

$$\begin{aligned} &\textbf{i.} \quad \int K_b(u)du = 1, \\ &\textbf{ii.} \quad \int uK_b(u)du = 0. \end{aligned} \tag{4.31}$$

Note that the domain u for the boundary kernel will depend on how much of the kernel falls within the domain of the problem being solved. This in turn depends on where the calculation point is located with respect to the lower or upper external boundary. Suppose that the random variable X corresponding to the unknown probability density function $f(x)$ is confined to the domain $[X_{\min}, X_{\max}]$. Using the same change of variables that was also used to derive the approximations for both bias and variance in Section 4.1, this means that the values for u must be restricted to one of the following two cases:

$$-1 \le u \le \min\left[\frac{x - X_{\min}}{h_x}, 1\right], \quad \text{or} \quad \max\left[-1, \frac{x - X_{\max}}{h_x}\right] \le u \le 1. \tag{4.32}$$

The first case corresponds to the lower boundary $X_{\min}$, whereas the second case corresponds to the upper boundary $X_{\max}$. Note that both cases can be defined in terms of a common variable p:

$$p = \left|\frac{\Delta_x}{h_x}\right|, \tag{4.33}$$

where Δ_x represents the distance from the calculation point to the nearest boundary (i.e., $x - X_{\min}$ or $x - X_{\max}$). If $p < 1$ and is defined with respect to the lower boundary $X_{\min}$, then the domain of u will be [-1, p]. Otherwise, if $p < 1$ and is defined with respect to the upper boundary $X_{\max}$, then the domain of u will be [-p, 1]. For all other cases the domain of u will be [-1, 1].

The constraints defined by Eq. 4.31 can be expressed in terms of a 2x2 matrix system to solve for the unknown coefficients:

$$\begin{bmatrix} a_0 & a_1 \\ a_1 & a_2 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{4.34}$$

where a_i are the partial moments of the standard kernel function $K(u)$:

$$a_i = \int_{u_{\min}}^{u_{\max}} u^i K(u) du. \tag{4.35}$$

Note that these partial moments are constant for a given calculation point x and observation point X_i. Therefore, solving the 2x2 matrix system results in the following linear combination for the 1D boundary kernel:

$$K_{\mathrm{b}}(u) = \frac{(a_2 - a_1 u) K(u)}{a_0 a_2 - a_1^2}. \tag{4.36}$$

For an interior point, all partial moments a_i of $K(u)$ will be defined on the domain [-1, 1]. Therefore, $a_0 = \kappa_0 = 1$, $a_1 = \kappa_1 = 0$, and $a_2 = \kappa_2$ will be non-zero for a second-order kernel. Substituting these values into Eq. 4.36 reduces $K_{\mathrm{b}}(u)$ to $K(u)$. However, if the calculation point is less than one bandwidth from any external boundary, then the partial moments of $K(u)$ will vary depending on how far away the point is from that boundary.

An example of the 1D boundary kernel based on the Epanechnikov kernel is shown in Figure 4.22 for a few values of p. Note that when $p \geq 1$, $K_b(u)$ is identical to $K(u)$ for both X_{min} and X_{max} boundaries. However, as the calculation point gets closer to the boundary, the form of $K_b(u)$ changes so that the $O(h^2)$ bias is preserved. Whereas second-order kernel functions are guaranteed to produce positive contributions, like higher-order kernel functions this is no longer always the case for the boundary kernel. As p gets closer to zero, note that there is a greater chance of negative contributions occurring. However, this is typically not an issue in practice unless $f(x)$ is approximately zero near the boundaries [7, 14].

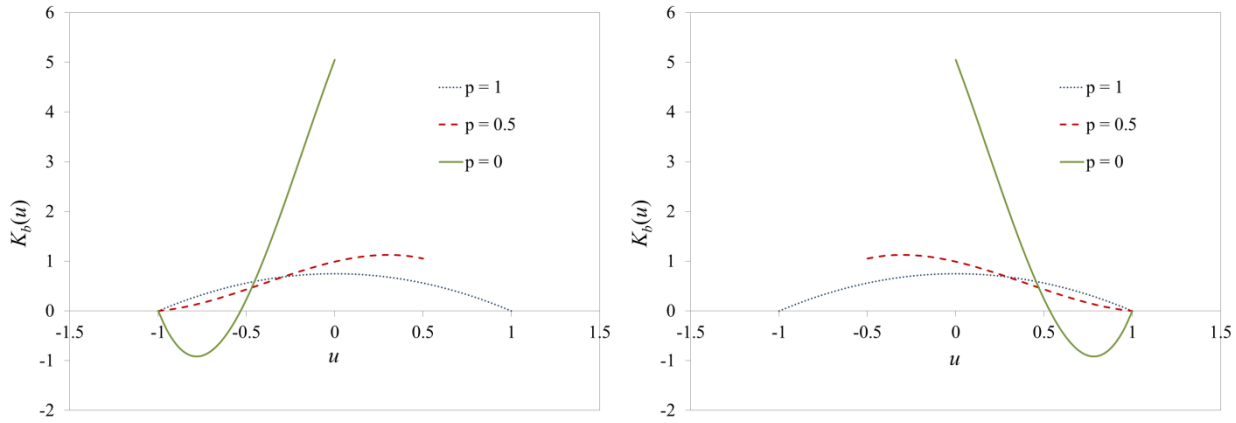

Figure 4.22 1D boundary kernels $K_b(u)$ based on the Epanechnikov kernel $K(u)$ for calculation points that are less than one bandwidth from the X_{min} (left) and X_{max} (right) boundaries.

4.5.2 3D Boundary Kernel Method

The 1D boundary kernel method just described was successfully used in previous work for 1D and 2D criticality problems defined on rectangular domains [7, 14]. For the 2D case, Eq. 4.36 was only used for coordinates of the calculation point that were less than one bandwidth from the lower or upper boundary in that dimension. All other coordinates were evaluated using the second-order Epanechnikov kernel from Table 2.1. As long as the 3D kernel function is a

product of three 1D kernels, this approach can also be extended to 3D problems defined on rectangular domains. The main challenge in using the 1D boundary kernel method for 3D problems is how to locate and tag boundary points on an arbitrary input mesh. Within DAGTally there is a prototype boundary correction tool that can be used during pre-processing for this purpose. This boundary correction tool determines the distance to the lower and upper boundaries in each dimension using the ray-fire method from the DAGMC toolkit [21]. Rays are fired in all six axis-aligned directions at all calculation points in the input mesh to find which ones are less than one bandwidth from any of the six external geometrical boundaries. Once these distances are computed, the results are then tagged to the mesh for use with any of the three KDE mesh tallies that are implemented within DAGTally.

All five test cases described in Chapter 3 were used to see if the 1D boundary kernel method worked for 3D problems by extending the mesh tally regions of interest to the boundaries. These test cases include vacuum and reflecting boundaries, as well as smooth and curved surfaces. Though the Uniform Volume Source test case was not defined on a rectangular domain, it was included in the analysis to see how effective the 1D boundary kernel method would be for a problem based on a curved domain. Table 4.8 shows a summary of how many nodes were corrected, along with the bandwidth vectors that were used with the KDE integral-track mesh tally to obtain the results. In most cases, the bandwidth was small enough so that only nodes on the surface of the input meshes needed correction. However, the Gradient Flux and Uniform Volume Source test cases also included some nodes that were not on the surface.

Table 4.8 Number of mesh nodes corrected using the 1D boundary kernel method.

Test Case	Bandwidth Vector (h_x, h_y, h_z)	Total Nodes	Corrected Nodes
Uniform Flux	(0.1, 0.0625, 0.0625)	8959	2434
Gradient Flux	(0.240632, 0.0926277, 0.092449)	8959	2884
Reflecting Boundaries	(0.148626, 0.0495376, 0.049546)	8959	2434
Material Discontinuity	(0.143026, 0.0574678, 0.0574548)	8959	2434
Uniform Volume Source	(0.00722668, 0.00721741, 0.00722104)	3551	1095

Partial solutions for the Gradient Flux, Reflecting Boundaries, and Material Discontinuity test cases are shown in Figure 4.23, both with and without boundary correction. Note that all of these results were obtained using 1E+05 particle histories. When the 1D boundary kernel method is used, the surface of the mesh now appears to be consistent with the rest of the domain. For the other two cases, however, the results were not as consistent. For the Uniform Flux test case there were 31 nodes along one row of mesh cells that reported negative tally values and statistical errors greater than 100%. Since the rest of the results were otherwise as expected, this indicates that there were fewer particle histories contributing to the approximation in this region. The Uniform Volume Source test case performed even worse, with 29 negative tally values that had statistical errors greater than 100%, and an extremely inconsistent solution across the surface of the mesh.

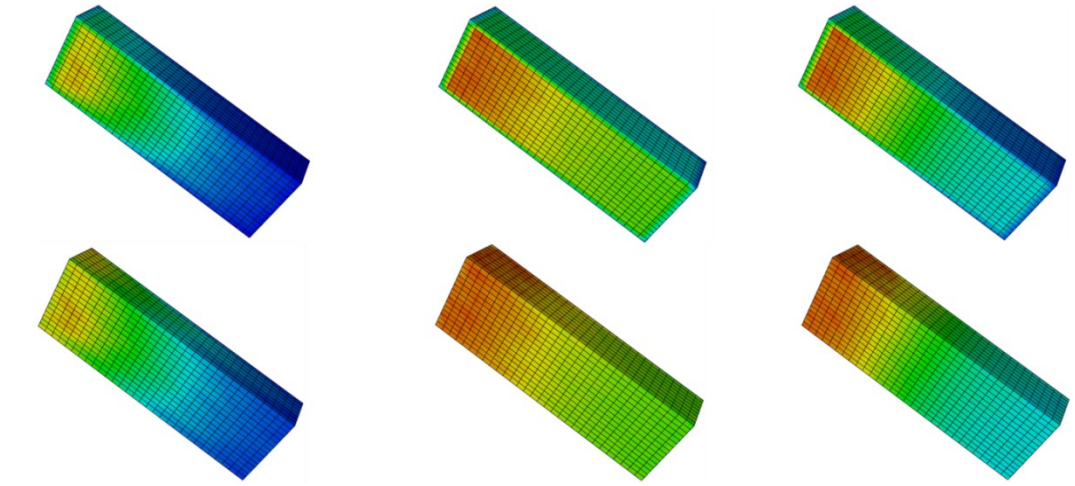

Figure 4.23 Partial solutions obtained using the KDE integral-track mesh tally without (top) and with (bottom) boundary correction based on the 1D boundary kernel method. From left to right: (*i*) Gradient Flux, (*ii*) Reflecting Boundaries, and (*iii*) Material Discontinuity.

Though the negative tally values for both the Uniform Flux and Uniform Volume Source test cases were eliminated by using more particle histories to reduce the statistical error, there is another way to improve the accuracy of the solution at the surface of the mesh. In general, the majority of calculation points near the external geometrical boundaries will only need correcting in one dimension because the bandwidth vectors that are used are typically small. However, for cases where correction is needed in more than one dimension, a more general 3D boundary kernel can be derived [32, 33]. Like the 1D case, this 3D boundary kernel is defined in terms of a linear combination:

$$K_{\mathrm{b}}(u,v,w) = (\alpha_0 + \alpha_1 u + \alpha_2 v + \alpha_3 w) K(u,v,w), \tag{4.37}$$

where it will be assumed in this dissertation that $K(u, v, w)$ is separable:

$$K(u,v,w) = K(u)\,K(v)\,K(w). \tag{4.38}$$

Four constraints are now needed to solve for the unknown coefficients α_0, α_1, α_2, and α_3. These constraints can be defined by requiring that Eq. 4.7 results in O(h^2) bias for all three dimensions:

$$\begin{aligned}
&\textbf{i.} && \iiint K_{\mathrm{b}}(u,v,w)\,du\,dv\,dw = 1,\\
&\textbf{ii.} && \iiint uK_{\mathrm{b}}(u,v,w)\,du\,dv\,dw = 0,\\
&\textbf{iii.} && \iiint vK_{\mathrm{b}}(u,v,w)\,du\,dv\,dw = 0,\\
&\textbf{iv.} && \iiint wK_{\mathrm{b}}(u,v,w)\,du\,dv\,dw = 0.
\end{aligned} \tag{4.39}$$

Note that, like the 1D boundary kernel method, these integrals depend on the portion of the kernel function $K(u, v, w)$ that falls within the domain of the problem being solved. An approximation can be used by assuming that the *uvw*-space is always rectangular, even for curved surfaces. Based on this approximation, there are now three *p*-variables in the form of Eq. 4.33 that must be computed with respect to the *x*, *y*, and *z* axes. Lower and upper boundaries must also be defined with respect to the location of the calculation point (x, y, z).

As before, the constraints defined by Eq. 4.39 can be expressed in terms of a 4x4 matrix system to solve for the unknown coefficients:

$$\begin{bmatrix} a_{000} & a_{100} & a_{010} & a_{001} \\ a_{100} & a_{200} & a_{110} & a_{101} \\ a_{010} & a_{110} & a_{020} & a_{011} \\ a_{001} & a_{101} & a_{011} & a_{002} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{4.40}$$

where a_{ijk} are products of the partial moments of $K(u)$, $K(v)$, and K(w):

$$a_{ijk} = a_i^u a_j^v a_k^w = \int_{u_{\min}}^{u_{\max}} u^i K(u)du \int_{v_{\min}}^{v_{\max}} v^j K(v)dv \int_{w_{\min}}^{w_{\max}} w^k K(w)dw. \tag{4.41}$$

These partial moments depend on the calculation point (x, y, z) and observation point (X_i, Y_i, Z_i). Since this 4x4 matrix system will need to be solved for numerous mesh nodes every time a tally event occurs, one method for doing this as efficiently as possible is described in Chapter 5.

The number of nodes that produced different results for the 1D and 3D boundary kernel methods are summarized in Table 4.9. Note that the bandwidth vectors from Table 4.8 were used to obtain the results for both methods. All four test cases defined on rectangular domains produced identical results for most of the nodes. The only exceptions occurred with nodes that were near the edges of the rectangular prism, where correction was needed in more than one dimension. For the Uniform Volume Source test case, however, the majority of the corrected nodes produced different results. The only nodes that produced identical results were the ones that were closely aligned to one of the three axes. These nodes only needed correction in one dimension, whereas the rest needed correction in two or more dimensions.

Table 4.9 Number of mesh nodes corrected using the 3D boundary kernel method.

Test Case	Total Nodes	Corrected Nodes	1D vs. 3D Difference
Uniform Flux	8959	2434	124
Gradient Flux	8959	2884	364
Reflecting Boundaries	8959	2434	244
Material Discontinuity	8959	2434	244
Uniform Volume Source	3551	1095	1085

Partial solutions for the Uniform Volume Source test case are shown in Figure 4.24 and Figure 4.25, based on 1E+06 and 1E+10 particle histories respectively. The results at the top of each figure correspond to the surface of the sphere, whereas the results at the bottom correspond to a cross-section through the center of the sphere. Note that the TET track mesh tally and the KDE integral-track mesh tally without boundary correction are included for reference purposes.

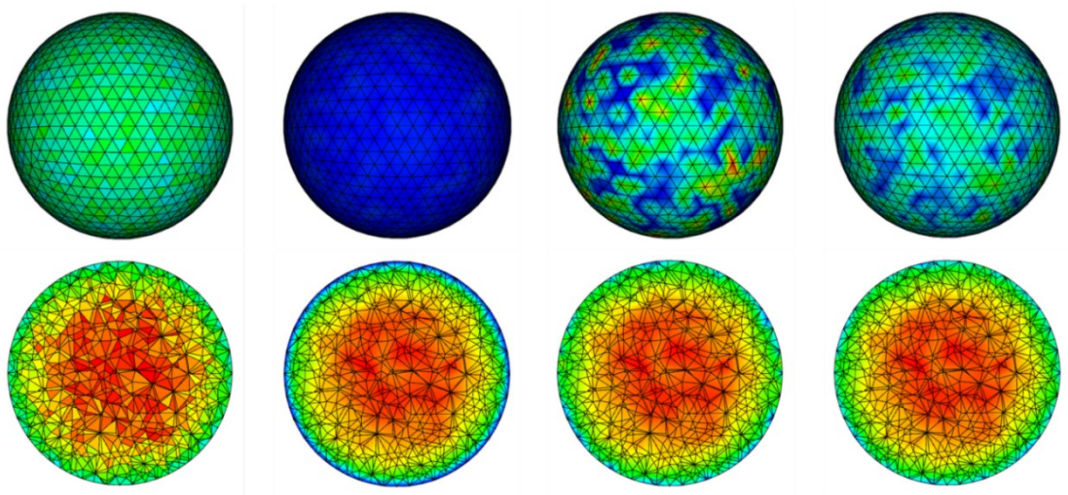

Figure 4.24 Partial solutions obtained for the Uniform Volume Source test case based on using different mesh tallies with 1E+06 particle histories. From left to right: (*i*) TET track, (*ii*) KDE integral-track without boundary correction, (*iii*) KDE integral-track with 1D boundary kernel method, and (*iv*) KDE integral-track with 3D boundary kernel method.

