The Dissertation Committee for Cem Bagdatlioglu Certifies that this is the approved version of the following dissertation:

A Hybrid Global Surrogate Modeling Software for Nuclear Reactor Cross Section Estimation

Committee:

Sheldon Landsberger, Supervisor

Steven R Biegalski

Derek A Haas

Anthony M Scopatz

A Hybrid Global Surrogate Modeling Software for Nuclear Reactor Cross Section Estimation

by

Cem Bagdatlioglu

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2017

Dedication

I dedicate this dissertation to my family for their unconditional love and support.

Acknowledgements

I am most grateful to Erich Schneider for his contributions to my work as well as my growth and education overall. This work would not have been possible without him. Now complete, it is so close to the original work we had envisioned.

I would like to express the deepest gratitude to Sheldon Landsberger. He has played many crucial roles in my graduate school career. It has been a pleasure working with him throughout rough times and worse. Without him this work may have never been completed.

I would like to give a very special thank you to Robert Flanagan, who has guided and helped me throughout my graduate school career. Thank you, Robert, for being ready to help whenever I needed it. You can't stop, won't stop!

Very special gratitude goes out to Anthony Scopatz, Derek Haas, Steve Biegalski, and Paul Wilson for all their help.

I would like to thank my fellow researchers Maggie, Will, Birdy, Harry, Jose, and Neal for their help and friendship throughout my graduate school career. Y'all are irreplaceable friends and colleagues!

Most importantly, I want to thank and acknowledge my family. I cannot express how grateful I am for my parents Dilek and Cemo, my sister Ece, and my wife Mel.

A Hybrid Global Surrogate Modeling Software for Nuclear Reactor Cross Section Estimation

Cem Bagdatlioglu, Ph.D.

The University of Texas at Austin, 2017

Supervisor: Sheldon Landsberger

Nuclear fuel cycle (NFC) simulators track the amount and composition of materials as they move through facilities such as mines, fuel fabrication plants, and nuclear reactors. A major task of a NFC simulator is to calculate the evolution of compositions of batches of nuclear materials as they are transmuted in reactors, decay, and are blended with other batches to create reactor fuel or be reprocessed or disposed. Codes used for NFC simulation that utilize intermediate data saved in databases which are calculated ahead of time are attractive since their fidelity can be improved by investing more resources in expanding their databases. Shifting the computational work ahead of the reactor simulation like this allows the fidelity to be improved without sacrificing runtime computational cost. This dissertation describes a method that attempts to maximize the fidelity increase per unit time invested during this precomputation step. Unlike previous work in the reactor simulation field, this methodology does not limit the number and type of runtime simulation inputs. NUDGE (NUclear Database GEneration software) is an implementation of this methodology. The methodology has two main steps where new data is added to databases. First is exploration, where inputs to the database are selected to be as uniformly distributed as possible within the problem input domain. Second step is exploitation, where output information is utilized to inform the selection of the next point to run. An improvement to exploitation, named Voronoi Cell Adjustment, is described in this dissertation and implemented in NUDGE. This improvement has been shown to benefit the average fidelity

increase during database building. A study of the scaling of the methodology, a comparison of error metrics, and an exploration of optimal values for several key parameters in the methodology are presented. NUDGE has also been used to create a global surrogate model of a NFC simulation software (named XSgen). This model shows better performance compared to models generated by other established methods under equal constraints.

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

Nuclear fuel cycle (NFC) simulators track the amount and composition of materials as they move through facilities such as mines, fuel fabrication plants, and nuclear reactors. A major task of a NFC simulator is to calculate the evolution of compositions of batches of nuclear materials as they are transmuted in reactors, decay, and are blended with other batches to create reactor fuel or be reprocessed or disposed. The compositions are used to assess supply and demand of materials, feasibility, impact and infrastructure requirements for fuel cycle facilities reactor technologies, and material accountability in the studied fuel cycle.

The challenge of determining continuously evolving compositions is a distinct feature of NFC simulators. While accurately calculating the material balance of a reactor within a narrow range of possible inputs is time-intensive but straightforward, this challenge is compounded in many NFCs. Advanced NFCs can incorporate a wide range of novel reactor technologies, in many of which fuel is reprocessed and recycled back into reactors. Since inputs depend on outputs in these recycling cases, the range of possible input compositions can become very large. This reality means that simulation software and methodologies targeted for specific reactor types with narrowly bounded inputs cannot easily be generalized to cover arbitrary fuel cycles and reactor technology combinations.

It is straightforward to add the ability to simulate a reactor technology to a simulator only if its space of feasible input fuel compositions and associated input to discharge composition transformations can be easily calculated at runtime, or preferably pre-calculated and parameterized before runtime. Since it is not tractable to model the spectral and burnup effects of changes in initial composition comprehensively by simulating all plausible combinations of every nuclide that may potentially be present in the input, simulation designers must introduce approximations which give rise to the well-known trade-off between accuracy and simulation time. As the precision of results are improved their cost typically increases.

Reactor simulation codes that utilize results or intermediate data saved in databases which are calculated ahead of time are attractive since their fidelity can be improved by investing more resources in expanding their databases. Shifting the computational work ahead of the reactor simulation allows the fidelity to be improved by expanding the results database without sacrificing runtime computational cost. This gives rise to the optimization goal of maximizing fidelity increase per unit time invested.

One such code that utilizes databases to improve fidelity is the Bright-lite reactor simulator [1] that is used in the NFC simulator Cyclus [2]. Bright-lite addresses the trade-off between accuracy and cost by shifting the computational burden of simulation to the front-end by utilizing pre-computed reactor libraries. These libraries are generated from time dependent cross-sections and track the neutron economy, burnup, and transmutation of input fuel nuclides as a function of fluence. The libraries are created and saved in library databases before the NFC simulations. The input values for these libraries are determined by defining plausible domains for the characteristics and range of inputs of reactors that will be modeled.

During runtime, the NFC simulator provides the Bright-lite reactor the composition and mass of all available materials that could be used to create a reactor's next fuel reload. Bright-lite is used during runtime to compute an appropriate blend of these materials given user-specified performance goals and approximate fuel burnup behavior from the Bright-lite reactor library. This method maintains quick execution with acceptable errors as long as the input composition does not deviate far from the initial assumptions made to create the pre-computed library [3].

If Bright-lite's available pre-computed libraries were not prepared using design parameters and input fuel composition closely matching that of the studied reactor, hard-to-quantify and potentially large errors may arise. To address this discrepancy, Bright-lite can approximate a library using an interpolation method on the available libraries [4]. The interpolation is carried out based on parameters which are generally not known until runtime (e.g., isotopic vectors of plutonium or transuranic fuel feed stocks). These inputs may vary between reactors of the same general type; they may also vary between fuel

reloading batches for a given reactor. The neutronic effects of such variations are addressed by interpolating on libraries in the database.

Increasing the number of precomputed libraries therefore reduces error by reducing the average distance between the interpolating libraries and the target point in input space. At present, though, since the libraries are generated in an ad hoc manner, it is not possible to ascertain or even bound the error introduced by using the interpolated library versus a library generated by carrying out a time-dependent radiation transport/burnup calculation. Verifying each interpolated library by performing a full simulation would be too time-intensive a step for an NFC simulator; it would also undermine the intended benefit of moving the transport calculations to a pre-computation step.

Generation of Bright-lite's precomputed libraries can be time consuming since it involves coupled transport-burnup calculations on a representative reactor unit cell or lattice. The parameters for the case to be simulated, such as the total time fuel will be burned, can strongly affect the execution time of the model used to generate the fluence-dependent cross sections, reaction rates, and transformation matrices that comprise the libraries. When categorical inputs (such as reactor type and number of core regions) are defined, there are still many (material densities, fuel cell geometry, fuel composition, fuel power, etc.) continuous variables in the input space. This high dimensionality makes laying a fine high-dimensional grid (i.e., each variable with many discrete values) impossible. For example, assuming the input space contains 20 continuous variables, assigning 6 discrete values for each of them and simulating every combination will require quadrillions of cases to be simulated. Even if each case required only 6 seconds, this would take as long as the half-life of U-235 (703,800,000 years) to complete.

However, this high dimensionality can be mitigated since many of these variables might have very negligible effect on neutron energy spectra and spatial distributions, and thus on material balances. Other inputs may have limited effects on outputs over most of their domain, but rise to significance in one region. A methodical, brute-force sweep considering every combination of inputs like the one described above ignores these facts. Therefore, the process of selecting inputs for libraries to be included in the Bright-lite

library database must be carefully considered to prioritize areas of the input space that sensitively affect outputs.

The challenge of retaining much of the fidelity of costly simulations without incurring their cost during runtime arises in many fields of science and engineering, and motivates the work to create a surrogate model (response surface model, metamodel, or emulator) which estimates the output of the simulation model (or full simulation, black box model). The competing goals of the methodology are minimizing the total time it takes to generate the database and bounding the interpolation error across the entire global domain of inputs. Minimizing the database generation time implies limiting the number of full simulations and concentrating expensive simulations in areas where outputs are found to sensitively depend on inputs. In cases with limited computational time, the goal is to use resources effectively to lower the final database error.

Unlike previous work in the reactor simulation field, this methodology does not a-priori limit the number and type of runtime simulation inputs. Any number of inputs can be specified and will be included in the database variable space.

The next section covers the literature in related fields including NFC simulators and fields related to building and testing a surrogate model. It demonstrates the novelty of the work as well as summarizing the selected methods and concepts for the methodology.

2. LITERATURE REVIEW

This section summarizes fields related to the work with two goals. First is to review approaches taken by NFC simulators to handle the challenge of modeling a large range of reactor types, configurations, fuel forms, and fuel compositions. Many NFC simulators assemble databases (of cross sections, material balances, or inputs to reduced-order models) by performing full-scale transport and burnup calculation in advance rather than runtime. This leads to a class of analogous problems in other fields. The second goal is to review approaches to handle costly simulations, dimensionality reduction, and uncertainty estimation in other fields.

The section is divided into six subsections. The first is a review of methods nuclear reactor simulators use to reduce runtime. Next, the field of global surrogate modeling, which deals with making a model of a simulation model, is introduced. Relevant concepts and definitions are defined in this subsection. The third subsection covers ways to estimate errors of surrogate models. Next, popular techniques for dimensionality reduction are presented, which will be necessary to screen inputs as well as to analyze and quantify outputs. Finally, similar applications in other fields of science and engineering are given.

2.1. Review of Methods Used to Reduce Nuclear Reactor Simulation Runtimes

This subsection will discuss methods engineers and software designers use to overcome the long runtimes required by high precision nuclear reactor simulations while minimizing the resulting loss of accuracy. As mentioned, the high precision simulations are costly. The iterative nature of the calculation prevents any meaningful separation of various parts of the workflow without significant changes to the simulation model.

Many input parameters are needed in order to accurately simulate the isotopic transformations that take place during fuel burnup. Some of these inputs such as core geometry, material densities, and coolant temperature are generally constant for a given reactor. Others such as fuel composition, local fuel power densities, and cycle length may change throughout the lifetime of the modeled reactor. A reactor simulation with very high precision (full simulation) requires values for all of these inputs to be determined before the calculation begins and does not make assumptions on inputs. These models require new simulation runs to be carried out even for small changes in individual inputs. In addition, simulations are very time consuming. Using a high fidelity reactor model within a NFC simulator is therefore generally unfeasible since the NFC simulation may have dozens of reactors each requiring runtimes totaling to prohibitively long simulations.

Building reactor simulation models that have sufficiently quick runtimes while maximizing accuracy involves simplifying the workflow to reduce the inputs needed during runtime and performing most of the costly computations prior to NFC runs. The most extreme reduction of inputs required during runtime is utilized in recipe reactors. A recipe reactor takes a constant input composition and always discharges the same resulting output. All reloads to that reactor type are treated as following that recipe. This reduction is utilized by the Code for Advanced Fuel Cycles Assessment (CAFCA) [5]. Developed by Massachusetts Institute of Technology, CAFCA is a fuel cycle simulator that utilizes only constant pre-computed data for its NFC material balances. Therefore, each reactor fleet in CAFCA has an invariable flow of input-output compositions, also called recipes. These

recipes (including core mass, cycle length, and capacity factor as well as compositions of fresh and used fuel) are provided through Excel spreadsheets.

A tool that does not sacrifice all dimensions for runtime calculations by using databases of recipes parameterized against major inputs is the Verifiable Fuel Cycle Simulation (VISION) [6]. Developed by the Idaho National Lab, VISION software is a tool for modeling material flows in a NFC. The software is a Powersim application, utilizing Excel spreadsheets for input and output functionality. The outputs are generated externally. The results of these calculations are saved as recipes to determine outputs of reactors by adjusting inputs such as composition, initial reactor core loading, and loading per fuel batch. VISION accounts for evolving input compositions "using interpolation within tabulated values, correlations of recipes as a function of key input parameter (e.g. UOX burnup) or using a perturbation method to cover the possible range of operations of certain type of fuel" [6].

Another NFC modeling software which reduces inputs needed during runtime is SImulation TOol for modelling the Nuclear fuel cycle (SITON) [7]. SITON uses fixed recipes for some reactor calculations. However, for reactor types where the output is determined to be sensitive to evolving input compositions (such as plutonium recycling) it uses a method its authors call FITXS where polynomial functions are fit to one-group microscopic cross-sections. The number densities of important actinides and fission products are used as fitting parameters, allowing the changing input fuel composition to be used to quickly estimate cross-sections during runtime.

The three main parts of the method are the selection of fitting parameters, transport calculation runs to generate a training database, and the determination of fitting parameters. The evaluation of the fitting parameters for the FITXS method requires several thousand training data points in the training database. The discrete points to sample for this database are chosen by random sampling of mass fractions after several constrains, such as ranges for Pu and MA fraction, are imposed. The number of inputs that can affect the material balance at runtime is determined by the chosen fitting parameters.

Commelini-Sicart (COSI) [8] is a nuclear fuel cycle simulator that uses the equivalence model for initial composition calculations and a Bateman equation solver with cross-sections obtained from transport calculations for depletion. The equivalence model effectively attempts to reduce the properties of isotopes in the available fuel streams into a single dimension. The aim is to determine a blend of available fuel materials that maintain core criticality at a given burnup. COSI comes with hard-coded equivalence model reduction parameters for re-enriched reprocessed uranium and MOX fuel streams. These models are generated using benchmark data and each model has an associated range for input composition. The users are also allowed to build custom models.

Depletion calculations in COSI are done using the CESAR software [9]. CESAR uses one-group cross-section libraries in its simplified depletion calculations. The validity of the cross-sections determine the validity range of inputs for the software. These ranges are given in terms of initial composition and maximum burnup. Therefore COSI effectively reduces the runtime computation to two or three dimensions.

It has been shown that there are many approaches taken to reduce NFC simulation runtimes while keeping error bounded within a desired range. One measure taken by most simulators is to limit the number and range of applicable runtime inputs. While some NFC simulators resort to recipes where the allowed inputs are reduced to a single categorical type, others such as the SITON approach parameterize material balances or surrogate models with respect to multiple input variables. However even the most general and extensible approach reviewed, that taken by SITON, requires expensive training of the database, limits to a few inputs that can be varied, cannot be easily generalized if other inputs (especially inputs of different categories such as geometry), and applies to only one reactor type. The aim of the work presented here is to expand the allowed inputs to dozens or hundreds in a manner that is flexibly applicable to any reactor type. The remainder of this section outlines methods for achieving this goal and reviews those used within analogous applications found in the literature.

2.2. Global Surrogate Modeling

Global surrogate modeling is the process of generating a model of a simulation software. A simulation model is defined as the digital prototype of a physical model used to emulate real world systems. A simulation model is run when system behavior at a specific set of inputs (or dimensions, parameters, levels) is evaluated. This subsection reviews methods to handling cases where the full simulation requires prohibitively high runtimes to complete, and/or total runtime becomes prohibitively large because many runs of the full simulation are expected. In these cases a fast executing global surrogate model is generated using outputs from the full simulation. Global surrogate models are unlike local surrogate models where only a subdomain of the problem space is approximated, usually to guide optimization, and later thrown out when an optimum is found.

Global surrogate modeling strategies assume the full simulation is deterministic, a black box model, and that there is no knowledge of the relationship between each input and output, for instance whether it is linear or even continuous [10]. In addition, these assumptions guarantee that there is no random error in the results. This implies that the surrogate model should not smooth across data points of the simulation model and should return the output of each input in the database exactly.

Space-filling design is a field of design of experiments which aims to spread out inputs across the design domain as uniformly as possible [11]. Unlike physical experiments, space-filling design does not need to have replication, blocking, or randomization [12]. If the total number of samples (input sets that are run in the full simulation) is determined beforehand and cannot be easily modified later it is called a one-shot design. On the other hand, if the design points are selected in groups after some sampling has occurred the process is named sequential design. Sequential design is beneficial in preventing oversampling and undersampling. The former occurs when a well-defined area of the domain is sampled without improvement to the surrogate model whereas the latter fails to prioritize picking areas that need more samples.

A desirable space-filling design has three features. First feature is good space-mapping properties. This means the selected samples must evaluate the space evenly,

attempting to extract as much information as possible from each sample [13]. Second is granularity, which is concerned with how samples are selected. One-shot designs have no granularity as all the samples are selected at once. A perfectly granular design will select samples one at a time with no upper limit on how many samples that can be generated. The final feature is good projective properties. Also called the non-collapsing property, a space-filling design with good projective properties ensures unique values of each variable for every sample. In other words, no two points should overlap when a d-dimensional design is projected on to a lower dimension.

Perhaps the simplest to implement one-shot design is a factorial experiment. It assigns a set of discrete values (also called levels) to each variable and samples every combination of input levels in the domain. Under most cases this design is infeasible due to the scaling of number of samples needed as number of inputs is increased. The number of cases scale as L^D where L is the number of levels per input and *D* is the number of inputs. An alternative is fractional design, which picks a subset of a factorial design and limits the total number of samples [14]. Various strategies exist on selecting a fractional design with good space-filling properties [15]. These strategies generally assume the outputs are dominated by main effects and low-order interactions (known as the sparsity-of-effects principle) to select a subset of the full factorial design. Fractional factorial designs are usually optimized for the specific number of levels, inputs, and desired runs.

Latin hypercube sampling (LHS) [16] is one of the most popular techniques used for computer experiments. LHS imposes additional requirements on fractional designs to obtain beneficial properties. The 2-dimensional analogy of LHS is the Latin square, where a square grid is laid and a sample is placed if and only if there are no other samples in the same column and row. LHS is the arbitrary-dimension equivalent of the Latin square where every dimension has an equal number of levels (named factors). It is important to note that given a number of dimensions and factors, there are many ways to fulfill the sampling requirements of LHS and therefore designs must be optimized for a relevant space-filling criterion. LHS optimizations become increasingly difficult as the dimensions and factors increase since the number of possible solutions scale as $L!^{D-1}$ [17]. Adding granularity to

LHS is also difficult but possible, such as using methods that utilize maximin distances [18] or low granularity nested designs that embed a LHS design within a coarser one [19].

Factorial experiments, fractional designs, and LHS are all one-shot designs. In all standard implementations of these methods the total number of samples must be decided before any sampling takes place. However, this lack of granularity prevents any feedback from the results of simulation runs. For example, if it is found that additional samples are needed after analyzing the results, most one-shot designs do not have a methodology to do so without sacrificing space filling properties. Sequential design should be used in order to efficiently sample the simulation domain by selecting as many new points as needed [20]. Sequential design allows for the analysis of existing outputs and using this information for the selection of next samples.

Using outputs to inform the selection of inputs is named exploitation. It is not possible to utilize exploitation on its own however, since at the beginning there is no information about the output space and nothing to exploit. Therefore, exploration is initially needed to capture the broad features of the domain. Given a limited time to generate the global surrogate model, the balance between exploration and exploitation should be carefully considered.

Optimal sequential design methods assume that the surrogate model type is known beforehand. This class of sequential design guides the sampling process based on the model type and the outputs. Examples of model-specific designs can be found in [21], where Gramacy and Lee utilize a model specific method that combines optimized one-shot strategies with active learning, and in [22] where the authors utilize a sampling methodology based on a multivariate rational interpolation model.

Generic sequential design methods, on the other hand, do not utilize information on the type of surrogate model for sample selection. Instead, only the locations of the sampled points and their outputs are used to guide sample selection. Since there could be little or no prior information known on the simulation model behavior, choosing an appropriate surrogate model may not be possible beforehand. Therefore, generic sequential designs are favored when the surrogate model could be changed later as more information on the

simulation model is gathered. In addition, generic sequential designs allow testing of various surrogate models from the same dataset.

A review of generic sequential designs is given in [23]. These methods utilize one or more criteria in the evaluation of candidate sample points. The points can be algorithmically generated or a Monte Carlo approach can be utilized. In the case of Monte Carlo methods, the number of candidate points generated at each iteration is scaled linearly with the number of existing sample points. Monte Carlo methods are generally preferred due to their computational efficiency and granularity [15]. Any number of criteria (such as average distance, maximin distance, projection) can be used to evaluate candidates points generated during the Monte Carlo algorithm.

As the name implies, the average distance criterion finds the average distance of the selected point to the rest of the points in the dataset. A more useful measure is the maximin distance, which finds the highest of the minimum distances between the selected point and the rest of the points in the dataset [24]. This criterion ensures that the points are selected so that the minimum distance between any two points are maximized in the dataset. As a result, the criterion "tends to place a large proportion of points at the corners and on the boundaries of the hypercube" that represents the dataset [25].

2.3.Database Error Estimation

It is necessary to bound the error of using the surrogate model compared to running the full simulation model for the same sample. The error can be used for the estimating the uncertainty in the results of the surrogate model as well as assessment and improvement of the surrogate model itself. Since there are no assumptions made about the outputs of the full simulation, this error is expected to depend on part of the domain as well as proximity to a sample point in the surrogate model database. Therefore the average error as well as the maximum error across the domain are relevant quantities to consider.

The best estimate of the error can be found by generating many random samples, running them in both the full simulation model and the surrogate, and comparing the results. This process, given enough random points, can give the *true error* of the surrogate model. However, this method requires numerous samples to be run after the surrogate model is built. Since a starting premise in this work is that the full simulation is costly, the error estimation method should not require any additional full simulation runs, not to mention many additional runs. Therefore finding the true error is very expensive and cannot be used in a practical application of the surrogate model.

One of the error estimation methods that does not require additional full simulation runs is called split sample [26]. The method divides the simulation run data into a *training* and a *testing* set. The training set is used in the surrogate model and the testing set is used to estimate the error. The results from this method rely heavily on how the data is split and the results therefore can show high variance.

An improvement on split sample for error estimation is cross-validation (also called leave-k-out approach [27]) [26]. Instead of dividing the data in two, this scheme divides the data into many sets of k points. Then, a surrogate model is constructed for all combinations of the data that exclude one of the sets. This procedure results in an error measure for each set, which are later averaged to get an estimate of the error. If k=1 the method is named leave-one-out. It is recommended to use k=1 for surrogate models with radial and low order polynomial functions (like the surrogate model used in this work) for best accuracy and precision of estimating the error [27].

