

University of New Mexico

UNM Digital Repository

Mechanical Engineering ETDs

Engineering ETDs

Fall 12-12-2020

Numerical Evaluation of Effective Thermal Response of Heterogeneous Materials

Kevin W. Irick

University of New Mexico

Follow this and additional works at: https://digitalrepository.unm.edu/me_etds



Part of the [Ceramic Materials Commons](#), [Computational Engineering Commons](#), and the [Heat Transfer, Combustion Commons](#)

Recommended Citation

Irick, Kevin W.. "Numerical Evaluation of Effective Thermal Response of Heterogeneous Materials." (2020). https://digitalrepository.unm.edu/me_etds/203

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Mechanical Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Kevin W. Irick

Candidate

Mechanical Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Dr. Nima Fathi, Chairperson

Dr. Yu-Lin Shen

Dr. Patrick J. McDaniel

Dr. Jens Lorenz

**Numerical Evaluation of Effective Thermal Response of
Heterogeneous Materials**

by

Kevin Irick

B.S., Mechanical Engineering, Utah State University, 2014
M.S., Mechanical Engineering, Utah State University, 2014

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

**Doctor of Philosophy
Engineering**

The University of New Mexico
Albuquerque, New Mexico

December, 2020

DEDICATION

I owe a great deal of thanks to my first professional manager, Greg Saiz (Applied Technology Associates), for his vigorous support and confidence in my potential, clearing the path for me to become everything he saw in me. I am especially grateful to Brent Taft and Andy Williams (U.S. Air Force Research Laboratory) for supporting my first steps on this journey. Their selflessness and willingness to work with me and The University of New Mexico were the catalyst for the years of research and work that followed. I must thank Angel Urbina (Sandia National Laboratories) for adopting me professionally and supporting me personally in the final year of my degree, showing compassion, reason, and a healthy level of expectation.

Derek Hengeveld, who I hope will remain a long-time colleague and mentor, will not comprehend the level of respect and admiration I have for him. Derek is responsible for inspiring and encouraging me to take the leap towards this degree. His technical competency, stable character, and overall example provided a path I could not resist following.

Nima Fathi began as merely an enthusiastic teacher for me. In short time, he became my academic supervisor and ultimately my advisor. Most importantly to me, Nima has become a dear friend and mentor, whom I respect and rely on deeply. Nima's character is unparalleled, and his respect for humanity and his passion for education are inspiring and guiding in knowing who I want to be and what I want to represent as a person.

My four beautiful, wonderful, talented, joy-filled children, Wayne, Cally, Olsen, and Avery, are treasures beyond compare. Their presence gives life true meaning. I owe so much to my children, without whose help I could have only completed twice as much work in half the time. Yet, I would not trade my time with them for anything.

Jessi, my beloved wife, best friend, confidante, and voice of reason has been with me since the first sleepless nights of undergraduate engineering. My struggles have been hers struggles. My successes have been her successes. In the most difficult of times, Jessi has lifted me up, encouraged me, put up with me, and loved me. The best of times has had everything to do with my beautiful and courageous wife. Jessi's selfless sacrifice of time, energy, and personal pursuits has given definition to what it means to be a dedicated wife and mother. Jessi, I am so proud of you. I love you with all my heart and am eternally grateful you chose me as a friend.

ACKNOWLEDGEMENT

I express my sincere gratitude for moral, technical, personal, and financial support offered by Applied Technology Associates (ATA), the Sandia National Laboratories Verification and Validation, Uncertainty Quantification, and Credibility Processes organization (1544), and the United States Air Force Research Laboratory Space Vehicles Directorate Integrated Structural Systems program (AFRL/RVSV). Although specific aspects of research were not directly funded by the named organizations, their support and investment, in many ways, provided avenues and opportunities for effectively pursuing this research and completing this degree.

Numerical Evaluation of Effective Thermal Response of Heterogeneous Materials

by

Kevin W. Irick

B.S., Mechanical Engineering, Utah State University, 2014

M.S., Mechanical Engineering, Utah State University, 2014

Ph.D., Engineering, The University of New Mexico, 2020

ABSTRACT

Heterogeneity in materials leads to increased complication of physics, at the level or scale where heterogeneity exists, making analysis and material behavior predictions more difficult. As a result of the intricate descriptions or understanding of the physics required to determine the behavior of a heterogeneous material or system, practitioners frequently prefer the use of effective properties or responses. This body of work is dedicated to performing computational analyses on the thermal behavior of heterogeneous materials and evaluating the corresponding analyses using formal verification approaches. Custom finite element method code was used for solving the heat equation in a variety of two-dimensional heterogeneous systems. Formal verification methods—including the method of manufactured solutions and the grid convergence index—were used to assess code performance and solution veracity. Functional correlations were drawn between properties of heterogeneity and effective thermal responses.

TABLE OF CONTENTS

Dedication	iii
Acknowledgement	iv
Abstract	v
Table of Contents	vi
Preface.....	xiii
CHAPTER 1: Background.....	1
1 Introduction.....	1
2 Heterogeneous Media in Thermal Applications	4
2.1 Gas Turbine Power Generation	6
2.2 Nuclear Power	8
2.3 Flame Retardants	10
2.4 Electronics	11
2.5 Other	12
3 Porous and Composite Media Analytical Correlations.....	13
4 Porous and Composite Media Empirical Data.....	16
5 Heterogeneous Media Computational Analysis.....	19
6 A Note on Verification and Validation	21
7 Conclusions.....	23
References.....	24
CHAPTER 2: Computational Evaluation of Thermal Response of Open-Cell Foam with Circular Pore	33

Abstract.....	33
Nomenclature.....	34
1 Introduction.....	36
2 Background.....	38
3 System Description.....	40
4 Numerical Analysis.....	41
4.1 Numerical Approach.....	41
4.1.1 PDE Discretization.....	45
4.1.2 Solver	50
4.2 Code Verification	51
4.3 System Response Quantity	53
5 Results and Discussion	54
5.1 Code and Solution Verification	54
5.2 Thermal Results.....	58
5.3 Analytical and Empirical Comparisons	63
6 Conclusion	64
References.....	65

CHAPTER 3: Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis.....	69
--	----

Abstract.....	69
Nomenclature.....	70
1 Introduction.....	73

2	System Description	76
3	Numerical Approach	78
3.1	Discretization	78
3.2	Solver	85
4	Verification Approach	85
4.1	Code Verification	85
4.2	Solution Verification	87
5	Results and Discussion	90
5.1	Code Verification	90
5.2	Solution Verification	94
5.3	Characteristic Trends	99
6	Conclusion	101
	References	102

CHAPTER 4: High-Fidelity Calculation of Effective Thermal Response of Composite Media with Heat Generation Source

	Abstract	106
	Nomenclature	107
1	Introduction	110
2	System Description	113
3	Discretization	116
3.1	Domain	117
3.2	Partial Differential Equation	121

4	Verification	125
4.1	Code Verification	126
4.2	Solution Verification	128
5	Results.....	131
6	Conclusion and Future Work	147
	References.....	148
CHAPTER 5: Evaluation of Pore Geometry Effects on Porous Cell Thermal Behavior.....		151
	Abstract.....	151
	Nomenclature.....	152
1	Introduction.....	155
2	System Description	158
3	Discretization	162
3.1	Domain	162
3.2	Partial Differential Equation.....	166
4	Solution and Evaluation.....	169
4.1	Solution.....	169
4.2	Code Verification	171
4.3	Solution Verification	176
5	Results and Discussion	179
6	Conclusion and Future Work	195
	References.....	196

CHAPTER 6: Pore Geometry Effects on Iron Chrome Aluminum (FeCrAl) Alloy Foam	
Effective Thermal Conductivity	200
Abstract	200
Nomenclature	201
1 Introduction	205
2 System Description	211
3 Discretization	215
3.1 Domain	216
3.2 Partial Differential Equation	220
4 Solution Evaluation	223
4.1 Solution	223
4.2 Code Verification	225
4.3 Solution Verification	232
5 Results and Discussion	234
6 Conclusion and Future Work	257
References	258
Closing Remarks	263
Appendix A: Galerkin Finite Element Method Description	265
Generalized Equation	265
Linear Triangle Element	269
Appendix B: Solver2D Source Code	274
Appendix C: Gmsh Input Files	376

Circle Pore	376
Circle Filler	377
Triangle Pore.....	379
Square Pore	381
Ellipse Pore	383
Appendix D: SolutionPlot.....	386
Appendix E: Solver2D User Manual	395
1 Introduction.....	395
1.1 Finite Element Method	395
1.2 Gmsh Mesh Generation	396
1.3 Solver2D Program	396
2 Detailed Description	397
2.1 Galerkin Finite Element Method	399
2.2 Element Types and Shape Functions	399
2.2.1 Linear Triangle.....	399
2.3 Gmsh Mesh Generation	400
2.3.1 Domain Construction	400
2.3.2 Meshing.....	402
2.3.3 Mesh Export File	404
2.3.4 Summary	407
2.4 Solver2D Program	407
2.4.1 Data Types and Variables	410

2.4.2	Functions	410
2.4.3	Analysis Case Settings	413
2.4.4	Mesh Import	415
2.4.5	Materials.....	415
2.4.6	Continuums	416
2.4.7	Boundaries.....	418
2.4.8	User Subroutines	419
2.4.9	Compile and Run.....	419
2.4.10	Summary	420
3	SolutionPlot.....	420
4	Example Application	421
4.1	Domain Definition.....	421
4.2	Mesh Generation.....	422
4.3	Analysis	433
4.4	Visualization.....	439
	References.....	440
	Appendix F: Awards, Honors and Publications.....	441

PREFACE

Given that the style of this dissertation may be considered atypical, it is appropriate to preface the document with a brief description of the delivery approach. The body of work comprising this dissertation and the underlying research centers around the numerical evaluation of effective thermal response of heterogeneous media. During the course of research, successive developments were made with regards to the computational systems of interest, the computational code used, and the verification methods employed. Such research developments were documented implicitly in a number of technical and professional conference and journal publications. Thus, each chapter of this dissertation is a unique publication that is standalone in its own right, with the exception of the first background chapter, which was not published outside of this dissertation. The reader will notice consistencies between chapters, where subtle but important differences between chapters are present, a significant component of the progression of the research and methods over time.

The reader's attention is also directed to the semantics of the title of this dissertation, where an emphasis is placed on the "evaluation" portion of the work. Although the primary focus of the research at hand is the analysis of thermal system responses and synthesis of the combined computational data, the backbone of credibility for that data analysis is the evaluation—in other words, verification—of the processes used to obtain the results and the quality of the results. The methods used for performing the verification are well-established, generally speaking, but their use is relatively sparse, especially in the open literature with respect to this specific field of engineering and science. Thus, a secondary purpose of this research is to contribute to the public a communication of the credibility pedigree of the computational analysis results. This communication assists in the future application of the

thermal response data presented here as well as in providing an analysis framework for individuals performing additional studies in the future.

CHAPTER 1: BACKGROUND¹

1 Introduction

“Heterogeneity” is a broad—and relatively subjective—classification of materials, where a macroscale material (on a relative scale) is considered to be comprised of multiple, different components. The components comprising the heterogeneous material may be different fundamental materials, different orientations of materials, similar or different materials separated by boundaries, and/or different material phases. A special case of heterogeneous materials is a porous material, where voids (i.e., locales of material absence) exist within the bulk material. The foregoing list is not exhaustive, but what should be noted is the subjectivity of the classification “heterogeneous.” Heterogeneity—as opposed to homogeneity—can be determined from different perspectives such as from a length scale or from a physics perspective. For example, a continuous volume of some metallic solid may be considered homogeneous for most practical engineering purposes, such as to determine stress-strain relationships or heat transfer. In contrast, the same continuous volume of metallic solid may be considered heterogeneous when one considers the presence of distributed impurities or grain boundaries throughout the volume, perhaps affecting crack propagation under stress or material evolution during heat treatment.

Oftentimes, heterogeneity leads to more complicated physics at the level or scale where heterogeneity exists, making analysis and material behavior predictions more difficult. As a result of the intricate descriptions or understanding of the physics required to determine the behavior of a heterogeneous material or system, practitioners frequently prefer the use of

¹ The content of this chapter will be used for a review journal article submitted to an ASME journal.

effective properties or responses. Effective properties are a means of describing the general behavior of the system—based on the characterization of the pertinent heterogeneities or conditions in the system. A commonplace example is the use of a convective heat transfer coefficient for fluid flow in a given flow regime. Although complicated fluid mechanics and dynamics may be taking place—such as turbulent mixing and boundary layer formation—many fluid flow conditions and geometries can be simply characterized, and the effective resulting thermal transport behavior can be approximated using a straightforward analytical correlation. As engineering systems become increasingly complex, it can be advantageous—end even necessary—to use effective properties to analyze and predict system performance in a feasible manner.

Complex engineering systems, such as might be described by heterogeneous materials, typically require the use of computational analysis tools and numerical methods to gain insight to details of the system behavior. At a material level, empirical data can be extremely difficult—and sometimes impossible—to obtain that give meaningful insight into the impactful characteristics of heterogeneity in material thermal response. When using computational tools, effort should be made to ensure the veracity of the results. Systematic approaches have been developed to provide credibility evidence for the means of achieving the numerical results and for the numerical results directly. The execution of the credibility assessment approaches is captured by rigorous verification and validation (V&V) processes, some of which are described in detail in this work and are used to foundationally support the findings presented here. Although the intent of this work is not to present novel V&V approaches, the focus on V&V presented here serves two significant purposes. First, rigor is expected to be given to describing and analyzing the physical processes, data collection

methods, and measurement uncertainty quantification (UQ) used in obtaining empirical scientific data. The same rigor that is expected of empirical data acquisition should be expected when obtaining computational scientific data, especially when little or no empirical data is available and—even more importantly—when the computational data is used to predict system behavior. Second, V&V is generally undervalued, underused, and misunderstood (if recognized at all) in engineering practice but should be more universally implemented. With the ubiquity of computational analysis, the focus on V&V in this work aims to emphasize the need to bring V&V to the forefront of engineering practice, and a more thorough explanation of the process used to perform V&V must be given.

The work covered here presents numerical evaluation of the effective thermal response of heterogeneous materials. Both porous and non-porous heterogeneous materials are considered, where the scale of heterogeneity is such that classical thermal transport can be assumed using the general governing partial differential equation (PDE), the heat equation. The governing heat equation is succinctly described by

$$\nabla \cdot (\mathbf{K}(\nabla T)) + S = \rho c_p \frac{\partial T}{\partial t}, \quad (1)$$

where \mathbf{K} is the directional thermal conductivity tensor, T is temperature, S is the volumetric heat source, ρ is mass density, c_p is specific heat, and t is time. The need for a deep understanding of effective thermal responses of heterogeneous materials can be appealed in part by computational modeling tools, but rigorous V&V processes should underpin the subsequent findings and conclusions. The remaining sections of this chapter elaborate on pertinent applications involving heterogeneous materials, existing empirical thermal data for heterogeneous materials, analytical correlations for effective thermal responses of

heterogeneous materials, and V&V context that facilitates a credibility evidence package for the numerical results produced in this work.

2 Heterogeneous Media in Thermal Applications

Recent developments in technology, from the petroleum industry to geo- and nano-sciences, has created an exponentially increasing demand for investigating the physical properties of composite materials [1]. Porous media are two-phase composites comprising solid (matrix) and fluid (void) phases. The void may be occupied by either a gas or a liquid. When all the three phases coexist in a medium it is called an unsaturated porous medium. On the other hand, when only one of the phases is involved it is said to be saturated [2].

Heterogeneous materials are prevalent in many high-consequence and high-value systems, where accurate thermal design and engineering are fundamental to the system's success. Heterogeneity in a given material's composition can significantly impact the material's thermal response. Thus, it is advantageous to evaluate the impact of heterogeneous characteristics on a material's thermal behavior. Systems can be impacted by heterogeneity on a fine scale (e.g., at the material level) and on a larger scale (e.g., at the system level). Effects of heterogeneity are especially important in porous materials -- e.g., nuclear fuel components - where pores in the bulk material matrix impede and divert heat flow within the material. Oftentimes, heterogeneity is caused by porous structures within a material's solid domain. This is especially true in energy systems where thermal barrier coatings [3-5], semi-conductors [6], and/or energy storage materials [7] are used.

The complexity of heat transfer by conduction extremely increases in heterogeneous structures [8], including multi-component systems, complex microstructures and porous structures with dependent thermal conductivity values. Any discontinuity of material

properties, especially thermal conductivity which is caused by cracks, gaps, voids, and changing crystal structures, and nonuniform heat generation would need further investigation to understand the thermal behavior of the system of interest. Many advanced thermal systems rely on heterogeneous materials for reliable system performance.

Heterogeneous materials are pervasive across all forms of engineering fields and play a critical role in the performance of advanced technologies. In this work, the term “heterogeneous” is used to refer generally to any material with non-continuous material structure properties, where different regions of the material structure are comprised of different material properties and/or behavior. Examples of commonplace heterogeneous materials and applications can include aerogel insulations, foams, 3D-printed structures, nuclear fuel assemblies, structural composites, thermal barrier coatings, and electronics dielectrics. Oftentimes heterogeneous materials—especially ceramics or ceramic-derivatives—are used in high-consequence and high-temperature applications where sparse empirical data is available. A combination of application criticality and lack of real-world material performance measurements in high-consequence or high-temperature applications necessitates the use of computational approaches to assess material behavior.

The following subsections shed light on a variety of high-visibility thermal applications where heterogeneous materials play a critical role, with an emphasis on gas turbine and nuclear power generation. Other such applications include hypersonics, advanced materials, flame retardants, electronics, energy storage, geosciences, and additive manufacturing. This variety of specialized thermal applications emphasizes the wide-reaching need for a developed understanding into the effective thermal response of heterogeneous materials.

2.1 Gas Turbine Power Generation

The future of increased efficiency in gas turbine-driven power plant energy generation is heavily dependent on the increased temperature of the turbine inlet temperature (TIT) [9-12]. An example schematic of a gas turbine power generation system is given in Figure 1, illustrating how air is brought into the system, compressed, mixed with fuel, then ignited, creating the hot gas with the TIT before hitting the turbine blades. In spite of the pursuit of higher TIT levels, virgin metal material in the turbine components are susceptible to a variety of aggressive and high-consequence degradation or failure modes. Such issues include accelerated thermal creep, material degradation due to oxidation, and cycle fatigue [9,13]. Thermal-related corrosion issues arise at various temperature levels anywhere in the range of 650-1700 °C [10,14-16]. Figure 2 qualitatively describes the nature of various attack trends on the turbine blades with respect to temperature.

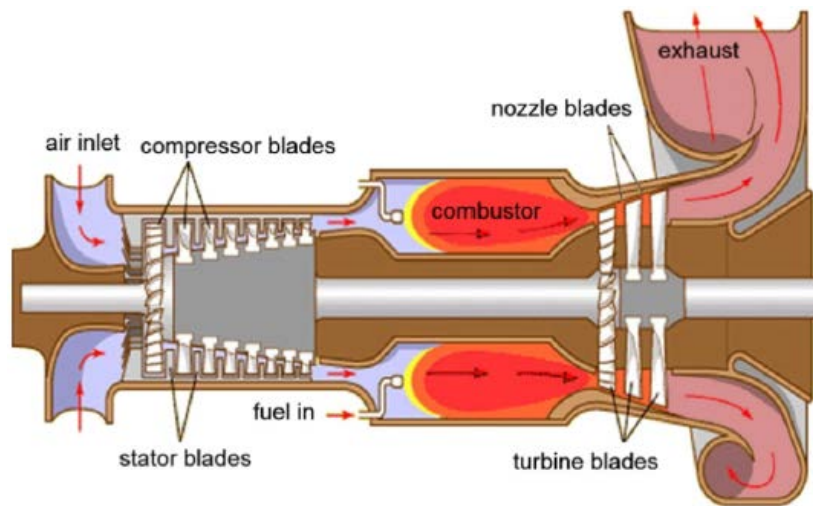


Figure 1. Schematic of a gas turbine power generation system [14]

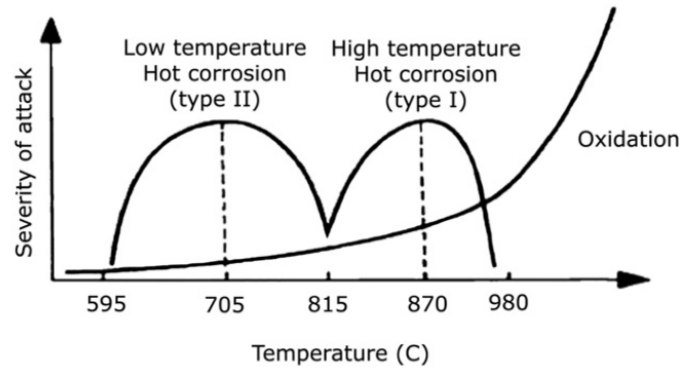


Figure 2. Thermally-induced degradation and failure trends [14]

To mitigate thermal issues, thermal barrier coatings (TBCs) are used to protect the virgin materials, as illustrated in the notional TBC-turbine blade schematic given in Figure 3. The TBC is bonded to the virgin metal turbine blade, acting as both an insulator and sacrificial layer to the virgin material against the flowing hot gas. Such coatings, including spray technologies are typically comprised of some form of ceramic composite material [9,17-19]. These composite materials are porous in nature and provide significant protection to the metal turbine components to allow for better overall performance and efficiency.

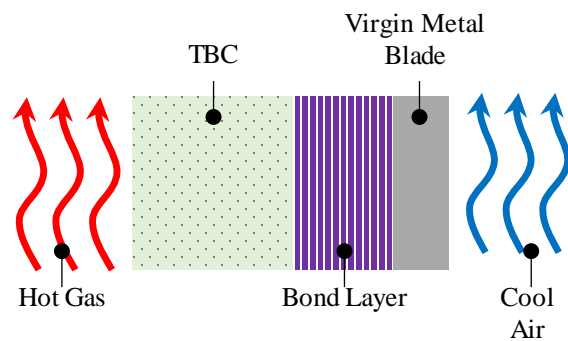


Figure 3. Notional schematic of thermal barrier coating application in gas turbine power generation system

It has been shown that the microstructures of the TBCs at a material level have significant effects on the effective thermal conductivity of the TBC. Effective thermal conductivity plays a significant role in the performance and longevity of the TBCs and consequently the turbine blades that they protect [3-5, 9].

Understanding the thermophysical properties of TBC materials is critical to the design and advancement of gas turbines for plant power generation for current and future installations [20]. As systems continue to produce higher TIT levels, the thermal performance of advanced TBCs becomes more critical. This investigation evaluates the effective thermal transport in simplified models of TBC porous structures, specifically two-phase porous media, as an initial study into TBC material nano-structure effective thermal response.

2.2 Nuclear Power

Nuclear fuels and fuel assemblies represent an important application of multi-scale material heterogeneity. Both the heat-generating fuel itself and the fuel assembly structure are heterogeneous or composite in nature [21,22]. An example of such a condition is illustrated by the advanced gas cooled reactor test experiment number two—illustrated in Figure 4—where the system is comprised of various components, including components made of heterogeneous materials [21]. Heterogeneous thermal conditions are crucial to heat transfer in nuclear fuel assemblies because the thermal behavior within the assemblies is governed significantly by the heterogeneous thermal conditions at both the system and component levels [23,24]. Having a better understanding of the thermal response of the unit cell of a composite that represents a fuel matrix cell would help to develop the next generation of nuclear fuel and understand potential performance enhancements.

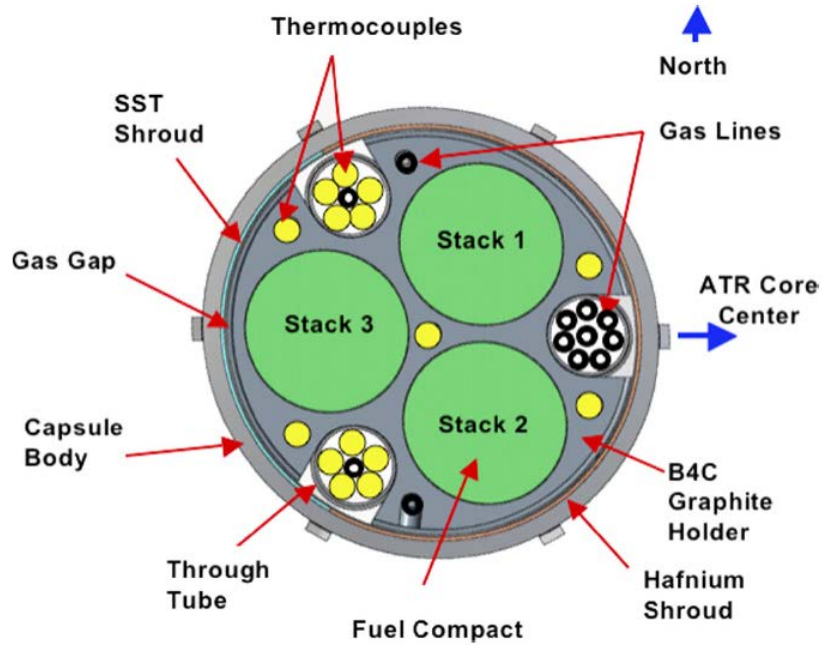


Figure 4. Advanced gas cooled reactor test experiment number two capsule cross-section [21]

The most conventional fuel material is uranium dioxide, UO_2 , because it exhibits suitable chemical and irradiation tolerances in thermal reactors. However, UO_2 's relatively low thermal conductivity can prove challenging from a thermal perspective. UO_2 is a porous media, where pore size, pore geometry, and pore distribution can have an effect on the thermal behavior of the fuel [25-31]. Likewise, if the UO_2 fuel were embedded in some non-heat-generating conductive matrix, the system would also be heterogeneous. Heat transport in such a critical system as nuclear fuel assemblies is a crucial to the advancement towards next-generation nuclear energy system development [32,33]. Failure modes and system performance in nuclear fuel assemblies depend on the thermal response and thermal tolerances of components and assemblies [23].

2.3 Flame Retardants

Carbon-nanotube-based materials can exhibit excellent flame retardant properties, potentially serving as safety and protective mechanisms in a wide variety of applications. In carbon nanofiber composites, the carbon nanotubes can act as filler materials in a bulk matrix, comprising a heterogeneous material structure, or can act as porous sheets protecting polymer surfaces. Kashiwagi showed the increase in thermal conductivity—implying increased heat dispersion—and heat rejection under incident radiant heat with higher concentrations of carbon nanotubes in the composite medium [34]. Likewise, Liu, Yang, and Ramanathan showed mechanical-thermal advantages to doping epoxy resins with graphene nanosheets [35-37]. Wu and Knight studied the thermal protection capabilities of porous carbon nanotube membranes [38,39]. Wu used these membranes on the surface of epoxy carbon fiber composites, where the pore size of the membrane was measured and used as a metric in characterizing the thermal performance [38]. An example image of carbon nanotube structures in a polymer matrix is shown in Figure 5. Understanding and characterizing the effective thermal properties of these types of heterogeneous materials can be critical in the development of next-generation flame retardants.

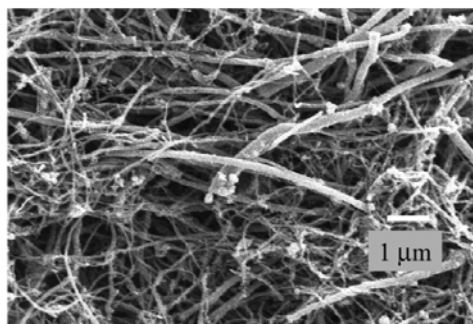


Figure 5. Multi-walled carbon nanotube structures in polymer-carbon composite with polymer removed [34]

2.4 Electronics

Heterogeneity in electronics materials plays a critical role in continued advancement of electronic technologies, especially as component packaging becomes smaller with higher heat generation, where heat density escalates. In many typical applications, enhanced thermal conductivity of polymeric electronic packaging material is crucial for component performance [40]. This is especially true for electronic components in aerospace systems where heat dissipation modes are heavily restricted and technologies performance must be knocked down for lack of heat dissipation capability [41,42]. Investigations of material thermal and electrical conductivity in electrical and electronic materials has been ongoing for over a decade, from fundamental physics [43-45] to conductive particle effects in polymer matrices [46,47]. For example, higher-conductivity nanocellulose particles can be mixed into polymer matrices to enhance thermal conductivity without significant compromise of dielectric properties [40, 48] and predictive models for thermal conductivity behavior can be developed [6]. Figure 6 shows an example of a polymer material filled with higher-conductivity material to enhance thermal conductivity of electronic packaging. The need for in-depth heterogenous material thermal response is essential to the future of electronics developments and their success in application.

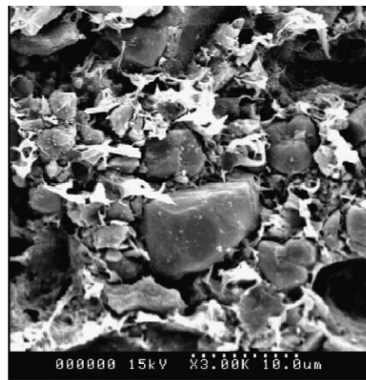


Figure 6. High density polyethylene filled with 60% aluminum nitride by volume [47]

2.5 Other

In addition to the critical industries described above, many other high-impact fields rely on heterogeneous materials as a foundational aspect of their functionality and performance in thermal applications. Figure 7 illustrates three examples of heterogeneous material structures in different applications. As an application example, many energy storage system concepts depend on multi-phase and/or multi-component materials [7,32]. Hypersonic flight vehicles experience extreme heat loads on leading edges and rely on composite and ceramic materials to prevent destruction of the vehicle [49-51]. In geosciences, evaluation of thermal behavior of heterogeneous—especially porous—earthen regions is important in assessing environmental responses to both natural and man-made conditions [24, 52-55]. The additive manufacturing industry has a natural need for understanding thermal transport in porous materials—considering the non-homogenous and high-temperature processes used—where additive manufacturing can be used specifically for thermal applications [56-58]. The thermal performance and advantage of other advanced materials—such as aerogels and doped polymers—are driven by their heterogeneous characteristics and can be seen in marine, oil and gas, aerospace, energy, and thermal management industries [59-62]. The expansive set of industries and specific thermal applications where heterogeneous materials are of immediate consequence is immeasurable. Thus, the pursuit of measuring, analyzing, and predicting the thermal performance of such materials is in high demand.

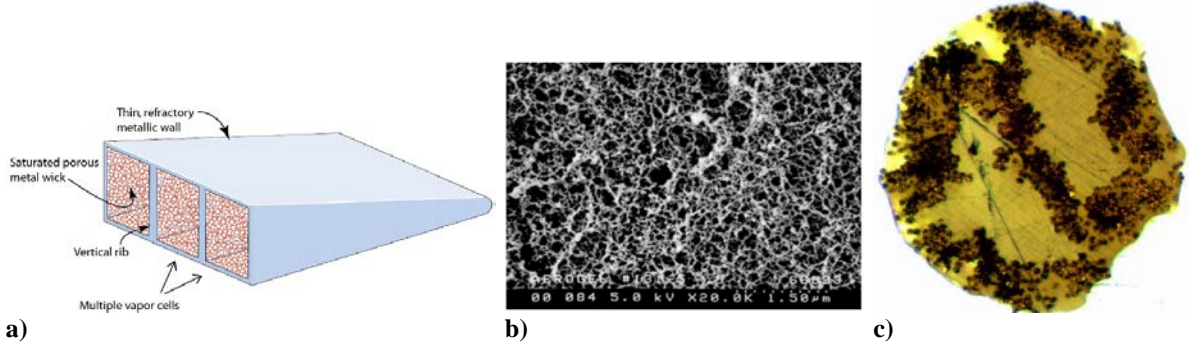


Figure 7. Examples of a) porous structure hypersonic leading edge [51], b) silica aerogel insulation material network [63], and c) 3D printing carbon fiber-polymer strand [58]

3 Porous and Composite Media Analytical Correlations

Since as early as the late 1800s, heavy interest has been placed on correlating the impact of heterogeneous properties in materials and media with the effective resulting material behavior, especially with regards to effective electrical and thermal conductivity. Note that electrical and thermal conductivity behavior are analogous, and, thus, correlations for effective electrical conductivity in this context can be extended to effective thermal conductivity. Following is a brief description of a historic evolution of published correlations surrounding effective thermal conductivity in heterogeneous media. Generally, the correlations relate the thermal conductivities of a matrix (continuous portion) and filler (discontinuous portion) material, k_1 and k_2 , respectively, the fill fraction of filler material volume to total material volume, α , to the effective thermal conductivity of the composite material, k_{eff} . For the content presented in this research, the dimensionless effective thermal conductivity, k^* , represents the ratio of the effective and matrix thermal conductivity such that

$$k^* = k_{eff}/k_1. \quad (2)$$

In 1873, Maxwell published “A Treatise on Electricity and Magnetism,” an extensive document discussing various properties of electrical behavior. Maxwell’s chapter dedicated to

conduction through heterogeneous media naturally parallels heat conduction [43]. Maxwell's correlation can be expressed as

$$k^* = \frac{(2\alpha + (3-2\alpha)\frac{k_2}{k_1})}{((3-\alpha)k_1 + \alpha k_2)}. \quad (3)$$

Note that Maxwell assumes that the volumetric ratio of filler material to matrix material is relatively small. In 1915, Burger extended Maxwell's analysis to include ellipsoidal fillers (as opposed to spherical fillers) [44].

Later, in 1892, Lord Rayleigh published a mathematically meticulous derivation for the effective thermal conductivity of a medium with cylindrical obstacles (fillers) packed in rectangular order [64], where he expressed

$$k^* = 1 - \frac{2\alpha}{\left(\frac{(1+k_2/k_1)}{(1-k_2/k_1)} + \alpha - \frac{3\left(1-\frac{k_2}{k_1}\right)\alpha^4}{\left(1+\frac{k_2}{k_1}\right)\pi^4} a_4^2 - \frac{7\left(1-\frac{k_2}{k_1}\right)\alpha^8}{\left(1+\frac{k_2}{k_1}\right)\pi^8} a_8^2 \right)}, \quad (4)$$

with a_4 and a_8 being particular infinite trigonometric summation series.

In 1952, Landauer presented a theoretical correlation for the effective electrical resistance of binary metallic media. As in previous works, Landauer assumes that conductive matrix medium is populated with spherical conductive fillers [45], and suggested that

$$k^* = \frac{1}{4k_1} \left((3\alpha - 1)k_2 + (2 - 3\alpha)k_1 + \left(((3\alpha - 1)k_2 + (2 - 3\alpha)k_1)^2 + 8k_1k_2 \right)^{\frac{1}{2}} \right). \quad (5)$$

In his work, Landauer does make a comparison of his correlation with empirical measurements of conductivity in multiple metallic materials over a wide range of fill ratios.

In 1962, Hamilton and Crosser presented a correlation for three-dimensional systems. The derivation of their correlation was informed by Maxwell's equation, but was extended for use in systems with particles of arbitrary geometry. However, the correlation involves a

parameter, n , that must be empirically-tuned to measurement data [65]. The Hamilton-Crosser correlation is given to be

$$k^* = \frac{(k_2 + (1-\alpha)(n-1)k_1)}{(k_2 + \alpha(k_1 - k_2) + (n-1)k_1)}. \quad (6)$$

In 1974, Nielsen summarizes results from both Lewis and Nielsen [66,67], where the correlation—as opposed to Hamilton’s and Crosser’s correlation—does not involve any empirically-tuned parameters [68]. Derived using the theory of elastic moduli of composite materials, the correlation takes into account the shape of the filler particles and the orientation of those particles with respect to the bulk heat flow through the composite material. Nielsen compares the correlation to a selection of empirical test results with relatively good agreement across a spectrum of fill fractions. Nielsen’s equation is

$$k^* = \frac{1 + (k_e - 1) \left(\frac{\frac{k_2}{k_1} - 1}{\frac{k_2}{k_1} + (k_e - 1)} \right) \alpha}{1 - \left(\frac{\frac{k_2}{k_1} - 1}{\frac{k_2}{k_1} + (k_e - 1)} \right) \left(1 + \left(\frac{1 - \alpha_m}{\alpha_m^2} \right) \alpha \right) \alpha}, \quad (7)$$

where k_e is the generalized Einstein coefficient, and α_m is the defined as the densest possible fill fraction of the given particle geometry. Figure 8 illustrates possible variations across some of the analytical correlations described above, where theories converge to agreement at low porosity values and spread drastically as porosity increases.

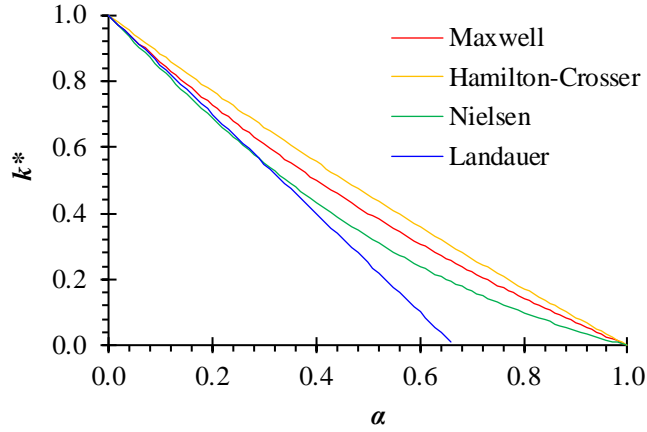


Figure 8. Selected analytical correlations for effective thermal conductivity as a function of porosity

A number of other works and investigations have been presented in the literature to provide theoretical and analytical correlations to describe the effective thermal conductivity in heterogeneous materials. Such works consider percolation theory (the concept non-separated filler particles [69-73]), component stratification [74,75], and interface resistance [76-78]. More developed correlations can be found in the literature based on specific types of applications [52,53,79-85].

The works and correlations described above do not constitute an exhaustive list of the attempts to analytically describe effective thermal conductivity in heterogeneous materials but illustrate the persistent and evolving interest in the matter. For referential reviews or summaries of the works mentioned above and other works, suggested readings include [86], [87], [88], and [89] for their concise and clear communication.

4 Porous and Composite Media Empirical Data

A variety of empirical data is available in the literature for the effective thermal conductivity of materials with respect to the material porosity. Although various studies show that multiple factors can impact the effective thermal conductivity of a porous material's

thermal response, a primary parameter is the porosity. A few sources have been identified to illustrate possible ranges of effective thermal responses over a large range of porosity values in materials. The investigation in [90] measured the effective thermal conductivity of porous zirconia and alumina ceramics which covered a range of porosity values from rough 7% to 48%. In [91], researchers measured thermal conductivities in porous inorganic polymer cements ranging from 30% to 70% porosity [92]. Thermal conductivity in porous building materials with porosities ranging from 10% to 50% was investigated in [93]. In [94], data was presented on highly porous zirconia ranging from 45% to 72% porosity. Thermal conductivities of organosilicate films as a function of porosity for material with porosity ranging from 10% to 50% were presented in [95]. Thermal conductivity of Xerogel dielectric films have been measured against porosities at 48% to 77% by [96] and 25% to 80% by [97]. Selected empirical data sets for porous media from the literature are plotted in Figure 9, illustrating the relatively large variation in k^* across the spectrum of porosity levels observed in reality.

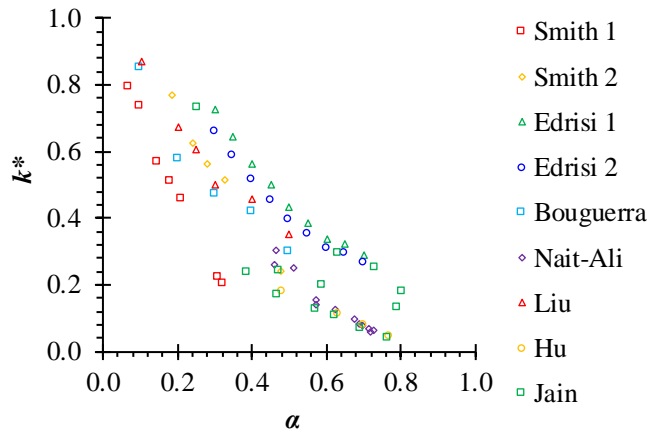


Figure 9. Selected empirical data sets for effective thermal conductivity as a function of porosity

As with porous media, general composite media (i.e, matrix material with filler components) empirical data is readily available in the literature for a variety of applications and media. For example, [47] presents effective thermal conductivity for high-density polyethylene filled with aluminum nitride, silicon carbide, boron nitride, and wollastonite to fill fraction of 10% to 75%. Effective thermal conductivity in 3D printed structures was investigated by [98] for ABS and ULTEM polymers with varying air gaps, and [99] shares empirical data for various 3D printed polymers and metals with a testing apparatus designed specifically for 3D printed materials. In [100], data was presented on PEEK filled with silver particles up to a 15% fill fraction to assess conductive composites for aeronautical applications. Empirical data on epoxy resins filled with aluminum particles from 5% to 45% fill fraction and cupric oxide particles from 2% to 27% fill fraction is presented in [101]. The effects of filling polypropylene with zinc oxide and calcium carbonate particles from 5% to 15% by weight was shown in [87]. Packed quartz sand beds were filled with water, air, and oil in [52] and consolidated sandstone was compared to unconsolidated sand in [53]. Experiments were performed to show the effects of adding water to porous soil over a wide range of fill percentage in [102] and [103].

Where empirical data was presented but not explicitly tabulated by the authors in the above references, plot data extraction and digitization were performed using an online tool from [104]. The aforementioned data sources represent a brief sample of the variation of available empirical data available on porous and composite material thermal conductivity, including a wide range of porosity and fill fraction values.

5 Heterogeneous Media Computational Analysis

Regarding the effective thermal response of heterogeneous materials, computational analyses can provide complimentary insight to empirical data provided by physical experiments. Numerical models and analyses allow for investigation into fine physical details of media and physics phenomena where empirical data may be limited by physical or temporal constraints. A wide variety of research has focused on performing computational analysis of heterogeneous material responses with varying approaches and perspectives. Some published analyses—such as those given in [105-108]—investigate effective thermal conductivity response in two-dimensional systems, whereas others—such as those given in [109-113]—analyze three-dimensional systems. Most commonly, elliptical shaped pores and fillers appear in two-dimensional system analyses [105-107], and ellipsoidal and cylindrical shaped pores and fillers appear in three-dimensional system analyses [109-113]. Some research incorporates the use of sample sets of random particle topological and geometric distribution to quantify statistical variations in heterogeneous media effective thermal responses [108,113]. Depending on the focus of the research, analysts also include interface thermal resistance between filler particles and the encapsulating matrix medium which has the potential to significantly alter the temperature distribution within the material domain and affect the overall effective thermal conductivity [105,107,109,111]. These descriptions of existing works are only an illustrative sample of the types of analyses available in the literature.

Figure 10 displays some example computational analysis approaches that have been used to better understand and predict the behavior of heterogeneous materials in thermal applications. It appears common for practitioners to use a finite element method as the primary numerical approach for solving the thermal response of heterogeneous material systems

[105,109-112]. However, other approaches include level-set [111], resistor network [110], and finite difference [107]. In [114], an online tool was developed that uses a combination of knowledge base, finite element analysis, and analytical correlations to predict effective thermal properties of a composite medium based on the input characteristics of the constituents. It is interesting to note that researchers often do not indicate the origin of the software used to perform their computational analyses, whether commercial, personal, open-source, or otherwise.

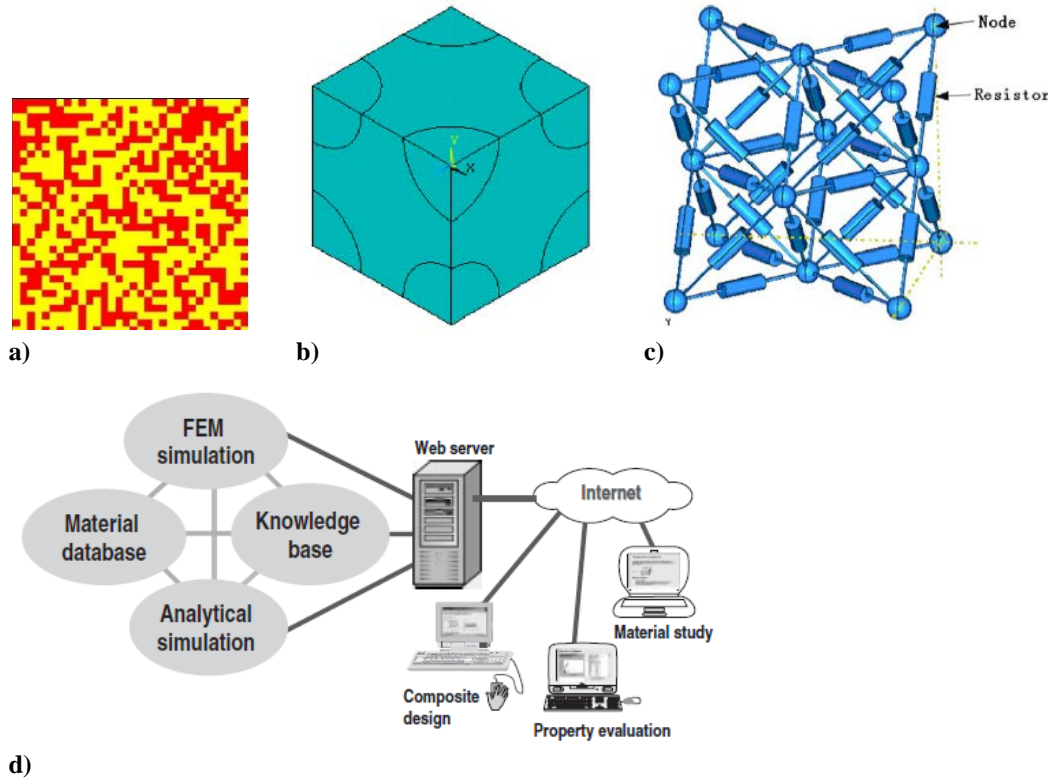


Figure 10. Example heterogeneous material computational analysis approaches including a) 2D multiscale fillers [108], b) 3D uniform filler arrangements [112], c) node-resistor networks [110], and d) hybrid tool analysis suite [114]

Although many practitioners do appear to qualitatively compare their computational results to some select set of empirical data, rigorous validation of results was not found to be

presented in the literature accompanying computational results. Furthermore, code and solution verification of any form appear to be generally absent from the body of literature on computational thermal analysis of heterogeneous materials, leaving some mystery as to the veracity and credibility of the employed computational methods.

6 A Note on Verification and Validation

Due to the lack of formal and/or rigorous V&V found in the literature supporting computational thermal analyses of heterogeneous material systems, a brief introduction to V&V is given here to guide future work and shed light on what may be an evasive topic to analysts and researchers. Figure 11 illustrates the two main aspects of V&V—namely verification (Figure 11a) and validation (Figure 11b)—and serves as a useful reference in understanding the process and purpose of V&V procedures.

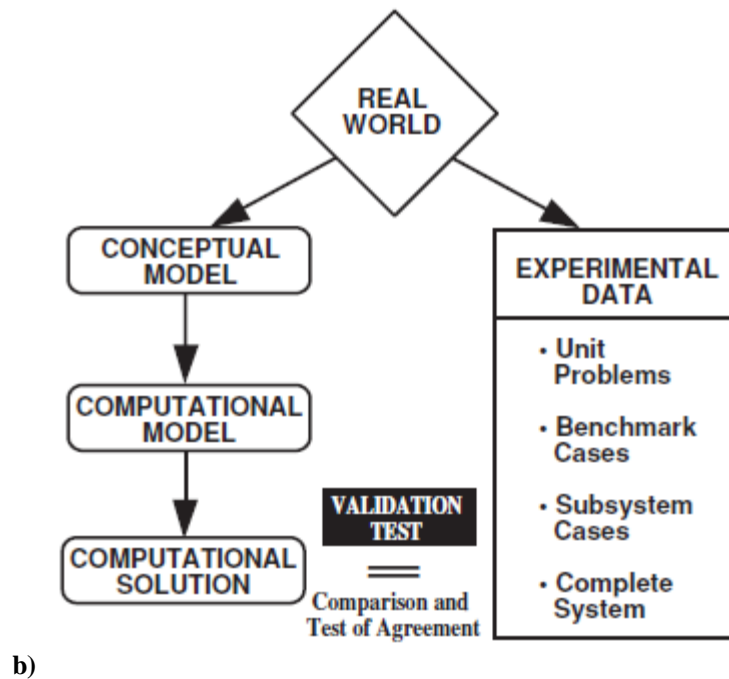
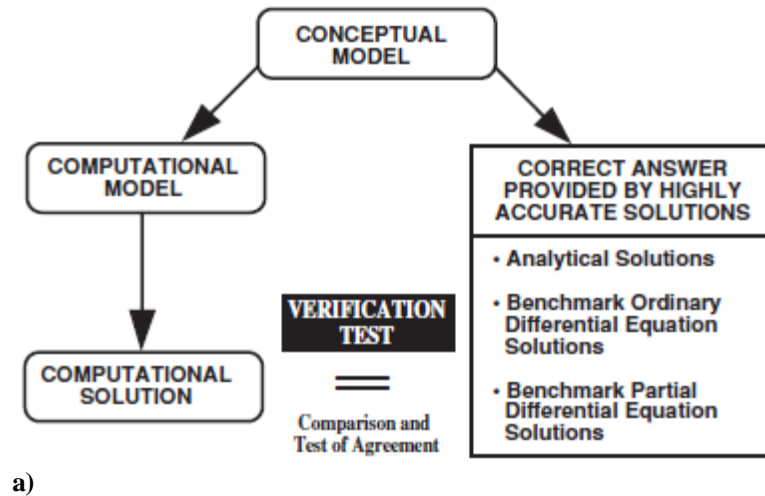


Figure 11. Flowcharts illustrating a) the verification process [115] and b) the validation process [115]

Verification is generally defined as the processes executed to assess the correctness of a numerical method in its ability to solve the defined mathematical problem. Verification can be broken into two steps: code verification and solution verification. Code verification is the process of assessing how well the implemented method (program, software, code, script, etc.) solves the governing mathematical equations—typically some set of partial differential

equations. Solution verification is the process of assessing how well the implemented method solves the mathematical problem in a defined computational system (i.e., the computational model's domain). Ultimately, verification is intended to be a process for quantifying the uncertainty in the numerical solution of the model and providing credibility evidence towards the confident use of the model for making predictions against real-world systems [116-118].

Validation is defined as the processes executed to assess how well a computational model predicts the real world. True validation necessarily uses empirical data measurements against which to compare and assess the performance of the computational model in predicting reality [116,118]. Validation is the end goal of V&V processes, but validation should be preceded by rigorous efforts to perform code and solution verification. It may be inferred that validation is not complete without verification.

The American Society of Mechanical Engineers (ASME) and the American Institute of Aeronautics and Astronautics (AIAA) have historically driven the development and call for implementation of V&V in computational analyses. ASME has many published standards and guides that describe—and guide by example—V&V procedures, most of which are targeted towards specific fields but are actually ubiquitously applicable [116,119,120]. AIAA also has a published guide to V&V related to computational fluid dynamics [115]. Extensive scholarly texts recognized as pinnacle guides to V&V include [117] and [118].

7 Conclusions

An overview discussion of the definition of heterogeneous materials and effective properties has been presented. Likewise, a description of pertinent thermal applications for heterogeneous material properties has been presented, showing that a pervasive need exists across many fields for characterizing and analyzing effective thermal responses of

heterogeneous media. A review of historical analytical correlations, empirical data measurement interests, and computational analysis efforts regarding effective thermal responses of heterogeneous materials has been presented. Finally, a brief discussion on the importance of V&V in computational analyses has been given, imploring that rigorous V&V should accompany development and presentation of computational analyses.

References

- 1 Kaviany, M., 1995, *Principles of Heat Transfer in Porous Media*, Springer, New York, NY.
- 2 Bergman, T. L., Incropera, F. P., and Lavine, A. S., 2011, *Fundamentals of Heat and Mass Transfer*, John Wiley & Sons, Hoboken, NJ.
- 3 Gu, S., Lu, T. J., Hass, D. D., and Wadley, H. N. G., 2001, "Thermal conductivity of zirconia coatings with zig-zag pore microstructures," *Acta Materialia*, **49**(13), pp. 2539-2547.
- 4 Flores Renteria, A., Saruhan, B., Schulz, U., Raetzer-Scheibe, H.-J., Haug, J., and Wiedenmann, A., 2006, "Effect of morphology on thermal conductivity of EB-PVD PYSZ TBCs," *Surface and Coatings Technology*, 201(6), pp. 2611-2620.
- 5 Zhu, W., Cai, X. N., Yang, L., Xia, J., Zhou, Y. C., and Pi, Z. P., 2019, "The evolution of pores in thermal barrier coatings under volcanic ash corrosion using X-ray computed tomography," *Surface and Coatings Technology*, 357, pp. 372-378.
- 6 Zhao, Z., Shang, H., Zou, G., Ren, H., Zhuang, W., Liu, L., and Zhou, Y. N., 2019, "A Predictive Model for Thermal Conductivity of Nano-Ag Sintered Interconnect for a SiC Die," *Journal of Electronic Materials*, 48, pp. 2811-2825.
- 7 Dinesh, B. V. S. and Bhattacharya, A., 2019, "Effect of foam geometry on heat absorption characteristics of PCM-metal foam composite thermal energy storage systems," *International Journal of Heat and Mass Transfer*, 134, pp. 866-883.
- 8 Jiji, L. M., 2009, *Heat Conduction*, Springer-Verlag Berlin Heidelberg.
- 9 Padture, N. P., Gell, M., and Jordan, E. H., 2002, "Thermal Barrier Coatings for Gas-Turbine Engine Applications," *Science*, **296**(5566), pp. 280-284.
- 10 Nayak, J. and Mahto, D., 2014, "Effect of Gas Turbine Inlet Temperature on Combined Cycle Performance," *International Conference on Recent Innovations in Engineering & Technology*.
- 11 Ibrahim, T. K. and Rahman, M. M., 2013, "Study on effective parameter of the triple-pressure reheat combined cycle performance," *Thermal Science*, **17**(2), pp. 497-508.

- 12 Fathi, N., McDaniel, P., Forsberg, C., and de Oliveira, C., 2018, "Power Cycle Assessment of Nuclear Systems, Providing Energy Storage for Low Carbon Grids," *Journal of Nuclear Engineering and Radiation Science*, **4**(2), 020911.
- 13 Hunter, I., Daleo, J., Wilson, J., and Ellison, K., 1999, "Analysis of Hot Section Failures on Gas Turbines in Process Plant Service," *Proceedings of the 28th Turbomachinery Symposium*, **28**, pp. 9-20.
- 14 Slaehnasab, B., Poursaeidi, E., Mortazavi, S. A., and Farokhian, G. H., 2016, "Hot corrosion failure in the first stage nozzle of a gas turbine engine," *Engineering Failure Analysis*, **60**, pp. 316-325.
- 15 Lai, G. Y., 2007, *High-Temperature Corrosion and Materials Applications*, ASM International, Novelty, OH.
- 16 Rao, A. D., 2012, *Combined Cycle Systems for Near-Zero Emission Power Generation*, Woodhead Publishing Limited, Cambridge, UK.
- 17 Ma, W., Li, X., Meng, X., Xue, Y., Bai, Y., Chen, W., and Dong, 2018, "Microstructure and Thermophysical Properties of SrZrO₃ Thermal Barrier Coating Prepared by Solution Precursor Plasma Spray," *Journal of Thermal Spray Technology*, **27**(7), pp. 1056-1063.
- 18 McCay, M. H., Hsu, P.-f., Croy, D. E., Moreno, D., and Zhang, M., 2017, "The Fabrication, High Heat Flux Testing, and Failure Analysis of Thermal Barrier Coatings for Power Generation Gas Turbines," *Turbo Expo: Power for Land, Sea, and Air*, **6**():V006T24A008.
- 19 Cernuschi, F., Bison, P., Mack, D. E., Merlini, M., Boldrini, S., Marchionna, S., Capelli, S., Concari, S., Famengo, A., Moscatelli, A., and Stamm, W., 2018, "Thermo-physical properties of as deposited and aged thermal barrier coatings (TBC) for gas turbines: State-of-the art and advanced TBCs," *Journal of the European Ceramic Society*, **38**(11), pp. 3945-3961.
- 20 Sahithi, M. S., Giridhara, G., and Kumar, R. S., 2016, "Development and analysis of thermal barrier coatings on gas turbine blades – A Review," *International Conference on Advanced Materials and Applications*, Bengaluru, Karnataka, India, pp. 2746-2751.
- 21 Hawkes, G. L., Sterbentz, J. W., and Pham, B., 2015, "Thermal Predictions of the AGR-2 Experiment with Variable Gas Gaps," *Nuclear Technology*, **190**(3), pp. 245-253.
- 22 Hawkes, G. L., Sterbentz, J. W., Maki, J. T., and Pham, B. T., 2017, "Thermal Predictions of the AGR-3/4 Experiment with Post Irradiation Examination Measured Time-Varying Gas Gaps," *Journal of Nuclear Engineering and Radiation Science*, **3**(), pp. 041007-1-041007-11.
- 23 Khattak, M. A., Borhana Omran, A. A., Kazi, S., Khan, M. S., Ali, H. M., Tariq, S. L., and Akram, M. A., 2019, "A Review of Failure Modes of Nuclear Fuel Cladding," *Journal of Engineering Science and Technology*, **14**(3), pp. 1520-1541.
- 24 Finsterle, S., Muller, R. A., Baltzer, R., Payer, J., and Rector, J. W., 2019, "Numerical Evaluation of Thermal Effects from Nuclear Waste Disposed in Horizontal Drillholes," *International High-Level Radioactive Waste Management Conference*.

- 25 Hobson, C., Taylor, R., and Ainscough, J. B., 1974, "Effect of porosity and stoichiometry on the thermal conductivity of uranium dioxide," *Journal of Physics D: Applied Physics*, **7**(7), 1003.
- 26 Lyons, M. F., Boyle, R. F., Davies, J. H., Hazel, V. E., and Rowland, T. C., 1972, "UO₂ properties affecting performance," *Nuclear Engineering and Design*, **21**(2), pp. 167-199.
- 27 Bakker, K., Kwast, H., and Cordfunke, E. H. P., 1995, "Determination of a porosity correction factor for the thermal conductivity of irradiated UO₂ fuel by means of the finite element method," *Journal of Nuclear Materials*, **226**(1), pp. 128-143.
- 28 Jin, E., Liu, C., and He, H., 2016, "The Influence of Microstructures on the Thermal Conductivity of Polycrystalline UO₂," *Proceedings of the ASME 2016 24th International Conference on Nuclear Engineering*, V001T02A028.
- 29 Newman, C., Hansen, G., and Gaston, D., 2009, "Three dimensional coupled simulation of thermomechanics, heat, and oxygen diffusion in UO₂ nuclear fuel rods," *Journal of Nuclear Materials*, **392**(1), pp. 6-15.
- 30 Ramirez, J. C., Stan, M., and Cristea, P., 2006, "Simulations of heat and oxygen diffusion in UO₂ nuclear fuel rods," *Journal of Nuclear Materials*, **359**(3), pp. 174-184.
- 31 Zohuri, B., and Nima Fathi. *Thermal-hydraulic analysis of nuclear reactors*. New York: Springer, 2015.
- 32 Rechard, R. P., Hadgu, T., Wang, Y., Sanchez, L. C., McDaniel, P., Skinner, C., and Fathi, N., 2017, "Technical Feasibility of Direct Disposal of Electrefiner Salt Waste," SAND2017-10554. Sandia National Laboratories, Albuquerque, NM.
- 33 Walker, C. T., Staicu, D., Sheindlin, M., Papaioannou, D., Goll, W., and Sontheimer, F., 2006, "On the thermal conductivity of UO₂ nuclear fuel at a high burn-up of around 100 MWd/kgHM," *Journal of Nuclear Materials*, **350**(1), pp. 19-39.
- 34 Kashiwagi, T., Grulke, E., Hilding, J., Groth, K., Harris, R., Butler, K., Shields, J., Kharchenko, S., and Douglas, J., 2004, "Thermal and flammability properties of polypropylene/carbon nanotube nanocomposites," *Polymer*, **45**(12), pp. 4227-4239.
- 35 Liu, S., Yan, H., Fang, Z., and Wang, H., 2014, "Effect of graphene nanosheets on morphology, thermal stability, and flame retardancy of epoxy resin," *Composites Science and Technology*, **90**, pp. 40-47.
- 36 Yang, S.-Y., Lin, W.-N., Huang, Y.-L., Tien, H.-W., Wang, J.-Y., Ma, C.-C. M., Li, S.-M., and Wang, Y.-S., 2011, "Synergetic effects of graphene platelets and carbon nanotubes on the mechanical and thermal properties of epoxy composites," *Carbon*, **49**(3), pp. 793-803.
- 37 Ramanathan, T., Abdala, A. A., Stankovich, S., Dikin, D. A., Herrera-Alonso, M., Piner, R. D., Adamson, D. H., Schniepp, H. C., Chen, X., Ruoff, R. S., Nguyen, S. T., Aksay, I. A., Prud'Homme, R. K., and Brinson, L. C., 2008, "Functionalized graphene sheets for polymer nanocomposites," *Nature Nanotechnology*, **3**, pp. 327-331.