In both Figure 4.24 and Figure 4.25, it appears as though the 3D boundary kernel method produces more consistent results on the surface of the sphere than the 1D boundary kernel method. For all interior nodes, however, the results are identical for all three KDE integral-track mesh tally cases. This supports the theory that correction is not needed if the mesh tally region of interest is not extended to the boundaries. Compared to the TET track mesh tally, note that there is a noticeable difference in the results on the surface of the sphere. As was explained in Chapter 3, this difference is typically caused by the TET track mesh tally reporting cell-averaged

values rather than nodal-based values. The true solution in this case is expected to be lower than these cell-averaged values due to the neutron flux dropping off at the boundary. Therefore, using boundary correction with the KDE integral-track mesh tally is expected to produce a more accurate approximation on the surface of the sphere than the TET track mesh tally.

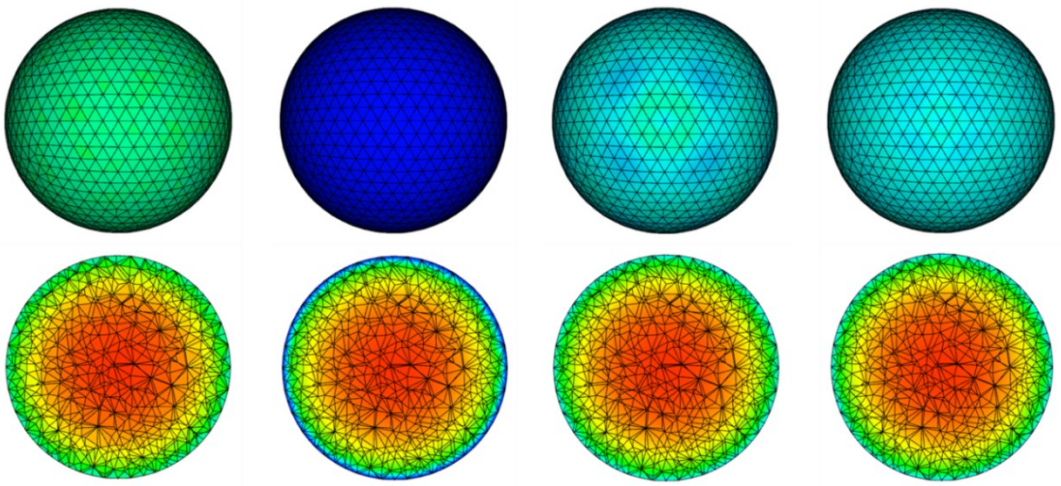

Figure 4.25 Partial solutions obtained for the Uniform Volume Source test case based on using different mesh tallies with 1E+10 particle histories. From left to right: (*i*) TET track, (*ii*) KDE integral-track without boundary correction, (*iii*) KDE integral-track with 1D boundary kernel method, and (*iv*) KDE integral-track with 3D boundary kernel method.

In addition to the visual representation, the minimum and maximum tally values and errors that were computed are summarized in Table 4.10 and Table 4.11 for a range of particle histories. Unlike the 1D boundary kernel method, no negative values with statistical errors greater than 100% were computed for the 3D boundary kernel method when only 1E+06 particle histories were used. These results show that this alternative method is generally a better option for curved surfaces in higher dimensions, especially for regions where fewer neutrons travel to the boundaries. Note, however, that the benefit of the 3D boundary kernel method gets smaller as the number of particle histories is increased. This can also be seen in the maximum tally

errors that were reported, which give some indication of the effectiveness of the boundary correction method. In contrast, if no boundary correction is included, then the lower variance reported does not accurately reflect the quality of the solution due to the increased bias.

Table 4.10 Tally values for the Uniform Volume Source test case based on different boundary correction methods and a range of particle histories.

Mesh Tally	***N* = 1E+06**		***N* = 1E+08**		***N* = 1E+10**	
	Min	**Max**	**Min**	**Max**	**Min**	**Max**
TET Track	2.280	6.238	2.801	5.611	2.817	5.580
KDE: No Boundary Correction	1.234	5.915	1.427	5.589	1.448	5.585
KDE: 1D Boundary Kernel Method	-1.602	8.205	1.932	5.589	2.268	5.585
KDE: 3D Boundary Kernel Method	0.5363	5.915	2.264	5.589	2.356	5.585

Table 4.11 Tally errors for the Uniform Volume Source test case based on different boundary correction methods and a range of particle histories.

Mesh Tally	***N* = 1E+06**		***N* = 1E+08**		***N* = 1E+10**	
	Min	**Max**	**Min**	**Max**	**Min**	**Max**
TET Track	3.15E-2	1.12E-1	3.16E-3	1.01E-2	3.16E-4	1.01E-3
KDE: No Boundary Correction	3.31E-2	5.99E-2	3.37E-3	5.57E-3	3.38E-4	5.54E-4
KDE: 1D Boundary Kernel Method	3.31E-2	3.80E+3	3.37E-3	6.79E-2	3.38E-4	6.09E-3
KDE: 3D Boundary Kernel Method	3.31E-2	9.45E-1	3.37E-3	3.06E-2	3.38E-4	2.90E-3

CHAPTER 5

Efficiency of KDE Mesh Tallies

Though it is always important to get the correct solution, efficiency can be just as important for a production Monte Carlo transport code. Efficiency can impact the performance of both pre- and post-processing tasks, as well as the actual cost of running the Monte Carlo simulation. While substantial progress has been made on the accuracy of KDE tallies, there has been little work done on assessing or improving their efficiency. Previous work has suggested that because of the finite support of most kernel functions, computing scores at all of the calculation points with every tally event is unnecessary and could become computationally burdensome [16, 18]. One proposed solution for this problem is to use a nearest-neighbor list [18]. A nearest-neighbor list approximates the true neighborhood region introduced in Chapter 2 by sorting nodes from the input mesh into a new structured mesh using their coordinates. Regions within this structured mesh are defined with lengths equal to the largest bandwidth in each dimension. Then, during the transport stage, only nodes within the same region as the collision and all adjacent regions are included when computing tally scores. Though the nearest-neighbor list can improve the efficiency of KDE collision tallies, it may not be as effective for the KDE integral-track mesh tally because the size of the neighborhood region also varies with properties of the track segments. If those track segments are long enough to cross multiple regions within the structured mesh, then even more adjacent regions will need to be considered. This could significantly increase the number of trivial tally scores that are computed.

In this chapter, topics related to the efficiency of the KDE integral-track mesh tally are explored. Since it is assumed that pre-processing tasks such as finding an optimal bandwidth and preparing the input mesh are done *a priori*, only the cost of running the Monte Carlo simulation is considered. The general performance of the KDE integral-track mesh tally is first compared to other mesh tallies in Section 5.1 for the five test cases defined in Chapter 3. Then, the KDE integral-track mesh tally implementation from DAGTally is profiled in Section 5.2 to identify the most time-consuming tasks. Finally, Section 5.3 presents the results of some scaling analyses that show how the performance of the KDE integral-track mesh tally scales with respect to different problem characteristics. Note that all results presented in this chapter were obtained using the parallel version of DAG-MCNP5 with native MCNP5 geometry on one node of the Advanced Computing Infrastructure (ACI) high-performance cluster at the University of Wisconsin - Madison. Each of these nodes has sixteen 2.2GHz CPU cores and 4 GB of RAM per core [34].

5.1 KDE Integral-track Mesh Tally Performance

Chapter 3 showed that the KDE integral-track mesh tally is capable of producing either identical or comparable results to other mesh tallies for simple fixed-source problems. In this section, the same five test cases and input parameters described in Table 3.1 were used to compare their performance. Separate DAG-MCNP5 input files were first created for the conventional mesh tally,[9] the KDE collision mesh tally, the KDE integral-track mesh tally, and the KDE sub-track mesh tally with a range of sub-tracks up to $n = 100$. Then, after running the

[9] MCNP5 mesh tallies are used for structured meshes, and the TET track mesh tally is used for unstructured meshes.

set of input files belonging to each test case on the same compute node, the time spent in the MCRUN module of DAG-MCNP5 was obtained. MCRUN is the module that runs the particle transport component of the Monte Carlo simulation [3], which is responsible for both simulating particle histories and computing tally scores.

Table 5.1 shows the time per tally point that different mesh tallies needed to produce a solution for all five test cases. Since identical input meshes were used, the number of tally points refers to cells for the conventional mesh tallies, and nodes for the KDE mesh tallies.

Table 5.1 Time spent in the MCRUN module of DAG-MCNP5 per tally point.

Test Case	TIME (MIN PER TALLY POINT)			
	Conventional	**KDE Collision**	**KDE Sub-track (n = 2)**	**KDE Integral-track**
Uniform Flux	6.51E-06	-	2.91E-04	2.91E-04
Gradient Flux	3.47E-05	2.45E-05	1.55E-04	1.47E-04
Reflecting Boundaries	2.15E-03	2.07E-03	5.63E-03	5.49E-03
Material Discontinuity	8.45E-04	7.76E-04	2.47E-03	2.39E-03
Uniform Volume Source	1.63E-03	9.90E-05	1.98E-04	1.88E-04

From Table 5.1, it seems as though the KDE collision mesh tally is the most efficient option per tally point for the input parameters used. However, the conventional mesh tally also performed well for the four test cases based on structured meshes. Compared to the conventional mesh tally, the two track-based KDE mesh tallies took around three to four times longer per tally point for the Gradient Flux, Reflecting Boundaries, and Material Discontinuity test cases. The

difference in performance was even greater for the Uniform Flux test case, primarily because the long track segments that were tallied produced larger neighborhood regions. Although the track-based KDE mesh tallies were less efficient for structured meshes, an unstructured mesh was used for the Uniform Volume Source test case. For this test case, all of the KDE mesh tallies were at least eight times faster per tally point than the conventional mesh tally.

Whereas both track-based KDE mesh tallies were generally less efficient than the conventional approach for structured meshes, and more efficient for the unstructured one, compared to each other there was a negligible difference in performance. For all five test cases, the KDE integral-track mesh tally took no longer per tally point than the KDE sub-track mesh tally with $n = 2$. Since the KDE integral-track mesh tally integrates the path length kernel defined by Eq. 2.15 to obtain each tally score, this was unexpected. However, note that the KDE sub-track mesh tally has to create and store random points along the track, as well as evaluate its 3D kernel function for each of those random points. As a result, for the same number of calculation points processed per tally event, these tasks appear to take around the same amount of time as integrating the path length kernel. The method used for performing the integration of the path length kernel will be described in Section 5.2.

Though the KDE sub-track mesh tally with $n = 2$ was only slightly less efficient than the KDE integral-track mesh tally, recall from Chapter 2 that using too many sub-tracks can become computationally burdensome. Previous work reported results for 1D and 2D examples that only used two to four sub-tracks [7, 13, 14, 15]. Figure 5.1 shows what happens to the performance of the KDE sub-track mesh tally when more than four sub-tracks are used. The dashed lines

represent equivalent times that were reported for the KDE integral-track mesh tally. In general, the initial performance of both KDE integral-track and KDE sub-track mesh tallies seems to depend on the number of contributions that are tallied. Both the Reflecting Boundaries and Material Discontinuity test cases took longer than the others because using a reflective boundary condition results in more tally events per particle history. As more sub-tracks are included, however, the time spent in the MCRUN module increases roughly linearly on a semi-log scale. The Uniform Flux case appears to be the only exception to this general trend, since it always tallies single tracks for each particle history. Though the time taken to run this test case also increases slightly with the number of sub-tracks, the actual difference in performance between the KDE integral-track and KDE sub-track mesh tallies is negligible.

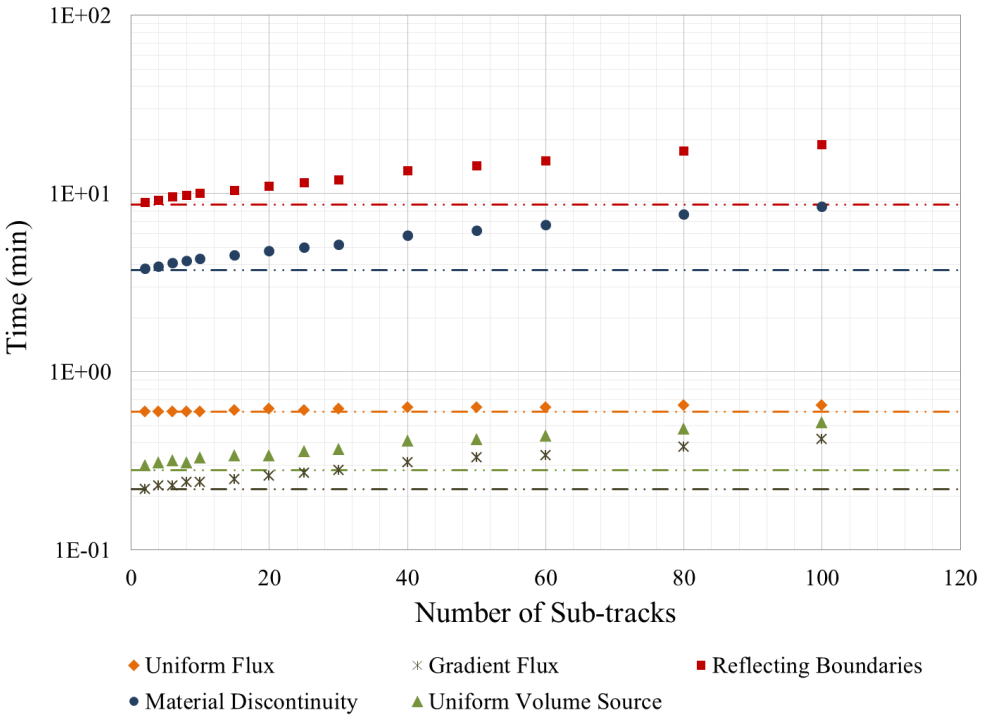

Figure 5.1 Time spent in the MCRUN module of DAG-MCNP5 for KDE integral-track and KDE sub-track mesh tallies versus the number of sub-tracks.

5.2 Performance Factors

To determine why the KDE integral-track mesh tally is less efficient than the conventional mesh tally for structured meshes, a profiling analysis was performed on the Reflecting Boundaries test case using the Callgrind tool in Valgrind [35, 36]. Of the 6.73 minutes spent in the MCRUN module, only about 22.9% was due to simulating particle histories. The other 77.1% was due to tally-specific tasks. In comparison, the conventional mesh tally only spent about 32.7% of its 1.86 minutes in the MCRUN module on tally-specific tasks. This 32.7% consisted of 29.1% for computing tally scores and adding them to the tally, and 3.6% for processing end of histories. Table 5.2 provides a breakdown of tasks performed by the KDE integral-track mesh tally, including both the number of CPU instructions and the approximate time required to complete the task. This profiling analysis shows that defining the neighborhood region is clearly the most time-consuming task. Other significant tasks include computing tally scores, and adding those scores to the final results for the tally.

Table 5.2 Breakdown of the time spent in the MCRUN module of DAG-MCNP5 for specific tasks performed by the KDE integral-track mesh tally.

Task	CPU Instructions (%)	Time (min)
Defining the neighborhood region	68.7	4.62
Computing tally scores	3.5	0.24
Adding scores to tally	2.3	0.15
Processing end of histories	0.7	0.05
Copying track data to DAGTally	0.2	0.01
Miscellaneous tasks	1.6	0.11

The following sections describe the implementation details of the neighborhood region and the method that is used to compute tally scores. The impact on the performance of including the two boundary correction methods described in Chapter 4 with the KDE integral-track mesh tally is also discussed.

5.2.1 Neighborhood Region

As shown in Table 5.2, defining the neighborhood region has the biggest impact on the performance of the KDE integral-track mesh tally. To construct the approximated neighborhood region shown in Figure 2.4, all of the nodes from the input mesh are first sorted into a k-d tree using the mesh oriented database (MOAB) library [37]. This k-d tree, which is only constructed during the tally setup stage, is a binary tree that offers O(log n) complexity on average for nearest neighbor searches [38]. Like the nearest-neighbor list, nodes are sorted according to their coordinates. Then, the neighborhood region is updated with every new tally event that occurs. Updating the neighborhood region consists of two stages. The first stage defines minimum and maximum boundaries of an axis-aligned box surrounding the track segment, while the second stage involves searching the k-d tree to find all nodes inside that box or on its surface. Only nodes that are included in the neighborhood region are passed back to the KDE integral-track mesh tally as possible calculation points. Of the two stages processed when updating the neighborhood region, the second stage is clearly the most significant factor since it accounts for around 68.4% of the 68.7% CPU instructions performed.