2.4.Dimensionality Reduction

Dimensionality reduction techniques attempt to reduce the number of independent variables in a dataset while retaining as much of the dependence of the outputs on the variables as feasible. Dimensionality reduction is used to minimize the time and storage space required for data, remove multicollinearity (where two or more variables are highly correlated), and assist in data analysis and interpretation across a wide range of information processing fields. Dimensionality reduction is necessary within the context of this work since the output from the full simulation model consists of fluence-dependent data that total to hundreds of values per sample. In order to analyze the outputs they need to be quantitatively compared, which will be achieved with dimensionality reduction. Several common dimensionality reduction techniques are introduced next.

In cases where values of a given variable (or dimension) are only weakly correlated with outputs within the dataset, a low variance filter can be used [28]. Data that shows minimal change does not carry much information, and can be removed. In contrast, highly correlated variables carry very similar information. In these cases a high correlation filter can be used to eliminate the excess variables. Both of these methods utilize the variation in data fields to evaluate the usefulness of their carried information.

Another dimensionality reduction technique is forward feature construction. This method starts by only including the variable that best explains the dataset behavior based on the covariance matrix of the dataset [29]. Next, new variables are added one at a time where each new variable is the best predictor of dataset behavior among the remaining variables. The addition of new variables stops when the selected variables sufficiently account for the output behavior. Backwards feature elimination utilizes the same information but instead starts with all the variables and removes them one a time. The methods discussed so far are geared towards very high dimensional datasets and may not be applicable in many cases where the number of starting dimensions is low.

One of the most widely used dimensionality reduction techniques that applies to very low dimensional as well as high dimensional datasets is principal component analysis (PCA). This method applies orthogonal transformation to the original set of variables to

generate a new set of linearly uncorrelated variables called principal components. “This is done by finding a linear basis of reduced dimensionality for the data, in which the amount of variance in the data is maximal” [30]. The first principal component is responsible for the highest variance in the dataset, followed by the second one, and so on. Even though the dimensionality is conserved, the later principal components can be discarded as they explain the least variation in the data. The number of principal components to retain depends on the importance of reducing dimensions in the dataset as well as the accuracy to be maintained. The units and ranges of variables need to be normalized as the transformation is highly sensitive to scaling.

2.5. Applications of Global Surrogate Modeling in Engineering

Global surrogate design techniques, which include space-mapping methods, are used in many fields of science and engineering. Six such applications of surrogate modeling are discussed below.

Crevecoeur *et al.* describe a method to optimize the electroencephalogram (EEG) measurement process utilizing similar methods [31]. In this work the simulation model solves an inverse problem that finds the best fit of the source to the measurement. The authors utilize a space-filling method to construct a "coarse" surrogate model for this problem. This work is similar to generating a surrogate model for use during a NFC simulation, since the surrogate model is also coarse.

Redhe and Nilsson present work on vehicle crash modeling where surrogate models are used for structural optimization of crashworthiness [32]. The authors use space-filling methods to construct a surrogate model and use this model during structural design optimization. The surrogate model in this work improves the time to find the optimal solution. It is similar to NFC optimization since the surrogate model is used to improve the time to achieve the final solution that is found using the full simulation model.

Hintermulller and Vicente describe the utilization of space-filling methods to solve optimal control problems for nonlinear partial differential equations [33]. These problems present "a significant numerical challenge due to the tremendous size and possible model difficulties (e.g. nonlinearities) of the discretized problems" [33]. The authors utilize space-filling technique that makes it easy to utilize different surrogate models. Again, similar to the surrogate modeling approach presented in this work, the authors use surrogate models to estimate the results of a full simulation model.

Encica *et al.* use space-mapping on the optimization of a cylindrical voice coil actuator [34] while Tu *et al.* use similar methods for the optimization of handset antennas [35]. Khlissa *et al.* describe a space-mapping methodology to optimize a permanent magnet machine used as an integrated starter generator, for instance for use in hybrid vehicles [36].

The authors of these works use the surrogate model to optimize the design of an engineered system and later switch to the full simulation model during the final stage of the design process. Utilizing the surrogate model and later switching to the full model is also used for high fidelity NFC optimization.

2.6.Summary of Selected Methods for Surrogate Modeling Workflow

Several NFC simulators have been reviewed. Currently there is no nuclear reactor simulation tool that allows the degrees of freedom during runtime as is proposed in this work. These high degrees of freedom will be achieved by creating a surrogate model. As discussed in the Global Surrogate Modeling subsection, the process will involve three main parts: screening, exploration, and exploitation. Screening will be done using quick partial runs of the full simulation to eliminate inputs with sufficiently low effects on outputs as well as to readjust ranges of variables. The methods used for screening will be formulated in the next section.

Exploration will use a Monte Carlo method for space-filling design along with maximin distance and projection criteria. The Monte Carlo approach allows for linear scaling of sample selection for any given number of inputs and sample points as well as being fully granular (selects one sample at a time with no lower or upper limit). The maximin distance criterion will allow for good space filling properties while the projection criterion will enable good projective properties. The granularity and the maximin criterion together will allow the model to be continuously evaluated during dataset generation as well as easy inclusion of pre-existing points in the dataset.

Exploitation will utilize a local gradient estimation methodology that is detailed in the next section. This method of selecting samples by utilizing outputs will be agnostic to the surrogate models method for estimation from the dataset. Exploitation will be hybrid method that considers both the space-filling criteria of the candidate sample as well as the nonlinearity in its vicinity.

The next section formulates the methodology used for screening, exploration, exploitation, and error estimation. The following section demonstrates the techniques with examples and presents results.

3. METHODOLOGY

3.1. Overview

The tool presented in this work is the NUclear Database GEneration software (NUDGE). The software uses an external tool to generate reactor libraries to be included in the reactor library database. Any number of inputs to be included in database generation can be used in NUDGE. The goal of the software is to use computational time as efficiently as possible to minimize the database error. The literature review shows that this can be accomplished by dividing the database generation task into three distinct stages, which will be covered next.

The method treats the tool used to generate reactor libraries as a black box and can function with any software it is able to interface. This treatment also does not make any assumptions about the computational cost of the black box simulation. Both slow (runtime in the order of hours or even days) and fast (runtime in the order of minutes or seconds) full simulations can be used with NUDGE.

In this work, the XSgen software is used to create reactor libraries [37]. It performs full transport-burnup simulations that use fuel, cladding, and coolant composition and densities, unit-cell geometry, and power as inputs to compute multi-group neutron cross-sections, compositions, burnups, and multiplication factors as a function of neutron fluence, the time integral of neutron flux. XSgen achieves this by coupling a neutron transport code (such as OpenMC [38]) to determine group fluxes and multiplication with a transmutation code (such as ORIGEN2.2 [39]) to find burnup and transmutation. Each XSgen run is costly, requiring many hours for a standard (using a single core i7 processor, 4 GB ram, 500 GB hard drive) computer.

Figure 1 visually summarizes the steps taken by NUDGE to generate a database. The three main steps of the workflow are separated by color. The first step shown on the left side of the figure is screening. No points are added to the final database at this stage since the goal is to reduce, if possible, the complexity of the problem before full simulations. This step is used with the black box model runs set to provide fast results,

even at the expense of accuracy. This step intends to identify sensitivities and correlations between inputs and the results. Therefore, speed is more important at this stage provided that the model still responds to inputs the same way. The goal of screening is to avoid performing redundant or unnecessary work during the later stages.

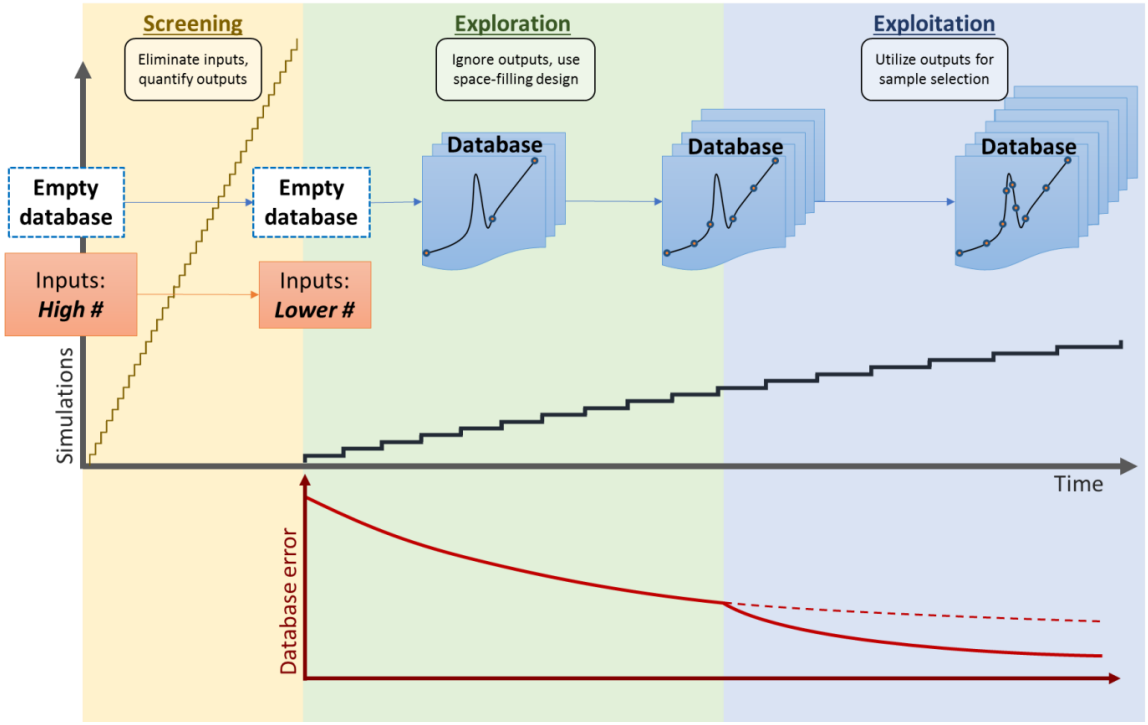


Figure 1 – NUDGE Workflow Overview

Next step is exploration, where inputs are selected to be as uniformly distributed as possible within the problem domain. The error of the database decreases with every additional point, as shown by the plot in the bottom of the figure. The plotted error is the average deviation from the full simulation incurred by using the database. This error depends on the interpolation method being used, as discussed in Section 3.5. An illustrative 1-dimensional output curve depicting the true behavior of the underlying function being computed by the black box tool, and the locations of sampled points are depicted inside the database boxes. At this stage, the simulation results are not yet fed back to inform the selection of the next point to run; instead, points are selected to be as far away from each

other as possible. The goal of this stage is to explore the output space and find important regions in the domain (Section 3.3).

The last step, as also depicted in Figure 1, is exploitation. It can be seen that the new points are focused near the sharp peak of the output. Focusing new points at this region improves the error reduction rate, as shown by the error plot at the bottom of the figure. Finding an appropriate allocation of time between screening, exploration, and exploitation is a key objective of the research. It depends on the runtime of the black box model, topology of the function being approximated, number of dimensions, and much more. For example, a function with constant first derivatives (a slanted plane) can be best identified via exploration alone. A function that is very complicated only in certain areas would benefit from more exploitation once these features are discovered during exploration.

In this work, the d-dimensional set of inputs that are used to run the full simulation are named *points*. The *domain* of the problem is determined by the number of varied inputs (or *dimensions*) and their ranges. Once a point is selected to be in the database it is a *sample*. A point is *sampled* by invoking the black box model to find their input-to-output transformation. The *size* of the database is the number of samples within it.

This section will first cover the methodology of each stage in detail. Next, error estimation will be presented followed by sections for proposed improvements to the methodology.

3.2. User Inputs and Initialization

There are two sets of inputs required by NUDGE methodology to begin generating a database. First input is a base case given by the user. This is the input for the black box full simulation software, where all inputs have an expected or "central" value. The inputs that are not varied in the database (the categorical inputs) are defaulted to the value given in the base case. The base case serves as a template for the XSgen inputs of all samples in a database. The combination of input values in the base case should not produce any errors when run in the full simulation. An example input for a LWR can be found in Appendix 6.1, and an example for a MOX reactor is given in Appendix 6.2.

Second input is the database input, used by NUDGE for database generation. It specifies the inputs to vary during database generation as well as how much to vary each input. In addition, database generation inputs such as tolerance values and database creation criteria (maximum error, mean error, maximum time, maximum number of points, etc.) are included in the database inputs.

The first step in the NUDGE workflow is to read these two sets of inputs. Once this step is completed the screening step begins, which is discussed next.

3.3.Screening

This subsection will discuss the methods used during the screening step. It is assumed that initially there is no data in the database and nothing is known about the relationship between the inputs and outputs. Therefore, screening is used to discover basic behavior of the outputs for the given domain of inputs. The information from this step is used to potentially eliminate those inputs that have sufficiently low correlation with the outputs.

3.3.1. SCOUT RUNS

Scout (‘s’peedy and ‘c’urtailed ‘out’puts) runs make use of less expensive, lower-fidelity executions of the black box model. In the context of this work, XSgen runs that have very low burnup values can be scout runs as well as runs with low number of Monte Carlo particles. Any modification to the standard costly full simulation run to significantly reduce the output generation time while still maintaining the main effects of inputs on the outputs can be considered a scout run.

Scout runs are used during screening to lay a grid on the problem domain much finer than what is possible with full runs. Generating more points for the screening database enables a better characterization of the input-output relationships. The method used to select input values during this step is the same as the method used during the exploration stage, which is discussed in Section 3.3.

Once scouting points are selected and run in the full simulation, a screening database is created. This database is used for screening analysis, as described for the remainder of Section 3.2.

3.3.2. OUTPUT QUANTIFICATION

XSgen outputs are composed of several text files each containing hundreds of values for fuel properties (listed below) at different fluences. To compare results and quantify the differences these outputs need to be reduced to a single representative value for each output. This subsection discusses the use of Principal Component Analysis (PCA) to quantify outputs.

PCA is a popular linear transformation method used in numerous applications in data analysis. It is useful in reducing a larger set of variables into a smaller subset of principal components. The principal components are artificial variables composed of a combination of the original variable set. In this work, the Python programming language package matplotlib is used for PCA analysis [40].

The selection of the outputs to apply PCA can depend on the goals of the database generation. While it is possible to use the entire output from each run to apply PCA, better information can be extracted from the outputs if redundant and inconsequential but highly varying data is removed beforehand. Certain use cases of the final database may dictate that, say, only the mass of an isotope of interest should be considered in outputs. In most use cases, however, outputs related to the neutron economy and burnup are of highest consequence. Therefore the outputs used for output quantification, unless others are specified by the user, are:

- Burnup at highest fluence (BU(F_{max})) [MWd/kgIHM]
- Neutron production rate at highest fluence (PR(F_{max})) [neutrons/kgIHM/s/flux]
- Neutron destruction rate at highest fluence (DR(F_{max})) [neutrons/kgIHM/s/flux]
- Masses of selected nuclides at highest fluence [g/kgIHM]

Note that the highest fluence used during screening will still be well under the maximum fluence for a typical fuel. The isotopes are selected based on reactor type. For a LWR database, for example, U-235 and Pu-239 masses are used. In the XSgen outputs these values are given for each isotope present in the initial fuel loadings. In order to reduce outputs to a single metric and to maintain consistency, the input fuel composition of the base case is used to combine the outputs in the nuclide level to a single fuel-level output

set. Next, the combined fuel-level outputs are normalized using the highest value of each output type in the dataset. These normalized outputs are used for PCA.

PCA returns a set of principal components so that first component best explains the variance in the data. For this reason the first principal component is selected to find the output value of a given point in the database. Due to the nature of PCA transformations it is not always possible to intuitively understand the meaning of the principal components (such as when the neutron production/destruction rates are combined with output compositions to arrive at a scalar principal component value). It can be said that it is the output combination that shows the greatest sensitivity to the inputs. These values are used in input elimination.

3.4.Exploration

This subsection describes the exploration methodology of NUDGE. The aim of exploration is to select points so that the domain is filled as evenly as possible. This subsection will discuss the generation of candidate points and the selection of the next point to sample during exploration.

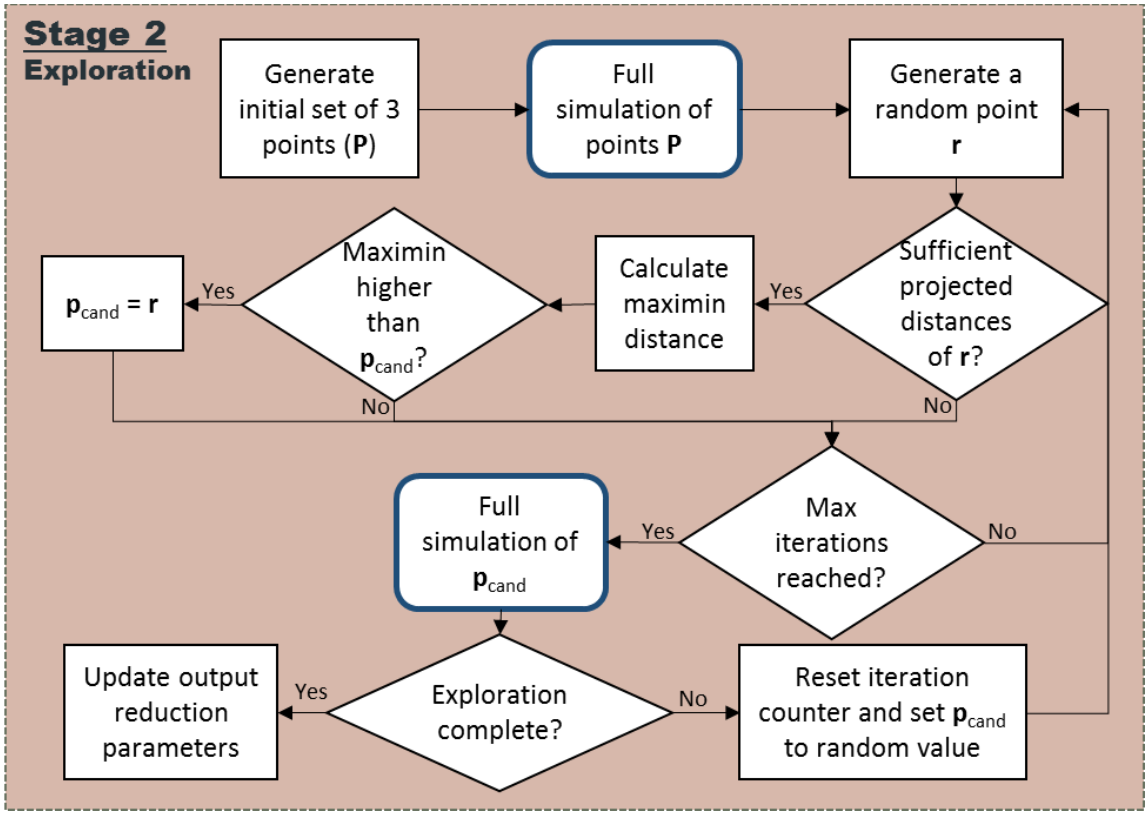


Figure 2 – NUDGE exploitation step workflow

Figure 2 summarizes the steps taken by NUDGE during the exploration step. First, an initial set of 3 input points are sampled. These initial points include the base case as well as two samples with all inputs set to their lowest and highest values. Since the exploration methodology selects new samples based on existing ones, at least one input is needed to begin this stage. The base case is selected by default as it is a representative input set. The

other two cases are chosen due to their simplicity and applicability for any number of dimensions.

Once these points are run, the main loop of the exploration phase is entered and Monte Carlo generation of candidates for the next point begins. Random points are generated and scored according to the selection criteria discussed next. The point with the highest score is selected as the next point to sample, and the process is repeated until exploration is completed.

Candidate points with randomly generated input values are evaluated against two space-filling criteria. The number of candidate random points is scaled linearly according to the size of the database. Given database size N (number of samples in the database) and a user-selected multiplier $k_{explore}$, the number of random points R is found by:

$$R = k_{explore} \cdot N \qquad (1)$$

The first criterion to evaluate candidate points is projected distance, also called the non-collapsing property [41]. The goal of imposing this criterion is to make sure that each point has a unique value for every dimension. For a D-dimensional candidate point input vector $\mathbf{r}=[r^1, r^2, \ldots, r^d, \ldots, r^D]$, set of points of the database $\mathbf{P}=(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n, \ldots, \mathbf{p}_N)$, points in the database $\mathbf{p}_i=[p_i^1, p_i^2, \ldots, p_i^d, \ldots, p_i^D]$, and a projection threshold ϵ_p; the projected distance criterion is:

$$\min_{d \in D}\left(\left|\mathbf{p}_i^d - \mathbf{r}^d\right|\right) > \epsilon_p \quad \text{for all } \mathbf{p} \in \mathbf{P} \qquad (2)$$

Candidate points that score below the threshold are eliminated since in at least one dimension the point will be too close to an already sampled value. Note that all inputs for any given sample i (p_i^d) are normalized so that they range [0, 1]. This is done by subtracting the absolute minimum (p_{min}^d), then dividing the range (p_{max}^d - p_{min}^d) of each input parameter

(such as fuel radius or U-235 fraction) as shown below (the max/min values are provided by the user):

$$\mathrm{p_i^d} = \frac{\mathrm{p_{i,unnormalized}^d} - \mathrm{p_{min}^d}}{\mathrm{p_{max}^d} - \mathrm{p_{min}^d}} \quad \text{for all } \mathbf{p} \in \mathbf{P}, \text{all d} \in \mathrm{D} \tag{3}$$

The acceptable proximity is determined by the threshold value. The threshold needs to be lowered as *N* increases. An initial value of 0.0001 is set for the threshold, and halved each time more than half the candidates are eliminated due to the projection distance criteria. The initial value is set to ensure that the threshold is not increased for the first few samples. The algorithm to update the value has been set for simplicity and may be improved later.

The second criterion is intersite distance, also called the maximin criterion [10]. It uses Euclidean distances to maximize the distance between the closest points:

$$\text{maximin} = \max\left(\min_{\boldsymbol{p}_i, \boldsymbol{r}} \left(\sqrt{\sum_{d=1}^{D} \left(\boldsymbol{p}_i^d - \boldsymbol{r}^d\right)^2} \; for \; \boldsymbol{p}_i \in \boldsymbol{P} \right) \text{ for all } \mathbf{r} \right) \tag{4}$$

For all random points, the closest distance between the random point and any sample in the database is found. The random point that passes the projected distance criterion and has the largest maximin distance is selected as the next sample. The process is repeated for each new point until exploration ends. The duration of exploitation is investigated in the results, and is included in future work.

Generation of the next point during the exploration stage is visualized in 2 input dimensions in Figure 3. The figure displays a database with four existing inputs, each at one corner of the domain, shown in blue large circles. The small green circles represent the randomly generated points. The red point is selected as the next sample since it is furthest away from all other existing blue points.