- 38 Wu, Q., Zhu, W., Zhang, C., Liang, Z., and Wang, B., 2010, "Study of fire retardant behavior of carbon nanotube membranes and carbon nanofiber paper in carbon fiber reinforced epoxy composites," *Carbon*, **48**(6), pp. 1799-1806.
- 39 Knight, C. C., Ip, F., Zeng, C., Zhang, C., and Wang, B., 2012, "A highly efficient fire-retardant nanomaterial based on carbon nanotubes and magnesium hydroxide," *Fire and Materials*, **37**(2), pp. 91-99.
- 40 Zhou, Y., Liu, F., and Wang, H., 2017, "Novel Organic-inorganic Composites with High Thermal Conductivity for Electronic Packaging Applications: A Key Issue Review," *Polymer Composites*, **38**(4), pp. 803-813.
- 41 Gilmore, D. G., 2002, "Spacecraft Thermal Control Handbook," The Aerospace Press and American Institute of Aeronautics and Astronautics, Inc.
- 42 Taft, B., "Space Vehicles Directorate Thermal Systems Roadmap," *Proceedings of the Spacecraft Thermal Control Workshop 2017*, The Aerospace Corporation, El Segundo, CA, 2017.
- 43 Maxwell, J. C., 1873, *A Treatise on Electricity and Magnetism*, **1**, Oxford at the Clarendon Press.
- 44 Burger, H. C., 1915, "Das Iertvermogen verdumter mischkristallfreier Ionsungen," *Phys. Zs* **20**, pp. 73-76.
- 45 Landauer, R., 1952, "The Electrical Resistance of Binary Metallic Mixtures," *Journal of Applied Physics*, **23**(7), pp. 779-184.
- 46 Procter, P. and Solc, J., 1999, "Improved thermal conductivity in microelectronic encapsulants," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, **14**(4), pp. 708-713.
- 47 Lee, G.-W., Park, M., Kim, J., Lee, J. I., and Yoon, H. G., 2006, "Enhanced thermal conductivity of polymer composites filled with hybrid filler," *Composite Part A: Applied Science and Manufacturing*, **37**(5), pp. 727-734.
- 48 Sato, K., Tominaga, Y., and Imai, Y., 2020, "Nanocelluloses and Related Materials Applicable in Thermal Management of Electronic Devices: A Review," *Nanomaterials*, **10**(448).
- 49 Glass, D. E., Dirling, R., Croop, H., Fry, T. J., and Frank, G. J., 2006, "Materials Development for Hypersonic Flight Vehicles," *Proceedings of the 14th AIAA/AHI Space Planes and Hypersonic Systems and Technologies Conference*, AIAA 2006-8122.
- 50 Tjong, W. C., 2007, "High Temperature Materials for Hypersonic Flight Vehicles," *Journal of The University of New South Wales at the Australian Defence Force Academy*, **1**(1).
- 51 Kasen, S. D., 2013, "Thermal Management at Hypersonic Leading Edges," Dissertation, University of Virginia.

- 52 Woodside, W., and Messmer, J. H., 1961, "Thermal Conductivity of Porous Media. I. Unconsolidated Sands," *Journal of Applied Physics*, **32**(9), pp. 1688–1699.
- 53 Woodside, W., and Messmer, J. H., 1961, "Thermal Conductivity of Porous Media. II. Consolidated Rocks," *Journal of Applied Physics*, **32**(9), pp. 1699–1706.
- 54 Kiyohashi, H., Sasaki, S., and Masuda, H., 2003, "Effective Thermal Conductivity of Silica Sand as a Filling Material for Crevices," *High-Temperatures-High Pressures*, **35**, pp. 179-192.
- 55 Wallen, B. M., Smits, K. M., Sakaki, T., Howington, S. E., T.K.K., C. D., 2016, "Thermal Conductivity of Binary Sand Mixtures Evaluated through Full Water Content Range," *Soil Science Society of America Journal*, 80, pp.592 -603.
- 56 Gobetz, Z., Rowen, A. Dickman, C., Meinert, K., and Martukanitz, R., 2016, "Utilization of Additive Manufacturing for Aerospace Exchangers," Office of Naval Research, Final Report.
- 57 Zhang, S., Lane, B., Whiting, J., and Chou, K., 2018, "An Investigation into Metallic Powder Thermal Conductivity in Laser Powder Bed Fusion Additive Manufacturing," *Proceedings of the 19th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference*, pp. 1796-1807.
- 58 Ibrahim, Y., Elkholy, A., Schofield, J. S., Melenka, G. W., and Kempers, R., 2020, "Effective Thermal Conductivity of 3D-printed Continuous Fiber Polymer Composites," *Advance Manufacturing: Polymer & Composites Science*, **6**(1), pp. 17-28.
- 59 Danes, F., Garnier, B., and Dupuis, T., 2003, "Predicting, Measuring, and Tailoring the Transverse Thermal Conductivity of Composites from Polymer Matrix and Metal Filler," *International Journal of Thermophysics*, **24**(3), pp. 727-734.
- 60 Zhao, J.-J., Duan, Y-Y., Wang X.-D., 2012, "Experimental and analytical analyses of the thermal conductivities and high-temperature characteristics of silica aerogels based on microstructures," *Journal of Physics D: Applied Physics*, 46, 015304.
- 61 Irick, K. W., 2017, "Effective Thermal Resistance Comparison of Aerogel and Multi-Layer Insulation as Radiative Barriers Using the Single-Sided Guarded Hot Plate Method," *Frontiers in Heat and Mass Transfer*, **8**(2).
- 62 Wu, S., Li, T., Tong, Z., Chao, J., Zhai, T., Xu, J., Yan, T., Wu, M., Xu, Z., Bao, H., Deng, T., and Wang, R., 2019, "High-Performance Thermally Conductive Phase Change Composites by Large-Size Oriented Graphite Sheets for Scalable Thermal Energy Harvesting." *Advanced Materials*, **31**(49), 1905099.
- 63 He, Y.-L. and Xie, T., 2015, "Advances of thermal conductivity models of nanoscale silica aerogel insulation material," *Applied Thermal Engineering*, (81), pp. 28-50.
- 64 Rayleigh, L., 1892, "On the Influence is Obstacles Arranged in Rectangular Order upon the Properties of a Medium," *Philosophical Magazine Series 5*, **34**(211), pp. 481-502.

- 65 Hamilton, R. L. and Crosser, O. K, 1962, "Thermal conductivity of heterogeneous two component systems," *Industrial and Engineering Chemistry Fundamentals*, **1**(3), pp. 187-191.
- 66 Lewis, T.B., Nielsen, L. E., 1970, "Dynamic mechanical properties of particulate-filled composites," *Journal of Applied Polymeric Science*, **14**(6), pp. 1449-1471.
- 67 Nielsen, L. E., 1970, "Generalized Equation for the Elastic Moduli of Composite Materials," *Journal of Applied Physics*, **41**(11), pp. 4626-4627.
- 68 Nielsen, L. E., 1974, "The Thermal and Electrical Conductivity of Two-Phase Systems," *Industrial and Engineering Chemistry Fundamentals*, **13**(1), pp. 17-20.
- 69 Chatterjee, A., Verma, R., Umashankar, H.P., Kasthuriengan, S., Shivaprakash, N.C., and Behera, U., 2019, "Heat conduction model based on percolation theory for thermal conductivity of composites with high volume fraction of filler in base matrix," *International Journal of Thermal Sciences*, **136**, pp. 389-395.
- 70 Devupra, A., Phelan, P. E., and Prasher, R. S., 2000 "Percolation theory applied to the analysis of thermal interface materials in flip-chip technology," *The Seventh Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, Las Vegas, NV, pp. 21-28.
- 71 Devupra, A., Phelan, P., and Prasher, R. S., 2001, "Size effects on the thermal conductivity of polymers laden with highly conductive filler particles," *Microscale Thermophysical Engineering*, **5**(3), pp. 177-189.
- 72 Xue, Q., 2003, "A percolation model of metal-insulator composites," *Physica B: Condensed Matter*, **325**, pp. 195-198.
- 73 Zhang, G., Xia, Y., Wang, H., Tao, Y., Tao, G., Tu, S., and Wu, H., 2009, "A percolation model of thermal conductivity for filled polymer composites," *Journal of Composite Materials*, **44**(8), pp. 963-970.
- 74 Krischer, O. and Kroell, K., 1956, "Die wissenschaftlichen Grundlagen der Trocknungstechnik," *Trocknungstechnik*, **1**, p. 48.
- 75 Yagi, S. and Kunii, D., 1957, "Studies on effective thermal conductivity in packed bed," *American Institute of Chemical Engineers Journal* **3**(3), pp. 373-381.
- 76 Nan, C. W., Birringer, R., Clarke, D. R., and Gleiter, H., 1997, "Effective thermal conductivity of particulate composites with interfacial thermal resistance," *Journal of Applied Physics*, **10**, pp. 6692-6699.
- 77 Every, A. G., Tzou, Y., Hasselman, D. P. H., and Ray, R., 1992, "The effect of particle size on the thermal conductivity of ZnS/diamond composites," *Acta Metallurgica et Materialia*, **40**, pp. 123-129.
- 78 Torquato, S. and Rintoul, M. D., 1995, "Effect of interface on the properties of composite media," *Physical Review Letters*, **75**, pp. 4067-4070.

- 79 Cheng, S. C. and Vachon, R. I., 1970, "A technique for predicting the thermal conductivity of suspensions, emulsions and porous materials," *International Journal of Heat and Mass Transfer*, **13**(3), pp. 537-546.
- 80 Agari, Y., Ueda A., Omura Y., and Nagai S., 1997, "Thermal diffusivity and conductivity of PMMA-PC blends," *Polymer*, **38**, pp. 801-807.
- 81 de Vries, D. A., 1952, *Mededlingen van de Landbouwhogeschool te Wageningen*.
- 82 Kunii, D. and Smith, J. M., 1960, "Heat transfer characteristics of porous rocks," *American Institute of Chemical Engineers Journal*, **6**, pp. 71-77.
- 83 Schlünder, E. U., 1958, "Wärme-und Stoffübertragung zwischen durchströmten Schüttungen und darin eingebetteten Einzelkörpern," *Chemie-Ing.-Techn.*, **38**, pp. 967-979.
- 84 Zehner, P. and Schlünder, E. U., 1970, "Wärmeleitfähigkeit von Schüttungen bei mäßigen Temperaturen," *Chemie-Ing.-Techn.*, **42**, pp. 933-941.
- 85 Krupiczka, R., 1967, "Analysis of thermal conductivity in granular materials," *International Chemical Engineering*, **7**, pp. 122-144.
- 86 Pietrak, K. and Wiśniewski, T. S., 2015, "A Review of Models for Effective Thermal Conductivity of Composite Materials," *Journal of Power Technologies* **95**(1), pp. 14-24.
- 87 Ebadi-Dehaghani, H. and Nazempour, M., 2012, "Thermal Conductivity of Nanoparticles Filled Polymers," *Smart Nanoparticles Technology*, IntechOpen.
- 88 Kandula, M., 2011, "On the Effective Thermal Conductivity of Porous Packed Beds with Uniform Spherical Particles," *Journal of Porous Media*, **14**(10), pp. 919-926.
- 89 Hahne, E., Song, Y. W., and Gross, U., 1991, "Measurements of Thermal Conductivity in Porous Media," *Convective Heat and Mass Transfer in Porous Media*, NATO ASI Series, **196**.
- 90 Smith, D. S., Alzina, A., Bourret, J., Nait-Ali, B., Pennec, F., Tessier-Doyen, N., Otsu, K., Matsubara, H., Elser, P., and Gonzenbach, U. T., 2013, "Thermal Conductivity of Porous Materials," *Journal of Materials Research*, **28**(17), pp. 2260-2272.
- 91 Kamseu, E., Nait-Ali, B., Bignozzi, M.C., Leonelli, C., Rossignol, S., and Smith, D. S., 2012, "Bulk Composition and Microstructure Dependence of effective thermal conductivity of porous inorganic polymer cements" *Journal of the European Ceramic Society*, **32**, pp. 1593-1603.
- 92 Edrisi, S., Bidhendi, N. K., and Haghight, M., 2017, "A new approach to modeling the effective thermal conductivity of ceramics porous media using a generalized self-consistent method," *Heat Mass Transfer*, **53**(1), pp. 321-330.
- 93 Bouguerra, A., Aït-Mokhtar, A., Amiri, O., and Dip, M. B., 2001, "Measurement of Thermal Conductivity, Thermal Diffusivity, and Heat Capacity of Highly Porous Building Materials Using Transient Plane Source Technique," *International Communications in Heat and Mass Transfer*, **28**(8), pp. 1065-1078.

- 94 Nait-Ali, B., Haberkro, K., Vesteghem, H., Absi, J., and Smith, D. S., 2006, "Thermal Conductivity of Highly Porous Zirconia," *Journal of the European Ceramic Society*, **26**(16), pp. 3567-3574.
- 95 Liu, J., Gan, D., Hu, C., Kiene, M., Ho, P. S., Volksen, W., and Miller, R. D., 2002, "Porosity Effect on the Dielectric Constant and Thermomechanical Properties of Organosilicate Films," *Applied Physics Letters*, **81**(22), pp. 4180-4182.
- 96 Hu, C., Morgan, M., Ho, P. S., Jain, A., Gill, W. N., Plawsky, J. L., and Wayner, Jr., P. C., "Thermal conductivity study of porous low-k dielectric materials," *Applied Physics Letters*, **77**(1), pp. 145-147.
- 97 Jain, A., Rogojevic, S., Ponoth, S., Gill, W. N., Plawsky, J. L., Simonyi, E., Chen, S.-T., and Ho, P. S., 2002, "Processing dependent thermal conductivity of nanoporous silica xerogel films," *Journal of Applied Physics*, **91**(5), pp. 3275-3281.
- 98 Prajapati, H., Ravoori, D., Woods, R.L., and Jain, A. 2018, "Measurement of Anisotropic Thermal Conductivity and Inter-Layer Thermal Contact Resistance in Polymer Fused Deposition Modeling (FDM)," *Additive Manufacturing* **21**, pp. 84-90.
- 99 Flaata, T., Michna, G.J., and Letcher, T., 2017, "Thermal Conductivity Testing Apparatus for 3d Printed Materials," *Proceedings of the ASME 2017 Summer Heat Transfer Conference*, HT2017-4856, Bellevue, WA.
- 100 Rivière, L., Lonjon, A., Dantras, E., Lacabanne, C., Olivier, P., and Gleizes, N., 2016, "Silver Fillers Aspect Ratio Influence on Electrical and Thermal Conductivity in PEEK/Ag Nano-Composites," *European Polymer Journal*, **85**, pp. 115-125.
- 101 Lin, F., Bhatia, G. S., and Ford, J. D., 1993, "Thermal Conductivities of Powder-Filled Epoxy Resins," *Journal of Applied Polymer Science*, **49**, pp. 1901-1908.
- 102 Lu, Y., Lu, S., Horton, R., and Ren, T., 2014, "An empirical model for estimating soil thermal conductivity from texture, water content, and bulk density," *Soil Science Society of America Journal*, **78**, pp. 1859–1868.
- 103 Jin, H., Wang, Y., Zheng, Q., Liu, H., and Chadwick, E. A., 2017, "Experimental Study and Modelling of the Thermal Conductivity of Sandy Soils of Different Porosities and Water Contents," *Journal of Applied Sciences*, **7**(119).
- 104 Rohatgi, A., 2020, "WebPlotDigitizer," Version 4.3, <https://automeris.io/WebPlotDigitizer>.
- 105 Davis, L. C. and Artz, B. E., 1995, "Thermal conductivity of metal-matrix composites," *Journal of Applied Physics*, **77**, pp. 4954.
- 106 Jopek, H. and Strek, T., 2011, "Optimization of the Effective Thermal Conductivity of a Composite," *Convection and Conduction Heat Transfer*, pp. 197-214.
- 107 Ganapathy, D., Singh, K., Phelan, P. E., and Prasher, R., 2005, "An effective unit cell approach to compute the thermal conductivity of composites with cylindrical particles," *Journal of Heat Transfer*, **127**(6), pp. 553-559.

- 108 Phinney, L. M., Erikson, W. W., and Lechman, J. B., 2014, "Uncertainty Quantification for Multiscale Thermal Transport Simulations," *Proceedings of the 11th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, Atlanta, GA.
- 109 Matt, C. F. and Cruz, M. E., 2008, "Effective thermal conductivity of composite materials with 3-d microstructures and interfacial thermal resistance," *Numerical Heat Transfer, Part A*, 53, pp. 577-604.
- 110 Yue, C., Zhang, Y., Hu, Z., Liu, J., and Cheng, Z., 2010, "Modeling of the effective thermal conductivity of composite materials with FEM based on resistor networks approach," *Microsystem Technologies*, 16, pp. 633-639..
- 111 Yvonnet, J., He, Q.-C., Toulemonde, C., 2008, "Numerical modelling of the effective conductivities of composites with arbitrarily shaped inclusions and highly conducting interface," *Composites Science and Technology*, 68, pp. 2818-2825.
- 112 Nayak, R., Tarkes, D. P., and Satapathy, A., 2010, "A computational and experimental investigation on thermal conductivity of particle reinforced epoxy composites," *Computational Materials Science*, 48(3), pp. 576-581.
- 113 Moghaddam, H. A. and Mertiny, P., 2018, "Stochastic Finite Element Analysis Framework for Modeling Thermal Conductivity of Particulate Modified Polymer Composites," *Results in Physics*, 11, pp. 905-914.
- 114 Xu, Y., Yamazaki, M., Wang, H., and Yagi, K., 2006, "Development of an internet system for composite design and thermophysical property prediction," *Materials Transactions*, 47(8), pp. 1882-1885.
- 115 American Institute of Aeronautics and Astronautics, 2002, *G-077-1998(2002) Guide for the Verification and Validation of Computational Fluid Dynamics Simulations*.
- 116 The American Society of Mechanical Engineers, 2009, *Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer*, ASME V&V 20-2009, ASME, New York, NY.
- 117 Roache, P. J., 2009, *Fundamentals of Verification and Validation*, Hermosa Publishers, Socorro, NM.
- 118 Oberkampf, W. L. and Roy, C. J., 2010, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge, UK.
- 119 The American Society of Mechanical Engineers, 2007, *VV10-2006 Guide for Verification and Validation in Computational Solid Mechanics*.
- 120 The American Society of Mechanical Engineers, 2018, *VV40-2018 Assessing Credibility of Computational Modeling through Verification and Validation: Application to Medical Devices*.

CHAPTER 2: COMPUTATIONAL EVALUATION OF THERMAL RESPONSE OF OPEN-CELL FOAM WITH CIRCULAR PORE²

Abstract

The evaluation of effective material properties in heterogeneous materials (e.g., composites or multicomponent structures) critically relevant to a wide spectrum of applications, including nuclear power, electronic packaging, flame retardants, hypersonics, and gas turbine power. The work described in this paper is centered around the numerical assessment of the thermal behavior of porous materials obtained from finite element thermal modeling and simulation. Two-dimensional, steady state analyses were performed on unit cells with centered, circular pores using a second order accurate Galerkin finite element method (FEM). The effective thermal conductivities of the porous systems were examined, encompassing a range of porosities from 4.9% to 60.1%. The geometries of the models were generated based on ordered circular pores for each modeled porosity level. The system response quantity (SRQ) under investigation was the dimensionless effective thermal conductivity across the unit cell. The dimensionless effective thermal conductivity was compared across all simulated cases, producing a trend between porosity and effective thermal conductivity. In the presented investigation, the method of manufactured solutions (MMS) was used to perform code verification, and the grid convergence index (GCI) was employed to estimate discretization uncertainty as solution verification. Code verification concluded an

² At the time of compiling this dissertation, this chapter was under review for publication in the ASME Journal of Verification and Validation under the title “Computational Evaluation of Thermal Response of Open-Cell Foam with Circular Pore.” The content in this chapter was reproduced directly from the journal article, differing only in style formatting. A less comprehensive version of this chapter was published in *Proceedings of the ASME 2018 Verification and Validation Symposium* under the title “Thermal Response of Open-Cell Porous Materials: A Numerical Study and Model Assessment.”

approximately second order accurate Galerkin FEM solver. It was found that the introduction of porosity to the unit cell material structure reduces effective thermal conductivity, as anticipated. Numerical results obtained in this study are compared to an analytical solution and to a sample of empirical data. This approach can be readily generalized to study a wide variety of porous solids from ranging from structures at the nano-scale—such as nano carbon tubes—to structures at macro-level scales—such as geological features.

Nomenclature

a	= apparent order of accuracy
A	= area, m ²
α	= porosity
Γ	= computational domain boundary
ε	= error
FS	= factor of safety
g	= convergent behavior function
\mathbf{G}	= conductance matrix
GCI	= grid convergence index
η	= dimensionless temperature
h	= mesh number
H	= characteristic mesh size, m
k	= thermal conductivity, W/m-K
\mathbf{K}	= thermal conductivity matrix
L	= cell length and width, m
\mathbf{n}	= boundary normal vector

N	= total count
N	= shape function
p	= order of accuracy
\mathbf{P}	= vertex heat load vector
q	= heat flux, W/m ²
\mathbf{q}	= heat flux vector
Q	= volumetric heat generation, W/m ³
R	= void radius, m
ρ	= energy balance residual, W
$\boldsymbol{\rho}$	= energy balance residual vector
s	= unit direction sign
T	= temperature, °C
\mathbf{T}	= triangle vertex temperature vector
U	= approximate numerical uncertainty
w	= interpolation weight
x	= x-coordinate, m
y	= y-coordinate, m
ω	= relaxation coefficient
Ω	= computational domain

Subscripts

a	= approximate
b	= bulk
eff	= effective

<i>ext</i>	= extrapolated
<i>f</i>	= formal
<i>h</i>	= mesh number
<i>i</i>	= index
<i>j</i>	= index
<i>m</i>	= modified
<i>MMS</i>	= manufactured solution
<i>n</i>	= normal
<i>new</i>	= updated
<i>num</i>	= numerical
<i>s</i>	= structure
<i>t</i>	= triangle
<i>tar</i>	= update target
<i>v</i>	= void or vertex
<i>x</i>	= x-direction
<i>y</i>	= y-direction
<i>1</i>	= material 1
<i>2</i>	= material 2

Superscripts

*	= dimensionless
---	-----------------

1 Introduction

Recent developments in technology, from the petroleum industry to geo- and nano-sciences, has created an exponentially increasing demand for investigating the physical

properties of composite materials [1]. Porous media are two-phase composites comprising solid (matrix) and fluid (void) phases. The void may be occupied by either a gas or a liquid, or the void may be evacuated. When all three material phases coexist in a medium it is called an unsaturated porous medium. On the other hand, when only one of the phases is involved it is said to be saturated [2]. The volume fraction of pores is calculated by dividing the total void volume by the total volume of the media, i.e. the combined matrix and pore volume. A pore might be connected to multiple pores (interconnected), or only connected to one other pore (dead end), or not have any connection to other pores (isolated) [1]. There has been recent interest toward predicting the behavior of porous media facilitating heat and mass transfer, mainly because of their key role in material design. In particular, the effective thermal conductivity has been the focus of many studies, both experimentally and analytically [3-8]. The analytical solution to the effective thermal conductivity of a two-phased composite material system is generally described by an equivalent model of parallel, series or combination of parallel and series arrangements. Heat conduction within porous media depends on the thermal conductivity of each constituent phase as well as the microstructure of the matrix [1].

A prominent application where porous materials are found is in flame retardant material development. Carbon-nanotube-based materials can exhibit excellent flame retardant properties, potentially serving as safety and protective mechanisms in a wide variety of applications. In carbon nanofiber composites, the carbon nanotubes can act as filler materials in a bulk matrix, comprising a heterogeneous material structure, or can act as porous sheets protecting polymer surfaces. In [9], an increase in thermal conductivity—implying increased heat dispersion—and heat rejection under incident radiant heat with higher concentrations of

carbon nanotubes in a composite medium were shown. Likewise, in [10-12], mechanical-thermal advantages to doping epoxy resins with graphene nanosheets was shown. In contrast, the thermal protection capabilities of porous carbon nanotube membranes were studied in [13] and [14]. The study in [13] used these membranes on the surface of epoxy carbon fiber composites, where the pore size of the membrane was measured and used as a metric in characterizing the thermal performance. The contrast of developing multi-component material systems against single-material porous systems using the same material sheds light on the spectrum of interest in heterogeneous material evaluation. Understanding and characterizing the effective thermal properties of these types of heterogeneous, porous materials can be critical in the development of next-generation flame retardants.

Here we evaluate the thermal behavior of porous materials obtained from thermal modeling and simulation by model verification. Specifically, the system response quantity (SRQ) under investigation is the dimensionless effective thermal conductivity, k^* , of a porous unit cell, modeled using a two-dimensional finite element method (FEM) approach. The remainder of this paper will first provide a brief background of porous media analysis, then lay out the modeled system, and then the discretization of the system and the governing partial differential equation (PDE) will be described. A brief description of the numerical PDE solver will be given, followed by code and solution verification. Lastly, a short sensitivity analysis is presented, and conclusions are drawn from the preceding processes.

2 Background

In 1952, Landauer presented an analytical solution to correlate effective electrical conductivity across a two-phase medium, where one phase takes the form of spherical pores [15]. This approach, based on theoretical assumptions, can be extended to correlate effective

thermal conductivity, as was done in [16], where Landauer's analytical formula is expressed as

$$k_{eff} = \frac{1}{4} [(3\alpha - 1)k_2 + (2 - 3\alpha)k_1 + \dots \frac{1}{([(3\alpha - 1)k_2 + (2 - 3\alpha)k_1]^2 + 8k_1k_2)^{\frac{1}{2}}}] \quad (1)$$

Here, k_{eff} , is the effective thermal conductivity across the porous medium with k_1 and k_2 representing the matrix and pore material thermal conductivities, respectively. Also, α denotes the volume fraction (or porosity) of the pore material with respect to the total composite material volume. However, an analytical approach is not the only means for predicting a material's true effective thermal conductivity.

A large number of other works and investigations have been presented in the literature to provide theoretical and analytical correlations to describe the effective thermal conductivity in heterogeneous materials. Such works consider percolation theory (the concept non-separated filler particles) [17-21], component stratification [22,23], and interface resistance [23-25]. More developed correlations can be found in the literature based on specific types of applications [26-34].

Computational methods and numerical analysis have become indispensable tools in predicting the behavior of engineered systems [35]. The accuracy of numerical methods varies significantly for different applications, for instance, from a very accurate prediction of a computer processor function [36] to less accurate prediction of the oceanic behavior. In many cases, the complexity of physical systems precludes any analytic solutions so simulations actually become a necessity. FEM, finite difference method, and lattice Boltzmann method are amongst the most commonly used numerical techniques [37-44]. Wang et al. [37] developed a

three-dimensional mesoscopic method to predict k_{eff} of multiphase random porous media. In that work, a lattice Boltzmann method was used to solve the energy transport equations for multiphase conjugate heat transfer through a porous structure. They concluded that the effective thermal conductivity of three-dimensional porous structures varies with the number of cells in the third dimension. Bakker [38] used FEM to compute k_{eff} of complex two-phased microstructures. By accounting for the influence of shape, orientation, and distribution of the dispersed phase (porosity or inclusions), he derived a relation to convert the two-dimensional result to the three-dimensional conductivity. Wang et al. [37] believed that this method of solving PDEs could be inexpedient when the microstructure is overly complex and demands too much computational cost, especially when the fluid–solid conjugate problem is considered. Fiedler et al. [39] applied Lattice Monte Carlo (LMC) and Finite Element (FE) analyses to calculate k_{eff} of sintered metallic hollow spheres structures.

3 System Description

The modeled system in this work is a unit cell nano-porous medium, as illustrated in Figure 1. The unit cell is of equal unit length and width, L , with a circular void centered in the cell of radius, R , oriented in the rectangular x - y plane. The thermal conductivity of the bulk continuum material is denoted by k_b . Five boundaries are defined in this system, labeled with 1 through 5 in the figure. Boundaries 1 and 3 are specified as constant Dirichlet boundaries, T_1 and T_3 , given to be 50 °C and 100 °C, respectively. Boundaries 2, 4, and 5 are given as Neumann adiabatic boundaries (0 °C/m gradient normal to boundary). The fixed temperature gradient results in an average boundary heat flux, q_3 , to be described later.

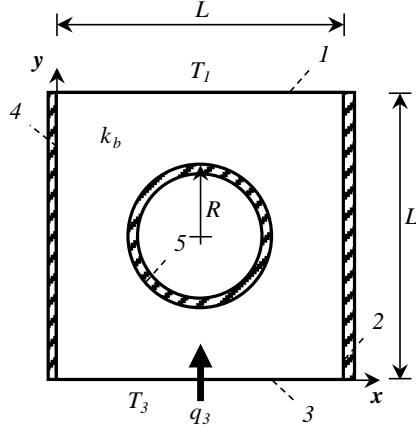


Figure 1. Unit cell nano-porous structure

The characteristic parameter of this system is the structure porosity, α , defined by

$$\alpha = A_v/A_s = \pi R^2/L^2, \quad (2)$$

where A_v and A_s are the void area and the bulk structure area, respectively. However, the SRQ for this study, k^* , is determined by the ratio of k_{eff} , defined as

$$k_{eff} = q_3 L / (T_3 - T_1), \quad (3)$$

to the bulk continuum thermal conductivity. Thus,

$$k^* = k_{eff}/k_b. \quad (4)$$

4 Numerical Analysis

4.1 Numerical Approach

The domain shown in Figure 1 is discretized in a series of unstructured triangular meshes. The mesh number, h , is used to denote the level of mesh refinement, with $h=1$ being the finest mesh and $h=5$ being the coarsest. Samples of systematic domain meshes with refinement are

shown in Figure 2 and Figure 3 for a representative quarter of the entire discretized domain, increasing in mesh refinement and decreasing sequentially in mesh number from $h=5$ in a) to $h=1$ e) for α of 19.6% and 30.7%, respectively. Figure 4 and Figure 5 show the coarsest and finest meshes, respectively, for each of the 11 modeled porosities, from a) to k): 4.9%, 7.7%, 11.0%, 15.0%, 19.6%, 24.9%, 30.7%, 37.1%, 44.2%, 51.8%, and 60.1%. A comparison of Figure 4 and Figure 5 shows that the progression of mesh refinement for different porosity models yields similar meshes at each level, suggesting that the domain geometry should have little effect on the system-to-system evaluation results. For the coarse meshes displayed in Figure 4, note the diminishing quality in triangle elements—especially the skewness of elements between the edges of the system boundary and the pore where the pore boundary is closest to the system edge—as the porosity of the domain increases. The issues due to mesh quality should resolve with mesh refinement, where the quality of the mesh is visually improved in the corresponding fine meshes shown in Figure 5.

Figure 6 notionally illustrates the triangle element structure formed by three vertices in a mesh, where t is the triangle, $(x_{t,i}, y_{t,i})$, are the x and y coordinates of the i^{th} vertex in element t , $T_{t,i}$, is the temperature at vertex i in triangle t , and $A_{t,i}$ is the area of triangle t that is associated with vertex i . Each $A_{t,i}$ is formed by connecting the centroid of the triangle to the midpoint of each triangle edge. Thus, the total area of element t , A_t , is described by Equation (5).

$$A_t = A_{t,1} + A_{t,2} + A_{t,3} \quad (5)$$

The characteristic mesh size, H_h , of each discretized domain is determined by Equation (6), where N_t is the total number of triangles in the mesh, and h is the mesh number in the series of refined meshes 1 through 5.

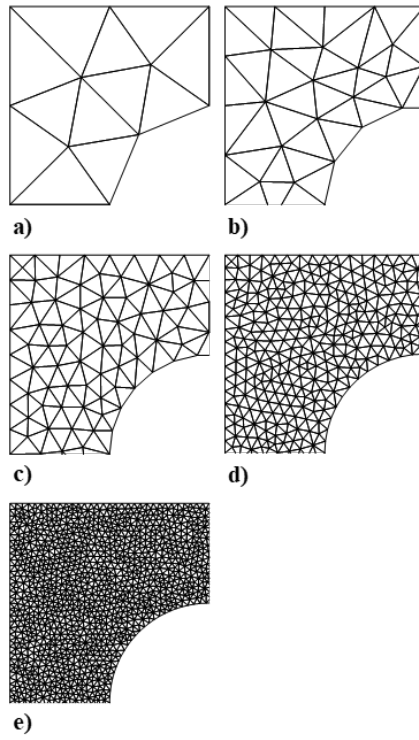


Figure 2. Systematic mesh refinement for $\alpha=19.6\%$

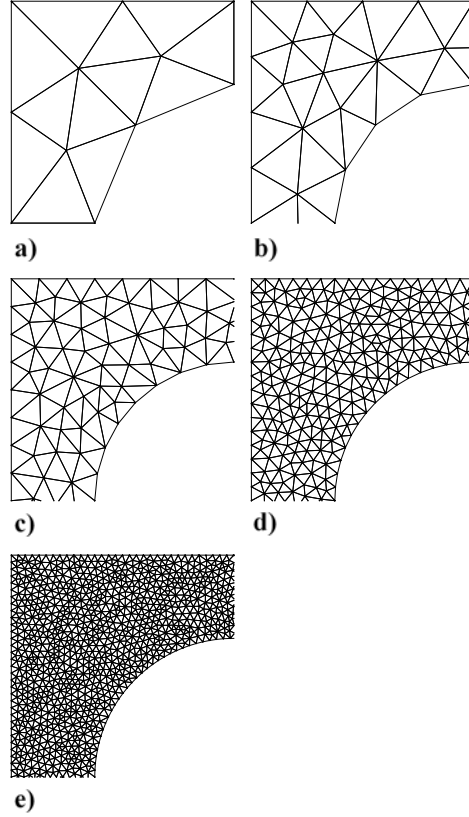


Figure 3. Systematic mesh refinement for $\alpha=30.7\%$

$$H_h = \sqrt{(1/N_t) \sum_{i=1}^{N_t} A_i} \quad (6)$$

A series of five meshes was used for each of the 11 porosities modeled, as given in Table 1, where the characteristic mesh sizes are summarized. Note that the mesh size is approximately halved between each successive mesh, facilitating the verification methods used in this work. In Table 1, the sub-figures of Figure 5 are indicated that correspond to the finest mesh of each porosity value.

Table 1. Characteristic mesh size for mesh refinement

Figure 5	α %	H				
		5	4	3	2	1
a	4.9	0.1139	0.0706	0.0366	0.0189	0.0096
b	7.7	0.1317	0.0727	0.0383	0.0190	0.0095
c	11.0	0.1297	0.0735	0.0377	0.0193	0.0096
d	15.0	0.1444	0.0741	0.0377	0.0194	0.0097
e	19.6	0.1310	0.0745	0.0388	0.0192	0.0096
f	24.9	0.1272	0.0769	0.0380	0.0189	0.0096
g	30.7	0.1228	0.0701	0.0387	0.0192	0.0097
h	37.1	0.1247	0.0711	0.0388	0.0194	0.0097
i	44.2	0.1182	0.0698	0.0389	0.0191	0.0095
j	51.8	0.0985	0.0692	0.0395	0.0192	0.0097
k	60.1	0.0791	0.0658	0.0365	0.0192	0.0096

4.1.1 PDE Discretization

As previously stated, the described physical discretization is employed to enable the numerical solution of the temperature distribution, T , across a domain, Ω , from the generalized steady state two-dimensional heat equation described by the PDE given in Equation (7), where Q is volumetric heat generation and k_x and k_y are the thermal conductivities in the x and y directions, respectively. Note that Equation (7) is valid for both orthotropic materials and materials with temperature-dependent thermal conductivities. In this work, however, the problem is simplified to an isotropic material with a constant thermal conductivity.

$$\frac{\partial}{\partial x} k_x \frac{\partial T}{\partial x} + \frac{\partial}{\partial y} k_y \frac{\partial T}{\partial y} + Q = 0 \quad (7)$$

The thermal conductivities and temperature gradients can be cast in matrix form as

$$\mathbf{K} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \text{ and } \nabla T = \left\{ \frac{\partial T}{\partial x}, \frac{\partial T}{\partial y} \right\}. \quad (8)$$

In this specific application, both k_x and k_y are equal to k_b . From Fourier's law it follows that the heat flux vector \mathbf{q} is comprised of x and y directional heat fluxes, q_x and q_y and looks like

$$\mathbf{q} = \begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = -K\nabla T. \quad (9)$$

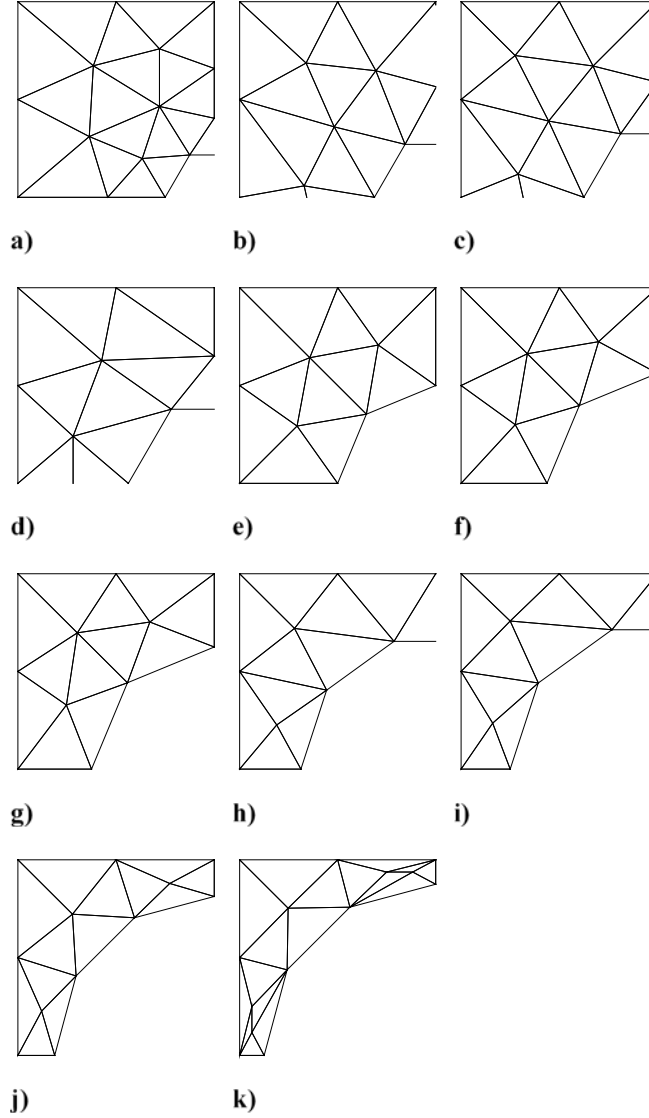
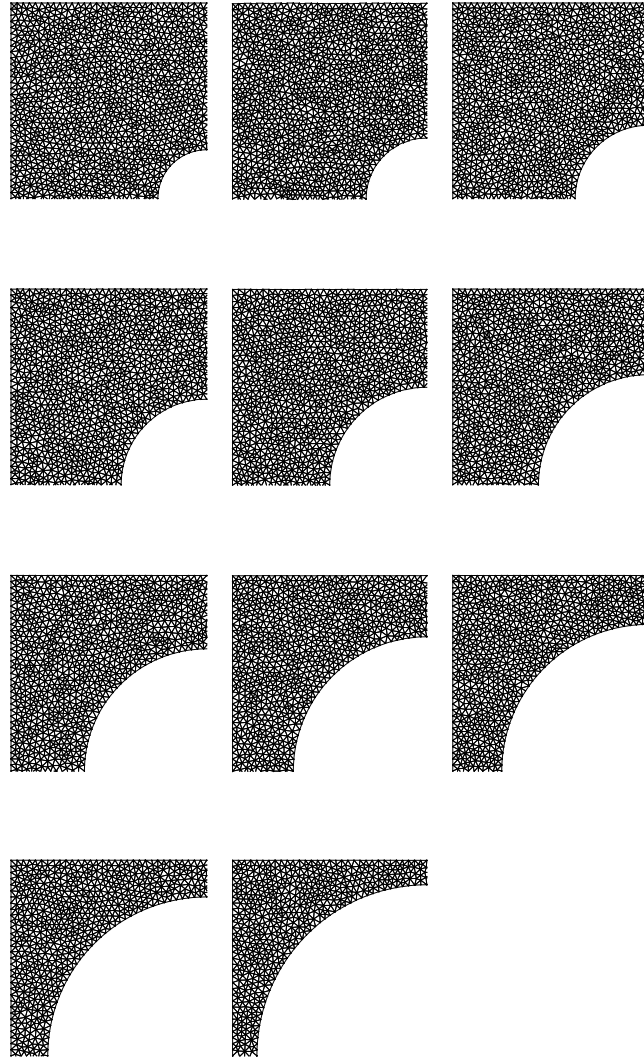


Figure 4. Coarsest mesh for varying porosity models



k)

Figure 5. Finest mesh for varying porosity models

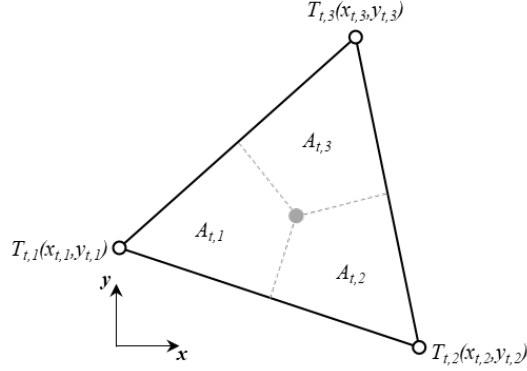


Figure 6. Notional triangle element

At the boundary, Γ , of Ω that has a surface normal vector, \mathbf{n} , the normal heat flux leaving Ω , q_n , is described by

$$q_n = \mathbf{q}^T \cdot \mathbf{n}. \quad (10)$$

The heat equation of Equation (7) can then be recast as

$$Q - \text{div}(\mathbf{q}) = Q + \text{div}(\mathbf{K}\nabla T). \quad (11)$$

This work uses the Galerkin finite element approach [45] to solve the governing PDE which requires the use of some weighting function, w , over the PDE. Multiplying Equation (11) by w and integrating over Ω results in the following

$$\int_{\Omega} (\nabla w) \mathbf{K} \nabla T d\Omega = \int_{\Omega} w Q d\Omega - \int_{\Gamma} w q_n d\Gamma. \quad (12)$$

Over any triangle element in the domain, Equation (12) holds. The Galerkin approach forces w and T to be interpolated using the same interpolation function. If T and w are interpolated in each element using the same polynomial shape function (i.e., interpolation function), N , such that in an element with vertex weighting function values of w_1 , w_2 , and w_3 , respectively,

$$T(x, y) = \mathbf{N}\mathbf{T} \text{ and } w(x, y) = \mathbf{N}\mathbf{w}, \quad (13)$$

where

$$\mathbf{T} = \begin{Bmatrix} T_{t,1} \\ T_{t,2} \\ T_{t,3} \end{Bmatrix} \text{ and } \mathbf{w} = \begin{Bmatrix} w_{t,1} \\ w_{t,2} \\ w_{t,3} \end{Bmatrix}. \quad (14)$$

Substitution of Equation (13) and Equation (14) into Equation (12) and eliminating constant and common \mathbf{w} terms yields the general finite element discretized heat equation, where

$$\int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \mathbf{T} = \int_{\Omega} \mathbf{N}^T Q d\Omega - \int_{\Gamma} \mathbf{N}^T q_n d\Gamma. \quad (15)$$

The constitutive equation drawn from Equation (15) describes the conductance, \mathbf{G} , relating \mathbf{T} and the vector of vertex net heat loads, \mathbf{P} , where

$$\mathbf{G} = \int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega, \quad (16)$$

and

$$\mathbf{P} = \int_{\Omega} \mathbf{N}^T Q d\Omega - \int_{\Gamma} \mathbf{N}^T q_n d\Gamma, \quad (17)$$

thus,

$$\mathbf{G}\mathbf{T} = \mathbf{P}. \quad (18)$$

The shape function for this work uses a linear interpolation scheme, defined in triangle t as

$$\mathbf{N} = \frac{1}{A_t} \begin{Bmatrix} x_{t,2}y_{t,3} - x_{t,3}y_{t,2} + x(y_{t,2} - y_{t,3}) + y(x_{t,3} - x_{t,2}) \\ x_{t,3}y_{t,1} - x_{t,1}y_{t,3} + x(y_{t,3} - y_{t,1}) + y(x_{t,1} - x_{t,3}) \\ x_{t,1}y_{t,2} - x_{t,2}y_{t,1} + x(y_{t,1} - y_{t,2}) + y(x_{t,2} - x_{t,1}) \end{Bmatrix}^T. \quad (19)$$

In solving the problem of interest over the system described by Figure 1, the following boundary conditions are implemented for boundaries 1 through 5:

$$\begin{aligned} 1: \quad & T(x, L) = 50 \text{ }^\circ\text{C} \\ 2: \quad & q_n(L, y) = 0 \text{ } \frac{\text{W}}{\text{m}^2} \\ 3: \quad & T(x, 0) = 100 \text{ }^\circ\text{C} \\ 4: \quad & q_n(0, y) = 0 \text{ } \frac{\text{W}}{\text{m}^2} \\ 5: \quad & q_n \left(x \in \left[\frac{L}{2} - R, \frac{L}{2} + R \right], y = \frac{L}{2} \pm \sqrt{R^2 - x^2} \right) = 0 \text{ } \frac{\text{W}}{\text{m}^2} \end{aligned} \quad (20)$$

4.1.2 Solver

The basic process of the solution method for the solver is akin to that described generally described in [46], where the discretized governing equation is a linear system, solved iteratively until some residual criteria is met. The vertex temperatures are given a target update value, $T_{tar,i}$, each iteration by solving Equation (21). Using a basic relaxation method (with relaxation coefficient, ω), the updated temperature value, $T_{new,i}$, is found by

$$T_{new,i} = T_i + \omega(T_{tar,i} - T_i). \quad (21)$$

After each update, the residual of the energy balance at each vertex, ρ_i , is computed by Equation (22), where $\boldsymbol{\rho}$ is the vector of vertex residuals, such that

$$\boldsymbol{\rho} = \mathbf{GT} - \mathbf{P} \quad (22)$$

to determine solution convergence.

4.2 Code Verification

In order to perform code verification and define the observed order of accuracy of the simulations, the method of manufactured solutions (MMS) was used, following the general guidelines given in [47,48]. In essence, MMS allows one to define a solution to the governing PDE, T_{MMS} , for the system and determine what boundary conditions and source terms satisfy that solution. By then applying the determined boundary conditions and source terms to the model, the simulations should return solutions approximating T_{MMS} (the “true” solution). In this way, one need not find an analytical solution to a given problem in order to determine the accuracy of the simulation results.

Often, T_{MMS} is constructed using exponential and/or trigonometric functions to guarantee differentiability in the governing PDE(s). In this case only trigonometric functions were employed with T_{MMS} , shown in Equation (23).

$$T_{MMS} = \cos\left(\frac{x\pi}{L}\right) \sin\left(\frac{y\pi}{L} + 0.75\right) (1^\circ\text{C}) \quad (23)$$

To obtain boundary conditions for boundaries 1 through 5—labeled in Fig. 1—the manufactured solution is operated on by the governing PDE of Equation (7). It is recommended that the boundary conditions used for the MSS problem represent as nearly as possible those used in actual problem of interest. However, an acceptable alternative is to use MMS boundary conditions of the same general type but different value as compared to the problem of interest. For this work, the alternative approach was used. Dirichlet boundary conditions were enforced on boundaries 1 and 3 for the MMS study. Likewise, Neumann boundary conditions were

prescribed for boundaries 2, 4, and 5. From the prescribed MMS temperature solution in Equation (23), the manufactured boundary conditions are given as

$$\begin{aligned}
1: \quad T_{MMS}(x, L) &= \cos\left(\frac{x\pi}{L}\right) \sin(\pi + 0.75) (1 \text{ }^\circ\text{C}) \\
2: \quad q_{n,MMS}(L, y) &= k_b \left\{ \begin{array}{c} 0 \text{ } \frac{^\circ\text{C}}{\text{m}} \\ -\cos\left(\frac{y\pi}{L} + 0.75\right)\pi \text{ } \frac{^\circ\text{C}}{\text{m}} \end{array} \right\}^T \cdot \begin{Bmatrix} -1 \\ 0 \end{Bmatrix} \\
3: \quad T_{MMS}(x, 0) &= \cos\left(\frac{x\pi}{L}\right) \sin(0.75) (1 \text{ }^\circ\text{C}) \\
4: \quad q_{n,MMS}(0, y) &= k_b \left\{ \begin{array}{c} 0 \text{ } \frac{^\circ\text{C}}{\text{m}} \\ \cos\left(\frac{y\pi}{L} + 0.75\right)\pi \text{ } \frac{^\circ\text{C}}{\text{m}} \end{array} \right\}^T \cdot \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} \\
5: \quad q_{n,MMS}(x, y) &= k_b \left\{ \begin{array}{c} -\sin\left(\frac{x\pi}{L}\right) \sin\left(\frac{y\pi}{L} + 0.75\right)\pi \text{ } \frac{^\circ\text{C}}{\text{m}} \\ \cos\left(\frac{x\pi}{L}\right) \cos\left(\frac{y\pi}{L} + 0.75\right)\pi \text{ } \frac{^\circ\text{C}}{\text{m}} \end{array} \right\}^T \cdot \begin{Bmatrix} \frac{(x - \frac{L}{2})}{R} \\ \frac{(y - \frac{L}{2})}{R} \end{Bmatrix}
\end{aligned} \tag{24}$$

It is clear from the conditions given in Equation (24), that Dirichlet boundary conditions are not constant and the Neumann boundary conditions are not all adiabatic, as opposed to the boundary conditions in the original problem of interest. This is acceptable where the MMS study boundary conditions match those of the original problem in type but not value. Similarly, the manufactured volumetric source term, Q_{MMS} , (defining the elements in \mathbf{P}) is found to be

$$Q_{MMS}(x, y) = \frac{2k_b \cos\left(\frac{x\pi}{L}\right) \sin\left(\frac{y\pi}{L} + 0.75\right)\pi^2}{L^2} (1 \text{ }^\circ\text{C}). \tag{25}$$

Note again that $k_x=k_y=k_b$ for this problem.

The overall error of the numerical MMS solution for a given mesh, ϵ_h , is determined by the RMS norm over the entire mesh, comparing the numerical vertex temperature solution to T_{MMS} , as shown in Equation (26). Here, x_i and y_i are the x and y coordinates of vertex i , and N_v is the total number of vertices in the mesh.

$$\epsilon_h = \sqrt{\sum_{i=1}^{N_v} [T_i - T_{MMS}(x_i, y_i)]^2 / N_v} \quad (26)$$

4.3 System Response Quantity

For the application of interest in this study, q_3 is found by averaging the sums of the ρ_i values for each of the i vertices along boundaries 1 and 3, respectively, induced by the constant T_l and T_3 boundary conditions, as shown by

$$q_3 = (\sum \rho_i|_{y=0} + \sum \rho_i|_{y=L})/2. \quad (27)$$

The residual at a Dirichlet boundary vertex is, in essence, the solved required heat load at that vertex to maintain the prescribed temperature at that vertex. Since L is given to be unity, and this problem is two-dimensional (i.e., domain with a depth of unity), the sum of the i heat fluxes across either boundary 1 or boundary 3 represents the average heat flux across that boundary.

Given the numerical solution provided in Equation (27), the SRQ of k^* found through Equation (3) and Equation (4) becomes

$$k^* = (\sum \rho_i|_{y=0} + \sum \rho_i|_{y=L})L/(2k_b(T_3 - T_1)). \quad (28)$$

5 Results and Discussion

5.1 Code and Solution Verification

Figure 7 shows the error convergence for each of the 11 porosity mesh series. The resulting observed order of accuracy, p_{ij} , comes from the evaluation of Equation (29), comparing two meshes of mesh size H_h where i and j are the finest and coarsest meshes of the pair, respectively.

$$p_{ij} = \ln(H_j/H_i) / \ln(\epsilon_j/\epsilon_i) \quad (29)$$

Table 2 shows the observed order of accuracy with successive mesh refinement for each of the porosities. The observed order of accuracy for each of the porosity models is approximately second order, evident by the convergent nature of p_{ij} approaching the finest mesh in each series. Table 2 also indicates the sub-figure from Figure 5 corresponding to the finest mesh of each porosity level.

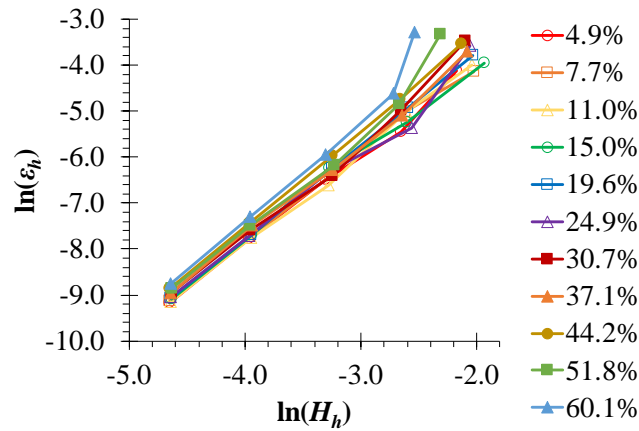


Figure 7. Mesh refinement error convergence

Table 2. Observed order of accuracy

Figure 5	α	p_{ij}				
	%	5	4	3	2	1
a	4.9	---	2.77	1.60	1.95	2.02
b	7.7	---	1.45	2.17	1.93	2.00
c	11.0	---	1.81	2.37	1.72	1.99
d	15.0	---	1.91	1.48	2.20	2.01
e	19.6	---	2.05	1.98	2.07	1.99
f	24.9	---	3.55	1.32	2.05	1.94
g	30.7	---	2.68	2.41	1.70	1.97
h	37.1	---	2.50	1.93	1.75	2.13
i	44.2	---	2.32	2.08	2.07	2.05
j	51.8	---	4.32	2.34	1.82	2.01
k	60.1	---	7.22	2.28	2.09	2.10

Figure 8 shows the convergence of the SRQ with mesh refinement for each of the α levels. The grid convergence index (GCI) is used in this work to define an approximation of numerical uncertainty, U_{num} , on the SRQ, where a brief comparison is made between the approaches using different factors of safety, FS , described in [49] and in [50]. This is done by first computing the apparent order of accuracy, p , across three successive mesh sizes, where 1, 2, and 3 are the finest to coarsest meshes of the set, respectively, as shown by

$$p = |\ln|(k_3^* - k_2^*)/(k_2^* - k_1^*)| + g(p)|/\ln(H_2/H_1). \quad (30)$$

Here, g is a convergent behavior function that is determined simultaneously with p as

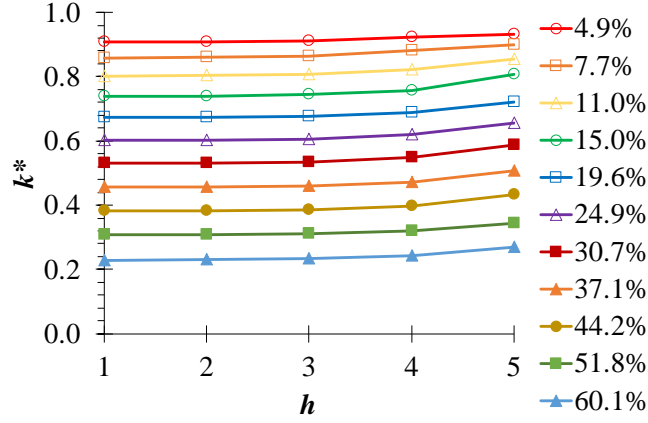


Figure 8. Solution convergence

$$g(p) = \ln(((H_2/H_1)^p - s)/((H_3/H_2)^p - s)). \quad (31)$$

where s is given to be

$$s = 1 * \text{sign}((k_3^* - k_2^*)/(k_2^* - k_1^*)). \quad (32)$$

Equation (30) and Equation (31) can be solved iteratively, due to their circular dependencies.

From here, the approximate relative error, ϵ_a , is defined between the two finest meshes as

$$\epsilon_a = |(k_1^* - k_2^*)/k_1^*|. \quad (33)$$

The formulation of the traditional GCI method in [49] deviates slightly from that used in [50] with the global deviation uncertainty estimator. The *FS* implications are of most interest here. The original *FS* given in [49] does not account for how close the solution is to the asymptotic region of convergence, but the *FS* using the global deviation uncertainty estimator accounts for the proximity of the solution convergence to the asymptotic convergence region. The traditional fine GCI is found by

$$GCI = (FS * \epsilon_a) / ((H_2/H_1)^p - 1), \quad (34)$$

where U_{num} is then defined as

$$U_{num} = GCI * k_1^* \quad (35)$$

The original FS used in Equation (34) from [49] is empirically prescribed to be 1.25 when more than two meshes are available for analysis.

However, the global deviation uncertainty estimator approach is a little more rigorous, where

$$p_m = p_f - \min[\min(|p_f - p|, 4p_f), 0.95p_f] \quad (36)$$

so that

$$FS = 3 - 1.9(p_m/p_f)^8 \quad (37)$$

and

$$U_{num} = FS |(k_2^* - k_1^*) / ((H_2/H_1)^{p_m} - 1)|. \quad (38)$$

Here, p_f is the formal order of accuracy of the numerical scheme. For this study, the formal order of accuracy is 2.

Extrapolated SRQ values, k_{ext}^* , and extrapolated relative error, ϵ_{ext} , can be computed using Equation (39) and Equation (40), where

$$k_{ext}^* = ((H_2/H_1)^p k_1^* - k_2^*) / ((H_2/H_1)^p - 1), \quad (39)$$

and

$$\epsilon_{ext} = |(k_{ext}^* - k_1^*)/k_{ext}^*|. \quad (40)$$

5.2 Thermal Results

Figure 9a through Figure 9k show the temperature contour plots for the $h=1$ for $\alpha=4.9\%$ through 60.1% , respectively. Then, Figure 10a through Figure 10e show a sample set of temperature contour plots for the $\alpha=51.8\%$ system with mesh refinement from $h=5$ to $h=1$, respectively. In both figures, temperature is in units of degrees Celsius.

In Table 3 the results of a sensitivity study on the effects of α are given with respect the GCI solution verification methods described, where the subscripts 1 and 2 on U_{num} in the table denote the original GCI approach and the global deviation uncertainty estimator approach, respectively. For most of the porosity models, the global deviation uncertainty estimator approach results in a larger U_{num} value than the traditional GCI method, implying that the original approach inadequately assumes that the solutions' convergence are closer to the asymptotic region than they really are.

Table 3. SRQ sensitivity study summary results

α	k^*	k_{ext}^*	p	ϵ_a	ϵ_{ext}	GCI	$U_{num,1}$	$U_{num,2}$
%	---	---	---	%	%	%	---	---
4.9	0.91	0.91	1.55	0.14	0.08	0.10	0.0009	0.0019
7.7	0.86	0.86	1.92	0.15	0.05	0.07	0.0006	0.0007
11.0	0.80	0.80	1.68	0.15	0.07	0.09	0.0007	0.0014
15.0	0.74	0.74	1.82	0.15	0.06	0.07	0.0005	0.0009
19.6	0.67	0.67	2.09	0.15	0.05	0.06	0.0004	0.0006
24.9	0.60	0.60	1.98	0.15	0.05	0.07	0.0004	0.0004
30.7	0.53	0.53	2.14	0.16	0.05	0.06	0.0003	0.0007
37.1	0.46	0.46	1.94	0.18	0.06	0.08	0.0004	0.0004
44.2	0.38	0.38	2.06	0.20	0.06	0.08	0.0003	0.0004
51.8	0.31	0.31	2.22	0.23	0.07	0.08	0.0003	0.0007
60.1	0.23	0.23	1.90	0.30	0.11	0.14	0.0003	0.0005

Figure 11 illustrates the culmination of this study, where the dimensionless effective thermal conductivity for a porous unit cell is plotted against cell porosity with error bars representing U_{num} for both approaches. Note that the error bars are hardly visible because uncertainties are so small. An inset magnification of the point with the largest uncertainty is shown in the figure.

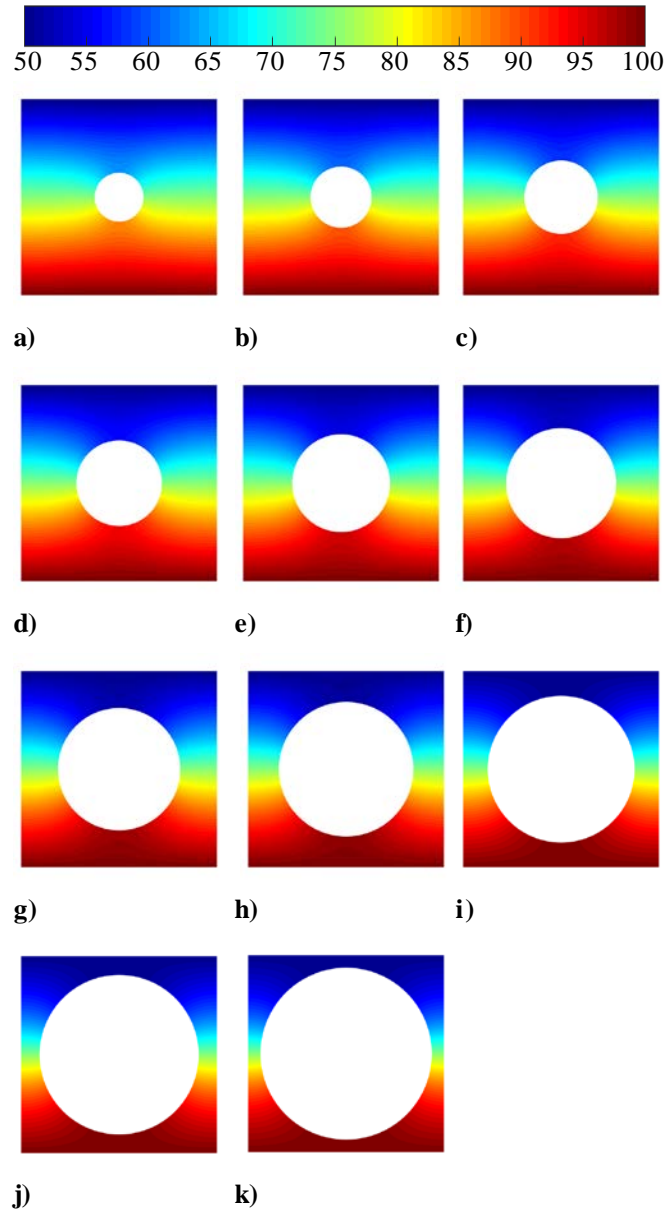


Figure 9. Finest mesh temperature contour plots for varying porosity models

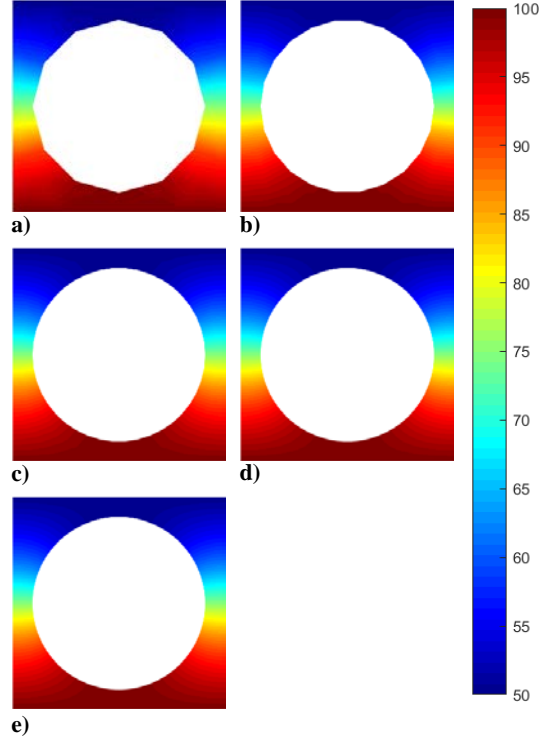


Figure 10. Temperature contour plots with mesh refinement for $\alpha=51.8\%$

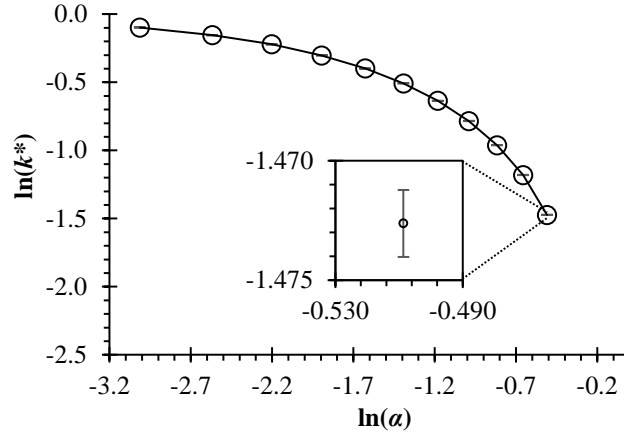


Figure 11. Dimensionless effective thermal conductivity vs. porosity

Since the computational domain is symmetric with respect to the pore geometry and the centerlines, the level of symmetry of the computational results were evaluated. To quantify

the level of symmetry two probes were defined that span the x -dimension of the cell, intersecting the radius of the pore at the minimum and maximum y -positions, as diagramed in Figure 12.