To define minimum and maximum boundaries for the axis-aligned box representing the neighborhood region, the path length kernel must be restricted to a finite domain. Suppose that

this domain can be described in each dimension by the interval [-1, 1], which is the case for a path length kernel constructed from 1D polynomial kernels. Being restricted to [-1, 1] means that, for example, the *x*-coordinate of the calculation point must satisfy the following constraint:

$$-1 \le \frac{x - X_{\mathrm{o}} - uS}{h_x} \le 1, \tag{5.1}$$

where the starting point of the track segment X_{o}, its direction u, and the bandwidth h_x are all fixed. Since the *x*-coordinate and path length S are the only unknowns, Eq. 5.1 can be rearranged to define values for x that produce non-zero kernel contributions for that dimension:

$$-h_x + X_{\mathrm{o}} + uS \;\le\; x \;\le\; h_x + X_{\mathrm{o}} + uS. \tag{5.2}$$

Now, note that the path length can range from $S = 0$ at the beginning of the track to $S = d_{ic}$ at the end of the track. This means that if $S = 0$, then x must be within the interval $[-h_x + X_{\mathrm{o}}, h_x + X_{\mathrm{o}}]$. Similarly, if $S = d_{ic}$, then x needs to be within the interval $[-h_x + X_{\mathrm{o}} + d_{ic}u, h_x + X_{\mathrm{o}} + d_{ic}u]$. From these two extreme cases, the minimum and maximum values of the axis-aligned box are obtained using the laws of inequalities:

$$\begin{aligned} &x \in [-h_x + X_{\mathrm{o}}, h_x + X_{\mathrm{o}}] && \text{if } u = 0, \\ &x \in [-h_x + X_{\mathrm{o}}, h_x + X_{\mathrm{o}} + d_{ic}u] && \text{if } u > 0, \\ &x \in [-h_x + X_{\mathrm{o}} + d_{ic}u, h_x + X_{\mathrm{o}}] && \text{if } u < 0. \end{aligned} \tag{5.3}$$

Defining versions of Eq. 5.3 for each of the three dimensions forms the six surfaces of the axis-aligned box used to approximate the neighborhood region. This typically produces a much smaller set of possible calculation points than including every node from the input mesh. Any nodes that fall outside this neighborhood region can be safely ignored because they will never

contribute to the tally. However, nodes inside this neighborhood region are not guaranteed to produce non-trivial contributions unless the track segments being tallied are parallel to one of the three coordinate axes. The impact of using the k-d tree search on performance with respect to the number of tally points will be explored further in Section 5.3.

5.2.2 Computing Tally Scores

After the neighborhood region has been defined, Table 5.2 shows that the next most time-consuming task for the KDE integral-track mesh tally is computing tally scores. These tally scores are determined for every node from the neighborhood region independently of the others, using Gaussian quadrature to integrate the path length kernel with respect to path length S. This means that for each integration the calculation point (x, y, z), starting point (X_o, Y_o, Z_o), direction (u, v, w), and bandwidth vector (h_x, h_y, h_z) are all treated as constants. Gaussian quadrature was used for the KDE integral-track mesh tally because it is an effective numerical integration technique that can integrate the path length kernel exactly. The order of the quadrature set needed to perform this integration can be determined during the tally setup stage once the kernel type is chosen. For example, only four quadrature points are needed to integrate a path length kernel defined as a product of three second-order Epanechnikov kernels.

Before Gaussian quadrature is performed on each node from the neighborhood region, the KDE integral-track mesh tally will first check for valid integration limits. Since the path length kernel is a product of three 1D kernel functions, the whole integrand will be zero if any of these individual terms evaluate to zero. As a result, the integration limits will depend on some interval $S = [S_{min}, S_{max}]$ for which the path length kernel is always non-zero. This may or may

not be equivalent to the full track length, which is defined on the interval T = [0, d_{ic}]. If there is no region of overlap between S and T, then there are no valid integration limits and no more calculations are needed. This will be the case for nodes that fall outside the true neighborhood region, but inside the approximated neighborhood region. Checking for valid integration limits prevents these trivial integrations from being performed.

The procedure for defining the interval S is shown graphically in Figure 5.2. First, valid ranges for all three dimensions (S_x, S_y, S_z) are determined by solving Eq. 5.1 for the path length variable S instead of the x-coordinate of the calculation point:

$$\begin{aligned} S_x &\in [0, d_{ic}] && \text{if } u = 0, \\ S_x &\in \left[\frac{-h_x + x - X_0}{u}, \frac{h_x + x - X_0}{u} \right] && \text{if } u > 0, \\ S_x &\in \left[\frac{h_x + x - X_0}{u}, \frac{-h_x + x - X_0}{u} \right] && \text{if } u < 0. \end{aligned} \tag{5.4}$$

Then, once valid ranges have been obtained for all three dimensions, the next step is to combine them into a common region of overlap to define S_{min} and S_{max}. The lowest value of this overlapping region becomes S_{min}, whereas the highest value becomes S_{max}. After S has been defined, the final step is to compare it to the full length of the track segment. If S lies completely within T, such as in Figure 5.2, then S_{min} and S_{max} become the lower and upper integration limits respectively. If S_{min} is less than zero, however, then the lower integration limit is set to zero. Similarly, if S_{max} is greater than d_{ic}, then the upper integration limit is set to d_{ic}. Once S_{min} and S_{max} have been determined, quadrature points are then chosen within this interval to ensure that no errors are introduced during the integration step.

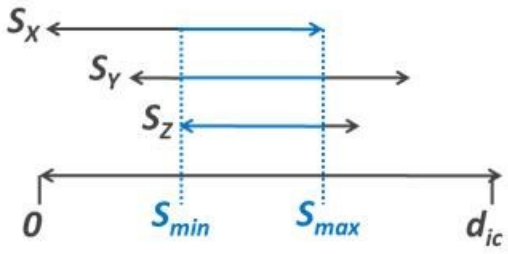

Figure 5.2 Method for determining the integration limits of the path length integral.

Since Gaussian quadrature is only performed for nodes with valid integration limits, these nodes represent calculation points that belong to the true neighborhood region and are therefore guaranteed to produce non-trivial contributions. As a result, it is expected that the performance of the KDE integral-track mesh tally may be affected by the complexity of the path length kernel that is integrated with each set of valid integration limits. For a path length kernel constructed from 1D polynomial kernels of smoothness s and order $2r$, the total number of arithmetic operations (denoted by *ops*) can be approximated by:

$$ops \approx 5 + l\left(22 + 3K_{\text{ops}}\right), \tag{5.5}$$

where K_{ops} is the number of arithmetic operations needed to evaluate each 1D polynomial kernel, and the number of quadrature points l is determined by:

$$l = 3(s + r) - 2. \tag{5.6}$$

Table 5.3 summarizes the values of *ops* and l needed to integrate a path length kernel constructed from the 1D kernel functions that have been discussed in this dissertation. All of these options result in a polynomial of order $6(s + r - 1)$, which is integrated exactly because Gaussian quadrature can integrate all polynomials of order $2l - 1$ or less. Note that the values shown for

K_{ops} do not include any arithmetic operations needed to form the constant multiplier or coefficients from Eq. 4.25. Since these values are all computed during the tally setup stage, they do not impact the performance of the KDE integral-track mesh tally during the transport stage. This allows each 1D kernel evaluation to be performed as efficiently as possible.

Table 5.3 Number of arithmetic operations needed to integrate a path length kernel based on different 1D kernel functions using Gaussian quadrature with l quadrature points.

Kernel Name	***s***	***r***	***l***	$\boldsymbol{K_{ops}}$	***ops***
Uniform	0	1	1	0	27
Epanechnikov-2	1	1	4	3	129
Epanechnikov-4	1	2	7	8	327
Epanechnikov-6	1	3	10	12	585
Biweight	2	1	7	4	243
Triweight	3	1	10	5	375

Although Gaussian quadrature was chosen as the integration technique for the KDE integral-track mesh tally, another option is to use analytical solutions for specific kernel functions instead. However, using analytical solutions can be significantly less efficient and are harder to generalize. For example, consider a path length kernel based on the second-order Epanechnikov kernel, which requires that a sixth-order polynomial be evaluated for every calculation point. The anti-derivative of this path length kernel is a complex seventh-order polynomial that takes over 120 arithmetic operations just to evaluate the largest of its eight terms

once. Since the fundamental theorem of calculus requires that this anti-derivative be evaluated twice, this means that substantially more than 240 arithmetic operations would be needed. In comparison, Table 5.3 shows that using Gaussian quadrature for this path length kernel only requires 129 arithmetic operations to produce identical results. The impact of using different kernel functions on performance will be presented in Section 5.3.

5.2.3 Boundary Correction

Another task not shown in Table 5.2 that can also affect the performance of the KDE integral-track mesh tally is applying a boundary correction. Both 1D and 3D boundary kernel methods described in Chapter 4 compute a correction factor for every tally score added to a boundary point. Recall that a boundary point is any calculation point that is less than one bandwidth from an external geometric boundary. Since interior points are not affected, the increase in time spent in the MCRUN module is indicative of the efficiency of the boundary correction method being used. Any difference in performance between 1D and 3D boundary kernel methods, however, also depends on how many dimensions need correcting. Both methods must compute p using Eq. 4.33 for each of these dimensions, as well as the partial moments a_0, a_1, and a_2. Note that the j^{th} partial moment of a 1D polynomial kernel can be integrated exactly using Gaussian quadrature with $\lfloor s + r + j/2 \rfloor$ quadrature points.

Once values for p and the partial moments are known, the only difference between 1D and 3D boundary kernel methods is how the correction factor is computed. Computing this correction factor for the 1D method only requires seven additional arithmetic operations for every dimension that needs correcting. In contrast, the 4x4 matrix system defined by Eq. 4.40

must be created and then solved to obtain the unknown coefficients α needed for the 3D method. To do this as efficiently as possible, the KDE integral-track mesh tally uses the symmetric matrix solver from the linear algebra package LAPACK [39]. In addition, the efficiency of the 3D method is also improved by reducing the order of the 4x4 matrix system to match the number of dimensions that actually need correcting. As shown in APPENDIX D, if correction is only needed in one dimension, then Eq. 4.40 reduces to the 2x2 matrix system defined by Eq. 4.34. This means that both the 1D and 3D boundary kernel methods end up using the same method for computing the correction factor. Similarly, if correction is only needed in two dimensions, then Eq. 4.40 reduces to a 3x3 matrix system representing a 2D boundary kernel. Using this laddered approach improves the performance of the 3D boundary kernel method when fewer nodes need correcting in all three dimensions.

A simple performance comparison for the different boundary correction methods based on the Reflecting Boundaries test case is summarized in Table 5.4. Note that the input mesh for this comparison was extended to the geometrical boundaries so that boundary correction would be needed, which resulted in 2434 of 8959 nodes being classified as boundary points. When no boundary correction was used, the time spent in the MCRUN module was 33.99 minutes, or 0.228 seconds per tally point. Including boundary correction only took 0.233 seconds per tally point for the 1D method, and 0.242 seconds per tally point for the 3D method. Though this example shows only a small difference between 1D and 3D boundary kernel methods, note that the efficiency of either method will also depend on how many calculation points are classified as boundary points. The impact on the performance of the KDE integral-track mesh tally with respect to the number of boundary points will be explored further in Section 5.3.

Table 5.4 Performance of the KDE integral-track mesh tally both with and without boundary correction for the Reflecting Boundaries test case.

Boundary Correction	MCRUN (min)	TOTAL (min)
None	33.99	37.02
1D Boundary Kernel Method	34.82	37.80
3D Boundary Kernel Method	36.20	39.68

5.3 Scaling Analysis

The previous sections described implementation details for three of the most time-consuming tasks performed by the KDE integral-track mesh tally. In this section, its efficiency is compared to other mesh tallies by performing a scaling analysis on the Reflecting Boundaries test case with respect to a number of different problem characteristics. One of the most significant problem characteristics for both KDE and conventional mesh tallies is total particle histories. However, other problem characteristics that can also affect the performance of the KDE integral-track mesh tally include kernel complexity and three different lengths: mesh resolution, bandwidth vector magnitude, and mean free path of the neutrons. The combination of these three characteristic lengths are responsible for determining the average number of nodes from the input mesh that will typically be included in a characteristic neighborhood region. Since the interaction between these three characteristic lengths is expected to be complex, as a first step this dissertation only considers mesh resolution and bandwidth vector magnitude as single effects for the Reflecting Boundaries test case. The mean free path for this problem is fixed at 0.53659 cm.

The default problem characteristics for the Reflecting Boundaries test case include:

- 1E+05 particle histories
- Input mesh with 864 cells and 1225 nodes (6 cm x 2 cm x 2 cm)
- Optimal bandwidth vector of (0.15, 0.05, 0.05)
- Second-order Epanechnikov kernel
- Neighborhood region approximated by k-d tree search method
- No boundary correction method
- Three sub-tracks (KDE sub-track mesh tally only)

Note that the input mesh is extended to the geometrical boundaries so that the impact of boundary correction on performance can also be explored. A series of DAG-MCNP5 input files using native MCNP5 geometry were created for each scaling analysis from the base case described above, with the primary difference being the key problem characteristic being analyzed. Each scaling analysis was run on the same compute node to obtain the total time spent in the MCRUN module. Before the results of the scaling analyses are presented, however, a general timing model for the KDE integral-track mesh tally will now be proposed.

5.3.1 General Timing Model

A general model for the time t_{ij} needed for the KDE integral-track mesh tally to process track j of history i can be described as:

$$t_{ij} = a_n + b_p V(d_{ij}, h)\rho_p, \tag{5.7}$$

where a_n is the average time for tracking a single track, b_p is the average time for calculating the contribution from a single track to a single tally point p, d_{ij} is the vector of track j of history i, h is the bandwidth vector, $V(d_{ij}, h)$ is the volume of the neighborhood region for a track of length $|d_{ij}|$ and bandwidth vector magnitude $|h|$, and ρ_p is the density of tally points. If there are no tally points, then t_{ij} will always equal a_n because the tracking cost for track j is independent of the tally behavior. Including tally points by varying ρ_p adds an additional cost to t_{ij} that is equivalent to the product of b_p and the number of tally points expected to be included in the neighborhood region, which in turn depends on both $V(d_{ij}, h)$ and ρ_p. The total time is obtained by taking the sum of Eq. 5.7 over all tracks j for all histories i from 1 to N.

Since only the total time spent in the MCRUN module can be measured using DAG-MCNP5, only average behavior with respect to the different problem characteristics can be reported. Furthermore, as the total volume V_{TOT} of the Reflecting Boundaries test case is fixed, ρ_p will also vary linearly with the total number of tally points, T. This means that Eq. 5.7 can also be expressed as a direct function of T:

$$t_{ij} = a_n + b_p V(d_{ij}, h)\frac{T}{V_{\text{TOT}}}. \tag{5.8}$$

Given a specific problem with fixed problem characteristics, the terms a_n, b_p, and T in Eq. 5.8 should be approximately the same for all of the tracks that are tallied. However, $V(d_{ij}, h)$ can vary significantly with each track j depending on both $|d_{ij}|$ and $|h|$. Assuming that the tally point density ρ_p is the same in all directions, there are two extreme cases that could potentially occur. The first case, which occurs when $|d_{ij}| << |h|$, will produce a volume $V(d_{ij}, h)$ that varies roughly

proportionally to $|h|^3$. The second case, which occurs when $|d_{ij}| >> |h|$, will produce a volume $V(d_{ij}, h)$ that varies according to the orientation of d_{ij}. For an axis-aligned track, $V(d_{ij}, h)$ will vary as $|d_{ij}|$. For d_{ij} within an axis plane, $V(d_{ij}, h)$ will vary as $|d_{ij}|^2$. Finally, for d_{ij} aligned to the vector (1, 1, 1), $V(d_{ij}, h)$ will vary as $|d_{ij}|^3$. Although most tracks will never be exactly one of these different cases, for a specific problem there will be some average value for $V(d_{ij}, h)$ that represents the size of a characteristic neighborhood region. This characteristic neighborhood region can be considered fixed as other problem characteristics such as ρ_p or N are varied.

In the following sections, all timing results presented with no boundary correction for the KDE integral-track mesh tally represent the sum of Eq. 5.8 over all tracks j for all histories i from 1 to N. Most of these results are also reported per tally point, which allows a direct comparison to be made to other mesh tally implementations that use the same input mesh, but a different set of tally points (i.e., the MCNP5 mesh tally).

5.3.2 Total Particle Histories

The first problem characteristic considered for a scaling analysis was total particle histories. Adding more particle histories leads to more tally events, which results in more tally scores being computed. In terms of the timing model described by Eq. 5.8, this simply increases the number of tracks j and histories i that will be included in the total sum for the KDE integral-track mesh tally. The average time needed per track will be about the same no matter how many particle histories are used. Beginning with the base case, the average number of collisions that occurred was 234 per history. The time per tally point needed to process these collisions is shown in Figure 5.3 versus total particle histories for the MCNP5, KDE integral-track, and KDE

collision mesh tallies. The KDE sub-track mesh tally is not shown because it produced similar results to the KDE integral-track mesh tally.