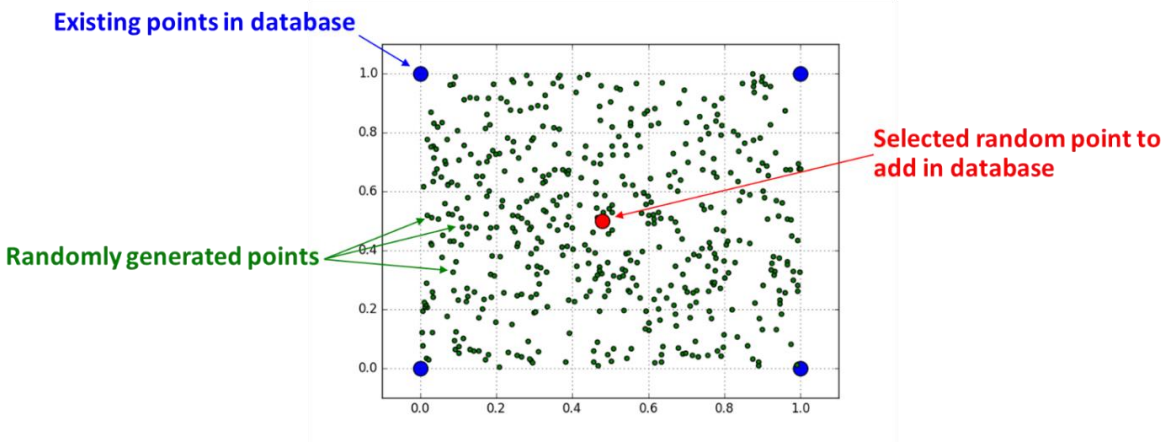


Figure 3 – Next sample selection during exploration.

The selection of the following samples is visualized in Figure 4. The same color scheme as the previous figure is used. The impact of the projection criterion can be observed in this figure, as the points are selected to avoid any overlap in any of the two dimensions (except the first 4 points which were selected beforehand). Although hard to visualize, the same process is used for higher dimensions.

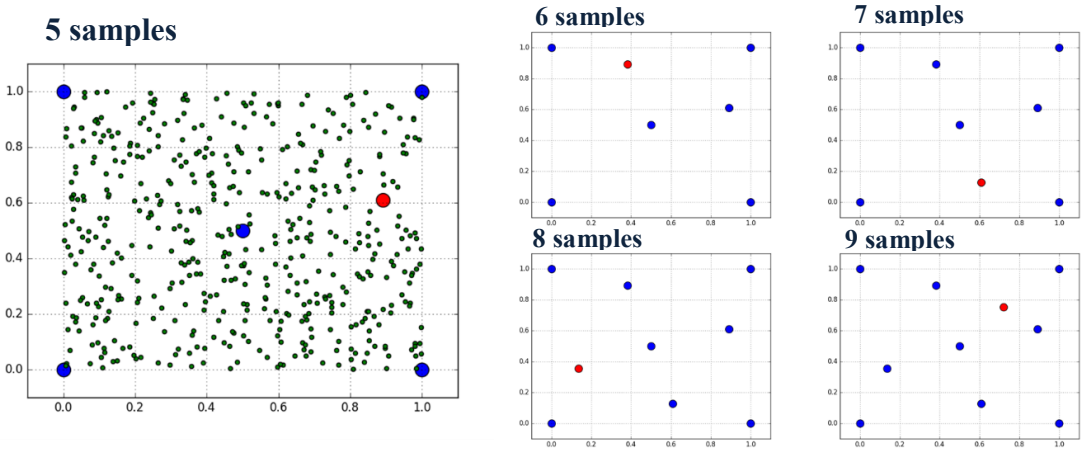


Figure 4 – Selection of five consecutive samples during exploration.

3.5. Exploitation

This subsection details the exploitation stage of NUDGE, where the topology of the output hypersurface is exploited to inform the selection of the next sample. Exploitation is most valuable if the hypersurface strongly varies with respect to one or more inputs in some parts of the domain, and exploitation aims to prioritize sampling near these parts in order to improve the error reduction rate. The cost of performing exploitation is the computational overhead, which may result in fewer total samples in the final database generated with a time constraint.

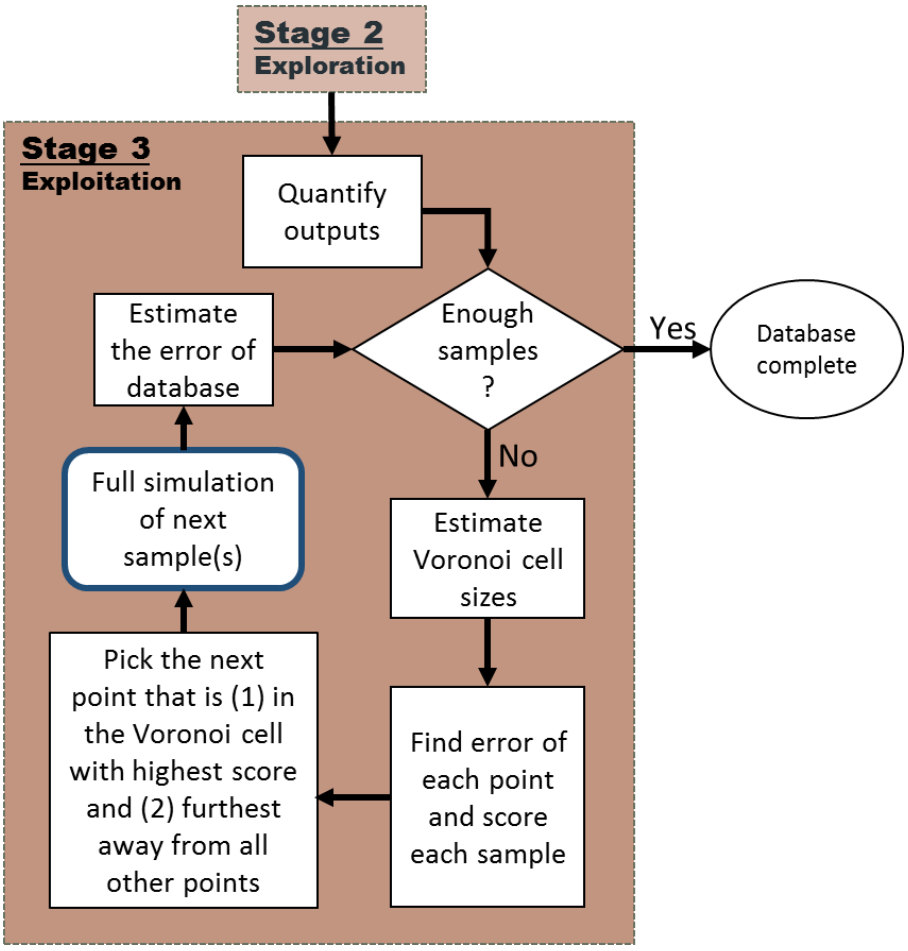


Figure 5 – Exploitation stage workflow.

The workflow of this stage is summarized in Figure 5. At beginning the outputs are reduced and quantified as described in Section 3.3.2. The output quantification is repeated since now full simulation data is available and the output surface is expected to change. However, unlike the screening stage, a different fluence can be selected. The "fluence when the base case has a k-infinity of unity" (F_B) is used (where $k = \frac{PR(F_B)}{DR(F_B)}$). It is found by:

$$PR(F_B) = DR(F_B) \qquad (5)$$

In the above equation PR is the neutron production rate, and DR is the neutron destruction rate (as introduced in Section 3.3.2). Sample selection during exploitation begins once the outputs are quantified.

New samples are added to the database based on a sample selection method that adds utilization of output behavior to space-filling criteria. First, the Voronoi cell sizes of each point are calculated [42]. The Voronoi cell, as explained further in the following subsection, is the space surrounding a point **p** so that anywhere within the space is closer to point **p** than any other point in set **P**. Next, the estimated maximum error in the output metric in the vicinity of each point is found by comparing its output value to one obtained by interpolation to that point using other points in the database (see Section 3.4.2). The Voronoi cell and error calculations, described next, will be used to score each point in the database and the next sample will be selected near the point with the highest score.

3.5.1. VORONOI CELL ESTIMATION

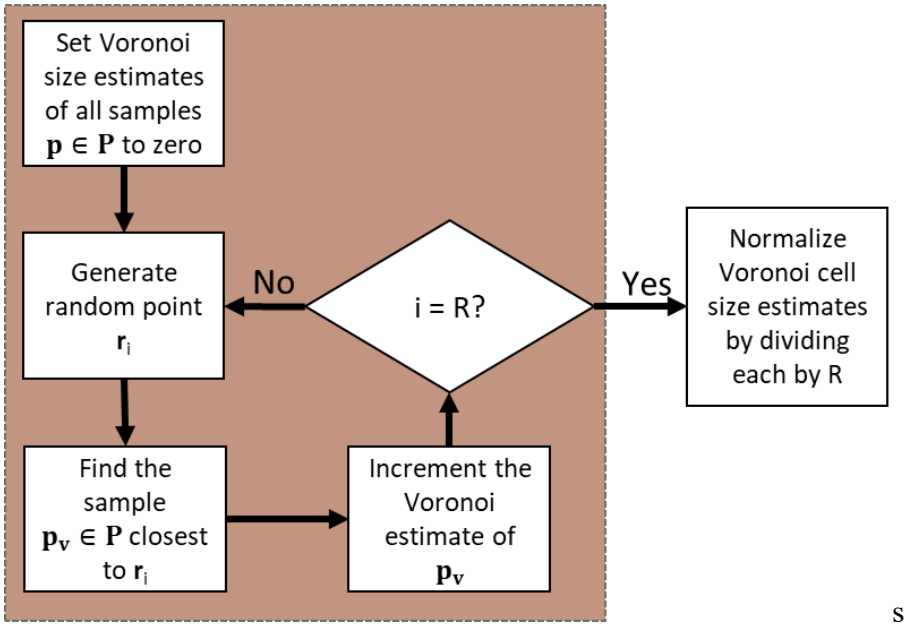
Voronoi cell sizes are used to quantify the space around each sample point in the database. A Voronoi diagram, as shown in Figure 6, is created by generating cells around each point in the database so that anywhere inside a cell is closest to the point within it. The "size" of the cell ("area" in 2D and "volume" in 3D) is found from the space it occupies.

Figure 6 – Voronoi cells of 20 points in 2D [43]

It may seem straightforward to determine Voronoi cells and their sizes in 2D, but the problem quickly becomes computationally expensive as the number of dimensions and points increase [10]. Therefore a Monte Carlo approach is used to estimate the Voronoi cell sizes.

s

Figure 7 summarizes this process. Similar to the method for exploration, random points are generated. The number of random points R is scaled similarly to the method given in Equation (1). For each random point, the sample in the database that lies closest to it is found and the tally of that sample is incremented by one. Once all random points are evaluated the Voronoi cell sizes are found by dividing tallies by R.



s

Figure 7 – Voronoi cell size estimation.

3.5.2. SAMPLE ERROR ESTIMATION

The goal of exploitation is to focus sample selection near areas of interest, specifically regions where database interpolation results in the highest errors. The error of each point is found by first excluding it from the database and using the remainder of samples to calculate by interpolation the output at the input coordinates of the excluded point. This method of calculating error is the leave-1-out method, discussed in Section 2.3. The true error cannot be calculated as it would require numerous repeated runs of the costly full simulation.

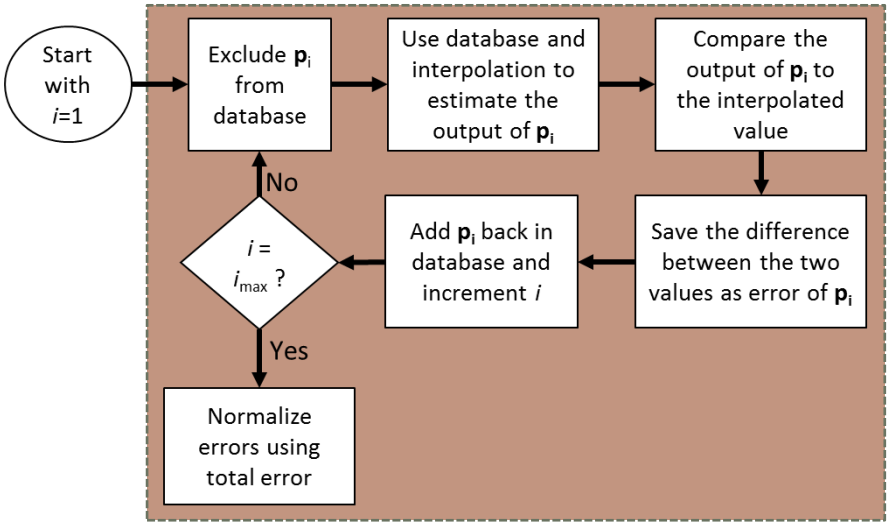


Figure 8 – Sample error estimation workflow.

Figure 8 above explains this process. Each point $\mathbf{p}_i$ is iteratively excluded from the database. The resulting database with an excluded point $\mathbf{p}_i$ is used to interpolate to the input coordinates of $\mathbf{p}_i$. The difference between the output of $\mathbf{p}_i$ (shown as $f(\mathbf{p}_i)$) and the interpolated result is saved as the non-normalized error of $\mathbf{p}_i$ (named E^*). This calculation is given below:

$$E^*(\mathbf{p}_i) = \frac{\left|\hat{f}^{(-i)}(\mathbf{p}_i) - f(\mathbf{p}_i)\right|}{f(\mathbf{p}_i)} \qquad (6)$$

Here, $\hat{f}^{(-i)}$ is the interpolation operator that calculates results using every point in the database except point *i*. This notation is adapted from [10].

Once all errors are found, they are normalized by dividing each value by the total error of all the samples. This normalization assures that all errors sum to unity regardless of number of samples in the database, which is the same scaling for Voronoi cell sizes. This calculation is given below, where $\sum E^*$ is the sum of all non-normalized errors in the database.

$$E(\mathbf{p}_\mathrm{i}) = \frac{E^*(\mathbf{p}_\mathrm{i})}{\sum E^*} \qquad (7)$$

Note that this means the error measure will decrease as the number of samples increase. Therefore, this measure represents the share of total error by each point.

3.5.3. EXPLOITATION METHODOLOGY

This subsection describes the combination of Voronoi cell sizes and sample errors to pick the next sample. As presented in Section 3.5.1, the sample selection process begins with the calculation of Voronoi cell sizes followed by sample error estimation. Next, each sample is scored according to the values of these two properties in the following way. For a sample i in the database:

$$S(\boldsymbol{p}_i) = V(\boldsymbol{p}_i) + cE(\boldsymbol{p}_i) \qquad (8)$$

where $S(\mathbf{p}_i)$ is the rank score, $V(\mathbf{p}_i)$ is the normalized Voronoi cell size, and $E(\mathbf{p}_i)$ is the sample error of point $\mathbf{p}_i$. The constant c is the error weighing factor; if c is very small, the methodology reverts to an exploration approach where only the density of points in the vicinity is considered. If c is large, the method can in theory populate points in arbitrarily close distance, although in practice the magnitude of the interpolation error term is not independent of the Voronoi cell size. Value for c are investigated in Section 4.5.2.

Once each sample is scored, the next sample is selected so that it is in the Voronoi cell of the sample with the highest score, but placed as far away from that sample as possible. The existing sample in the database with the highest rank score is referred as the base sample (p_{bi}, where i represents the number of samples in the database when the base sample is selected) for the next sample selection. Finding the point furthest away from the base sample within its Voronoi cell is computationally very expensive as it necessitates the determination of cell boundaries. The computational work done in the Voronoi cell size estimation is utilized to aid this problem. In short, during the Voronoi cell size estimation the furthest point in each cell is saved in case that cell scores the highest. This process is described in the listing below.

for all $\mathbf{p}_i \in \mathbf{P}$:

Use database $\mathbf{P}$-$\mathbf{p}_i$ to find interpolated estimate of $\mathbf{p}_i$

$E(\mathbf{p}_i)$ = the difference between estimate and output of $\mathbf{p}_i$

for all $\mathbf{p}_{rand} \in \mathbf{P}_{rand}$:

for all $\mathbf{p}_i \in \mathbf{P}$:

if $\|\mathbf{p}_{rand} - \mathbf{p}_i\| < d_{min}$:

$d_{min} = \| \mathbf{p}_{rand} - \mathbf{p}_i \|$

$\mathbf{p}_{closest} = \mathbf{p}_i$

if $d_{min} > d_{maxmin}(\mathbf{p}_i)$:

$d_{maxmin}(\mathbf{p}_i) = d_{min}$

$\mathbf{p}_{maxmin}(\mathbf{p}_i) = \mathbf{p}_{rand}$

$V(\mathbf{p}_{closest})$ += 1 / count($\mathbf{P}_{rand}$)

calculate score $S(\mathbf{p})$ for all $\mathbf{p} \in \mathbf{P}$ using $E(\mathbf{p})$ and $V(\mathbf{p})$

find $\mathbf{p}_{highest}$ so that max(S) = $S(\mathbf{p}_{highest})$

The first *for* loop performs the sample error estimation. The second one iterates through all random points, tallying which sample point the random point lies closest. During this iteration if the random point is the furthest point within the cell, it is saved ($\mathbf{p}_{maxmin}(\mathbf{p}_i)$). Once all random points are considered, the next sample is therefore $\mathbf{p}_{maxmin}(\mathbf{p}_{highest})$. This process is repeated until the end of exploitation.

Figure 9 below demonstrates differing patterns of sample selection for exploration versus exploitation. The background color gradient represents the underlying function that is being sampled. The function, given as F1 in Appendix 6.3, peaks at a value of one and

the lowest point is zero. The points are numbered in order of generation. While the left database was generated only with exploration, the database on the right switched to exploitation after the initial 30 points. While the database on the left does not take into account the output function values and attempts to fill the space as evenly as possible, the database on the right has populated samples close to the area where the slope of the output is changing quickly. Note that the specific locations of these samples will change if the process is repeated due to the stochastic nature of sample selection.

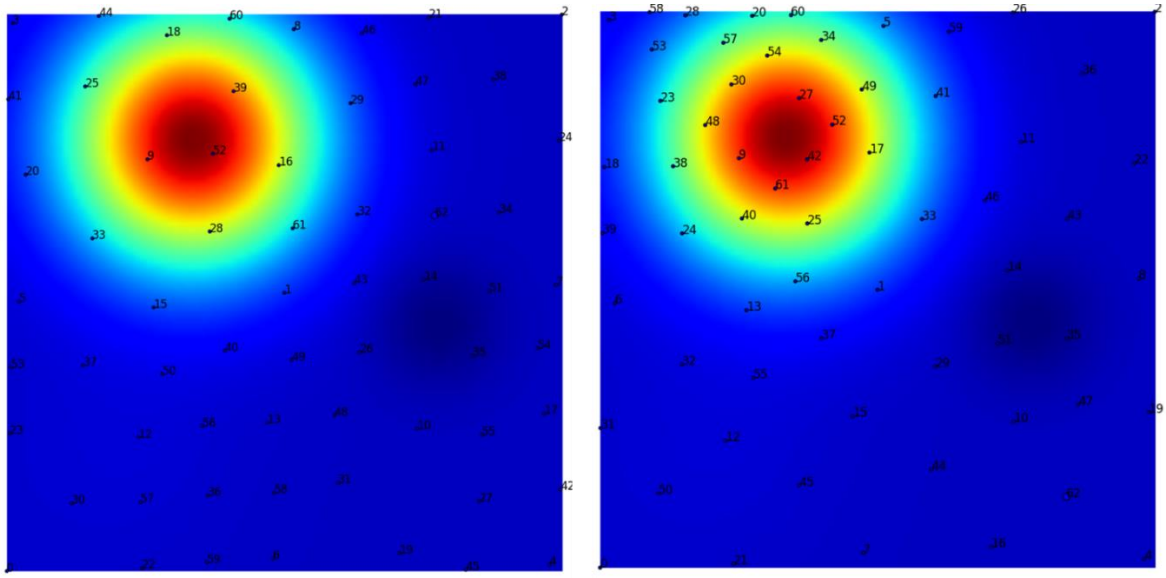


Figure 9 – Database sample selection with exploration-only (left) and exploitation of final 30 points (right) where each sampled in numbered in order of inclusion.

3.6.Error Estimation

This subsection presents two methods for error calculation. The error associated with a database is defined as the average of differences between the outputs found by interpolating the data in the database and the full simulation outputs for the input coordinates.

The error found by utilizing many outputs is named the *real error*. The calculation is done iteratively using R random points. The average real error is:

$$\mathrm{Error}_{\mathrm{real}} = \frac{1}{R}\sum_{i=1}^{R}\frac{\left|\hat{f}(\mathbf{r}_\mathrm{i}) - f(\mathbf{r}_\mathrm{i})\right|}{f(\mathbf{r}_\mathrm{i})} \qquad (9)$$

where $f(\mathbf{r}_\mathrm{i})$ is the output of $\mathbf{r}_\mathrm{i}$ using the full simulation and $\hat{f}(\mathbf{r}_\mathrm{i})$ is the interpolated output of $\mathbf{r}_\mathrm{i}$ using the database. The maximum real error is the highest value in the summation above. It is not feasible to find the real error during practical use cases where a costly full simulation is used. However, this error measure is useful to evaluate the workflow and assess the convergence rate of the error estimation method described in the following paragraphs. In test cases where a placeholder full simulation with quick execution is used, data can be generated solely for the purpose of error calculation and the real error can be calculated.

A sufficiently high value of R depends on many factors including the complexity of the underlying system behavior and the number of dimensions. It is found that a value of R=4000N yields an error accurate to two significant digits for 3-dimensional systems (where N is the number of samples in the database). This formulation of R is based on [41].

In practical use cases resources cannot be sacrificed solely for error calculation. Instead, the available data must be utilized to estimate the error. This is done using the leave-1-out strategy described in Section 2. Two error measures are significant: the maximum estimated error and the average estimated error; both derived from the errors used during exploitation (Section 3.4).

The maximum error is determined by:

$$\text{Error}_{\text{max}} = \max\big(E(\mathbf{p}_\text{i})\big) \text{ for } \mathbf{p}_\text{i} \in \mathbf{P} \tag{10}$$

where average error is determined by:

$$\text{Error}_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N} E(\mathbf{p}_\text{i}) \quad \text{for } \mathbf{p}_\text{i} \in \mathbf{P} \tag{11}$$

3.7. Voronoi Cell Adjustment

This subsection describes a potential improvement to the next sample selection step during exploitation. So far, as described in Section 3.5, the input values for the next sample are determined by using the inputs of the Monte Carlo point furthest away from the selected base sample ($\mathbf{p}_{bN}$) while still being within the Voronoi cell of this base sample. The *N*th base sample is the sample with the highest rank score *S* in the database when the database has *N* samples (as described in Section 3.5.3).