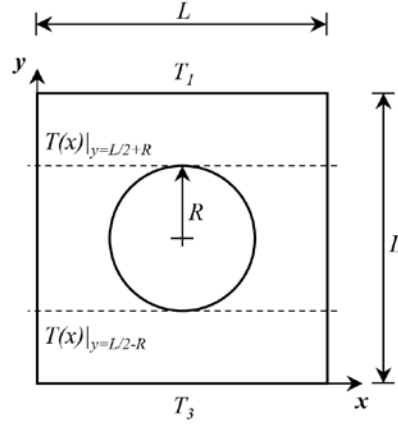


Figure 12. Line probe configurations for dimensionless temperature difference measurement

A dimensionless temperature difference, η , was defined as

$$\eta = \frac{2R \left(T(x)|_{y=\frac{L}{2}-R} - T(x)|_{y=\frac{L}{2}+R} \right)}{(L[T_b - T_l])}, \quad (41)$$

and is plotted against normalized x -position in Figure 13.

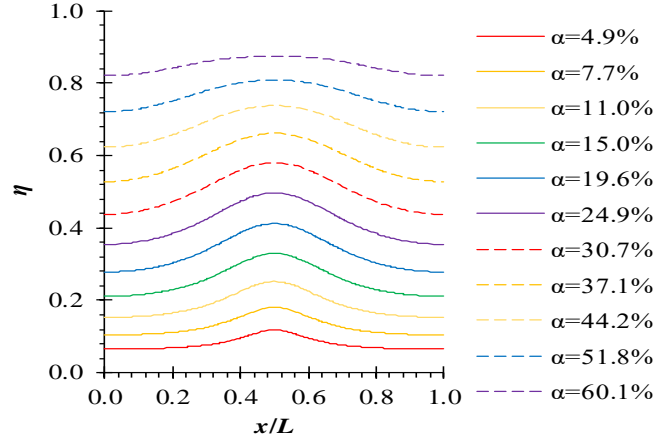


Figure 13. Dimensionless temperature difference profiles

As Figure 13 shows, by increasing the cell porosity, the uniformity of temperature across the pore increases. Stated differently, the temperature difference between the two probes from Figure 12 increases as the pore expands and the probes are located closer to the boundary conditions.

5.3 Analytical and Empirical Comparisons

As mentioned previously, Landauer proposed an analytical solution, shown in Equation (1), to predict k_{eff} as a function of α [15]. To be consistent with this study, k_2 —in Landauer’s equation—is set to 0 to represent the adiabatic void, and the entire equation is then normalized with respect to k_l , which is the same as k_b in this work. This modification yields k^* as predicted by Landauer. Landauer’s equation is shown in Figure 14 in comparison with the numerical results from this study.

In addition to numerical and analytical relationships, empirical data is also presented in Figure 14 from Smith et al. [16] and Edrisi, Bidhendi, and Haghighat [51]. Data from Smith et al. is taken from different alumina ceramic materials, and the data from Edrisi, Bidhendi, and Haghighat is taken from ceramic brick materials.

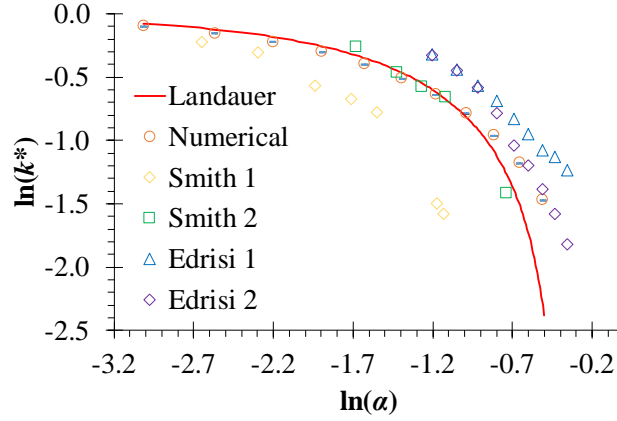


Figure 14. Analytical, numerical, and empirical dimensionless effective thermal conductivity comparisons

Although qualitative in nature, this side-by-side comparison of three different evaluation methods (analytical, numerical, and experimental) illustrates that both the analytical and numerical approaches are good approximating predictors for effective thermal conductivity values in porous media. However, the caveat remains that effective material property responses to porosity can vary drastically. Although model validation is not the primary effort of this study, such a qualitative comparison lends confidence to the numerical results obtained.

6 Conclusion

This work presented a set of second order accurate Galerkin FEM analyses that were used to discover the relationship between porosity and effective thermal conductivity in a two-dimensional porous unit cell of isotropic thermal conductivity with a centered circular void. A representative unit cell element was defined with 11 different porosities considered, ranging from 4.9% to 60.1%. Focus was added to the veracity of the computational simulation, where the method of manufactured solutions was used to perform code verification and evaluate order

of accuracy, and the GCI was used to approximate numerical uncertainty during solution verification. Two GCI-based solution verification approaches were performed, including a traditional approach by Celik et al. and a more modern global deviation uncertainty estimator approach from Phillips and Roy. Results from the study showed very small numerical uncertainty, on the order of 0.1% of the computed dimensionless effective thermal conductivity. The resulting thermal conductivity computations were compared to Landauer's analytical correlation and to a small set of empirical data with good agreement.

References

- 1 Kaviany, M., 1995, *Principles of Heat Transfer in Porous Media*, Springer, New York, NY.
- 2 Bergman, T. L., Incropera, F. P., and Lavine, A. S., 2011, *Fundamentals of Heat and Mass Transfer*, John Wiley & Sons, Hoboken, NJ.
- 3 Zhang, H.-F., X.-S. Ge, and H. Ye, 2006, "Effective thermal conductivity of two-scale porous media," *Applied Physics Letters*, **89**(8), 081908-1, 081908-3.
- 4 Sayari, A. and Jaroniec, M., Eds., 2005, *Nanoporous Materials IV*, Elsevier Science
- 5 Nield, D. A. and Bejan, A., 2006, "Convection with Change of Phase," *Convection in Porous Media*, pp. 305-344.
- 6 Berryman, J. G., 2005, "Thermal conductivity of porous media," *Applied Physics Letters*, **86**(3), pp. 032905-1, 032905-3.
- 7 Singh, K. J., Singh, R., and Chaudhary, D. R., 1998, "Heat conduction and a porosity correction term for spherical and cubic particles in a simple cubic packing," *Journal of Physics D: Applied Physics*, **31**(14), p. 1681.
- 8 Thovert, J. F., Wary, F., and Adler, P. M., 1990, "Thermal conductivity of random media and regular fractals," *Journal of Applied Physics*, **68**(8), pp. 3872-3883.
- 9 Kashiwagi, T., Grulke, E., Hilding, J., Groth, K., Harris, R., Butler, K., Shields, J., Kharchenko, S., and Douglas, J., 2004, "Thermal and flammability properties of polypropylene/carbon nanotube nanocomposites," *Polymer*, **45**(12), pp. 4227-4239.
- 10 Liu, S., Yan, H., Fang, Z., and Wang, H., 2014, "Effect of graphene nanosheets on morphology, thermal stability, and flame retardancy of epoxy resin," *Composites Science and Technology*, **90**, pp. 40-47.
- 11 Yang, S.-Y., Lin, W.-N., Huang, Y.-L., Tien, H.-W., Wang, J.-Y., Ma, C.-C. M., Li, S.-M., and Wang, Y.-S., 2011, "Synergetic effects of graphene platelets and carbon

- nanotubes on the mechanical and thermal properties of epoxy composites,” *Carbon*, **49**(3), pp. 793-803.
- 12 Ramanathan, T., Abdala, A. A., Stankovich, S., Dikin, D. A., Herrera-Alonso, M., Piner, R. D., Adamson, D. H., Schniepp, H. C., Chen, X., Ruoff, R. S., Nguyen, S. T., Aksay, I. A., Prud’Homme, R. K., and Brinson, L. C., 2008, “Functionalized graphene sheets for polymer nanocomposites,” *Nature Nanotechnology*, **3**, pp. 327-331.
 - 13 Wu, Q., Zhu, W., Zhang, C., Liang, Z., and Wang, B., 2010, “Study of fire retardant behavior of carbon nanotube membranes and carbon nanofiber paper in carbon fiber reinforced epoxy composites,” *Carbon*, **48**(6), pp. 1799-1806.
 - 14 Knight, C. C., Ip, F., Zeng, C., Zhang, C., and Wang, B., 2012, “A highly efficient fire-retardant nanomaterial based on carbon nanotubes and magnesium hydroxide,” *Fire and Materials*, **37**(2), pp. 91-99.
 - 15 Landauer, R., 1952, "The Electrical Resistance of Binary Metallic Mixtures," *Journal of Applied Physics*, **23**(7), pp. 779-184.
 - 16 Smith, D. S., Alzina, A., Bourret, J., Nait-Ali, B., Pennec, F., Tessier-Doyen, N., Otsu, K., Matsubara, H., Elser, P., and Gonzenbach, U. T., 2013, "Thermal Conductivity of Porous Materials," *Journal of Materials Research*, **28**(17), pp. 2260-2272.
 - 17 Chatterjee, A., Verma, R., Umashankar, H.P., Kasthuriengan, S., Shivaprakash, N.C., and Behera, U., 2019, “Heat conduction model based on percolation theory for thermal conductivity of composites with high volume fraction of filler in base matrix,” *International Journal of Thermal Sciences*, **136**, pp. 389-395.
 - 18 Devupra, A., Phelan, P. E., and Prasher, R. S., 2000 “Percolation theory applied to the analysis of thermal interface materials in flip-chip technology,” *The Seventh Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, Las Vegas, NV, pp. 21-28.
 - 19 Devupra, A., Phelan, P., and Prasher, R. S., 2001, “Size effects on the thermal conductivity of polymers laden with highly conductive filler particles,” *Microscale Thermophysical Engineering*, **5**(3), pp. 177-189.
 - 20 Xue, Q., 2003, “A percolation model of metal-insulator composites,” *Physica B: Condensed Matter*, **325**, pp. 195-198.
 - 21 Zhang, G., Xia, Y., Wang, H., Tao, Y., Tao, G., Tu, S., and Wu, H., 2009, “A percolation model of thermal conductivity for filled polymer composites,” *Journal of Composite Materials*, **44**(8), pp. 963-970.
 - 22 Krischer, O. and Kroell, K., 1956, “Die wissenschaftlichen Grundlagen der Trocknungstechnik,” *Trocknungstechnik*, **1**, p. 48.
 - 23 Yagi, S. and Kunii, D., 1957, “Studies on effective thermal conductivities in packed beds,” *American Institute of Chemical Engineers Journal*, **3**(3).

- 24 Nan, C. W., Birringer, R., Clarke, D. R., and Gleiter, H., 1997, "Effective thermal conductivity of particulate composites with interfacial thermal resistance," *Journal of Applied Physics*, **10**, pp. 6692–6699.
- 25 Every, A. G., Tzou, Y., Hasselman, D. P. H., and Ray, R., 1992, "The effect of particle size on the thermal conductivity of ZnS/diamond composites," *Acta Metallurgica et Materialia*, **40**, pp. 123–129.
- 26 Torquato, S. and Rintoul, M. D., 1995, "Effect of interface on the properties of composite media," *Physical Review Letters*, **75**, pp. 4067–4070.
- 27 Cheng, S. C. and Vachon, R. I., 1970, "A technique for predicting the thermal conductivity of suspensions, emulsions and porous materials," *International Journal of Heat and Mass Transfer*, **13**(3), pp. 537-546.
- 28 Agari Y., Ueda A., Omura Y., and Nagai S., 1997, "Thermal diffusivity and conductivity of PMMA–PC blends," *Polymer*, **38**, pp. 801–807.
- 29 de Vries, D. A., 1952, *Mededlingen van de Landbouwhogeschool te Wageningen*.
- 30 Kunii, D. and Smith, J. M., 1960, "Heat transfer characteristics of porous rocks," *American Institute of Chemical Engineers Journal*, **6**, pp. 71–77.
- 31 Schlünder, E. U., 1958, "Wärme-und Stoffübertragung zwischen durchströmten Schüttungen und darin eingebetteten Einzelkörpern," *Chemie-Ing.-Techn.*, **38**, pp. 967–979.
- 32 Zehner, P. and Schlünder, E. U., 1970, "Wärmeleitfähigkeit von Schüttungen bei mäßigen Temperaturen," *Chemie-Ing.-Techn.*, **42**, pp. 933–941.
- 33 Woodside, W. and Messmer, J. H., 1961, "Thermal conductivity of porous media," *Journal of Applied Physics*, **32**, pp. 1688–1706.
- 34 Krupiczka, R., 1967, "Analysis of thermal conductivity in granular materials," *International Chemical Engineering*, **7**, pp. 122–144.
- 35 Oden, J. T., Belytschko, T., Fish, J., Hughes, T. J., Johnson, C., Keyes, D., Laub, A., Petzold, L., Srolovitz, D., and Yip, S., 2006, "Revolutionizing Engineering Science through Simulation," National Science Foundation, Alexandria, VA.
- 36 Kwaśniewski, L., 2013, "Application of grid convergence index in FE computation," *Bulletin of the Polish Academy of Sciences: Technical Sciences*, **61** (1), pp. 123-128.
- 37 Wang, M., Wang, J., Pan, N., Chen, S., and He, J., 2007, "Three-dimensional effect on the effective thermal conductivity of porous media," *Journal of Physics D: Applied Physics*, **40**(1), p. 260.
- 38 Bakker, K., 1997, "Using the finite element method to compute the influence of complex porosity and inclusion structures on the thermal and electrical conductivity," *International Journal of Heat and Mass Transfer*, **40**(15), pp. 3503-3511.

- 39 Fiedler, T., Löffler, R., Bernthaler, T., Winkler, R., Belova, I. V., Murch, G. E., and Öchsner, A., 2009, "Numerical analyses of the thermal conductivity of random hollow sphere structures." *Materials Letters*, **63**(13), pp. 1125-1127.
- 40 Kou, J., Wu, F. Lu, H., Xu, Y., and Song, F., 2009, "The effective thermal conductivity of porous media based on statistical self-similarity," *Physics Letters A*, **374**(1), pp. 62-65.
- 41 Wang, J., Carson, J. K., North, M. F., and Cleland, D. J., 2006, "A new approach to modelling the effective thermal conductivity of heterogeneous materials," *International Journal of Heat and Mass Transfer*, **49**(17), pp. 3075-3083.
- 42 Wang, M., Wang, J., Pan, N., and Chen, S., 2007, "Mesoscopic predictions of the effective thermal conductivity for microscale random porous media," *Physical Review E*, **75**(3), pp. 036702-1, 036702-10.
- 43 Rocha, R. P. A. and Cruz, M. A. E., 2001, "Computation of the effective conductivity of unidirectional fibrous composites with an interfacial thermal resistance," *Numerical Heat Transfer: Part A: Applications*, **39**(2), pp. 179-203.
- 44 Alaie, S., Goettler, D. F., Su, M., Leseman, Z. C., Reinke, C. M., and El-Kady, I., 2015, "Thermal transport in phononic crystals and the observation of coherent phonon scattering at room temperature," *Nature Communications*, **6**, pp. 1-8.
- 45 Cook, R. D., Malkus, D. S., Plesha, M. E., and Witt, R. J., 2002, *Concepts and Applications of Finite Element Analysis*, 4th Ed., John Wiley & Sons, Inc., Hoboken, NJ.
- 46 Patankar, S. V., 1980, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publishing Corporation.
- 47 Roache, P. J., 2009, *Fundamentals of Verification and Validation*, Hermosa Publishers, Socorro, NM.
- 48 Oberkampf, W. L. and Roy, C. J., 2010, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge, UK.
- 49 Celik, I. B., Ghia, U., Roache, P. J., Freitas, C. J., Coleman, H., and Raad, P. E., 2008, "Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications," *Journal of Fluids Engineering*, **130**(7), pp. 0780011-0780014.
- 50 Phillips, T. S. and Roy, C. J., 2016 "A New Extrapolation-Based Uncertainty Estimator for Computational Fluid Dynamics," *Journal of Verification, Validation and Uncertainty Quantification*, **1**(4), pp. 041006-1, 041006-13.
- 51 Edrisi, S., Bidhendi, N. K., and Haghighi, M., 2017, "A new approach to modeling the effective thermal conductivity of ceramics porous media using a generalized self-consistent method," *Heat Mass Transfer*, **53**(1), pp. 321-330.

CHAPTER 3: COMPUTATIONAL EVALUATION OF THERMAL BARRIER

COATINGS: TWO-PHASE THERMAL TRANSPORT ANALYSIS³

Abstract

The turbine inlet temperature (TIT) of gas turbine systems is a cardinal factor for the efficiency and the overall—often combined—thermal power cycle efficiency. Efficient of the gas turbine system can be increased by increasing the TIT. Consequently, an increase of TIT also increases the turbine component temperature which can have critical adverse effects, such as hot gas attack, corrosion, and thermal creep. Thermal barrier coatings (TBCs)—multi-component media coatings—can mitigate these problems and protect the virgin material of the underlying metal turbine blade. The effective thermal conductivity of the TBC composite is extremely important in the design and thermal/structural assessments of a gas turbine system. In the work presented here, TBC material structure is represented in a simplified two-dimensional model. The effective thermal conductivity of the material system is evaluated under a variety of material configurations. This article details a numerical study on the steady-state thermal response of two-component composite media in two dimensions using personal finite element analysis (FEA) code. Specifically, the system response quantity (SRQ) under investigation is the dimensionless effective thermal conductivity of the domain which relates the effective thermal conductivity of the modified system to the absolute thermal conductivity

³ At the time of compiling this dissertation, this chapter was under review for publication in the Journal of Applied Thermal Engineering under the title “**A Computational Approach to Study the Thermal Response of Thermal Barrier Coatings.**” The content in this chapter was reproduced directly from the journal article, differing only in style formatting. A less comprehensive version of this chapter was published in *Proceedings of the ASME 2019 Verification and Validation Symposium* under the title “Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis.”

of the original matrix material. A thermally conductive solid matrix domain is modeled with a thermally conductive solid circular filler arranged in a uniform packing configuration. Both the filler size and the filler thermal conductivity are varied over a range of values to investigate the relative effects on the SRQ. Fill fraction was varied from 2% to 78%, and the filler-to-matrix thermal conductivity ratio was varied from 0 to 2. Most related works in the open literature fail to provide any—let alone formal and rigorous—verification for computational results, undermining the credibility of the presented data. In this investigation, an emphasis is placed on using code and solution verification techniques to evaluate the obtained results and provide credibility evidence to computationally produced data. The method of manufactured solutions (MMS) was used to perform code verification for the study, showing the FEA code to be second order accurate. Solution verification was performed using the grid convergence index (GCI) approach with the global deviation uncertainty estimator on a series of five systematically refined meshes for each fill fraction and thermal conductivity model configuration. A comparison of the SRQs across all domain configurations is made, including numerical uncertainty derived through the GCI analysis. Trends for the effective thermal response of the multi-component system are shown and briefly compared to empirical data from open literature, showing qualitative trend agreement between the computational and empirical data.

Nomenclature

A	= area
α	= fill fraction
Γ	= domain boundary
ε	= error

FS	= factor of safety
\mathbf{G}	= conductance matrix
h	= characteristic mesh size
H	= mesh number
i	= index
k	= thermal conductivity
\mathbf{K}	= thermal conductivity matrix
L	= cell length
N	= total quantity
N	= shape function
Ω	= domain
p	= order of accuracy
\mathbf{P}	= vertex heat load vector
\mathbf{q}	= heat flux vector
Q	= heat flow per unit length
r	= mesh ratio
R	= void radius
ρ	= energy balance residual
$\boldsymbol{\rho}$	= energy balance residual vector
S	= energy source
t	= index
T	= temperature
\mathbf{T}	= triangle vertex temperature vector

u = system response quantity

U = uncertainty

W = cell half-width

x = x-coordinate

y = y-coordinate

Subscripts

C = cold

eff = effective

f = formal

H = mesh number, hot

i = index

j = index

L^∞ = L_∞ norm

MMS = manufactured solution

n = normal

num = numerical

O = observed

t = triangle, transcendental, index

v = vertex

x = x-direction

y = y-direction

Superscripts

$*$ = dimensionless, global

1 Introduction

The future of increased efficiency in gas turbine-driven power plant energy generation is heavily dependent on the increased temperature of the turbine inlet temperature (TIT) [1-3]. In spite of the pursuit of higher TIT levels, virgin metal material in the turbine components are susceptible to a variety of aggressive and high-consequence degradation or failure modes. Such issues include accelerated thermal creep, material degradation due to oxidation, and cycle fatigue [4]. Thermal-related corrosion issues arise at various temperature levels anywhere in the range of 650-1700 °C [2,5-7].

To mitigate thermal issues, thermal barrier coatings (TBCs) are used to protect the virgin blade materials, as illustrated in the notional TBC-turbine blade schematic given in Figure 1. Such coatings, including spray technologies are typically comprised of some form of ceramic composite material [8-10]. These composite materials are heterogeneous in nature and provide significant protection to the metal turbine components to allow for better overall performance and efficiency.

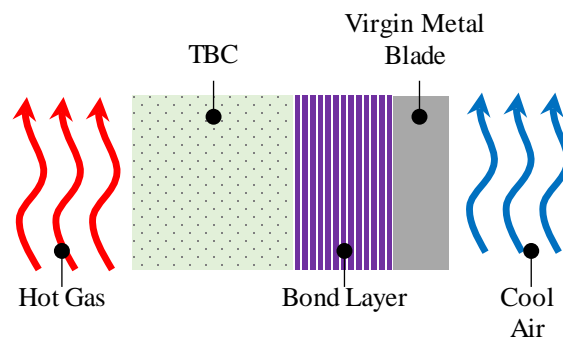


Figure 1. Notional schematic of TBC application in gas turbine power generation system

“Heterogeneous” is a broad and relatively subjective classification of materials, where a macroscale material—on a relative scale—is considered to be comprised of multiple,

different components, such as with many TBCs. The components comprising the heterogeneous material may be different fundamental materials, different orientations of materials, similar or different materials separated by boundaries, and/or different material phases. A special case of heterogeneous materials is a porous material, where voids (i.e., locales of material absence) exist within the bulk material. The foregoing list is not exhaustive, but what should be noted is the subjectivity of the classification “heterogeneous.” Heterogeneity—as opposed to homogeneity—can be determined from different perspectives such as from a length scale or from a physics perspective. For example, a continuous volume of some metallic solid may be considered homogeneous for most practical engineering purposes, such as to determine stress-strain relationships or heat transfer. In contrast, the same continuous volume of metallic solid may be considered heterogeneous when one considers the presence of distributed impurities or grain boundaries throughout the volume, perhaps affecting crack propagation under stress or material evolution during heat treatment.

Oftentimes, heterogeneity leads to more complicated physics at the level or scale where heterogeneity exists, making analysis and material behavior predictions more difficult. As a result of the intricate descriptions or understanding of the physics required to determine the behavior of a heterogeneous material or system, practitioners frequently prefer the use of effective properties or responses. Effective properties are a means of describing the general behavior of the system—based on the characterization of the pertinent heterogeneities or conditions in the system. A commonplace example is the use of a convective heat transfer coefficient for fluid flow in a given flow regime. Although complicated fluid mechanics and dynamics may be taking place—such as turbulent mixing and boundary layer formation—many fluid flow conditions and geometries can be simply characterized, and the effective

resulting thermal transport behavior can be approximated using a straightforward analytical correlation. As engineering systems become increasingly complex, it can be advantageous—and even necessary—to use effective properties to analyze and predict system performance in a feasible manner.

It has been shown that the heterogeneous microstructures of the TBCs at a material level have significant effects on the effective thermal conductivity of the TBC. Effective thermal conductivity plays a significant role in the performance and longevity of the TBCs and consequently the turbine blades that they protect [11-14].

Understanding the thermophysical properties of multi-component TBC materials is critical to the design and advancement of gas turbines for plant power generation for current and future installations. As systems continue to produce higher TIT levels, the thermal performance of advanced TBCs becomes more critical. This investigation evaluates the effective thermal transport in simplified models of heterogeneous TBC structures, specifically two-component media, as an initial study into TBC material nano-structure effective thermal response.

For many decades, researchers have investigated the effects of material fillers components on the effective thermal conductivity of a medium, developing theoretical analytical correlations to describe the implications of filler size and geometry on the thermal behavior of the composite medium [15-21]. However, with the advent of computational analysis, researchers have continued the work around heterogeneous material effective thermal behavior from a simulation-based approach [22-25]. It is the author's observation that a general lack of formal verification and/or validation is found in the literature to support the presented heterogeneous material computational analyses. While computational tools are extremely

powerful in discovering physical behaviors in engineering applications, processes ought to be followed to provide evidence of the veracity of the computational results. Thus, in this work formal code and solution verification is procedures are presented as foundational aspects of the presented results.

2 System Description

In order to analyze the thermal transport phenomena within the TBC as a part of the authors' efforts, a two-dimensional system is constructed, consistent with the illustration given in Figure 2. For this study, a unit half-cell is given with height, L , and width, W , and is oriented in the Cartesian x - y plane with unit thickness out-of-plane. The bulk matrix material is defined as the first material phase with thermal conductivity k_1 , where the filler phase is defined to have thermal conductivity k_2 . The filler pattern is circular, centered in the unit cell, with radius, R . Both the $x=0$ and $x=W$ boundaries are given periodic boundary conditions for system symmetry, and the $y=0$ and $y=L$ boundaries have enforced hot and cold boundary temperatures, T_H and T_C , respectively. By enforcing Dirichlet boundary conditions at the hot and cold boundaries, heat flow—per unit length—is induced in the positive y -direction, perpendicular to each boundary, indicated by Q_H and Q_C , respectively.

TBCs, the ultimate system response quantity (SRQ) of interest in this analysis is the dimensionless effective thermal conductivity, k^* , which is merely the ratio of the effective thermal conductivity with the matrix material thermal conductivity, simply evaluated as

$$k^* = k_{eff}/k_1. \quad (3)$$

This study will show the effects on the SRQ of varying the k_2 -to- k_1 ratio along with varying the α , using numerical verification techniques to quantify uncertainty in that value. Ratios in the two-component thermal conductivities are evaluated from 0 to 2, and α values are varied from 2% to 78%.

3 Numerical Approach

3.1 Discretization

In order to determine the SRQ, the temperature distribution, T , and resultant heat source terms, S , must be solved to satisfy the governing partial differential equation (PDE), the heat equation, as expressed in Equation (4), where k_x and k_y are the directional thermal conductivities in the x -direction and y -direction, respectively.

$$\frac{\partial}{\partial x} \left(k_x \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(k_y \frac{\partial T}{\partial y} \right) + S = 0 \quad (4)$$

To numerically discretize the heat equation, unstructured triangular meshes were generated over the domain within each material phase. To accommodate the verification approaches used in this study, systematically refined meshes were generated for each fill fraction level, an example of which is shown in Figure 3a through Figure 3e, for mesh number, H , 5 through 1, respectively. Likewise, the finest meshes for all of the analyzed fill fraction

models of 2%, 5%, 15%, 25%, 35%, 45%, 65%, 70%, 75%, and 78% are given in Figure 4a through Figure 4k, respectively.

Each t^{th} triangle—with area, A_t —of a given mesh is configured with three vertices such that the i^{th} vertex is located at $(x_{t,i}, y_{t,i})$ with temperature $T_{t,i}$, as is shown in Figure 5. Each H^{th} mesh for a given fill fraction level is then described with a characteristic mesh size h_H . If $N_{H,t}$ is the total number of triangles in mesh H , then

$$h_H = \sqrt{\frac{1}{N_{H,t}} \sum_{t=1}^{N_{H,t}} A_t}. \quad (5)$$

Thus, systematic mesh refinement here approximately halves the characteristic mesh size from mesh $H+1$ to mesh H .

The domain discretization is used with a second order accurate Galerkin finite element method (FEM) approach which is employed to solve Equation (4) over the entire domain [26]. Equation (4) can be solved over each finite element, individually, and using a discretized PDE matrix equation. The thermal conductivities and temperature gradients from Equation (4) can be cast in matrix form as

$$\mathbf{K} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \quad \text{and} \quad \nabla T = \left\{ \frac{\partial T}{\partial x}, \frac{\partial T}{\partial y} \right\}, \quad (6)$$

where \mathbf{K} is the thermal conductivity tensor. For the system at hand, isotropic thermal conductivity is used for both the matrix and filler materials, such that k_x and k_y are the same. From Fourier's law, it follows that the heat flux vector, \mathbf{q} , is comprised of x -directional and y -directional heat fluxes, q_x and q_y , respectively, and looks like

$$\mathbf{q} = \begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = -\mathbf{K}\nabla T. \quad (7)$$

Let Ω represent a domain, with Γ being the boundary of the domain. The surface normal unit vector, \mathbf{n} , is used to express the normal heat flux, q_n , leaving Ω , such that

$$q_n = \mathbf{q}^T \cdot \mathbf{n}. \quad (8)$$

Thus, the governing heat equation can be expressed equally as

$$S - \text{div}(\mathbf{q}) = S + \text{div}(\mathbf{K}\nabla T). \quad (9)$$

The Galerkin finite element approach—used in this work to solve the governing PDE—requires the use of some weighting function, w , over the PDE. Multiplying Equation (9) by w and integrating over Ω yields

$$\int_{\Omega} (\nabla w) \mathbf{K} \nabla T d\Omega = \int_{\Omega} w S d\Omega - \int_{\Gamma} w q_n d\Gamma. \quad (10)$$

For each triangle in the finite element domain, Equation (10) is true, and the Galerkin method prescribes that both w and T be interpolated across the finite element domain using the same row vector polynomial shape function, or interpolation function, N . If a triangle element has weighting function and temperature values defined at each of its three vertices as $w_{t,i}$ and $T_{t,i}$, respectively, for $i=1,2,3$ corresponding to each of the i^{th} vertices of the t^{th} triangle, then let

$$\mathbf{T} = \begin{Bmatrix} T_{t,1} \\ T_{t,2} \\ T_{t,3} \end{Bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{Bmatrix} w_{t,1} \\ w_{t,2} \\ w_{t,3} \end{Bmatrix}, \quad (11)$$

for a given triangle. Thus, the field variable and weight function within the bounds of the finite element are described similarly as

$$T(x, y) = \mathbf{N}\mathbf{T} \quad \text{and} \quad w(x, y) = \mathbf{N}\mathbf{w}. \quad (12)$$

By using matrix transpose rules, substituting the definitions of Equation (12) into Equation (10), and eliminating common constant terms, the discretized finite element heat equation is produced as

$$\int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \mathbf{T} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma. \quad (13)$$

Furthermore, the different parts of Equation (6) are condensed and defined as the conductance matrix, \mathbf{G} , and the heat load matrix \mathbf{P} , where

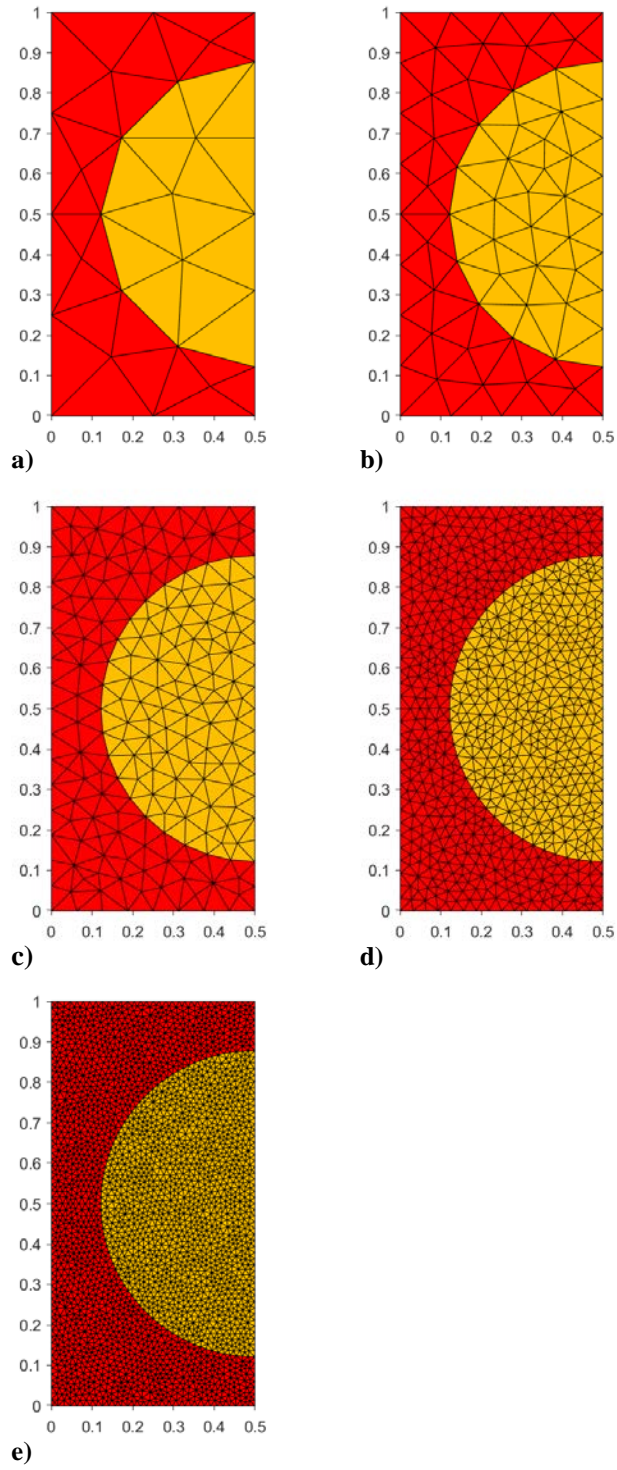


Figure 3. Systematic mesh refinement for $\alpha=45\%$

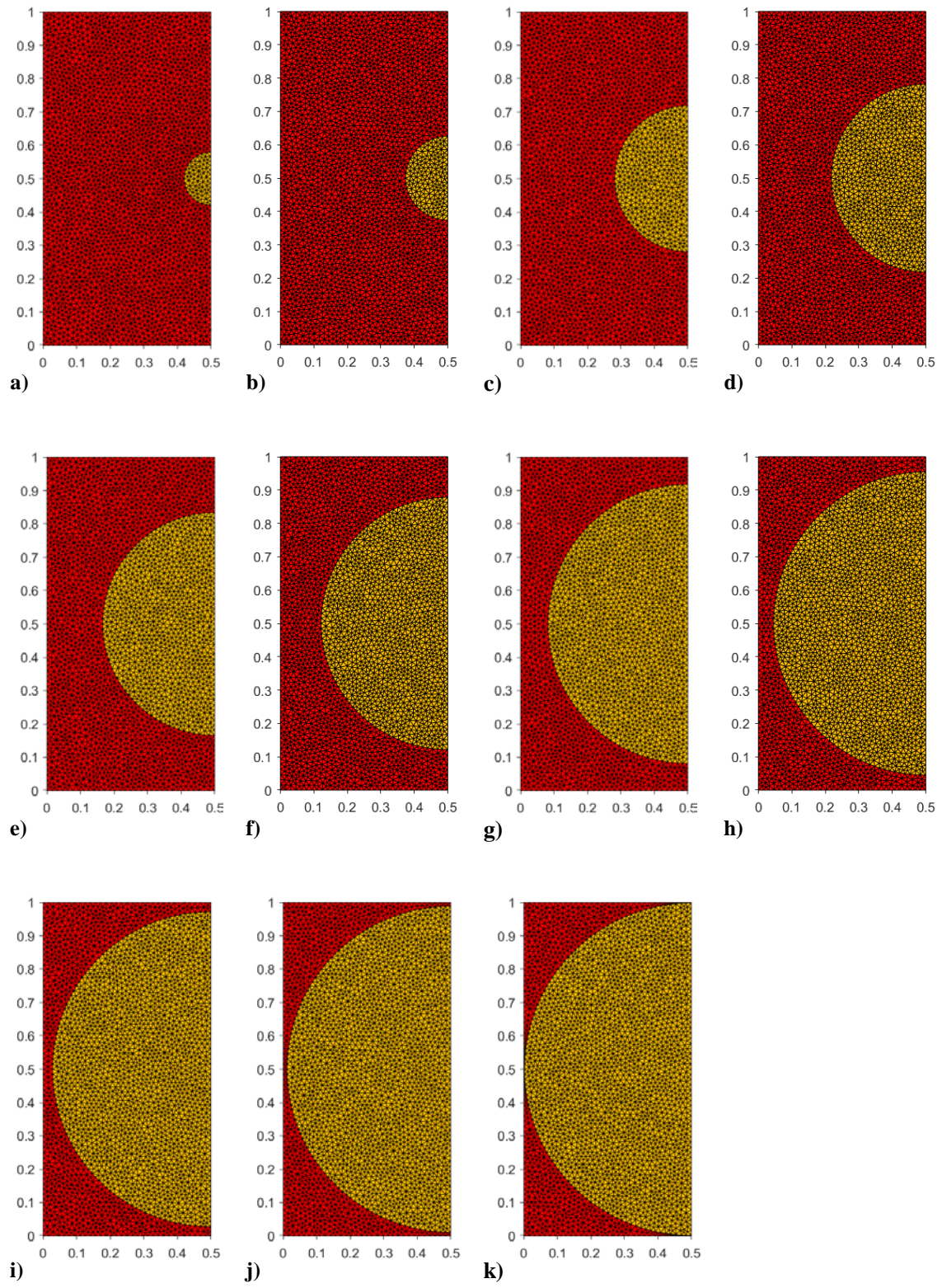


Figure 4. Finest mesh for varying fill fraction levels

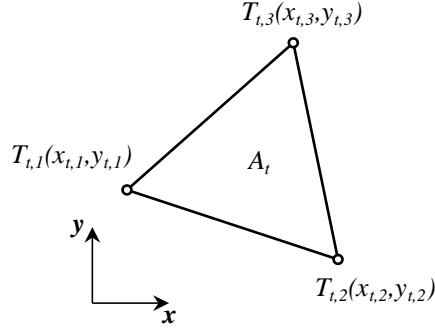


Figure 5. Notional linear triangle element

$$\mathbf{G} = \int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega. \quad (14)$$

and

$$\mathbf{P} = \int_{\Omega} \mathbf{N}^T \mathbf{S} d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma, \quad (15)$$

such that

$$\mathbf{G}\mathbf{T} = \mathbf{P}. \quad (16)$$

In this study, a linear triangle shape function is used to interpolate temperature and weight functions by

$$\mathbf{N} = \frac{1}{A_t} \begin{Bmatrix} x_{t,2}y_{t,3} - x_{t,3}y_{t,2} + x(y_{t,2} - y_{t,3}) + y(x_{t,3} - x_{t,2}) \\ x_{t,3}y_{t,1} - x_{t,1}y_{t,3} + x(y_{t,3} - y_{t,1}) + y(x_{t,1} - x_{t,3}) \\ x_{t,1}y_{t,2} - x_{t,2}y_{t,1} + x(y_{t,1} - y_{t,2}) + y(x_{t,2} - x_{t,1}) \end{Bmatrix}^T. \quad (17)$$

3.2 Solver

The meshes employed in this study were generated using open-source Gmsh software [27]. The custom code used for this study was written in Fortran as a generic two-dimensional FEM heat transfer solver. A basic successive over-relaxation method was used to iteratively update solutions to Equation (16) until some residual level exit criteria was met.

The residual vector, ρ , for the system is computed as

$$\rho = GT - P. \quad (18)$$

where the L_∞ norm residual, ρ_{L_∞} , defined as

$$\rho_{L_\infty} = \max|\rho|. \quad (19)$$

A ρ_{L_∞} value of 10^{-8} was used as exit criteria for solution completion.

4 Verification Approach

4.1 Code Verification

The order of accuracy of a finite element computational method describes the rate of change in error with respect to change in finite element size. For example, a second order accurate model is a model whose solution error is quartered when the mesh elements are halved. Code verification procedures seek to evaluate how well and/or if code correctly solves the governing PDE(s). In order to verify the formally second order accurate Galerkin finite element solution used for this study, the Method of Manufactured Solutions (MMS) was employed [28,29]. The MMS approach allows for a user-defined temperature distribution to be selected, T_{MMS} , where the operator—in this case defined by the PDE in Equation (4)—acts on the manufactured solution. In general, the user should select a solution that exercised all of

the terms and dimensions of the problem at hand. Typically, trigonometric and exponential functions are used to define the manufactured solution because they are smooth and infinitely differentiable. The resulting source term, S_{MMS} , can be determined by applying the operator to the manufactured solution, where

$$S_{MMS} = -k \left(\frac{\partial^2 T_{MMS}}{\partial x^2} + \frac{\partial^2 T_{MMS}}{\partial y^2} \right). \quad (20)$$

Thus, by applying S_{MMS} and the boundary conditions that satisfy T_{MMS} to the domain, the solved system of equations should converge to T_{MMS} with mesh refinement. Here, T_{MMS} is defined as

$$T_{MMS}(x, y) = \cos(2\pi x) \sin(\pi y + 0.75) \quad (21)$$

and is applied as a boundary condition at $y=0$ and $y=L$ as Dirichlet conditions. It follows that the first partial derivatives are given to be

$$\begin{aligned} \frac{\partial T_{MMS}}{\partial x} &= -2\pi \sin(2\pi x) \sin(\pi y + 0.75) \\ \frac{\partial T_{MMS}}{\partial y} &= \pi \cos(2\pi x) \cos(\pi y + 0.75) \end{aligned} \quad (22)$$

and are enforced at the boundaries where $x=0$ and $x=W$ as Neumann boundary conditions.

With the second partial derivatives of temperature being

$$\begin{aligned} \frac{\partial^2 T_{MMS}}{\partial x^2} &= -4\pi^2 \cos(2\pi x) \sin(\pi y + 0.75) \\ \frac{\partial^2 T_{MMS}}{\partial y^2} &= -\pi^2 \cos(2\pi x) \sin(\pi y + 0.75) \end{aligned}, \quad (23)$$

the resulting source term distribution is then

$$S_{MMS}(x, y) = 5\pi^2 k \cos(2\pi x) \sin(\pi y + 0.75). \quad (24)$$

The MMS study is performed by applying the boundary conditions to the model that are consistent with the manufactured solution, as described above, and applying the S_{MMS} distribution to the computational domain. The model with the implemented MMS boundary conditions and source terms can then be solved to produce the resulting numerically-determined temperature distribution. The numerical temperature solution should approach the prescribed T_{MMS} solution as the finite element mesh is refined. Studying the rate of change of error between the numerical and MMS solutions with respect to mesh refinement evaluates the actual or observed order of accuracy of the implementation of the numerical methods. For this study, the L_∞ norm metric is used on temperature error, $\epsilon_{MMS,H}$, over all N_v vertices in mesh H , where

$$\epsilon_{MMS,H} = \max_i |T_i - T_{MMS,i}| \quad (25)$$

based on the solution temperature, T_i , of the i^{th} vertex. The observed order of accuracy for mesh H , $p_{O,H}$, is determined by

$$p_{O,H} = \ln(\epsilon_{MMS,H}/\epsilon_{MMS,H+1}) / \ln(h_H/h_{H+1}). \quad (26)$$

4.2 Solution Verification

Solution verification, as opposed to code verification, seeks to evaluate how well the code solves the mathematical model for the problem at hand. Computational methods inherently contain uncertainty stemming from the numerical implementation of the model. Traditionally, the expectation exists that as a computational mesh is refined, the numerical

solution converges on a single behavior or value, in this case k^* . If a computational mesh is too coarse, the solution may not be near some area of conventional convergence. In other words, during mesh refinement, the solution changes significantly and sometimes not according to the formal rules of the order of accuracy of the employed numerical method. This regime of solutions is considered the “non-asymptotic” solution space. When the solution systematically converges to a single behavior or value with mesh refinement according to formal rules of the computational method, the solution is considered to be in the “asymptotic” range.

The numerical uncertainty, U_{num} , for the SRQ, k^* , is approximated in this method by performing solution verification using a modern version of a more traditional grid convergence index (GCI) [29,30] approach. The GCI approach used here incorporates a global deviation estimator for the solution [31]. This approach uses an empirical trend to scale a factor of safety, FS , for U_{num} based on how closely the convergent observed order of accuracy is to the formal order of accuracy, thus acknowledging how close the solution is to the asymptotic solution region.

The modified transcendental order of accuracy, p_t , used for this study is given by

$$p_t = \ln \left[(r_{1,2}^{p_t} - 1) \left(\left| \frac{u_3 - u_2}{u_2 - u_1} \right| \right) + r_{1,2}^{p_t} \right] / \ln(r_{1,2} r_{2,3}), \quad (27)$$

where u_H represents the SRQ of interest as computed on mesh H , and

$$r_{i,j} = h_j/h_i. \quad (28)$$

For this problem, where the SRQ is k^* , the global deviation parameter, Δp , is computed as

$$\Delta p = \min(|p_f - p_t|, 4p_f, 0.95p_f), \quad (29)$$

with p_f being the formal order of accuracy of the solution method which, in this study, is 2.

Lastly, the global order of accuracy, p^* , used to compute the global deviation GCI FS is

$$p^* = p_f - \Delta p, \quad (30)$$

such that

$$FS = 3.0 - 1.9(p^*/p_f)^8. \quad (31)$$

The intent of the factor of safety in this approach is to scale the numerical uncertainty estimate based on how close the solution appears to be to the asymptotic region, as illustrated in Figure 6, based on the apparent convergent behavior of the SRQ with respect to mesh refinement. The FS value is then used to estimate the numerical uncertainty on the fine mesh SRQ solution, where

$$U_{num} = FS \left| (u_2 - u_1) / (r_{1,2}^{p^*} - 1) \right|. \quad (32)$$

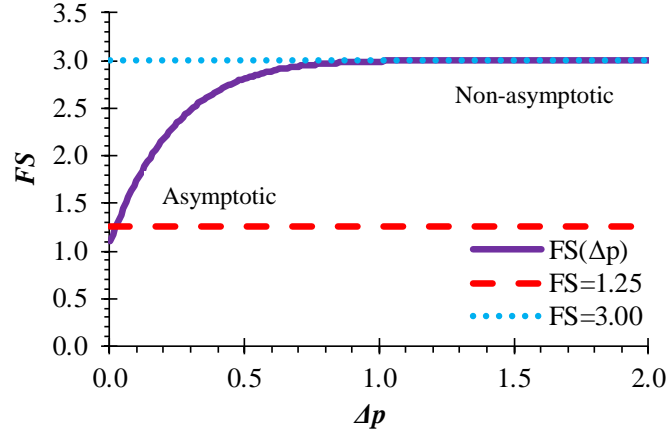


Figure 6. Global deviation estimator factor of safety

5 Results and Discussion

5.1 Code Verification

Using the MMS solution and conditions described in Equation (21) through Equation (24) to perform the MMS verification, the trends of logarithmic error against logarithmic mesh size are shown in Figure 7a and Figure 7b, for the root mean square (RMS) of the error and the L_∞ norm of the error (described by Equation (6) and used as the primary evaluation metric), respectively, where the trend of a formal second order accurate solution is also shown in qualitative comparison. Likewise, Figure 8a and Figure 8b show the convergence of the observed order of accuracy with respect to mesh number, based on the RMS and L_∞ norm of the error, respectively.

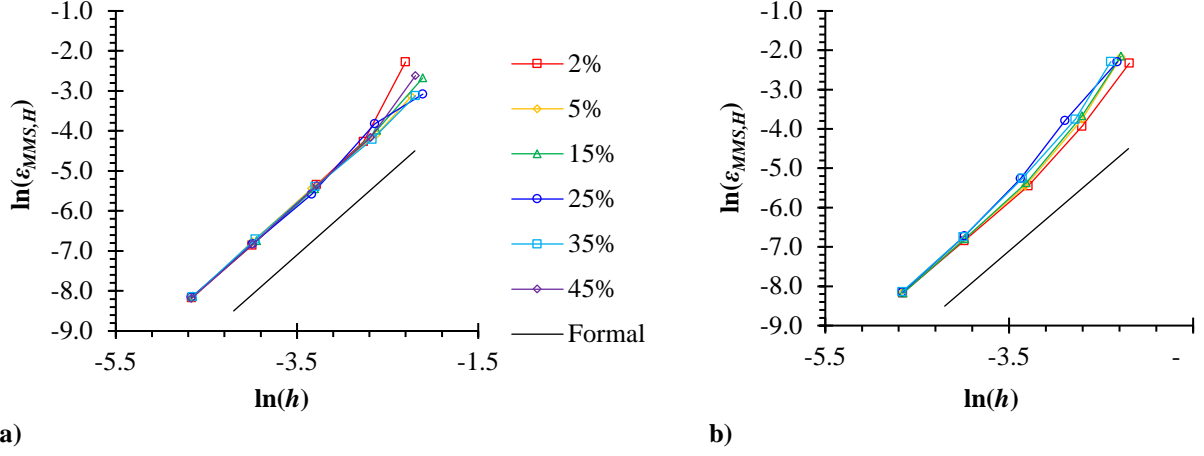


Figure 7. MMS error trend with mesh refinement based on a) RMS and b) L_∞ norm

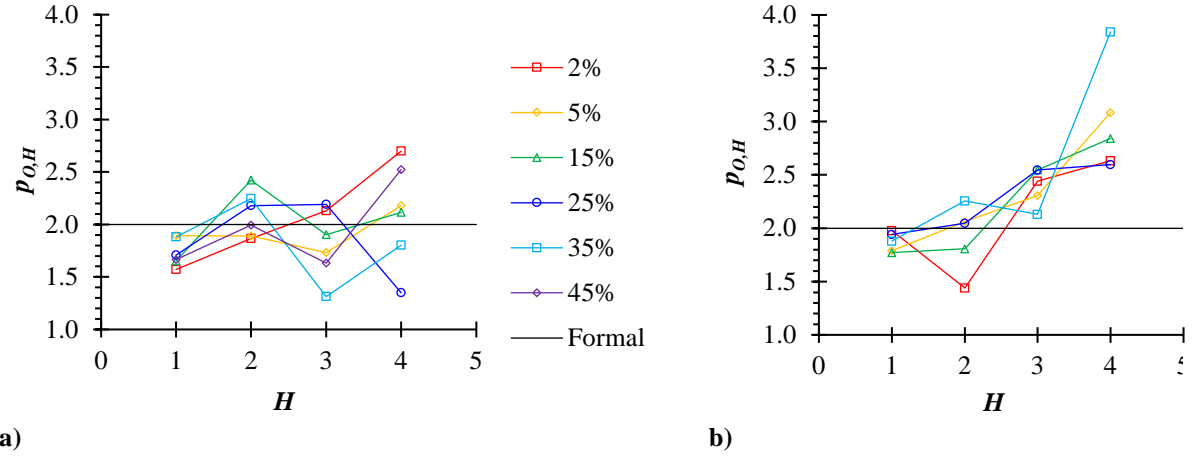


Figure 8. MMS observed order of accuracy convergence based on a) RMS and b) L_∞ norm

Table 1 through Table 8 detail the specific mesh refinement and MMS observed order of accuracy parameters, showing an approximately second order accurate scheme.

Table 1. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=2\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.30	-1.79	---
4	1.59	-2.76	-3.04	2.70
3	1.68	-3.28	-4.15	2.13

2	2.04	-4.00	-5.47	1.86
1	1.95	-4.66	-6.52	1.57

Table 2. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=5\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.24	-2.15	---
4	1.46	-2.62	-2.98	2.18
3	2.05	-3.34	-4.22	1.73
2	1.93	-3.99	-5.46	1.89
1	1.93	-4.65	-6.71	1.89

Table 3. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=15\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.11	-1.53	---
4	1.66	-2.62	-2.60	2.11
3	1.99	-3.31	-3.91	1.90
2	1.90	-3.95	-5.46	2.42
1	2.02	-4.65	-6.62	1.65

Table 4. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=25\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.11	-1.87	---
4	1.70	-2.64	-2.59	1.35
3	2.01	-3.34	-4.12	2.19
2	1.92	-3.99	-5.54	2.18
1	1.96	-4.67	-6.69	1.71

Table 5. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=35\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.19	-2.15	---
4	1.61	-2.67	-3.02	1.80
3	1.88	-3.30	-3.84	1.31
2	1.93	-3.96	-5.32	2.25
1	2.01	-4.65	-6.63	1.88

Table 6. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=45\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.19	-1.89	---
4	1.64	-2.69	-3.15	2.52
3	1.81	-3.28	-4.11	1.63
2	2.03	-3.99	-5.52	1.99
1	1.97	-4.67	-6.65	1.66

Table 7. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=55\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.19	-1.54	---
4	1.67	-2.71	-2.89	2.63
3	1.80	-3.29	-4.32	2.44
2	2.01	-3.99	-5.33	1.44
1	1.95	-4.66	-6.65	1.98

Table 8. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=65\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.28	-1.39	---
4	1.53	-2.71	-2.70	3.08
3	1.86	-3.32	-4.13	2.31
2	1.91	-3.97	-5.45	2.06
1	1.98	-4.65	-6.68	1.79

Table 9. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=70\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.28	-1.39	---
4	1.53	-2.71	-2.60	2.84
3	1.84	-3.31	-4.15	2.55
2	1.95	-3.98	-5.35	1.81
1	1.96	-4.65	-6.54	1.77

Table 10. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=75\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.32	-1.40	---

4	1.77	-2.89	-2.87	2.60
3	1.62	-3.37	-4.10	2.54
2	1.84	-3.98	-5.35	2.05
1	1.97	-4.66	-6.67	1.94

Table 11. Systematic mesh refinement and MMS observed order of accuracy parameters for $\alpha=78\%$

H	$r_{H,H+1}$	$\ln(h_H)$	$\ln(\epsilon_{MMS,H})$	$p_{O,H}$
5	---	-2.39	-1.32	---
4	1.47	-2.78	-2.80	3.84
3	1.78	-3.35	-4.03	2.13
2	1.90	-4.00	-5.47	2.26
1	1.94	-4.66	-6.72	1.88

5.2 Solution Verification

As mentioned previously, k^* is computed using the average numerically-produced values of Q_H and Q_C , as described by Equation (2) and Equation (3). These heat flow values are computed by summing the residual values, implied from Equation (18), of all vertices on the Q_H and Q_C boundaries where T_H and T_C are enforced, respectively.

After performing the solution verification procedures described with Equation (27) through Equation (32), the following tables show the computed global deviation parameters and ultimate U_{num} approximation for the finest mesh in each material configuration model. Note that the U_{num} values given in these tables are expressed as percent values relative to the presented fine mesh k^* values. All U_{num} values are obtained to be less than 0.2% of the fine mesh k^* values.