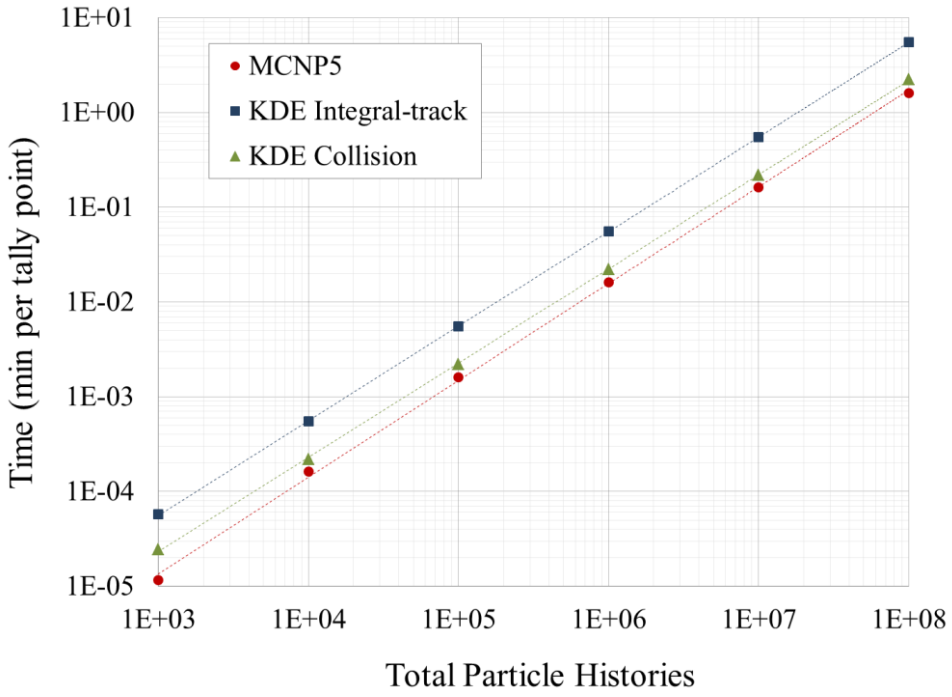

Figure 5.3 Time spent in the MCRUN module of DAG-MCNP5 per tally point versus total particle histories.

Figure 5.3 shows that all KDE mesh tallies scale with particle histories in a similar manner to the MCNP5 mesh tally. This relationship is proportional to a power law (i.e., ax^k), with k equal to around 0.994 for the KDE collision mesh tally, 0.9981 for the KDE integral-track mesh tally, and 1.0206 for the MCNP5 mesh tally. These results show that, in general, if the particle histories are increased by a factor of 10, then the time spent in the MCRUN module will also increase by around a factor of 10. The most significant difference is the magnitude of a, which is 1E-08 for the MCNP5 mesh tally, 2E-08 for the KDE collision mesh tally, and 6E-08 for the KDE integral-track mesh tally. This indicates that the KDE integral-track mesh tally

should take at most six times longer than the MCNP5 mesh tally per tally point for a fixed number of particle histories, which is consistent with the results from Section 5.1. Note, however, that the extent of this difference in the magnitude of a will also depend on other problem characteristics that can affect the average time needed per tally event.

5.3.3 Mesh Resolution

The second problem characteristic considered for a scaling analysis was mesh resolution. Refining the resolution of a structured input mesh for a fixed domain size reduces the spacing between the mesh cells and increases the node density. For the timing model described by Eq. 5.8, increasing the node density corresponds to an increase in the total number of tally points, T. This means that more tally points could potentially be included as calculation points within the fixed volume $V(d_{ij}, h)$ of each neighborhood region, which in turn increases the time that it takes to process each track j. For the following analysis, T was varied by systematically refining the resolution of the input mesh in all three dimensions for a fixed domain size. A description of the important properties of the input meshes that are included can be found in Table 5.5. Note that surface nodes refer to the number of nodes on the outer skin of the mesh, whereas boundary points refer to nodes that are less than one bandwidth from that outer skin.

Table 5.5 Properties of the input meshes used for the scaling analysis on the number of tally points.

Cells	Nodes	Surface Nodes	Boundary Points
4 (4 x 1 x 1)	20	20	20
32 (8 x 2 x 2)	81	74	74

Cells	Nodes	Surface Nodes	Boundary Points
108 (12 x 3 x 3)	208	164	164
256 (16 x 4 x 4)	425	290	290
500 (20 x 5 x 5)	756	452	452
864 (24 x 6 x 6)	1225	650	650
1372 (28 x 7 x 7)	1856	884	884
2048 (32 x 8 x 8)	2673	1154	1154
2916 (36 x 9 x 9)	3700	1460	1460
4000 (40 x 10 x 10)	4961	1802	1837
32,000 (80 x 20 x 20)	35,721	7202	8208
500,000 (200 x 50 x 50)	522,801	45,002	102,344
4,000,000 (400 x 100 x 100)	4,090,601	-	-
6,912,000 (480 x 120 x 120)	7,042,321	-	-
13,500,000 (600 x 150 x 150)	13,703,401	-	-

Figure 5.4 plots the time per tally point versus *T* for MCNP5, KDE integral-track, and KDE collision mesh tallies.[10] Like the previous scaling analysis, the KDE sub-track mesh tally is not shown as it produced similar results to the KDE integral-track mesh tally. Although results from the previous analysis indicated that all mesh tallies scale with total particle histories

[10] *T* is the number of mesh cells for the MCNP5 mesh tally, and the number of mesh nodes for all KDE mesh tallies.

at approximately the same rate, Figure 5.4 shows that this is not the case when T is varied by changing the resolution of the input mesh for a fixed domain size. This difference in behavior can be explained by referring to the timing model described by Eq. 5.8. Increasing total particle histories only adds more tracks that will be tallied, which produces an approximately linear increase when all other problem characteristics are fixed. In contrast, the impact of increasing T will primarily depend on b_p, the average cost for processing each tally score. For structured meshes this cost is much greater for KDE mesh tallies than it is for the MCNP5 mesh tally.

Since the MCNP5 mesh tally has such a low cost per tally score, the majority of its time is spent tracking particle histories. If the total time spent tracking N particle histories is denoted by a_N, then the time per tally point should be approximately equivalent to a_N/T. Although Figure 5.4 shows that this is true for the MCNP5 mesh tally initially, beyond around 4000 tally points the results start to deviate slightly from this a_N/T relationship because the time spent computing tally scores becomes more significant. In comparison, the cost of computing tally scores for the KDE mesh tallies is much more dominant than the cost for tracking particle histories. Therefore, dividing Eq. 5.8 by T suggests that the time per tally point for the KDE integral-track mesh tally should and does approach a constant cost as T gets larger and all other problem characteristics remain fixed. The same behavior also occurs for the KDE collision mesh tally, which has a lower overall cost than the KDE integral-track mesh tally because its neighborhood regions are typically smaller (i.e., $V(h) < V(d_{ij}, h)$) and computing each tally score for a collision requires significantly less effort. Note that Figure 5.4 also shows a slight increase from the expected results for the KDE integral-track mesh tally, which is a topic of future study that will include a deeper understanding of the interactions between the bandwidth, neighborhood, and k-d tree

performance. The key point to take from these results is that KDE mesh tallies will become less efficient than the MCNP5 mesh tally as the density of tally points increases due to actions such as refining the input mesh.

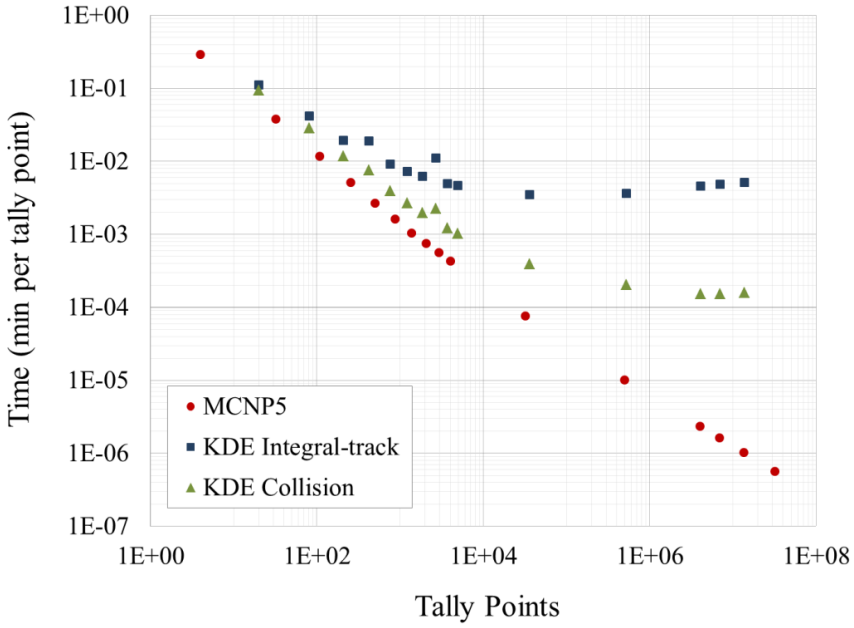

Figure 5.4 Time spent in the MCRUN module of DAG-MCNP5 per tally point versus the number of tally points.

In addition to showing that the mesh tallies scale differently with the number of tally points, Figure 5.4 also suggests that there are two notable outliers from the general trend for the KDE mesh tallies. These outliers corresponds to the input mesh with 2673 nodes, and to a lesser extent for the input mesh with 425 nodes. Both of these input meshes took longer to process than other input meshes with more nodes. To see if this deviation was caused by the method used to define the neighborhood region, the scaling analysis was repeated for input meshes up to 4000 mesh cells with the k-d tree search turned off. Turning off the k-d tree search ensured that a tally score was computed for every node with every tally event. The results for this analysis

are shown in Figure 5.5, which indicate that without the k-d tree search there is now a consistently strong linear relationship between time and *T*. As more tally points are added, however, the time spent in the MCRUN module significantly increases. The rate of this increase depends on the type of tally being used, with the KDE sub-track mesh tally being the least efficient option because it always evaluates its 3D kernel function multiple times per node. Though the KDE integral-track mesh tally also evaluates the path length kernel multiple times, recall from Section 5.2 that it only does so for nodes with valid integration limits.

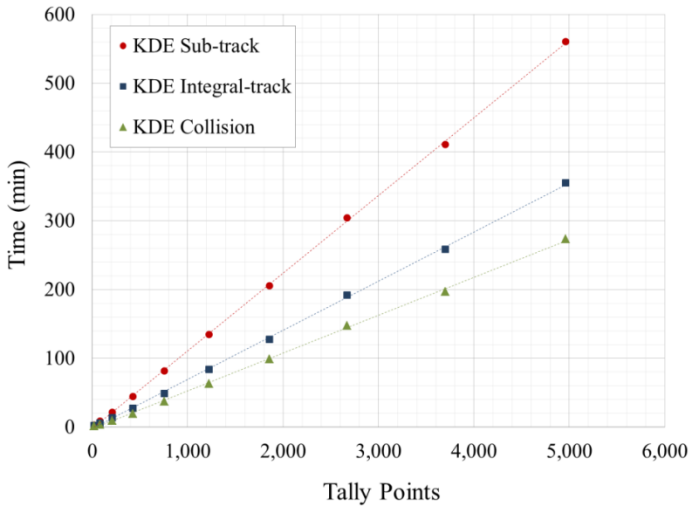

Figure 5.5 Time spent in the MCRUN module of DAG-MCNP5 versus the number of tally points, without the k-d tree search.

To see the benefit of using the k-d tree search, Figure 5.6 plots the speedup factor versus *T* for the input meshes up to 4000 mesh cells. This speedup factor initially increases as more tally points are added, indicating that the k-d tree search becomes even more efficient for meshes with a higher node density. For the input mesh with 4000 cells and 4961 nodes, the speedup factor ranges from around 15 times faster for the KDE integral-track mesh tally, to around 54

times faster for the KDE collision mesh tally. Even though the input mesh with 2673 nodes still deviates from the general trend, these results show that using the k-d tree search significantly improves the performance for all three KDE mesh tallies. Note also that without the k-d tree search, the time per tally point was approximately constant for these input meshes. For example, the KDE integral-track mesh tally only needed around 0.07 minutes per tally point. This indicates that the second term of Eq. 5.8 becomes even more dominant than the first term without the k-d tree search, primarily because $V(d_{ij}, h)$ always equals V_{TOT} for every track j that is tallied.

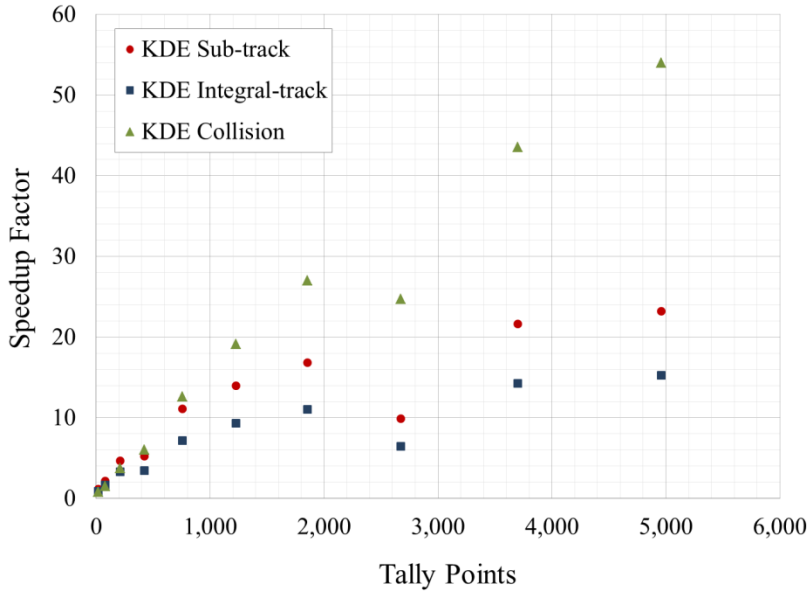

Figure 5.6 Speedup factor for using the k-d tree search to define the neighborhood region compared to computing tally contributions at every mesh node.

While using the k-d tree search clearly improves the performance of all KDE mesh tallies, recall that no boundary correction was used for any of the previous analyses. Since using boundary correction can also affect performance, the scaling analysis was repeated with the k-d tree search and 3D boundary kernel method turned on. Without this boundary correction, all

nodes that fall within any given neighborhood region will only compute a standard kernel contribution based on $K(u, v, w)$. Including boundary correction means that an additional correction factor is also computed for the boundary points in each neighborhood region by solving the 4x4 matrix system defined by Eq. 4.40. This step is done after all of the calculation points in the neighborhood region have been determined and their standard kernel computations have already been computed, which means that the only difference when including boundary correction is the computation of the correction factors. For the Reflecting Boundaries test case, correction is typically only needed in one dimension for most of the boundary points when the optimal bandwidth is used. This means that, in general, the additional time needed to compute the correction factors should increase linearly with the number of boundary points.

The ratio of the time spent in the MCRUN module with and without the 3D boundary kernel method is shown in Figure 5.7 versus the number of boundary points. Note that the number of boundary points per input mesh is summarized in Table 5.5. Figure 5.7 shows that as more boundary points are added, the ratio of the time with and without boundary correction also generally increases. For all cases up to 8208 boundary points, this increase is less than 4% for the KDE collision mesh tally, 14% for the KDE integral-track mesh tally, and 16% for the KDE sub-track mesh tally. Increasing the number of boundary points to 102,344 increases those upper limits to 14%, 17%, and 25% respectively. Since these results show that the cost of using boundary correction is non-negligible, especially for a large number of boundary points, it may be worth avoiding it whenever performance is more important than obtaining results on the boundaries. This can be done fairly easily by shrinking the input mesh so that the nodes on its surface are at a distance greater than one bandwidth from any external geometrical boundary.

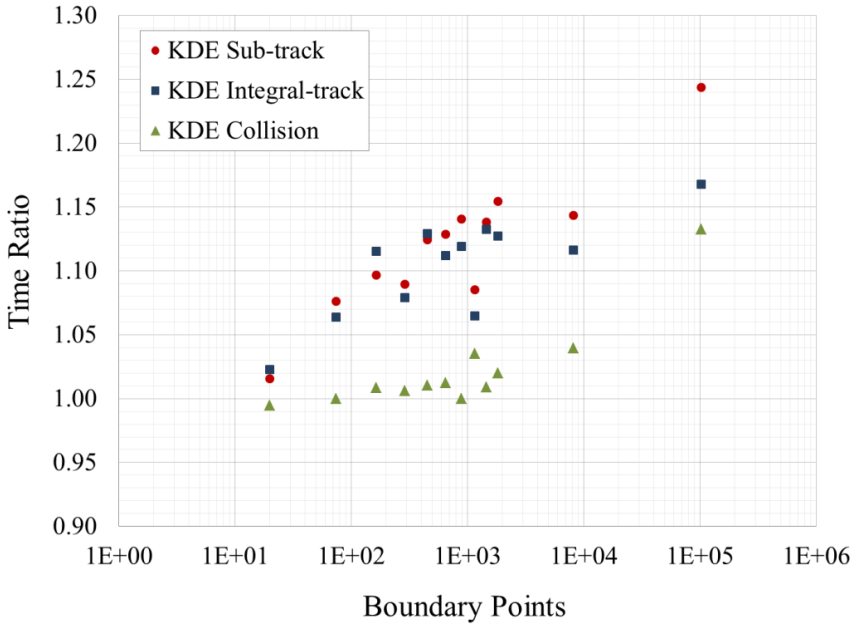

Figure 5.7 Ratio of the time spent in the MCRUN module of DAG-MCNP5 with and without the 3D boundary kernel method versus the number of boundary points.