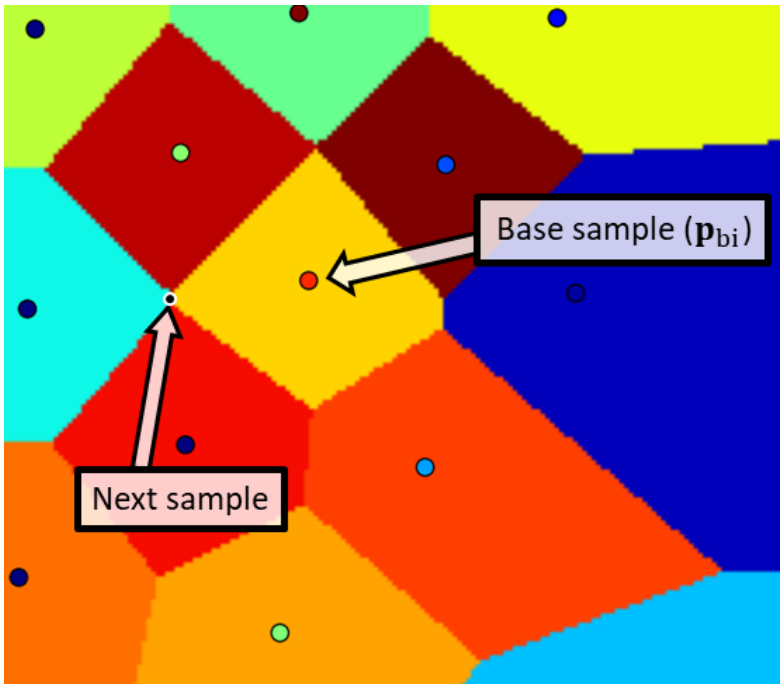


Figure 10 demonstrates the sample selection process presented in Section 3.5. The Voronoi cells in this picture are arbitrarily colored, while the colors of the sample points correspond to the estimated error of the points (red corresponds to high error and blue to low). Given that the marked Base Sample is found, the algorithm will place the next sample to the location marked 'Next sample' since this is still within the Voronoi cell of the base sample while being as far away as possible from the point itself.

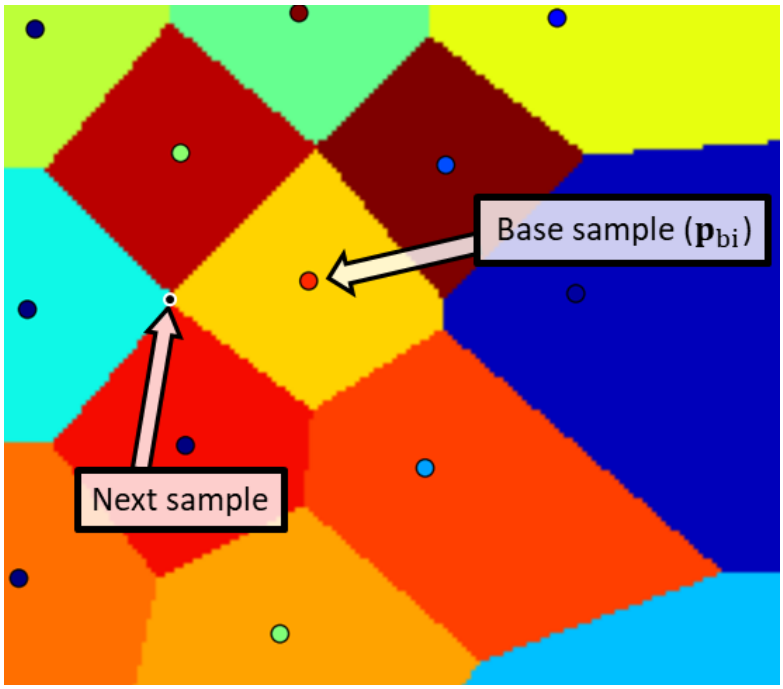


Figure 10 – Voronoi cells (arbitrary coloring) of a database where sample colors correspond to estimated errors so that red is high error.

The methodology described in this section adjusts the Voronoi cells based on the estimated errors to guide the sample selection towards areas of higher error. This means that once the base sample is selected the method attempts to shrink the Voronoi cells associated with high error samples and expand the cells associated with low error samples. This transformation effectively moves the location of the next sample, $\mathbf{p}_{i+1}$, closer to the neighbor point with the highest estimated error, if it is higher than the error associated with $\mathbf{p}_{bN}$.

In order to achieve this Voronoi cell adjustment, first the estimated errors (E^*) of each sample are calculated (as described in Section 3.5.2). These errors are normalized using the estimated error of the base sample ($E^*(\mathbf{p}_{bN})$) as given in the following expression:

$$E_0(\mathbf{p}_i) = \frac{E^*(\mathbf{p}_i)}{E^*(\mathbf{p}_{bN})} - 1 \quad (12)$$

where $\mathbf{p}_\mathrm{i}$ is a sample in the database. This operation assigns positive values for all samples with errors higher than the error of the base sample, and negative values for those with lower errors.

Next, user specified 'distance factor' (d_f) is used. The distance factor is a value between zero and unity where zero corresponds to no Voronoi cell adjustment and unity corresponds to the doubling of the most adjusted distance, as will be shown next. The values of $E_0(\mathbf{p}_\mathrm{i})$ are checked against this factor so that none are higher than d_f:

$$\begin{aligned}&\text{If} \quad \max(|E_0(\mathbf{p}_\mathrm{i})|) > d_f\text{:}\\&\text{Then:}\\&E_0^*(\mathbf{p}_\mathrm{i}) = \frac{d_f}{\max(|E_{norm}(\mathbf{p}_\mathrm{i})|)} E_0(\mathbf{p}_\mathrm{i})\\&E_0(\mathbf{p}_\mathrm{i}) = E_0^*(\mathbf{p}_\mathrm{i})\end{aligned} \qquad (13)$$

where $E_0^{prev}(\mathbf{p}_\mathrm{i})$ is equal to $E_0(\mathbf{p}_\mathrm{i})$ before the 'If' statement. Finally, the Voronoi adjustment factors $\theta(\mathbf{p}_\mathrm{i})$ are found using the following equation:

$$\theta(\mathbf{p}_\mathrm{i}) = E_0(\mathbf{p}_\mathrm{i}) \cdot d_f + 1 \quad (14)$$

These factors are used to divide the distance between the sample ($\mathbf{p}_\mathrm{i}$) and the random Monte Carlo point ($\mathbf{p}_\mathrm{r}$) during the Voronoi cell calculation (as described in Section 3.5.1). This adjustment changes the 'nearest sample' for some Monte Carlo points during this step, as a result adjusting the boundaries of each Voronoi cell.

Figure 11 shows the Voronoi cells with a d_f value of 0.8. The location of the base sample and the next sample before and after the Voronoi cell adjustment are also shown. It can be seen that the cells of samples with low errors are bigger while the cells of samples with high errors are smaller. Thus, the new next sample is selected closer to the samples with higher relative error (above and upper-left of the base sample). In this case, a different vertex becomes farthest from $\mathbf{p}_{\mathrm{b}N}$, so the direction vector pointing from $\mathbf{p}_{\mathrm{b}N}$ to $\mathbf{p}_{\mathrm{i}+1}$undergoes a substantial change. Numerical evaluation shows that this change takes place at d_f = 0.55 for this specific example. If d_f is smaller than this value, the modified

$\mathbf{p}_{i+1}$ remains in the vicinity of the $\mathbf{p}_{i+1}$ that would be chosen under the original method (i.e., with $d_f = 0$). Note these are stochastic results and may change from one trial to the other.

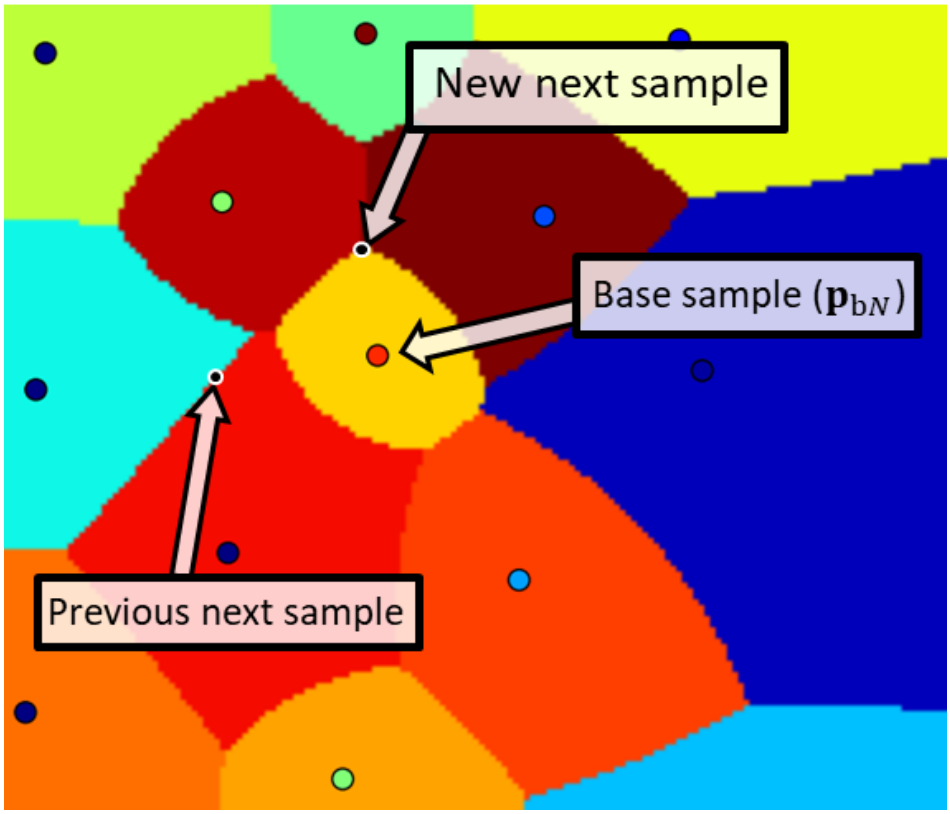


Figure 11 – Adjusted Voronoi cells using d_f=0.8 (arbitrary Voronoi cell colors).

The final step of Voronoi cell adjusted sample selection is to shift the next sample in order to preserve good space filling properties. This is necessary since the edges of the adjusted Voronoi cells do not guarantee to be far from other samples. In fact, as d_f increases the space filling quality of the unshifted next sample will decrease.

The following method is used to shift the next sample in arbitrary dimensions without excess computational cost. First, the closest point to the next sample is found using unadjusted Euclidean distances (as explained in Section 3.5.1). Second, the next sample is shifted away from this sample iteratively using a small increment. The new closest point is

found at each iteration until the next sample is closer to a new point (or at the edge of the domain).

The shifting is demonstrated in Figure 12. The next sample is moved along the grey line pointing away from the base sample. At each iteration the closest sample to the next sample is determined. The next sample is shifted until there is a new closest sample.

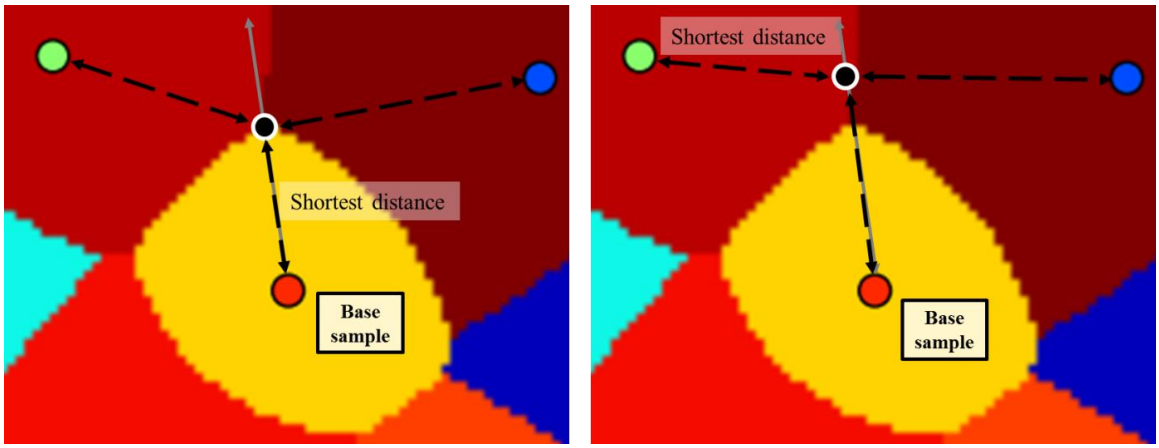


Figure 12 – Adjusted Voronoi cell next sample shifting.

The location of the shifted next sample relative to the unadjusted Voronoi cells are given in Figure 13. It is important to note that the shifted next sample being on the boundary of the Voronoi cells is not a coincidence. In fact, this method guarantees that the shifted next sample will lie on the boundary of the Voronoi cell without having to go through the computationally expensive calculation of determining these cell surfaces.

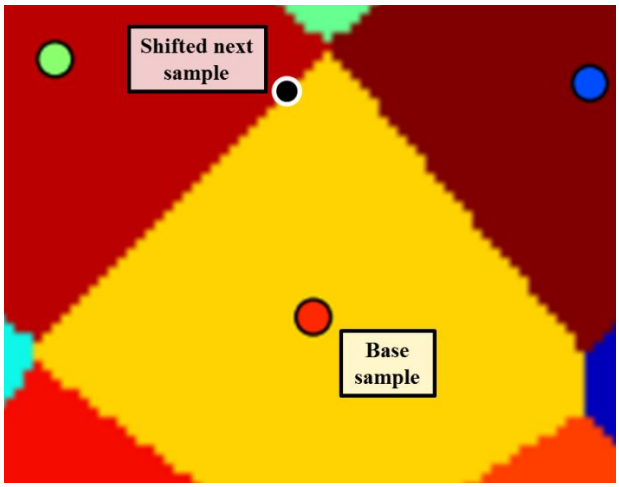


Figure 13 – Shifted next sample shown over unadjusted Voronoi cells.

4. RESULTS

4.1.Comparison of Error Measures

This case is chosen to demonstrate the available error measures. The black box model depicted in Figure 9 was used for this 2-dimensional case (given as F1 in Appendix 6.3). The results are the average errors of 40 databases with 96 samples each, where the final 40 samples were generated using exploitation. The uncertainties of the errors were calculated from the standard deviation of errors from the repeated databases. The values for d_f, $k_{explore}$, and $k_{exploit}$ that were used in this section are same to the values given in Section 4.6

There are two measures of error available during a regular database generation: the average estimated error and the maximum estimated error. They are found by comparing the results of the samples in the database to the interpolated estimates, as described in Section 3.5. In addition to these errors, the real error and the maximum real error can be found if many samples can be generated. This case presents all four types of errors. The errors are calculated and recorded after each new sample and plotted in Figure 14.

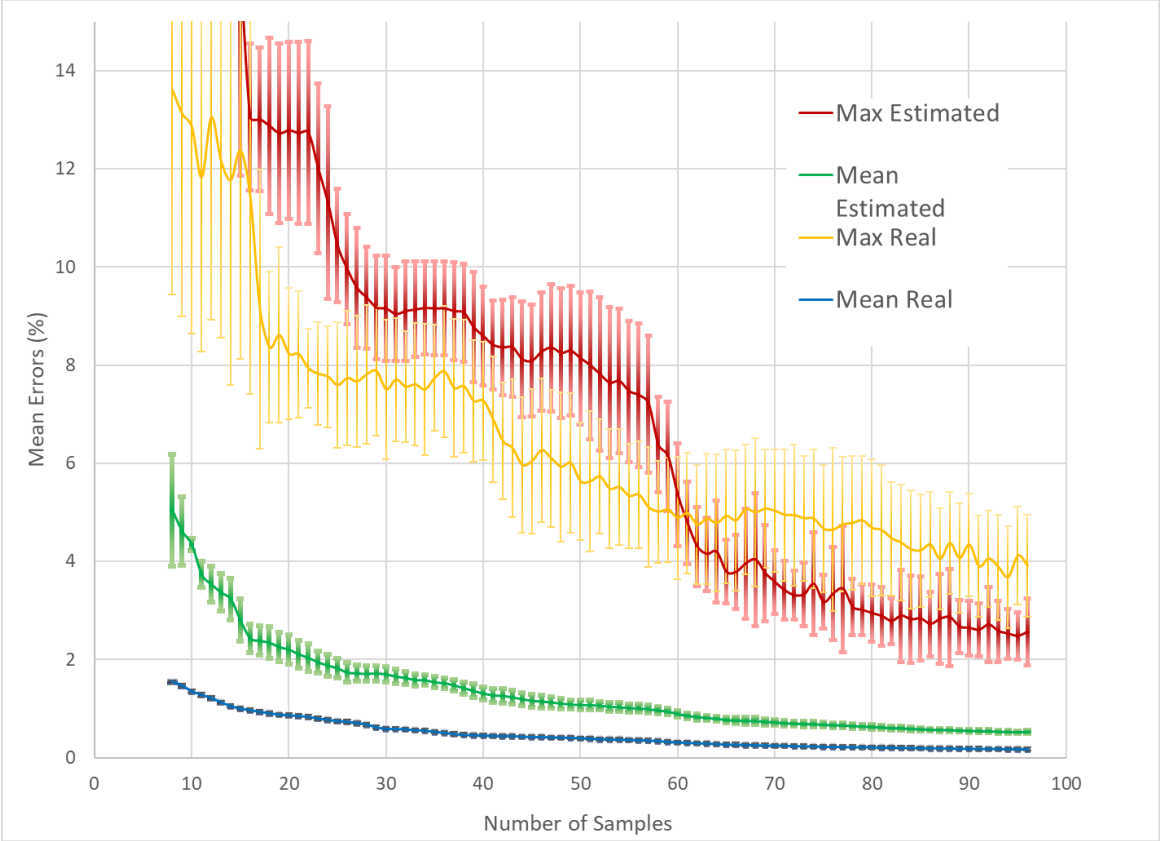


Figure 14 – Comparison of error measures for an example database.

Looking at this figure, it can be seen that the real error is consistently lower than the estimated error. This result is expected since the estimated error is found by iteratively removing samples from the database. Since every removed sample has good space-filling properties, the error of estimating the full simulation output at those points should be the higher than the average error.

The maximum errors are higher than both these measures. The switch to exploitation can be observed from the maximum estimated error. Since exploitation prioritizes sampling near these points, it is expected to see this behavior.

4.2. Exploitation Method Test

One of the many factors affecting the reduction in error over time is when the transition to exploitation should occur. This subsection presents results found by repeatedly generating databases for different exploitation start points. Each database was generated using 2 varied inputs with a total of 88 samples. A testing function for the black box model was used. The testing function was used to have full control over the black box model and to be able to calculate the real error. Six types of databases were generated with 30 databases for each type, totaling 180 databases. The first type is named Random: all samples were selected randomly with no regard to the location of other points. The second type is Explore Only, where there was no exploitation. This approach should improve on random since exploration involves taking care to distribute sampled points evenly in the problem domain. The remaining four database types switch to exploitation at different times. Exploit 20 databases use exploitation on the final 20 samples whereas Exploit 80 databases use exploitation on the final 80.

The real error is plotted as a function of number of points in the database for the average of each type. Results can be seen in Figure 15. The blue line is databases generated using random samples. It can be clearly seen that randomly selecting samples is not desirable.

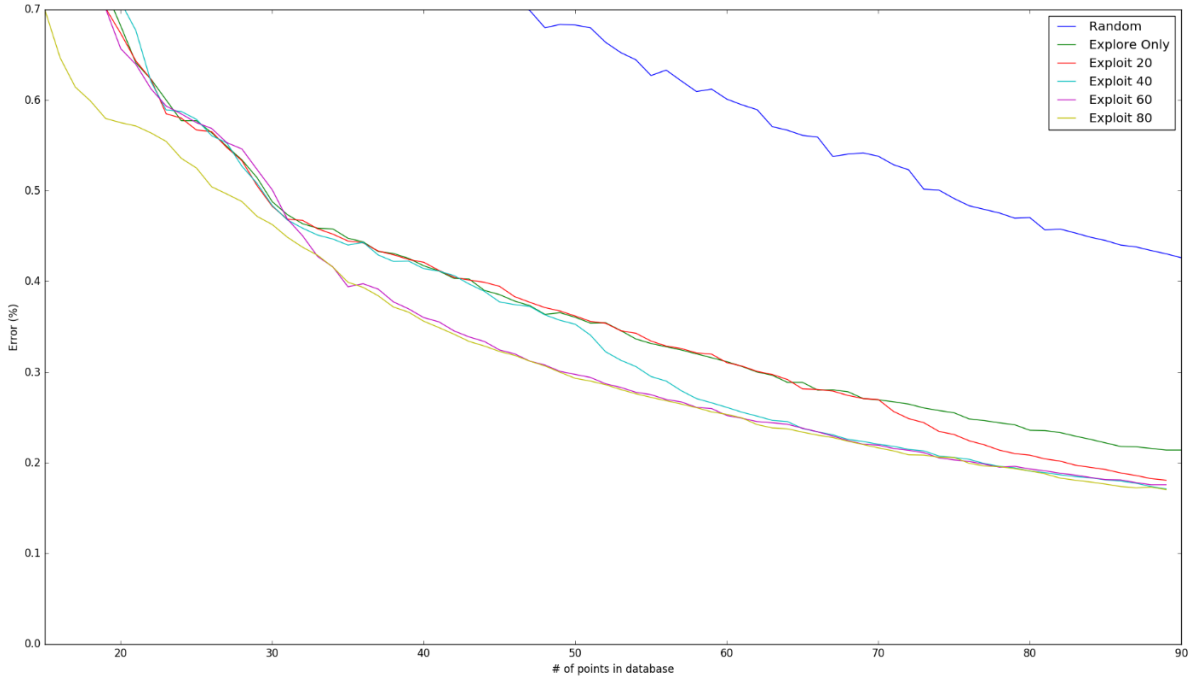


Figure 15 – Comparison of average database real errors generated using different exploitation start times.

The green line in Figure 15 is the behavior of the error if there is no exploitation. It can be observed that the error predictably decreases with each new sample. The remaining lines represent databases created with exploitation. The point where exploitation begins is clear for each case, where each sampled point leads to a greater reduction in error. However, since all databases with exploitation end with very similar errors, it can be concluded that there is a diminishing benefit of exploitation and the reduction of error begins to match the exploration case. While the first samples once exploitation begins yield significant improvements on the error, once these interesting areas are sampled the benefit of additional samples seem to move towards that of exploration only.

Figure 16 compares the four error measures available for the Explore Only and Exploit 40 cases. The single solid lines are the error measures for the Explore Only case. The maximum errors are on a different axis. The figure shows the maximum estimated error is higher than the maximum real error until exploitation starts. Since exploitation focuses on these high error points, it forces the error to go down.