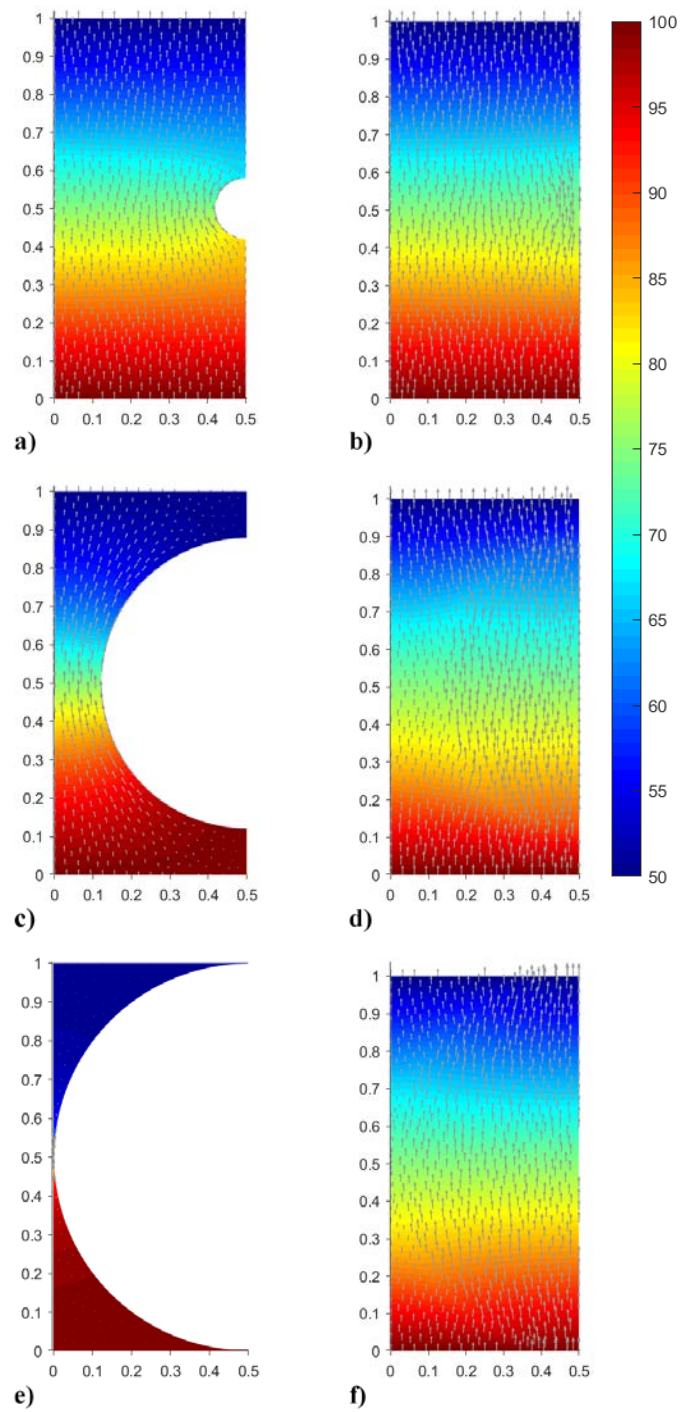


Figure 9. Temperature contour plots with heat flux vector fields for various fine mesh geometries and thermal conductivity ratios

Table 12. Global deviation GCI parameters for $\alpha=2\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.96	1.95	2.47	0.47	1.53	2.78	0.153%
0.25	0.98	1.95	2.56	0.56	1.44	2.86	0.082%
0.50	0.99	1.95	2.64	0.64	1.36	2.91	0.041%
0.75	0.99	1.95	2.71	0.71	1.29	2.94	0.016%
1.00	1.00	1.95	0.00	1.90	0.10	3.00	0.000%
1.25	1.00	1.95	2.84	0.84	1.16	2.98	0.011%
1.50	1.01	1.95	2.90	0.90	1.10	2.98	0.019%
1.75	1.01	1.95	2.97	0.97	1.03	2.99	0.025%
2.00	1.01	1.95	3.03	1.03	0.97	2.99	0.030%

Table 13. Global deviation GCI parameters for $\alpha=5\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.91	1.93	1.68	0.32	1.68	2.52	0.147%
0.25	0.94	1.93	1.67	0.33	1.67	2.54	0.072%
0.50	0.97	1.93	1.66	0.34	1.66	2.57	0.034%
0.75	0.99	1.93	1.64	0.36	1.64	2.61	0.013%
1.00	1.00	1.93	0.00	1.90	0.10	3.00	0.000%
1.25	1.01	1.93	1.60	0.40	1.60	2.68	0.008%
1.50	1.02	1.93	1.57	0.43	1.57	2.72	0.014%
1.75	1.03	1.93	1.55	0.45	1.55	2.75	0.018%
2.00	1.03	1.93	1.52	0.48	1.52	2.79	0.021%

Table 14. Global deviation GCI parameters for $\alpha=15\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.74	2.02	1.89	0.11	1.89	1.79	0.097%
0.25	0.84	2.02	1.88	0.12	1.88	1.82	0.047%
0.50	0.90	2.02	1.88	0.12	1.88	1.85	0.023%
0.75	0.96	2.02	1.87	0.13	1.87	1.89	0.009%
1.00	1.00	2.02	0.00	1.90	0.10	3.00	0.000%
1.25	1.03	2.02	1.86	0.14	1.86	1.96	0.006%
1.50	1.06	2.02	1.85	0.15	1.85	1.99	0.009%
1.75	1.09	2.02	1.84	0.16	1.84	2.02	0.012%
2.00	1.11	2.02	1.83	0.17	1.83	2.06	0.014%

Table 15. Global deviation GCI parameters for $\alpha=25\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.60	1.96	1.91	0.09	1.91	1.70	0.095%
0.25	0.74	1.96	1.91	0.09	1.91	1.70	0.044%
0.50	0.85	1.96	1.91	0.09	1.91	1.71	0.020%
0.75	0.93	1.96	1.90	0.10	1.90	1.72	0.007%
1.00	1.00	1.96	0.06	1.90	0.10	3.00	0.000%
1.25	1.06	1.96	1.90	0.10	1.90	1.74	0.004%
1.50	1.11	1.96	1.90	0.10	1.90	1.75	0.007%
1.75	1.15	1.96	1.89	0.11	1.89	1.77	0.009%
2.00	1.18	1.96	1.89	0.11	1.89	1.78	0.010%

Table 16. Global deviation GCI parameters for $\alpha=35\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.48	2.01	1.95	0.05	1.95	1.45	0.090%
0.25	0.65	2.01	1.94	0.06	1.94	1.51	0.040%
0.50	0.79	2.01	1.93	0.07	1.93	1.57	0.019%
0.75	0.90	2.01	1.92	0.08	1.92	1.64	0.007%
1.00	1.00	2.01	0.09	1.90	0.10	3.00	0.000%
1.25	1.08	2.01	1.89	0.11	1.89	1.78	0.005%
1.50	1.15	2.01	1.88	0.12	1.88	1.84	0.008%
1.75	1.21	2.01	1.87	0.13	1.87	1.91	0.010%
2.00	1.26	2.01	1.85	0.15	1.85	1.98	0.012%

Table 17. Global deviation GCI parameters for $\alpha=45\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.37	1.97	2.09	0.09	1.91	1.67	0.115%
0.25	0.57	1.97	2.05	0.05	1.95	1.45	0.038%
0.50	0.74	1.97	2.02	0.02	1.98	1.22	0.013%
0.75	0.88	1.97	1.98	0.02	1.98	1.21	0.005%
1.00	1.00	1.97	0.00	1.90	0.10	3.00	0.000%
1.25	1.11	1.97	1.92	0.08	1.92	1.61	0.004%
1.50	1.20	1.97	1.89	0.11	1.89	1.79	0.008%
1.75	1.28	1.97	1.86	0.14	1.86	1.94	0.011%
2.00	1.35	1.97	1.83	0.17	1.83	2.08	0.013%

Table 18. Global deviation GCI parameters for $\alpha=55\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.28	1.95	1.95	0.05	1.95	1.44	0.130%
0.25	0.50	1.95	1.96	0.04	1.96	1.35	0.040%
0.50	0.69	1.95	1.97	0.03	1.97	1.30	0.016%
0.75	0.85	1.95	1.98	0.02	1.98	1.26	0.005%
1.00	1.00	1.95	0.00	1.90	0.10	3.00	0.000%
1.25	1.13	1.95	1.99	0.01	1.99	1.16	0.003%
1.50	1.25	1.95	2.00	0.00	2.00	1.11	0.004%
1.75	1.35	1.95	2.01	0.01	1.99	1.15	0.006%
2.00	1.45	1.95	2.01	0.01	1.99	1.20	0.007%

Table 19. Global deviation GCI parameters for $\alpha=65\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.18	1.98	2.02	0.02	1.98	1.26	0.168%
0.25	0.43	1.98	2.03	0.03	1.97	1.34	0.044%
0.50	0.64	1.98	2.03	0.03	1.97	1.29	0.016%
0.75	0.83	1.98	2.02	0.02	1.98	1.24	0.005%
1.00	1.00	1.98	0.10	1.90	0.10	3.00	0.000%
1.25	1.16	1.98	2.02	0.02	1.98	1.21	0.003%
1.50	1.30	1.98	2.02	0.02	1.98	1.25	0.005%
1.75	1.43	1.98	2.03	0.03	1.97	1.33	0.007%
2.00	1.56	1.98	2.05	0.05	1.95	1.42	0.009%

Table 20. Global deviation GCI parameters for $\alpha=70\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.13	1.96	1.93	0.07	1.93	1.58	0.333%
0.25	0.40	1.96	2.03	0.03	1.97	1.31	0.048%
0.50	0.62	1.96	2.06	0.06	1.94	1.51	0.021%
0.75	0.82	1.96	2.08	0.08	1.92	1.61	0.008%
1.00	1.00	1.96	---	---	---	---	---
1.25	1.17	1.96	2.11	0.11	1.89	1.79	0.005%
1.50	1.33	1.96	2.13	0.13	1.87	1.90	0.009%
1.75	1.48	1.96	2.16	0.16	1.84	2.02	0.013%
2.00	1.62	1.96	2.19	0.19	1.81	2.13	0.016%

Table 21. Global deviation GCI parameters for $\alpha=75\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.08	1.97	2.10	0.10	1.90	1.76	0.704%
0.25	0.36	1.97	2.14	0.14	1.86	1.95	0.084%
0.50	0.60	1.97	2.17	0.17	1.83	2.05	0.031%
0.75	0.81	1.97	2.18	0.18	1.82	2.09	0.011%
1.00	1.00	1.97	0.00	1.90	0.10	3.00	0.000%
1.25	1.18	1.97	2.18	0.18	1.82	2.12	0.007%
1.50	1.35	1.97	2.19	0.19	1.81	2.15	0.011%
1.75	1.52	1.97	2.20	0.20	1.80	2.18	0.015%
2.00	1.68	1.97	2.21	0.21	1.79	2.22	0.018%

Table 22. Global deviation GCI parameters for $\alpha=78\%$

k_2/k_1	k^*	$r_{1,2}$	p_t	Δp	p^*	FS	U_{num}
0.00	0.03	1.94	2.05	0.05	1.95	1.42	3.083%
0.25	0.34	1.94	2.13	0.13	1.87	1.90	0.088%
0.50	0.58	1.94	2.17	0.17	1.83	2.07	0.031%
0.75	0.80	1.94	2.19	0.19	1.81	2.15	0.011%
1.00	1.00	1.94	0.56	1.44	0.56	3.00	0.000%
1.25	1.19	1.94	2.21	0.21	1.79	2.22	0.007%
1.50	1.37	1.94	2.22	0.22	1.78	2.25	0.012%
1.75	1.55	1.94	2.23	0.23	1.77	2.28	0.016%
2.00	1.72	1.94	2.24	0.24	1.76	2.31	0.019%

5.3 Characteristic Trends

Compiling all the k^* data from Table 12 through Table 22 shows a family of similar curves—roughly quadratic in form—relating k^* to both α and k_2/k_1 , as illustrated in Figure 10.

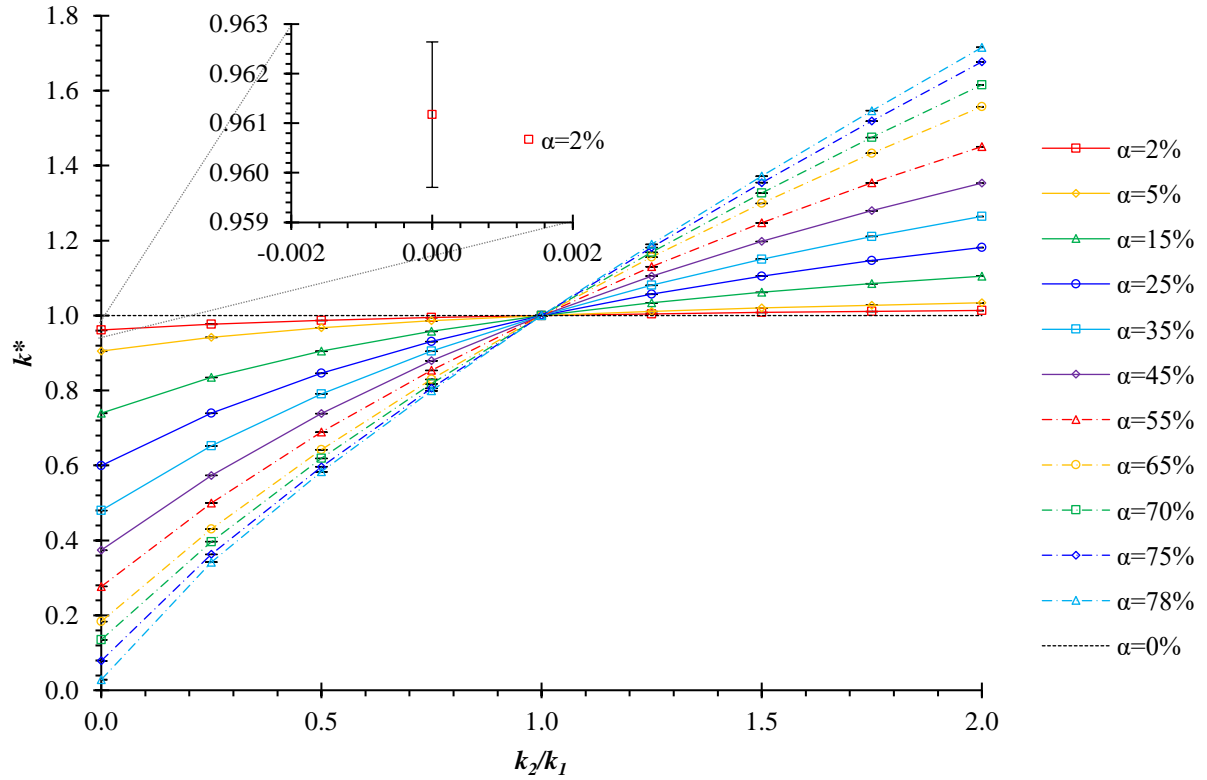


Figure 10. Dimensionless effective thermal conductivity characteristic curves

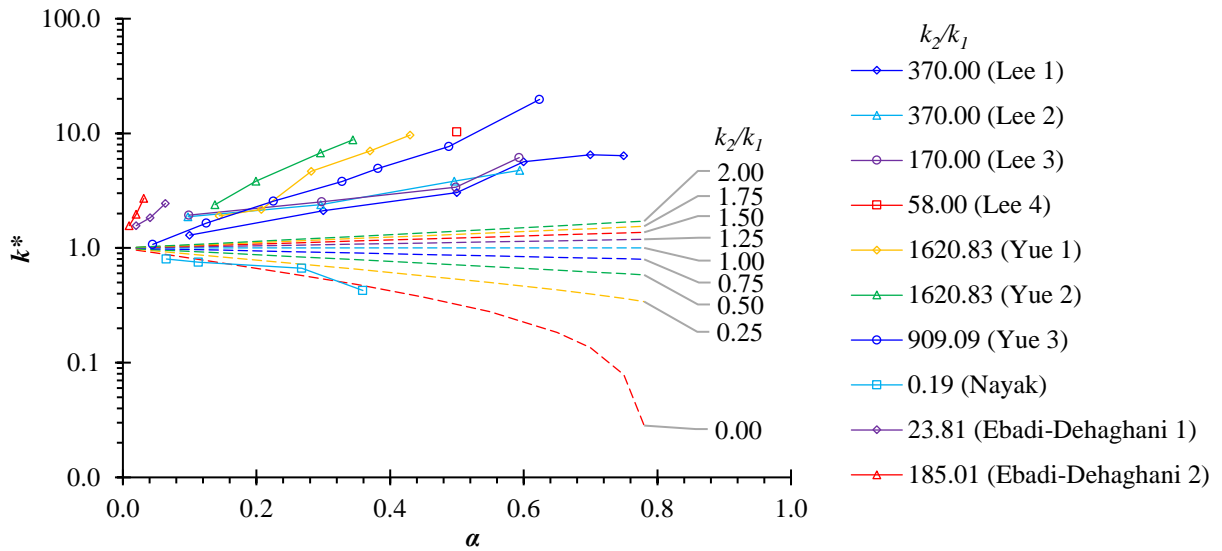


Figure 11. Overlay of empirical non-TBC data with numerical results for various k_2/k_1 systems

In Figure 10, it is clear that an increase in the filler material thermal conductivity leads to an increase in overall effective thermal conductivity of the composite material. Also, it is apparent from Figure 10 that an increase in filler size amplifies the relative effect of k_2 on k^* . All curves intersect at $(k_2/k_1=1.0, k^*=1.0)$, due to the fact that the condition of $k_2/k_1=1.0$ implies that the composite material behaves as a single homogeneous material from the perspective of steady-state thermal transport. Note that error bars on the k^* values are indistinguishable due to their relatively low values. An inset magnification of a single data point with a relatively high numerical uncertainty estimate—relative to this study—is given to illustrate the low level of estimated numerical uncertainty.

Figure 11 presents the computational results in an alternative manner, organizing trends by k_2/k_1 ratios and as a function of α . A set of empirical data measurements found in the literature is also plotted with computational data from this study [32-35]. Note that for all but one of the empirical data sets shown, the k_2/k_1 ratios are much higher than what was analyzed computationally in this study. For some of the data sets, k^* values and/or α values were not directly reported by the original authors and had to be computed for comparative purposes in this work. This was done with the data presented from [35], which appears suspect, not following the general trends of the rest of the empirical and computational data sets. Due process was given in attempting to find TBC-related data in the open literature that could be translated into the α - k_2/k_1 data format of this work, but virtually no such data could be found.

6 Conclusion

This work has presented a numerical approach to capturing the effective thermal transport behavior of a two-component composite material with respect to filler thermal conductivity and filler size. It was shown that the employed FEM code was approximately

second order accurate using the MMS code verification approach, and numerical uncertainty was estimated using the global deviation estimator GCI method. Trends given here show the relationship between fill fraction, component thermal conductivities, and overall effective thermal conductivity of the multi-component medium. A range of fill fractions from 2% to 78% were analyzed, using filler-to-matrix thermal conductivity ratios ranging from 0 to 2. Numerical uncertainty on the computationally determined dimensionless effective thermal conductivity was estimate to be extremely low, with most being less than 0.1%. The computational solutions were plotted against a range of non-TBC empirical data sets, most of which had k_2/k_1 ratios well above the max of 2 that was used in this study. Future work should seek to expand the scope of this analysis to k_2/k_1 ratios on the order of 1,000 to compare with open literature data. However, for more accessible evaluation of computational results with respect to empirical TBC material thermal performance, TBC effective thermal conductivity would need to be more readily available in the open literature, including information on the structural composition (e.g., fill fraction) and thermophysical properties of the components (e.g., thermal conductivity).

References

- 1 Ibrahim, T. K. and Rahman, M. M., 2013, "Study on effective parameter of the triple-pressure reheat combined cycle performance," *Thermal Science*, **17**(2), pp. 497-508.
- 2 Nayak, J. and Mahto, D., 2014, "Effect of Gas Turbine Inlet Temperature on Combined Cycle Performance," *International Conference on Recent Innovations in Engineering & Technology*.
- 3 Fathi, N., McDaniel, P., Forsberg, C., and de Oliveira, C., 2018, "Power Cycle Assessment of Nuclear Systems, Providing Energy Storage for Low Carbon Grids," *Journal of Nuclear Engineering and Radiation Science*, **4**(2), 020911.
- 4 Hunter, I., Daleo, J., Wilson, J., and Ellison, K., 1999, "Analysis of Hot Section Failures on Gas Turbines in Process Plant Service," *Proceedings of the 28th Turbomachinery Symposium*, **28**, pp. 9-20.

- 5 Salehnasab, B., Poursaeidi, E., Mortazavi, S. A., and Farokhian, G. H, 2016, "Hot corrosion failure in the first stage nozzle of a gas turbine engine," *Engineering Failure Analysis*, **60**, pp. 316-325.
- 6 Lai, G. Y., 2007, *High-Temperature Corrosion and Materials Applications*, ASM International, Novelty, OH.
- 7 Rao, A. D, 2012, *Combined Cycle Systems for Near-Zero Emission Power Generation*, Woodhead Publishing Limited, Cambridge, UK.
- 8 Ma, W., Li, X., Meng, X, Xue, Y, Bai, Y, Chen, W., and Dong, 2018, "Microstructure and Thermophysical Properties of SrZrO₃ Thermal Barrier Coating Prepared by Solution Precursor Plasma Spray," *Journal of Thermal Spray Technology*, **27**(7), pp. 1056-1063.
- 9 McCay, M. H., Hsu, P.-f., Croy, D. E., Moreno, D., and Zhang, M., 2017, "The Fabrication, High Heat Flux Testing, and Failure Analysis of Thermal Barrier Coatings for Power Generation Gas Turbines," *Turbo Expo: Power for Land, Sea, and Air*, **6**():V006T24A008.
- 10 Cernuschi, F., Bison, P., Mack, D. E., Merlini, M., Boldrini, S., Marchionna, S., Capelli, S., Concari, S., Famengo, A., Moscatelli, A., and Stamm, W., 2018, "Thermo-physical properties of as deposited and aged thermal barrier coatings (TBC) for gas turbines: State-of-the art and advanced TBCs," *Journal of the European Ceramic Society*, **38**(11), pp. 3945-3961.
- 11 Padture, N. P., Gell, M., and Jordan, E. H., 2002, "Thermal Barrier Coatings for Gas-Turbine Engine Applications," *Science*, **296**(5566), pp. 280-284.
- 12 Zhu, W., Cai, X. N., Yang, L., Xia, J., Zhou, Y. C., and Pi, Z. P., 2019, "The evolution of pores in thermal barrier coatings under volcanic ash corrosion using X-ray computed tomography," *Surface and Coatings Technology*, **357**, pp. 372-378.
- 13 Gu, S., Lu, T. J., Hass, D. D., and Wadley, H. N. G., 2001, "Thermal conductivity of zirconia coatings with zig-zag pore microstructures," *Acta Materialia*, **49**(13), pp. 2539-2547.
- 14 Flores Renteria, A., Saruhan, B., Schulz, U., Raetzer-Scheibe, H.-J., Haug, J., and Wiedenmann, A., 2006, "Effect of morphology on thermal conductivity of EB-PVD PYSZ TBCs," *Surface and Coatings Technology*, **201**(6), pp. 2611-2620.
- 15 Maxwell, J. C., 1873, *A Treatise on Electricity and Magnetism*, **1**, Oxford at the Clarendon Press.
- 16 Burger, H. C., 1915, "Das Iertvermogen verdumter mischkristallfreier Ionsungen," *Phys. Zs* **20**, pp. 73-76.
- 17 Landauer, R., 1952, "The Electrical Resistance of Binary Metallic Mixtures," *Journal of Applied Physics*, **23**(7), pp. 779-184.
- 18 Nielsen, L. E., 1974, "The Thermal and Electrical Conductivity of Two-Phase Systems," *Industrial and Engineering Chemistry Fundamentals*, **13**(1), pp. 17-20.

- 19 Krischer, O. and Kroell, K., 1956, "Die wissenschaftlichen Grundlagen der Trocknungstechnik," *Trocknungstechnik*, **1**, p. 48.
- 20 de Vries, D. A., 1952, *Mededlingen van de Landbouwhogeschool te Wageningen*.
- 21 Krupiczka, R., 1967, "Analysis of thermal conductivity in granular materials," *International Chemical Engineering*, **7**, pp. 122–144.
- 22 Davis, L. C. and Artz, B. E., 1995, "Thermal conductivity of metal-matrix composites," *Journal of Applied Physics*, **77**, pp. 4954.
- 23 Ganapathy, D., Singh, K., Phelan, P. E., and Prasher, R., 2005, "An effective unit cell approach to compute the thermal conductivity of composites with cylindrical particles," *Journal of Heat Transfer*, **127**(6), pp. 553-559.
- 24 Moghaddam, H. A. and Mertiny, P., 2018, "Stochastic Finite Element Analysis Framework for Modeling Thermal Conductivity of Particulate Modified Polymer Composites," *Results in Physics*, **11**, pp. 905-914.
- 25 Xu, Y., Yamazaki, M., Wang, H., and Yagi, K., 2006, "Development of an internet system for composite design and thermophysical property prediction," *Materials Transactions*, **47**(8), pp. 1882-1885.
- 26 Irick, K. and Fathi, N., 2018, "Thermal Response of Open-Cell Porous Materials: A Numerical Study and Model Assessment," *ASME 2018 Verification and Validation Symposium ()*:V001T03A002.
- 27 Geuzaine, C. and Remacle, J.-F., 2009, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering* **79**(11), pp. 1309-1331.
- 28 Oberkampf, W. L. and Roy, C. J., 2010, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge, UK.
- 29 Roache, P. J., 2009, *Fundamentals of Verification and Validation*, Hermosa Publishers, Socorro, NM.
- 30 Celik, I. B., Ghia, U., Roache, P. J., Freitas, C. J., Coleman, H., and Raad, P. E., 2008, "Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications," *Journal of Fluids Engineering*, **130**(7), pp. 0780011-0780014.
- 31 Phillips, T. S. and Roy, C. J., 2016 "A New Extrapolation-Based Uncertainty Estimator for Computational Fluid Dynamics," *Journal of Verification, Validation and Uncertainty Quantification*, **1**(4), pp. 041006-1, 041006-13
- 32 Lee, G.-W., Park, M., Kim, J., Lee, J. I., and Yoon, H. G., 2006, "Enhanced thermal conductivity of polymer composites filled with hybrid filler," *Composite Part A: Applied Science and Manufacturing*, **37**(5), pp. 727-734.
- 33 Yue, C., Zhang, Y., Hu, Z., Liu, J., and Cheng, Z., 2010, "Modling of the effective thermal conductivity of composite materials with FEM based on resistor networks approach," *Microsystem Technologies*, **16**, pp. 633-639.

- 34 Nayak, R., Tarkes, D. P., and Satapathy, A., 2010, "A computational and experimental investigation on thermal conductivity of particle reinforced epoxy composites," *Computational Materials Science*, **48**(3), pp. 576-581.
- 35 Ebadi-Dehaghani, H. and Nazempour, M., 2012, "Thermal Conductivity of Nanoparticles Filled Polymers," *Smart Nanoparticles Technology*, IntechOpen.

CHAPTER 4: HIGH-FIDELITY CALCULATION OF EFFECTIVE THERMAL RESPONSE OF COMPOSITE MEDIA WITH HEAT GENERATION SOURCE⁴

Abstract

As heterogeneity in a structure increases, the complexity of conductive heat transfer typically increases as well. Any discontinuity of material property—especially thermal conductivity—would warrant a thorough analysis to evaluate the thermal behavior of the system of interest. In nuclear fuel assemblies, heterogeneous thermal conditions are crucial to heat transfer physics. The thermal behavior within the assemblies is governed significantly by the heterogeneous thermal conditions at both the system and component levels, representing heterogeneity at various scales. Multi-component, solid-phase systems with a source of internal thermal heat generation is an important example of a heterogeneous system in nuclear fuel assemblies. A variety of materials have been used as nuclear fuels, the most conventional of which is uranium dioxide, UO_2 . UO_2 exhibits suitable chemical and irradiation tolerances in thermal reactors. Despite the robust nature of UO_2 in chemical and radiative aspects, the low thermal conductivity of porous UO_2 can open up challenges with respect to thermal conditions. Thus, a crucial, ongoing topic of investigation in the industry is the feasibility of enhancing the thermal conductivity of oxide fuels by adding a high-conductivity secondary solid component. Undoubtedly, long-term, stable development of clean nuclear energy would depend on research and development of innovative reactor designs and fuel systems. A unit

⁴ This chapter will be under review for publication in an **Elsevier Journal** in late 2020. The content in this chapter was reproduced directly from the journal version, differing only in style formatting. A less comprehensive version of this chapter was published in *Proceedings of the ASME 2019 Verification and Validation Symposium* under the title “Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis.”

cell of a composite that represents a fuel matrix cell can be used to investigate thermal responses in nuclear fuel. Such information would help to develop the next generation of nuclear fuel and understand potential performance enhancements. The goal of this article is to present an evaluation of a high-fidelity computational model response of heterogeneous materials with heat generation in circular fillers. Two-dimensional, steady-state systems were defined with a circular, heat-generating filler centered in a unit-cell domain. A Galerkin finite element method (FEM) program was written in Fortran to solve the heat equation on an unstructured triangular mesh of the composite systems. This paper presents a study on the effects of a heat-generating filler's relative size and thermal conductivity on effective thermal conductance, G_{eff} , within a heterogeneous material. The method of manufactured solutions (MMS) was used to perform code verification, showing a second-order accurate numerical implementation. A global deviation grid convergence index (GCI) method was used to perform solution verification to assess solution convergence and estimate numerical uncertainty, U_{num} , on computationally-determined system thermal conductance. Trend results are presented, showing variable response in G_{eff} to filler size and thermal conductivity.

Nomenclature

A	= area
α	= fill fraction
c	= coefficient
Γ	= domain boundary
d	= depth
ε	= error
FS	= factor of safety

G	= thermal conductance
\mathbf{G}	= conductance matrix
h	= characteristic mesh size
H	= mesh number
k	= thermal conductivity
\mathbf{K}	= thermal conductivity matrix
L	= cell length
\mathbf{n}	= unit normal vector
N	= total quantity
\mathbf{N}	= shape function vector
\mathbf{p}	= direction vector
\mathbf{P}	= heat load vector
p	= order of accuracy
q	= heat
\mathbf{q}	= directional heat flux vector
r	= mesh size ratio
R	= radius
ρ	= residual
$\boldsymbol{\rho}$	= residual vector
S	= energy source
t	= index
T	= temperature
\mathbf{T}	= temperature vector

U = uncertainty
 w = weighting function
 \mathbf{w} = weighting function vector
 W = cell width
 x = x-coordinate
 y = y-coordinate
 Ω = domain

Subscripts

α = filler
 C = cold
 eff = effective
 f = formal
 H = mesh number
 L_∞ = L_∞ norm
 MMS = manufactured solution
 n = normal
 num = numerical
 O = observed
 RMS = root mean square
 $slab$ = slab equivalent
 t = triangle, transcendental
 v = vertex
 x = x-direction

y = y-direction

I = matrix

2 = filler

Superscripts

$*$ = dimensionless, average

$'$ = per unit length

$''$ = per unit area

$'''$ = per unit volume

1 Introduction

“Heterogeneity” is a broad—and relatively subjective—classification of materials, where a macroscale material (on a relative scale) is considered to be comprised of multiple, different components. The components comprising the heterogeneous material may be different fundamental materials, different orientations of materials, similar or different materials separated by boundaries, and/or different material phases. A special case of heterogeneous materials is a porous material, where voids (i.e., locales of material absence) exist within the bulk material. The foregoing list is not exhaustive, but what should be noted is the subjectivity of the classification “heterogeneous.” Heterogeneity—as opposed to homogeneity—can be determined from different perspectives such as from a length scale or from a physics perspective. For example, a continuous volume of some metallic solid may be considered homogeneous for most practical engineering purposes, such as to determine stress-strain relationships or heat transfer. In contrast, the same continuous volume of metallic solid may be considered heterogeneous when one considers the presence of distributed impurities or

grain boundaries throughout the volume, perhaps affecting crack propagation under stress or material evolution during heat treatment.

Nuclear fuels and fuel assemblies represent an important application of multi-scale material heterogeneity. Both the heat-generating fuel itself and the fuel assembly structure are heterogeneous or composite in nature [1,2]. An example of such a condition is illustrated by the advanced gas cooled reactor test experiment number two—illustrated in Figure 1—where the system is comprised of various components, including components made of heterogeneous materials [1]. Heterogeneous thermal conditions are crucial to heat transfer in nuclear fuel assemblies because the thermal behavior within the assemblies is governed significantly by the heterogeneous thermal conditions at both the system and component levels [3,4]. Having a better understanding of the thermal response of the unit cell of a composite that represents a fuel matrix cell would help to develop the next generation of nuclear fuel and understand potential performance enhancements.

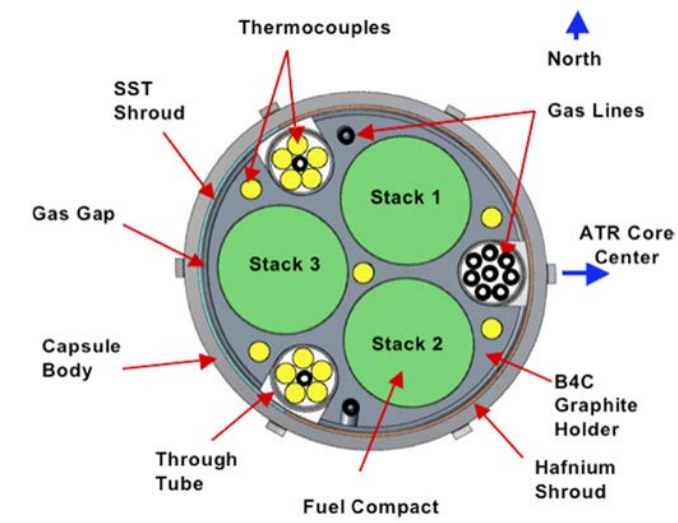


Figure 1. Advanced gas cooled reactor test experiment number two capsule cross-section [1]

The most conventional fuel material is uranium dioxide, UO_2 , because it exhibits suitable chemical and irradiation tolerances in thermal reactors. However, UO_2 's relatively low thermal conductivity can prove challenging from a thermal perspective. UO_2 is a porous media, where pore size, pore geometry, and pore distribution can have an effect on the thermal behavior of the fuel [5-11]. Likewise, if the UO_2 fuel were embedded in some non-heat-generating conductive matrix, the system would also be heterogeneous. Heat transport in such a critical system as nuclear fuel assemblies is a crucial to the advancement towards next-generation nuclear energy system development [12,13]. Failure modes and system performance in nuclear fuel assemblies depend on the thermal response and thermal tolerances of components and assemblies [3].

This study is dedicated to understanding the effects of composite material response by providing an assessment of a high-fidelity computational model of heterogeneous materials with heat generation in circular fillers. The finite element method (FEM) was used to generate results for computing. The two-dimensional response assessment provided includes code and solution verification efforts to offer credibility evidence towards the veracity of the computational data as well as estimate the numerical uncertainty, U_{num} , in the results. Critical systems, such as those mentioned above, should validate the results presented in this work against empirical data.

Complex engineering systems, such as might be described by heterogeneous materials, typically require the use of computational analysis tools and numerical methods to gain insight to details of the system behavior. At a material level, empirical data can be extremely difficult—and sometimes impossible—to obtain that give meaningful insight into the impactful characteristics of heterogeneity in material thermal response. When using

computational tools, effort should be made to ensure the veracity of the results. Systematic approaches have been developed to provide credibility evidence for the means of achieving the numerical results and for the numerical results directly. The execution of the credibility assessment approaches is captured by rigorous verification and validation (V&V) processes, some of which are described in detail in this work and are used to foundationally support the findings presented here. Although the intent of this work is not to present novel V&V approaches, the focus on V&V presented here serves two significant purposes. First, rigor is expected to be given to describing and analyzing the physical processes, data collection methods, and measurement uncertainty quantification (UQ) used in obtaining empirical scientific data. The same rigor that is expected of empirical data acquisition should be expected when obtaining computational scientific data, especially when little or no empirical data is available and—even more importantly—when the computational data is used to predict system behavior. Second, V&V is generally undervalued, underused, and misunderstood (if recognized at all) in engineering practice but should be more universally implemented. With the ubiquity of computational analysis, the focus on V&V in this work aims to emphasize the need to bring V&V to the forefront of engineering practice, and a more thorough explanation of the process used to perform V&V must be given.

2 System Description

Heterogeneous systems were modeled for this study using a solid circular filler material geometry embedded in a solid square matrix. As illustrated in Figure 2, a half-cell composite structure system was generated to represent a square bulk domain of width W in the x -direction and length L in the y -direction with a centered filler of radius R . Domain symmetry was employed, cutting the computational domain down to width $W/2$. The bulk matrix material and

the circular filler have thermal conductivity k_1 and k_2 , respectively. At the interface between the filler and the matrix, intimate thermal contact is assumed, disregarding any form of interface thermal resistance. The system is defined with Dirichlet boundary conditions of cold temperature, T_C , along the domain negative x -boundary and the domain negative and positive y -boundaries. The positive x -boundary has an enforced periodic boundary condition, and the circular filler has uniform volumetric heat generation, q''' . Let T_a be the resultant temperature at the filler center, with coordinates $(W/2, L/2)$.

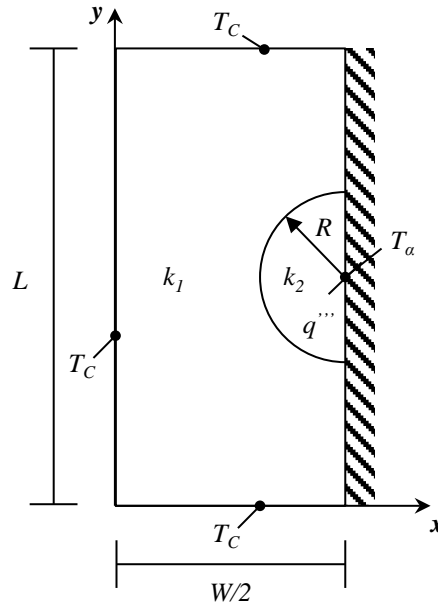


Figure 2. Unit half-cell composite structure

For this study, fill fraction, α , and filler-to-matrix thermal conductivity ratio, k_2/k_1 , effects on thermal behavior were analyzed. Fill fraction is computed as

$$\alpha = \frac{\pi R^2}{WL}. \quad (1)$$

The thermal performance of the system is characterized by the effective thermal conductance of the cell system, G_{eff} . If T_α is the resultant temperature at the filler center, then

$$G_{eff} = \frac{q''' \pi R^2 d}{2(T_\alpha - T_C)}, \quad (2)$$

where d is the depth of the domain—unit length in this analysis. A representative slab conductance, G_{slab} , can also be established to characterize the system, given a composite slab system—as shown in Figure 3—equivalent in material fractions to a respective composite structure from Figure 2. In the slab system one-dimensional heat flow is assumed across a laminate system with the same thermal conductivities as in the filler system, where q' is some heat load per unit length. By the well-known fact that

$$G_{slab} = \frac{2L}{W} \left(\frac{(1-\alpha)}{k_1} + \frac{\alpha}{k_2} \right)^{-1}, \quad (3)$$

G_{eff} can be normalized with G_{slab} to yield a dimensionless system conductance, G^* , computed by

$$G^* = \frac{G_{eff}}{G_{slab}}. \quad (4)$$

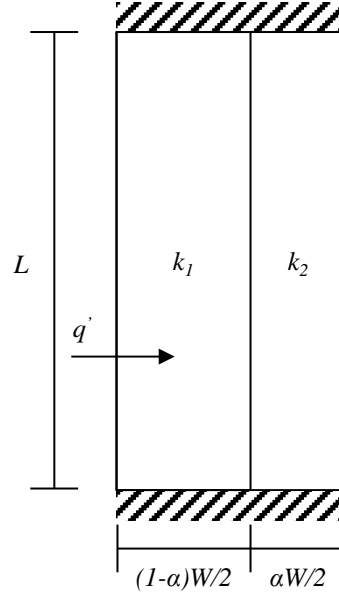


Figure 3. Notional equivalent composite slab system

17 different α sizes were analyzed to include 2%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 65%, 70%, and 78% fill fractions. Filler-to-matrix thermal conductivity ratios ranging from 0.01 to 1,000.00 were also used in this work.

3 Discretization

The temperature distribution in the computational domain was determined by using a FEM program that solves the steady-state, two-dimension heat equation,

$$k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + S = 0, \quad (5)$$

throughout the composite system, where k is thermal conductivity, T is the temperature distribution, and S is the generic volumetric energy source function. The remainder of this section describes the discretization of the domain and the governing partial differential equation (PDE) used for solving the temperature distribution in this work.

3.1 Domain

Open-source *Gmsh* software was used to discretize the solid domain into unstructured triangular meshes [14], where a single triangle mesh element is illustrated in Figure 4. The t^{th} triangle has area A_t and is defined by three vertices, where the i^{th} vertex of the t^{th} triangle has temperature $T_{t,i}$ and coordinates $(x_{t,i}, y_{t,i})$. Systematic mesh refinement was used to facilitate code and solution verification efforts for this work. Figure 5a through Figure 5e show an example of the systematic mesh refinement for a single fill fraction, 40%, from the coarsest to the finest mesh, respectively, where five different mesh size levels were used. Figure 6a through Figure 6j and Figure 7a through Figure 7g show the finest meshes used for all the systems in this study.

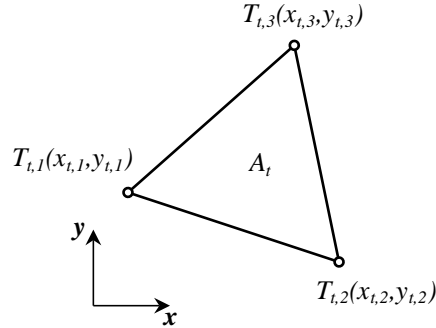


Figure 4. Notional linear triangle element

The mesh number, H , is used to describe the mesh refinement level of each system domain, where $H=1$ is the finest mesh and $H=5$ is the coarsest mesh. The characteristic mesh size, h_H , of mesh H , is given to be

$$h_H = \sqrt{\frac{1}{N_{H,t}} \sum_{t=1}^{N_{H,t}} A_t}, \quad (6)$$

where $N_{H,t}$ is the total number of triangles in the H^{th} mesh. Characteristic mesh sizes for meshes $H=5$ through $H=1$ used in this study were approximately 0.11, 0.07, 0.04, 0.02, and 0.01. Note that mesh sizes were approximately halved for each successive refinement.

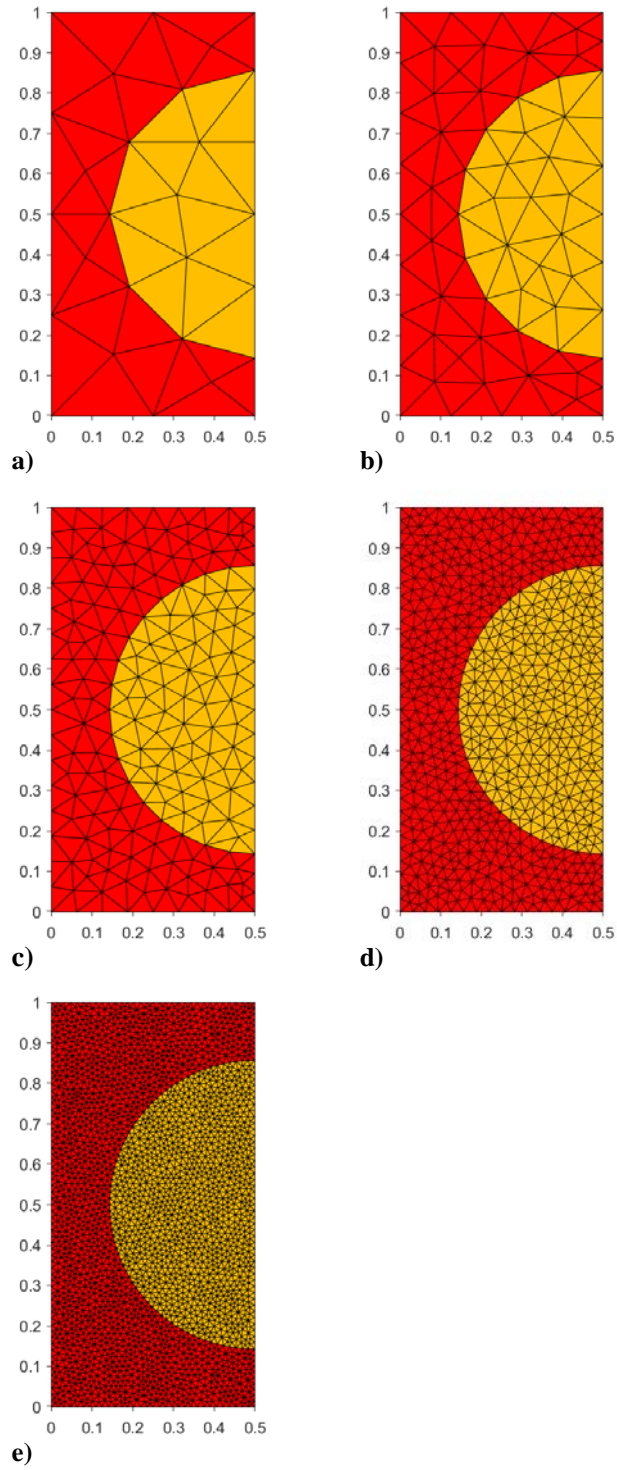


Figure 5. Systematic mesh refinement for $\alpha=40\%$

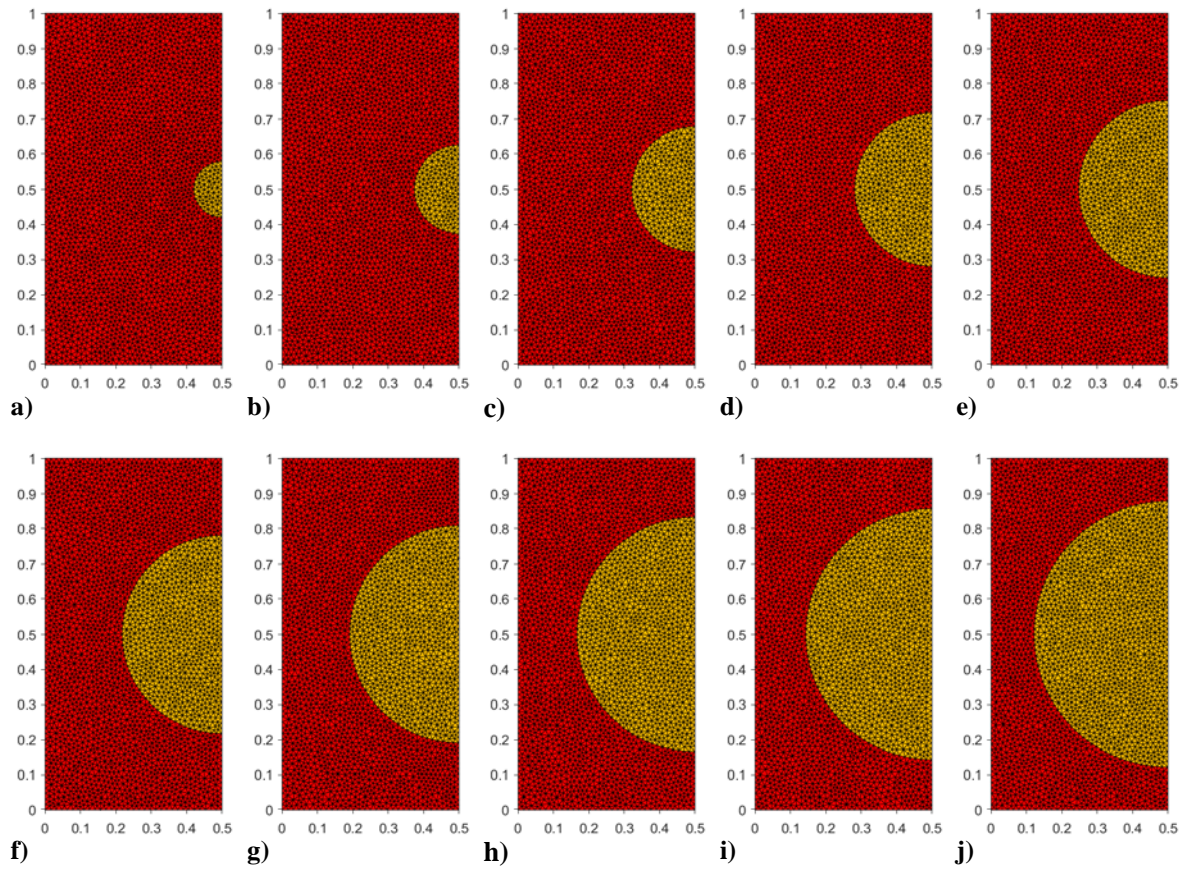


Figure 6. Finest mesh for fill fraction systems a) 2%, b) 5%, c) 10%, d) 15%, e) 20%, f) 25%, g) 30%, h) 35%, i) 40%, and j) 45%

“Hete efficiency.

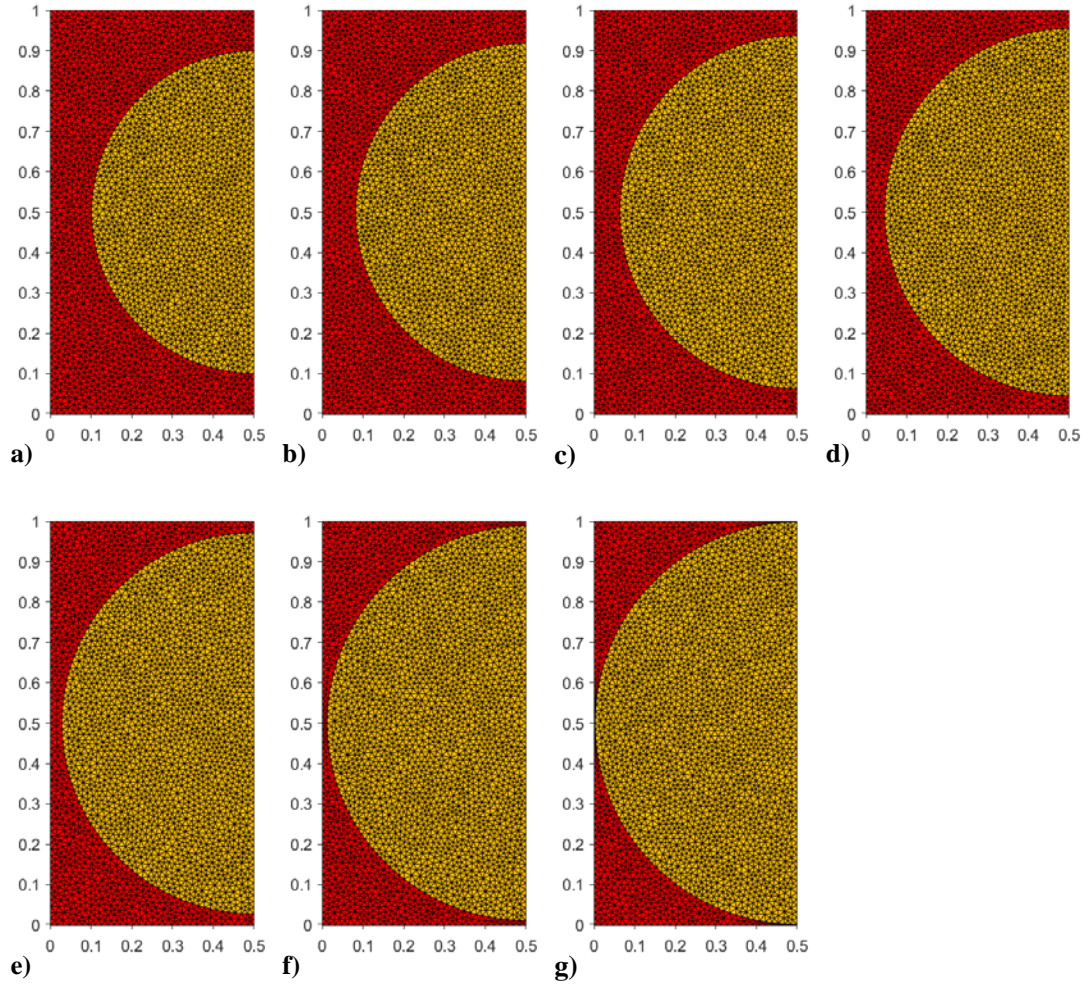


Figure 7. Finest mesh for fill fraction systems a) 50%, b) 55%, c) 60%, d) 65%, e) 70%, f) 75%, and g) 78%

3.2 Partial Differential Equation

A Galerkin FEM was written in Fortran to solve Equation (5) throughout the composite computational domain. Equation (5) can be solved over each finite element, individually, using a discretized PDE matrix equation. The thermal conductivities and temperature gradients from Equation (5) can be cast in matrix form as

$$\mathbf{K} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \quad \text{and} \quad \nabla T = \begin{Bmatrix} \partial T / \partial x \\ \partial T / \partial y \end{Bmatrix}, \quad (7)$$

where \mathbf{K} is the thermal conductivity tensor. For the system at hand, isotropic thermal conductivity is used for both the matrix and filler materials, such that k_x and k_y are the same. From Fourier's law, it follows that the heat flux vector, \mathbf{q} , is comprised of x -directional and y -directional heat fluxes, q_x and q_y , respectively, and looks like

$$\mathbf{q} = \begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = -\mathbf{K}\nabla T. \quad (8)$$

Let Ω represent a domain, with Γ being the boundary of the domain. The surface normal unit vector, \mathbf{n} , is used to express the normal heat flux, q_n , leaving Ω , such that

$$q_n = \mathbf{q}^T \cdot \mathbf{n}. \quad (9)$$

Thus, the governing heat equation can be expressed equally as

$$S - \text{div}(\mathbf{q}) = S + \text{div}(\mathbf{K}\nabla T). \quad (10)$$

The Galerkin finite element approach—used in this work to solve the governing PDE—requires the use of some weighting function, w , over the PDE. Multiplying Equation (10) by w and integrating over Ω yields

$$\int_{\Omega} (\nabla w) \mathbf{K} \nabla T d\Omega = \int_{\Omega} w S d\Omega - \int_{\Gamma} w q_n d\Gamma. \quad (11)$$

For each triangle in the finite element domain, Equation (11) is true, and the Galerkin method prescribes that both w and T be interpolated across the finite element domain using the same row vector polynomial shape function, or interpolation function, \mathbf{N} . If a triangle element

has weighting function and temperature values defined at each of its three vertices as $w_{t,i}$ and $T_{t,i}$, respectively, for $i=1,2,3$ corresponding to each of the i^{th} vertices of the t^{th} triangle, then let

$$\mathbf{T} = \begin{Bmatrix} T_{t,1} \\ T_{t,2} \\ T_{t,3} \end{Bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{Bmatrix} w_{t,1} \\ w_{t,2} \\ w_{t,3} \end{Bmatrix}, \quad (12)$$

for a given triangle. Thus, the field variable and weight function within the bounds of the finite element are described similarly as

$$T(x, y) = \mathbf{N}\mathbf{T} \quad \text{and} \quad w(x, y) = \mathbf{N}\mathbf{w}. \quad (13)$$

By using matrix transpose rules, substituting the definitions of Equation (13) into Equation (11), and eliminating common constant terms, the discretized governing PDE solved over each element is

$$\int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \mathbf{T} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma. \quad (14)$$

Portions of Equation (14) can be further condensed and defined as the conductance matrix, \mathbf{G} , and the heat load matrix \mathbf{P} , where

$$\mathbf{G} = \int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \quad (15)$$

and

$$\mathbf{P} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma, \quad (16)$$

such that

$$\mathbf{GT} = \mathbf{P}. \quad (17)$$

In the numerical scheme, the imbalance of Equation (17) represents the residual energy balance at each vertex, ρ_i , where $\boldsymbol{\rho}$ is the vector of vertex residuals, and

$$\boldsymbol{\rho} = \mathbf{P} - \mathbf{GT}. \quad (18)$$

In this study, a linear triangle shape function is used to interpolate temperature and weight functions by

$$\mathbf{N} = \frac{1}{A_t} \begin{Bmatrix} x_{t,2}y_{t,3} - x_{t,3}y_{t,2} + x(y_{t,2} - y_{t,3}) + y(x_{t,3} - x_{t,2}) \\ x_{t,3}y_{t,1} - x_{t,1}y_{t,3} + x(y_{t,3} - y_{t,1}) + y(x_{t,1} - x_{t,3}) \\ x_{t,1}y_{t,2} - x_{t,2}y_{t,1} + x(y_{t,1} - y_{t,2}) + y(x_{t,2} - x_{t,1}) \end{Bmatrix}^T, \quad (19)$$

where the superscript T is the matrix transpose operator.

A conjugate gradient method was used in the numerical implementation to update the solution of Equation (17), driving energy residuals to zero. The general conjugate gradient scheme is iterative, where the initialized (iteration $i=0$) direction vector, \mathbf{p}_0 , is initialized as

$$\mathbf{p}_0 = \boldsymbol{\rho}_0. \quad (20)$$

For a given i^{th} iteration, starting at $i=0$, coefficient, $c_{1,i}$ is computed as

$$c_{1,i} = (\boldsymbol{\rho}_i^T \boldsymbol{\rho}_i) / (\mathbf{p}_i^T \mathbf{G} \mathbf{p}_i), \quad (21)$$

the solution is updated to

$$\mathbf{T}_{i+1} = \mathbf{T}_i + c_{1,i} \mathbf{p}_i, \quad (22)$$

and the residual is updated to

$$\boldsymbol{\rho}_{i+1} = \boldsymbol{\rho}_i - c_{1,i} \mathbf{G} \mathbf{p}_i. \quad (23)$$

Lastly, the coefficient, $c_{2,i}$, is calculated as

$$c_{2,i} = (\boldsymbol{\rho}_{i+1}^T \boldsymbol{\rho}_{i+1}) / (\boldsymbol{\rho}_i^T \boldsymbol{\rho}_i), \quad (24)$$

and the direction vector for the next iteration is updated as

$$\mathbf{p}_{i+1} = \boldsymbol{\rho}_{i+1} + c_{2,i} \mathbf{p}_i. \quad (25)$$

Iterations cease once sufficiently small residual conditions are met. For this study, the root mean square (RMS) of residuals, ρ_{RMS} , was used as the iteration exit criteria, where

$$\rho_{RMS} = \sqrt{\sum_{i=1}^{N_v} (\rho_i^2) / N_v}, \quad (26)$$

and a ρ_{RMS} value of 1.0×10^{-8} was used as the exit value. Although the RMS was used as exit criteria for this study, the L_∞ norm of residuals, $\rho_{L\infty}$, was computed and retained, where

$$\rho_{L\infty} = \max_{1 \leq i \leq N_v} |\rho_i|. \quad (27)$$

4 Verification

The second order accurate Galerkin FEM and a basic conjugate gradient iterative solver scheme were implemented in Fortran to update solutions to the heat equation until the RMS of the heat equation residuals across the domain fell below 10^{-8} . A complimentary description of the custom code used in this work is given in [15] and [16].

Verification is generally defined as the processes executed to assess the correctness of a numerical method in its ability to solve the defined mathematical problem. Verification can be broken into two steps: code verification and solution verification. Code verification is the process of assessing how well the implemented method (program, software, code, script, etc.) solves the governing mathematical equations—typically some set of partial differential equations. Solution verification is the process of assessing how well the implemented method solves the mathematical problem in a defined computational system (i.e., the computational model’s domain). Ultimately, verification is intended to be a process for quantifying the uncertainty in the numerical solution of the model and providing credibility evidence towards the confident use of the model for making predictions against real-world systems [17-19]. Both code and solution verification were performed in this study as foundational elements for establishing credibility to the provided results, especially in light of fact that personal code was used to generate the numerical results. The following subsections describe the methods used.

4.1 Code Verification

The method of manufactured solutions (MMS) is a reliable approach to vet code performance and verify correct implementation of numerical methods for solving a given PDE or set of PDEs [18,20]. MMS was used in this study to assess the observed order of accuracy, $p_{O,H}$, in light of the expected second order formal order of accuracy, p_f , of the employed FEM. Order of accuracy refers to the rate at which the error of the computational solution with respect to the true mathematical solution decreases with refinement of mesh (i.e., reduction in mesh size). For example, the solution error of a second order accurate method would be expected to be quartered if the size of the elements in the discretized domain were halved. Thus, the error reduces twice as fast—second order—as the mesh size reduction.

Traditionally, numerical methods solve a governing PDE over a prescribed domain with defined boundary conditions and source terms in order to determine a resultant field variable profile. However, exact or analytical solutions are rarely available for real systems of interest against which to evaluate the error in the numerical method's ability to solve the governing PDE. Yet, in MMS the user prescribes some PDE solution, in this case, T_{MMS} , on which the governing PDE can operate. Typically, a function of trigonometric or exponential form is chosen for smoothness and infinite differentiability. In this study, T_{MMS} was selected to be

$$T_{MMS}(x, y) = 25 \cos(\pi(2x - 1)) \sin(\pi y + 0.75) + 50, \quad (28)$$

where the second spatial derivatives are

$$\begin{aligned} \frac{\partial^2 T_{MMS}}{\partial x^2} &= -100\pi^2 \cos(\pi(2x - 1)) \sin(\pi y + 0.75) \\ \frac{\partial^2 T_{MMS}}{\partial y^2} &= -25\pi^2 \cos(\pi(2x - 1)) \sin(\pi y + 0.75) \end{aligned} \quad (29)$$

By operation on T_{MMS} with Equation (5), the manufactured source term, S_{MMS} , was solved for and found to be

$$S_{MMS}(x, y) = 125\pi^2 k \cos(\pi(2x - 1)) \sin(\pi y + 0.75). \quad (30)$$

By applying the manufactured source term across the computational system domain and applying appropriate boundary conditions to the system that match the manufactured temperature solution, the finite element solution is employed to solve for the computational temperature solution. The error, ε_i , between the computational solution and the prescribed T_{MMS} at each i^{th} vertex is computed. The RMS of the errors, $\varepsilon_{RMS,j}$, for mesh j is computed as

$$\epsilon_{RMS,j} = \sqrt{\sum_{i=1}^{N_v} (\epsilon_i^2) / N_v}, \quad (31)$$

and the L_∞ norm of errors, $\epsilon_{L\infty,j}$, is computed as

$$\epsilon_{L\infty,j} = \max_{1 \leq i \leq N_v} |\epsilon_i|. \quad (32)$$

Between two sequential meshes in the systematic refinement, $p_{O,H}$ on the finer of the two meshes is computed as

$$p_{O,H} = \ln(\epsilon_j/\epsilon_i) / \ln(h_j/h_i), \quad (33)$$

where ϵ in Equation (33) represents either ϵ_{RMS} or $\epsilon_{L\infty}$, and subscripts j and i denote the finer and coarser of the two meshes, respectively. Using $k_2/k_1=1$ for convenience for the MMS study, the $p_{O,H}$ values for mesh H of each system configuration was expected to converge to 2, representing p_f . Companion, detailed descriptions of the theory and application behind the code verification used in this study are also explained in [15] and [16].

4.2 Solution Verification

Solution verification was performed on G^* for the heat-generating filler problem. A global deviation grid convergence index (GCI) approach was used to estimate the numerical error, U_{num} , on G^* [20]. Theoretically—and often in practice—a numerical solution is expected to asymptotically approach some solution with successive mesh refinement. This is often referred to as the “asymptotic regime.” On the other hand, if a mesh is too coarse, the computational solution may diverge from the asymptotic solution and even behave erratically with mesh refinement. This is considered the “non-asymptotic regime.” When a solution is in

the asymptotic regime, systematic solution changes can help to infer the final asymptotic solution—or at least bounds for the asymptotic solution. The global deviation GCI method computes a factor of safety on the finest mesh solution quantity—in this case G^* —value based on a quantitative estimation of where the solution falls with respect to the asymptotic solution regime. References [20] and [16] also give a clear description of the global deviation GCI approach as used in this work.

To approximate U_{num} with this method on a fine mesh, first a modified transcendental order of accuracy on the finest of three meshes, p_t , is determined iteratively from

$$p_t = \ln \left[(r_{1,2}^{p_t} - 1) \left(\left| \frac{G_3^* - G_2^*}{G_2^* - G_1^*} \right| \right) + r_{1,2}^{p_t} \right] / \ln(r_{1,2} r_{2,3}), \quad (34)$$

where the subscripts 1, 2, and 3 represent the finest, middle, and coarsest of three meshes, successively refined. Likewise, $r_{i,j}$ is mesh ratio between two successive meshes, where

$$r_{i,j} = h_j / h_i. \quad (35)$$

For this application, a global order of accuracy deviation value, Δp , is determined by

$$\Delta p = \min(|p_f - p_t|, 4p_f, 0.95p_f). \quad (36)$$

The deviation is used to determine the average global order of accuracy, p^* , defined as

$$p^* = p_f - \Delta p, \quad (37)$$

which in turn is used to compute a factor of safety, FS —based on the inferred proximity of the solution to the asymptotic regime—where,

$$FS = 3.0 - 1.9(p^*/p_f)^8. \quad (38)$$

The behavior of the FS with respect order of accuracy deviation is depicted in Figure 8, where the FS converges to 1.1 in the asymptotic regime and to 3.0 in the non-asymptotic regime. Note that the original GCI method in [18] offered only a binary option of 1.25 or 3.0, depending on the number of meshes used and/or deviation of the observed order of accuracy from the formal order of accuracy. The method employed here—from [20]—scales the FS value continuously between the asymptotic and non-asymptotic regimes.

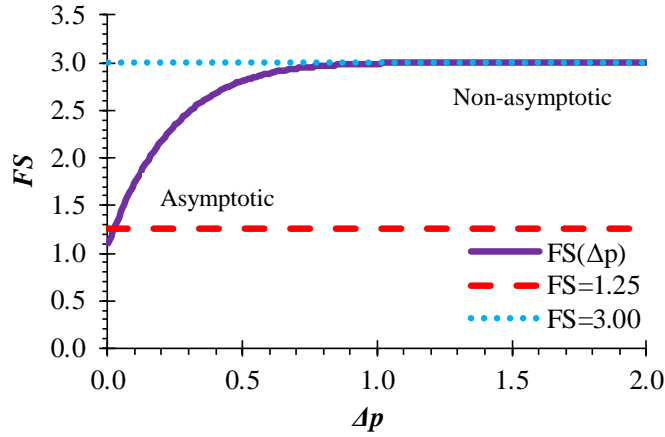


Figure 8. Global deviation estimator factor of safety

Finally, U_{num} on for G^* the finest of the three meshes is computed as

$$U_{num} = FS \left| (G_2^* - G_1^*) / (r_{1,2}^{p^*} - 1) \right|. \quad (39)$$

Thus, an estimate for the numerical uncertainty, induced by the discretization of the PDE, is determined based on the resultant behavior of G^* with mesh refinement and the solution's apparent position relative to the asymptotic regime.

5 Results

Using MMS with T_{MMS} and S_{MMS} as described above, the system domains for each α value were analyzed with $k_2/k_1=1$ for convenience. The prescribed MMS temperature solution from Equation (28) is plotted on the domain in Figure 9a with the resultant MMS source term distribution of Equation (30) plotted in Figure 9b. Note the similarities between contours for T_{MMS} and S_{MMS} due to the trigonometric function selection. With Figure 9b representing the applied source term to the computational domain for the MMS study, Figure 9a represents the temperature distribution to which the numerical solution is expected to converge with mesh refinement. Such convergence in mesh refinement is shown qualitatively in Figure 10, Figure 11, and Figure 12 for 2%, 40%, and 78% fill fraction mesh systems, respectively. Since $k_2/k_1=1$ for all MMS studies here, only difference between the analyzed systems is the mesh layout. This mesh variability illustrates that the numerical solutions are sensitive to mesh organization, but mesh refinement eventually converges all numerical solutions to the true solution. The resulting sets of $p_{O,H}$ with mesh refinement—indicated by changing H —are shown in Figure 13a and Figure 13b for the L_∞ norm and Figure 14a and Figure 14b for the RMS for each fill fraction system. The observed order of accuracy appears to approximately converge to second order with mesh refinement, thus providing evidence that the custom FEM code correctly solves the governing heat equation.