5.3.4 Bandwidth Vector Magnitude

The third problem characteristic considered for a scaling analysis was the bandwidth vector magnitude, which only affects KDE mesh tallies. Referring back to the timing model for the KDE integral-track mesh tally described by Eq. 5.8, the bandwidth is primarily responsible for determining the volume of the neighborhood region, $V(d_{ij}, h)$. As was previously explained, the impact that the bandwidth vector will have on $V(d_{ij}, h)$ depends on its magnitude with respect to the magnitude of the vector defining the track. This particular scaling analysis represents the case where $|d_{ij}|$ is approximately the same order of magnitude as $|h|$, which means that changing the bandwidth is expected to have a noticeable impact on the total time.

Like increasing T by refining mesh resolution for a fixed bandwidth, increasing $V(d_{ij}, h)$ for a fixed mesh resolution by changing $|h|$ can also increase the number of tally points expected

to be included as calculation points per tally event. However, this number will also depend on the ratio between the mesh spacing and the bandwidth in each dimension. If the bandwidth is significantly smaller than the mesh spacing, then increasing bandwidth size will have little impact on performance because the chance of including more tally points in the neighborhood region will be small. Therefore, the input mesh with 32,000 cells and 35,721 nodes was used for the following analysis to ensure that the mesh spacing was small enough to capture any differences due to changing the bandwidth vector magnitude. This input mesh produced mesh cells of dimensions 0.075 cm by 0.1 cm by 0.1 cm, which means that more nodes in the *x*-direction have a greater chance of being included than in the *y* or *z*-directions. To measure how the choice of bandwidth vector affects the performance of KDE mesh tallies for this fixed input mesh, different values systematically ranging from the optimal bandwidth of (0.15, 0.05, 0.05) to (1.5, 0.5, 0.5) were considered. Figure 5.8 plots the resulting time per tally point as a function of the bandwidth vector magnitude, on both linear-linear and log-log scales.

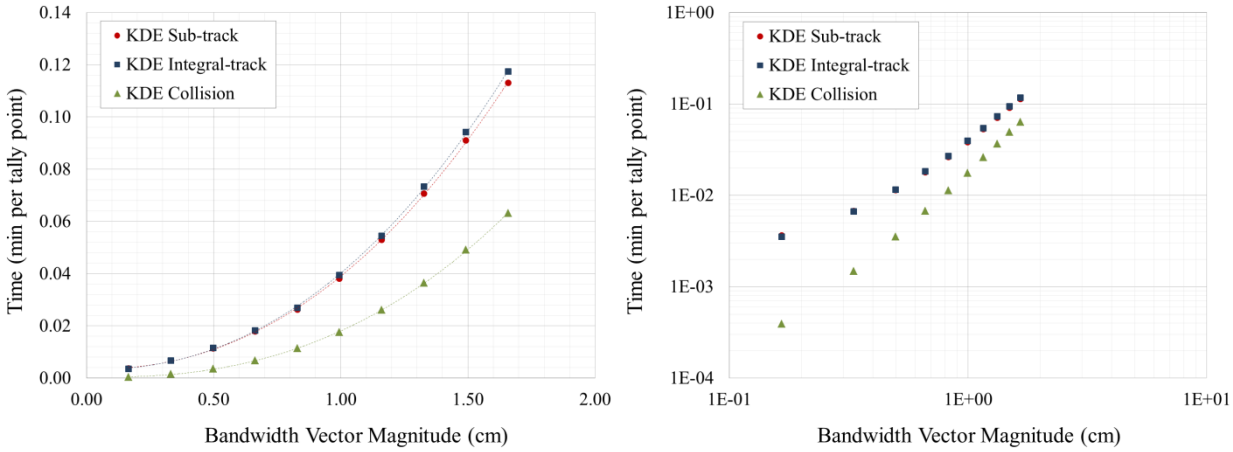

Figure 5.8 Time spent in the MCRUN module of DAG-MCNP5 per tally point versus the bandwidth vector magnitude.

Figure 5.8 shows that the time per tally point versus bandwidth vector magnitude increases according to a polynomial relationship for all three KDE mesh tallies when all other problem characteristics remain fixed. As with the previous two scaling analyses, the time per tally point with respect to bandwidth vector magnitude is also larger for the two track-based KDE mesh tallies. Though all KDE mesh tallies use the same k-d tree search to define their neighborhood regions, there are two reasons why the KDE collision mesh tally is usually more efficient. First, the neighborhood region defined for the KDE collision mesh tally is typically smaller because its size is only dependent on the bandwidth vector. Therefore, fewer calculation points will be included per tally event compared to the two track-based KDE mesh tallies. Second, the KDE collision estimator only evaluates its 3D kernel function once per tally score. Both the KDE sub-track and KDE integral-track estimators evaluate their 3D kernel functions multiple times, depending on either the number of sub-track points, or the number of quadrature points. The slight difference shown in the left plot of Figure 5.8 between the KDE integral-track and KDE sub-track mesh tallies occurs because increasing the size of the bandwidth also increases the number of calculation points with valid integration limits, which means that more integrations will be performed by the KDE integral-track mesh tally.

The polynomial that is fitted to the results shown in the left plot of Figure 5.8 is of third order, with an R^2 value of around 0.9999 in all three cases. Although this is equivalent to the expected behavior when $|d_{ij}| << |h|$, a second-order polynomial also fits the data well with the exception of the lowest bandwidth values. This conflicting behavior could be due to the fact that $|d_{ij}|$ is approximately equal to $|h|$, or because the tally point density that was used is not uniform in all directions. Using a uniform tally point density with a uniform bandwidth that is much

greater than the mean free path should produce behavior that is more consistent with the expected $|h|^3$ behavior. To help explain why this particular polynomial relationship occurs between time per tally point and bandwidth vector magnitude, Figure 5.9 plots the average number of nodes that are typically included in a neighborhood region per tally event as the bandwidth vector magnitude changes. Note that the average number of nodes was computed theoretically, which is exact for the KDE collision mesh tally. For the two track-based KDE mesh tallies, the size of the neighborhood region will differ with each track since it is a function of track length and orientation, in addition to the bandwidth. Therefore, a characteristic neighborhood was chosen based on a track with length equal to the mean free path (0.53659 cm) and traveling in the *x*-direction.

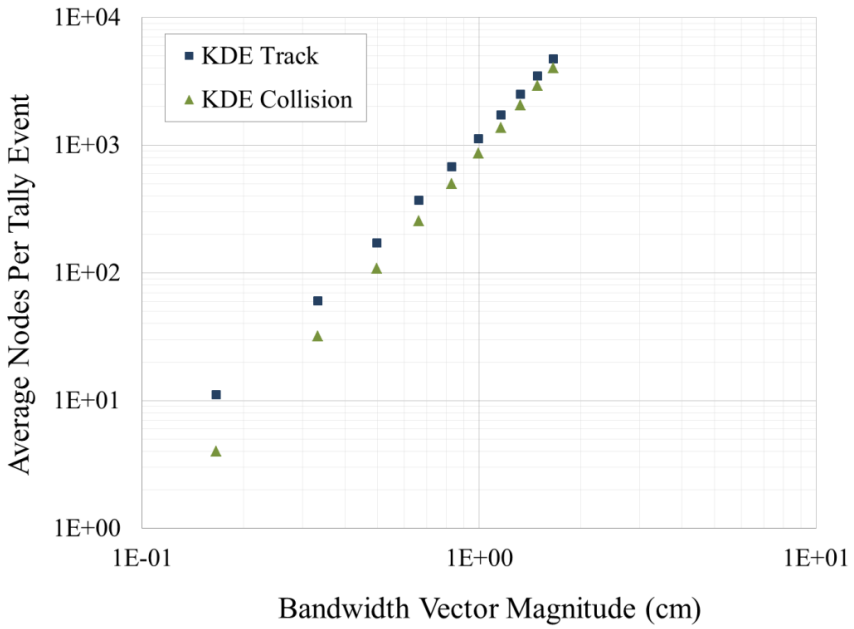

Figure 5.9 Average nodes in the neighborhood region per tally event versus the bandwidth vector magnitude.

Like Figure 5.8, the relationship between average nodes per tally event and bandwidth vector magnitude shown in Figure 5.9 can also be represented by a third-order polynomial for both KDE collision and KDE track mesh tallies. As the bandwidth vector magnitude increases for a fixed input mesh, more nodes are processed per tally event and the time spent in the MCRUN module increases according to the number of nodes processed. Since each node on average will take around the same amount of time to process for a given type of KDE mesh tally, the rate of increase with respect to the average number of nodes processed should become approximately constant at larger bandwidths as the time spent performing other tasks with a fixed cost become less significant. This effect can be seen in Figure 5.10, which plots time per tally point versus the average number of nodes processed per tally event.

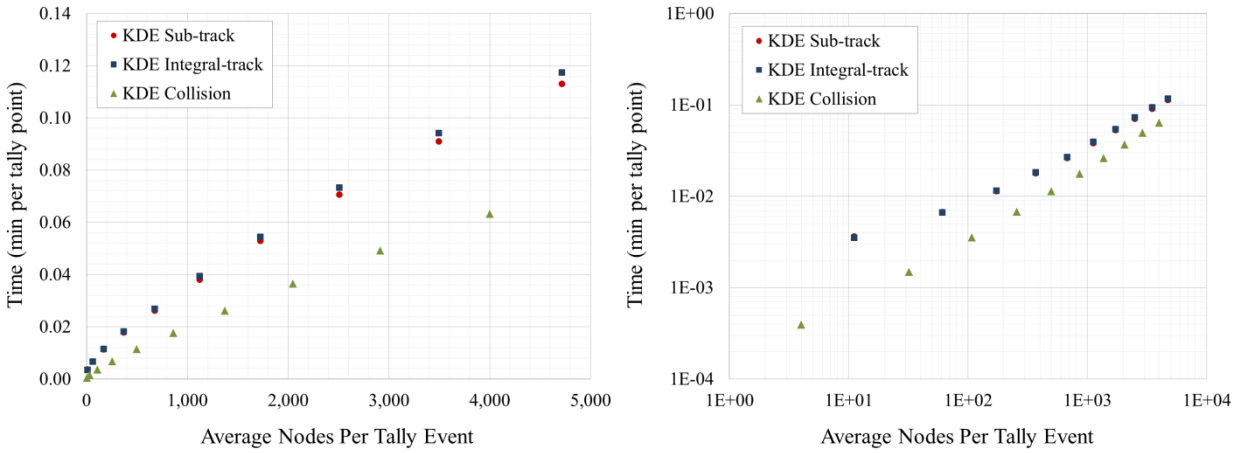

Figure 5.10 Time spent in the MCRUN module of DAG-MCNP5 per tally point versus the average nodes in the neighborhood region per tally event.

As well as affecting the general performance of the KDE mesh tallies, the choice of bandwidth vector can also impact the efficiency of any boundary correction method that is used. Not only can the number of boundary points change, but those boundary points are more likely

to be included in the neighborhood regions for tally events that occur near any of the boundaries. In addition, more boundary points could need correction in more than one dimension, which requires that the matrix system defined by Eq. 4.40 be solved. Therefore, the scaling analysis was repeated with the k-d tree search and the 3D boundary kernel method turned on. The ratio of the time spent in the MCRUN module with and without the 3D boundary kernel method is shown in Figure 5.11 versus bandwidth vector magnitude. As the bandwidth vector magnitude increases, the cost of using boundary correction on performance becomes increasingly more expensive. This large cost is primarily caused by the percentage of nodes considered to be boundary points, which varies from 23% of the total nodes for the optimal bandwidth, up to 90% for the largest bandwidth considered.

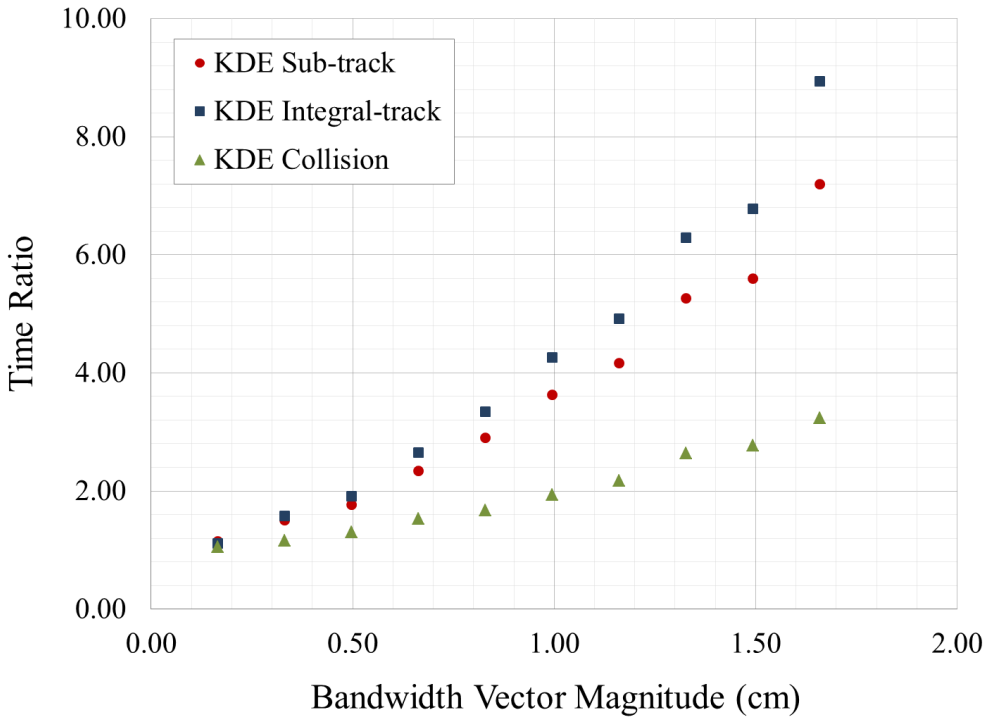

Figure 5.11 Ratio of the time spent in the MCRUN module of DAG-MCNP5 with and without the 3D boundary kernel method versus bandwidth vector magnitude.

In addition to showing that the efficiency of the 3D boundary kernel method depends on the bandwidth vector magnitude, Figure 5.11 also shows that the impact of including boundary correction for the KDE integral-track mesh tally is noticeably larger than for the KDE sub-track mesh tally as the bandwidth vector magnitude gets larger. Although the KDE integral-track mesh tally generally performed equivalent to or better than the KDE sub-track mesh tally, this is no longer true when the bandwidth is increased and boundary correction is used. The reason that this occurs is because more nodes have valid integration limits, which means that correction factors need to be computed for more boundary points per neighborhood region. For practical applications, however, any difference in performance will be minor because the bandwidth vector is usually small and the KDE sub-track mesh tally needs to use more than three sub-tracks to obtain the same level of accuracy.

5.3.5 Kernel Complexity

The final problem characteristic considered for a scaling analysis was kernel complexity, which was quantified by the number of arithmetic operations needed to compute the unique portion of one tally score. Like the previous scaling analysis, kernel complexity only affects KDE mesh tallies. With respect to the timing model described by Eq. 5.8, increasing kernel complexity can only increase b_p, which is the average cost for calculating each tally score. Recall from Section 5.2 that the KDE integral-track mesh tally computes its scores by integrating the path length kernel using Gaussian quadrature, which requires one evaluation per quadrature point. Similarly, each score for the KDE sub-track mesh tally requires one evaluation of its 3D kernel function per pseudo-collision, which is equivalent to around $n(13 + 3K_{ops}) + 1$ arithmetic

operations. In contrast, each score for the KDE collision mesh tally only evaluates its 3D kernel function once. This requires $12 + 3K_{ops}$ arithmetic operations. The kernel complexity for a variety of 1D kernel functions is summarized in Table 5.6.

Table 5.6 Number of arithmetic operations needed to compute the unique portion of one tally score for a KDE mesh tally.

Kernel Name	NUMBER OF ARITHMETIC OPERATIONS			
	KDE Collision	**KDE Sub-track (n = 3)**	**KDE Sub-track (n = 100)**	**KDE Integral-track**
Uniform	12	40	1301	27
Epanechnikov-2	21	67	2201	129
Epanechnikov-4	36	112	3701	327
Epanechnikov-6	48	148	4901	585
Biweight	24	76	2501	243
Triweight	27	85	2801	375

Compared to other problem characteristics such as total particle histories or mesh resolution, kernel complexity appears to have a minor impact on performance. Of the four cases listed in Table 5.6, only the KDE integral-track mesh tally and the KDE sub-track mesh tally with n = 100 showed any signs of a relationship between kernel complexity and performance. This relationship, plotted in Figure 5.12 for both mesh tallies, is approximately linear with relatively flat slopes. The difference in time between the uniform kernel and the sixth-order Epanechnikov kernel is only 3% for the KDE integral-track mesh tally, and 2% for the KDE sub-track mesh tally with n = 100. In contrast, the KDE collision mesh tally and the KDE sub-track

mesh tally with $n = 3$ took around 2.7 minutes and 7.2 minutes respectively for all of the kernel functions considered. These results indicate that, at least for the polynomial kernel family, the kernel function that is used with any KDE mesh tally should generally be chosen for its impact on accuracy, and not for its impact on performance. Since kernels are only needed for the task of computing tally scores, according to Table 5.2 this will only account for about 3.5% of the total time needed for the Reflecting Boundaries test case. If this percentage were to increase, perhaps due to improvements in defining the neighborhood region, then kernel complexity may become a more significant factor.