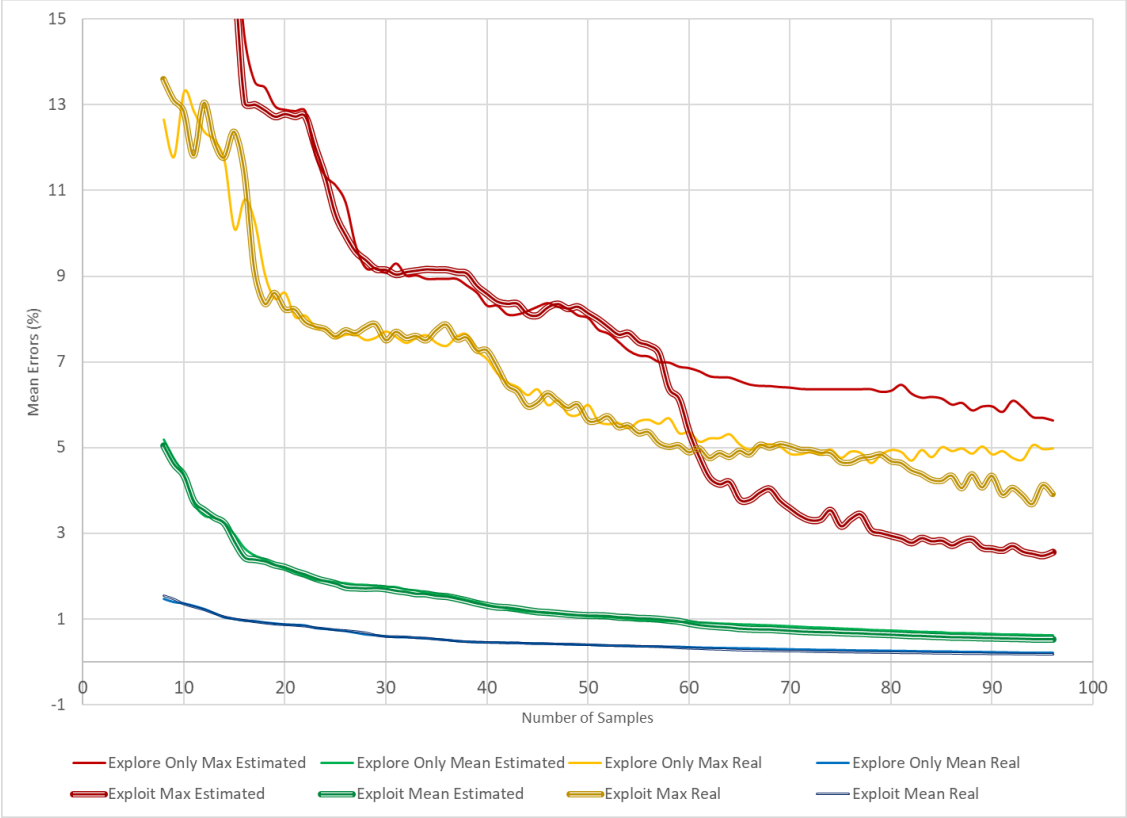


Figure 16 – Comparison of the four error measures for two cases.

It is important to note that given enough samples, all errors will reach zero regardless of the selection method. The goal is to reduce the error quickly, and exploitation achieves this goal better than exploration alone for this black box function. If instead the black box function is very simple, the overhead of exploitation calculations will not be worth sacrificing adding more samples to the final database.

4.3.Software Runtime and Scaling

This section analyzes the software runtime for different dimensions and samples. Given that there is a limited time to generate a database, the total time to complete one should be predictable in order to quantify the underlying limitations of the methodology. This total time to complete a database can be expressed as a summation of the time taken to complete each step:

$$T_{total} = T_{screen} + T_{explore} + T_{exploit} \tag{15}$$

where T_{screen} is the total time for screening, $T_{explore}$ total time for exploration, and $T_{exploit}$ total time for exploitation. The limitation is that $T_{total} \leq T_{max}$, where T_{max} is the maximum time allowed for database generation. The objective of the methodology is therefore to achieve:

$$\min(\text{Error}_{\max}) \text{ while } T_{total} \leq T_{max} \tag{16}$$

Given that it takes t_{screen} for a typical screening run, and t_{full} for a full simulation run, each of these times can be written as a function of constants and parameters described above:

$$T_{screen}\left(t_{screen}, k_{explore}, \epsilon_p\right) = N_{screen} \cdot t_{screen} \sum_{n=1}^{N_{screen}} \left(C_{screen}\, D\, n\right) \tag{17}$$

$$T_{explore}\left(t_{full}, k_{explore}, \epsilon_p\right) = N_{explore} \cdot t_{full} \sum_{n=1}^{N_{explore}} \left(C_{explore}\, D\, n^2\right) \tag{18}$$

$$T_{exploit}\left(t_{full}, k_{exploit}\right) = N_{exploit} \cdot t_{full} \sum_{n=1}^{N_{exploit}} \left(C_{exploit}\, D\, n^{1.5}\right) \tag{19}$$

where D is the number of dimensions, the C_x's are the computation overhead constants, and N_x's are the number of samples per step. The C_x's may be dependent on n

and D based on the software implementation. The powers of n in these equations are estimated based on analyzing the methodology. In addition, these equations are estimates of the methodology runtime and do not include the additional computational cost of increased dimensions and samples in the software architecture as well as the cost of the sample interpolation method used during the exploitation step (Equation (19)). It can be observed that the methodology is linearly dependent on the number of dimensions in each step, whereas the dependence of number of samples varies.

Screening is a preliminary step that can have various implementations of data analysis methods (Section 3.3). Most of these methods (dimensionality reduction, such as PCA) are designed for data that is typically much larger than the scope of data in this work and are expected to scale well beyond the limitations of the following steps of exploration and exploitation (for example PCA is regularly applied to hundreds of dimensions and samples). Therefore, the scaling of screening is not covered in this section.

Below, the time it takes to find a sample for the exploration and exploitation steps are given for various numbers of dimensions. The timing results were repeated 12 times and averaged for all plots in this section. An average uncertainty of 3.5% was measured for all timing measurements.

Figure 17 shows the times for the exploration step with a Monte Carlo multiplier ($k_{explore}$) set to 400. This value is high enough to reduce the stochasticity of the method to generate samples in the same vicinity (relative to other samples) when the database is repeated for that sample. In other words, using this $k_{explore}$ value, when a sample is generated again it is put in the same approximate location in the database relative to other samples, i.e. same neighbors. $C_{explore}$ is linearly correlated to this value.

The time it takes to run the simulation to generate outputs is not considered in this section. The output generation time depends on factors such as the simulation being used as well as the input parameters. The NUDGE software runtime scaling is not affected by these factors.

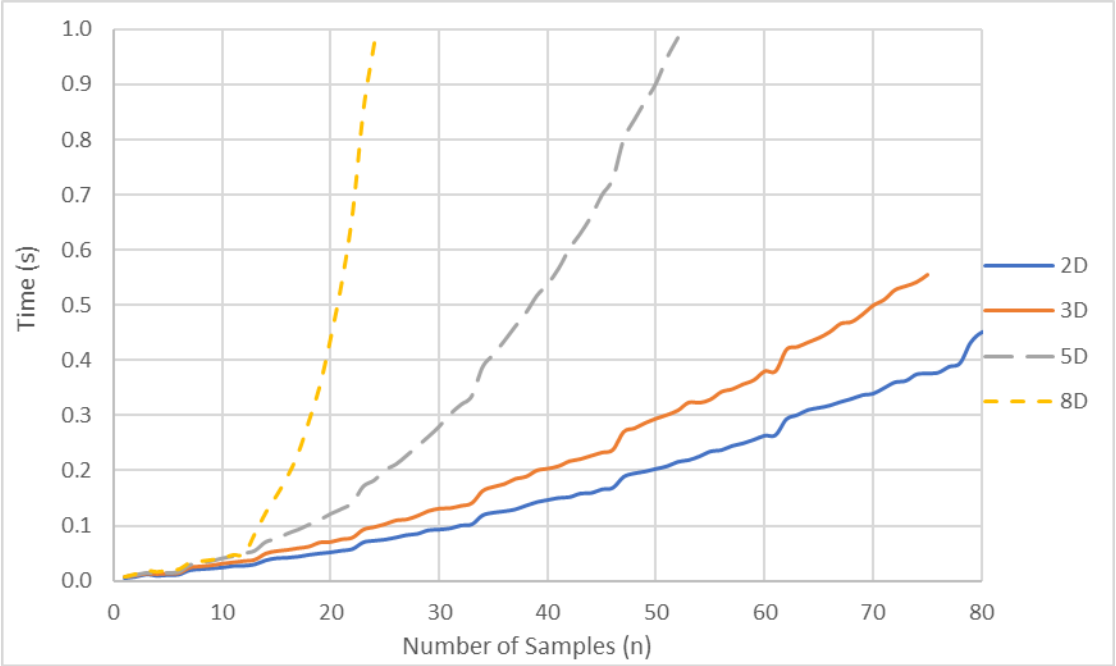


Figure 17 – Exploration time per sample for different dimensions.

Figure 18 shows the time for exploitation steps with a $k_{exploit}$ value of 400. This value is selected to match $k_{explore}$. Similar to exploration, $C_{exploit}$ is a function of $k_{exploit}$. Note that the minimum number of necessary exploration steps increases as dimensions are increased, hence the starting point of each curve in this figure moves forward with increasing dimensions. It can be observed from these two plots that finding the next sample during exploitation is approximately an order of magnitude longer than exploration. As noted before, these times depend on the software implementation as well the interpolation function being used.

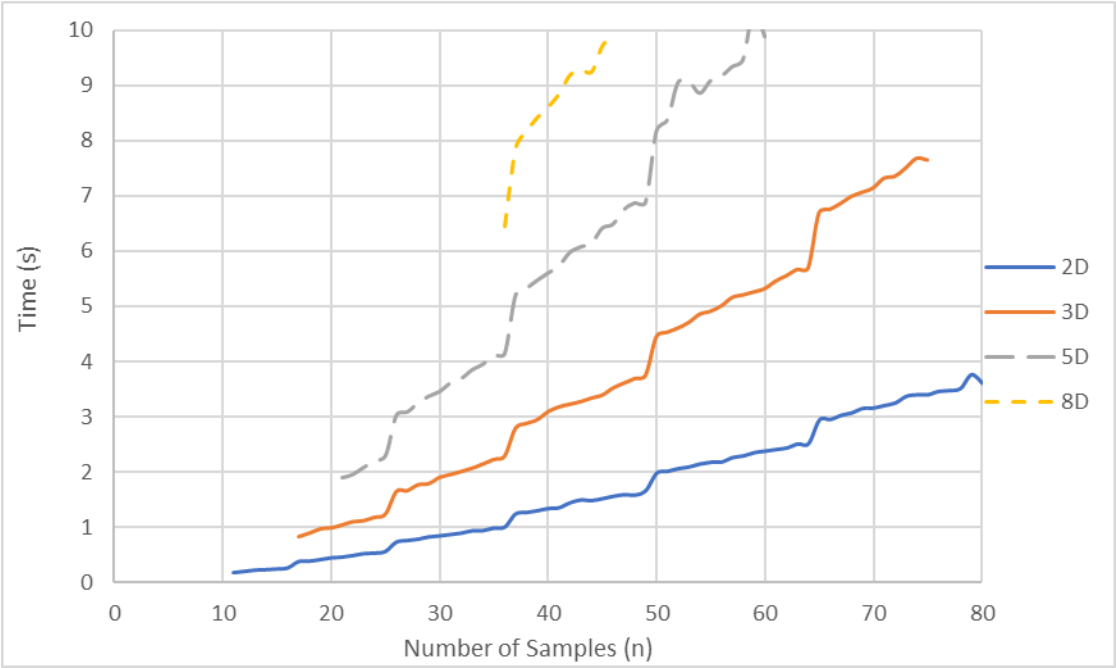


Figure 18 – Exploitation time per sample for different dimensions.

Equation (18) shows the dependence of sample generation time in the exploration step (Section 3.4). In order to demonstrate the accuracy of this equation, databases with varying dimensions were created while measuring the time it takes, for each sample, to find the inputs of the next sample. For each sample and dimension, the estimated computational effort was found using Equation (18) ($C_{explore}\ D\ n^2$). Figure 19 below shows the sample generation times as a function of estimated number of CPU operations. Therefore the number of operations based on n and d are found for each sample using equation (18), followed by plotting the time for a given sample as a function the CPU operations.

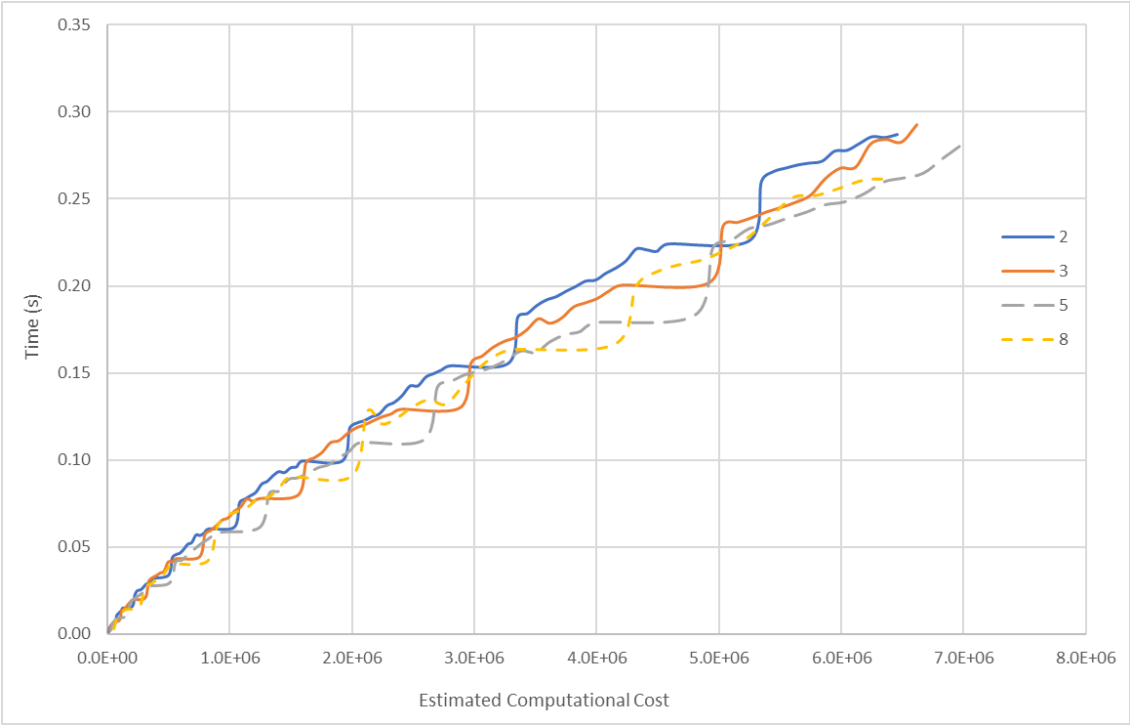


Figure 19 – Exploration step sample generation time vs. estimated computational cost for different number of dimensions.

While there were 100 samples for 2 dimensions, the 8-dimensional curve in this figure goes up to 50 samples. It can be seen from this figure the dependence is well characterized and can be concluded that the estimates for the scaling of the exploration step are accurate.

Next, the exploitation step was measured (Section 3.5). This step is comprised of two parts: error estimation (Section 3.5.2) and Voronoi cell estimation (Section 3.5.1). The scaling of error estimation depends on the chosen method of interpolation, where the interpolation is repeated n times each time a new sample is to be found. Conversely, the Voronoi cell estimation time, as shown in Equation (19), depends on $n^{1.5}$. The following plot repeats the methodology of the previous figure (Figure 19) for execution times of the Voronoi estimation step, again as a function of estimated computational cost.

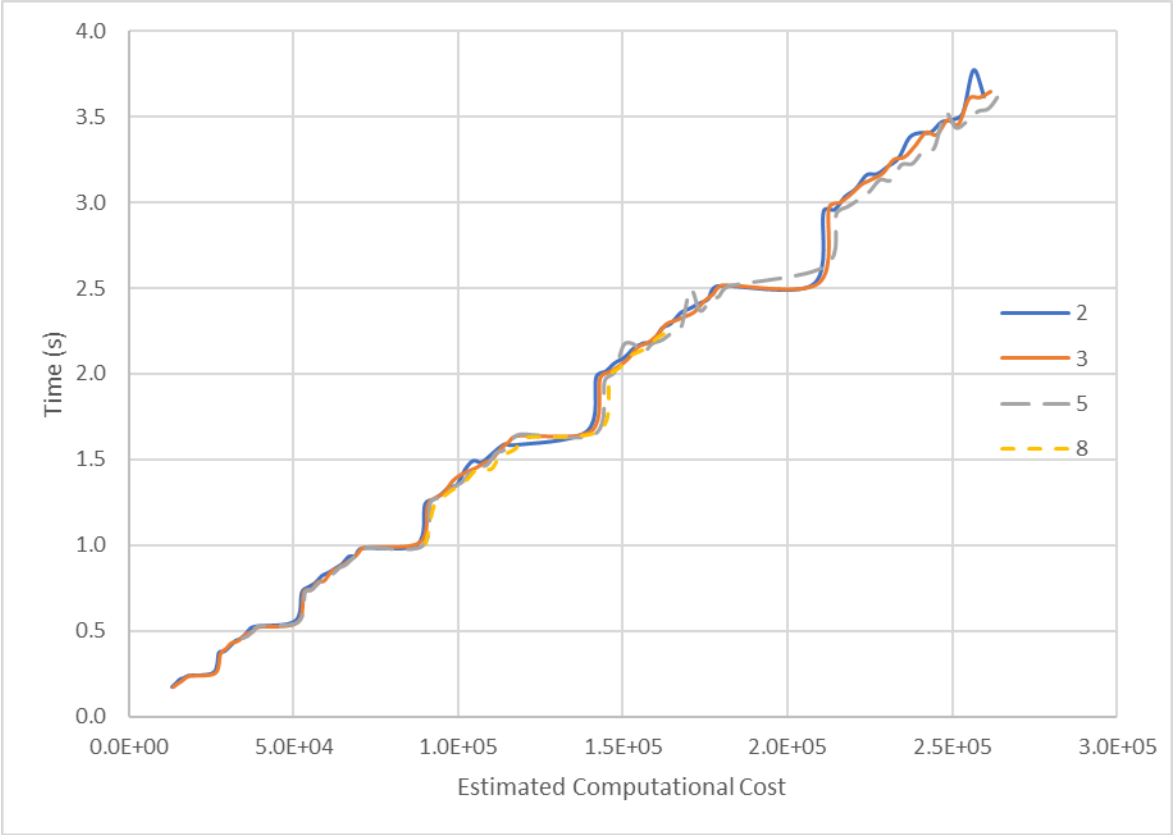


Figure 20 – Exploitation step Voronoi cell estimation time vs. estimated computational cost for different number of dimensions.

Here in Figure 20 the 2-dimensional data goes up to 90 samples while the 8-dimensional data goes up to 60. Databases for each dimension were repeated 12 times and averaged. The "dips" in this data are a result of the method used to select number of Monte Carlo points, where the floor of the square root of number of samples ($\text{floor}(\sqrt{n})$) is used as a multiplication factor to the total number of Monte Carlo points. Each dip corresponds to a number of samples that has an integer square root. Looking at this plot it can be concluded that the dependence of the Voronoi cell estimation times are well characterized in Equation (19).

Finally, the scaling of the Voronoi cell adjustment method is measured (Section 3.7). In short, this method repeats the Voronoi cell estimation and it is expected to take

approximately twice as longer as a single Voronoi cell adjustment step. Results for the timing measurements of Voronoi cell adjustment is given below in Figure 21. The number of samples for each dimension used to create this figure is the same as Figure 20. The same behavior as Figure 20 can be seen, except the times are about twice as long, as expected.

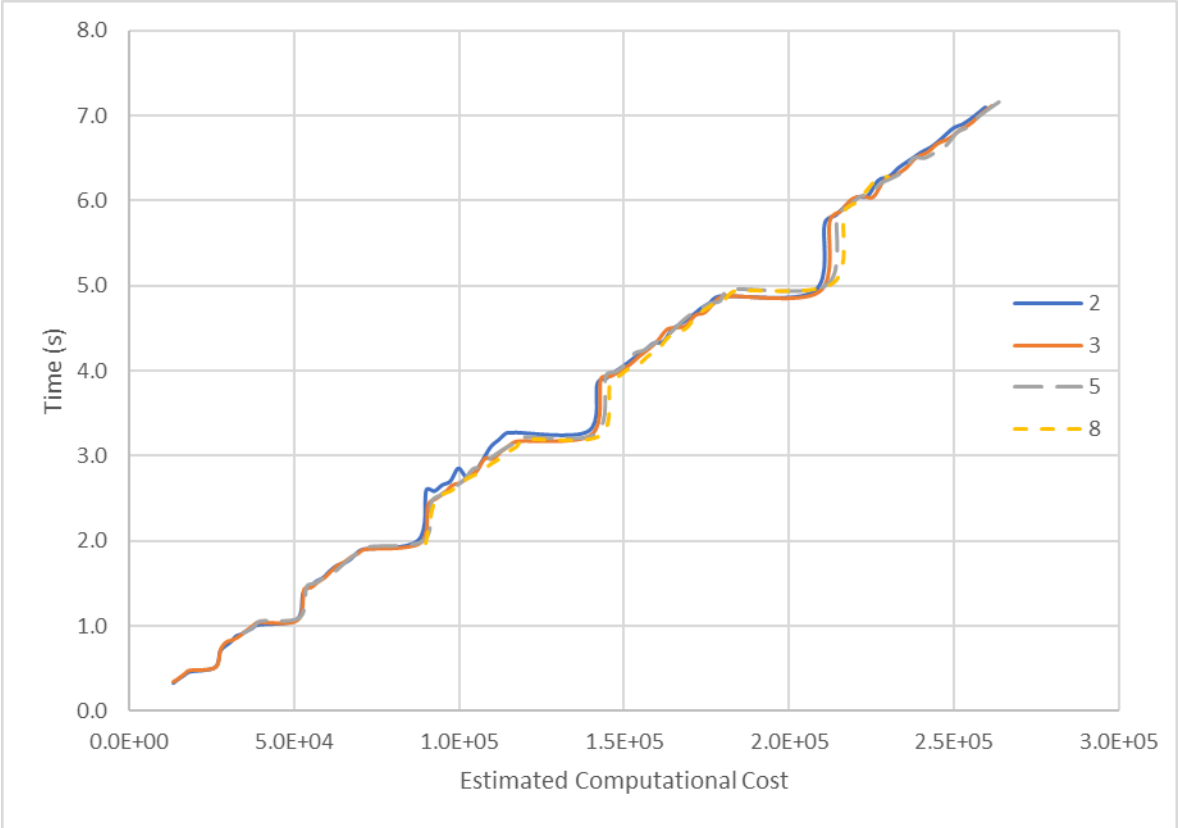


Figure 21 – Exploitation step adjusted Voronoi cell estimation step time vs. estimated computational cost for different number of dimensions.

It should be emphasized that the times reported here are not the total time that the user has to wait for a sample to be generated. The real time will depend on various other factors such as the time to read/write files and the implementation of the software. However, once several samples are generated and the overhead values (the C_x's) can be determined, the given equations can be used to estimate the time for later samples.

Therefore (ignoring the scaling of interpolation method and implementation used in the software) it can be concluded that doubling the number of dimensions will double

the next sample calculation time for all steps; while doubling the number of samples will increase the next sample calculation time approximately by a factor 4 (2^2) for exploration and by a factor of 2.83 ($2^{1.5}$) for exploitation.

4.4.XSgen Output Behavior and Placeholder Function Generation

A placeholder function for the XSgen software was generated. The placeholder is built to quickly generate output values that behave similarly to outputs from XSgen. The placeholder function was used to test database generation strategies discussed later. This section will first demonstrate the output behavior of XSgen, followed by describing the methodology used to create the placeholder function.

First, the output sensitivity of eight XSgen inputs was tested. The tested input variables were: Cell Height, Cell Pitch, Clad Density, Clad Radius, Coolant Density, Flux, Fuel Density, Fuel Radius, and Void Radius. A base case was defined as a typical LWR (this input file is given in Appendix 6.1). For each input variable listed above, a sample was generated with a 10% increase of that variable. These steps were repeated with a 10% decrease. Therefore 1+8+8=17 samples were run (1 base case and 8 for increase/decrease of each input variable). The outputs were quantified as described in Section 3.3.2.