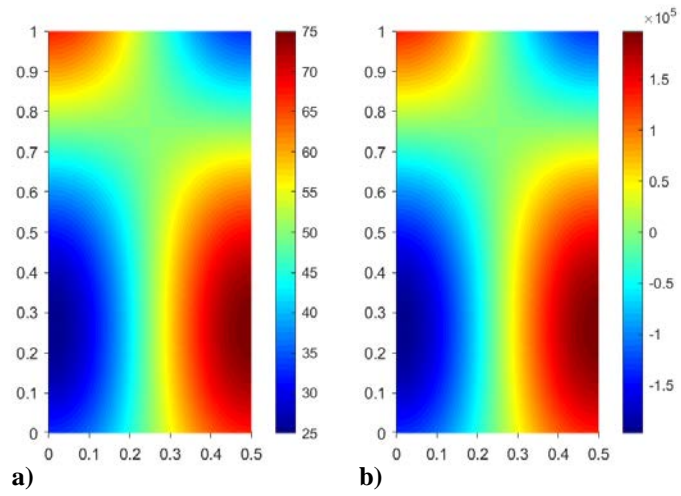


Figure 9. Contours for MMS a) temperature and b) source term

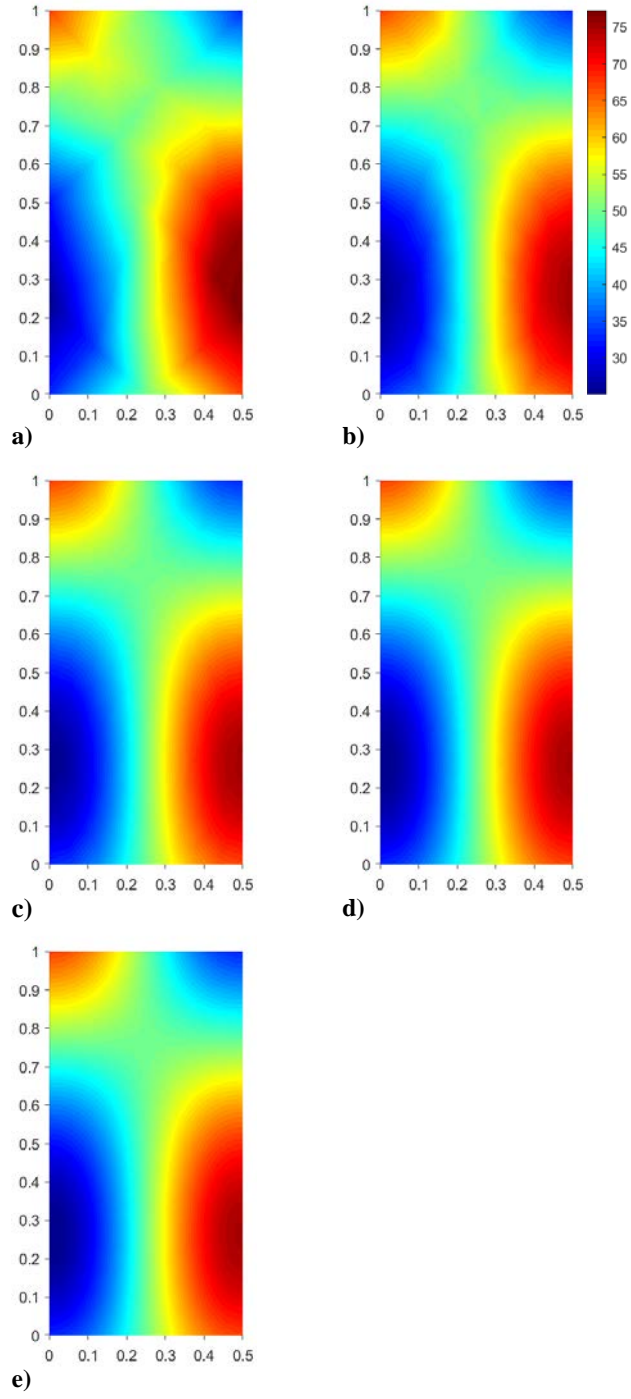


Figure 10. Numerical solution temperature contours with MMS source for $\alpha=2\%$ and $k_2/k_1=1.00$ with mesh refinement on meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

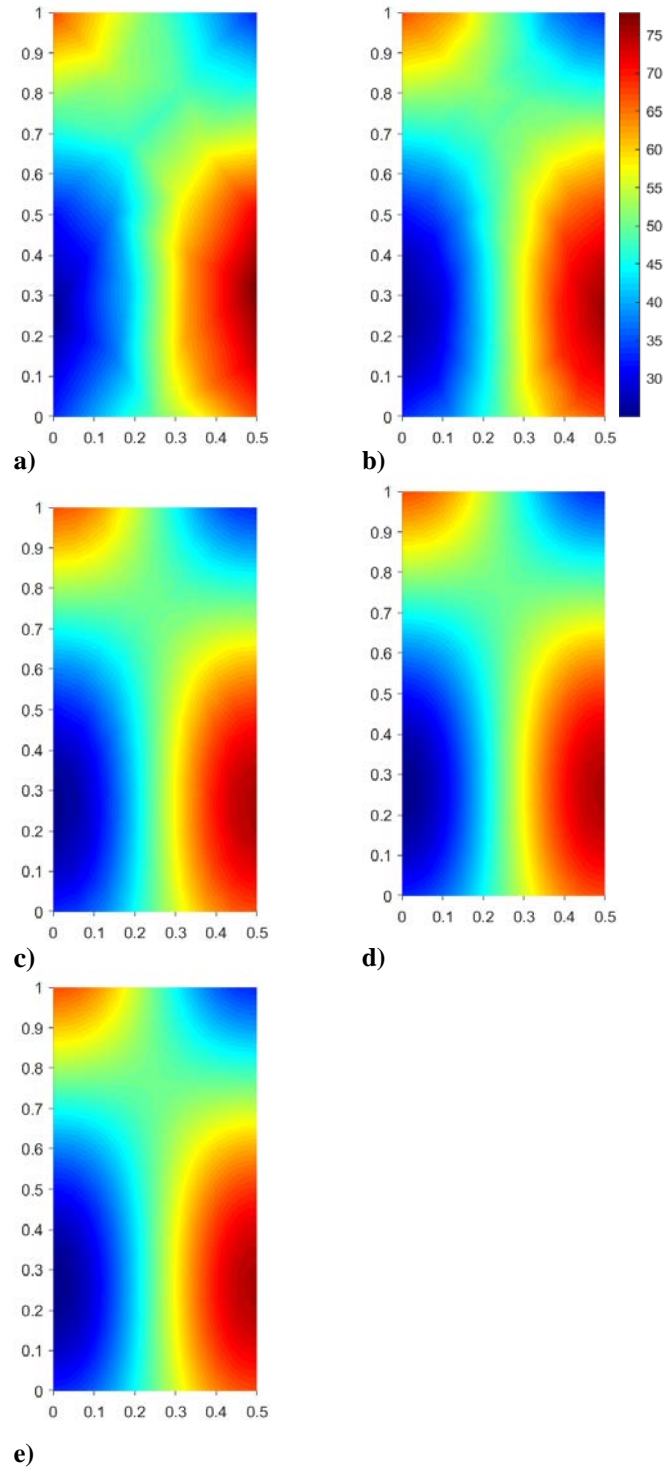


Figure 11. Numerical solution temperature contours with MMS source for $\alpha=40\%$ and $k_2/k_1=1.00$ with mesh refinement on meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

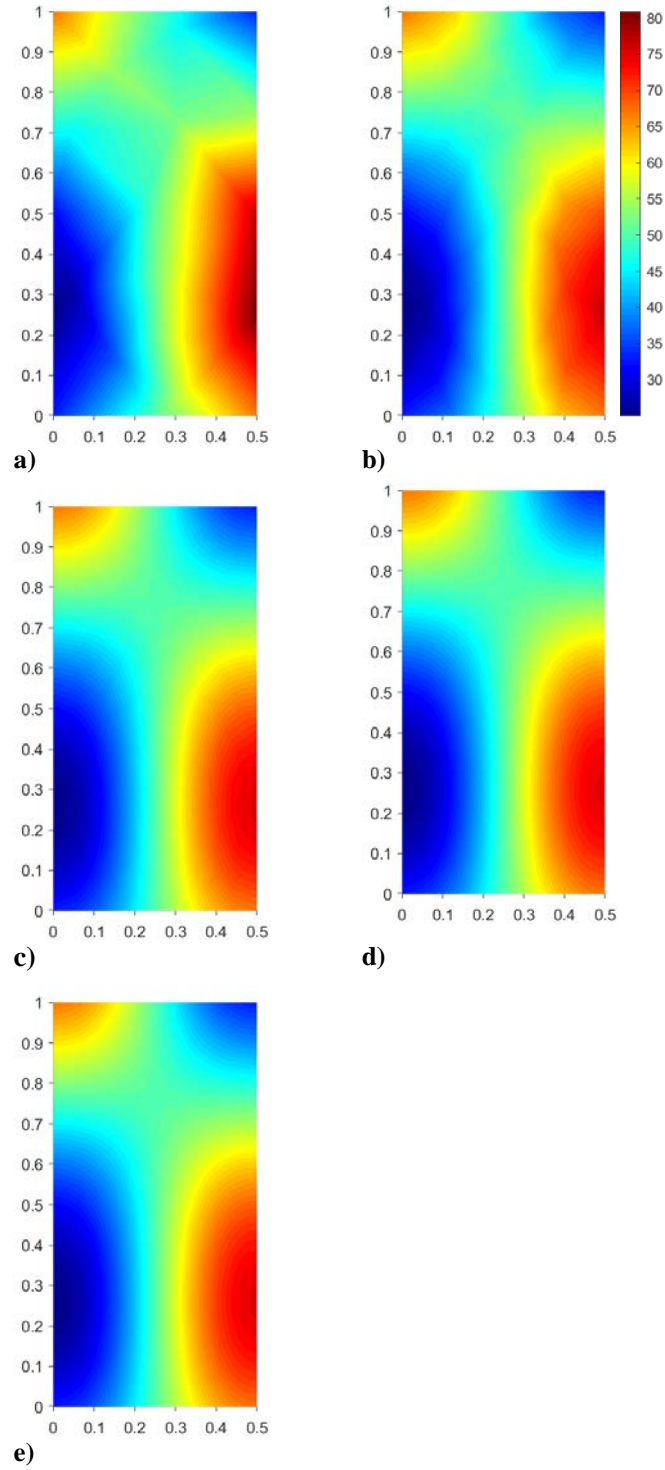


Figure 12. Numerical solution temperature contours with MMS source for $\alpha=78\%$ and $k_2/k_1=1.00$ with mesh refinement on meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

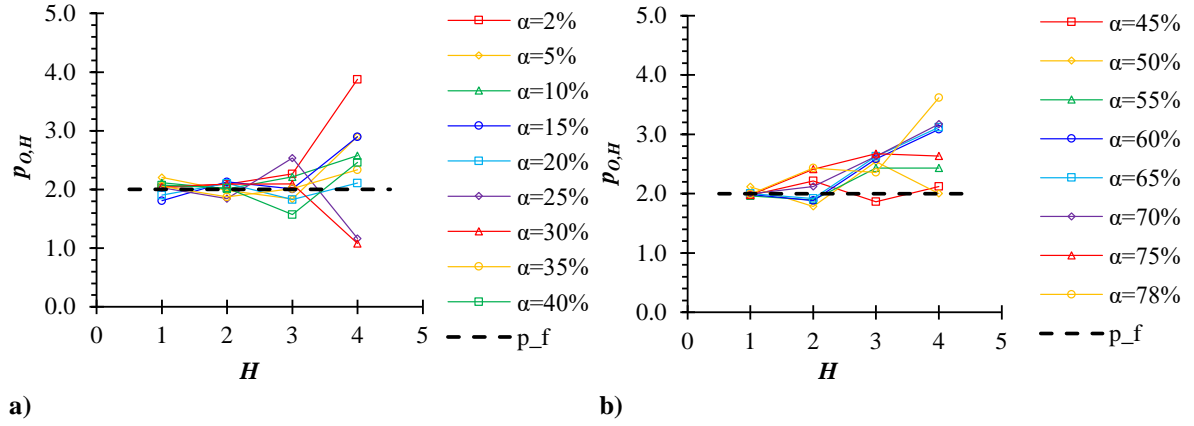


Figure 13. MMS L_∞ norm observed order of accuracy convergence for a) α of 2% through 40% and b) α of 45% through 78%

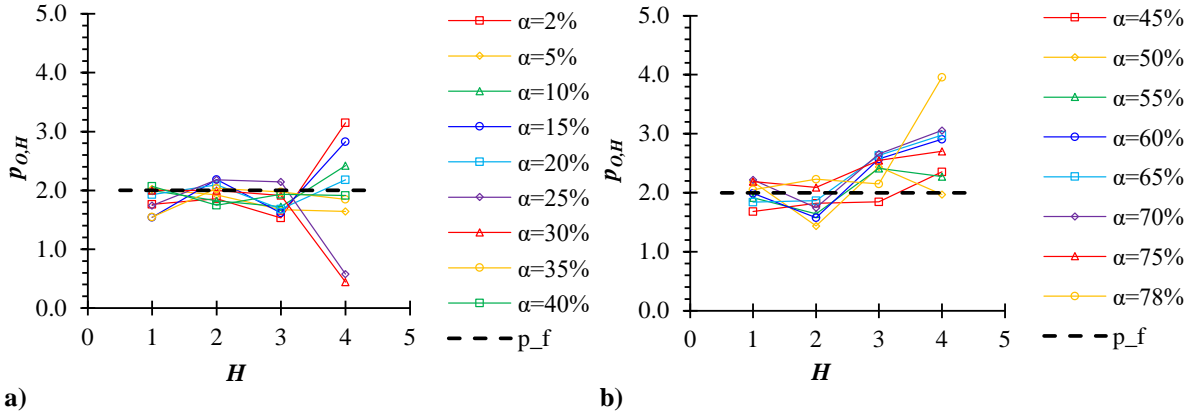


Figure 14. MMS RMS observed order of accuracy convergence for a) α of 2% through 40% and b) α of 45% through 78%

Figure 15a through Figure 15e show the qualitative computational temperature solution convergence with mesh refinement on a system with 2% fill fraction and with a filler thermal conductivity two orders of magnitude smaller than the matrix thermal conductivity, from meshes $H=5$ to $H=1$, respectively. Likewise, Figure 16a through Figure 16e show the temperature contours with mesh refinement for a system with 2% fill fraction and a filler

thermal conductivity three orders of magnitude larger than the matrix thermal conductivity. For these simulations, the same absolute heat load was applied to the filler region. It is readily observed that the overall system temperature significantly decreases when the filler thermal conductivity is increased, due to the fact that the filler more readily dissipates heat to the surrounding matrix material. Notice that the temperature profile near the filler is relatively circular and axisymmetric for the small fill fraction system.

Figure 17a through Figure 17e show temperature contours with mesh refinement on a system with 78% fill fraction and with a filler-to-matrix thermal conductivity ratio of 0.01, from meshes $H=5$ to $H=1$, respectively. Figure 18a through Figure 18e also show a system with 78% fill fraction but with a thermal conductivity ratio of 1,000.00. As expected, system temperatures decrease with increased filler thermal conductivity. Note that as opposed to the system with a smaller fill fraction, the hot temperature profile is less circular and more rectangular, due to the fact that the domain boundary is rectangular with a circular. Thus, the temperature profile is much less axisymmetric.

Figure 19a through Figure 19f show the finest mesh impact of increasing the k_2/k_1 ratio from 0.01 to 0.10, 1.00, 10.00, 100.00, and 1,000.00, respectively, for a 2% fill fraction. Likewise, Figure 20a through Figure 20f show finest mesh temperature contours on a 78% fill fraction system for the same thermal conductivity ratios. Note that although the same amount of heat is added to the small filler as is added to the large filler, the maximum temperature on the large filler system is significantly higher than the small filler system for the same thermal conductivity ratios. Although the local heat flux for the large filler is smaller than with the small filler, the small fillers offer a much shorter path from the center of the heat load to the surrounding matrix, thus implying a higher effective conductance---as defined here---for

smaller fill fractions. Lastly, Figure 21a through Figure 21e show the finest mesh temperature contours for a system with k_2/k_1 at 0.01 for fill fractions increasing from 2% to 25%, 45%, 65%, and 78%, respectively. In Figure 22a through Figure 22e and Figure 23a through Figure 23e, temperature contours are shown for increasing fill fraction on systems with $k_2/k_1=10.00$ and $k_2/k_1=1,000.00$, respectively. It is interesting to note that as filler thermal conductivity increases, the maximum temperature for systems with the same k_2/k_1 ratios is found at lower fill fractions. This is surely due to the fact that as the filler increases in size and thermal conductivity, it more readily dissipates heat from the center of the filler to cold boundary, where the thermal resistance of matrix material is decreasing with decreasing filler-to-boundary distance.

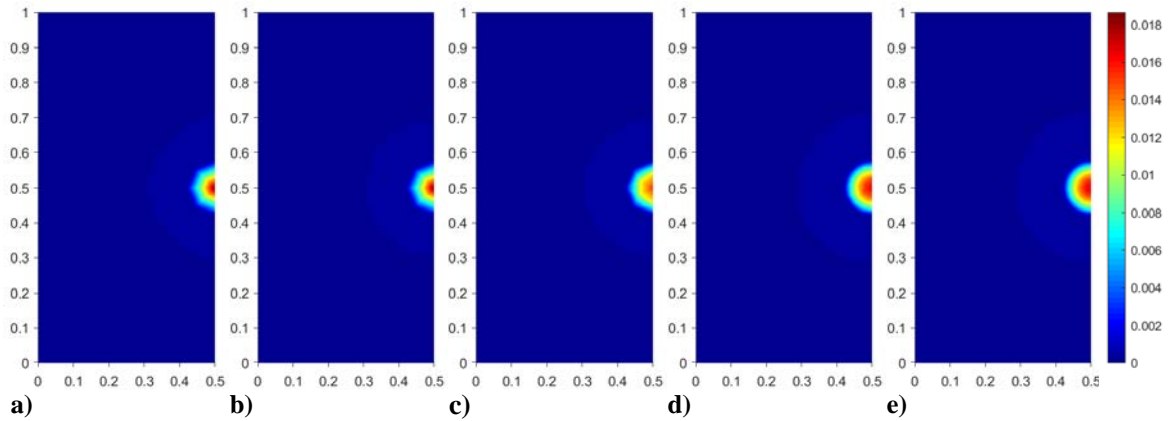


Figure 15. Temperature contours with mesh refinement for $\alpha=2\%$ and $k_2/k_1=0.01$ for meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

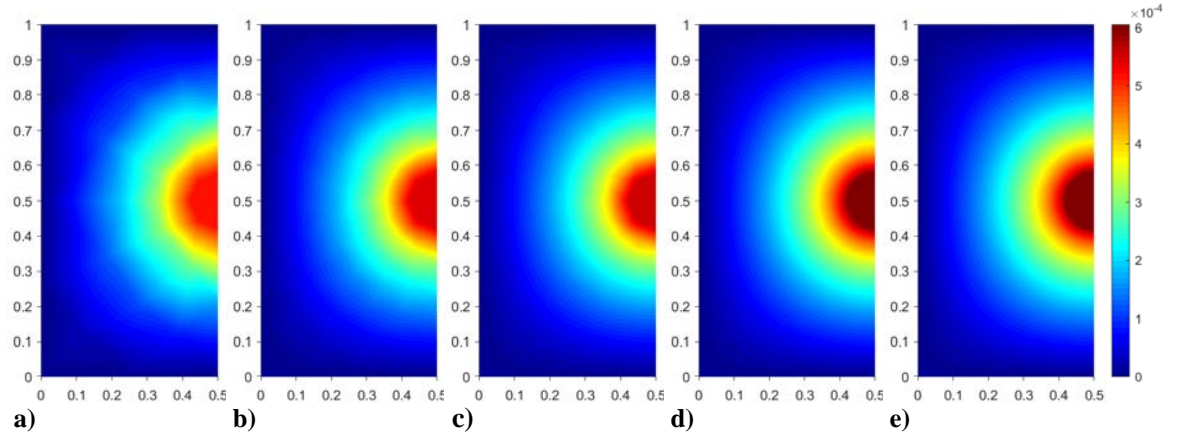


Figure 16. Temperature contours with mesh refinement for $\alpha=2\%$ and $k_2/k_1=1,000.00$ for meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

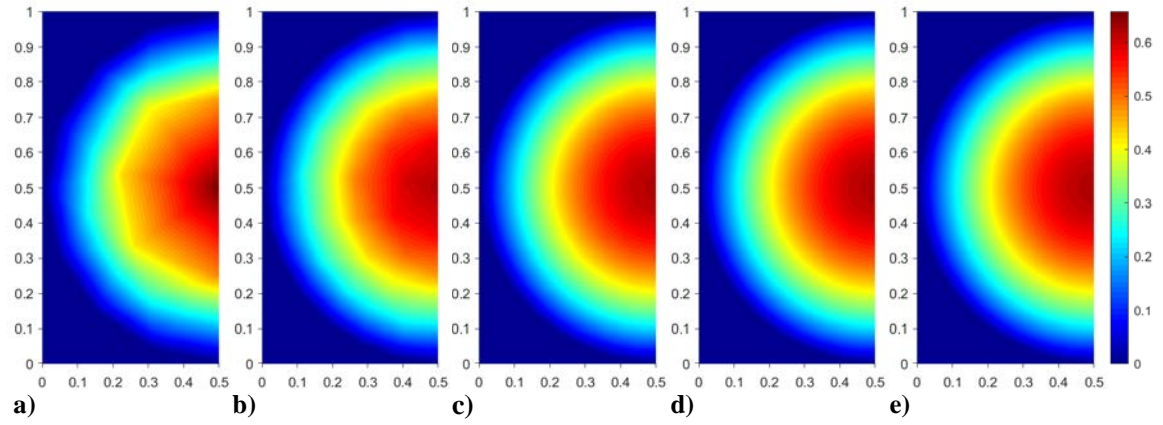


Figure 17. Temperature contours with mesh refinement for $\alpha=78\%$ and $k_2/k_1=0.01$ for meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

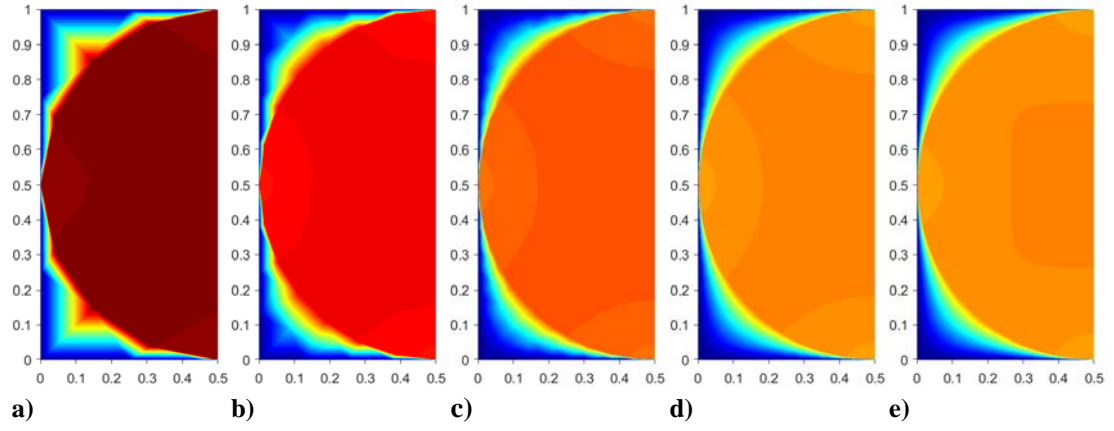


Figure 18. Temperature contours with mesh refinement for $\alpha=78\%$ and $k_2/k_1=1,000.00$ for meshes a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

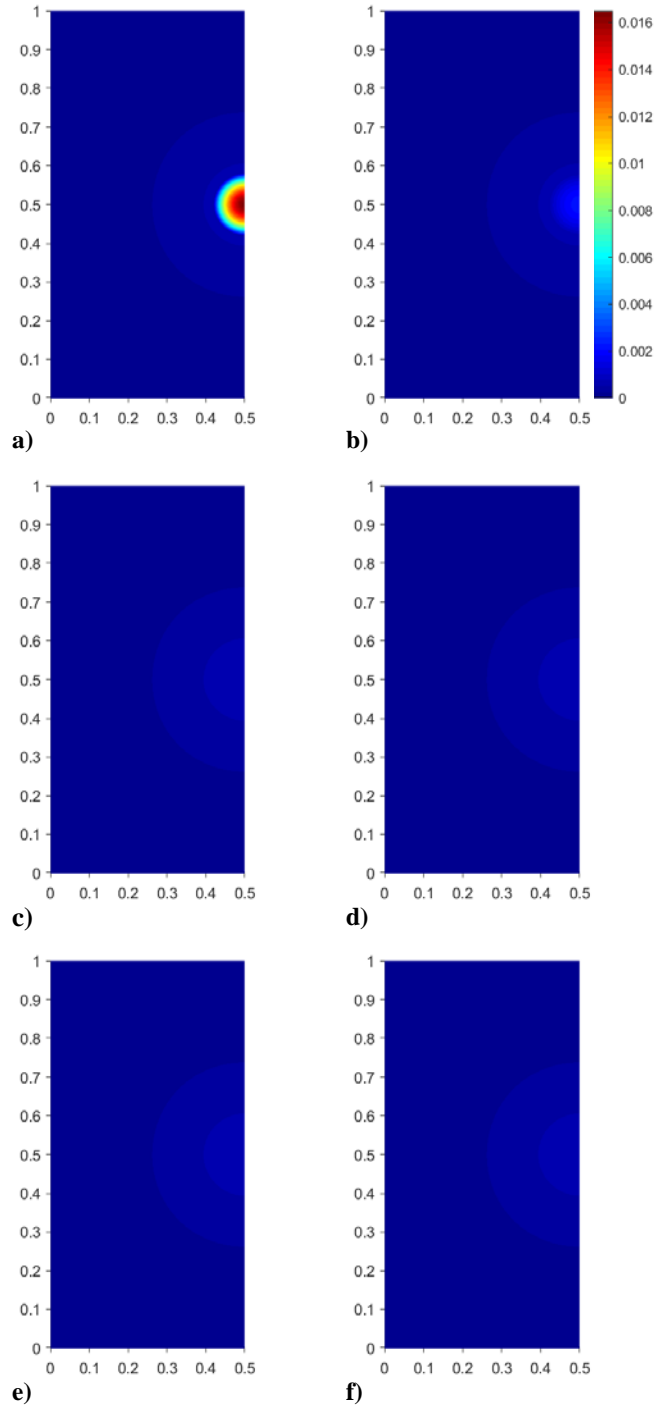


Figure 19. Temperature contours on the finest mesh for $\alpha=2\%$ with a) $k_2/k_1=0.01$, b) $k_2/k_1=0.10$, a) $k_2/k_1=1.00$, a) $k_2/k_1=10.00$, a) $k_2/k_1=100.00$, and a) $k_2/k_1=1,000.00$

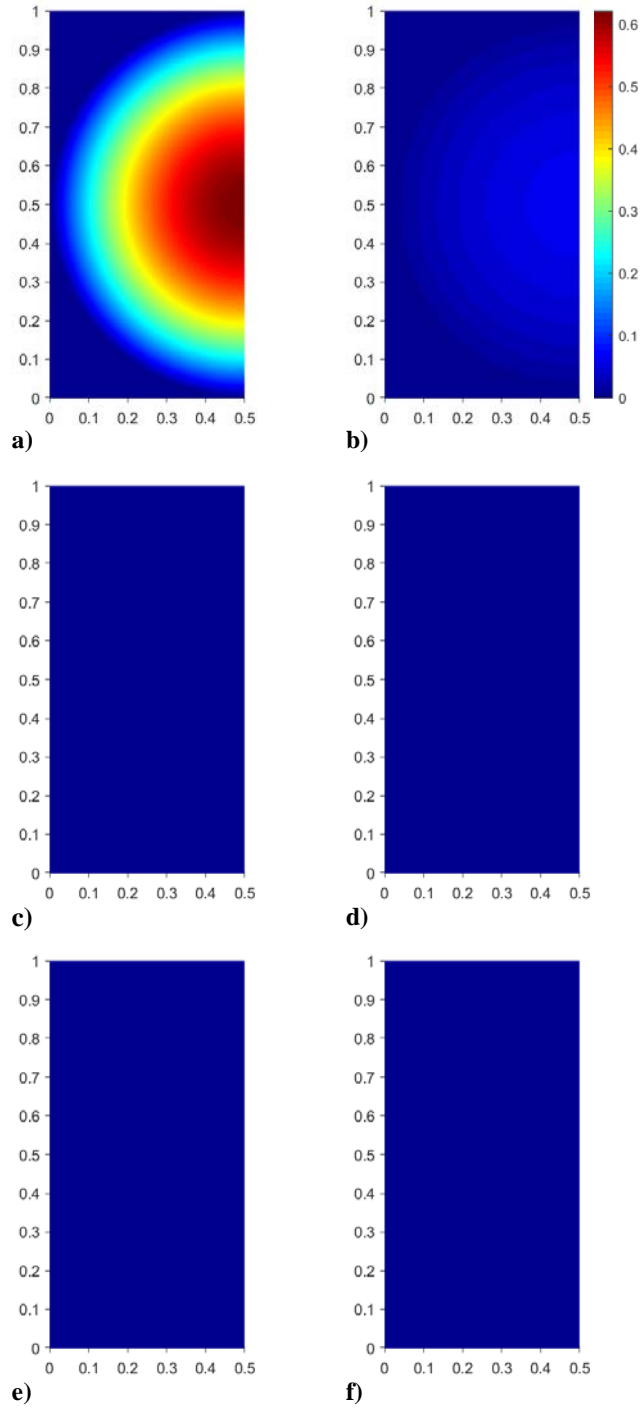


Figure 20. Temperature contours on the finest mesh for $\alpha=78\%$ with a) $k_2/k_1=0.01$, b) $k_2/k_1=0.10$, a) $k_2/k_1=1.00$, a) $k_2/k_1=10.00$, a) $k_2/k_1=100.00$, and a) $k_2/k_1=1,000.00$

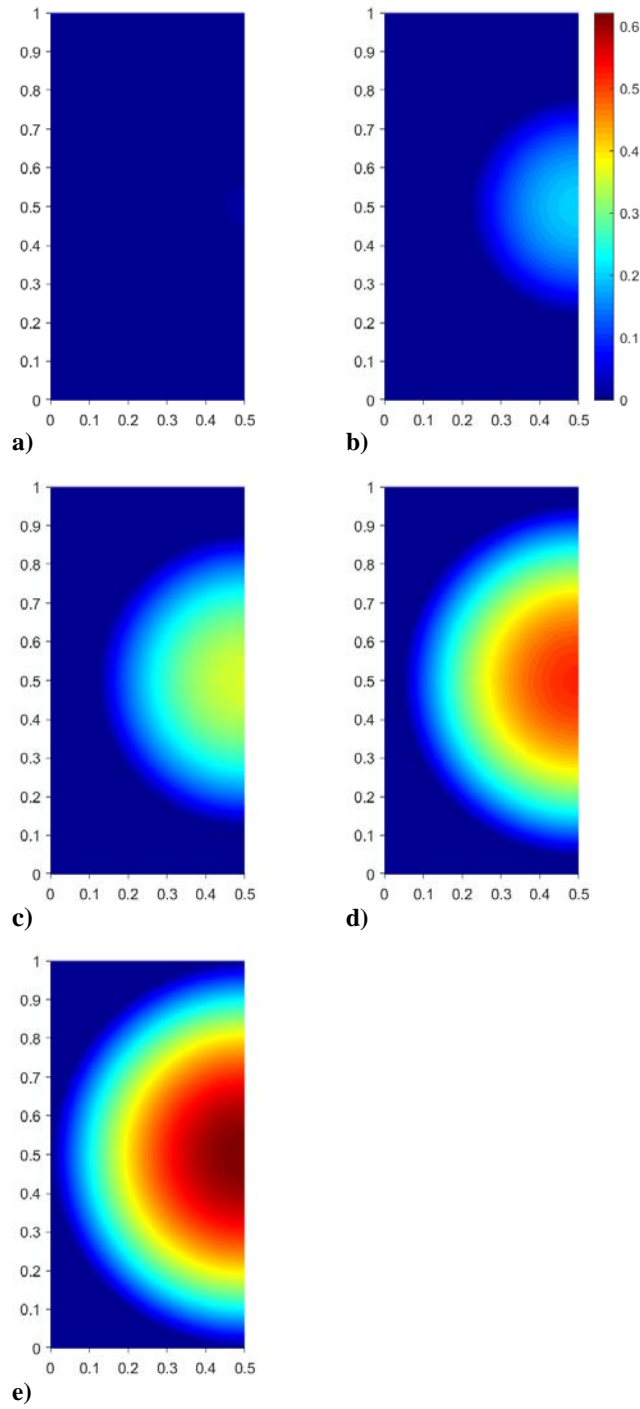


Figure 21. Temperature contours on the finest mesh for $k_2/k_1=0.01$ with a) $\alpha=2\%$, b) $\alpha=25\%$, c) $\alpha=45\%$, d) $\alpha=65\%$, and e) $\alpha=78\%$

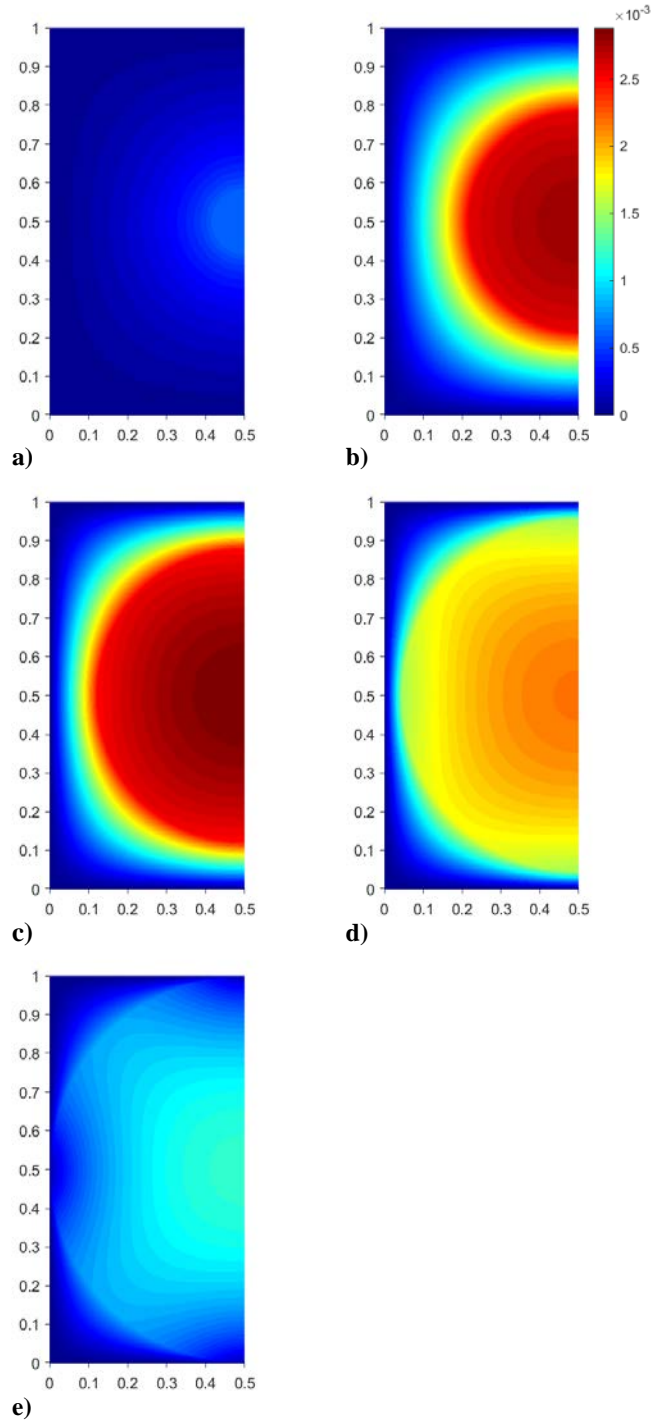


Figure 22. Temperature contours on the finest mesh for $k_2/k_1=10.00$ with a) $\alpha=2\%$, b) $\alpha=25\%$, c) $\alpha=45\%$, d) $\alpha=65\%$, and e) $\alpha=78\%$

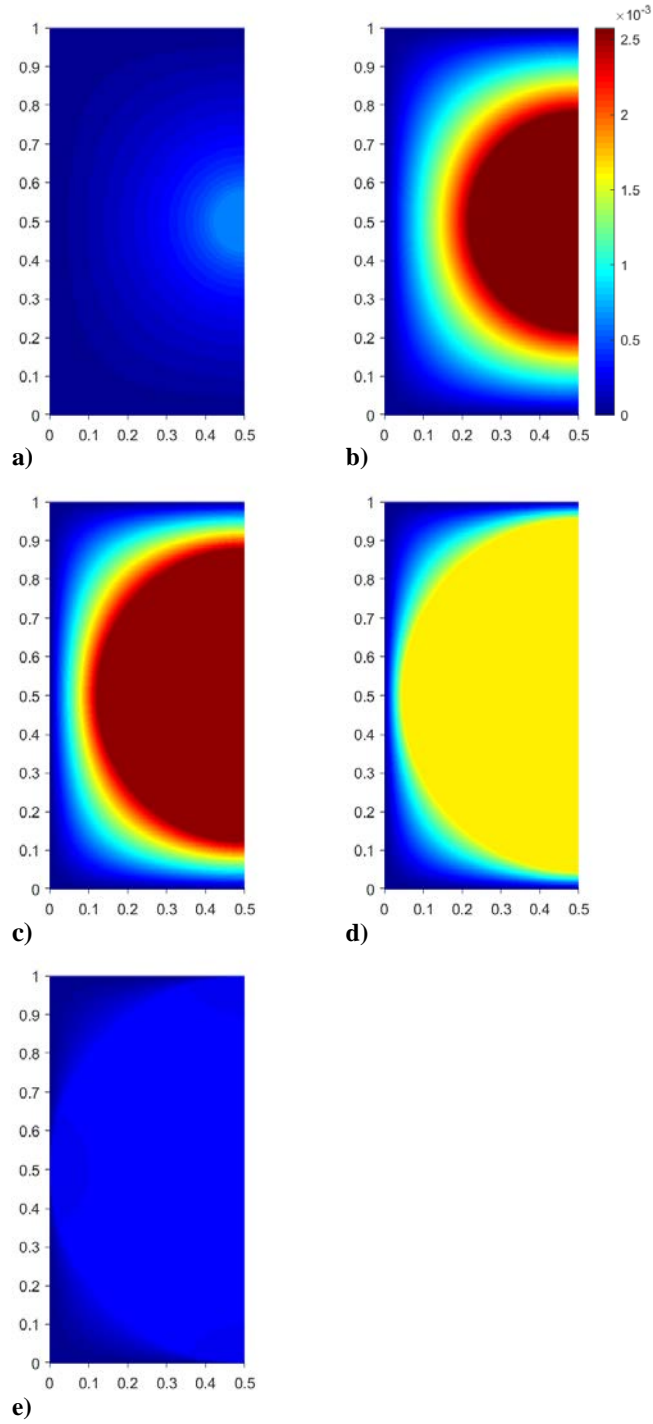


Figure 23. Temperature contours on the finest mesh for $k_2/k_1=1,000.00$ with a) $\alpha=2\%$, b) $\alpha=25\%$, c) $\alpha=45\%$, d) $\alpha=65\%$, and e) $\alpha=78\%$

From the finest meshes for each system configuration, G^* values were generated for each of the seven fill fractions with 27 different thermal conductivity ratios. The system responses as characterized by G^* are given in Figure 24a and Figure 24b. Note that the ordinate axes are shown in logarithmic base 2 scale. As should be expected, G^* increases with α because the physical separation between the T_C boundary and the heat source decreases as the filler boundary expands. Naturally, G^* increases as k_2/k_1 increases due to the fact that heat is conducted away from the center of the filler more readily.

Using the global deviation GCI method to perform solution verification on the G^* values determined through simulation, U_{num} on G^* was estimated to be below 3.0% of G^* for all but six of the 459 data points, with 92% of the computed data points having U_{num} below 1.0% of the reported G^* value. The largest four estimated uncertainties were computed at the four largest k_2/k_1 values for $\alpha=78\%$. The larger uncertainties could possibly be due to mesh distortion (e.g., high aspect ratio triangle elements) in the large fill fraction mesh between the edge of the filler and the system boundary. However, a major contributor is that the observed average order of accuracy is much higher than the formal order of accuracy, implying significant deviation from the formal order and solutions in the non-asymptotic solution regime. The solution does converge well, but the GCI method used conservatively estimates a larger uncertainty when the observed order of accuracy deviates from the formal order of accuracy. The U_{num} estimates are presented graphically in Figure 25a and Figure 25b.

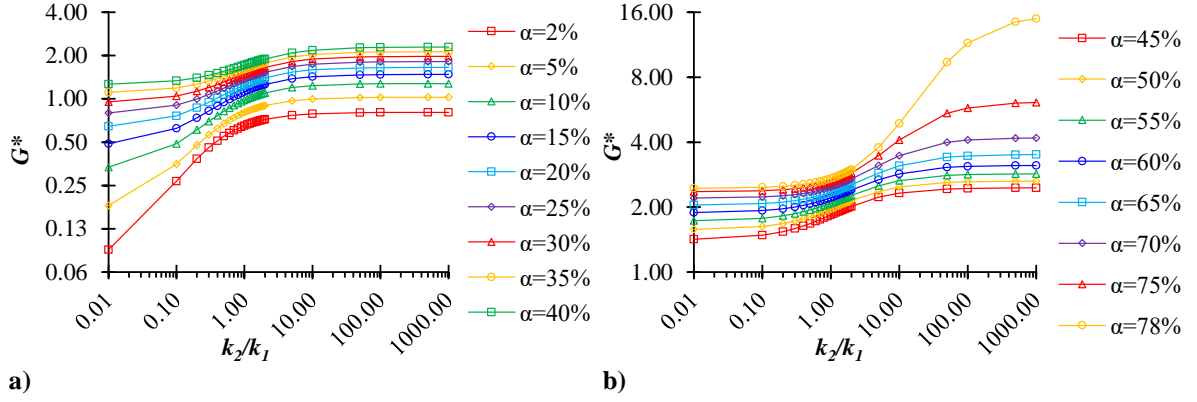


Figure 24. G^* response to α and k_2/k_1 for a) $\alpha=2\%$ through 40% and b) $\alpha=45\%$ through 78%

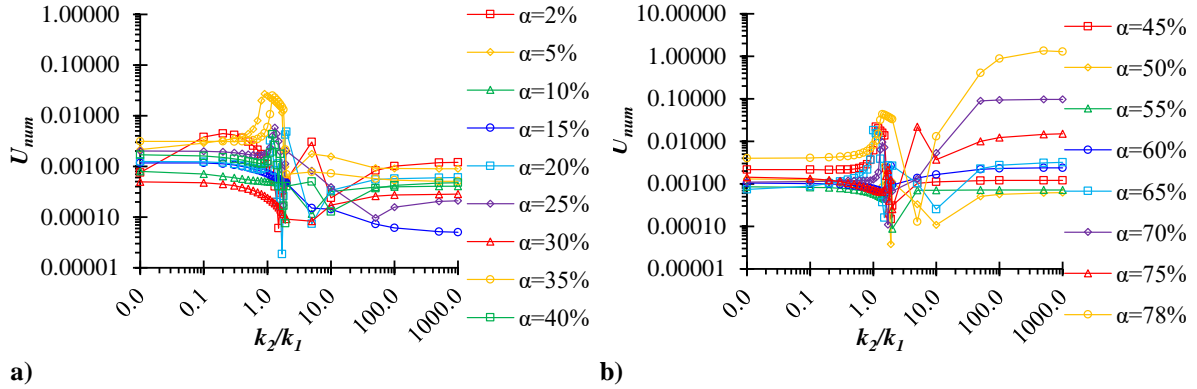


Figure 25. U_{num} against k_2/k_1 for a) $\alpha=2\%$ through 40% and b) $\alpha=45\%$ through 78%

6 Conclusion and Future Work

In this study, the systematic thermal conductance response of a composite material with filler heat generation was tracked with respect to filler size and filler thermal conductivity. A formalized MMS study showed that the Galerkin FEM approach used to solve the governing heat equation PDE was implemented correctly with approximately second order accuracy. Solution verification was performed using the global deviation GCI method that considers a solution's convergence rate to infer how distant the solution is from the asymptotic solution

regime. The GCI method estimated U_{num} on the dimensionless G^* value to be less than 1.0% in the vast majority of data points. The global deviation GCI method appeared to estimate conservatively large numerical uncertainties for converging system quantities due to convergence rates being higher than the formal order of accuracy. It was shown that for a given thermal conductivity ratio, decreasing the fill fraction increases the effective thermal conductance. As expected, decreasing the filler thermal conductivity decreases the effective thermal conductance.

Non-circular filler geometries should be considered in future work. Studies could also be performed on the effects of filler location and orientation within the unit cell. Aside from multi-material heterogeneous materials, closed-cell porous materials could be analyzed with heat flux boundaries at the pore interface. A study could also be performed in which the filler is maintained at a constant temperature with the system exterior boundary fixed at a different temperature. Whereas this work considered uniform Dirichlet boundary conditions on the system, other types of conditions could be enforced, or a non-uniform temperature condition could be maintained. These studies could help to characterize the thermal responses of heterogeneous materials under a variety of application conditions.

References

- 1 Hawkes, G. L., Sterbentz, J. W., and Pham, B., 2015, "Thermal Predictions of the AGR-2 Experiment with Variable Gas Gaps," *Nuclear Technology*, **190**(3), pp. 245-253.
- 2 Hawkes, G. L., Sterbentz, J. W., Maki, J. T., and Pham, B. T., 2017, "Thermal Predictions of the AGR-3/4 Experiment with Post Irradiation Examination Measured Time-Varying Gas Gaps," *Journal of Nuclear Engineering and Radiation Science*, **3**(), pp. 041007-1-041007-11.
- 3 Khattak, M. A., Borhana Omran, A. A., Kazi, S., Khan, M. S., Ali, H. M., Tariq, S. L., and Akram, M. A., 2019, "A Review of Failure Modes of Nuclear Fuel Cladding," *Journal of Engineering Science and Technology*, **14**(3), pp. 1520-1541.

- 4 Finsterle, S., Muller, R. A., Baltzer, R., Payer, J., and Rector, J. W., 2019, "Numerical Evaluation of Thermal Effects from Nuclear Waste Disposed in Horizontal Drillholes," *International High-Level Radioactive Waste Management Conference*.
- 5 Lyons, M. F., Boyle, R. F., Davies, J. H., Hazel, V. E., and Rowland, T. C., 1972, "UO₂ properties affecting performance," *Nuclear Engineering and Design*, **21**(2), pp. 167-199.
- 6 Hobson, C., Taylor, R., and Ainscough, J. B., 1974, "Effect of porosity and stoichiometry on the thermal conductivity of uranium dioxide," *Journal of Physics D: Applied Physics*, **7**(7), 1003.
- 7 Bakker, K., Kwast, H., and Cordfunke, E. H. P., 1995, "Determination of a porosity correction factor for the thermal conductivity of irradiated UO₂ fuel by means of the finite element method," *Journal of Nuclear Materials*, **226**(1), pp. 128-143.
- 8 Jin, E., Liu, C., and He, H., 2016, "The Influence of Microstructures on the Thermal Conductivity of Polycrystalline UO₂," *Proceedings of the ASME 2016 24th International Conference on Nuclear Engineering*, V001T02A028.
- 9 Newman, C., Hansen, G., and Gaston, D., 2009, "Three dimensional coupled simulation of thermomechanics, heat, and oxygen diffusion in UO₂ nuclear fuel rods," *Journal of Nuclear Materials*, **392**(1), pp. 6-15.
- 10 Ramirez, J. C., Stan, M., and Cristea, P., 2006, "Simulations of heat and oxygen diffusion in UO₂ nuclear fuel rods," *Journal of Nuclear Materials*, **359**(3), pp. 174-184.
- 11 Zohuri, Bahman, and Nima Fathi. *Thermal-hydraulic analysis of nuclear reactors*. New York: Springer, 2015.
- 12 Rechard, Robert P., Teklu Hadgu, Yifeng Wang, Lawrence C. Sanchez, Patrick McDaniel, Corey Skinner, and Nima Fathi. *Technical Feasibility of Direct Disposal of Electrefiner Salt Waste*. No. SAND2017-10554. Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), 2017.
- 13 Walker, C. T., Staicu, D., Sheindlin, M., Papaioannou, D., Goll, W., and Sontheimer, F., 2006, "On the thermal conductivity of UO₂ nuclear fuel at a high burn-up of around 100 MWd/kgHM," *Journal of Nuclear Materials*, **350**(1), pp. 19-39.
- 14 Geuzaine, C. and Remacle, J.-F., 2009, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering* **79**(11), pp. 1309-1331.
- 15 Irick, K. W. and Fathi, N., 2018, "Thermal Response of Open-Cell Porous Materials: A Numerical Study and Model Assessment," *Proceedings of the ASME 2018 Verification and Validation Symposium*, V001T03A002.
- 16 Irick, K. W. and Fathi, N., 2019, "Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis," *Proceedings of the ASME 2019 Verification and Validation Symposium*, V001T12A002.

- 17 The American Society of Mechanical Engineers, 2009, *Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer*, ASME V&V 20-2009, ASME, New York, NY.
- 18 Roache, P. J., 2009, *Fundamentals of Verification and Validation*, Hermosa Publishers, Socorro, NM.
- 19 Oberkampf, W. L. and Roy, C. J., 2010, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge, UK.
- 20 Phillips, T. S. and Roy, C. J., 2016 “A New Extrapolation-Based Uncertainty Estimator for Computational Fluid Dynamics,” *Journal of Verification, Validation and Uncertainty Quantification*, **1**(4), pp. 041006-1, 041006-13.

CHAPTER 5: EVALUATION OF PORE GEOMETRY EFFECTS ON POROUS CELL THERMAL BEHAVIOR⁵

Abstract

Heterogeneous material composition can play a significant role in the thermal response of a material system at both fine-scale and macro-scale levels. The complexity of heat transfer by conduction can increase significantly in heterogeneous structures, including multi-component systems, complex microstructures and porous structures with dependent thermal conductivity values. This is especially true for porous materials, where voids in the matrix material induce thermal resistance within the conductive system and divert heat flow within the material. Many high-impact fields rely on heterogeneous materials as a foundational aspect of their functionality and performance in thermal applications. The thermal response of porous materials can be driven by pore size and geometry, both of which can be affected by manufacturing process or environmental conditions. The investigation presented in this article is dedicated to studying the effects of pore geometry on the effective thermal conductivity, k_{eff} , of a whole porous unit cell. To study a simple system porous unit cell's, simulations were performed on non-circular porous material computational models. A set of two-dimensional, steady-state models were constructed to observe the thermal effects due to different pore geometries—including triangular, square, and elliptical—centered in a unit cell porous medium. The finite element method (FEM) was implemented in Fortran—using unstructured

⁵ This chapter will be under review for publication in an **Elsevier Journal** in late 2020. The content in this chapter was reproduced directly from the journal version, differing only in style formatting. A less comprehensive version of this chapter was published in *Proceedings of the ASME 2019 Verification and Validation Symposium* under the title “Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis.”

triangular meshes—to simulate the thermal response of the systems. The method of manufactured solutions (MMS) was used to perform code verification and evaluate the code’s ability to solve the governing heat conduction equation. The results of the MMS study show that the implementation of the numerical FEM is approximately second order accurate. This article presents a study on the effects of pore size, shape, and rotational orientation on a heterogeneous material’s dimensionless effective thermal conductivity, k^* . A modern grid convergence index method was used to perform solution verification on the k^* results, estimating the numerical uncertainty in k^* to be less than 6% in almost all cases. Comparative thermal responses show the sensitivity of k^* with respect to pore shape and the variable sensitivity of response to pore orientation, where pore size is a major driver in the effective thermal response. A regression function is finally presented with less than 10% deviation from the computationally-determined results.

Nomenclature

a	= coefficient
A	= area
α	= porosity
b	= base, side
c	= coefficient
f	= function
FS	= factor of safety
Γ	= domain boundary
\mathbf{G}	= conductance matrix
h	= characteristic mesh size

H	= mesh number
k	= thermal conductivity
\mathbf{K}	= thermal conductivity matrix
L	= cell length
\mathbf{n}	= normal vector
N	= total quantity
\mathbf{N}	= vector shape function
p	= order of accuracy
\mathbf{p}	= direction vector
\mathbf{P}	= heat load vector
\mathbf{q}	= heat flux vector
R	= radius
ρ	= residual
$\boldsymbol{\rho}$	= residual vector
S	= energy source
t	= index
T	= temperature
\mathbf{T}	= vertex temperature vector
θ	= pore clocking angle
U	= uncertainty
w	= weight function
\mathbf{w}	= vertex weight function vector
W	= cell width

x = x-coordinate
 y = y-coordinate
 ψ = dimensionless clocking angle
 Ω = domain

Subscripts

a = semi-major
 α = pore
 c = calibration
 cyc = cyclic
 C = cold
 eff = effective
 f = formal
 g = geometric
 H = mesh number, hot
 L^∞ = L-infinity norm
 MMS = manufactured solution
 n = normal
 num = numerical
 O = observed
 p = semi-minor
 ref = reflection
 reg = regression
 RMS = root mean square

t = triangle, transcendental

Superscripts

$*$ = dimensionless, average

$'$ = per unit length

1 Introduction

Heterogeneous materials are prevalent in many high-consequence and high-value systems, where accurate thermal design and engineering are fundamental to the system's success. Heterogeneity in a given material's composition can significantly impact the material's thermal response. Thus, it is advantageous to evaluate the impact of heterogeneous characteristics on a material's thermal behavior. Systems can be impacted by heterogeneity on a fine scale (e.g., at the material level) and on a larger scale (e.g., at the system level). Effects of heterogeneity are especially important in porous materials—e.g., nuclear fuel components—where pores in the bulk material matrix impede and divert heat flow within the material. Oftentimes, heterogeneity is caused by porous structures within a material's solid domain. This is especially true in energy systems where thermal barrier coatings [1-4], semi-conductors [5], and/or energy storage materials [6] are used.

The complexity of heat transfer by conduction can increase significantly in heterogeneous structures, including multi-component systems, complex microstructures and porous structures with dependent thermal conductivity values. Any discontinuity of material properties, especially thermal conductivity which is caused by cracks, gaps, voids, changing crystal structures, and nonuniform heat generation would need further investigation to understand the thermal behavior of the system of interest. Many advanced thermal systems rely on heterogeneous materials for reliable system performance.

Heterogeneous materials are pervasive across all forms of engineering fields and play a critical role in the performance of advanced technologies. In this work, the term “heterogeneous” is used to refer generally to any material with non-continuous material structure properties, where different regions of the material structure are comprised of different material properties and/or behavior. Examples of commonplace heterogeneous materials and applications can include aerogel insulations, foams, 3D-printed structures, nuclear fuel assemblies, structural composites, thermal barrier coatings, and electronics dielectrics. Oftentimes heterogeneous materials—especially ceramics or ceramic-derivatives—are used in high-consequence and high-temperature applications where sparse empirical data is available. A combination of application criticality and lack of real-world material performance measurements in high-consequence or high-temperature applications necessitates the use of computational approaches to assess material behavior.

Many high-impact fields rely on heterogeneous materials as a foundational aspect of their functionality and performance in thermal applications. Figure 1 illustrates three examples of heterogeneous material structures in different applications. As an application example, many energy storage system concepts depend on multi-phase and/or multi-component materials [6,7]. Hypersonic flight vehicles experience extreme heat loads on leading edges and rely on composite and ceramic materials to prevent destruction of the vehicle [8-10]. In geosciences, evaluation of thermal behavior of heterogeneous—especially porous—earthen regions is important in assessing environmental responses to both natural and man-made conditions [11-15]. The additive manufacturing industry has a natural need for understanding thermal transport in porous materials—considering the non-homogenous and high-temperature processes used—where additive manufacturing can be used specifically for thermal

applications [16-18]. The thermal performance and advantage of other advanced materials—such as aerogels and doped polymers—are driven by their heterogeneous characteristics and can be seen in marine, oil and gas, aerospace, energy, and thermal management industries [19-22]. The expansive set of industries and specific thermal applications where heterogeneous materials are of immediate consequence is immeasurable. Consequently, the pursuit of measuring, analyzing, and predicting the thermal performance of such materials is in high demand.

The thermal response of porous materials can be driven by pore size and geometry, both of which can be affected by manufacturing process or environmental conditions [3,23,24]. Heat transfer in nuclear fuel elements is heavily dependent on these all of these phenomena. A variety of materials have been applied and proposed to be used as reactor fuels since decades ago. In thermal reactors, uranium dioxide, UO_2 , has shown satisfactory chemical and irradiation tolerance despite the disadvantages of low thermal conductivity and uranium atom density [25]. Oxide fuel is manufactured by sintering pressed powdered UO_2 and/or mixed oxides at high temperature values to produce ceramic pellets. The pellets of fuel are manufactured with the nominal values of 5 to 10% porosity to prevent pellet swelling from gaseous fission product species.

The effective thermal conductivity, k_{eff} , of a porous medium can be calculated from the traditional linear correlation as a function of the void or pore fraction, α , and the bulk matrix material thermal conductivity, k_1 , as defined below [26].

$$k_{eff} = (1 - \alpha)k_1 \quad (1)$$

While Equation (1) can be useful for some fundamental calculations to analyze simple geometries and composites, it does not provide reliable results for nonlinear geometries and functions, e.g., radial temperature distribution or further nonlinearity.

Understanding the effective thermal behavior of porous materials can be essential in engineering and design of critical next-generation thermal systems [27-31]. This paper is dedicated to investigating the effective thermal response of porous materials using numerical assessments to correlate thermal response with pore characteristics.

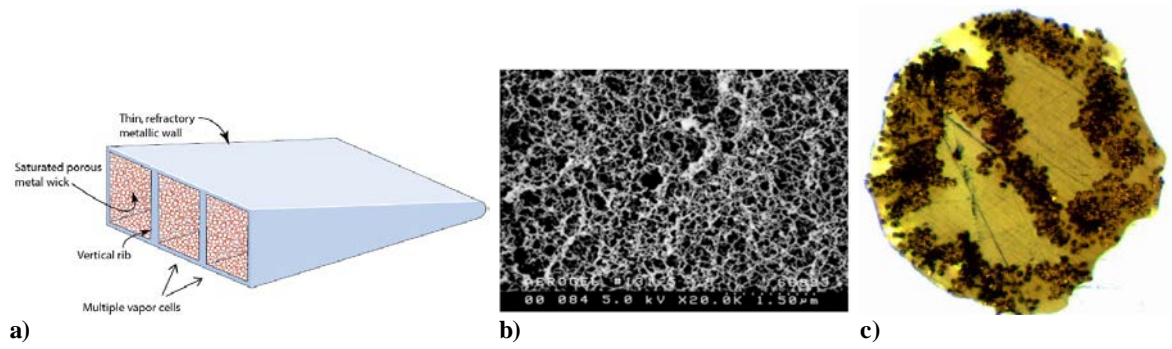


Figure 1. Examples of a) porous structure hypersonic leading edge [10], b) silica aerogel insulation material network [32], and c) 3D printing carbon fiber-polymer strand [3218]

2 System Description

Porous material systems were modeled in this work using three different two-dimensional pore geometry types: triangle, square, and ellipse. The triangle pore was restricted to be equilateral, and the ellipse semi-major-to-semi-minor axis ratio was maintained at 2-to-1. As shown in Figure 2 through Figure 4, the centroid of each pore was centered in a square solid domain with thermal conductivity, k_I , and side dimensions W and L in the x -direction and y -direction, respectively. Here, $W=L=I$. The triangle pore is characterized by its base, b , with pore area, A_a , computed as

$$A_\alpha = \frac{\sqrt{3}}{4} b^2. \quad (2)$$

The square pore is characterized by its side length, b , with pore area computed as

$$A_\alpha = b^2. \quad (3)$$

The ellipse pore is characterized by its semi-minor axis, R_p , and semi-major axis, R_a , with pore area computed as

$$A_\alpha = \pi R_p R_a, \quad (4)$$

where the ratio $R_a/R_p=2$ was maintained. The systems were analyzed at different clocking angles, θ_α , about the pore centroid with respect to the positive y -direction vector. Boundary conditions on the domains were enforced such that the positive y face of the domain was fixed at a cold temperature, T_C , the negative y face of the domain was fixed at a hot temperature, T_H , and all other solid domain boundaries were considered adiabatic. The enforced temperature gradient across the domain induces an average heat flow per unit length, q' , across the system Dirichlet boundaries.