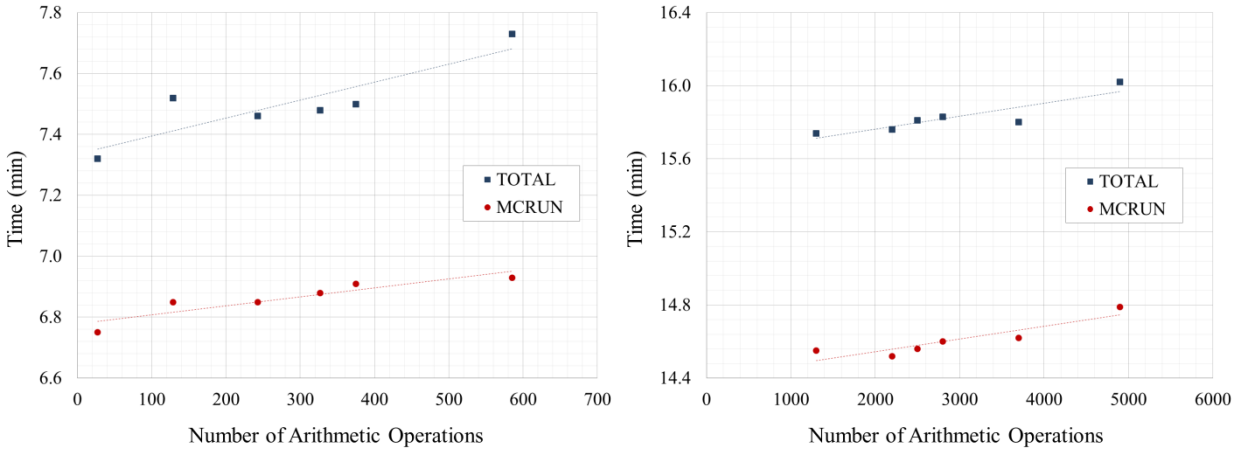

Figure 5.12 Impact of kernel complexity on the KDE integral-track mesh tally (left) and the KDE sub-track mesh tally with $n = 100$ (right).

CHAPTER 6

Summary and Future Work

The work presented in this dissertation has shown that the KDE integral-track mesh tally is a promising alternative to the conventional mesh tally for solving fixed-source problems on arbitrary 3D input meshes.

With respect to accuracy, the KDE integral-track mesh tally was shown to produce more accurate results than an equivalent KDE sub-track mesh tally with no performance penalty. Compared to conventional mesh tallies, the KDE integral-track mesh tally was also shown to produce identical or comparable neutron flux distributions for some simple test cases. However, unlike conventional methods, there is more room for improvement because the KDE integral-track mesh tally can use different bandwidths and kernels for the same input mesh.

With respect to efficiency, the KDE integral-track mesh tally was shown to perform better than the conventional mesh tally for unstructured meshes, but was less efficient for structured meshes. Recall that this analysis was based on using identical input meshes for both mesh tallies. Since KDE mesh tallies produce results at specific calculation points instead of piecewise constant approximations, coarser meshes could theoretically be used to improve efficiency, while still maintaining the same level of accuracy. Whereas coarser meshes reduce accuracy for conventional mesh tallies, for KDE mesh tallies the result at any given point is independent of the mesh resolution. Therefore, the accuracy at that point will remain unaffected by coarsening the mesh.

With respect to ease of use, the current implementation of the KDE integral-track mesh tally can be integrated into the existing tally functionality of any Monte Carlo transport code using the new DAGTally library. Like conventional mesh tallies, both a tally value and relative standard error representing the variance of the results are reported. Choosing a bandwidth is equivalent to selecting the size of the mesh cells for a conventional mesh tally, with smaller values generally producing more accurate results than larger ones, but with greater variance. The primary difference in using the KDE integral-track mesh tally is that a kernel function must also be chosen, but for most cases the default second-order Epanechnikov kernel should be sufficient.

A detailed summary of the important findings in each of the preceding chapters is now presented. Section 6.1 first summarizes how the kernel density estimator was implemented as a mesh tally in a production Monte Carlo transport code. Then, key results from the quantitative mesh tally comparisons of Chapter 3 are presented in Section 6.2. Sections 6.3 and 6.4 highlight the important points that were discovered by analyzing the accuracy and efficiency of the KDE integral-track mesh tally respectively. Finally, Section 6.5 introduces some key areas for future work that are needed to improve the viability of the KDE integral-track mesh tally for solving a wide variety of practical problems.

6.1 Kernel Density Estimated Monte Carlo Mesh Tallies

Chapter 2 introduced the multivariate kernel density estimator as a Monte Carlo mesh tally for approximating the neutron flux at any calculation point (x, y, z). Based on an average contribution of N particle histories, KDE mesh tallies require that both a bandwidth vector and kernel function be chosen by the user *a priori*. Choosing a bandwidth is similar to selecting bin

width for a conventional mesh tally, whereas the kernel shape determines the weighting factor of each contribution. The set of calculation points is represented by the nodes from some arbitrary input mesh, which can be considered as a series of fixed individual tally points or detectors.

Like conventional mesh tallies, both collision and track length estimators can be used with KDE mesh tallies. However, track length estimators are preferred because they produce better results in regions with low collision densities, and can also be used in void regions. The original KDE track length estimator, the KDE sub-track estimator, only included contributions from discrete points or pseudo-collisions along the track. Since the pseudo-collision density affects both the accuracy and efficiency of the KDE sub-track estimator, a new estimator based on integrated particle tracks was proposed. This new KDE integral-track estimator was derived by redefining the pseudo-collisions in terms of a common path length variable, and then by taking the limit as the number of sub-tracks approached infinity.

The KDE integral-track estimator was implemented as an alternative mesh tally in a new C++ library called DAGTally, which allows it to be used with any Monte Carlo transport code. As with other KDE mesh tallies, the choice of bandwidth should be made carefully. Values that are too low produce results with more variance, yet values that are too high produce results that are more biased. However, it is possible to find an optimal value that minimizes both bias and variance. In addition to choosing the bandwidth, the KDE integral-track mesh tally also allows for kernels of different shapes and orders. All kernels from the polynomial kernel family are available, with the second-order Epanechnikov kernel being the most commonly used in practice.

Other options available to the KDE integral-track mesh tally are the inclusion of either boundary correction and/or neighborhood region searches. Boundary correction is only needed to correct the increase in bias that occurs for calculation points that are less than one bandwidth from an external geometrical boundary. If there are no such boundary points, then boundary correction has no impact on the results and can be omitted. In contrast, the neighborhood region search should always be turned on for efficiency purposes. Since typical input meshes can have thousands of nodes, using the neighborhood region search limits the set of possible calculation points to the ones expected to produce non-trivial contributions.

6.2 Quantitative Mesh Tally Comparisons

The KDE integral-track mesh tally introduced in Chapter 2 was quantitatively compared to other mesh tallies using the five test cases described in Chapter 3. Each of these test cases represented specific features commonly found in fixed-source problems – like producing a uniform or gradient flux, using a vacuum or reflective boundary condition, including a material discontinuity, or defining a uniform volume source in a highly absorbing medium. The first quantitative comparison was performed between KDE sub-track and KDE integral-track mesh tallies. Different KDE mesh tallies can be compared directly because they all produce nodal-based results. Results from this first comparison showed that the KDE sub-track estimator consistently produces tally values and errors that converged to the KDE integral-track results as more sub-tracks were added. This means that the KDE integral-track mesh tally will generally produce more accurate results than an equivalent KDE sub-track mesh tally, since it essentially uses an infinite number of sub-tracks.

The second quantitative comparison was performed between KDE integral-track and conventional mesh tallies. However, even when identical particle histories are used, it can still be challenging to prove that the nodal-based KDE results are equivalent to the cell-averaged ones of the conventional mesh tally. Two different approaches were considered, with the first being the direct method. The direct method is based on the fact that the 1D histogram and kernel density estimator will produce identical approximations under appropriate conditions. Extending this concept to mesh tallies requires that identical particle histories are used, the conventional mesh cells align with the KDE mesh nodes, and the KDE integral-track mesh tally uses the uniform kernel with a bandwidth vector equivalent to the size of the mesh cells.

Using the direct method, KDE integral-track and MCNP5 mesh tallies were compared for the four test cases based on structured meshes. Most KDE tally values and errors were identical up to the six significant figures reported by the MCNP5 mesh tallies, with the few exceptions being attributed to rounding errors. Though these results show that KDE integral-track and MCNP5 mesh tallies can produce identical flux distributions, note that MCNP5 always produces the same results for any given input mesh. In contrast, the KDE integral-track mesh tally can use other kernels and bandwidths to improve its accuracy. However, it is important to remember that using different kernels with the KDE integral-track mesh tally will only be more effective if the input mesh is coarse enough for the MCNP5 mesh tally to produce less accurate results.

The second approach considered for comparing KDE integral-track and conventional mesh tallies was the data transfer method. This data transfer method uses finite element theory to interpolate the neutron flux at the nodes of each mesh cell to a set of cubature points, which

can then be used with a Gaussian cubature rule to approximate cell-averaged values. Since the direct method only works for overlapping structured meshes, the data transfer method is needed whenever identical or unstructured meshes are used. As part of this data transfer method, first an appropriate finite element must be chosen. Higher-order elements will generally work better for regions with greater curvature. Once an appropriate finite element is chosen, it is also necessary for the resolution of the input mesh to be fine enough so that the conventional mesh tally will produce accurate results. Using a resolution that is too coarse can negatively skew the results of the comparison because the accuracy of KDE mesh tallies does not depend on mesh resolution.

Based on the data transfer method, KDE integral-track and conventional mesh tallies were compared for all five test cases. A convergence analysis of the KDE integral-track results showed that as more particle histories were added, the approximated cell-averaged values that were produced converged both locally and globally to the reference solutions for all five test cases. These results indicate that the KDE integral-track mesh tally can produce comparable results to the conventional mesh tally, for both structured and unstructured meshes. However, the KDE integral-track mesh tally is also capable of producing more accurate results for the same input mesh and particle histories, as was demonstrated with the problem computing the helium production in the flow insert in the detailed blanket module model of ITER.

6.3 Accuracy of KDE Mesh Tallies

Whereas Chapter 3 showed that the KDE integral-track mesh tally can produce identical or comparable flux distributions to other mesh tallies, Chapter 4 explored the different factors that can affect its accuracy. These factors included both the bandwidth and kernel, as well as the

option of using boundary correction. Prior to performing a bandwidth sensitivity experiment, bias and variance approximations for the 3D kernel density estimator were first derived. The bias was shown to be proportional to $O(h^2)$, the kernel variance, and the curvature of the true density $f(x, y, z)$. In contrast, the variance was shown to be inversely proportional to $O(h^3)$, but proportional to the kernel roughness and $f(x, y, z)$. Results from the bandwidth sensitivity experiment were mostly consistent with these theoretical expectations. However, integrating the path length kernel changes the behavior of the variance. Instead of being inversely proportional to $O(h^3)$, the KDE integral-track mesh tally was shown to be inversely proportional to $O(h^2)$.

As part of the bandwidth sensitivity experiment, two distinct points on the input mesh were considered. The first was an interior point with lower curvature and higher neutron flux, whereas the second was a boundary point with higher curvature and lower neutron flux. In general, the bias and variance for both of these points were inversely related in the sense that using a different bandwidth to improve one usually came at the cost of the other. However, the impact of this trade-off is much more noticeable for the boundary point because bias is greater in regions with higher curvature. In regions with less curvature, where bias becomes negligible, the bandwidth will only affect the variance.

The trade-off between bias and variance for the KDE integral-track mesh tally can make it challenging to choose effective bandwidths, especially when the neutron flux distribution has highly contrasting levels of curvature. Though using a global bandwidth at all calculation points is the easiest option, a known 1D mixture distribution was used to show that reducing this global bandwidth in one region can also increase variance in other regions with lower curvature. Since

the solution to some multi-dimensional transport problems could have even greater fluctuations in curvature, regional bandwidths associated with the calculation points should be preferred for most practical applications. Allowing for multiple bandwidths means that regions with higher curvature can use smaller values to reduce their bias, without negatively impacting other regions with lower curvature. Associating those bandwidths with the calculation points instead of the observation points also produces better approximations near the regional boundaries, while still maintaining the same level of accuracy everywhere else.

Regional bandwidths associated with the calculation points are only one option for improving the accuracy of the KDE integral-track mesh tally. Another option is to consider alternative kernel functions, which is not possible for conventional mesh tallies. The bandwidth sensitivity experiment showed that the second-order Epanechnikov kernel produced results that were 40% less biased than the uniform kernel. However, no matter what second-order kernel is used, their bias is limited to $O(h^2)$ for regions with non-zero curvature. Chapter 4 therefore also explored higher-order kernels, which are kernel functions whose first non-zero moment is greater than κ_2. For a higher-order kernel of order v, the bias should be proportional to $O(h^v)$. Though the expected $O(h^v)$ behavior was not observed for fourth-order or sixth-order Epanechnikov kernels, the bias was still noticeably reduced at larger bandwidths. This reduced sensitivity to the bandwidth means that higher-order kernels could be a viable bias reduction tool.

The final part of Chapter 4 described the boundary kernel as a boundary correction method, which defines a new kernel function from a linear combination of the kernel function that is used at the interior points. For second-order kernels, the unknown coefficients of this

linear combination are obtained by requiring that it integrates to unity, and that its first moment in each dimension is zero. An approximation of the 3D boundary kernel was shown to be more effective than a 1D boundary kernel used in each dimension, especially for curved surfaces. Although the extent of the difference between the two methods depends on how many dimensions need correcting, as well as how many particle histories are used, the 3D boundary kernel method will generally be preferred because of its improved accuracy.

6.4 Efficiency of KDE Mesh Tallies

After exploring the accuracy of the KDE integral-track mesh tally in Chapter 4, the focus in Chapter 5 shifted to its efficiency. The performance of the KDE integral-track mesh tally was first compared to other mesh tallies using the same five test cases from Chapter 3. Compared to the conventional mesh tally, the KDE integral-track mesh tally was around three to four times slower per tally point for structured meshes, but around eight times faster for the unstructured mesh. Similar results were reported for the KDE sub-track mesh tally with $n = 2$. Increasing the number of sub-tracks also reduces the performance of the KDE sub-track mesh tally, which means that the KDE integral-track mesh tally is not only more accurate, but also more efficient.

Though its efficiency depends on many factors, a profiling analysis of the KDE integral-track mesh tally showed that its most time-consuming task was defining the neighborhood region. Defining the neighborhood region required about 68.7% of the total central processing unit (CPU) instructions, compared to 22.9% for simulating the particle histories, and 5.8% for processing the tally scores. The default procedure used to define the neighborhood region locates possible calculation points using a k-d tree search, restricting the search based on the

bandwidth and other properties of the track segment being tallied. Once the neighborhood region is known, scores are only computed for calculation points with valid integration limits. This helps make the task of processing the tally scores as efficient as possible.

Another task that can affect the efficiency of the KDE integral-track mesh tally is the optional boundary correction method. Both 1D and 3D boundary kernel methods introduced in Chapter 4 must compute an additional correction factor for every score added to a boundary point. If correction is only needed in one dimension, then both methods use exactly the same process for computing this correction factor. However, if correction is needed in more than one dimension, then the 3D method must create and solve a matrix system to form the boundary kernel. For two dimensions this matrix will be 3x3, whereas for three dimensions it will be 4x4. Even though solving a matrix system makes the 3D method slightly less efficient, it will still be preferred to the 1D method when accuracy is more important.

The final part of Chapter 5 showed how the performance of the KDE integral-track mesh tally scales with respect to a number of problem characteristics for a structured mesh problem – including total particle histories, mesh resolution, bandwidth vector magnitude, and kernel complexity. Both KDE and conventional mesh tallies scale in a similar manner with respect to total particle histories, which is the most significant factor in the overall performance. However, the performance of the KDE integral-track mesh tally was also shown to be much more dependent than the conventional mesh tally on mesh resolution, or more specifically node density. This node density is one factor that can influence the average number of nodes that will be processed in a characteristic neighborhood region, along with the bandwidth vector magnitude

and mean free path. Note that the impact of each of these factors is also multiplicative with respect to total particle histories, since adding more histories increases the number of neighborhood regions that will be processed. Of all the scaling analyses performed, the problem characteristic shown to have the smallest impact on performance was kernel complexity, which indicates that polynomial kernels should generally be chosen for accuracy and not for efficiency.