The behavior of the first principal component (PC1) with a 10% change in each input is shown in Figure 22. PC1 accounts for 80.9% of the output response. It can be observed that a 10% change from the base case in some variables, such as cell height and clad density, has barely noticeable effects on the outputs (in this case the output is only PC1). In the remaining input variables it's observed that the magnitude and direction of the change varies. Similar but more pronounced behavior can be observed for principal component 2 (PC2) results, as shown in Figure 23. PC2, which accounts for 16.7% of the output response, has a response similar to PC1.

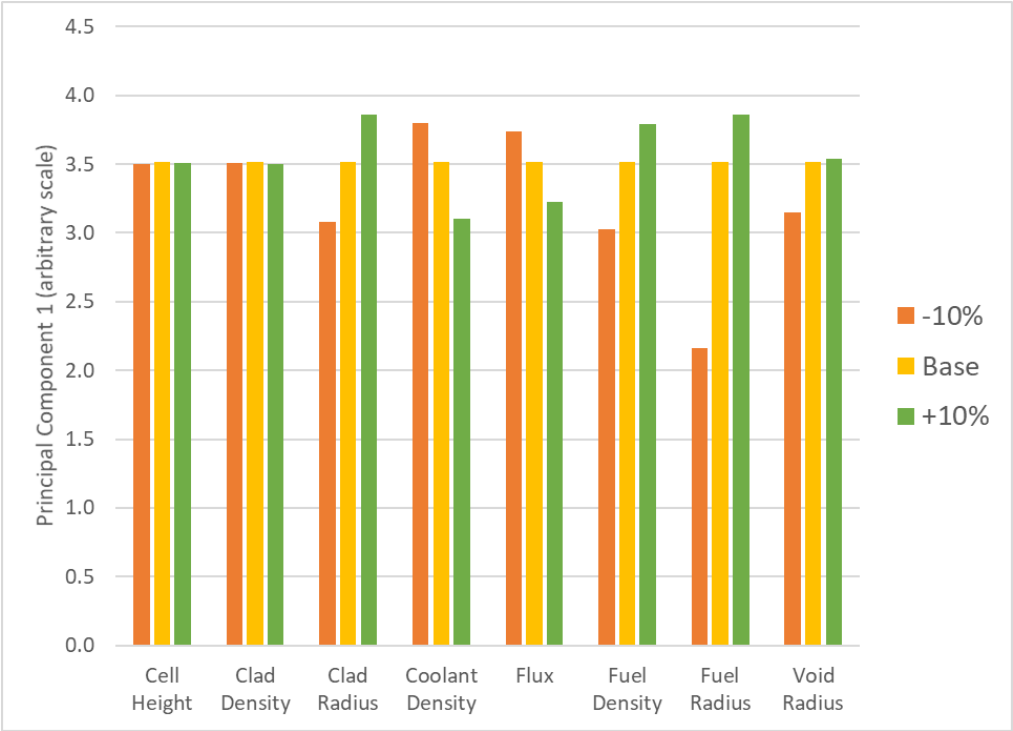


Figure 22 – XSgen Output Sensitivities on Principal Component 1.

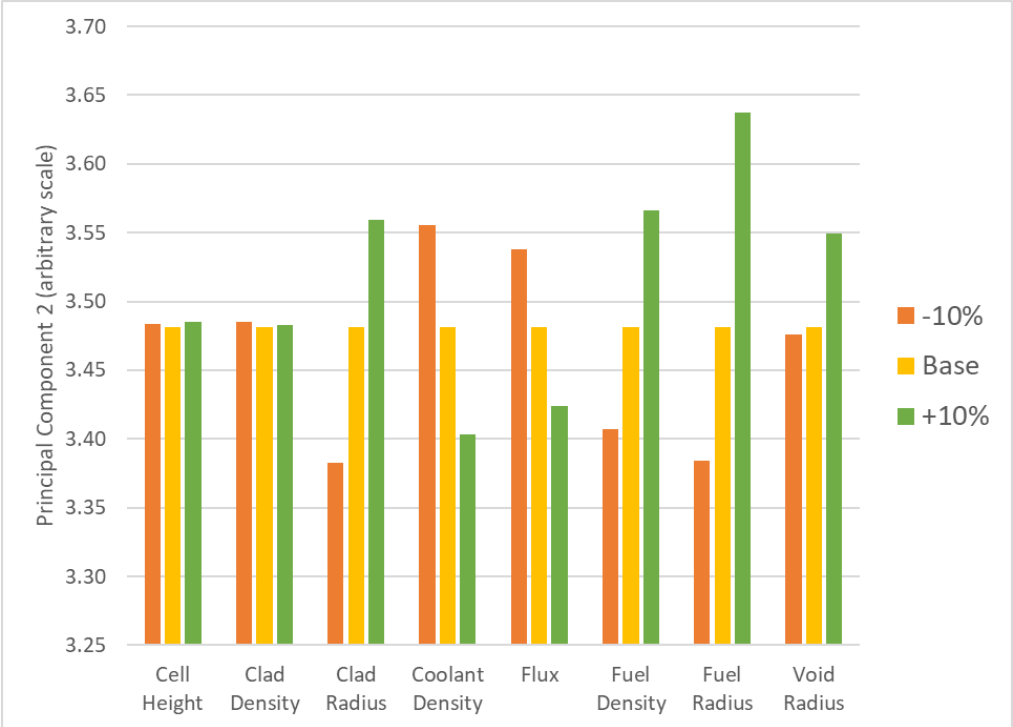


Figure 23 – XSgen Output Sensitivities on Principal Component 2.

Next, the output behavior of XSgen was tested in 2-dimensional space. Enrichment (fraction of U-235 in the fuel) and Fuel Density were chosen for the two input variables to study. Enrichment was varied between 2.5% - 7.0% with 0.5% increments, and Fuel Density was varied between 7 [g/cc] – 14 [g/cc] with 1 [g/cc] increments. This forms an 8x8 grid and therefore requires 64 runs of XSgen, which takes several days to complete.

The outputs are tabulated as function of fluence. It was observed that the initial behavior of the outputs (first 10 days of irradiation) were different than the final behavior of the outputs (at 2,000 days of irradiation). Therefore the results shown in Figure 24 are divided into two groups: the outputs from XSgen at initial fluence on the left side, and final fluence on the right side.

Figure 24 and Figure 25 show predictable dependence on the change of inputs. Initial values for burnup, neutron production, and neutron destruction are higher than their final values on the right side, whereas Pu-239 slowly builds in and appears in higher concentration for the final values. An increase in enrichment causes all these parameters to increase due to the increase amount of fissile material in the fuel. Increasing the fuel density seems to have a similar effect but lower in magnitude, especially for final values.

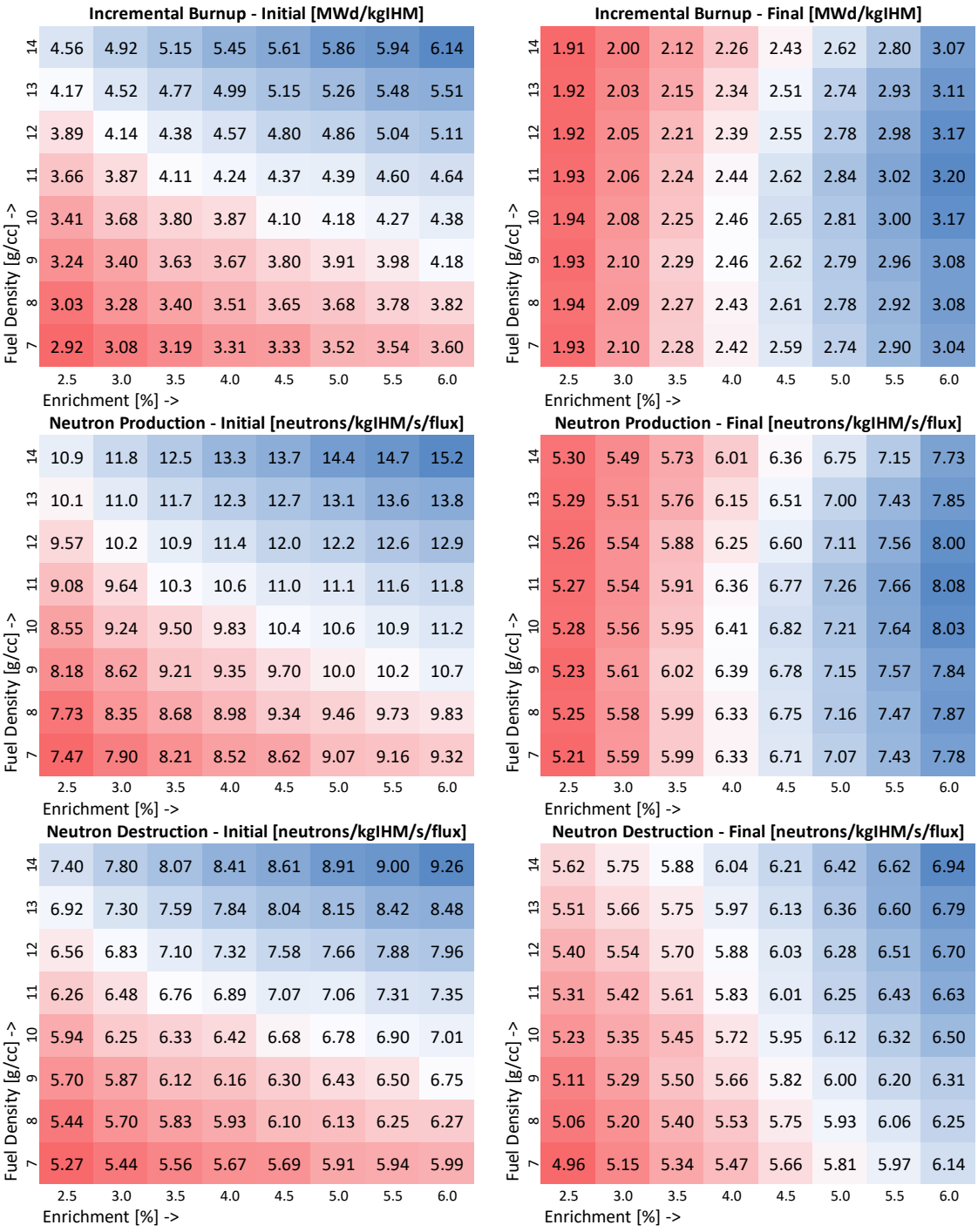


Figure 24 – XSgen Neutron Production, Neutron Destruction, and Burnup Output Map.

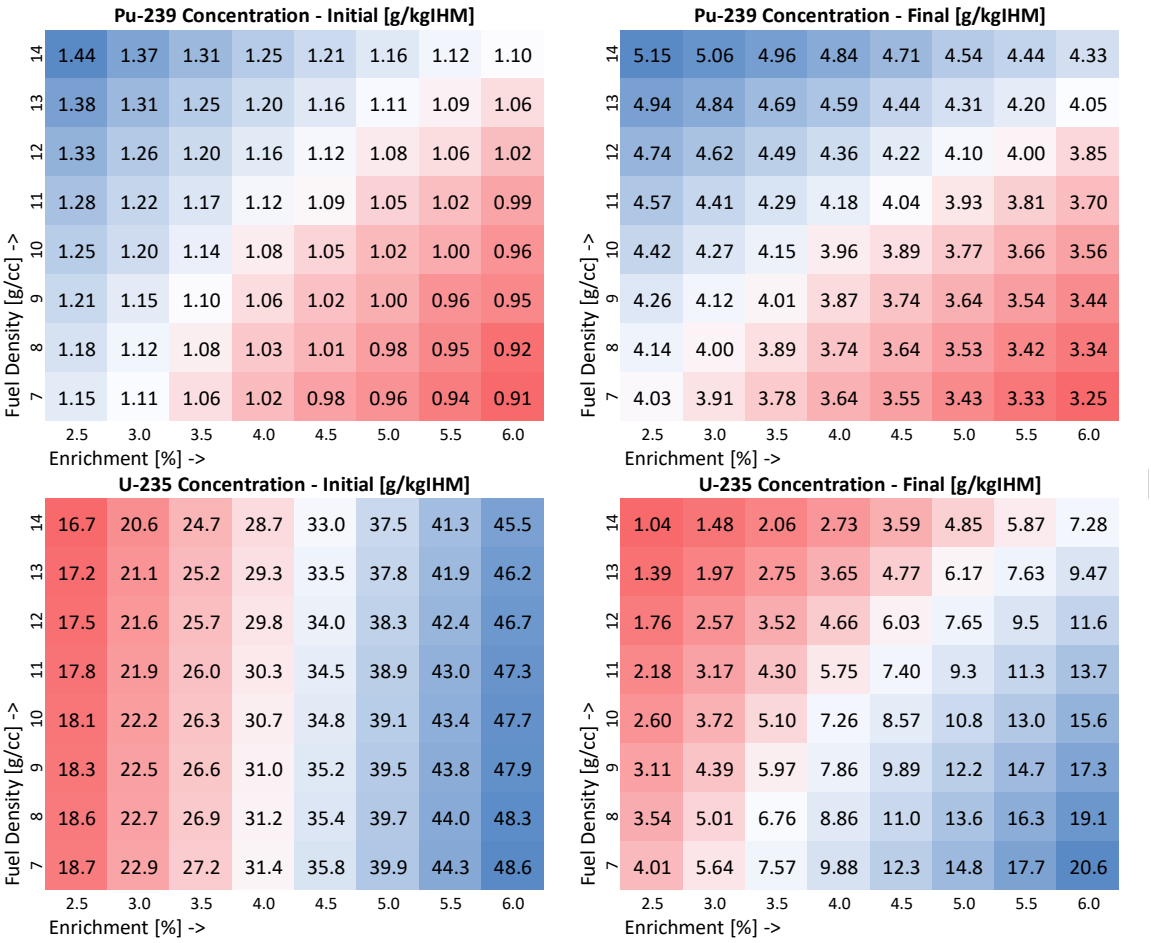


Figure 25 – XSgen Pu-239 and U-235 Concentration Output Map.

The principal components of these outputs are shown in Figure 26. They were calculated as described in Section 3.3.2. Data shows that the behavior of the output is closely captured by PC1, which accounts for 63.7% of the variation in the outputs. In contrast, PC4 seems to behave erratically unlike any output variable in the XSgen outputs, as it only accounts for 0.4 % of the output variation.

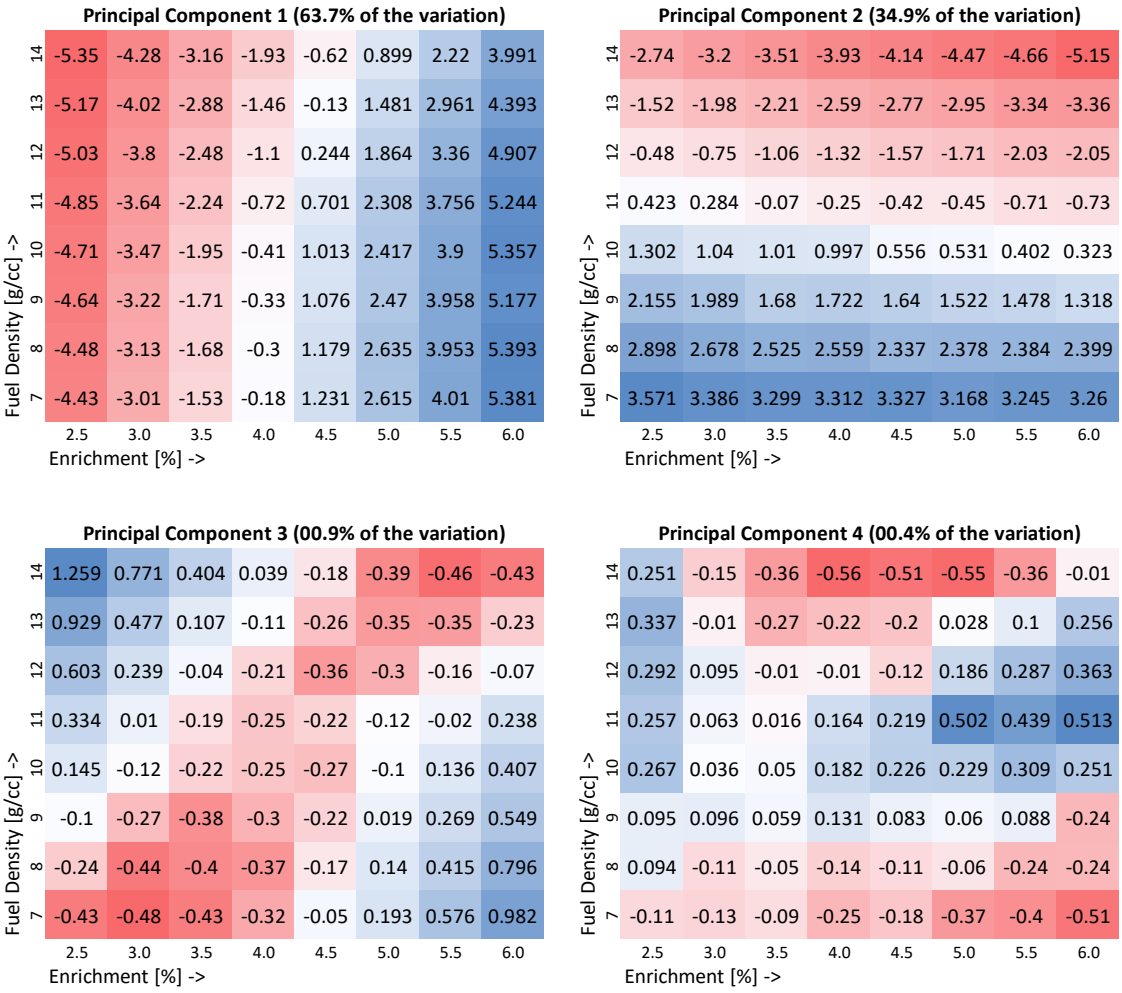


Figure 26 – XSgen Software Principal Components 2D Map.

4.5.Study of Database Building Parameters

This section presents results of studies on optimizing the values for parameters used in NUDGE. First, the effect of different values of the distance factor d_f (as described in Section 3.7) on final errors is reported. This study is followed by a similar one on the error weighing factor c (described in Section 3.5.3). Finally, the results from studying the effects of when to switch from exploration to exploitation are reported.

4.5.1. DISTANCE FACTOR

This subsection presents results of the study to find a potential optimal value for the distance factor d_f. This factor is defined in Section 3.7 and it controls the magnitude of Voronoi cell adjustment. There are several parameters that can be changed to evaluate the effectiveness of Voronoi cell adjustment under different conditions. These parameters include the number of dimensions of the database (D), the underlying function being modeled (the black box function), the exploitation strategy (when and how much to use exploitation as opposed to exploration), and total number of samples in the database (N). In addition, for each set of these parameters different values for the Voronoi adjustment factor (d_f) can be tried. Finally, since the database creation is a stochastic process, each database type needs to be repeated to be able to make statistically significant conclusions. As a result, this section aims to demonstrate that the Voronoi cell adjustment methodology is beneficial under some conditions, while being equivalent to no Voronoi cell adjustment when there is no benefit.

First example is a 2-dimensional database study. Each database has 50 total samples (20 exploration followed by 30 exploitation). The function being modeled in this study is given as Function F4 in Appendix 6.3 and shown in Figure 27. The complex behavior of the function can be observed from this figure, plotted in the range [0, 1] on both dimensions.

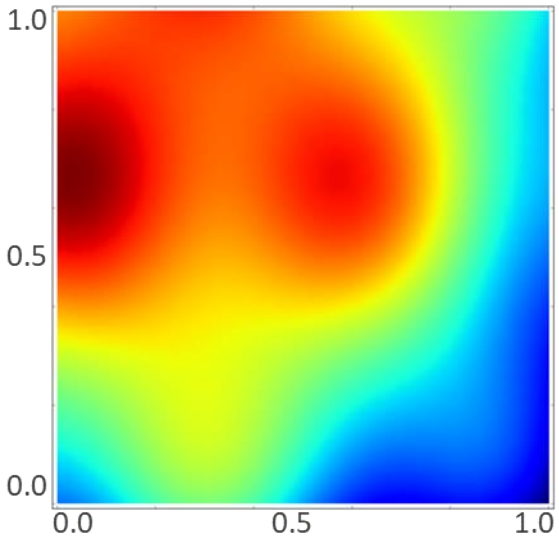


Figure 27 – 2D Case black box function.

This function was used to create databases with three different d_f values (including zero). The parameters for each database study and the average errors are given in Table 1 below. The uncertainty in the error values are calculated using the standard deviation across the repeated databases for a given type. It can be concluded that for this set of parameters, Voronoi cell adjustment is ineffective at making a statistically significant improvement of the final database error.

Table 1 – 2D Voronoi cell adjustment study.

Database Type	Repeated Databases	Voronoi Adjuster (d_f)	Samples per database	Real Error (%)	Max Real Error (%)
2D Furthest	18	0.00	50	0.0132 ±0.00042	0.194 ±0.0064
2D Guided 1	18	0.50	50	0.0147 ±0.00048	0.227 ±0.0069
2D Guided 2	18	0.85	50	0.0135 ±0.00039	0.191 ±0.0062

Five other 2-dimensional black box functions with various output behaviors were tested. Next, the placeholder XSgen program was tested with several 2-input combinations. Similar results were found for all, where the Voronoi cell adjustment did not cause a noticeable change in final errors.

3-dimensional cases were investigated next. Three types of databases were modeled using three different d_f values. The placeholder XSgen function in 3-dimensions was used for this study with inputs Coolant Density, Fuel Density, and Fuel Radius. Each input was varied plus/minus 20 % about their values from the base case (as given in Appendix 6.1).

The parameters used in these cases and the final errors of the databases are given next in Table 2. For this study, each database was started with 30 exploration samples followed by 50 exploitation samples. Note that the methodology for exploration is the same for all database types. The notably high maximum real error values are due to the high range of output values of the black box function.

Table 2 – 3D Voronoi cell adjustment study.

Database Type	**Repeated Databases**	**Voronoi Adjuster (d_f)**	**Samples per database**	**Real Error (%)**	**Max Real Error (%)**
3D Furthest	24	0.00	80	15.2 ±0.51	247 ±8.2
3D Guided 1	24	0.50	80	13.7 ±0.77	208 ±12
3D Guided 2	24	0.85	80	13.6 ±0.68	231 ±11

The values of the real error given in the previous table are visually presented next in Figure 28. It can be concluded that there is a statistically significant benefit of using Voronoi cell adjustment for this database type. The results indicate that the error can be improved, on average, in the order of 10 % compared to the error of the database with no Voronoi cell adjustment.