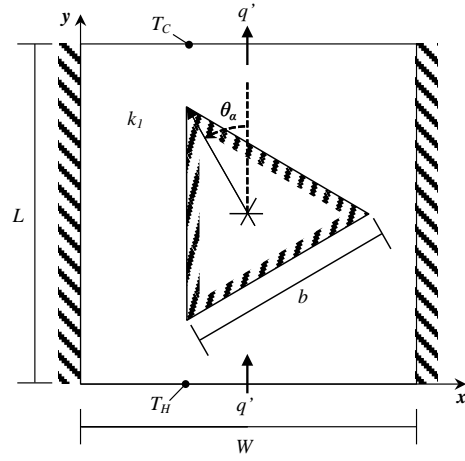


Figure 2. Unit cell triangular nano-porous structure

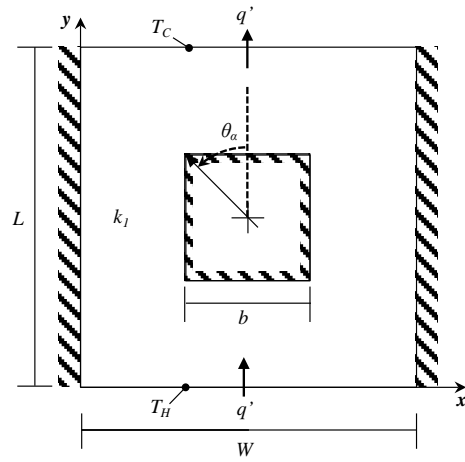


Figure 3. Unit cell square nano-porous structure

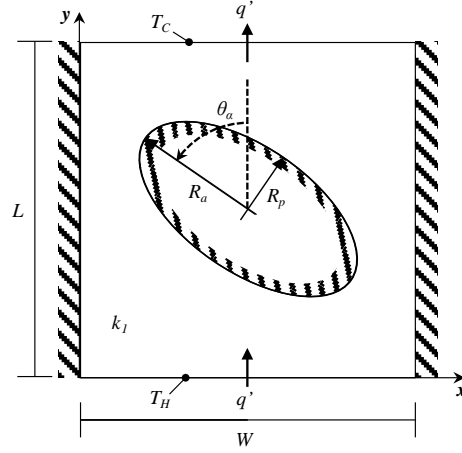


Figure 4. Unit cell elliptical nano-porous structure

In this study, the effects of both θ_α and porosity, α , were investigated, where porosity is defined as

$$\alpha = \frac{A_\alpha}{WL}. \quad (5)$$

With changing θ_α and α , the q' magnitude is affected, and an effective thermal conductivity, k_{eff} , of the system can be computed by

$$k_{eff} = \frac{q'L}{W(T_H - T_C)}, \quad (6)$$

where linear conductive heat transfer across the system implies an average q' at the boundaries due to a fixed temperature difference across the system boundaries. A dimensionless effective thermal conductivity, k^* , represents the normalization of k_{eff} by the solid domain thermal conductivity, where

$$k^* = \frac{k_{eff}}{k_1}. \quad (7)$$

The different pore geometries were evaluated at 2.5% increments in α with an additional minimum and maximum and at 2.5 ° increments in θ_α , with configurations summarized in Table 1. The upper bounds of θ_α represent the angle at which rotational symmetry—or reflection—occurs for the system geometry, θ_{ref} . Symmetry also occurs at $\theta_\alpha=0^\circ$.

Table 1. Geometry configurations

Geometry	α (%)	θ_α (°)
Triangle	1.0, [2.5,30.0], 31.0	[0,30]
Square	1.0, [2.5,47.5], 49.0	[0,45]
Ellipse	1.0, [2.5,37.5], 38.0	[0,90]

3 Discretization

A finite element (FE) approach was used to determine the resultant k^* values for the different system configurations described above. The governing steady-state, two-dimensional heat equation, given to be

$$k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + S = 0, \quad (8)$$

was solved across the solid domain, where k is thermal conductivity, T is the temperature distribution, and S is the volumetric source function. Given in the following subsections are descriptions of the domain and partial differential equation (PDE) discretization to facilitate the FE method

3.1 Domain

The porous system domains were discretized using unstructured triangular meshes. Figure 5 notionally illustrates the triangle element structure formed by three vertices in a mesh,

where t is the triangle, $(x_{t,i}, y_{t,i})$, are the x and y coordinates of the i^{th} vertex in element t , $T_{t,i}$, is the temperature at vertex i in triangle t , and A_t is the area of triangle t .

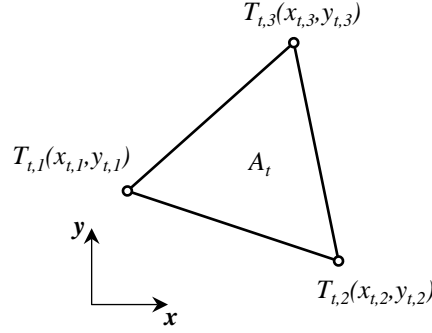


Figure 5. Notional triangle element

Systematic mesh refinement facilitated the verification methods described in subsequent sections of this report, where the mesh number, H , is used to describe the mesh refinement level of the domain, with $H=1$ and $H=5$ being the finest and coarsest meshes of a given domain, respectively. The characteristic mesh size, h_H , of mesh H , is computed as

$$h_H = \sqrt{\frac{1}{N_{H,t}} \sum_{t=1}^{N_{H,t}} A_t}, \quad (9)$$

where $N_{H,t}$ is the total number of triangles in the H^{th} mesh. Characteristic mesh sizes for meshes $H=5$ through $H=1$ used in this study were approximately 0.12, 0.07, 0.04, 0.02, and 0.01. Figure 6 shows an example mesh refinement used in this study for an example triangle pore configuration with α of 17.5% and θ_α of 17.5°. Figure 7 shows example fine meshes for a variety of α and θ_α combinations for triangle pores. Likewise, Figure 8 shows mesh refinement for an example square pore configuration with α of 17.5% and θ_α of 22.5°, and Figure 9 shows multiple fine meshes for square pore geometries. Lastly, Figure 10 gives an example mesh

refinement for an ellipse pore with α of 17.5% and θ_α of 45.0 °, Figure 11 shows fine mesh ellipse pore geometries for various α and θ_α combinations.

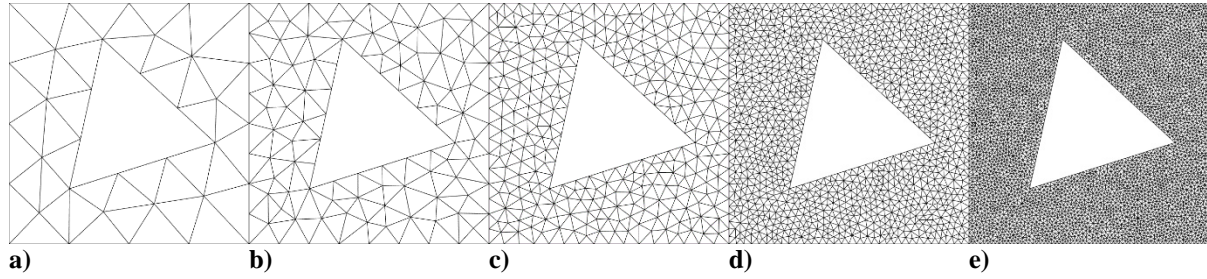


Figure 6. Domain mesh for triangle geometry at 17.5 ° clocking angle and 17.5% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

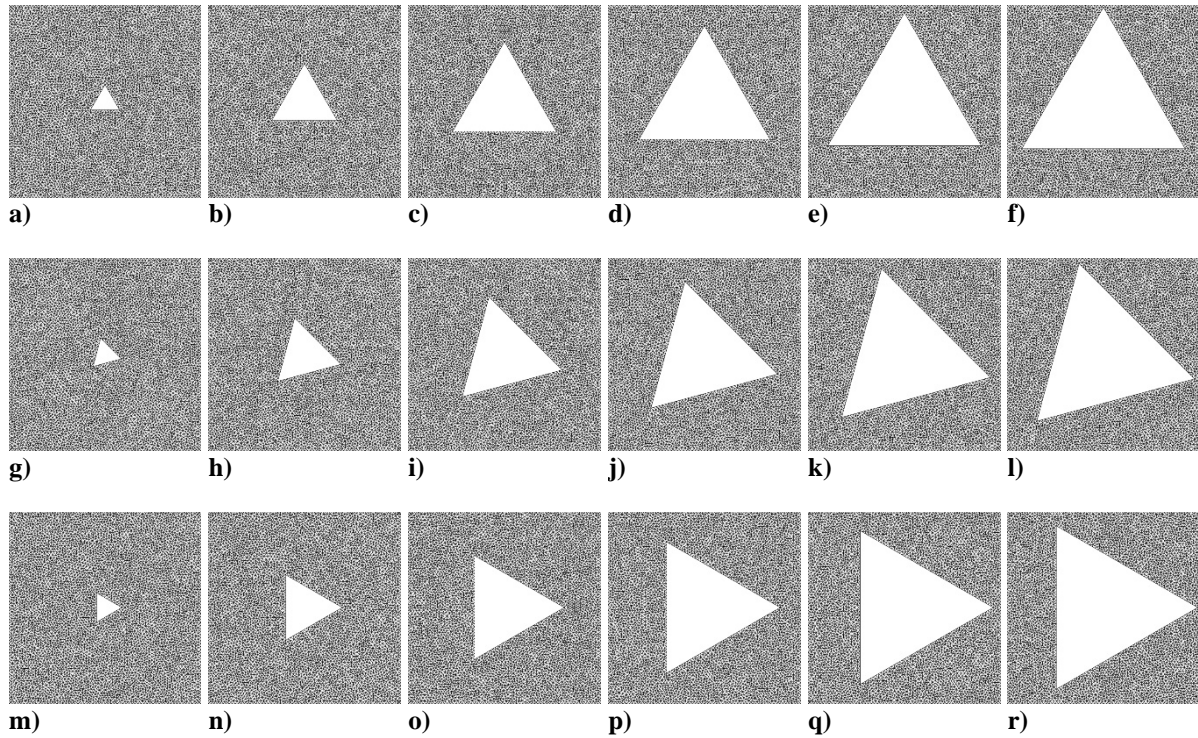


Figure 7. Finest mesh for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$

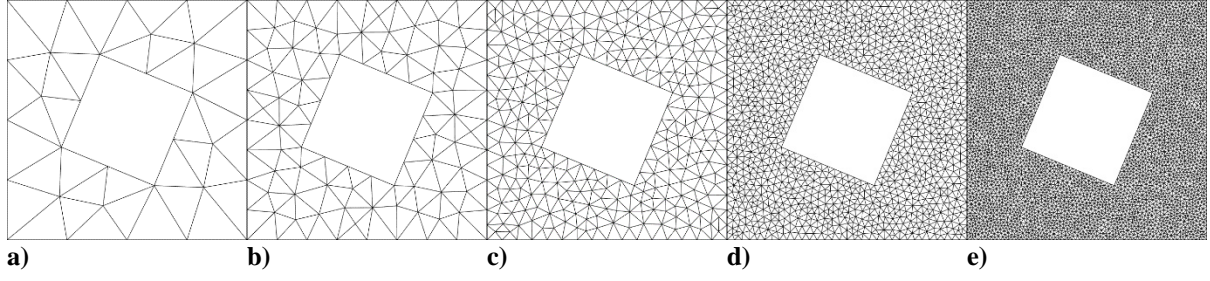


Figure 8. Domain mesh for square geometry at 22.5° clocking angle and 17.5% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

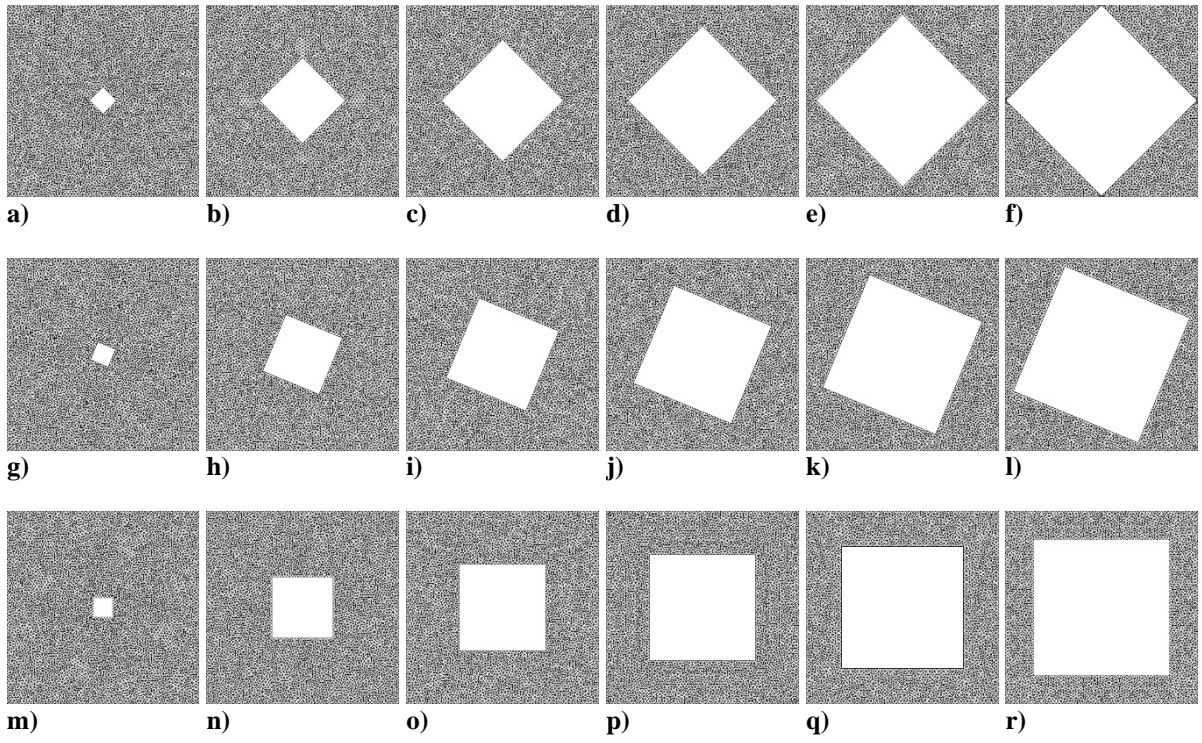
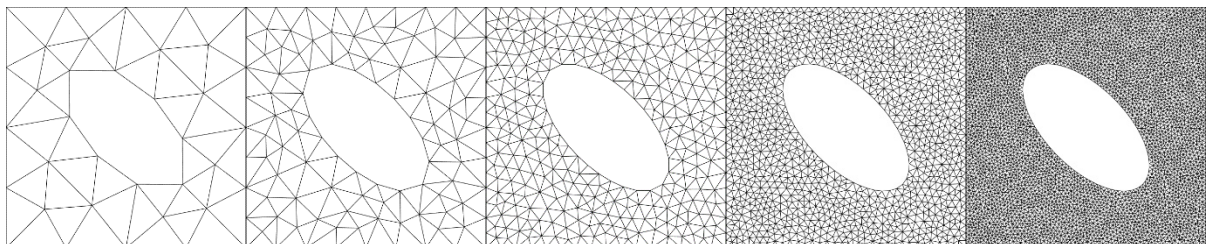


Figure 9. Finest mesh for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$



a) b) c) d) e)

Figure 10. Domain mesh for ellipse geometry at 45.0 ° clocking angle and 17.5% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

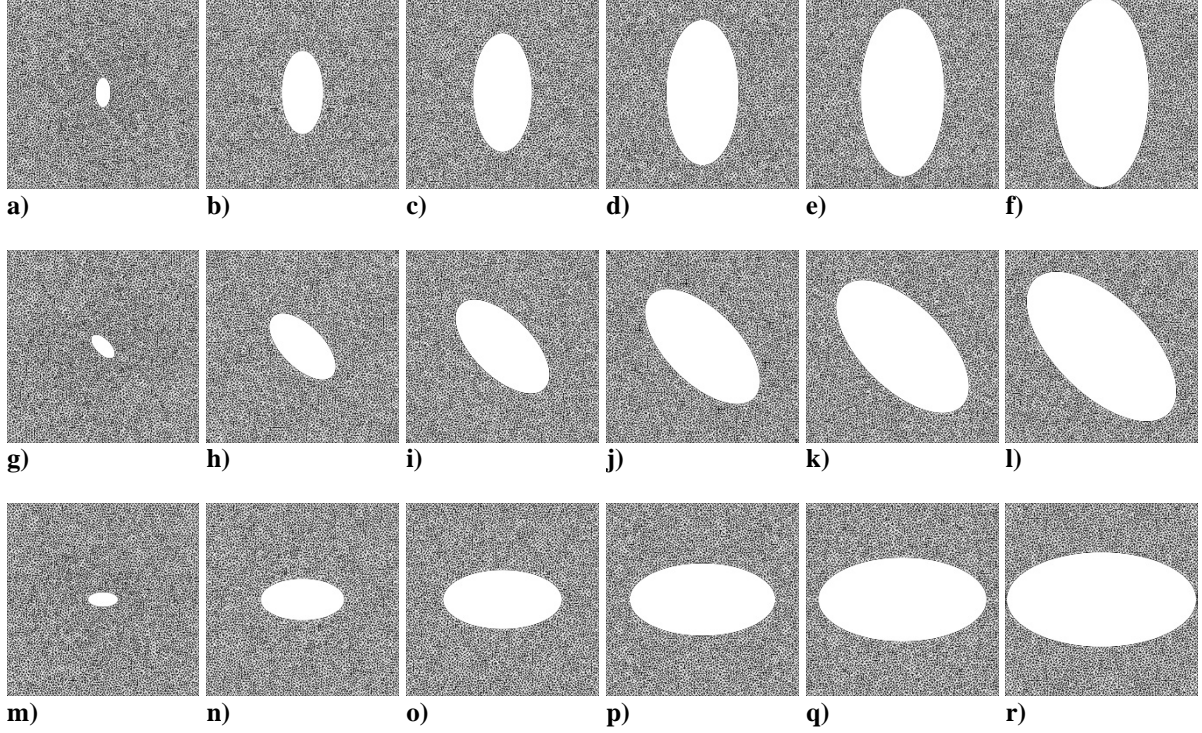


Figure 11. Finest mesh for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_a=0.0^\circ$, g) through l) $\theta_a=45.0^\circ$, and m) through r) $\theta_a=90.0^\circ$

3.2 Partial Differential Equation

A second order accurate Galerkin FE approach was used to solve the governing heat equation PDE over the solid computational domain. Equation (8) can be solved over each finite element, individually, using a discretized PDE matrix equation. The thermal conductivities and temperature gradients from Equation (8) can be cast in matrix form as

$$\mathbf{K} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \quad \text{and} \quad \nabla T = \begin{Bmatrix} \partial T / \partial x \\ \partial T / \partial y \end{Bmatrix}, \quad (10)$$

where \mathbf{K} is the thermal conductivity tensor. For the system at hand, isotropic thermal conductivity is used for both the matrix and filler materials, such that k_x and k_y are the same. From Fourier's law, it follows that the heat flux vector, \mathbf{q} , is comprised of x -directional and y -directional heat fluxes, q_x and q_y , respectively, and looks like

$$\mathbf{q} = \begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = -\mathbf{K}\nabla T. \quad (11)$$

At the boundary, Γ , of Ω that has a surface normal vector, \mathbf{n} , the normal heat flux leaving Ω , q_n , is described by

$$q_n = \mathbf{q}^T \cdot \mathbf{n}. \quad (12)$$

The heat equation of Equation (8) can then be recast as

$$S - \text{div}(\mathbf{q}) = S + \text{div}(\mathbf{K}\nabla T). \quad (13)$$

This work uses the Galerkin FE approach to solve the governing PDE which requires the use of some weighting function, w , over the PDE. Multiplying Equation (13) by w and integrating over Ω results in the following

$$\int_{\Omega} (\nabla w) \mathbf{K} \nabla T d\Omega = \int_{\Omega} w S d\Omega - \int_{\Gamma} w q_n d\Gamma. \quad (14)$$

For each triangle in the finite element domain, Equation (14) is true, and the Galerkin method prescribes that both w and T be interpolated across the finite element domain using the same row vector polynomial shape function, or interpolation function, N . If a triangle element

has weighting function and temperature values defined at each of its three vertices as $w_{t,i}$ and $T_{t,i}$, respectively, for $i=1,2,3$ corresponding to each of the i^{th} vertices of the t^{th} triangle, then let

$$\mathbf{T} = \begin{Bmatrix} T_{t,1} \\ T_{t,2} \\ T_{t,3} \end{Bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{Bmatrix} w_{t,1} \\ w_{t,2} \\ w_{t,3} \end{Bmatrix}, \quad (15)$$

for a given triangle. Thus, the field variable and weight function within the bounds of the finite element are described similarly as

$$T(x, y) = \mathbf{N}\mathbf{T} \quad \text{and} \quad w(x, y) = \mathbf{N}\mathbf{w}, \quad (16)$$

using the row vector interpolation function, \mathbf{N} . By using matrix transpose rules, substituting the definitions of Equation (16) into Equation (14), and eliminating common constant terms, the discretized form of the governing PDE solved over each linear triangle element in the discretized domain is

$$\int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \mathbf{T} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma. \quad (17)$$

In the discretized PDE for this work, \mathbf{N} is the linear interpolation shape function row vector for elemental field variables.

Portions of Equation (17) can be further condensed and defined as the conductance matrix, \mathbf{G} , and the heat load vector \mathbf{P} , where

$$\mathbf{G} = \int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \quad (18)$$

and

$$\mathbf{P} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma \quad (19)$$

such that

$$\mathbf{GT} = \mathbf{P}. \quad (20)$$

4 Solution and Evaluation

Open-source Gmsh software [33] was used to discretize the computational domain into unstructured triangle meshes, and personal Fortran code was written to solve the two-dimensional heat equation using the Galerkin FE method. A conjugate gradient iterative solver scheme was implemented to update solutions to the heat equation until the root mean square (RMS) of the energy equation residuals across the domain fell below 10^{-8} . To provide credibility evidence for the summary results obtained in this study, code and solution verification were performed. The following subsections briefly describe the solution update method and both verification processes used.

4.1 Solution

In the FE method, the imbalance of Equation (20) is considered the residual energy balance at each vertex, ρ_i , where $\boldsymbol{\rho}$ is the vector of vertex residuals, and

$$\boldsymbol{\rho} = \mathbf{P} - \mathbf{GT}. \quad (21)$$

The implementation of the numerical method employed a conjugate gradient method to update the solution of Equation (20), driving energy residuals to zero. The iterative conjugate gradient scheme initializes (iteration $i=0$) the direction vector, \mathbf{p}_0 , as

$$\mathbf{p}_0 = \boldsymbol{\rho}_0. \quad (22)$$

On the i^{th} iteration, starting at $i=0$, coefficient, $c_{1,i}$ is

$$c_{1,i} = (\boldsymbol{\rho}_i^T \boldsymbol{\rho}_i) / (\boldsymbol{p}_i^T \boldsymbol{G} \boldsymbol{p}_i), \quad (23)$$

the temperature solution is updated by

$$\boldsymbol{T}_{i+1} = \boldsymbol{T}_i + c_{1,i} \boldsymbol{p}_i, \quad (24)$$

and the residuals are updated as

$$\boldsymbol{\rho}_{i+1} = \boldsymbol{\rho}_i - c_{1,i} \boldsymbol{G} \boldsymbol{p}_i. \quad (25)$$

For further iteration, a coefficient, $c_{2,i}$, is defined as

$$c_{2,i} = (\boldsymbol{\rho}_{i+1}^T \boldsymbol{\rho}_{i+1}) / (\boldsymbol{\rho}_i^T \boldsymbol{\rho}_i), \quad (26)$$

and the direction vector for the next iteration is updated to be

$$\boldsymbol{p}_{i+1} = \boldsymbol{\rho}_{i+1} + c_{2,i} \boldsymbol{p}_i. \quad (27)$$

The process terminates when sufficiently small residual conditions are met. In this work, the RMS of residuals, ρ_{RMS} , was used as termination criteria, where

$$\rho_{RMS} = \sqrt{\sum_{i=1}^{N_v} (\rho_i^2) / N_v}, \quad (28)$$

and a ρ_{RMS} value of 1.0×10^{-8} was used as the exit value. Although the RMS was used as exit criteria for this study, the L_∞ norm of residuals, ρ_{L_∞} , was computed and retained, where

$$\rho_{L\infty} = \max_{1 \leq i \leq N_v} |\rho_i|. \quad (29)$$

4.2 Code Verification

The method of manufactured solutions (MMS) was used on each porous system to show convergence of the observed order of accuracy of the numerical methods used for this work [34,35]. The order of accuracy of a finite element computational method describes the rate of change in numerical solution error with respect to change in finite element size. For example, a second order accurate model is a model whose solution error is quartered when the mesh elements are halved. In most meaningful engineering systems, an exact solution for the governing PDEs over the system does not exist, thus a surrogate exact solution is constructed. MMS allows one to define a solution to the governing PDE, T_{MMS} , for the system and determine what boundary conditions and source terms satisfy that solution. By then applying the determined boundary conditions and source terms to the model, the simulations should return solutions approximating T_{MMS} (the “true” solution). In this way, one need not find an analytical solution to a given problem in order to determine the accuracy of the simulation results.

For this study, T_{MMS} was selected to be

$$T_{MMS}(x, y) = \cos(2\pi x) \sin(\pi y + 0.75), \quad (30)$$

where its first and second spatial derivatives are

$$\begin{aligned}
\frac{\partial T}{\partial x} &= -2\pi \sin(2\pi x) \sin(\pi y + 0.75) \\
\frac{\partial T}{\partial y} &= \pi \cos(2\pi x) \cos(\pi y + 0.75) \\
\frac{\partial^2 T}{\partial x^2} &= -4\pi^2 \cos(2\pi x) \sin(\pi y + 0.75) \\
\frac{\partial^2 T}{\partial y^2} &= -\pi^2 \cos(2\pi x) \sin(\pi y + 0.75)
\end{aligned} \tag{31}$$

By using the MMS derivatives from Eq. (31) in Eq. (8), the resulting MMS volumetric source distribution, S_{MMS} , is evaluated as

$$S_{MMS}(x, y) = 5\pi^2 k \cos(2\pi x) \sin(\pi y + 0.75). \tag{32}$$

To solve the MMS problem, all boundary conditions on the system are replaced with values that correspond to the T_{MMS} and its derivatives, consistent with the boundary condition types in the true problem of interest. Finally, S_{MMS} is applied across the entire computational domain. The system is then solved numerically, where the computational solution is expected to approach the prescribed T_{MMS} with mesh refinement.

With k set to 100.0 for this study, Figure 12 and Figure 13 show the selected MMS temperature field and applied MMS source field contours, respectively, on various triangle geometry α and θ_α system configurations. Figure 14 and Figure 15 show the selected MMS temperature field and applied MMS source field contours, respectively, on various square geometry α and θ_α system configurations. Figure 16 and Figure 17 show the selected MMS temperature field and applied MMS source field contours, respectively, on various ellipse geometry α and θ_α system configurations.

The observed order of accuracy, $p_{O,H}$, for mesh H of each system configuration was expected to converge to the formal order of accuracy, p_f , of 2 with mesh refinement, as

predicted by the second order accurate Galerkin FE method used. Nodal temperature solutions across the mesh are compared against the prescribed T_{MMS} distribution for MMS convergence evaluation.

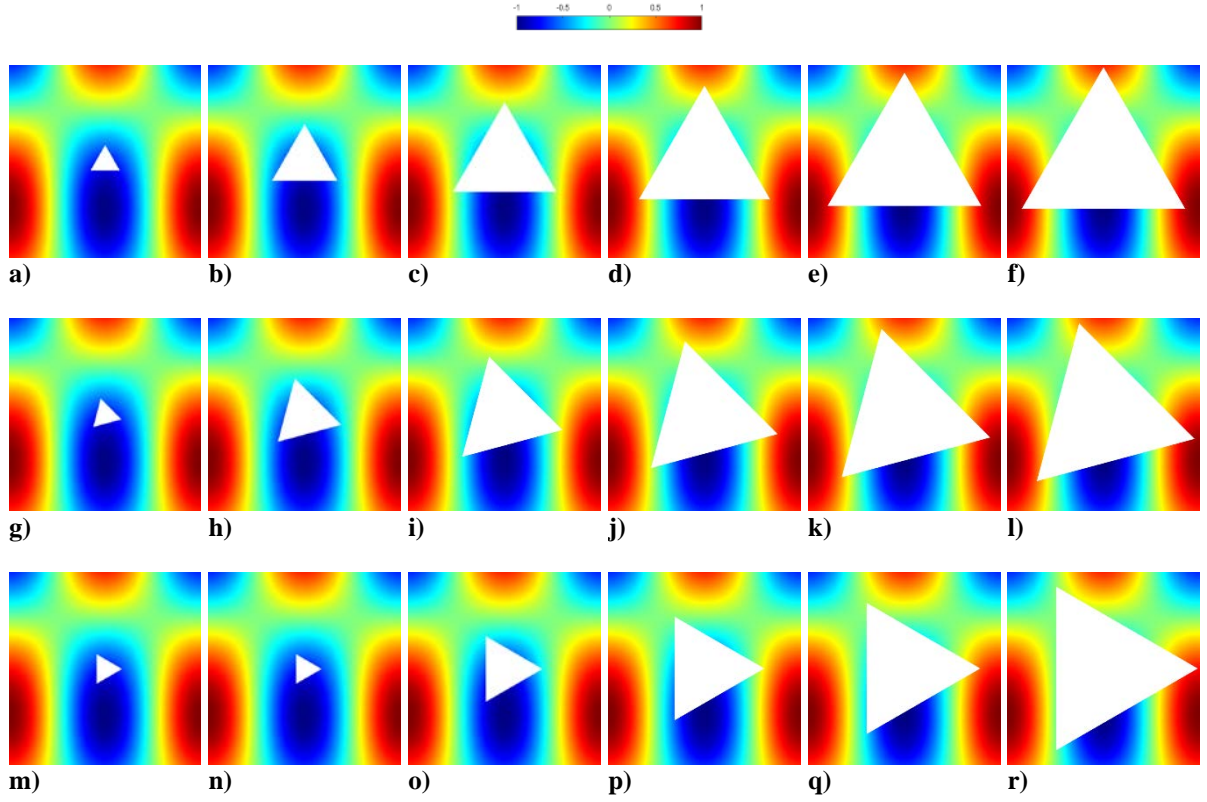
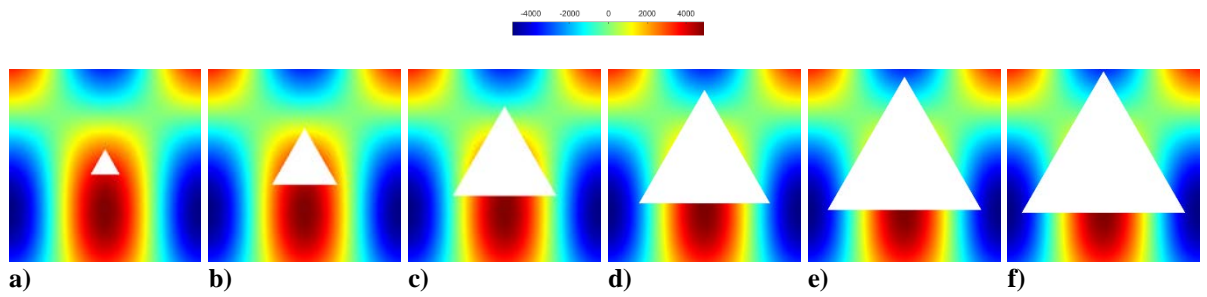


Figure 12. Prescribed MMS temperature field contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5,$ and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$



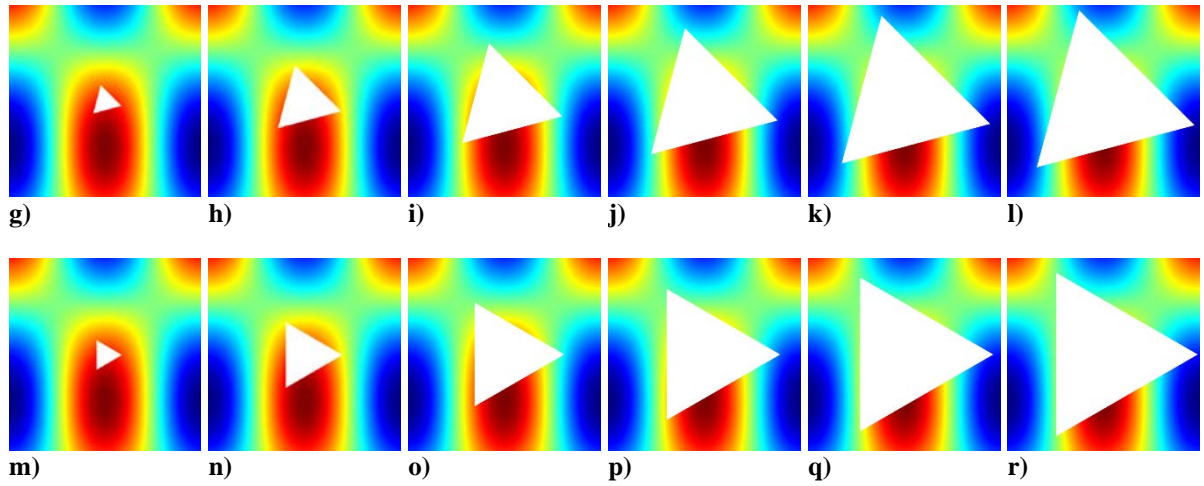


Figure 13. Applied MMS source field contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5,$ and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$

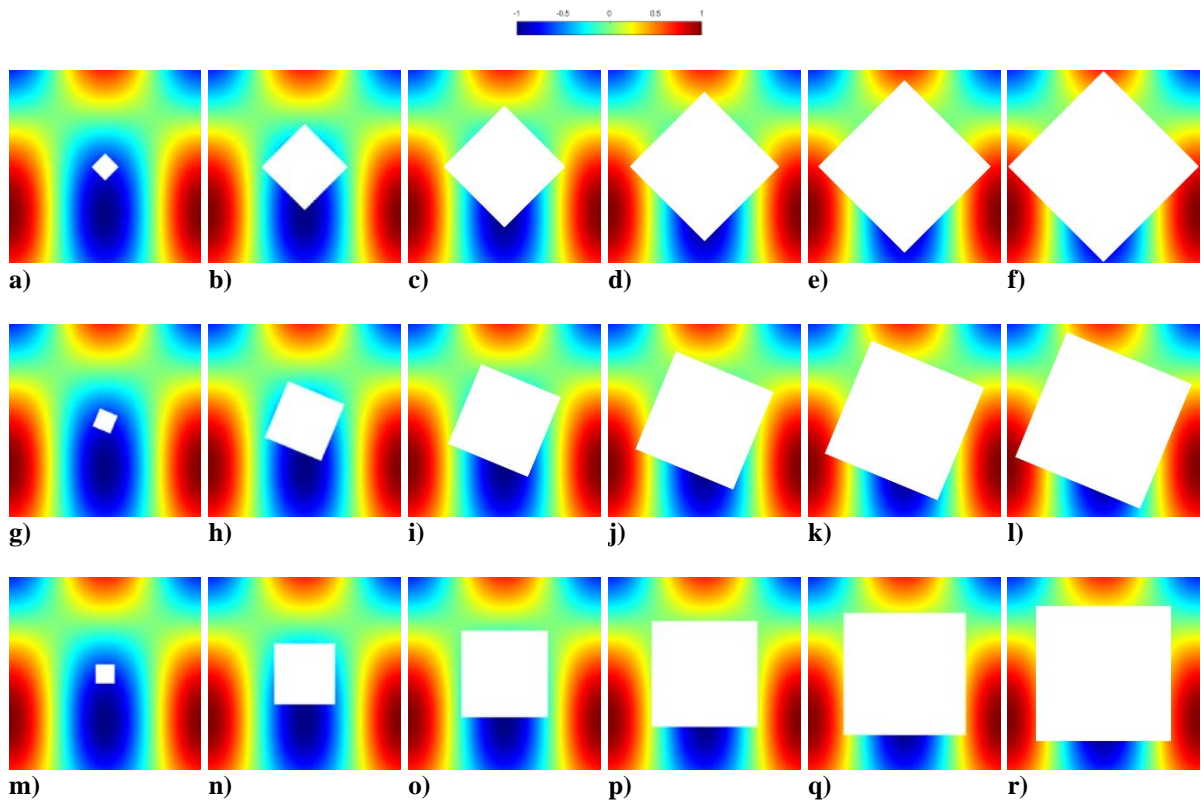


Figure 14. Prescribed MMS temperature field contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0,$ and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$

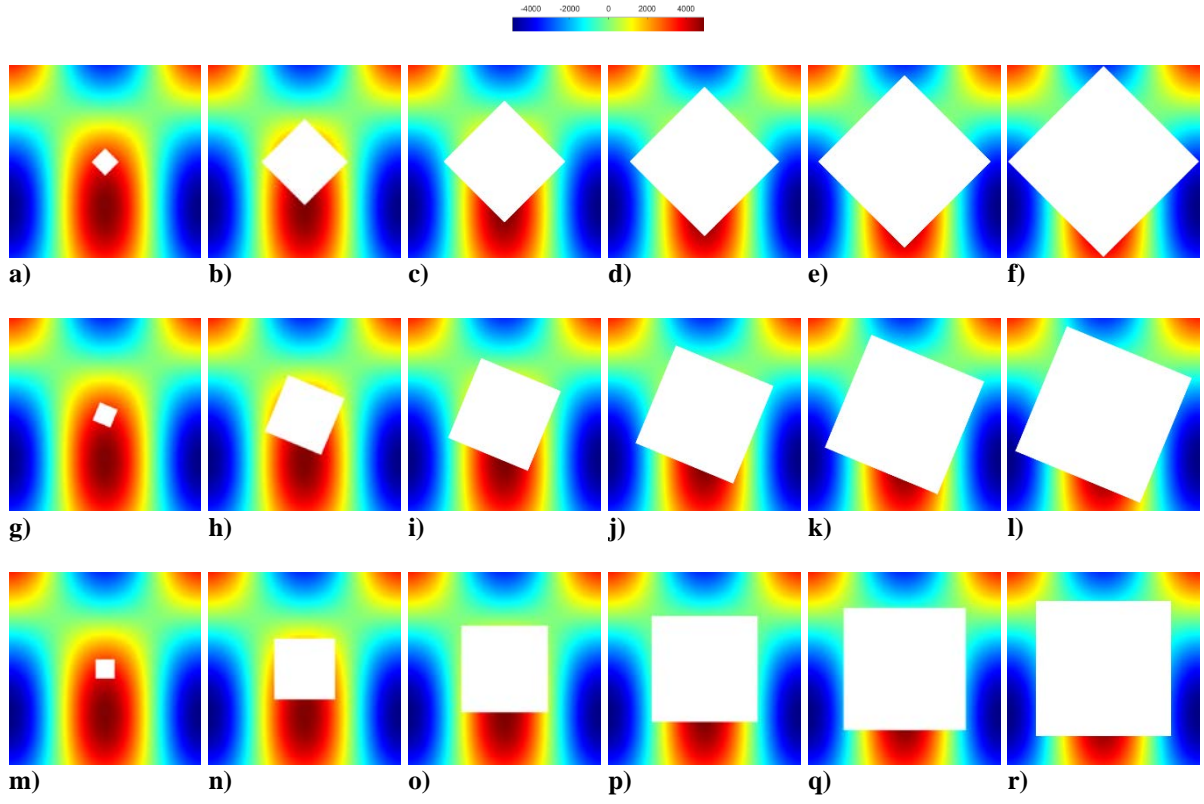
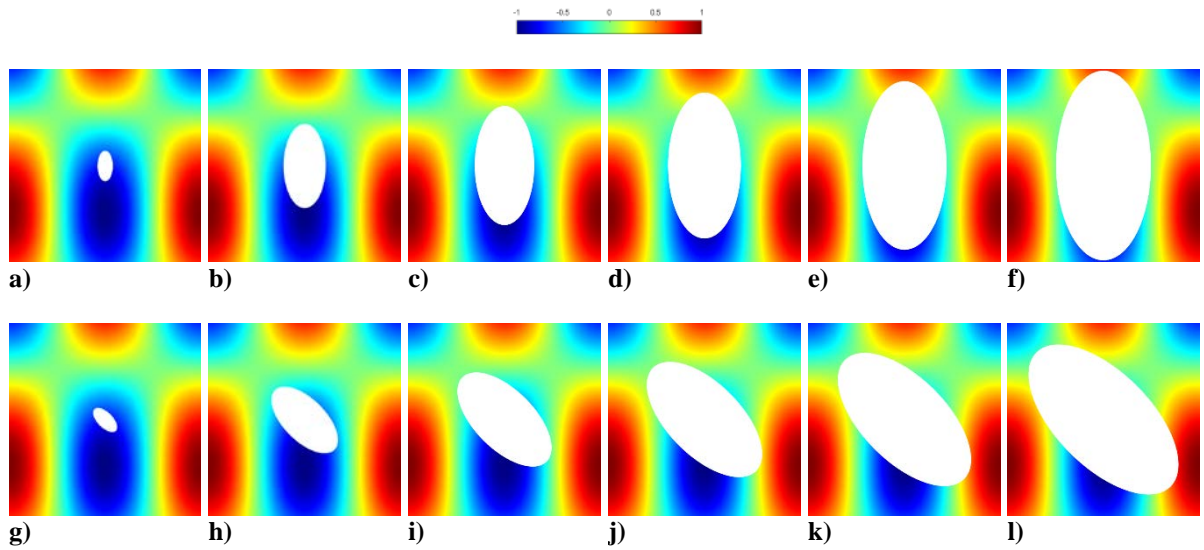


Figure 15. Applied MMS source field contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0,$ and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$



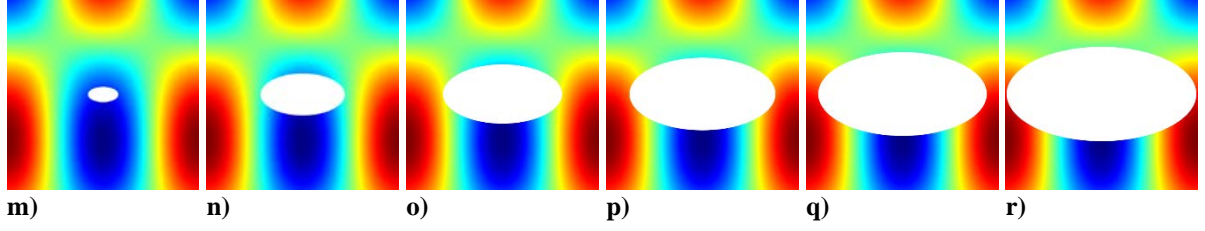


Figure 16. Prescribed MMS temperature field contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

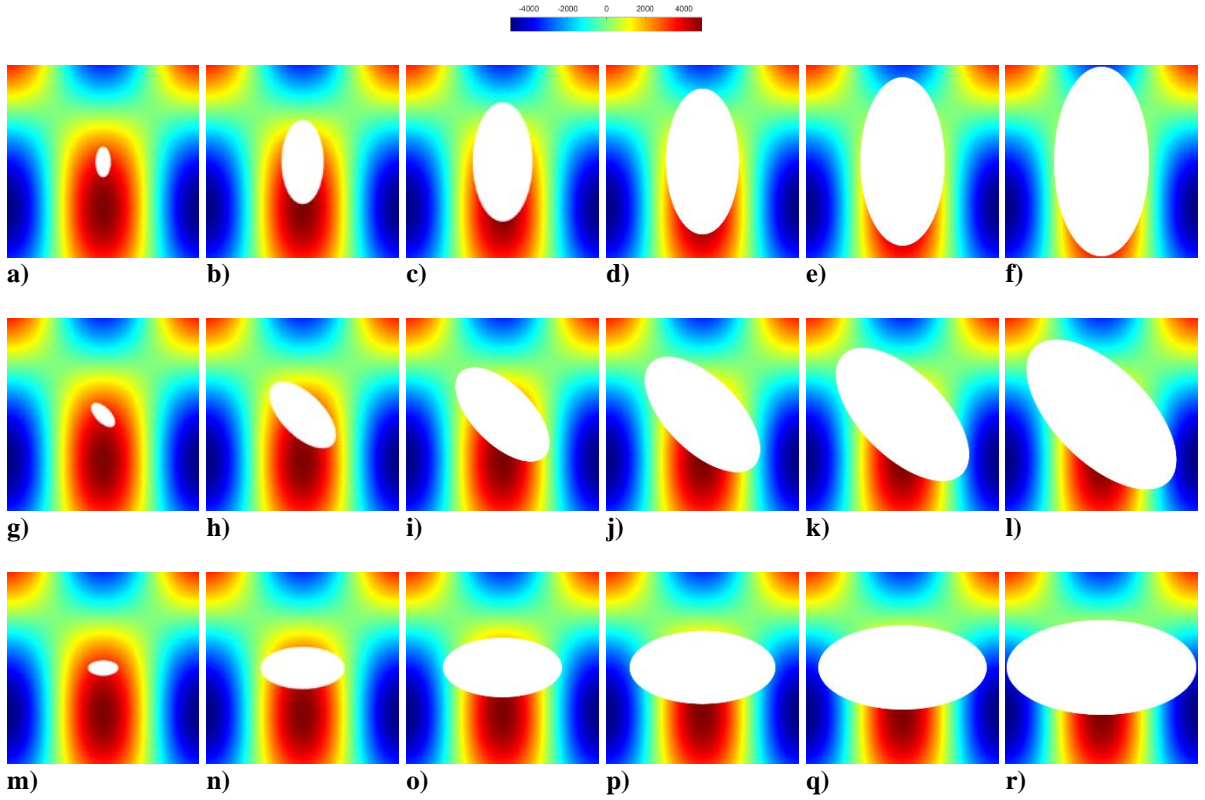


Figure 17. Applied MMS source field contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

4.3 Solution Verification

Solution verification was performed on k^* for the problem of interest. A global deviation grid convergence index (GCI) approach was used to estimate the numerical error,

U_{num} , on k^* [36]. A series of five systematically refined meshes was used for each system configuration to evaluate convergence of k^* with mesh refinement. Theory suggests that as a computational mesh is refined, the numerical solution will asymptotically approach some distinct solution. When the solution is in the region of the asymptotic solution, the solution is said to be in the “asymptotic regime.” Conversely, solutions that are not converging systematically near the asymptotic solution are said to be in the “non-asymptotic regime.” The empirically-based global deviation GCI method computes a factor of safety on the finest mesh k^* solution value based on a quantitative estimation of where the solution falls with respect to the asymptotic solution regime.

To approximate U_{num} with this global deviation GCI method on a fine mesh, first a modified transcendental order of accuracy on the finest of three meshes, p_t , is determined iteratively from

$$p_t = \ln \left[(r_{1,2}^{p_t} - 1) \left(\left| \frac{k_3^* - k_2^*}{k_2^* - k_1^*} \right| \right) + r_{1,2}^{p_t} \right] / \ln(r_{1,2} r_{2,3}), \quad (33)$$

where the subscripts 1, 2, and 3 denote the finest, middle, and coarsest of three meshes, successively refined. Likewise, $r_{i,j}$ is the mesh ratio between two successive meshes, where

$$r_{i,j} = h_j / h_i. \quad (34)$$

For this application, a global order of accuracy deviation value, Δp , is determined by

$$\Delta p = \min(|p_f - p_t|, 4p_f, 0.95p_f). \quad (35)$$

The deviation is used to determine the average global order of accuracy, p^* , defined as

$$p^* = p_f - \Delta p, \quad (36)$$

which in turn is used to compute a factor of safety, FS —based on the inferred proximity of the solution to the asymptotic regime—where,

$$FS = 3.0 - 1.9(p^*/p_f)^8. \quad (37)$$

Figure 18 depicts the behavior and smooth transition of FS with respect to the global order of accuracy deviation for a formally second order accurate method. In the asymptotic regime, FS approaches 1.1, and in the non-asymptotic regime, FS approaches 3.0. The original GCI method assumed that FS would take on the value of either 1.25 or 3.00 depending on the number of meshes and the observed order of accuracy [37], however the method used here continuously scales FS .

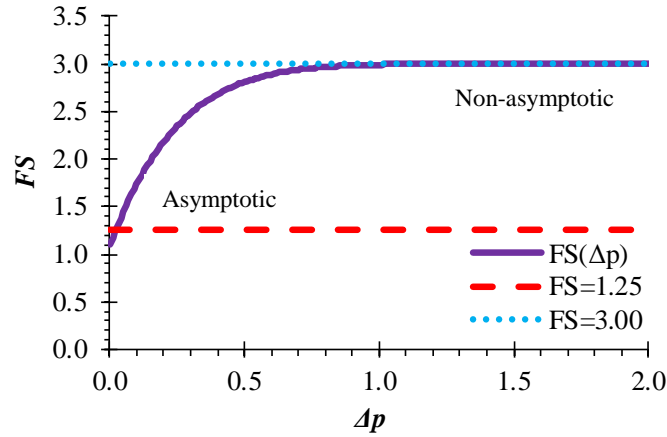


Figure 18. Global deviation estimator factor of safety

An estimate for U_{num} on k^* from the finest of the three systematically refined meshes is

$$U_{num} = FS \left| (k_2^* - k_1^*) / (r_{1,2}^{p^*} - 1) \right|, \quad (38)$$

providing a convergence-based uncertainty estimator for discretization-induced error.

5 Results and Discussion

Using the MMS solution described above for all the geometry combinations given in Table 1 the MMS observed order of accuracy is shown to converge to approximately second order for all the analyzed systems, where the computational solution temperature is compared against the prescribed T_{MMS} distribution. Figure 19 shows an example of the computational MMS temperature solution on a triangle pore mesh. The solution was solved with the applied MMS source term distribution described previously and boundary conditions enforced on the domain consistent in type with the true problem and consistent in value with T_{MMS} . Figure 20 shows the computational MMS temperature solution for fine meshes on a variety of triangle pore system configurations. Likewise, Figure 21 and Figure 22 show computational MMS temperature solutions with mesh refinement and on fine meshes, respectively, for square pore geometries. Lastly, Figure 23 and Figure 24 show computational MMS temperature solutions with mesh refinement and on fine meshes, respectively, for ellipse pore system configurations. Note that for each pore type, the numerical temperature distribution approaches the expected, prescribed MMS temperature distribution. Similarly, note that the fine mesh computational MMS solutions are virtually qualitatively indistinguishable from the prescribed MMS temperature solution.

Figure 25 through Figure 27 illustrate the convergence of $p_{O,H}$ with H , where p_f is indicated by the dashed line, suggesting that the numerical method was correctly implemented to solve the heat equation, being second order accurate.

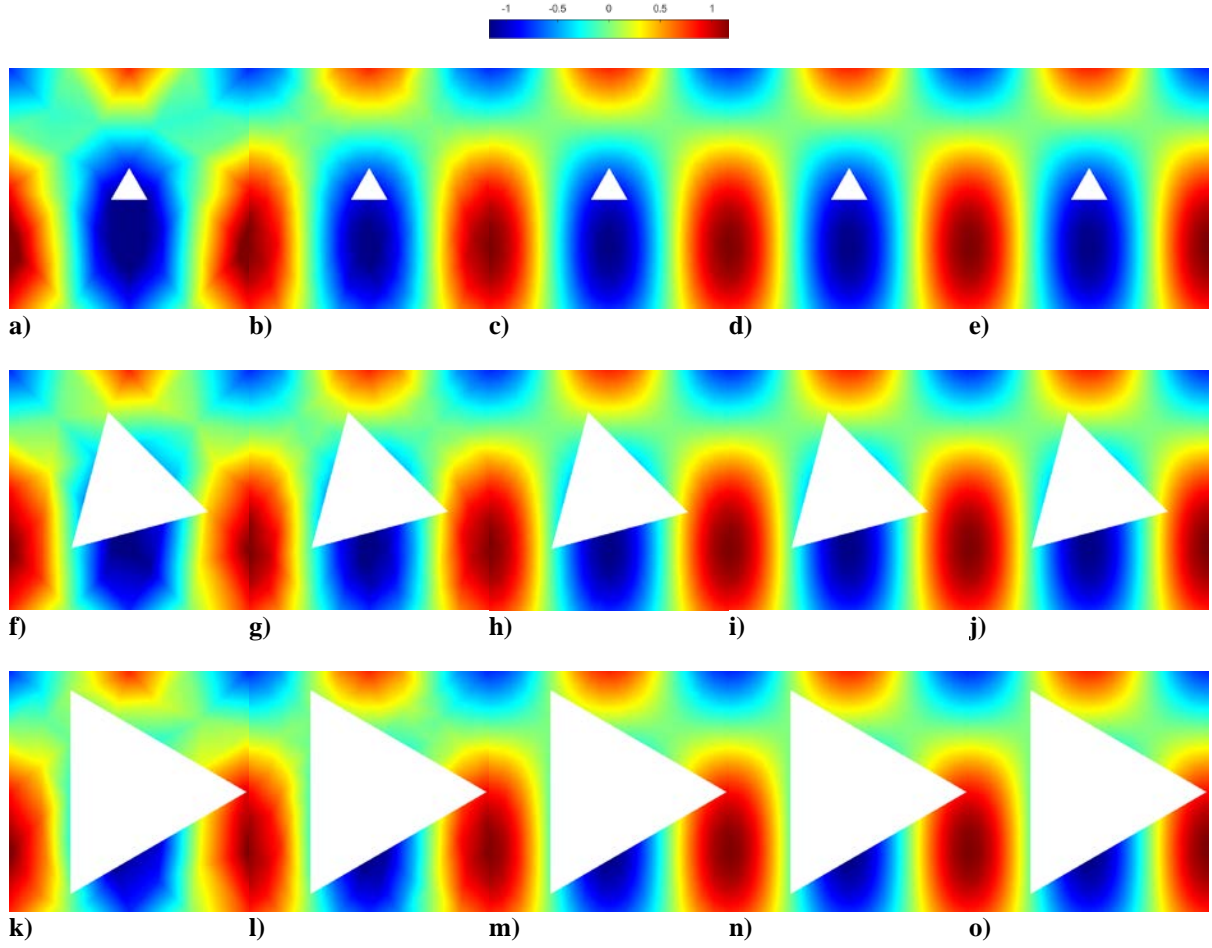
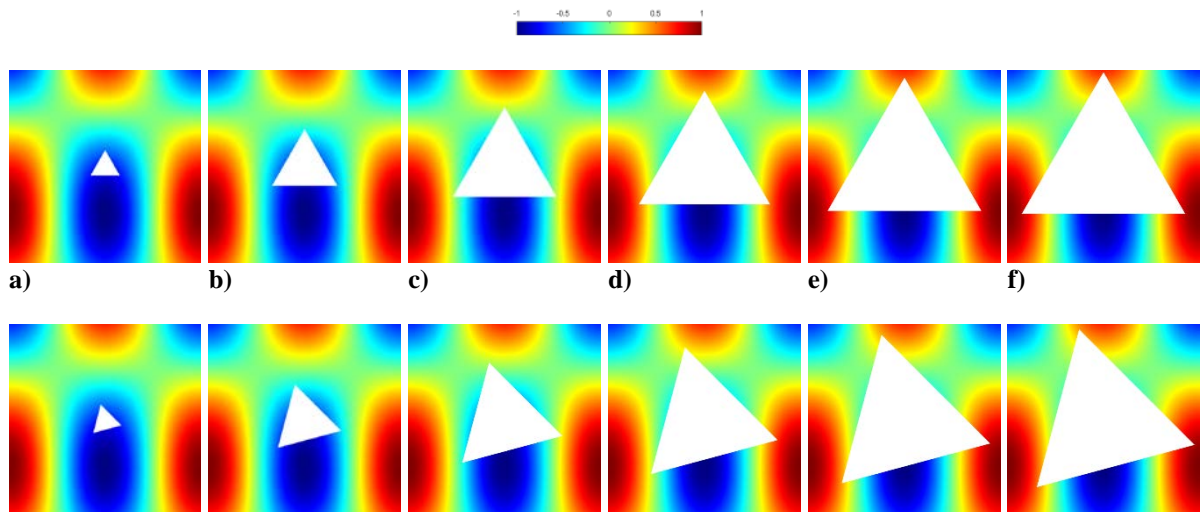


Figure 19. Computational MMS temperature contours for triangle geometries with mesh refinement for $H=5, 4, 3, 2, 1$ with a) through e) $\alpha=1.0\%$ and $\theta_\alpha=0.0^\circ$, f) through j) $\alpha=15.0\%$ and $\theta_\alpha=15.0^\circ$, and k) through o) $\alpha=31.0\%$ and $\theta_\alpha=30.0^\circ$



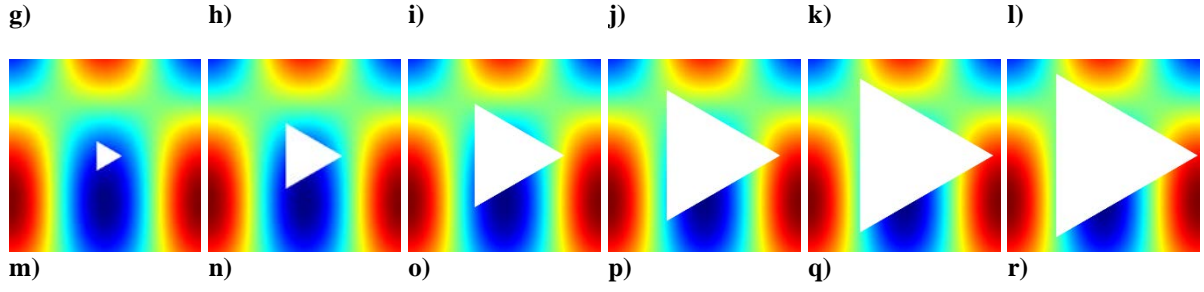


Figure 20. Computational MMS temperature contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$

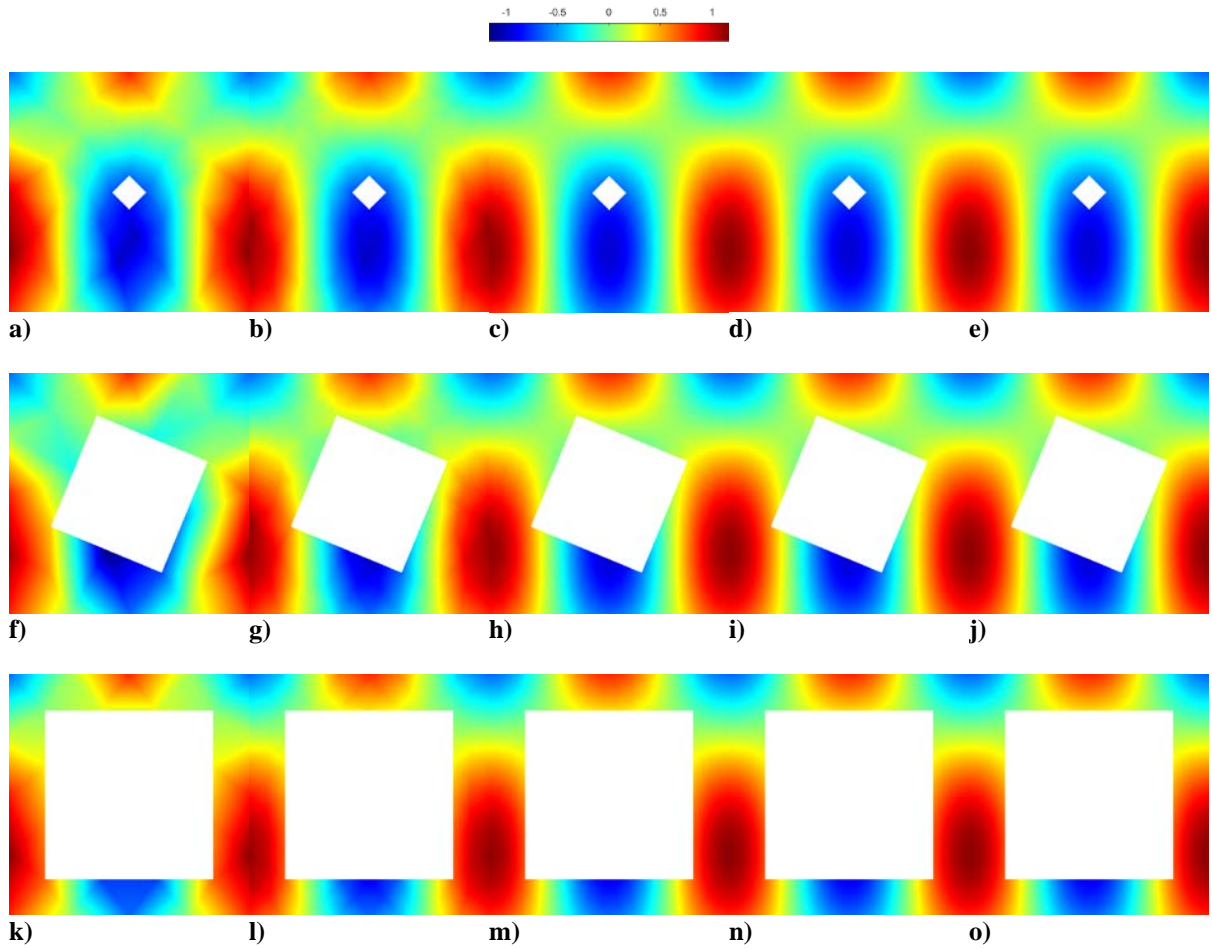


Figure 21. Computational MMS temperature contours for square geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_\alpha=0.0^\circ$, f) through j) $\alpha=25.0\%$ and $\theta_\alpha=22.5^\circ$, and k) through o) $\alpha=49.0\%$ and $\theta_\alpha=45.0^\circ$