As part of the scaling analyses, the impact of using the optional k-d tree search and 3D boundary kernel method were also considered. While not all of the input meshes were consistent with the general trend, using the k-d tree search significantly improved performance with respect to the number of tally points. In contrast, using boundary correction always decreased the performance even when the k-d tree search was turned on. The cost of using boundary correction was shown to be dependent on mesh resolution and the bandwidth, which can both affect the number of boundary points. Although this cost is non-negligible, it can easily be avoided by defining input meshes so that nodes on their surfaces are at a distance greater than one bandwidth from any external geometrical boundary.

6.5 Future Work

Although the KDE integral-track mesh tally introduced in this dissertation has shown potential for solving fixed-source problems on arbitrary 3D input meshes, this section discusses some key areas for future work that will improve its use for practical applications.

6.5.1 Implementation and Testing

The first area for future work is to improve the implementation of the KDE integral-track mesh tally. Though using regional bandwidths associated with the calculation points were

shown to be more effective than choosing one global bandwidth for a 1D example, this method also needs to be tested against a more complex 3D problem with varying levels of curvature. Given that the current implementation only allows users to assign a "global" bandwidth to individual input meshes, it could become time-consuming to set up such a problem with many regions. Therefore, it may be preferable to allow for different bandwidths to be defined on one input mesh. This could be automated during pre-processing by tagging each node in the input mesh with its own bandwidth, or even by creating unique mesh sets for each region. If regional bandwidths are implemented for an input mesh, however, then the implementation of the neighborhood region search would need to be modified because its size depends on the bandwidth. One option is to base the neighborhood region on the largest bandwidth.

In addition to automating regional bandwidths, more work is needed on improving the implementation of the 3D boundary kernel method. The current approximation requires that a pre-processing tool be used to tag boundary points on the input mesh, and can also only be used with second-order kernels. Extending this method to higher-order kernels would be straightforward, requiring only that higher-order terms and additional constraints be defined so that the resulting boundary kernel will always be proportional to $O(h^{\nu})$. However, note that adding more constraints will increase the size of the matrix system that needs to be solved.

6.5.2 Accuracy

The second area for future work is to continue researching the accuracy of the KDE integral-track mesh tally with respect to both bandwidth and kernel function. Though regional bandwidths are effective, and also require less computational effort than other more complicated

methods, it was shown that the accuracy of this approach will depend on the number of regions used. Adding too many regions reduces accuracy because the optimally-chosen local bandwidths become so small that variance starts to dominate. One interesting study could therefore focus on how to choose both regions and optimal local bandwidths for a 3D problem with many materials. Separating regions by materials with significantly different absorption or scattering cross sections could provide a good starting point. Another interesting study could consider the convergence of the MISE versus the cost of increased variance as a result of using higher-order kernels in different scenarios. Additional kernel options such as the Gaussian kernel, or even a non-separable 3D kernel function could be included as part of this analysis.

Instead of using regional bandwidths or different kernel functions to improve accuracy, an alternative option is to first use variance reduction techniques to establish a uniform track density. With a uniform track density, the bandwidth vector that is used with the KDE integral-track mesh tally is expected to only affect the variance of the approximation.

Another option for future work on accuracy is to consider using the adaptive kernel method. The adaptive kernel method requires a two-stage procedure [5], which makes it an attractive option for solving fixed-source problems. This two-stage procedure first requires that an initial approximation of the neutron flux distribution is obtained, using either Monte Carlo or deterministic methods. Local bandwidths for the final Monte Carlo simulation would then be chosen from this initial approximation based on its curvature. Using an adaptive kernel method instead of the standard kernel density estimator can produce results that are as accurate as using higher-order kernels, without introducing any negative results [5].

The final option for exploring the accuracy of the KDE integral-track mesh tally is to consider alternative boundary correction methods. If the problem uses reflecting boundaries, for example, a 3D version of the reflection method [5] may be more accurate and/or more efficient. Alternatively, though the 3D boundary kernel was shown to be effective for simple 3D problems, more work needs to be done on improving the approximation of the *uvw*-space over which the integration is performed. Restricting the *uvw*-space to a rectangular domain might be less accurate for larger bandwidths because there is a bigger discrepancy between the true domain and the approximated domain.

6.5.3 Efficiency

The third area for future work is to continue researching the efficiency of the KDE integral-track mesh tally. Although this dissertation provided some insight into how its performance scales with different problem characteristics in terms of single effects, there are other possibilities that need to be considered. The most notable of these possibilities is to study the interaction between mesh resolution, bandwidth vector magnitude, and mean free path.

Another interesting area of study could be to compare the performance of the KDE integral-track mesh tally to the conventional mesh tally using more complicated problems. The structured mesh and rectangular geometry that were used for the scaling analyses in this dissertation were very efficient for the conventional mesh tally. However, the KDE integral-track mesh tally should be more efficient than the conventional mesh tally for problems defined on unstructured meshes. As such, the scaling analyses should be repeated using an unstructured mesh with more complex surface crossings.

With respect to improving efficiency, the primary focus should be on the neighborhood region to obtain the most benefit. There are two potential options that can be considered. The first option is to try different search methods and tree structures that may be more efficient than the k-d tree implementation. Another option is to improve the approximation of the true neighborhood region to reduce the number of trivial contributions that are computed. One possibility is to consider a cylindrical approximation with maximum radius $r_{\max}$ defined by:

$$r_{\max} = \sqrt{h_x^2 + h_y^2 + h_z^2}\,. \tag{6.1}$$

The efficiency of this cylindrical approximation, like the current axis-aligned box approximation, will depend on the orientation of the track segment being tallied.

Rather than redefining the neighborhood region, another option for improving efficiency is to perform all tally computations during post-processing. This would require that the particle history data be stored, which is already possible in DAG-MCNP5 using the PTRAC option [3]. There are two main benefits to separating the tally computations from the transport process. First, different bandwidth options could be explored without having to simulate the histories multiple times. Second, since each node of the input mesh computes its score independently of the others, all of the calculations could be performed in parallel on the graphical processing unit (GPU). GPU programming languages such as the compute unified device architecture (CUDA) are very efficient at performing the same task multiple times [40], which makes it an attractive option for parallelizing the KDE integral-track mesh tally.

6.5.4 Possible Applications

The fourth and final area for future work is to consider possible applications for the KDE integral-track mesh tally. One such application is to see if KDE mesh tallies are a viable alternative for calculating moments of the neutron flux ϕ_{lm} based on the uncollided flux estimator. The conventional approach tallies weighted path lengths $\bar{l}$ that are defined by [41]:

$$\bar{l} = \int_0^d w_o e^{-\sigma s} ds = \frac{w_o}{\sigma}\left(1 - e^{-\sigma d}\right), \tag{6.2}$$

where w_o is the particle weight, σ is the cross section of the material the uncollided flux is traveling through, and d is the total distance traveled. Weighted path lengths obtained using Eq. 6.2 can be tallied in some region of interest with volume V by [41]:

$$\phi_{lm} = \frac{1}{NV}\sum_{n=1}^{N} \bar{l}_n Y_{lm}(\hat{\Omega}_n), \tag{6.3}$$

where N is the total particle histories, and Y_{lm} is the spherical harmonic for the direction $\hat{\Omega}_n$ in which the n^{th} particle is traveling. By using $\bar{l}$ as the observations for which the kernel contributions are evaluated, it should be relatively simple to convert Eq. 6.3 into one based on the KDE integral-track approach for computing the moments of the neutron flux at some calculation point (x, y, z).

A second possible application for the KDE integral-track mesh tally is to use it for solving energy-dependent transport problems. Though the current implementation does include support for energy bins and tally multipliers, this dissertation only focused on energy-

independent transport problems. Including multiple energy groups with the KDE integral-track mesh tally will generally, however, require that each energy bin define its own global or regional bandwidths in order to obtain accurate solutions across the entire energy spectrum.

A third possible application for the KDE integral-track mesh tally is to use it for resolving neutron flux distributions based on a hp-adaptivity technique. The KDE integral-track mesh tally would first be used to obtain accurate tally results at specific nodes throughout some coarse input mesh. Then, polynomial functions of order "p" would be used to interpolate the results between those nodes. This technique may be able to provide more accurate results throughout the domain more efficiently than conventional methods.

REFERENCES

[1] L. L. Carter and E. D. Cashwell, "Particle-transport Simulation with the Monte Carlo Method," Technical Information Center, U.S. Energy Research and Development Administration, Oak Ridge, TN (1975).

[2] E. E. Lewis and W. F. Miller Jr., *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, IL (1993).

[3] X-5 Monte Carlo Team, "MCNP5 – A General Monte Carlo N-Particle Transport Code, Version 5," LA-UR-03-1987, Los Alamos National Laboratory, NM (2003).

[4] MCNP6 Development Team, "Initial MCNP6 Release Overview – MCNP6 Beta 2, LA-UR-11-07082," Los Alamos National Laboratory, NM (2011).

[5] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London, England (1986).

[6] D. P. Griesheimer, "Functional Expansion Tallies for Monte Carlo simulations," Ph.D. Thesis, Nuclear Engineering and Radiological Sciences, University of Michigan, Ann Arbor, MI (2005).

[7] K. Banerjee, "Kernel Density Estimator Methods for Monte Carlo Radiation Transport," Ph.D. Thesis, Nuclear Engineering and Radiological Sciences, University of Michigan, Ann Arbor, MI (2010).

[8] B. W. Silverman and M. C. Jones, "E. Fix and J. L. Hodges (1951): An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges (1951)," *International Statistical Review*, **57**, 3, pp. 233-238 (1989).

[9] M. Rosenblatt, “Remarks on Some Nonparametric Estimates of a Density Function,” *The Annals of Mathematical Statistics*, **27**, 3, pp. 832-837 (1956).

[10] T. Cacoullos, “Estimation of a Multivariate Density,” *Annals of the Institute of Statistical Mathematics*, **18**, 1, pp. 179-189 (1966).

[11] J. Du and W. R. Martin, “Adaptive Kernel Density Estimation and Monte Carlo Sampling,” *Transactions of the American Nuclear Society*, **74**, pp. 171-172 (1996).

[12] N. Tyagi and W. R. Martin et al., “A Proposed Alternative to Phase-space Recycling using the Adaptive Kernel Density Estimator Method,” *Medical Physics*, **33**, 2, pp. 553- 560 (2006).

[13] K. Banerjee and W. R. Martin, “Kernel Density Estimated Monte Carlo Global Flux Tallies,” *Proceedings of the International Conference on Mathematics, Computational Methods & Reactor Physics*, Saratoga Springs, NY, May 3-7 (2009).

[14] K. Banerjee and W. R. Martin, “Kernel Density Estimation Method for Monte Carlo Global Flux Tallies,” *Nuclear Science and Engineering*, **170**, pp. 234-250 (2012).

[15] K. Banerjee and W. R. Martin, “Kernel Density Estimation Method for Monte Carlo Point Detector and Surface Crossing Flux Tallies,” *Nuclear Science and Engineering*, **174**, pp. 30-45, 2013.

[16] K. L. Dunn and P. P. H. Wilson, “Kernel Density Estimators for Monte Carlo Tallies on Unstructured Meshes,” *Transactions of the American Nuclear Society*, **107**, pp. 490-493 (2012).

[17] K. L. Dunn and P. P. H. Wilson, “Monte Carlo Mesh Tallies based on a Kernel Density Estimator Approach using Integrated Particle Tracks,” *Proceedings of the International Conference on Mathematics, Computational Methods & Reactor Physics*, Sun Valley, ID, May 5-9 (2013).

[18] T. P. Burke, B. C. Kiedrowski, and W. R. Martin, "Flux and Reaction Rate Kernel Density Estimators in OpenMC," *Transactions of the American Nuclear Society*, **109**, pp. 683-686 (2013).

[19] H. G. Müller, "Smooth Optimal Kernel Estimators of Densities, Regression Curves and Modes," *The Annals of Statistics*, **12**, 2, pp. 766-774 (1984).

[20] B. E. Hansen, "Exact Mean Integrated Squared Error of Higher Order Kernel Estimators," *Econometric Theory*, **21**, 6, pp. 1031-1057 (2005).

[21] T. J. Tautges and P. P. H. Wilson et al., "Acceleration Techniques for Direct Use of CAD-based Geometries in Monte Carlo Radiation Transport," *Proceedings of the International Conference on Mathematics, Computational Methods & Reactor Physics*, Saratoga Springs, NY, May 3-7 (2009).

[22] M. P. Wand and W. R. Schucany, "Gaussian-based Kernels," *The Canadian Journal of Statistics*, **18**, 3, pp. 197-204 (1990).

[23] T. J. R. Hughes, *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Dover Publications, Mineola, NY (2000).

[24] R Core Team, "R: A Language and Environment for Statistical Computing," R Foundation for Statistical Computing, Vienna, Austria, http://www.R-project.org (2013).

[25] M. C. Jones, "Simple Boundary Correction for Kernel Density Estimation," *Statistics and Computing*, **3**, pp. 135-146 (1993).

[26] J. S. Marron, "Visual Understanding of Higher-Order Kernels," *Journal of Computational and Graphical Statistics*, **3**, 4, pp. 447-458 (1994).

[27] J. S. Marron and D. Ruppert, "Transformations to Reduce Boundary Bias in Kernel Density Estimation," *Journal of the Royal Statistical Society. Series B*, **56**, 4, pp. 653-671 (1994).

[28] A. Cowling and P. Hall, "On Pseudodata Methods for Removing Boundary Effects in Kernel Density Estimation," *Journal of the Royal Statistical Society. Series B*, **58**, 3, pp. 551-563 (1996).

[29] M. Y. Cheng, J. Fan, and J. S. Marron, "On Automatic Boundary Corrections," *The Annals of Statistics*, **25**, 4, pp. 16971-1708 (1997).

[30] R. J. Karunamuni and T. Alberts, "On Boundary Correction in Kernel Density Estimation," *Statistical Methodology*, **2**, 3, pp. 191-212 (2005).

[31] R. J. Karunamuni and S. Zhang, "Some Improvements on a Boundary Corrected Kernel Density Estimator," *Statistics and Probability Letters*, **78**, 5, pp. 499-507 (2008).

[32] H. G. Müller and U. Stadtmüller, "Multivariate Boundary Kernels and a Continuous Least Squares Principle," *Journal of the Royal Statistical Society: Series B*, **61**, 2, pp. 439-458 (1999).

[33] B. C. Kiedrowski, "Bivariate and Multivariate Boundary Kernels," unpublished (2013).

[34] Center for High Throughput Computing, "HPC Cluster Basic Use Guide," University of Wisconsin-Madison, http://chtc.cs.wisc.edu/HPCuseguide.shtml (2013).

[35] J. Weidendorfer, M. Kowarschik, and C. Trinitis, "A Tool Suite for Simulation Based Analysis of Memory Access Behavior," *Proceedings of the 4th International Conference on Computational Science*, Krakow, Poland, June (2004).

[36] N. Nethercote and J. Seward, "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation," *Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation*, San Diego, CA, June (2007).

[37] T. J. Tautges and R. Meyers et al., "MOAB: A Mesh-Oriented Database," SAND2004-1592, Sandia National Laboratories, Albuquerque, NM (2004).

[38] J. L. Bentley, "Multidimensional Binary Search Trees used for Associative Searching," *Communications of the ACM*, **18**, 9, pp. 509-517 (1975).

[39] E. Anderson and Z. Bai et al., "LAPACK Users' Guide," 3rd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA (1999).

[40] NVIDIA Corporation, "CUDA C Programming Guide," PG-02829-001_v6.0, http://docs.nvidia.com/cuda/cuda-c-programming-guide/ (2014).

[41] T. M. Evans, Private Communication (2013)

[42] E. Gamma and R. Helm et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Westford, MA (1995).

[43] G. F. Knoll, *Radiation Detection and Measurement*, 3rd Edition, Wiley and Sons, Hoboken, NJ (2000).

APPENDIX A: DAGTally Design Overview

DAGTally is based on an abstract design called the observer pattern, which is used in object-oriented software development to communicate changes of state to a list of dependents [42]. The state is maintained by an object known as the observable (or subject), which also manages a list of dependents known as the observers. If the state changes, the observable will then notify all of its active observers. Each observer can use the data stored in the state however it chooses, independently of the other observers. Figure A.1 shows how the observer pattern is applied to Monte Carlo tallies, which identifies the TallyEvent as the state, the TallyManager as the observable, and the individual Tally objects as the observers. Note that each Tally is derived from a common base class that defines the interface needed for it to work within a Monte Carlo transport simulation. Using this polymorphic class hierarchy allows the TallyManager to interact with any type of Tally, including both standard tallies and mesh tallies.