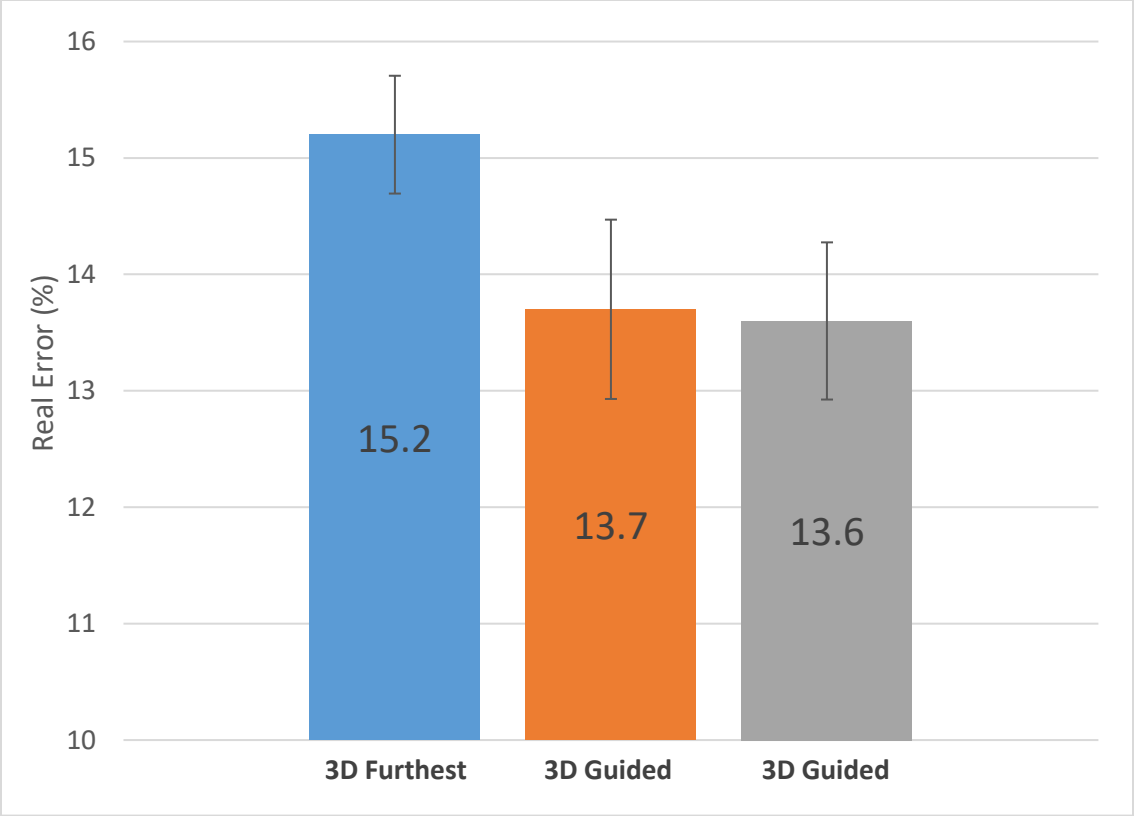


Figure 28 – Real errors of the 3D Voronoi adjustment study (y-axis starts at 10 %).

The final database study presented here uses two 7-dimensional database types. An 8-dimensional XSgen placeholder function was used for this study. The software was allowed to eliminate one dimension (Fuel Cell Height) during screening. The remaining dimensions were Cell Pitch, Clad Density, Clad Radius, Coolant Density, Flux, Fuel Density, Fuel Radius, and Void Radius. Each input was varied plus/minus 20 % from the base value (as defined in Appendix 6.1).

Images of this function are not shown due to the difficulty of presenting a select few 2D slices that demonstrate the behavior of a 7D function. A partial representation of the behavior of this function in lower dimensions is given in Section 4.4.

The parameters and results of the 7D study are shown in Table 3. Each database has 200 samples, where the first 60 are exploration samples and the remainder are found using exploitation.

Table 3 – 7D Voronoi cell adjustment study.

Database Type	Repeated Databases	Voronoi Adjuster (d_f)	Samples per database	Real Error (%)	Max Real Error (%)
7D Furthest	42	0.00	200	34.3 ±1.3	144 ±10.0
7D Guided	42	0.60	200	29.9 ±1.1	132 ±12.3

Looking at the errors it can be concluded that this case also demonstrates the benefit of the guided method, again in the order of 10 % improvement compared to the error of the database with no Voronoi cell adjustment.

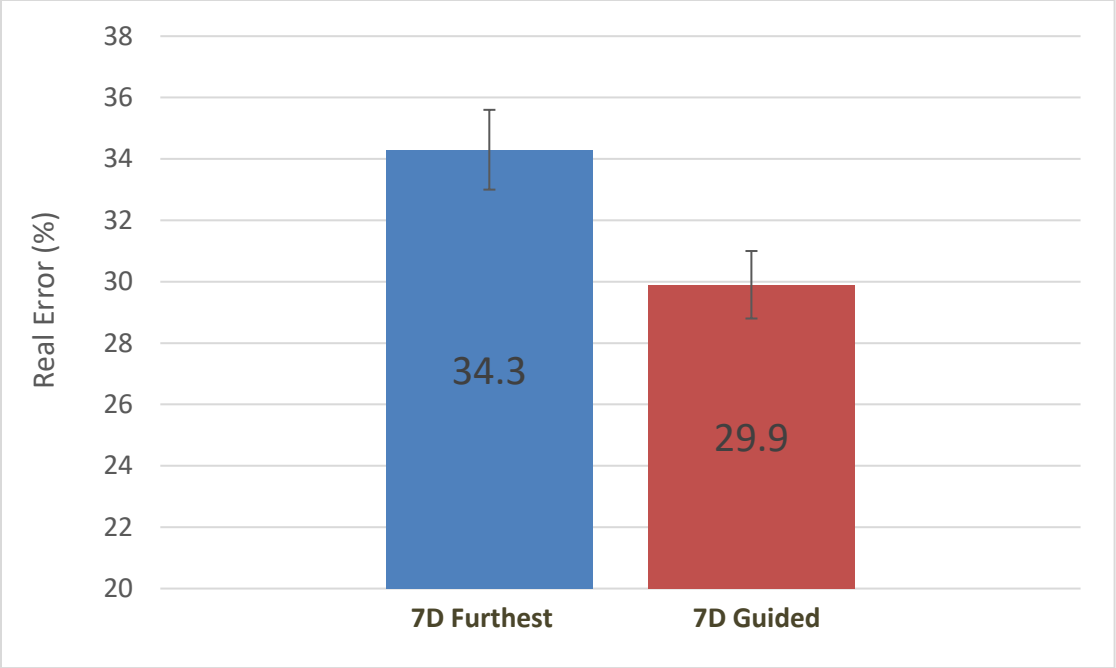


Figure 29 – Real errors of the 7D Voronoi adjustment study (y-axis starts at 20 %).

It has been observed that as the number of dimensions increase, the variation of the real error across databases of the same type also increase. This could be a result of the change in behavior of the underlying function as well. In any case, as a result of this behavior the highest max error of the databases in 7D Guided is in fact higher than the lowest max error of the databases in 7D Furthest. This is a natural result of the stochastic nature of the sample selection methodology.

It should again be noted that the potential improvement from using Voronoi cell adjustment depends highly on the parameters of the database and the features of the black box function. For example, any method will eventually reach an error of zero given enough samples (without collisions), albeit this number scales with number of dimensions. Also, given a very simple black box function (such as a simple hyperplane that can be expressed by a low order polynomial), only a few samples will be sufficient to model the behavior of the function and the sample selection method will not matter. However, given that a starting assumption of this work is that the black box function behavior in unknown before database creation, potential improvement methods to the error such as Voronoi cell adjustment should be considered.

4.5.2. ERROR WEIGHING FACTOR

This subsection presents results from the study for a potential optimal value for the error weighing factor *c.* This factor is used to multiply the sample error $E(\mathbf{p}_i)$ as given in Equation (8). If *c* is set to zero, then the errors of the samples are not considered during exploitation. Five values for *c* were selected ranging between 0.5 and 4. For each value of *c*, 36 databases were created and their real errors were averaged. Each database had 20 exploration and 30 exploitation samples totaling 50.

The placeholder XSgen function in 3-dimensions was used for this study. Several combinations of input parameters (as discussed in Section 4.5.1) were tested with equivalent results. The reported results here are for the inputs Coolant Density, Fuel Density, and Fuel Radius. Each input was varied plus/minus 20 % about their values from the base case (as given in Appendix 6.1). The real errors for each type of database (types here are defined by their *c* value) are plotted as a function of samples in the database in Figure 30.

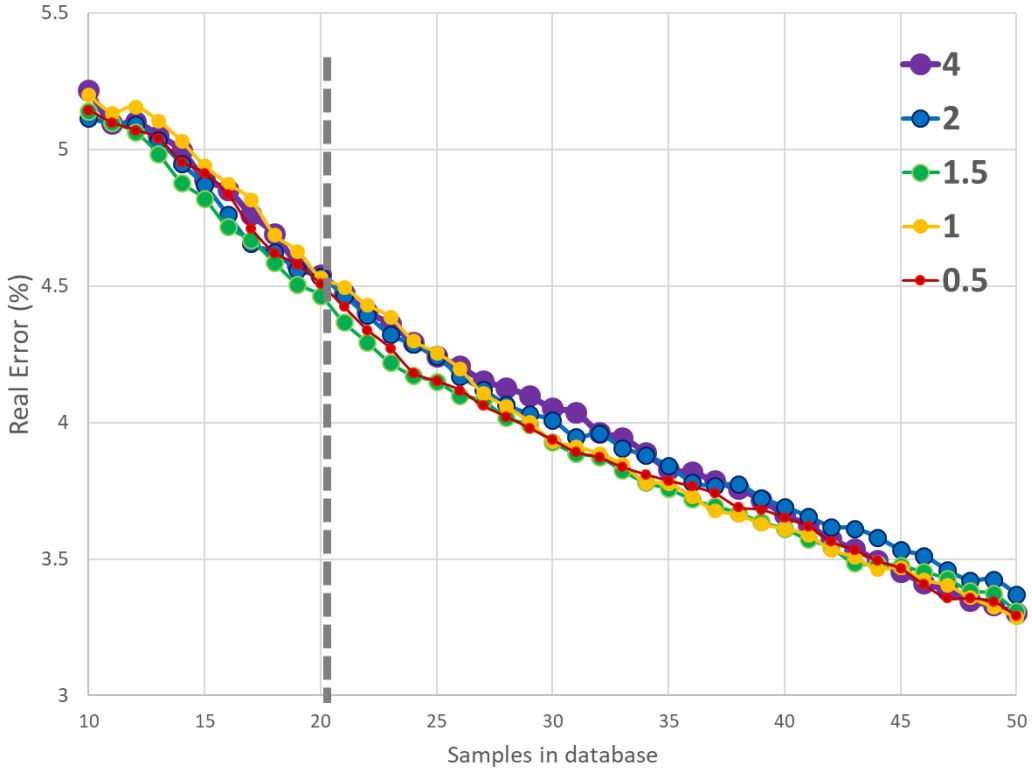


Figure 30 – Results of Changing the Error Weighing Factor c.

Note that in Figure 30 every point represents an average of 36 samples for that type of database. The uncertainties of the errors in this figure are all under 4 %. No difference between various values of *c* can be seen from this figure. Several other dimensions, samples, and inputs were tested with similar conclusions to the one given in this figure. It is concluded, based on the results of this placeholder function over the tested ranges, that as long as an extreme value for *c* is not used (c=1 is typical) the differences in the value for *c* become negligible. It should be emphasized that these conclusions heavily depend on parameter choices, i.e. dimensions, samples, inputs.

4.5.3. SWITCHING FROM EXPLORATION TO EXPLOITATION

The NUDGE methodology is defined to first perform exploration, followed by exploitation. However, it is possible to switch back to exploration to discover more of the output space. This section investigates various options of switching back-and-forth between the two strategies.

The placeholder XSgen function as well as five alternative placeholder functions were used to study the effects of switching strategies. The number of samples used varied using engineering judgement based on the errors of the databases. The switching strategies were tested by increasing the number of switches per database. One switch means there is one exploration stage followed by one exploitation stage, whereas two switches means the database was built by exploration-exploitation-exploration-exploitation. All studies ended in exploitation. A total number of 60 samples per database was used for 2-, 3-, and 4-dimensional cases.

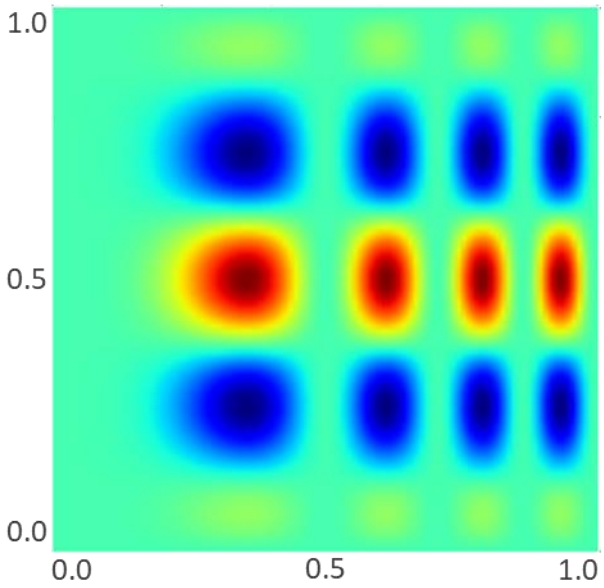


Figure 31 – Placeholder function (F3) output map over the utilized domain.

None of the studied options demonstrated clear benefit. The results are demonstrated on an example 2-dimensional placeholder function (Function F3 in Appendix 6.3) given below in Figure 31. This function is chosen for the relative complexity of the output (approximately 8 distinct peaks) which ranges between [0, 1]. It is adapted from a function taken from the reference manual of SciPy [44].

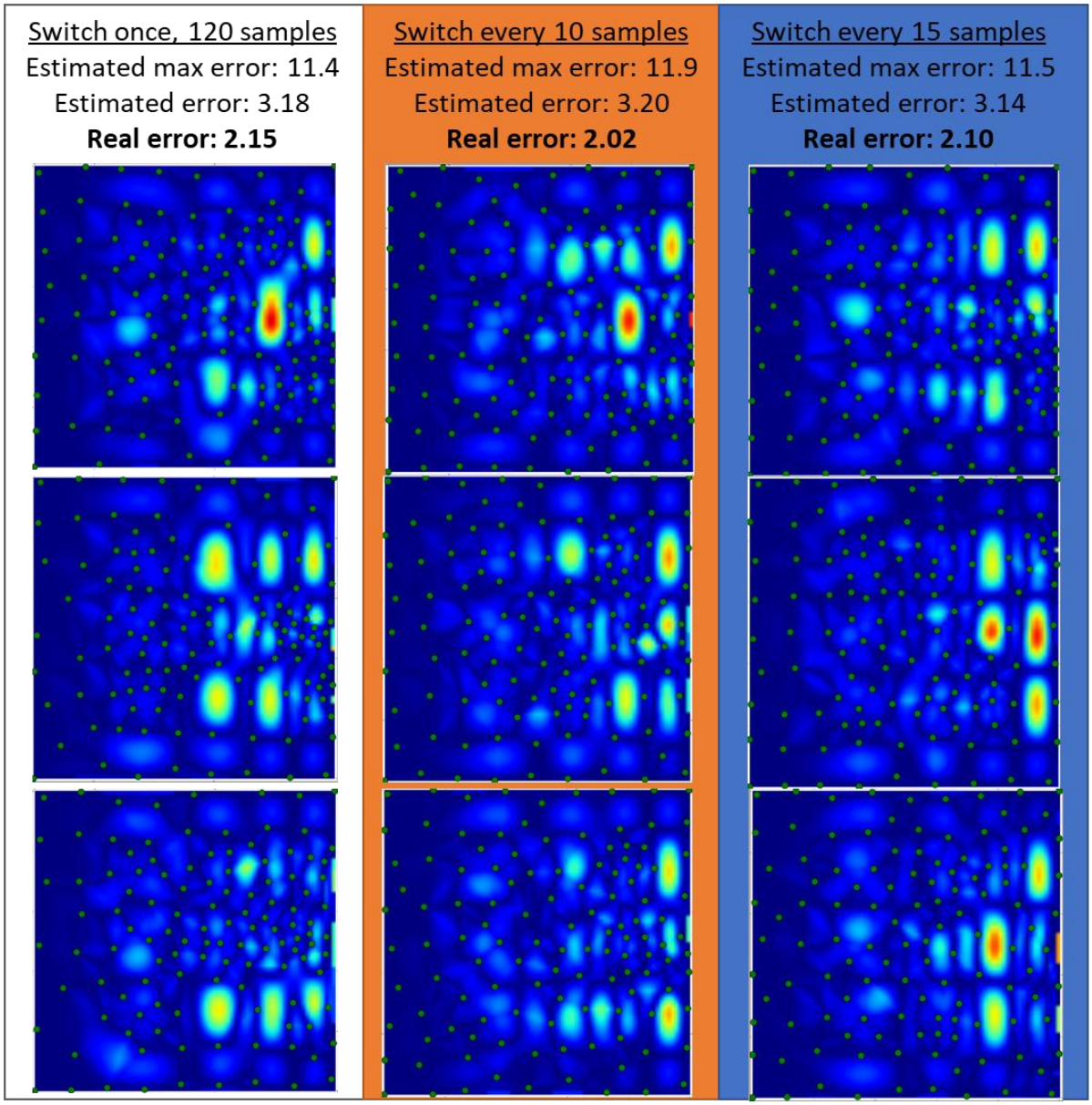


Figure 32 – Comparison of Various Switching Strategies.

Figure 32 shows maps of local errors. Each map is colored to show the real error of each point in the domain space (differences between estimated and real value of the function). Dark blue represents little to no difference while red represents the highest difference between the real output and the estimated output. The color scale is the same for all nine plots. Three examples of each strategy is presented in this figure to provide a representative set of databases under the given switching strategy.

It can be observed from Figure 32 that a statistically significant difference in final errors is not present (all final errors are close to 2 %). As stated previously, various dimensions, placeholder functions, and samples were tested with similar results. It is concluded that for a predetermined number of total samples for exploration and exploitation, the strategy used to switch between them does not have a statistically significant effect on the final database errors.

4.6. Building A Global Surrogate Model of XSgen Using NUDGE

Material composition vectors continuously evolve in MOX NFC's that have fuel recycling. This makes it challenging to predict the composition of a recycled fuel ahead of time, especially after several recycles. XSgen relies on accurate input material compositions for its results. Therefore, building a global surrogate model of XSgen for use in a MOX fuel cycle study can substantially decrease runtime while maintaining accuracy (compared to using one XSgen output or an arbitrary set of XSgen outputs in the fuel cycle study). This section presents results from a comparative study of building global surrogate models, including using the method presented in this work and implemented in NUDGE (this methodology is outlined in Section 3.1).

A researcher studying a MOX NFC utilizing XSgen and Cyclus has several options to account for changes in reactor feedstock compositions. In the first option, estimates of the input fuel compositions within the fuel cycle, if available, can be used for the XSgen runs ahead of NFC simulation runs. The results of XSgen runs will be saved in a database, effectively creating a surrogate model with user-provided sample input values. Since knowledge of the reactor feedstock compositions are unavailable when a NFC is simulated for the first time, none of the pre-computed XSgen outputs can be setup accurately for the fuel cycle prior to studying it. Therefore, this option will allow very fast runtimes during the fuel cycle study; however it has the highest expected uncertainty in the final results.

The second option is to provide a new XSgen run for every new input fuel composition encountered during a NFC. In a fuel cycle with several reactors this may require a new XSgen run for every fuel reloading. This option is expected to give the lowest uncertainty and as a result, but typically also a prohibitively long runtime.

The final option is to run XSgen several times during a precomputation step, and use these results to estimate XSgen outputs during NFC simulation runtime. This approach improves final results without sacrificing computational cost during NFC simulation runtime. As discussed in Section 2.2, there are many ways to pick the specific sets of values to use for XSgen inputs to build a surrogate model.

The benefit of using NUDGE and its methodology to build a XSgen surrogate model will be demonstrated in this subsection. The results will show that the fidelity increase per unit time invested during database building is highest for NUDGE compared to a space-filling only surrogate model building methodology (i.e. using only Exploration, Section 3.4) given the same constraints and parameters. The selection of the space-filling method for comparison against NUDGE is explained next, followed by an explanation of the NUDGE database building method.

SF Database: XSgen Surrogate Model Using Space-Filling (SF) Criteria

A standard and well-documented approach to building a surrogate model of a full simulation like XSgen is using space-filling criteria to select sample points, as discussed in Section 2.2. Space-filling criteria allows for an arbitrary number samples to be added to the model database, and works for any number of input dimensions (scaling of the method, as implemented for this work, is covered in Section 4.3). For these reasons, a space-filling criteria based surrogate model (SF Database) will be used to compare against the NUDGE XSgen database.

NUDGE Database: XSgen Surrogate Model Using NUDGE

The methodology described in this work and implemented in NUDGE software will be used to build the second XSgen surrogate model, named the NUDGE Database. The same base case, varied inputs and their ranges, number of dimensions, and number of database samples will be used as SF Database (these values are presented in the following pages).

The NUDGE Database will equally partition exploration and exploitation samples (based on results from Section 4.5.3). Exploitation will be done with Voronoi cell adjustment (Section 3.7). A distance factor (d_f) of 0.60 will be used (value chosen based on results from Section 4.5.1). The error weighing factor c will be assigned 1 (based on results from Section 4.5.2).

During the exploitation step of NUDGE the XSgen outputs will be quantified using PCA (see Section 3.3.2). Specifically, for each XSgen run, the output library for each

nuclide will be combined using the composition of the base case, and PCA will be applied to the output variables listed in Section 3.3.

All results from this section will be compared to the representative MOX fuel case taken from *Burn-up credit criticality benchmark: Phase IV-B, Case A* [45]. The XSgen input given in Appendix 6.2 is built to closely match this benchmark.

Base Case and Database Inputs

Both cases will be constrained to 150 samples in the database and 6 dimensions. Since adding one sample to the database (running XSgen once) takes well over an order of magnitude longer than the determination of the inputs for a sample, time constraints will not be considered. It is assumed that with basic software parallelization the sample selection step time differences will be negligible between SF Database and NUDGE Database (since XSgen runs take much longer than deciding what sample to run, regardless of the method).

The base case and input ranges will be kept the same for both database building methods (SF Database and NUDGE Database). The varied inputs will be input fuel compositions. The varied nuclides, their base case fractions, minimum and maximum values, and ranges are given in Table 4. Note that in this table the fraction of U-238 maintains total fuel mass. The base case for both methods is the same as the one given in Appendix 6.2, except the compositions of the nuclides given in this table.

Table 4 – Nuclide Fractions for XSgen MOX Global Surrogate Models.