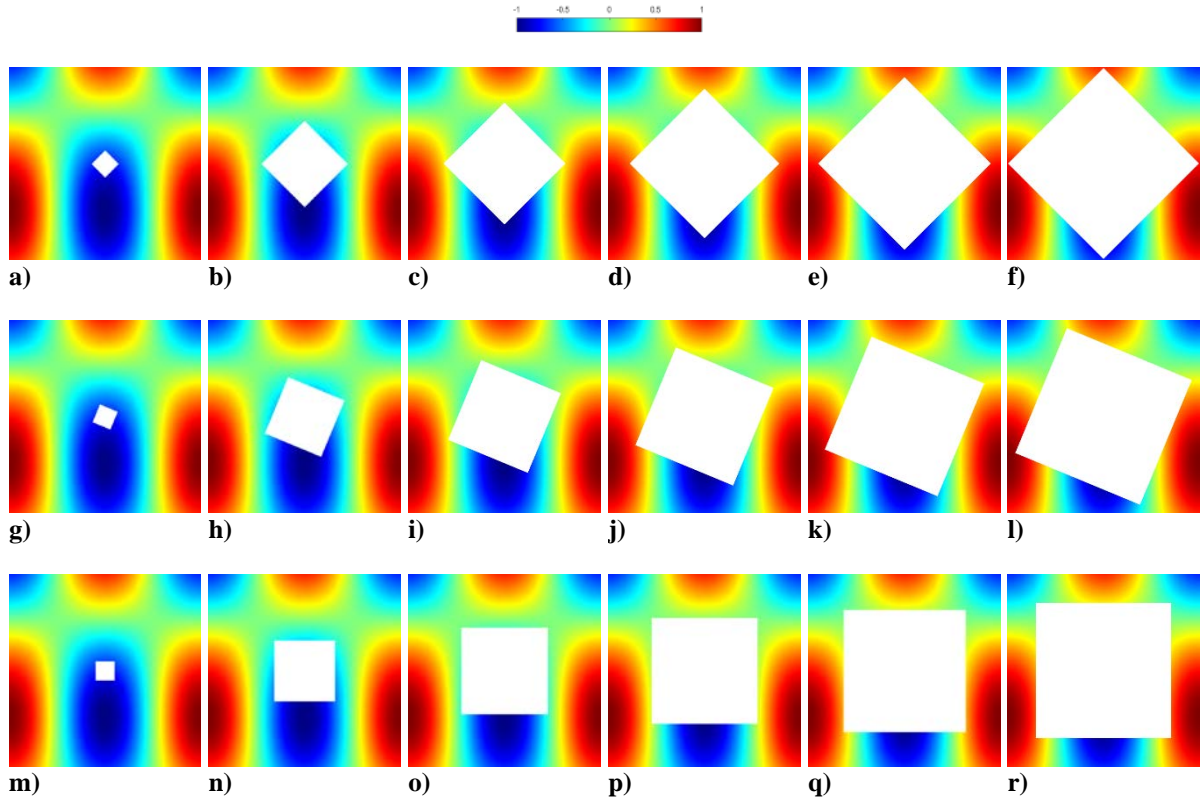
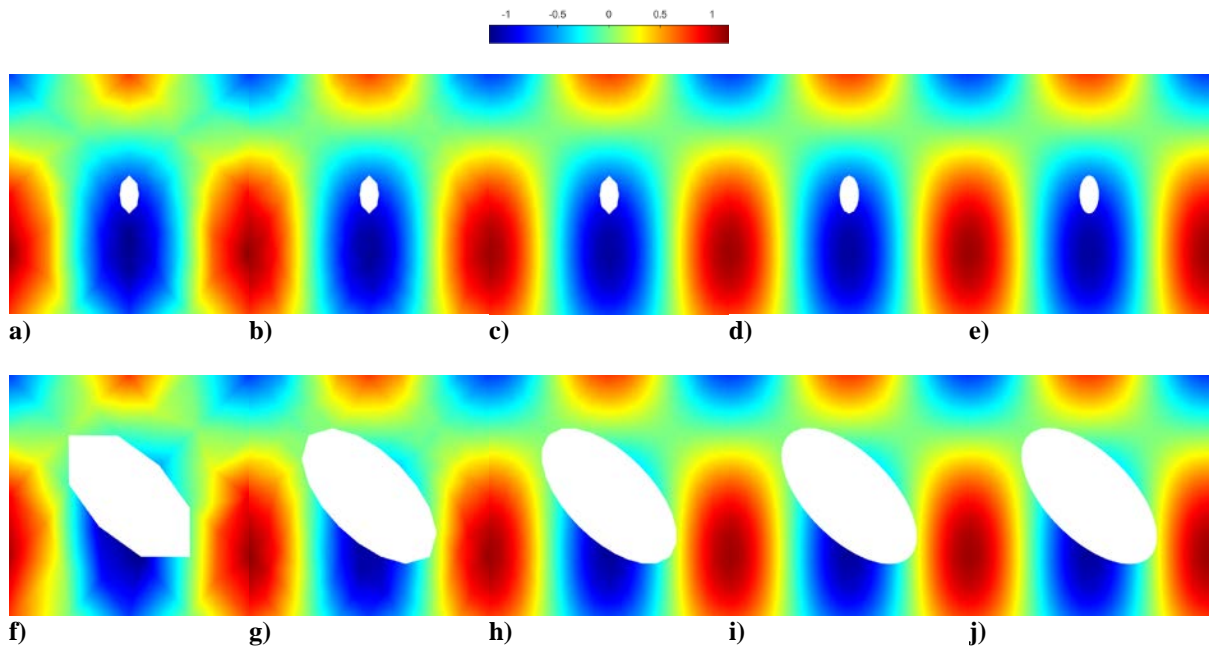


Figure 22. Computational MMS temperature contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$



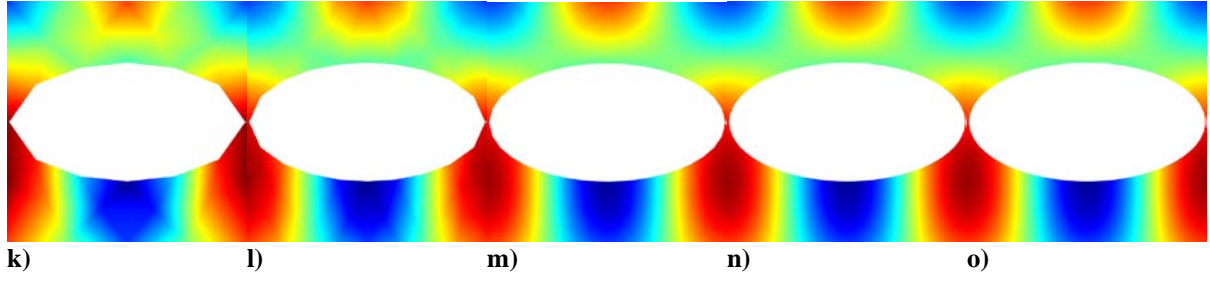


Figure 23. Computational MMS temperature contours for ellipse geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_a=0.0^\circ$, f) through j) $\alpha=20.0\%$ and $\theta_a=45.0^\circ$, and k) through o) $\alpha=38.0\%$ and $\theta_a=90.0^\circ$

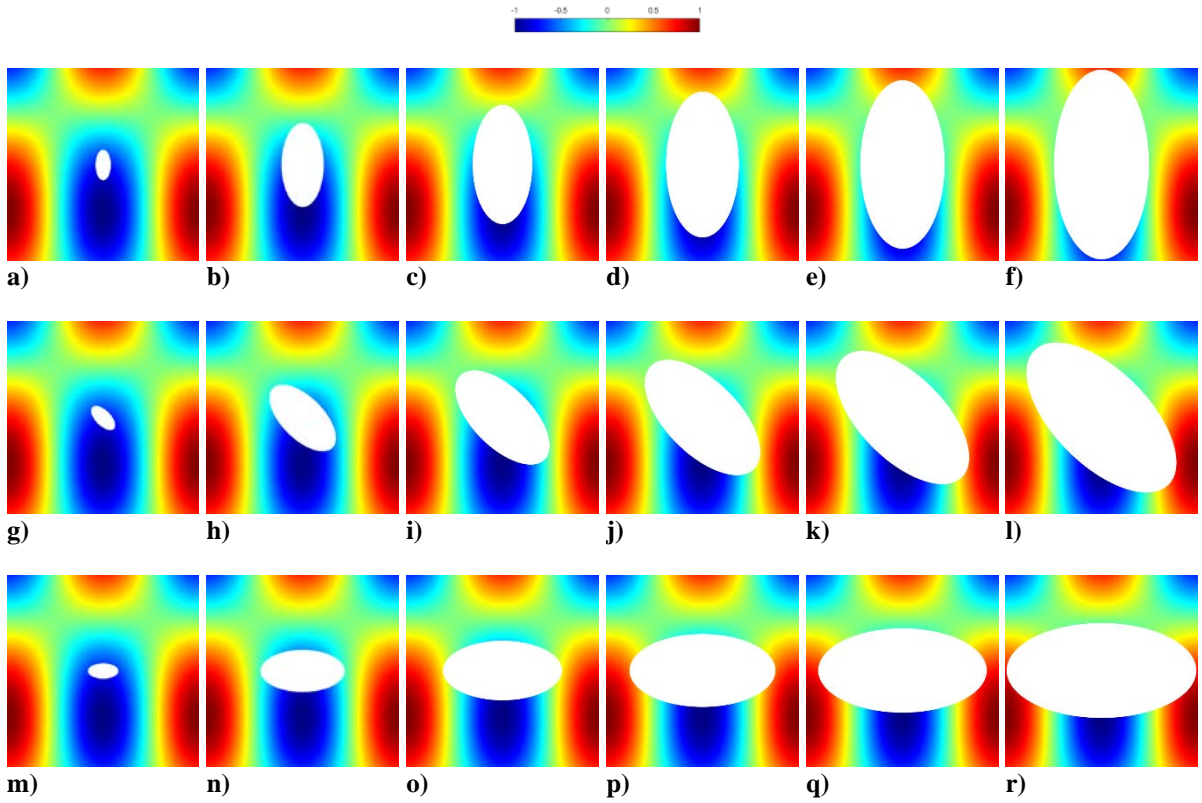


Figure 24. Computational MMS temperature contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_a=0.0^\circ$, g) through l) $\theta_a=45.0^\circ$, and m) through r) $\theta_a=90.0^\circ$

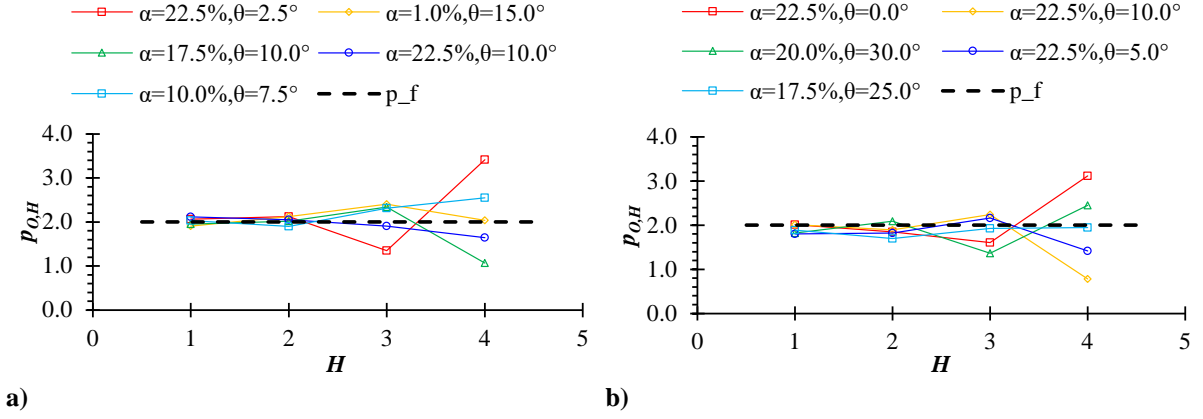


Figure 25. MMS observed order of accuracy convergence with mesh number for select triangle pore systems using a) RMS error and b) L_∞ error

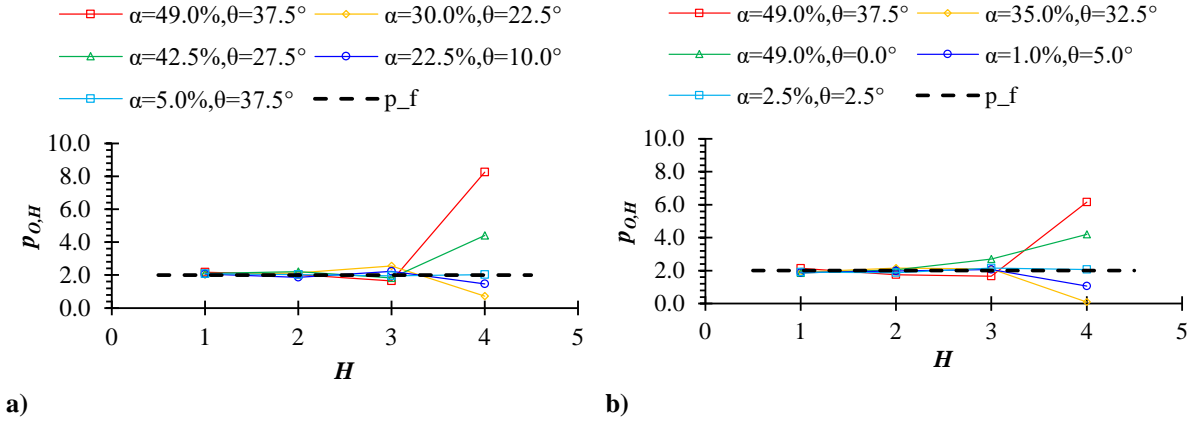


Figure 26. MMS observed order of accuracy convergence with mesh number for select square pore systems using a) RMS error and b) L_∞ error

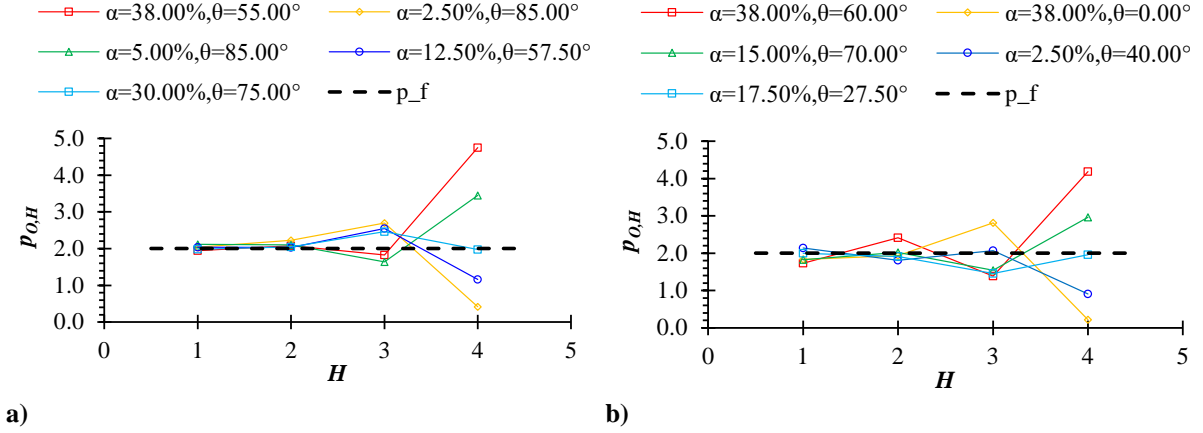


Figure 27. MMS observed order of accuracy convergence with mesh number for select ellipse pore systems using a) RMS error and b) L_∞ error

Systematically refined meshes were used with the GCI method to estimate U_{num} in the computationally-determined k^* for each system configuration. Figure 28, Figure 30, and Figure 32 show resultant temperature distribution contour plots for selected geometry configurations with mesh refinement with qualitative convergence on the temperature distribution solutions. Figure 29, Figure 31, and Figure 33 present contour plots for the finest meshes of changing α and θ_α for each of the three pore geometries. These figures show the qualitative effects of pore size and pore rotation on the temperature distributions with the unit cell. From the contour plots, it is evident that increasing pore size generally increases temperature gradient near the centerline of the system between the hot and cold boundaries. Rotation of the pores has asymmetric effects on the temperature profiles with varying effects on the gradients.

By normalizing clocking angles by 180° , rotation of the pore can be defined on a scale of 0 to 1, where the dimensionless clocking angle, ψ , is simply

$$\psi = \theta_\alpha / 180^\circ. \quad (39)$$

Resulting k^* curve families are shown in Figure 34 for the triangle, square, and ellipse pore geometries as determined by the finest mesh computational solutions. In the plots, the dashed lines are extensions of the computed values based on geometric symmetry. It is evident from these plots that the effects of θ_α on k^* are generally accentuated with increasing α , and k^* decreases monotonically with increased α , as expected. However, k^* appears to have virtually no change with respect to ψ for constant α in the triangle pore systems. The constant k^* values are likely due to the invariable integral of the contraction and restriction profiles in the direction of the domain temperature gradient. Note the relatively disparate behaviors of the different geometry systems.

Figure 35a through Figure 35c show the estimated U_{num} values as computed using the global deviation GCI method on the finest mesh k^* values, normalized by the k^* values at the respective positions. The triangular pore geometry numerical uncertainties are generally constant with changing θ_α . The higher uncertainty levels are manifested for the square pore geometries when the solid domain is restricted at the adiabatic domain edges near $x=L/2$. However, U_{num} stays below 6.0% for the triangle systems in all but a single point, where the numerical uncertainty is still less than 10.0%. For the square geometries, uncertainties remain below 6.0% in all but four data points. The highest uncertainty value is 37.0% for the highest porosity at 0.0 ° rotation. This is likely due to poor mesh refinement between the tips of the pore and the adiabatic external boundaries where the largest temperature gradients occur due to heat path restrictions. Note that the color legend in Figure 35b was set to a maximum of 10.0% to show the generally low U_{num} distribution for the vast majority of the parameter domain. For the ellipse geometries, all uncertainties remain below 4.0%.

Using the normalized angle of reflection, ψ_{ref} , a continuously cyclic normalized clocking angle, ψ_{cyc} , is described by

$$\psi_{cyc} = \psi_{ref} \left(\frac{1}{\pi} \sin^{-1} \left(\sin \left(\pi \left(\frac{\psi}{\psi_{ref}} - 0.5 \right) \right) \right) + 0.5 \right). \quad (40)$$

Using this cyclic angle parameter, the k^* responses within the α ranges given in Table 1 can be characterized using a regression function, f_{reg} , of the form

$$\begin{aligned} f_{reg} = & \quad 1 + a_{c,1}\alpha + a_{c,2}\alpha^2 + \dots \\ & a_{g,1}((a_{c,3} - a_{c,1})\alpha + (a_{c,4} - a_{c,2})\alpha^2)\psi_{cyc}^2 + \dots, \\ & a_{g,2}((a_{c,1} - a_{c,3})\alpha + (a_{c,2} - a_{c,4})\alpha^2)\psi_{cyc}^3 \end{aligned} \quad (41)$$

with coefficients $a_{c,1}$ through $a_{c,4}$ and $a_{g,1}$ and $a_{g,2}$. Here, the four a_c coefficients are calibration parameters, tuned to minimize the difference between f_{reg} and the computational k^* values. The a_g coefficients are geometric constants unique to each pore geometry. Table 2 gives the coefficient values for f_{reg} for each of the pore geometry types. The cyclic angle formulation employed in Equation (41) allows the regression function to be applied to any clocking angle without restriction to the angle ranges shown in Table 1. Figure 36 illustrates the regression function applied to the three different geometries in the same manner as the computational results in Figure 34. The relative difference between the regression evaluations and the simulation results for triangle, square, and ellipse pore geometries are given in Figure 37a through Figure 37c, respectively. The relative difference, δk^* , is defined as

$$\delta k^* = (f_{reg} - k^*)/k^*. \quad (42)$$

The regressions agree with the numerical results to within less than 2.5% for the triangular pore, less than 5.0% for the square pore, and less than 6.0% for the elliptical pore.

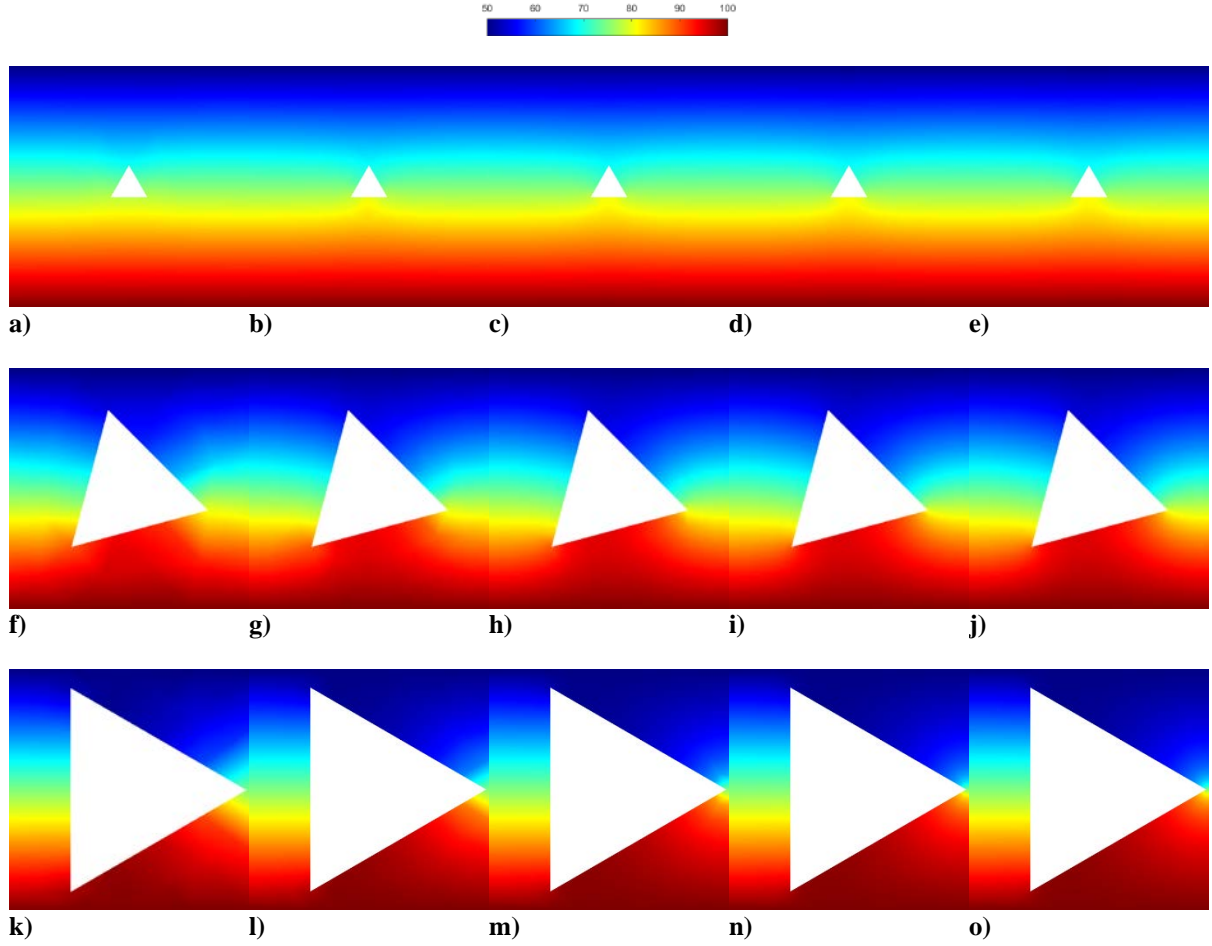
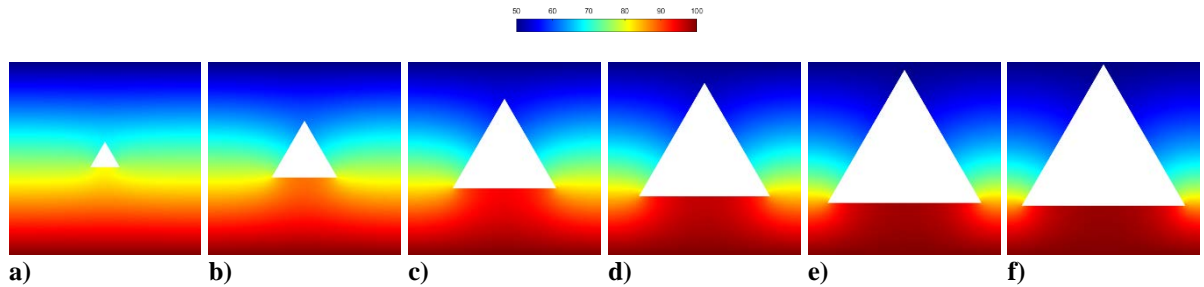


Figure 28. Temperature solution contours for triangle geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_a=0.0^\circ$, f) through j) $\alpha=15.0\%$ and $\theta_a=15.0^\circ$, and k) through o) $\alpha=31.0\%$ and $\theta_a=30.0^\circ$



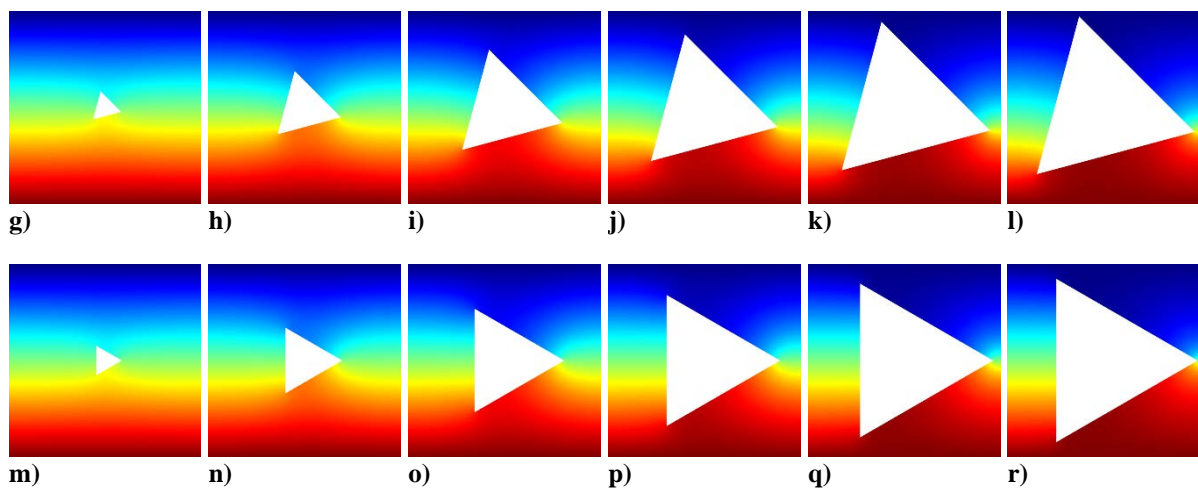


Figure 29. Temperature solution contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_a=0.0^\circ$, g) through l) $\theta_a=15.0^\circ$, and m) through r) $\theta_a=30.0^\circ$

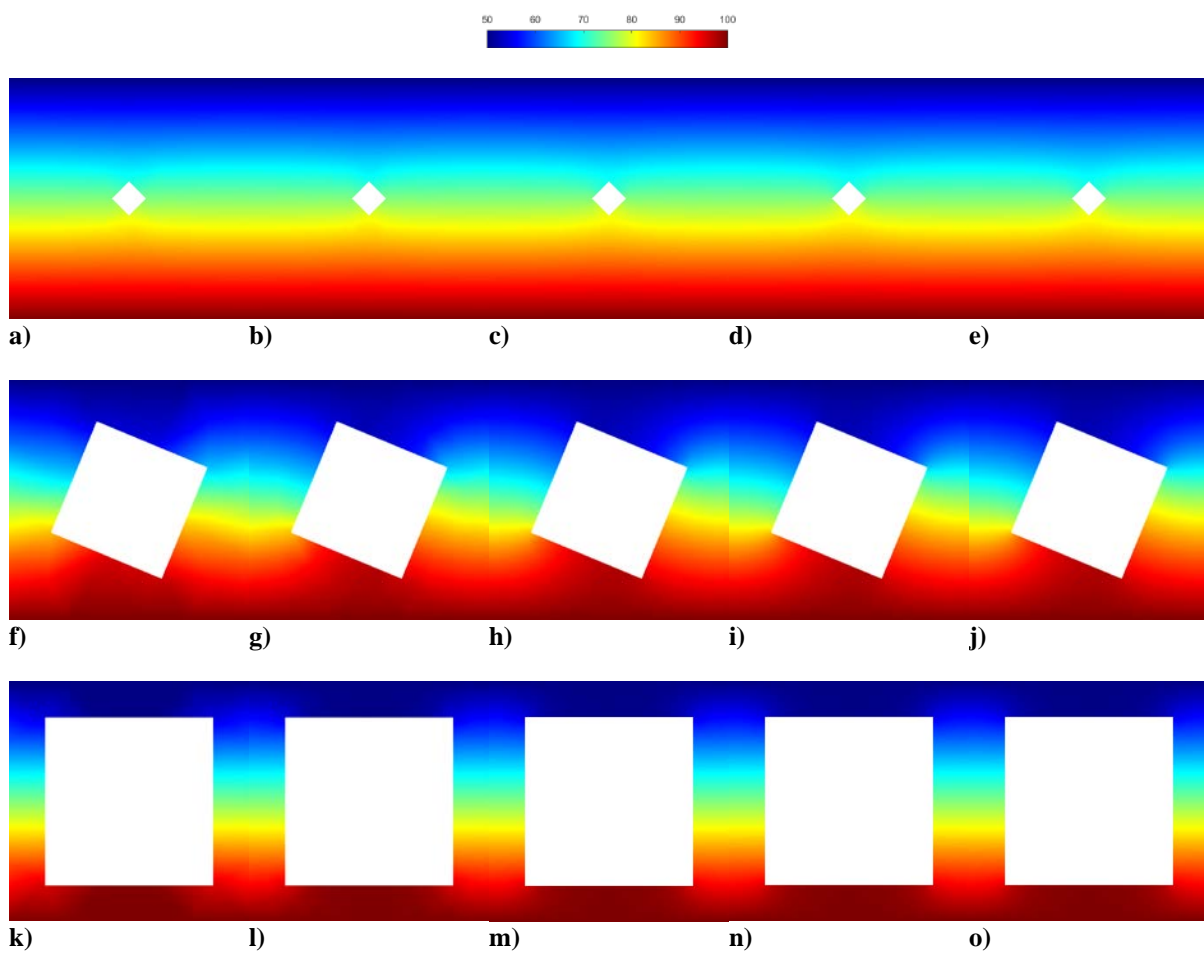


Figure 30. Temperature solution contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$

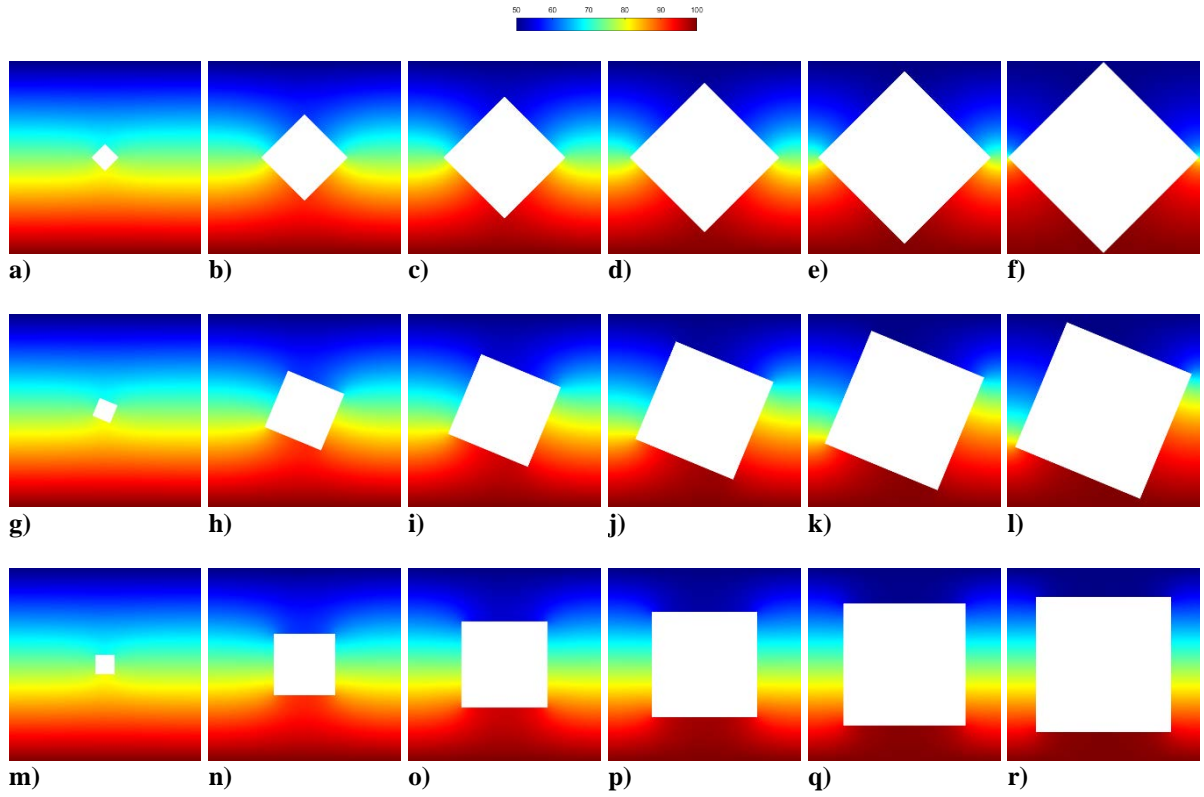
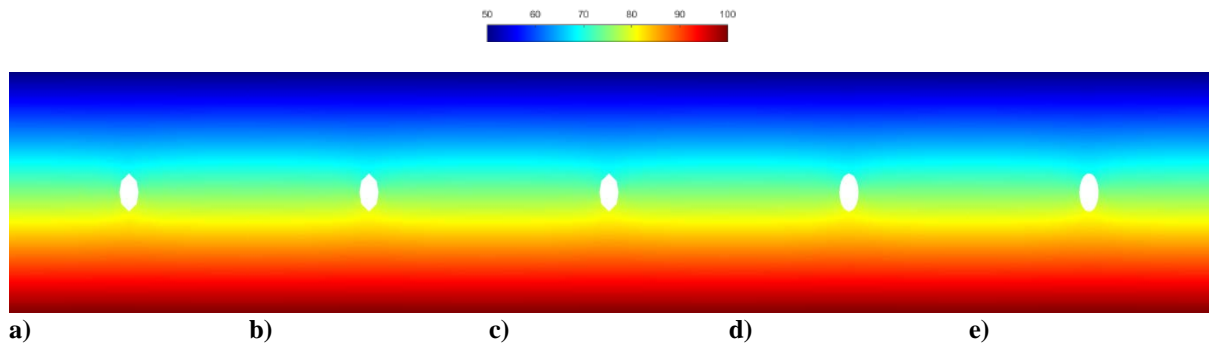


Figure 31. Temperature solution contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$



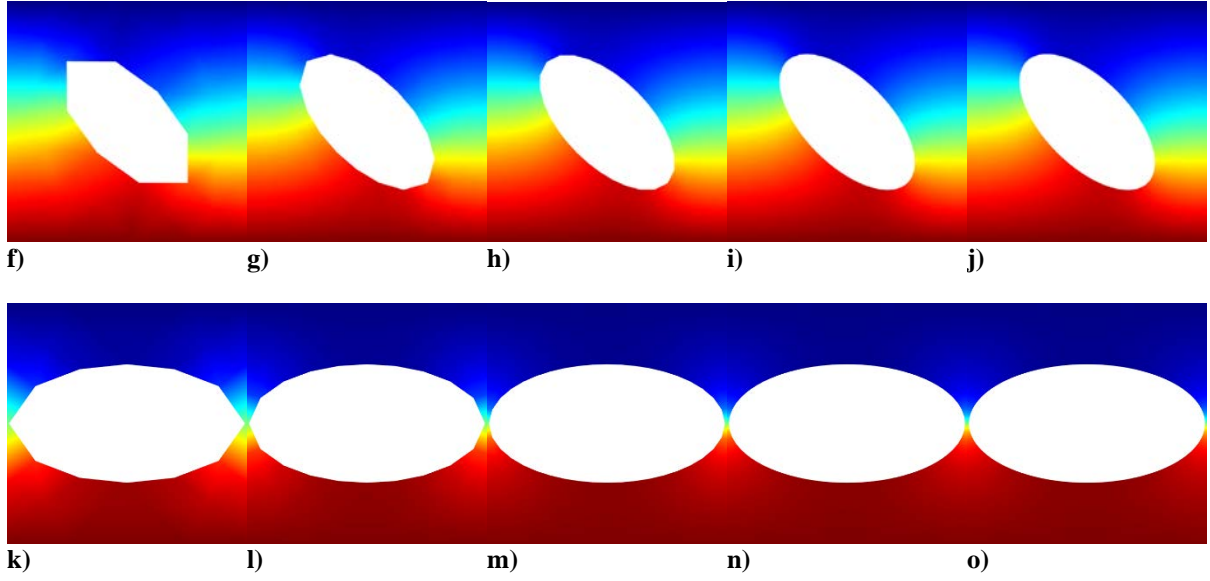


Figure 32. Temperature solution contours for ellipse geometries with mesh refinement for $H=5, 4, 3, 2,$ and 1 with a) through e) $\alpha=1.0\%$ and $\theta_\alpha=0.0^\circ$, f) through j) $\alpha=20.0\%$ and $\theta_\alpha=45.0^\circ$, and k) through o) $\alpha=38.0\%$ and $\theta_\alpha=90.0^\circ$

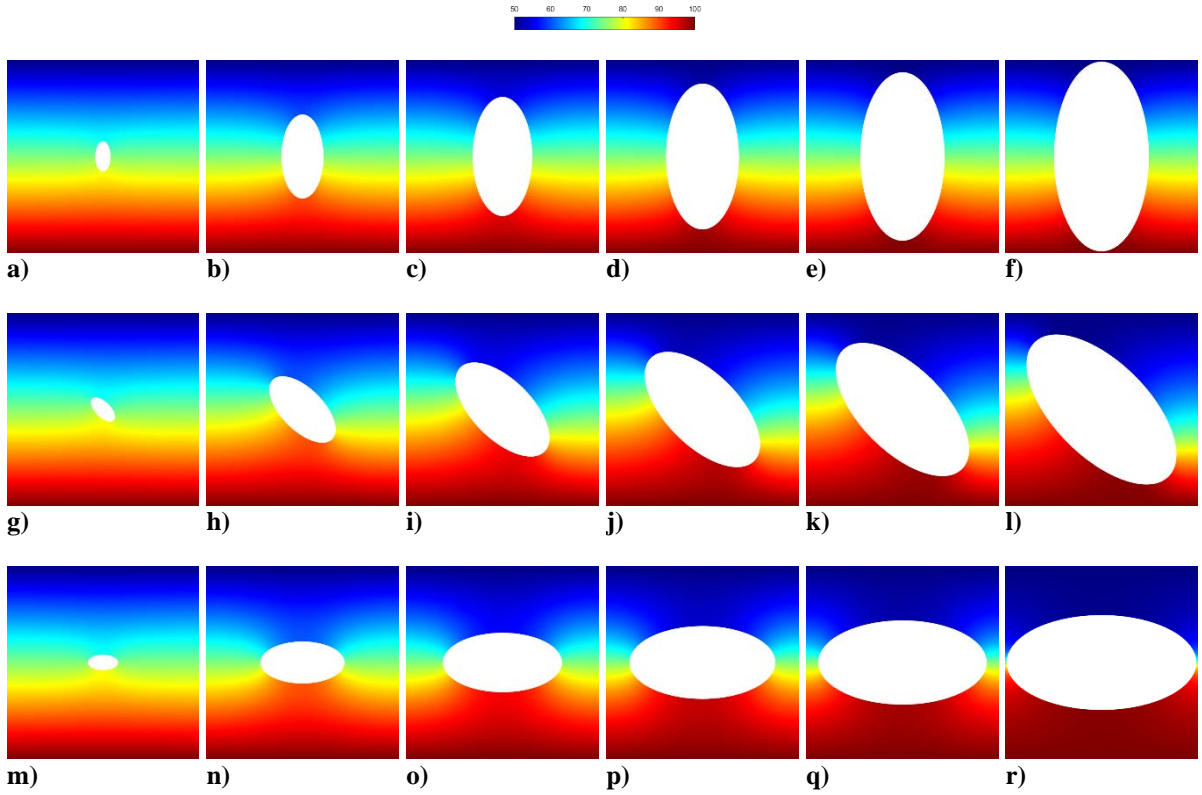


Figure 33. Temperature solution contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

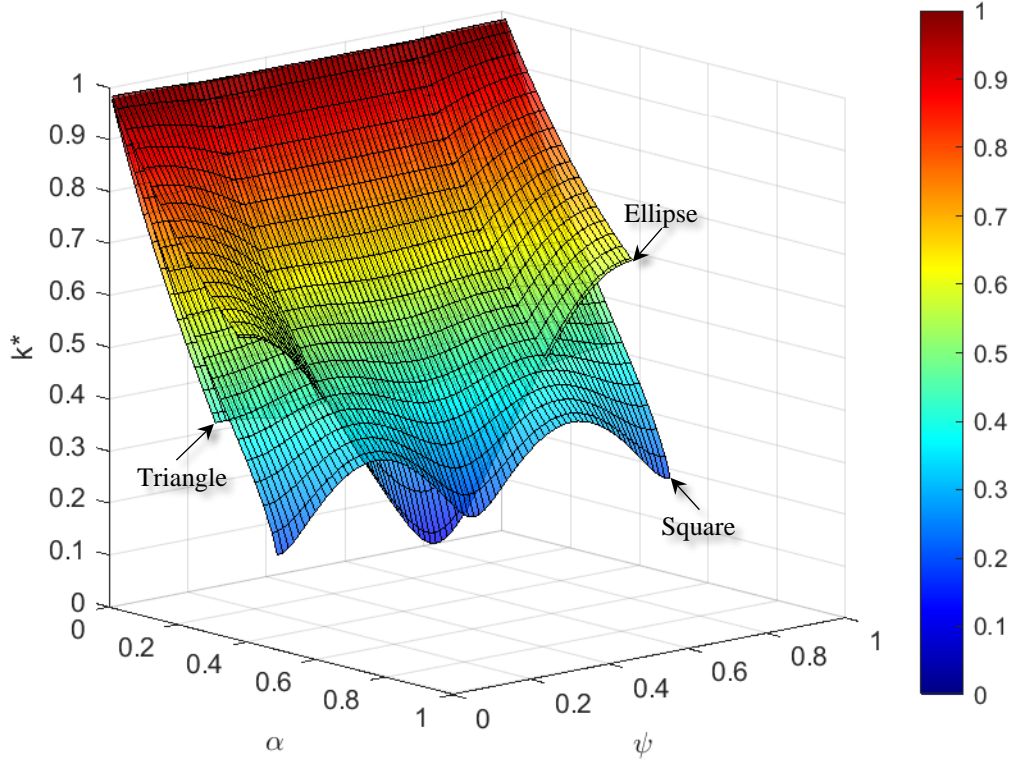
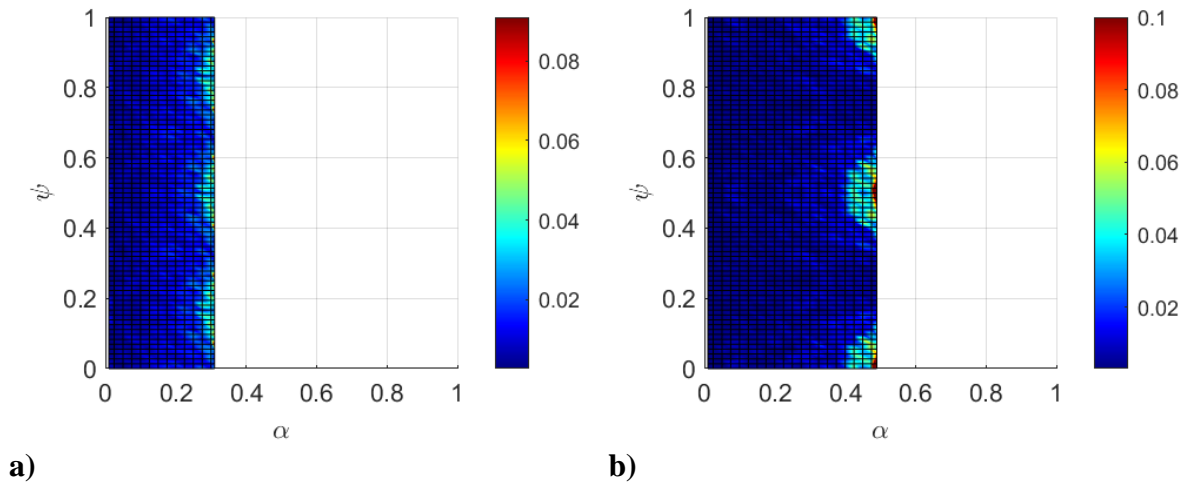
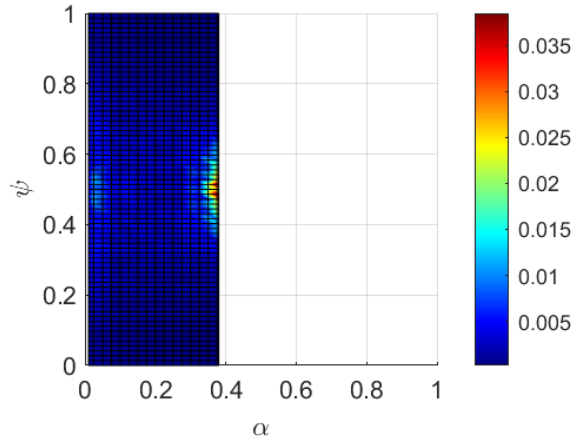


Figure 34. Computational k^* response as a function of porosity and clocking angle for triangle, square, and ellipse pore geometries





c)

Figure 35. U_{num} estimates for k^* as a function of porosity and clocking angle for a) triangle, b) square, and c) ellipse pore geometries

Table 2. Regression coefficients

	Triangle	Square	Ellipse
$a_{c,1}$	-2.305720	-1.619511	-1.521327
$a_{c,2}$	1.197482	-0.075605	1.482067
$a_{c,3}$	-2.305720	-2.086114	-2.413232
$a_{c,4}$	1.197482	1.518092	0.275217
$a_{g,1}$	108.0	48.0	12.0
$a_{g,2}$	432.0	128.0	16.0

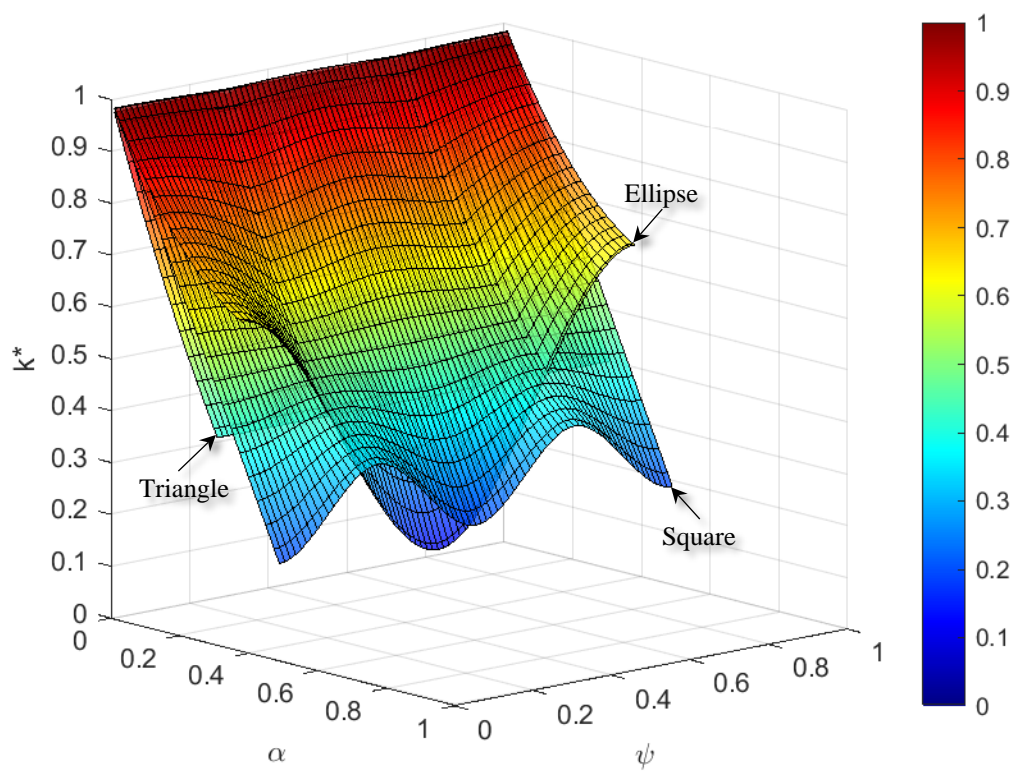
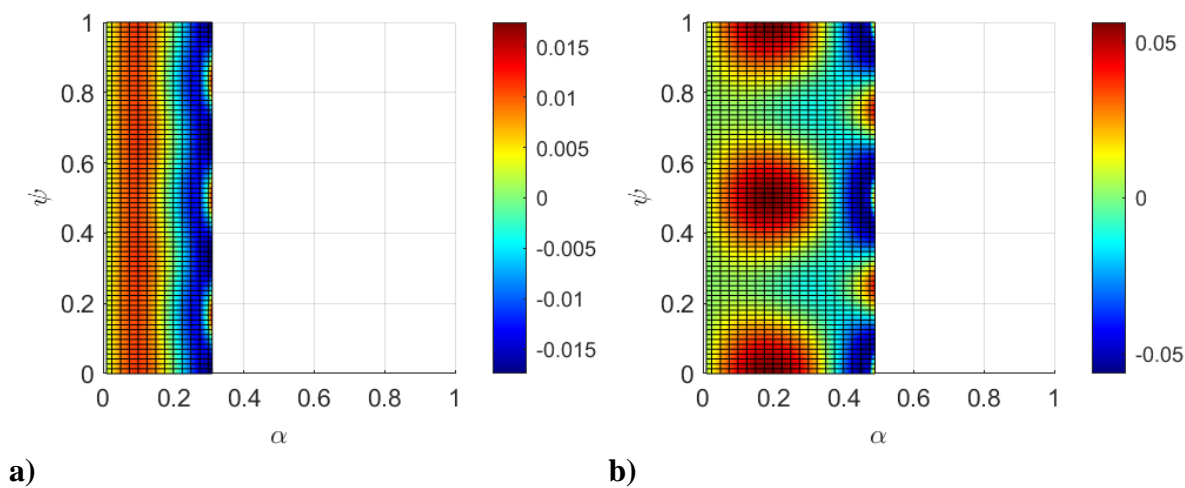
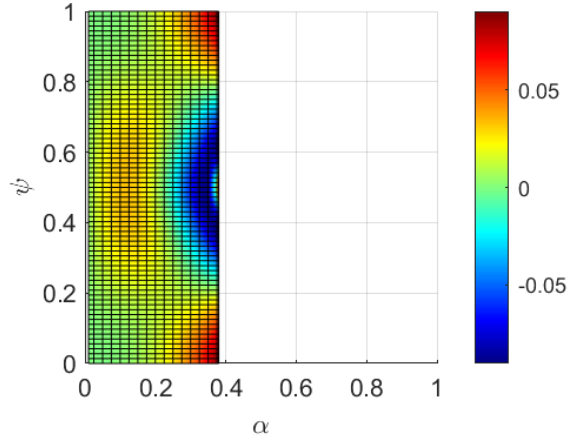


Figure 36. Regression k^* response as a function of porosity and clocking angle for triangle, square, and ellipse pore geometries





c)

Figure 37. Relative difference in regression k^* from computational k^* with respect to porosity and clocking angle for a) triangle, b) square, and c) ellipse pore geometries

The resultant k^* values as presented here are only valid for system configurations that agree with those used in the system models. That is to say that the effective thermal conductivity is valid in the direction of heat flow and when periodic boundary conditions are assumed on boundaries perpendicular to the heat flow and in assumed two-dimensional systems.

6 Conclusion and Future Work

The numerical analyses performed in this work have covered a range of two-dimensional heterogeneous, porous material structure geometries. This paper has presented a numerical approach to characterizing the effective thermal response of two-dimensional heterogeneous structures with respect to material porosity, pore geometry, and pore orientation. Numerical results have been presented for the effective thermal transport response of porous media with non-circular pore geometries, including triangle, square, and ellipse pore systems.

The method of manufactured solutions was used to perform code verification and showed that an approximately second order accurate numerical method was implemented. The global deviation grid convergence index was used to approximate numerical uncertainty on effective thermal conductivity using systematic mesh refinement. Numerical uncertainty, was estimated to be less than 6.0% for triangle and square geometries (with some exceptions) and less than 4.0% for ellipse geometries. A functional regression form was presented to map pore size and orientation to effective thermal conductivity for triangular, square, and elliptical pores to within less than 2.0%, 6.0%, and 10.0% difference, respectively, from the numerical results.

Future work should incorporate numerical uncertainty with the effective thermal conductivity correlation to give a prescribed level of overall uncertainty for the correlation. Further dimensionality could be added to the effective thermal conductivity correlation by investigating the effects internal angle ratios in the polygonal pores and the effects of aspect ratio in elliptical pores.

References

- 1 Gu, S., Lu, T. J., Hass, D. D., and Wadley, H. N. G., 2001, "Thermal conductivity of zirconia coatings with zig-zag pore microstructures," *Acta Materialia*, **49**(13), pp. 2539-2547.
- 2 Flores Renteria, A., Saruhan, B., Schulz, U., Raetzer-Scheibe, H.-J., Haug, J., and Wiedenmann, A., 2006, "Effect of morphology on thermal conductivity of EB-PVD PYSZ TBCs," *Surface and Coatings Technology*, **201**(6), pp. 2611-2620.
- 3 Zhu, W., Cai, X. N., Yang, L., Xia, J., Zhou, Y. C., and Pi, Z. P., 2019, "The evolution of pores in thermal barrier coatings under volcanic ash corrosion using X-ray computed tomography," *Surface and Coatings Technology*, **357**, pp. 372-378.
- 4 Irick, K. W. and Fathi, N., 2019, "Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis," *Proceedings of the ASME 2019 Verification and Validation Symposium*, V001T12A002.
- 5 Zhao, Z., Shang, H., Zou, G., Ren, H., Zhuang, W., Liu, L., and Zhou, Y. N., 2019, "A Predictive Model for Thermal Conductivity of Nano-Ag Sintered Interconnect for a SiC Die," *Journal of Electronic Materials*, **48**, pp. 2811-2825.

- 6 Dinesh, B. V. S. and Bhattacharya, A., 2019, "Effect of foam geometry on heat absorption characteristics of PCM-metal foam composite thermal energy storage systems," *International Journal of Heat and Mass Transfer*, **134**, pp. 866-883.
- 7 Rechard, R. P., Hadgu, T., Wang, Y., Sanchez, L. C., McDaniel, P., Skinner, C., and Fathi, N., 2017, "Technical Feasibility of Direct Disposal of Electrefiner Salt Waste," SAND2017-10554, Sandia National Laboratories (SNL-NM), Albuquerque, NM, USA.
- 8 Glass, D. E., Dirling, R., Croop, H., Fry, T. J., and Frank, G. J., 2006, "Materials Development for Hypersonic Flight Vehicles," *Proceedings of the 14th AIAA/AHI Space Planes and Hypersonic Systems and Technologies Conference*, AIAA 2006-8122.
- 9 Tjong, W. C., 2007, "High Temperature Materials for Hypersonic Flight Vehicles," *Journal of The University of New South Wales at the Australian Defence Force Academy*, **1**(1).
- 10 Kasen, S. D., 2013, "Thermal Management at Hypersonic Leading Edges," Dissertation, University of Virginia.
- 11 Finsterle, S., Muller, R. A., Baltzer, R., Payer, J., and Rector, J. W., 2019, "Numerical Evaluation of Thermal Effects from Nuclear Waste Disposed in Horizontal Drillholes," *International High-Level Radioactive Waste Management Conference*.
- 12 Woodside, W., and Messmer, J. H., 1961, "Thermal Conductivity of Porous Media. I. Unconsolidated Sands," *Journal of Applied Physics*, **32**(9), pp. 1688–1699.
- 13 Woodside, W., and Messmer, J. H., 1961, "Thermal Conductivity of Porous Media. II. Consolidated Rocks," *Journal of Applied Physics*, **32**(9), pp. 1699–1706.
- 14 Kiyohashi, H., Sasaki, S., and Masuda, H., 2003, "Effective Thermal Conductivity of Silica Sand as a Filling Material for Crevices," *High-Temperatures-High Pressures*, **35**, pp. 179-192.
- 15 Wallen, B. M., Smits, K. M., Sakaki, T., Howington, S. E., T.K.K., C. D., 2016, "Thermal Conductivity of Binary Sand Mixtures Evaluated through Full Water Content Range," *Soil Science Society of America Journal*, **80**, pp.592 -603.
- 16 Gobetz, Z., Rowen, A. Dickman, C., Meinert, K., and Martukanitz, R., 2016, "Utilization of Additive Manufacturing for Aerospace Exchangers," *Office of Naval Research, Final Report*.
- 17 Zhang, S., Lane, B., Whiting, J., and Chou, K., 2018, "An Investigation into Metallic Powder Thermal Conductivity in Laser Powder Bed Fusion Additive Manufacturing," *Proceedings of the 19th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference*, pp. 1796-1807.
- 18 Ibrahim, Y., Elkholy, A., Schofield, J. S., Melenka, G. W., and Kempers, R., 2020, "Effective Thermal Conductivity of 3D-printed Continuous Fiber Polymer Composites," *Advance Manufacturing: Polymer & Composites Science*, **6**(1), pp. 17-28.

- 19 Danes, F., Garnier, B., and Dupuis, T., 2003, "Predicting, Measuring, and Tailoring the Transverse Thermal Conductivity of Composites from Polymer Matrix and Metal Filler," *International Journal of Thermophysics*, **24**(3), pp. 727-734.
- 20 Zhao, J.-J., Duan, Y.-Y., Wang X.-D., 2012, "Experimental and analytical analyses of the thermal conductivities and high-temperature characteristics of silica aerogels based on microstructures," *Journal of Physics D: Applied Physics*, **46**, 015304.
- 21 Irick, K. W., 2017, "Effective Thermal Resistance Comparison of Aerogel and Multi-Layer Insulation as Radiative Barriers Using the Single-Sided Guarded Hot Plate Method," *Frontiers in Heat and Mass Transfer*, **8**(2).
- 22 Wu, S., Li, T., Tong, Z., Chao, J., Zhai, T., Xu, J., Yan, T., Wu, M., Xu, Z., Bao, H., Deng, T., and Wang, R., 2019, "High-Performance Thermally Conductive Phase Change Composites by Large-Size Oriented Graphite Sheets for Scalable Thermal Energy Harvesting." *Advanced Materials*, **31**(49), 1905099.
- 23 Pabst, W., Gregorová, Sedlářová, I., and Černý, M., 2011, "Preparation and characterization of porous alumina-zirconia composite ceramics," *Journal of the European Ceramic Society*, **31**(14), pp. 2721-2731.
- 24 Mirabolghasemi, A., Akbarzadeh, A. H., Rodrige, D., and Therriault, D., 2019, "Thermal conductivity of architected cellular metamaterials," *Acta Materialia*, **174**, pp. 61-80.
- 25 Zohuri, Bahman, and Nima Fathi. *Thermal-hydraulic analysis of nuclear reactors*. New York: Springer, 2015.
- 26 Kämpf, H., and G. Karsten. "Effects of different types of void volumes on the radial temperature distribution of fuel pins." *Nuclear Applications and Technology* 9, no. 3 (1970): 288-300.
- 27 Ibrahim, T. K. and Rahman, M. M., 2013, "Study on effective parameter of the triple-pressure reheat combined cycle performance," *Thermal Science*, **17**(2), pp. 497-508.
- 28 Fathi, N., McDaniel, P., Forsberg, C., and de Oliveira, C., 2018, "Power Cycle Assessment of Nuclear Systems, Providing Energy Storage for Low Carbon Grids," *Journal of Nuclear Engineering and Radiation Science*, **4**(2), 020911.
- 29 Fathi, N., McDaniel, P., Forsberg, C., and de Oliveira, C., 2016, "Nuclear Systems for a Low Carbon Electrical Grid," *In 2016 24th International Conference on Nuclear Engineering*, pp. V001T03A007-V001T03A007.
- 30 Irick, K. W. and Fathi, N., 2018, "Thermal Response of Open-Cell Porous Materials: A Numerical Study and Model Assessment," *Proceedings of the ASME 2018 Verification and Validation Symposium*, V001T03A002.
- 31 Rechar, R. P., Hadgu, T., Wang, Y., Sanchez, L. C., McDaniel, P., Skinner, C., and Fathi, N., 2017, "Technical Feasibility of Direct Disposal of Electrorefiner Salt Waste," *SAND2017-10554*, Sandia National Laboratories (SNL-NM), Albuquerque, NM, USA.
- 32 He., Y.-L. and Xie, T., 2015, "Advances of thermal conductivity models of nanoscale silica aerogel insulation material," *Applied Thermal Engineering*, **81**, pp. 28-50.

- 33 Geuzaine, C. and Remacle, J.-F., 2009, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering* **79**(11), pp. 1309-1331.
- 34 Oberkampf, W. L. and Roy, C. J., 2010, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge, UK.
- 35 Roache, P. J., 2009, *Fundamentals of Verification and Validation*, Hermosa Publishers, Socorro, NM.
- 36 Phillips, T. S. and Roy, C. J., 2016 "A New Extrapolation-Based Uncertainty Estimator for Computational Fluid Dynamics," *Journal of Verification, Validation and Uncertainty Quantification*, **1**(4), pp. 041006-1, 041006-13.
- 37 Celik, I. B., Ghia, U., Roache, P. J., Freitas, C. J., Coleman, H., and Raad, P. E., 2008, "Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications," *Journal of Fluids Engineering*, **130**(7), pp. 0780011-0780014.

CHAPTER 6: PORE GEOMETRY EFFECTS ON IRON CHROME ALUMINUM

(FeCrAl) ALLOY FOAM EFFECTIVE THERMAL CONDUCTIVITY⁶

Abstract

Iron chrome aluminum (FeCrAl) alloy is an extremely attractive material for use as coating and cladding for radioactive fuel in nuclear fuel rod assemblies. In both normal and abnormal thermal operating conditions for nuclear energy generation, potentially dangerous high-temperature reactions can occur between the protective fuel cladding and the coolant environment. The heterogeneous structure of porous FeCrAl can play a critical role in determining the thermal behavior of the material, especially under high heat conditions such as are present in nuclear reactors. The complexity of heat transfer by conduction can increase significantly in heterogeneous structures, including multi-component systems, complex microstructures and porous structures with dependent thermal conductivity values. This is especially true for porous materials, such as FeCrAl foam, where voids in the matrix material induce thermal resistance within the conductive system and divert heat flow within the material. Many high-impact fields rely on heterogeneous materials as a foundational aspect of their functionality and performance in thermal applications. The thermal response of FeCrAl in nuclear applications can be driven by pore size and geometry, both of which can be affected by manufacturing processes or environmental conditions. The study presented in this article is dedicated to studying the effects of pore geometry on the effective thermal conductivity, k_{eff} , of a whole FeCrAl alloy porous unit cell. To study a simple system porous unit cell, simulations

⁶ This chapter will be under review for publication in the Journal of Nuclear Engineering and Radiation Science (ASME) in late 2020. The content in this chapter was reproduced directly from the journal version, differing only in style formatting.

were performed on various porous material computational models. Multiple two-dimensional, steady-state models were constructed to observe the thermal response of porous FeCrAl foam to different pore geometries—including triangle, square, circle, and ellipse—centered in a unit cell porous medium. The finite element method (FEM) was implemented in Fortran—using unstructured triangular meshes—to simulate the thermal response of the systems. The method of manufactured solutions (MMS) was used to perform code verification and evaluate the code’s ability to solve the governing heat conduction equation. The results of the MMS study show that the implementation of the numerical FEM is approximately second order accurate. This article presents a study on the effects of pore size, shape, and rotational orientation on FeCrAl foam’s dimensionless effective thermal conductivity, k^* . A modern grid convergence index method was used to perform solution verification on the k^* results, estimating the numerical uncertainty in k^* to be less than 6% in almost all cases. Comparative thermal responses show the sensitivity of k^* with respect to pore shape and the variable sensitivity of response to pore orientation, where pore size is a major driver in the effective thermal response. A regression function is finally presented with less than 10% deviation from the computationally-determined results. Finally, a comparison of computational and regression results is made against empirically measured FeCrAl foam thermal conductivity data using calibration of a regression parameter.