The general workflow of a Monte Carlo transport simulation that uses the TallyManager shown in Figure A.1 begins at the physics interface. As particles are born and transported through the geometry, the physics interface transfers the data describing each new TallyEvent to the TallyManager through the *setTallyEvent* method. The type of data that will be transferred will depend on whether the event is defined as a collision (collision location, macroscopic cross section, etc.) or a track (track length, starting location, direction, etc.). Once the TallyManager receives the new data, it then updates the TallyEvent that it stores as a private member. After the TallyEvent is updated, the physics interface can then prompt the TallyManager to notify all of its active Tally observers. This step in the workflow is separated from the change of state since

there are three different scenarios for which the TallyManager needs to notify its Tally observers. Table A.1 provides a summary of these scenarios, including the purpose of each notification and how the Tally observers will generally respond. However, the details of each response is completely up to the individual Tally implementations.

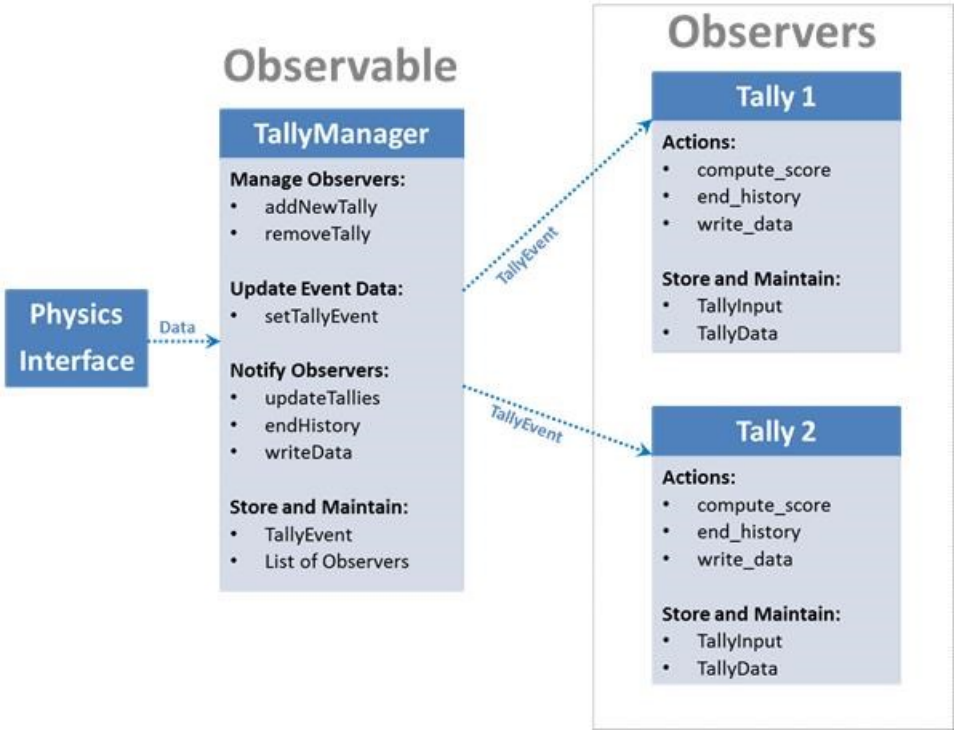

Figure A.1 Application of the observer pattern to Monte Carlo tallies

Table A.1 Summary of scenarios for which the TallyManager can notify its Tally observers.

Scenario	Purpose of Notification	Action of Tally Observer
updateTallies	New TallyEvent has occurred	Compute new tally scores
endHistory	Current particle history has ended	Perform end of history calculations
writeData	Monte Carlo simulation has finished	Write results to output format

APPENDIX B: 2D Example of Data Transfer Method

The data transfer method described in Chapter 3 allows nodal-based results to be converted into cell-averaged values for comparison purposes. This appendix includes a simple 2D example of how the data transfer method works for a bilinear quadrilateral element. A similar procedure can be used for the 3D finite elements shown in Figure 3.9. Most of the background information for this example was obtained from *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis* by Hughes [23].

A bilinear quadrilateral element is a 2D finite element that has straight edges connecting the four nodes at each corner of its domain. Figure B.1 shows how these nodes are mapped into the natural coordinates (ξ, η) of the element. The equations for these mappings are of the form:

$$\begin{aligned} x(\xi,\eta) &= \sum_{k=1}^{n} x_k N_k(\xi,\eta) \\ y(\xi,\eta) &= \sum_{k=1}^{n} y_k N_k(\xi,\eta) \end{aligned}, \tag{B.1}$$

where the four basis functions N_k are defined at each node k as:

$$\begin{aligned} N_1(\xi,\eta) &= 1/4\,(1-\xi)(1-\eta) \\ N_2(\xi,\eta) &= 1/4\,(1+\xi)(1-\eta) \\ N_3(\xi,\eta) &= 1/4\,(1+\xi)(1+\eta) \\ N_4(\xi,\eta) &= 1/4\,(1-\xi)(1+\eta) \end{aligned}. \tag{B.2}$$

Note that these basis functions are derived by first assuming a bilinear expansion for both $x(\xi, \eta)$ and $y(\xi, \eta)$, and then by requiring that $N_k = 1$ for node k and $N_k = 0$ for the other three nodes.

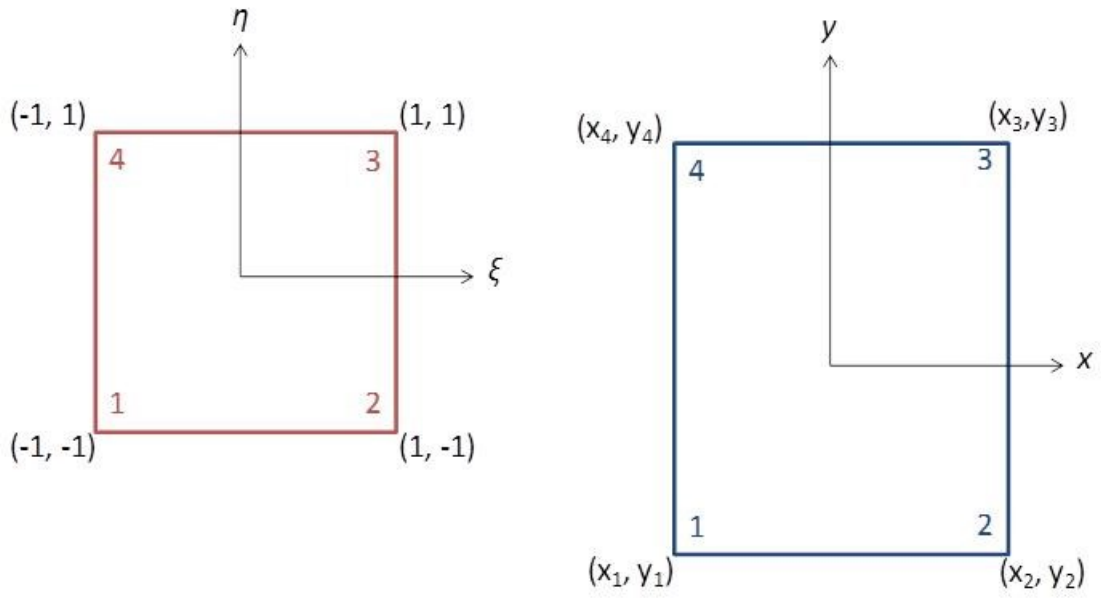

Figure B.1 Bilinear quadrilateral element (right) mapped to its natural coordinates (left).

Suppose that the KDE integral-track mesh tally was used to obtain the neutron flux $\hat{\varphi}_k$ at the four nodes of an arbitrary mesh cell represented as a bilinear quadrilateral element. From these nodal-based results, an approximation for the cell-averaged neutron flux can be computed using the 2D equivalent of Eq. 3.9:

$$\widetilde{\varphi} = \frac{\int_A \varphi(x,y)dA}{\int_A dA} \approx \frac{1}{A}\sum_{l=1}^{p} w^l j^l \left(\sum_{k=1}^{4} \hat{\varphi}_k N_k^l \right), \tag{B.3}$$

where A is the surface area of the mesh cell. To evaluate Eq. B.3, first the number of cubature points must be chosen. For simplicity, this 2D example will use only one cubature point located at the natural coordinates $l = (0, 0)$. This means that the cubature weight is $w^l = 4$, and that all four of the basis functions defined by Eq. B.2 will evaluate to $N_k = 0.25$.

The final component of Eq. B.3 needed to obtain the approximation for the cell-averaged neutron flux involves the Jacobian matrix. The 2D Jacobian matrix J for mapping the original coordinates to their natural coordinates is defined by:

$$J = \frac{\partial(x,y)}{\partial(\xi,\eta)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}. \tag{B.4}$$

For the structured mesh cell represented in Figure B.1, the determinant of J evaluated at the cubature point $l = (0, 0)$ always evaluates to $j^l = A/4$. Therefore, substituting j^l back into Eq. B.3 with the other known terms produces the following simplified approximation for the cell-averaged neutron flux:

$$\widetilde{\varphi} \approx \frac{1}{4}\sum_{k=1}^{4} \hat{\varphi}_k, \tag{B.5}$$

which is equivalent to simply averaging the four nodal-based results that were originally obtained using the KDE integral-track mesh tally. Though the data transfer method does not always reduce to a simple average, the same general procedure that was followed for this 2D example can also be used to derive equivalent formulae for more complicated cases.

APPENDIX C: Approximating Variance in Relative Discrepancy

The relative discrepancy δ_{ijk} was defined in Chapter 3 as the error metric used to quantitatively compare results from different mesh tally implementations. However, each result computed by a mesh tally also includes an associated statistical error. This means that there will be some variance in the relative discrepancy, depending on how many particle histories were used. Approximating this variance can be done using error propagation theory, based on the assumption that all tally results can be treated as independent random variables. Given *n* independent random variables $x_1, x_2 \ldots x_n$ with known statistical errors $\sigma_{x_1}, \sigma_{x_2} \ldots \sigma_{x_n}$, the error propagation formula for some function $f(x_1, x_2 \ldots x_n)$ is defined by [43]:

$$\sigma_f^2 = \left(\frac{\partial f}{\partial x_1}\right)^2 \sigma_{x_1}^2 + \left(\frac{\partial f}{\partial x_2}\right)^2 \sigma_{x_2}^2 + \ldots + \left(\frac{\partial f}{\partial x_n}\right)^2 \sigma_{x_n}^2 . \tag{C.1}$$

For the relative discrepancy between cell-averaged results obtained using KDE integral-track and conventional mesh tallies, the function *f* refers to Eq. 3.1. This function consists of two random variables: $\tilde{\varphi}_{ijk}$ for the KDE integral-track mesh tally, and $\bar{\varphi}_{ijk}$ for the conventional mesh tally. Using Eq. C.1 allows the variance of the relative discrepancy between these two mesh tallies to be approximated by:

$$\sigma_{\delta_{ijk}}^2 \approx \left(\frac{\partial \delta_{ijk}}{\partial \tilde{\varphi}_{ijk}}\right)^2 \sigma_{\tilde{\varphi}_{ijk}}^2 + \left(\frac{\partial \delta_{ijk}}{\partial \bar{\varphi}_{ijk}}\right)^2 \sigma_{\bar{\varphi}_{ijk}}^2 = \left(\frac{\sigma_{\tilde{\varphi}_{ijk}}}{\bar{\varphi}_{ijk}}\right)^2 + \left(\frac{\tilde{\varphi}_{ijk}\sigma_{\bar{\varphi}_{ijk}}}{\bar{\varphi}_{ijk}^2}\right)^2 . \tag{C.2}$$

Note that the statistical error for the conventional mesh tally $\sigma_{\bar{\varphi}_{ijk}}$ can be obtained by simply multiplying the relative standard error reported by DAG-MCNP5 by $\bar{\varphi}_{ijk}$ for mesh cell (i, j, k).

For obtaining the statistical error for the KDE integral-track mesh tally $\sigma_{\tilde{\varphi}_{ijk}}$, however, an additional approximation is needed because the relative standard error that is reported only refers to the original nodal-based results.

Since all tally results are assumed to be independent random variables, then Eq. C.1 can be also used to propagate the statistical error from the nodal-based results. The function *f* in this case refers to the node-to-cell conversion formula defined by Eq. 3.9. This formula converts the known tally results at the *n* nodes of mesh cell (*i*, *j*, *k*) into an equivalent cell-averaged result using Gaussian cubature with *p* cubature points. To approximate the variance of this cell-averaged result, the first step is to determine the derivative of Eq. 3.9 with respect to the tally result at every mesh node $\hat{\varphi}_m$:

$$\frac{\partial \tilde{\varphi}}{\partial \hat{\varphi}_m} = \frac{1}{V}\sum_{l=1}^{p} w^l j^l \frac{\partial}{\partial \hat{\varphi}_m}\left(\sum_{k=1}^{n} \hat{\varphi}_k N_k^l\right) = \frac{1}{V}\sum_{l=1}^{p} w^l j^l N_m^l, \tag{C.3}$$

where $m \in [1, n]$. Substituting the derivatives defined by Eq. C.3 for all *n* nodes into Eq. C.1 produces the following approximation for the variance of the converted KDE integral-track mesh tally result $\sigma_{\tilde{\varphi}_{ijk}}$:

$$\sigma^2_{\tilde{\varphi}_{ijk}} \approx \sum_{m=1}^{n}\left(\frac{\partial \tilde{\varphi}}{\partial \hat{\varphi}_m}\right)^2 \sigma^2_{\hat{\varphi}_m} = \sum_{m=1}^{n}\left\{\frac{\sigma_{\hat{\varphi}_m}}{V}\sum_{l=1}^{p} w^l j^l N_m^l\right\}^2, \tag{C.4}$$

where $\sigma_{\hat{\varphi}_m}$ is obtained by multiplying the relative standard error reported by DAGTally by $\hat{\varphi}_m$ for the m^{th} node. Therefore, using Eq. C.4 with Eq. C.2 and the actual tally results provides a method for approximating the variance of the relative discrepancy for mesh cell (*i*, *j*, *k*).

APPENDIX D: Correction Matrix for the 3D Boundary Kernel

The 3D boundary kernel method described in Chapter 4 for second-order kernels solves a 4x4 matrix system representing a set of constraints needed to force O(h^2) bias for all calculation points. This appendix shows how the order of the 3D correction matrix (see Eq. 4.40 and 4.41) can be reduced to 1D or 2D, depending on the number of dimensions that need correcting.

Since it is assumed that the 3D kernel function $K(u, v, w)$ is separable, each dimension will have its own distinct set of 1D partial moments:

$$\textbf{i}.\ a_0^u = \int_{u_{\min}}^{u_{\max}} K(u)du, \qquad \textbf{ii}.\ a_1^u = \int_{u_{\min}}^{u_{\max}} uK(u)du, \qquad \textbf{iii}.\ a_2^u = \int_{u_{\min}}^{u_{\max}} u^2K(u)du. \tag{D.1}$$

If boundary correction is not needed, then Eq. D.1 will always equate to $a_0 = 1$, $a_1 = 0$, and $a_2 \neq 0$ for all three dimensions u, v, and w. These are fixed properties of second-order kernel functions. When boundary correction is applied to any given dimension, however, then Eq. D.1 can no longer be integrated over the full domain of $K(u)$ for that dimension. Therefore, the values of a_0, a_1 and a_2 will change depending on the distance to the boundary in that dimension.

As a first example, suppose that only the u dimension needs correcting. This means that for the other two dimensions: $a_0 = 1$, $a_1 = 0$, and $a_2 \neq 0$. Substituting these values for the partial moments into Eq. 4.40 produces the following matrix system:

$$\begin{bmatrix} a_0^u & a_1^u & 0 & 0 \\ a_1^u & a_2^u & 0 & 0 \\ 0 & 0 & a_0^u a_2^v & 0 \\ 0 & 0 & 0 & a_0^u a_2^w \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{D.2}$$

Therefore, both α_2 and α_3 are zero and the 3D boundary kernel reduces to:

$$K_b(u,v,w) = (\alpha_0 + \alpha_1 u)K(u)K(v)K(w), \tag{D.3}$$

where α_0 and α_1 can be obtained by solving the reduced 2x2 matrix system for the 1D boundary kernel defined by Eq. 4.34.

Now, suppose that both the *u* and *v* dimensions need correcting, but the *w* dimension does not. Substituting $a_0 = 1$, $a_1 = 0$, and $a_2 \neq 0$ for the partial moments of the *w* dimension into Eq. 4.40 produces the following matrix system:

$$\begin{bmatrix} a_0^u a_0^v & a_1^u a_0^v & a_0^u a_1^v & 0 \\ a_1^u a_0^v & a_2^u a_0^v & a_1^u a_1^v & 0 \\ a_0^u a_1^v & a_1^u a_1^v & a_0^u a_2^v & 0 \\ 0 & 0 & 0 & a_0^u a_0^v a_2^w \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{D.4}$$

As before, α_3 is zero. However, for this example the 3D boundary kernel includes an additional term for the *v* dimension in the correction factor:

$$K_b(u,v,w) = (\alpha_0 + \alpha_1 u + \alpha_2 v)K(u)K(v)K(w), \tag{D.5}$$

where α_0, α_1, and α_2 can be obtained by solving the reduced 3x3 matrix system for the 2D boundary kernel.