Nuclide	Base Case	Min	Max	Range
Pu-238	5.59E-05	3.73E-05	6.99E-05	1.86E-05
Pu-239	1.22E-03	8.12E-04	1.52E-03	4.06E-04
Pu-240	5.79E-04	3.86E-04	7.24E-04	1.93E-04
Pu-241	2.10E-04	1.40E-04	2.62E-04	7.00E-05
Pu-242	1.58E-04	1.06E-04	1.98E-04	5.28E-05
U-235	6.51E-05	4.34E-05	8.14E-05	2.17E-05
U-238	*2.18E-02*	*2.13E-02*	*2.26E-02*	*1.29E-03*

The databases for SF Database and NUDGE Database methods were built, each with a total of 150 samples. Next, the benchmark input composition was fed to both database models to estimate XSgen outputs at those values. The estimation was performed on the 'fuel' outputs of XSgen at the fluence specified by the benchmark. In addition, XSgen was run directly for the benchmark inputs. The nuclide composition results these datasets are given in Table 5. The reference input composition and the benchmark output are taken from [45]. The fourth column (XSgen output) in this table shows the best estimate of XSgen for this benchmark (based on the input given in Appendix 6.2). The two models (SF Database and NUDGE Database) are considered to have zero surrogate error if they can exactly reproduce this data. Their benchmark error (difference from the benchmark data) are expected to be higher than their surrogate error (difference from the XSgen data).

Table 5 – Output Nuclide Fractions for XSgen MOX Global Surrogate Models.

Nuclide	Reference Input Composition	Benchmark Output	XSgen Output	SF Database Output	NUDGE Database Output
U-235	5.43E-05	4.37E-05	5.09E-05	5.67E-05	5.59E-05
U-238	2.14E-02	2.12E-02	2.11E-02	2.33E-02	2.32E-02
Pu-238	4.66E-05	4.13E-05	4.46E-05	4.95E-05	4.86E-05
Pu-239	1.02E-03	7.89E-04	9.25E-04	1.02E-03	1.03E-03
Pu-240	4.83E-04	4.77E-04	4.79E-04	5.30E-04	5.24E-04
Pu-241	1.75E-04	2.16E-04	1.71E-04	1.90E-04	1.87E-04
Pu-242	1.32E-04	1.36E-04	1.24E-04	1.39E-04	1.37E-04

Looking at Table 5, it can be observed that XSgen output has comparable values with that of the benchmark. SF Database and NUDGE Database outputs are closer to the XSgen output, since these datasets are generated from surrogate models of XSgen. The remaining outputs (tracked nuclides, neutron production, and neutron destruction) show similar behavior. Since these values are not available in the benchmark study they are not shown here.

The benefit of using NUDGE over the method of SF Database for this NFC case is quantified in Table 6. This table is constructed by comparing the outputs of the two

methods (SF Database Output and NUDGE Database Output) with the output of XSgen (XSgen Output). The error statistics given in this table are a result of comparing the output nuclide fractions. It can be seen that the NUDGE database has performed about 10 % better than the SF Database. This means that using the database created by NUDGE for this fuel cycle case and set of inputs provides approximately 10 % more accurate results compared to the results from SF Database.

Table 6 – XSgen Surrogate Model Errors Comparison.

	SF Database Surrogate Errors	**NUDGE Database Surrogate Errors**	**NUDGE Database Error Improvements**
Mean Error [%]	11.0	9.94	**10.5**
Median Error [%]	11.0	9.92	**11.2**
Max Error [%]	11.8	10.8	**8.65**

It should be noted that these results are highly dependent on the problem and model definition. The complexity of the simulation being used (in this case XSgen), the varied inputs and their ranges, the number of total samples in the database, and the scheme used to utilize the data in the database all affect the performance of the method for building a surrogate model. It is impossible to cover the entire range of these parameters to conclusively determine the benefit of any database building methodology due to the prohibitively large option space.

This case was chosen to be a representative use of XSgen and Cyclus, as MOX fuel cycles are commonly studied in industry and research (as evidence by the international participation in the chosen benchmark). Therefore, it is concluded from these results that there exists realistic use-cases where NUDGE provides a meaningful benefit to the accuracy of end results. It does not show that NUDGE will always deliver this benefit, especially for simple black box functions (as discussed in the end of 4.5.1).

5. CONCLUSIONS AND RECOMMENDATIONS

The challenge of retaining much of the fidelity of costly simulations without incurring their cost during runtime arises in many fields of science and engineering. Unlike previous work in the nuclear reactor simulation field, the methodology presented here does not limit the number and type of runtime simulation inputs. The method is meant to maximize the fidelity increase per unit time invested during a NFC simulation precomputation step. Pre-computation of data is needed in large NFC simulations due to the high computational resources needed to complete them. This data can then be utilized in NFC simulations without limit. Given that time will be spent and costly simulation runs are going to be performed, this work aims to find a method to gain the most benefit from this effort. This is achieved using NUDGE (NUclear Database GEneration software), which is an implementation of the methodology presented in this work.

The two types of error measures in NUDGE, estimated and real errors, are demonstrated. Various strategies for switching from exploration to exploitation are tested. It is shown that all tested switching strategies yield similar benefits given the same constraints.

An improvement to the exploitation step of this methodology, named Voronoi Cell Adjustment, is described, implemented, and tested. An improvement in the order of 10 % is demonstrated when using the Voronoi Cell Adjustment method compared to the error of the database with no Voronoi cell adjustment.

The scaling of the software is tested for increasing dimensions and total samples. It is concluded that doubling the number of dimensions will double the next sample calculation time for all steps; while doubling the number of samples will increase the next sample calculation time at most by a factor 4.

NUDGE has been used to create a global surrogate model of XSgen, based on a MOX benchmark. The model generated using NUDGE shows better performance compared to an alternative approach. It is concluded from these results that there exists realistic NFC simulation use-cases where NUDGE provides a meaningful benefit to the accuracy of end results.

NUDGE aims to reduce the time and cost for a researcher studying NFCs. There are many research questions where alternate fuel cycles and their metrics need to be studied. Examples where many NFC cases need to be simulated and compared include studies related to material security and nonproliferation, moving to a closed fuel cycle, elimination of the surplus of a given nuclide or material, tracking the movement of materials in a fuel cycle, and evaluation of the feasibility of new reactor technologies. In these cases, NUDGE allows the researcher to improve the fidelity of NFC simulation outputs without sacrificing additional runtime during the simulations. The impact of this work, therefore, can manifest as a reduction of total simulations needed to find an optimal answer.

It is recommended to further test NUDGE and its methodology for different black box functions, samples, and dimensions. The accuracy of the surrogate model under varying conditions can be studied to better understand the benefits and limitations of the methodology. In addition, the methodology can be adapted to include categorical inputs as well as a special treatment of samples near the edge of the domain. Since samples near the edge of the input domain are surrounded by fewer samples, their estimated errors tend to be determined to be higher.

It is predicted that the runtime of NUDGE can be significantly improved by software parallelization and optimization. For users who want to utilize NUDGE for full simulations other than XSgen, the software interfacing framework can be generalized for formats other than input-output file based frameworks. The dimensionality reduction methods available in NUDGE can replaced with a more robust, data analysis focused toolkit. The Voronoi Adjustment Method can be further investigated, and the adjustment of Voronoi cells potentially improved. Finally, it is recommended that new methods of utilizing the available database (such as machine learning) be tested by replacing the existing interpolation tool.

Surrogate models like NUDGE provide a good starting point to study NFCs. The methodology presented here does not limit the number and type of runtime simulation inputs, which allows users to pick inputs based on the nature of their studied problem. The

surrogate model can be used to quickly run any number of NFC simulations, while increasing the fidelity of a surrogate model can be as simple as adding more data to the model's database. This work provides a methodology to select the new samples to include in any given database to maximize the gain from the new data.

6. APPENDIX

6.1.XSgen Base Input for LWR

```
import numpy as np
from numpy import logspace
from xsgen.nuc_track import transmute
reactor = "0base"
plugins = ['xsgen.pre', 'xsgen.buk']
solver = 'openmc+origen'
formats = ('brightlite',)
burn_regions = 1
burn_time = 365*10
time_step = 100
burn_times = [0, 3]
burn_times.extend(range(100, 4001, 100))
batches = 3
fuel_cell_radius = 0.410
void_cell_radius = 0.4185
clad_cell_radius = 0.475
unit_cell_pitch  = 0.65635 * 2.0
unit_cell_height = 10.0
fuel_density = 10.7  # Fuel density [g/cc]
clad_density = 5.87  # Cladding Density [g/cc]
cool_density = 0.73  # Coolant Density [g/cc]
flux = 3e14
initial_heavy_metal = {     # Initial heavy metal mass fraction distribution
   922350: 0.033,
   922380: 0.967,
   }
fuel_chemical_form = {              # Dictionary of initial fuel loading.
   80160: 2.0,
   "IHM": 1.0,
   }
k_particles   = 1000      # Number of particles to run per kcode cycle
k_cycles      = 100       # Number of kcode cycles to run
k_cycles_skip = 30        # Number of kcode cycles to run but not tally at the begining.
group_structure = [1.0e-9, 10]
openmc_group_struct = np.logspace(1, -9, 101)
temperature = 300
track_nucs = ["U235", "U238"]
```

6.2.XSgen Input for MOX

```
import numpy as np
from numpy import logspace
from xsgen.nuc_track import transmute
reactor = "moxbenchmark"
plugins = ['xsgen.pre', 'xsgen.buk']
solver = 'openmc+origen'
formats = ('brightlite',)
burn_regions = 1
burn_time = 440
time_step = 20
burn_times = [0, 1, 3]
burn_times.extend(range(20, 460, 20))
batches = 3
fuel_cell_radius = 0.410
void_cell_radius = 0.4101
clad_cell_radius = 0.475
unit_cell_pitch  = 1.3127
unit_cell_height = 10.0
fuel_density = 10.7  # Fuel density [g/cc]
clad_density = 5.87  # Cladding Density [g/cc]
cool_density = 0.7245  # Coolant Density [g/cc]
fuel_specific_power = 16.0   # Power garnered from fuel [W / g]
initial_heavy_metal = {
    922340: 2.5952E-7,
    922350: 5.4287E-5,
    922380: 2.1387E-2,
    942380: 4.6610E-5,
    942390: 1.0156E-3,
    942400: 4.8255E-4,
    942410: 1.7491E-4,
    942420: 1.3201E-4,
    }
fuel_chemical_form = {
    50100: 0.000200,
    50110: 0.001000,
   80160: 2.0,
    "IHM": 1.0,
    }

k_particles   = 6000
k_cycles      = 150
```

```
k_cycles_skip = 30        # Number of kcode cycles to run but not tally at the begining.
group_structure = [1.0e-9, 10]
openmc_group_struct = np.logspace(1, -9, 101)
temperature = 600

track_nucs = ["Ac227",
              "Am241",
              "AM242",
              "BA140",
              "C14",
              "CM251",
              "CS141",
              "CS142",
              "CS147",
              "H1",
              "H3",
              "PU236",
              "PU237",
              "Pu238",
              "Pu239",
              "Pu240",
              "Pu241",
              "Th228",
              "Th229",
              "Th230",
              "Th232",
              "U230",
              "U231",
              "U232",
              "U233",
              "U234",
              "U235",
              "U236",
              "U237",
              "U238",
              "U239",
              "Zr93",
              "ZR95",
]
```

6.3. Placeholder Functions

FUNCTION F1

```
def f1(x, y, z):
    x1 = 0.75 * np.exp(-0.25 * ((9 * x - 2) ** 2 + (9 * y - 2) ** 2 + (9 * z - 2) ** 2))
    x2 = 0.75 * np.exp(- (9 * x + 1) ** 2 / 49 - (9 * y + 1) ** 2 / 10 - (9 * z + 1) ** 2 / 10)
    x3 = 0.50 * np.exp(-0.25 * ((9 * x - 7) ** 2 + (9 * y - 3) ** 2 + (9 * z - 5) ** 2))
    x4 = -0.2 * np.exp(-(9 * x - 4) ** 2 - (9 * y - 7) ** 2 - (9 * z - 5) ** 2)
    return x1 + x2 + x3 + x4 + 1
```

FUNCTION F2

```
def f2(x, y, z):
    return 1 / np.sqrt(1 + 2 * np.exp(-3 * (np.sqrt(x ** 2 + y ** 2 + z ** 2) - 6.7)))
```

FUNCTION F3

```
def f3(x, y):
    return x * (1 - x) * np.cos(4 * np.pi * x) * np.sin(4 * np.pi * y ** 2) ** 2 + 1
```

FUNCTION F4

```
def f4(x, y):
    A = np.sqrt(64 - 1 * ((x - 0.5) ** 2 + (y - 0.52) ** 2 + (z - 0.47) ** 2)) / 4
    return A + f2(x, y)/10 + f1(x, y, z) / 75
```

GLOSSARY

- BU(F): Burnup at fluence F [MWd/kgIHM]
- c: the error weighing factor
- D: number of dimensions (varied variables) in a database
- d: dimension index, ranges [1, D]
- d_f: The distance factor, used to determine the magnitude of Voronoi cell adjustment
- DR(F): Neutron destruction rate at fluence F [neutrons/kgIHM/s/flux]
- $E(\mathbf{p}_i)$: point $\mathbf{p}_i$'s share of the total estimated error
- ϵ_p: projection threshold (exploration method)
- Estimated error: the error of the database (percent difference between the output of the full simulation and the surrogate model) estimated using only the data within the database
- F: Fluence
- F_B: Fluence when the base case has a k-infinity of unity (exploitation method)
- $f(\mathbf{p}_i)$: the output of $\mathbf{p}_i$ from the full simulation
- $\hat{f}(\mathbf{p}_i)$: the interpolation operator that estimates the output of $\mathbf{p}_i$ utilizing all the samples in the database
- $\hat{f}^{(-i)}(\mathbf{p}_j)$: the interpolation operator that estimates the output of $\mathbf{p}_j$ using every point in the database except point i
- Full simulation: the black box function that the surrogate model estimates
- $k_{explore}$: The Monte-Carlo multiplier of the exploration method
- $k_{exploit}$: The Monte-Carlo multiplier of the exploitation method
- N: Database size (number of samples in the database)
- N_{screen}: Number of total screening samples in a database
- $N_{explore}$: Number of total exploration samples in a database
- $N_{exploit}$: Number of total exploitation samples in a database
- $\mathbf{P}$: Set of points in a database of size N, $\mathbf{P}=(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n, \ldots, \mathbf{p}_N)$

- $\mathbf{p_i}$: D-dimensional point in a database, $\mathbf{p_i}=[p_i^1, p_i^2, \ldots, p_i^d, \ldots, p_i^D]$
- $\mathbf{p}_{bN}$: Base sample, the sample with the highest rank score S in the database when the database has N samples
- PR(F): Neutron production rate at fluence F [neutrons/kgIHM/s/flux]
- $\mathbf{r}$: D-dimensional candidate point input vector, $\mathbf{r}=[r^1, r^2, \ldots, r^d, \ldots, r^D]$
- R: number of random points used during Monte-Carlo methods
- Real error: the error of the database (percent difference between the output of the full simulation and the surrogate model) determined using numerous runs of the full simulation
- $S(\boldsymbol{p}_i)$: the rank score of point $\mathbf{p}_i$ (exploitation method)
- $\theta(\mathbf{p}_i)$: the Voronoi adjustment factor of point $\mathbf{p}_i$
- $V(\boldsymbol{p}_i)$: The Voronoi cell size of point $\mathbf{p}_i$

REFERENCES

[1] C. Bagdatlioglu and E. Schneider, "Method for accounting for macroscopic heterogeneities in reactor material balance generation in fuel cycle simulations," *Nuclear Engineering and Design,* vol. 302, pp. 37-45, 2016.

[2] CNERG Fuel Cycle Group, "Cyclus," 01 01 2014. [Online]. Available: http://fuelcycle.org/. [Accessed 01 01 2014].

[3] C. Bagdatlioglu, R. Flanagan and E. Schneider, "Fuel Cycle Analysis Using Bright-lite in the Cyclus Simulator," in *PHYSOR*, Sun Valley, 2016.

[4] R. Flanagan, *Novel Methods for Generalizing Nuclear Fuel Design and Fuel Burnup Modeling (Doctoral dissertation),* University of Texas - Austin, 2015.

[5] L. Guerin, "Impact of Alternative Nuclear Fuel Cycle Options on Infrastructure and Fuel Requirements, Actinide and Waste Inventories, and Economics.," MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2009.

[6] J. J. Jacobson, G. E. Matthern, S. J. Piet and D. E. Shropshire, "VISION: Verifiable Fuel Cycle Simulation Model," Idaho National Laboratory, 2009.

[7] A. Brolly, M. Szieberth, M. Hal´asz and e. al., "Development and application of siton, a new fuel cycle simulation code," in *13th Information Exchange Meeting on Actinide and Fission Product Partitioning and Transmutation (IEMPT13)*, Seoul, Republic of Korea, 2014.

[8] L. Boucher and J.-P. Grouiller, *COSI: a simulation software for a pool of reactors and fuel cycle plants,* China: Atomic Energy Press, 2005.

[9] J. Grouiller, J. Vidal, A. Launay, Y. Berthion, A. Marc and H. Toubon, "CESAR: A Code for Nuclear Fuel and Waste Characterisation," in *WM*, Tucson, AZ, 2006.

[10] K. Crombecq, L. De Tommasi, D. Gorissen and T. Ghaene, "A novel sequential design strategy for global surrogate modeling," in *Winter Simulation Conference*, 2009.

[11] L. Pronzato and W. Muller, "Design of computer experiments: space filling and beyond," *Statistics and Computing,* vol. 22.3, pp. 681-701, 2012.

[12] R. T. Johnson, D. C. Montgomery, B. Jones and J. W. Fowler, "Comparing Designs for Computer Simulation Experiments," in *Winter Simulation Conference*, Miami, FL, 2008.

[13] F. Wahl, C. Mercadier and C. Helbert, "A standardized distance-based index to assess the quality of space-filling designs," *Statistics and Computing,* pp. 1-11, 2016.

[14] G. E. Box, S. Hunter and W. G. Hunter, Statistics for experimenters: design, innovation, and discovery, New York: Wiley-Interscience, 2005.

[15] K. Crombecq, E. Laermans and T. Dhaene, "Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling," *European Journal of Operational Research 214.3,* pp. 683-696, 2011.

[16] F. A. Viana, G. Venter and V. Balabanov, "An algorithm for fast optimal Latin hypercube design of experiments," *International journal for numerical methods in engineering,* pp. 135-156, 2010.

[17] B. G. M. Husslage, "Maximin designs for computer experiments," *Tilburg University, School of Economics and Management,* 2006.

[18] T. Long, D. Wu, Y. Wang and L. Liu, "A Sequential Maximin Latin Hypercube Sampling Method and Its Application to Aircraft Design," in *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Dallas, TX, 2015.

[19] P. Z. G. Qian, "Nested Latin hypercube designs," *Biometrika,* p. asp045, 2009.

[20] T. W. Simpson, J. D. Peplinski, P. N. Koch and J. K. Allen, "Metamodels for Computer-based Engineering Design: Survey and recommendations," *Engineering with Computers,* vol. 17, pp. 129-150, 2001.

[21] R. B. Gramacy and H. K. H. Lee, "Adaptive design of supercomputer experiments," Dept. of Applied Math & Statistics, University of California, Santa Cruz, 2006.

[22] R. Lehmensiek and P. Meyer, "Creating accurate multivariate rational interpolation models of microwave circuits by using efficient adaptive sampling to minimize the number of computational electromagnetic analyses," *IEEE,* pp. 1419 - 1430, 2001.

[23] L. Pronzato and W. G. Muller, "Design of computer experiments: space filling and beyond," Statistics and Computing, 2012.

[24] M. E. Johnson, L. M. Moore and D. Ylvisaker, "Minimax and maximin distance designs," *Journal of Statistical Planning and Inference,* vol. 26, no. 2, pp. 131-148, 1990.

[25] R. Joseph, E. Gul and S. Ba, "Maximum projection designs for computer experiments," *Biometrika,* vol. asv002, pp. 1-10, 2015.

[26] N. V. Queipo, R. Haftka, W. Shyy, T. Goel, R. Vaidyanathan and K. Tucker, "Surrogate-based analysis and optimization," *Progress in Aerospace Sciences,* vol. 41, pp. 1-28, 2005.

[27] M. Meckesheimer¤ and A. J. Booker, "Computationally Inexpensive Metamodel Assessment Strategies," *AIAA Journal,* vol. 40, no. 10, pp. 2053-2060, 2002.

[28] L. Van Der Maaten, E. Postma and J. Van den Herik, "Dimensionality Reduction: A Comparative Review," *J Mach Learn Res,* vol. 10, pp. 66-71, 2009.

[29] A. W. K. Donald, "Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation," *Econometrica,* vol. 59, no. 3, pp. 817-858, 1991.

[30] W. Svante, K. Esbensen and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems,* Vols. 2.1-3, pp. 37-52, 1987.

[31] G. Crevecoeur, H. Hallez, P. Van Hese, L. Dupre and R. Van de Walle, "EEG source analysis using space mapping techniques," Journal of Computational and Applied Mathematics, 2008.

[32] M. Redhe and L. Nilsson, "Optimization of the new Saab 9-3 exposed to impact load using a space mapping technique," Struct Multidisc Optim 27, 2004.

[33] M. Hintermuller and L. N. Vicente, "Space Mapping for Optimal Control of Partial Differential Equations," *SIAM Journal on Optimization 15.4,* pp. 1002-1025, 2005.

[34] L. Encica, J. Makarovic, E. A. Lomonova and A. J. A. Vandenput, "Space Mapping Optimization of a Cylindrical Voice Coil Actuator," *Industry Applications, IEEE Transactions on 42.6,* pp. 1437-1444, 2006.

[35] S. Tu, Q. S. Cheng, Y. Zhang, J. W. Bandler and N. K. Nikolova, "Space Mapping Optimization of Handset Antennas Exploiting Thin-Wire Models," *Antennas and Propagation, IEEE Transactions on 61.7,* pp. 3797-3807, 2013.

[36] R. Khlissa, S. Vivier, L. A. Ospina Vargas and G. Friedrich, "Application of Output Space Mapping Method for Fast Optimization Using Multi-Physical Modeling," IEEE, 2012.

[37] "XSgen," University of Wisconsin Computational Nuclear Engineering Research Group, [Online]. Available: https://github.com/bright-dev/xsgen. [Accessed 2016].

[38] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget and K. Smith, "OpenMC: A State-of-the-Art Monte Carlo Code for Research and Development," *Ann. Nucl. Energy,* vol. 82, pp. 90-97, 2015.

[39] M. Bell, "ORIGEN: the ORNL isotope generation and depletion code," Oak Ridge National Lab, Tenn. (USA), 1973.

[40] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing In Science & Engineering,* vol. 9, no. 3, pp. 90-95, 2007.

[41] K. Crombecq, E. Laermans and T. Dhaene, "Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling," *European Journal of Operational Research,* vol. 3, no. 214, pp. 683-696, 2011.

[42] E. W. Weisstein, "Voronoi Cell," From MathWorld--A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/VoronoiCell.html. [Accessed 15 11 2016].

[43] B. Ertl, "File:Manhattan Voronoi Diagram.svg," Wikimedia Commons, 2 2015. [Online]. Available: https://commons.wikimedia.org/wiki/File:Manhattan_Voronoi_Diagram.svg. [Accessed 15 11 2016].

[44] E. Jones, T. Oliphant and P. Peterson, "SciPy: Open source scientific tools for Python," 2001. [Online]. Available: www.scipy.org. [Accessed 02 2017].

[45] G. J. O'Connor and Nuclear Energy Agency, Burn-up credit criticality benchmark: Phase IV-B: results and analysis of MOX fuel depletion calculations., OECD, 2003.