Nomenclature

a	= coefficient
A	= area
α	= porosity
b	= base, side

c	= coefficient
δ	= difference
ε	= error
f	= function
FS	= factor of safety
Γ	= domain boundary
G	= conductance
\mathbf{G}	= conductance matrix
h	= characteristic mesh size
H	= mesh number
i	= index
j	= index
k	= thermal conductivity
\mathbf{K}	= thermal conductivity matrix
L	= cell length
m	= index
\mathbf{n}	= normal vector
N	= total quantity
\mathbf{N}	= vector shape function
p	= order of accuracy
\mathbf{p}	= direction vector
P	= heat source
\mathbf{P}	= heat load vector

\mathbf{q}	= heat flux vector
R	= radius
ρ	= residual
$\boldsymbol{\rho}$	= residual vector
S	= energy source
t	= index
T	= temperature
\mathbf{T}	= vertex temperature vector
θ	= pore clocking angle
U	= uncertainty
w	= weight function
\mathbf{w}	= vertex weight function vector
W	= cell width
x	= x-coordinate
y	= y-coordinate
ψ	= dimensionless clocking angle
ω	= relaxation coefficient
Ω	= domain

Subscripts

a	= semi-major
α	= pore
c	= calibration
cyc	= cyclic

C	= cold
eff	= effective
f	= formal
g	= geometric
H	= mesh number, hot
i	= index
j	= index
k^*	= dimensionless effective thermal conductivity
L^∞	= L-infinity norm
m	= index
MMS	= manufactured solution
n	= normal
num	= numerical
O	= observed
p	= semi-minor
ref	= reflection
reg	= regression
RMS	= root mean square
t	= triangle, transcendental

Superscripts

$*$	= dimensionless, average
$'$	= per unit length

1 Introduction

Heterogeneous materials are prevalent in many high-consequence and high-value systems, where accurate thermal design and engineering are fundamental to the system's success. Heterogeneity in a given material's composition can significantly impact the material's thermal response. Thus, it is advantageous to evaluate the impact of heterogeneous characteristics on a material's thermal behavior. Systems can be impacted by heterogeneity on a fine scale (e.g., at the material level) and on a larger scale (e.g., at the system level). Effects of heterogeneity are especially important in porous materials—e.g., nuclear fuel components—where pores in the bulk material matrix impede and divert heat flow within the material. Oftentimes, heterogeneity is caused by porous structures within a material's solid domain. This is especially true in energy systems where thermal barrier coatings [1-4], semi-conductors [5], and/or energy storage materials [6] are used.

The complexity of heat transfer by conduction can increase significantly in heterogeneous structures, including multi-component systems, complex microstructures and porous structures with dependent thermal conductivity values. Any discontinuity of material properties, especially thermal conductivity which is caused by cracks, gaps, voids, changing crystal structures, and nonuniform heat generation would need further investigation to understand the thermal behavior of the system of interest. Many advanced thermal systems rely on heterogeneous materials for reliable system performance.

Heterogeneous materials are pervasive across all forms of engineering fields and play a critical role in the performance of advanced technologies. In this work, the term “heterogeneous” is used to refer generally to any material with non-continuous material structure properties, where different regions of the material structure are comprised of different

material properties and/or behavior. Examples of commonplace heterogeneous materials and applications can include aerogel insulations, foams, 3D-printed structures, nuclear fuel assemblies, structural composites, thermal barrier coatings, and electronics dielectrics. Oftentimes heterogeneous materials—especially ceramics or ceramic-derivatives—are used in high-consequence and high-temperature applications where sparse empirical data is available. A combination of application criticality and lack of real-world material performance measurements in high-consequence or high-temperature applications necessitates the use of computational approaches to assess material behavior.

Many high-impact fields rely on heterogeneous materials as a foundational aspect of their functionality and performance in thermal applications. As an application example, many energy storage system concepts depend on multi-phase and/or multi-component materials [6,7]. Hypersonic flight vehicles experience extreme heat loads on leading edges and rely on composite and ceramic materials to prevent destruction of the vehicle [8-10]. In geosciences, evaluation of thermal behavior of heterogeneous—especially porous—earthen regions is important in assessing environmental responses to both natural and man-made conditions [11-15]. The additive manufacturing industry has a natural need for understanding thermal transport in porous materials—considering the non-homogenous and high-temperature processes used—where additive manufacturing can be used specifically for thermal applications [16-18]. The thermal performance and advantage of other advanced materials—such as aerogels and doped polymers—are driven by their heterogeneous characteristics and can be seen in marine, oil and gas, aerospace, energy, and thermal management industries [19-22]. The expansive set of industries and specific thermal applications where heterogeneous materials are of immediate consequence is immeasurable. Consequently, the pursuit of

measuring, analyzing, and predicting the thermal performance of such materials is in high demand.

The thermal response of porous materials can be driven by pore size and geometry, both of which can be affected by manufacturing process or environmental conditions [3,23,24]. Heat transfer in nuclear fuel elements is heavily dependent on these all of these phenomena. Figure 1 depicts an example nuclear fuel assembly, where bundles of fuel rods are assembled in a single package. Fuel rods contain stacks of fuel pellets, like the uranium dioxide (UO_2) sintered pellets shown in Figure 2. A variety of materials have been applied and proposed to be used as reactor fuels since decades ago. In thermal reactors, UO_2 , has shown satisfactory chemical and irradiation tolerance despite the disadvantages of low thermal conductivity and uranium atom density [25]. Oxide fuel is manufactured by sintering pressed powdered UO_2 and/or mixed oxides at high temperature values to produce ceramic pellets. The pellets of fuel are manufactured with the nominal values of 5 to 10% porosity to prevent pellet swelling from gaseous fission product species. Each fuel rod may be comprised of a composite material system, as illustrated in Figure 3, with the heat-generating fuel pellets located at the center of the rod. The fuel pellets are often enveloped in a cladding material, and the cladding may even be coated with a protective layer.

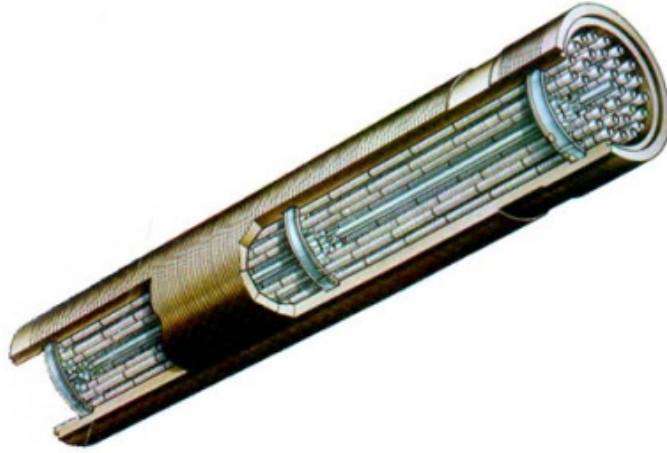


Figure 1. Example nuclear fuel assembly [26]



Figure 2. UO₂ sintered pellets [27]

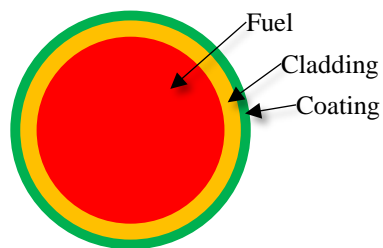


Figure 3. Notional fuel rod composition cross-sectional axial diagram

In nuclear reactors, the nuclear fuel assembly is suspended in a liquid coolant—primarily water—that is heated by the energy emanating from the fuel within the assembly. With high temperatures induced in the heated assembly and water, superheated steam is generated at the interface between the cladding and the water. The high-temperature steam and cladding have the potential to react, generating hydrogen gas, which can be problematic in driving up temperature and pressure [28]. This kind of problem motivates the desire for so-called accident tolerant fuels (ATFs). One solution to designing an ATF is to mitigate the oxidation reaction between the fuel cladding and the steam using an oxidation resistive coating. Iron chrome aluminum alloy (FeCrAl) is an attractive candidate for use as an ATF coating due to its improvement in oxidation resistance over the tradition zirconium cladding material in both normal and abnormal operation conditions [28,29]. FeCrAl can be used either as a cold sprayed coating over an underlying cladding layer or as a monolithic cladding material.

FeCrAl can be manifested as a solid continuous material or as a porous metal foam. The solid FeCrAl material exhibits thermal conductivity values between approximately 10 W/m-K and 30 W/m-K. Idaho National Laboratory—the premier center for nuclear energy development in the United States—uses BISON for finite element-based nuclear fuel performance analyses [30]. BISON has three built-in thermal conductivity models for FeCrAl, the trends of which are displayed in Figure 4, with properties coming from [31], [32], and [33]. However, when FeCrAl is found in its porous form, as shown in Figure 5, with voids comprising up to around 90% of the material volume, the effective material thermal conductivity can significantly decrease. Noting the possible thermal issues surrounding nuclear fuel cladding, reductions in thermal conductivity due to FeCrAl porosity may have critical impacts on operational thermal responses.

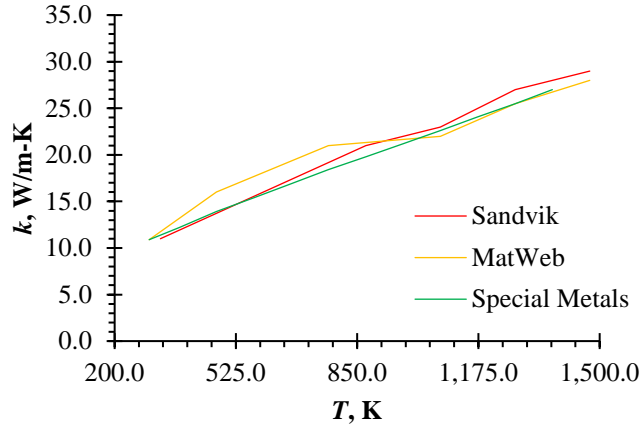


Figure 4. FeCrAl matrix thermal conductivity

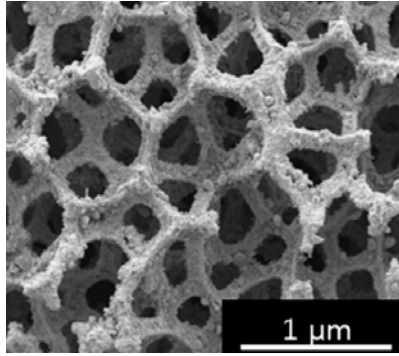


Figure 5. Scanning electron microscope image of FeCrAl foam [34]

The effective thermal conductivity, k_{eff} , of a porous medium can be calculated from the traditional linear correlation as a function of the void or pore fraction, α , and the bulk matrix material thermal conductivity, k_1 , as defined here [35]:

$$k_{eff} = (1 - \alpha)k_1. \quad (1)$$

While Equation (1) can be useful for some fundamental calculations to analyze simple geometries and composites, it does not provide reliable results for nonlinear geometries and functions, e.g., radial temperature distribution or further nonlinearity.

Understanding the effective thermal behavior of porous materials can be essential in engineering and design of critical next-generation thermal systems [36-40]. This paper is dedicated to investigating the effective thermal response of porous materials using numerical assessments to correlate thermal response with pore characteristics.

2 System Description

Porous material systems were modeled in this work using three different two-dimensional pore geometry types: triangle, square, and ellipse. The triangle pore was restricted to be equilateral, and the ellipse semi-major-to-semi-minor axis ratio was maintained at 2-to-1. As shown in Figure 6 through Figure 9, the centroid of each pore was centered in a square solid domain with thermal conductivity, k_I , and side dimensions W and L in the x -direction and y -direction, respectively. Here, $W=L=1$. The triangle pore is characterized by its base, b , with pore area, A_α , computed as

$$A_\alpha = \frac{\sqrt{3}}{4} b^2. \quad (2)$$

The square pore is characterized by its side length, b , with pore area computed as

$$A_\alpha = b^2. \quad (3)$$

The circle pore is characterized by its radius, R , with pore area computed as

$$A_\alpha = \pi R^2. \quad (4)$$

The ellipse pore is characterized by its semi-minor axis, R_p , and semi-major axis, R_a , with pore area computed as

$$A_\alpha = \pi R_p R_a, \quad (5)$$

where the ratio $R_a/R_p=2$ was maintained. The systems—except for the circle pore system—were analyzed at different clocking angles, θ_α , about the pore centroid with respect to the positive y-direction vector. Boundary conditions on the domains were enforced such that the positive y face of the domain was fixed at a cold temperature, T_C , the negative y face of the domain was fixed at a hot temperature, T_H , and all other solid domain boundaries were considered adiabatic. The enforced temperature gradient across the domain induces an average heat flow per unit length, q' , across the system Dirichlet boundaries.

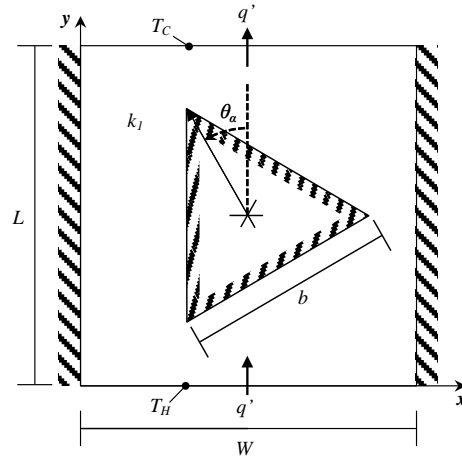


Figure 6. Unit cell triangular nano-porous structure

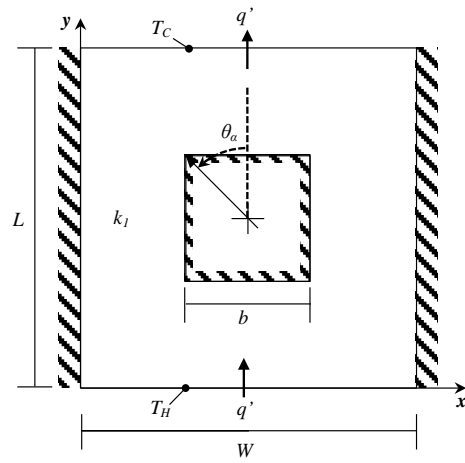


Figure 7. Unit cell square nano-porous structure

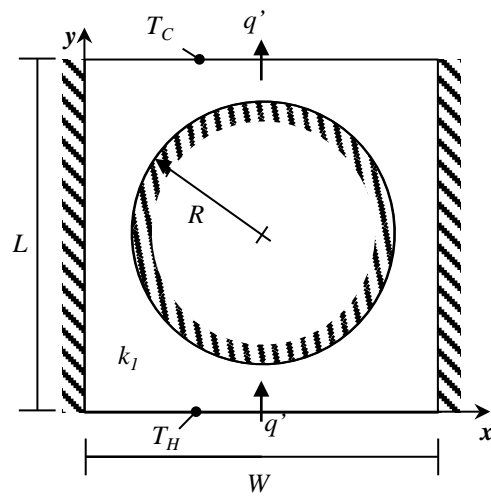


Figure 8. Unit cell circular nano-porous structure

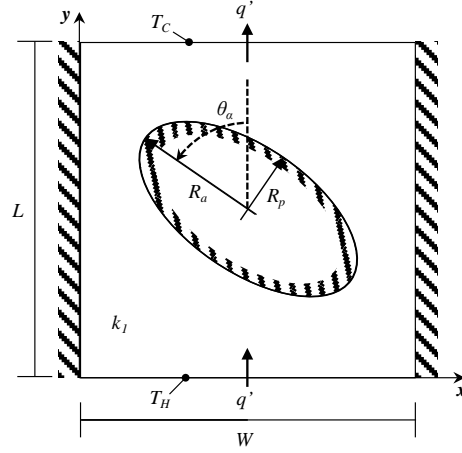


Figure 9. Unit cell elliptical nano-porous structure

In this study, the effects of both θ_α and porosity, α , were investigated, where porosity is defined as

$$\alpha = \frac{A_\alpha}{WL}. \quad (6)$$

With changing θ_α and α , the q' magnitude is affected, and an effective thermal conductivity, k_{eff} , of the system can be computed by

$$k_{eff} = \frac{q'L}{W(T_H - T_C)}, \quad (7)$$

where linear conductive heat transfer across the system implies an average q' at the boundaries due to a fixed temperature difference across the system boundaries. A dimensionless effective thermal conductivity, k^* , represents the normalization of k_{eff} by the solid domain thermal conductivity, where

$$k^* = \frac{k_{eff}}{k_1}. \quad (8)$$

The triangle, square, and ellipse pore geometries were evaluated at 2.5% increments in α with an additional minimum and maximum and at 2.5 ° increments in θ_α , with configurations summarized in Table 1. Note that the analyses for the circular pore geometries were originally performed as part of a separate study from the non-circular pore geometries, thus the analyzed porosity values for the circular systems do not match those of the non-circular systems. Likewise, since the circle pore systems are infinitely symmetric with respect to clocking angle, no analysis of clocking angle sensitivity is required or possible. The upper bounds of θ_α represent the angle at which rotational symmetry—or reflection—occurs for the system geometry, θ_{ref} . Symmetry also occurs at $\theta_\alpha=0^\circ$.

Table 1. Geometry configurations

Geometry	α (%)	θ_α (°)
Triangle	1.0, [2.5,30.0], 31.0	[0,30]
Square	1.0, [2.5,47.5], 49.0	[0,45]
Circle	4.9, 7.7, 11.0, 15.0, 19.6, 24.9, 30.7, 37.1, 44.2, 51.8, 60.1	---
Ellipse	1.0, [2.5,37.5], 38.0	[0,90]

3 Discretization

A finite element (FE) approach was used to determine the resultant k^* values for the different system configurations described above. The governing steady-state, two-dimensional heat equation, given to be

$$k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + S = 0, \quad (9)$$

was solved across the solid domain, where k is thermal conductivity, T is the temperature distribution, and S is the volumetric source function. Given in the following subsections are

descriptions of the domain and partial differential equation (PDE) discretization to facilitate the FE method.

3.1 Domain

The porous system domains were discretized using unstructured triangular meshes. Figure 10 notionally illustrates the triangle element structure formed by three vertices in a mesh, where t is the triangle, $(x_{t,i}, y_{t,i})$, are the x and y coordinates of the i^{th} vertex in element t , $T_{t,i}$, is the temperature at vertex i in triangle t , and A_t is the area of triangle t .

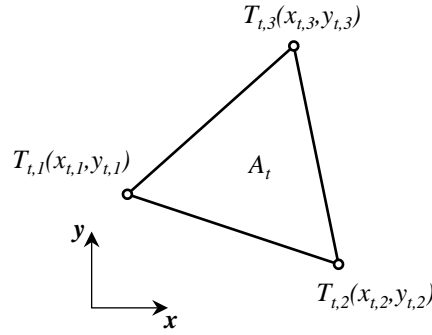


Figure 10. Notional triangle element

Systematic mesh refinement facilitated the verification methods described in subsequent sections of this report, where the mesh number, H , is used to describe the mesh refinement level of the domain, with $H=1$ and $H=5$ being the finest and coarsest meshes of a given domain, respectively. The characteristic mesh size, h_H , of mesh H , is computed as

$$h_H = \sqrt{\frac{1}{N_{H,t}} \sum_{t=1}^{N_{H,t}} A_t}, \quad (10)$$

where $N_{H,t}$ is the total number of triangles in the H^{th} mesh. Characteristic mesh sizes for meshes $H=5$ through $H=1$ used in this study were approximately 0.12, 0.07, 0.04, 0.02, and 0.01.

Figure 11 shows an example mesh refinement used in this study for an example triangle pore configuration with α of 17.5% and θ_α of 17.5°. Figure 12 shows example fine meshes for a variety of α and θ_α combinations for triangle pores. Likewise, Figure 13 shows mesh refinement for an example square pore configuration with α of 17.5% and θ_α of 22.5°, and Figure 14 shows multiple fine meshes for square pore geometries. Figure 15 shows an example mesh refinement for a circle pore with α of 30.7%, and Figure 16 shows fine mesh circle pore geometries for various α values. Lastly, Figure 17 gives an example mesh refinement for an ellipse pore with α of 17.5% and θ_α of 45.0°, Figure 18 shows fine mesh ellipse pore geometries for various α and θ_α combinations.

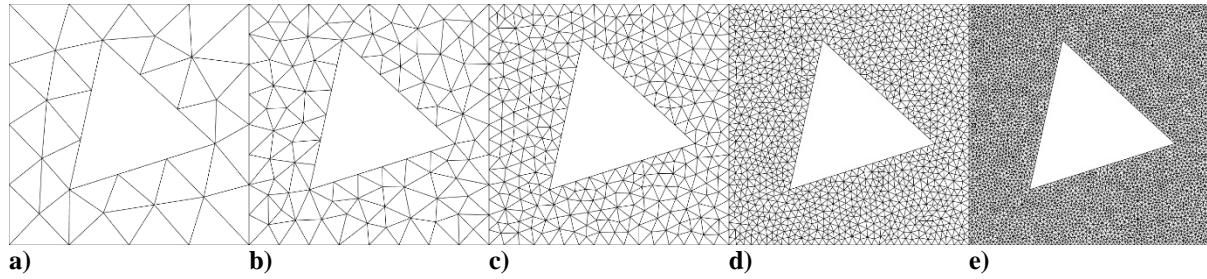
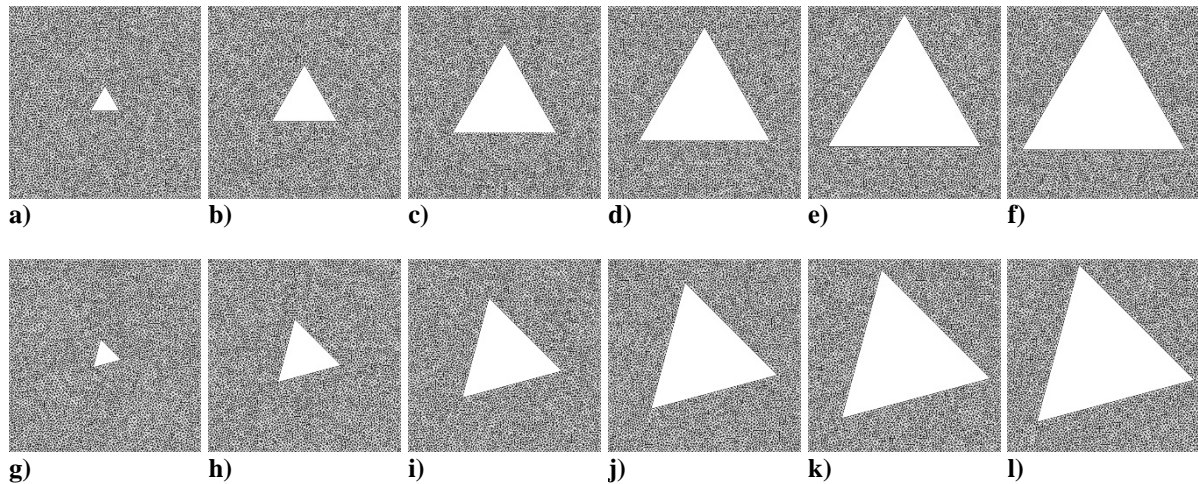


Figure 11. Domain mesh for triangle geometry at 17.5° clocking angle and 17.5% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$



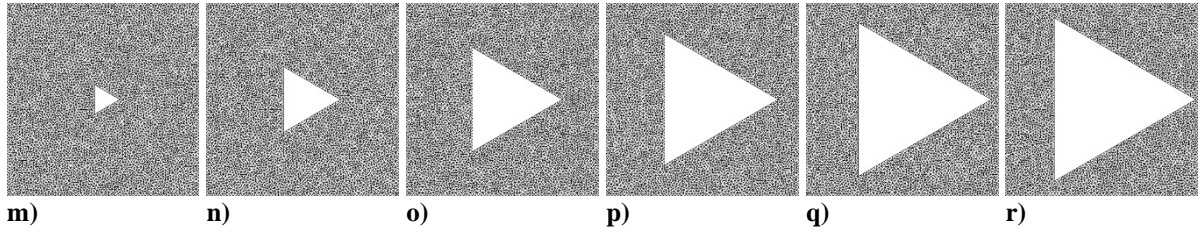


Figure 12. Finest mesh for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_a=0.0^\circ$, g) through l) $\theta_a=15.0^\circ$, and m) through r) $\theta_a=30.0^\circ$

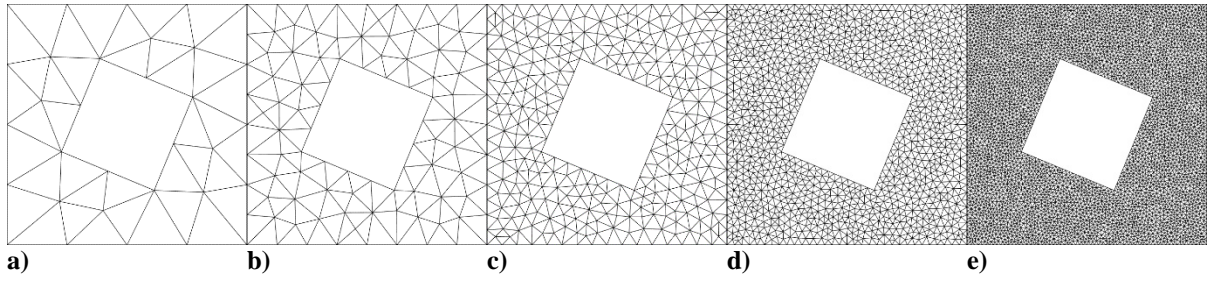
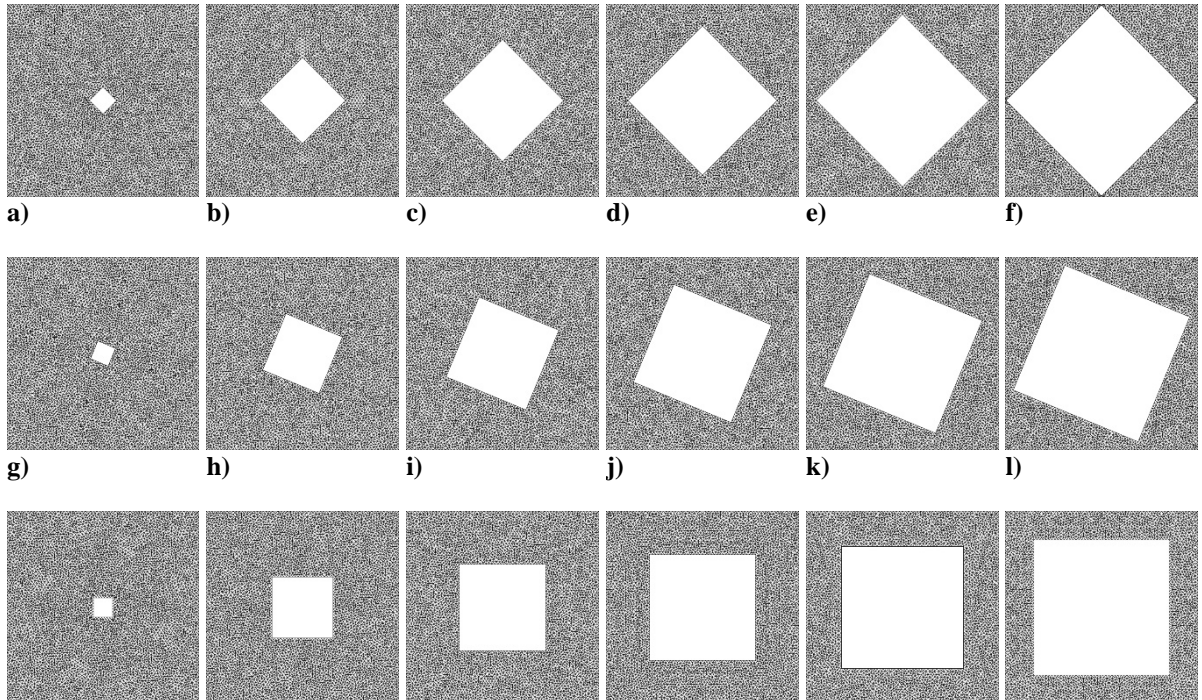


Figure 13. Domain mesh for square geometry at 22.5° clocking angle and 17.5% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$



m) n) o) p) q) r)

Figure 14. Finest mesh for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$

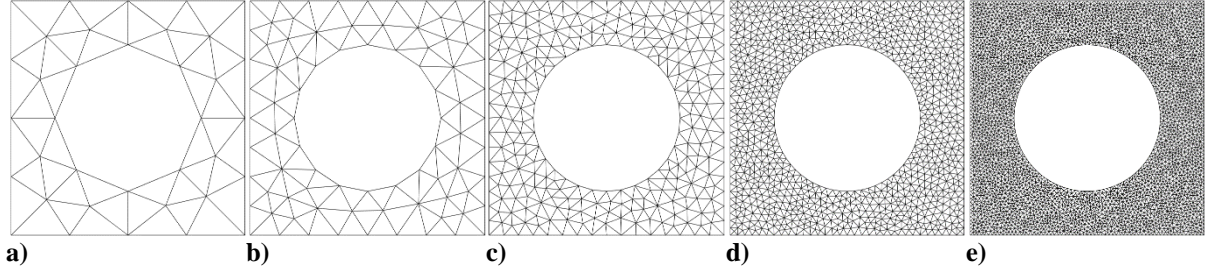


Figure 15. Domain mesh for circle geometry at 30.7% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

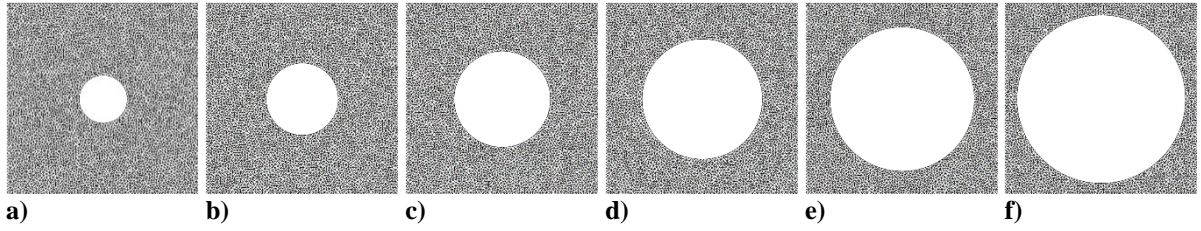


Figure 16. Finest mesh for circle geometries with a) $\alpha=4.9\%$, b) $\alpha=11.0\%$, c) $\alpha=19.6\%$, d) $\alpha=30.7\%$, e) $\alpha=44.2\%$, and f) $\alpha=60.1\%$

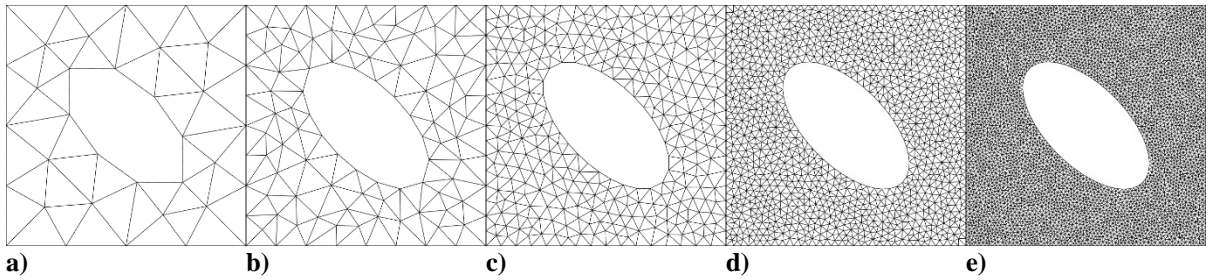


Figure 17. Domain mesh for ellipse geometry at 45.0° clocking angle and 17.5% porosity with a) $H=5$, b) $H=4$, c) $H=3$, d) $H=2$, and e) $H=1$

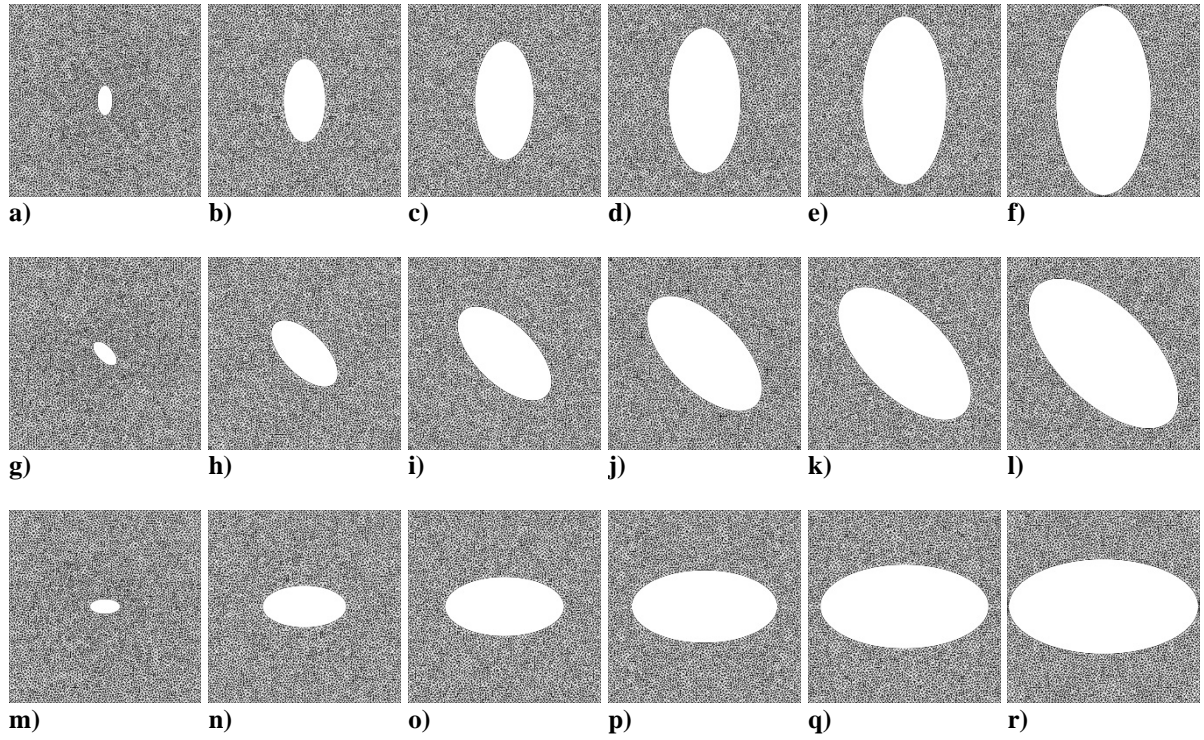


Figure 18. Finest mesh for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

3.2 Partial Differential Equation

A second order accurate Galerkin FE approach was used to solve the governing heat equation PDE over the solid computational domain. Equation (9) can be solved over each finite element, individually, using a discretized PDE matrix equation. The thermal conductivities and temperature gradients from Equation (9) can be cast in matrix form as

$$\mathbf{K} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \quad \text{and} \quad \nabla T = \begin{Bmatrix} \partial T / \partial x \\ \partial T / \partial y \end{Bmatrix}, \quad (11)$$

where \mathbf{K} is the thermal conductivity tensor. For the system at hand, isotropic thermal conductivity is used for both the matrix and filler materials, such that k_x and k_y are the same.

From Fourier's law, it follows that the heat flux vector, \mathbf{q} , is comprised of x -directional and y -directional heat fluxes, q_x and q_y , respectively, and looks like

$$\mathbf{q} = \begin{Bmatrix} q_x \\ q_y \end{Bmatrix} = -\mathbf{K}\nabla T. \quad (12)$$

At the boundary, Γ , of Ω that has a surface normal vector, \mathbf{n} , the normal heat flux leaving Ω , q_n , is described by

$$q_n = \mathbf{q}^T \cdot \mathbf{n}. \quad (13)$$

The heat equation of Equation (9) can then be recast as

$$S - \text{div}(\mathbf{q}) = S + \text{div}(\mathbf{K}\nabla T). \quad (14)$$

This work uses the Galerkin FE approach to solve the governing PDE which requires the use of some weighting function, w , over the PDE. Multiplying Equation (14) by w and integrating over Ω results in the following

$$\int_{\Omega} (\nabla w) \mathbf{K} \nabla T d\Omega = \int_{\Omega} w S d\Omega - \int_{\Gamma} w q_n d\Gamma. \quad (15)$$

For each triangle in the finite element domain, Equation (15) is true, and the Galerkin method prescribes that both w and T be interpolated across the finite element domain using the same row vector polynomial shape function, or interpolation function, \mathbf{N} . If a triangle element has weighting function and temperature values defined at each of its three vertices as $w_{t,i}$ and $T_{t,i}$, respectively, for $i=1,2,3$ corresponding to each of the i^{th} vertices of the t^{th} triangle, then let

$$\mathbf{T} = \begin{Bmatrix} T_{t,1} \\ T_{t,2} \\ T_{t,3} \end{Bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{Bmatrix} w_{t,1} \\ w_{t,2} \\ w_{t,3} \end{Bmatrix}, \quad (16)$$

for a given triangle. Thus, the field variable and weight function within the bounds of the finite element are described similarly as

$$T(x, y) = \mathbf{N}\mathbf{T} \quad \text{and} \quad w(x, y) = \mathbf{N}\mathbf{w}, \quad (17)$$

using the row vector interpolation function, \mathbf{N} . By using matrix transpose rules, substituting the definitions of Equation (17) into Equation (15), and eliminating common constant terms, the discretized form of the governing PDE solved over each linear triangle element in the discretized domain is

$$\int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \mathbf{T} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma. \quad (18)$$

In the discretized PDE for this work, \mathbf{N} is the linear interpolation shape function row vector for elemental field variables.

Portions of Equation (18) can be further condensed and defined as the conductance matrix, \mathbf{G} , and the heat load vector \mathbf{P} , where

$$\mathbf{G} = \int_{\Omega} (\nabla \mathbf{N})^T \mathbf{K} \nabla \mathbf{N} d\Omega \quad (19)$$

and

$$\mathbf{P} = \int_{\Omega} \mathbf{N}^T S d\Omega - \int_{\Gamma} \mathbf{N}^T \mathbf{q}_n d\Gamma \quad (20)$$

such that

$$GT = P. \quad (21)$$

4 Solution Evaluation

Open-source Gmsh software [41] was used to discretize the computational domain into unstructured triangle meshes, and personal Fortran code was written to solve the two-dimensional heat equation using the Galerkin FE method. A conjugate gradient iterative solver scheme was implemented to update solutions to the heat equation on the triangle, square, and ellipse pore geometries until the root mean square (RMS) of the energy equation residuals across the domain fell below 10^{-8} . A basic method—akin to that described in [42]—was used to update solutions for the circle pore systems. To provide credibility evidence for the summary results obtained in this study, code and solution verification were performed. The following subsections briefly describe the solution update methods and both verification processes used.

4.1 Solution

In the FE method, the imbalance of Equation (21) is considered the residual energy balance at each vertex, ρ_i , where ρ is the vector of vertex residuals, and

$$\rho = P - GT. \quad (22)$$

The implementation of the numerical method employed a conjugate gradient method to update the solution of Equation (21), driving energy residuals to zero. The iterative conjugate gradient scheme initializes (iteration $i=0$) the direction vector, p_0 , as

$$p_0 = \rho_0. \quad (23)$$

On the i^{th} iteration, starting at $i=0$, coefficient, $c_{1,i}$ is

$$c_{1,i} = (\boldsymbol{\rho}_i^T \boldsymbol{\rho}_i) / (\mathbf{p}_i^T \mathbf{G} \mathbf{p}_i), \quad (24)$$

the temperature solution is updated by

$$\mathbf{T}_{i+1} = \mathbf{T}_i + c_{1,i} \mathbf{p}_i, \quad (25)$$

and the residuals are updated as

$$\boldsymbol{\rho}_{i+1} = \boldsymbol{\rho}_i - c_{1,i} \mathbf{G} \mathbf{p}_i. \quad (26)$$

For further iteration, a coefficient, $c_{2,i}$, is defined as

$$c_{2,i} = (\boldsymbol{\rho}_{i+1}^T \boldsymbol{\rho}_{i+1}) / (\boldsymbol{\rho}_i^T \boldsymbol{\rho}_i), \quad (27)$$

and the direction vector for the next iteration is updated to be

$$\mathbf{p}_{i+1} = \boldsymbol{\rho}_{i+1} + c_{2,i} \mathbf{p}_i. \quad (28)$$

The relaxation method used for the circle pore systems uses a relaxation coefficient, ω , to modulate the rate at which the temperature solution is updated between iterations. The temperature of vertex j at timestep i , $T_{j,i}$, is updated to the next timestep by

$$T_{j,i+1} = \omega (P_j - \sum_{m=1}^{N_v} G_{j,m} T_{m,i}) / G_{j,j} + T_{j,i}, \quad (29)$$

where P_j is the heat source term of vertex i and $G_{j,m}$ is the conductance between vertices j and m . The processes terminate when sufficiently small residual conditions are met. In this work, the RMS of residuals, ρ_{RMS} , was used as termination criteria for the triangle, square, and ellipse pore geometry systems, where

$$\rho_{RMS} = \sqrt{\sum_{i=1}^{N_v} (\rho_i^2) / N_v}, \quad (30)$$

and a ρ_{RMS} value of 1.0×10^{-8} was used as the exit value. Although the RMS was used as exit criteria for the three non-circular geometries in this study, the L_∞ norm of residuals, $\rho_{L\infty}$, was computed and retained, where

$$\rho_{L\infty} = \max_{1 \leq i \leq N_v} |\rho_i|. \quad (31)$$

For the circle pore geometry systems, $\rho_{L\infty}$ was used as the exit metric with 1.0×10^{-9} being the cutoff value.

4.2 Code Verification

The method of manufactured solutions (MMS) was used on each porous system to show convergence of the observed order of accuracy of the numerical methods used for this work [43,44]. The order of accuracy of a finite element computational method describes the rate of change in numerical solution error with respect to change in finite element size. For example, a second order accurate model is a model whose solution error is quartered when the mesh elements are halved. In most meaningful engineering systems, an exact solution for the governing PDEs over the system does not exist, thus a surrogate exact solution is constructed. MMS allows one to define a solution to the governing PDE, T_{MMS} , for the system and determine what boundary conditions and source terms satisfy that solution. By then applying the determined boundary conditions and source terms to the model, the simulations should return solutions approximating T_{MMS} (the “true” solution). In this way, one need not find an analytical solution to a given problem in order to determine the accuracy of the simulation results.

For the triangle, square and ellipse pore geometries in this study, T_{MMS} was selected to be

$$T_{MMS}(x, y) = \cos(2\pi x) \sin(\pi y + 0.75), \quad (32)$$

where its first and second spatial derivatives are

$$\begin{aligned} \frac{\partial T}{\partial x} &= -2\pi \sin(2\pi x) \sin(\pi y + 0.75) \\ \frac{\partial T}{\partial y} &= \pi \cos(2\pi x) \cos(\pi y + 0.75) \\ \frac{\partial^2 T}{\partial x^2} &= -4\pi^2 \cos(2\pi x) \sin(\pi y + 0.75) \\ \frac{\partial^2 T}{\partial y^2} &= -\pi^2 \cos(2\pi x) \sin(\pi y + 0.75) \end{aligned} \quad (33)$$

For the circle pore geometries in this study, T_{MMS} was selected to be

$$T_{MMS}(x, y) = \cos(\pi x) \sin(\pi y + 0.75), \quad (34)$$

where its first and second spatial derivatives are

$$\begin{aligned} \frac{\partial T}{\partial x} &= -\pi \sin(\pi x) \sin(\pi y + 0.75) \\ \frac{\partial T}{\partial y} &= \pi \cos(\pi x) \cos(\pi y + 0.75) \\ \frac{\partial^2 T}{\partial x^2} &= -\pi^2 \cos(\pi x) \sin(\pi y + 0.75) \\ \frac{\partial^2 T}{\partial y^2} &= -\pi^2 \cos(\pi x) \sin(\pi y + 0.75) \end{aligned} \quad (35)$$

By using the MMS derivatives from Equation (33) in Equation (9), the resulting MMS volumetric source distribution, S_{MMS} , is evaluated as

$$S_{MMS}(x, y) = 5\pi^2 k \cos(2\pi x) \sin(\pi y + 0.75). \quad (36)$$

Likewise, by using the MMS derivatives from Equation (35) in Equation (9), S_{MMS} for the circle geometries is evaluated as

$$S_{MMS}(x, y) = 2\pi^2 k \cos(\pi x) \sin(\pi y + 0.75). \quad (37)$$

To solve the MMS problem, all boundary conditions on the system are replaced with values that correspond to the T_{MMS} and its derivatives, consistent with the boundary condition types in the true problem of interest. Finally, S_{MMS} is applied across the entire computational domain. The system is then solved numerically, where the computational solution is expected to approach the prescribed T_{MMS} with mesh refinement.

With k set to 100.0 for this study for the triangle, square, and ellipse systems and 310.0 for the circle systems, Figure 19 and Figure 20 show the selected MMS temperature field and applied MMS source field contours, respectively, on various triangle geometry α and θ_α system configurations. Figure 21 and Figure 22 show the selected MMS temperature field and applied MMS source field contours, respectively, on various square geometry α and θ_α system configurations. Figure 23 and Figure 24 show the selected MMS temperature field and applied MMS source field contours, respectively, on various circle geometry α system configurations. Figure 25 and Figure 26 show the selected MMS temperature field and applied MMS source field contours, respectively, on various ellipse geometry α and θ_α system configurations. Note that in the S_{MMS} contour plots here, that the plotted source term is merely the opposite of S_{MMS} defined in Equation (36) and Equation (37) to match application formulations in the FE code.

The observed order of accuracy, $p_{O,H}$, for mesh H of each system configuration was expected to converge to the formal order of accuracy, p_f , of 2 with mesh refinement, as predicted by the second order accurate Galerkin FE method used. Nodal temperature solutions across the mesh are compared against the prescribed T_{MMS} distribution for MMS convergence evaluation, where the RMS error for all nodes and the L_∞ norm of errors was tracked.

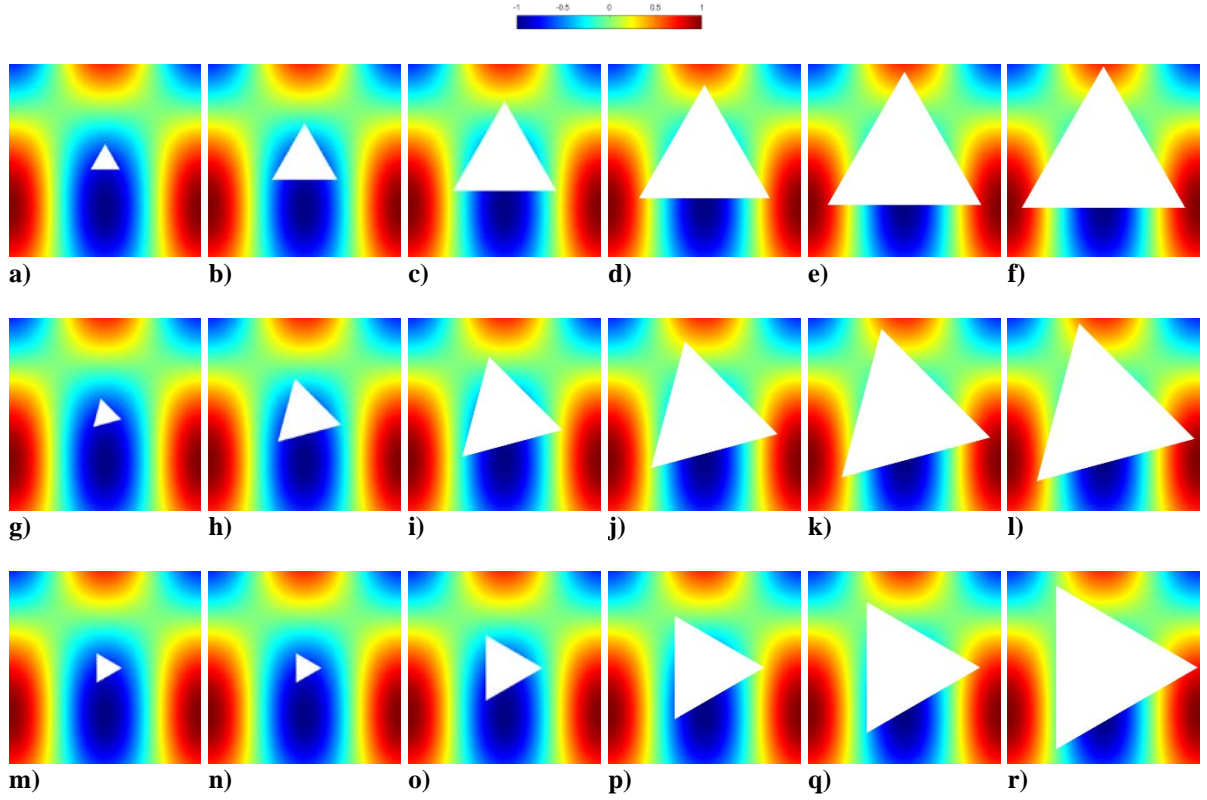
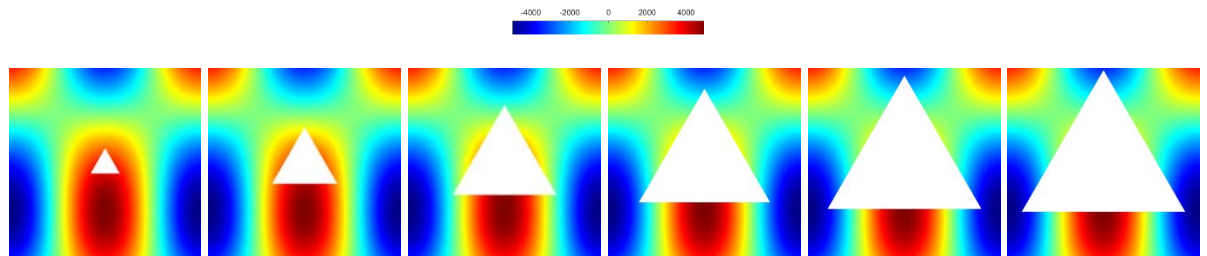


Figure 19. Prescribed MMS temperature field contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$



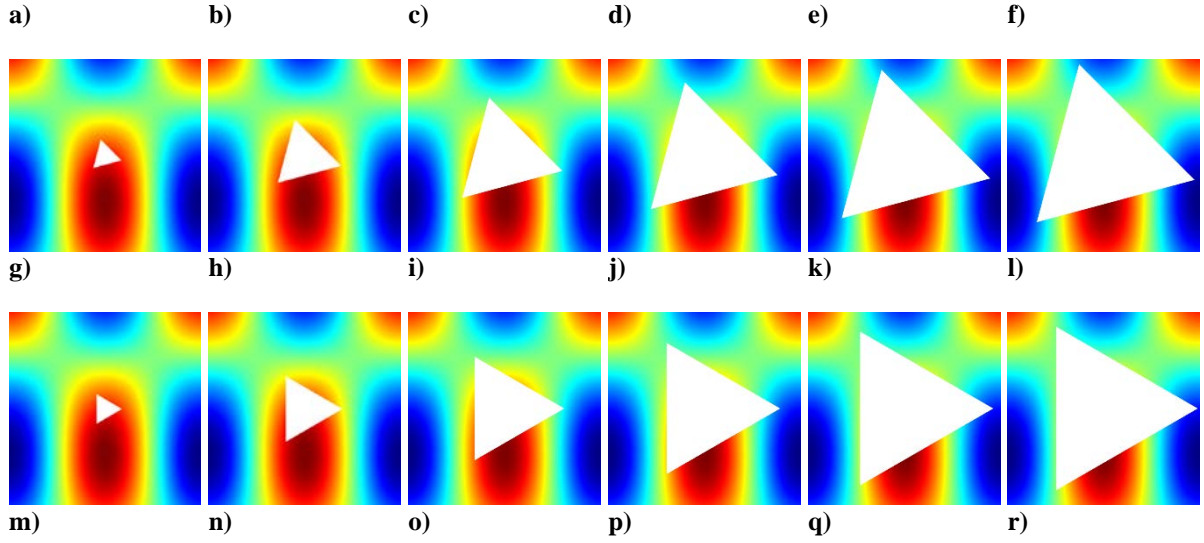


Figure 20. Applied MMS source field contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5,$ and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$

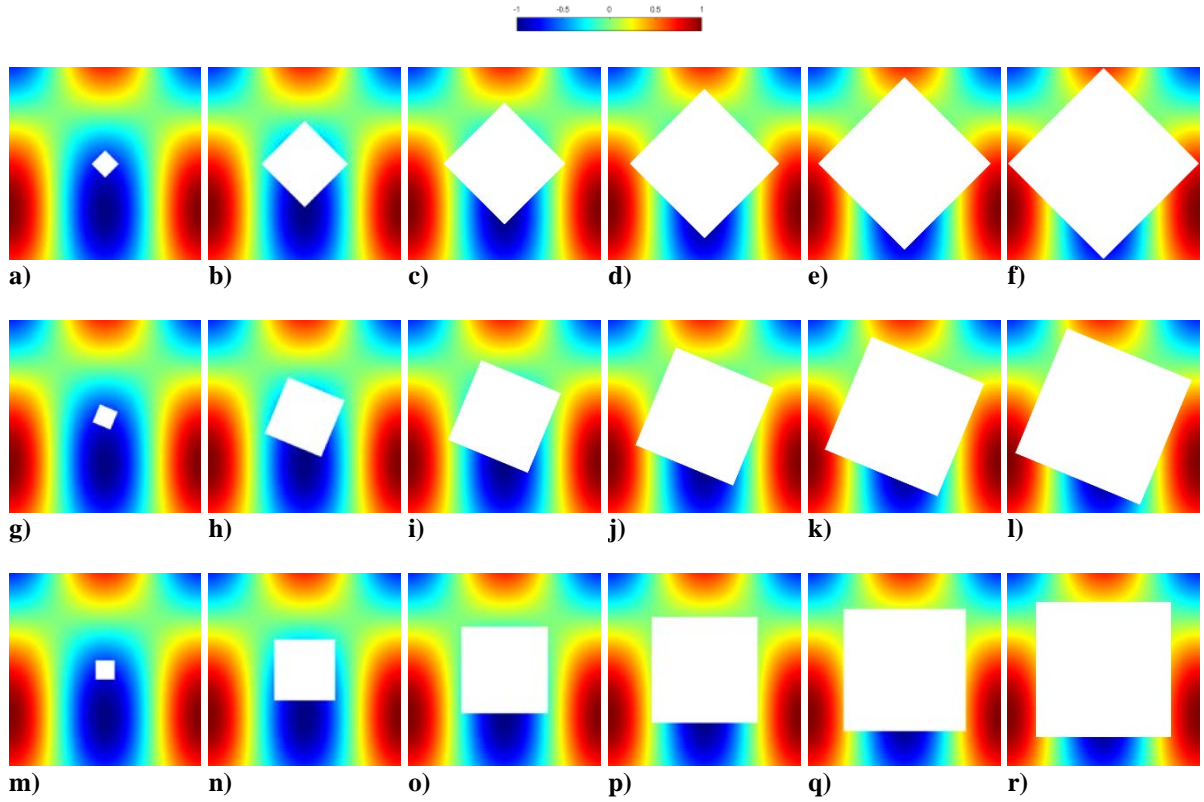


Figure 21. Prescribed MMS temperature field contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$

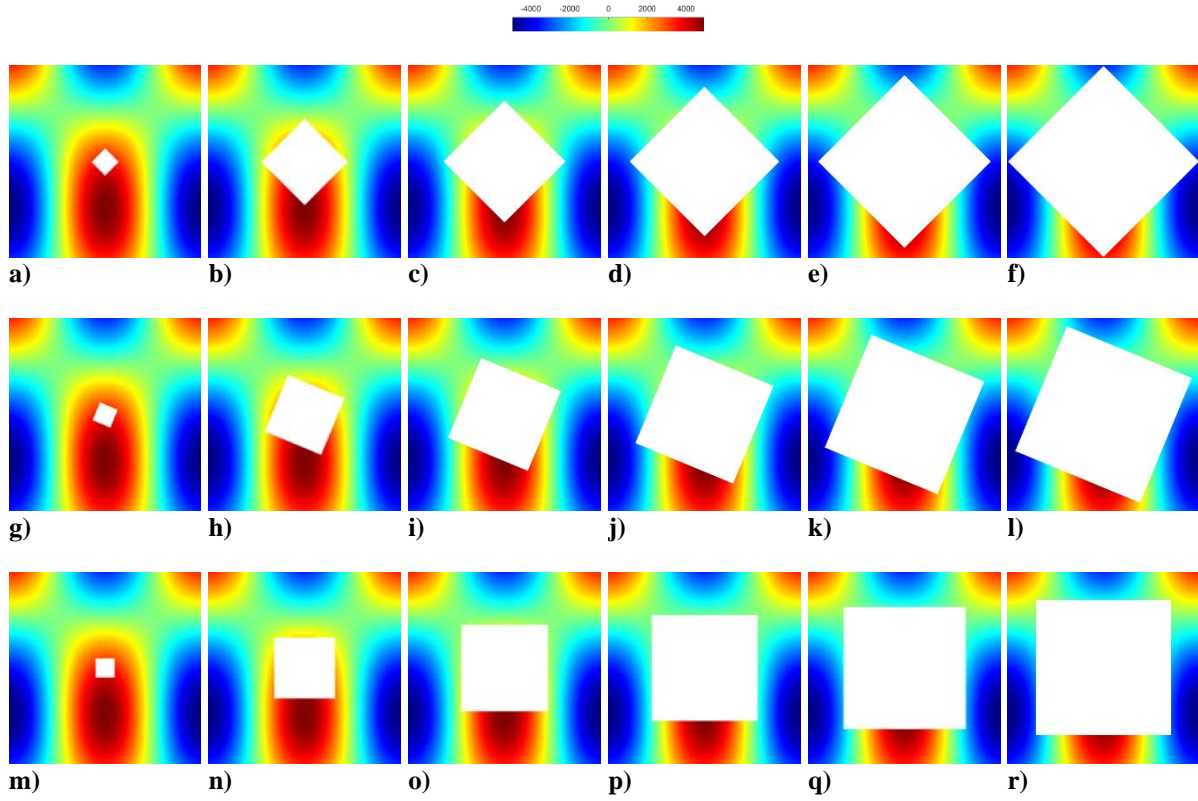


Figure 22. Applied MMS source field contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$

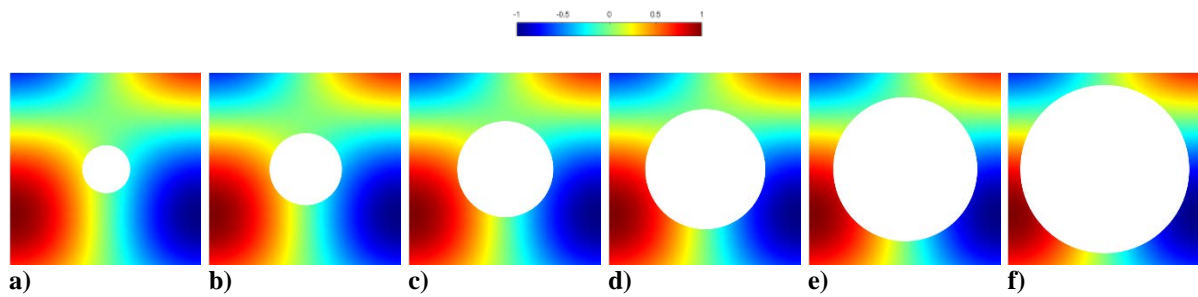


Figure 23. Prescribed MMS temperature field contours for circle geometries with a) $\alpha=4.9\%$, b) $\alpha=11.0\%$, c) $\alpha=19.6\%$, d) $\alpha=30.7\%$, e) $\alpha=44.2\%$, and f) $\alpha=60.1\%$

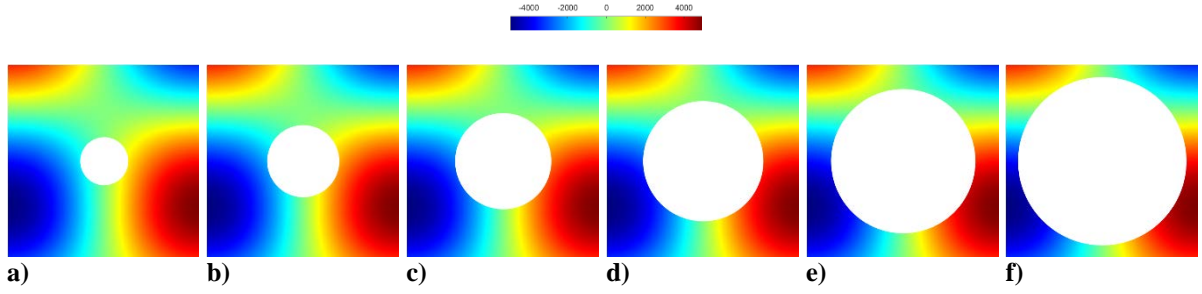


Figure 24. Applied MMS source field contours for circle geometries with a) $\alpha=4.9\%$, b) $\alpha=11.0\%$, c) $\alpha=19.6\%$, d) $\alpha=30.7\%$, e) $\alpha=44.2\%$, and f) $\alpha=60.1\%$

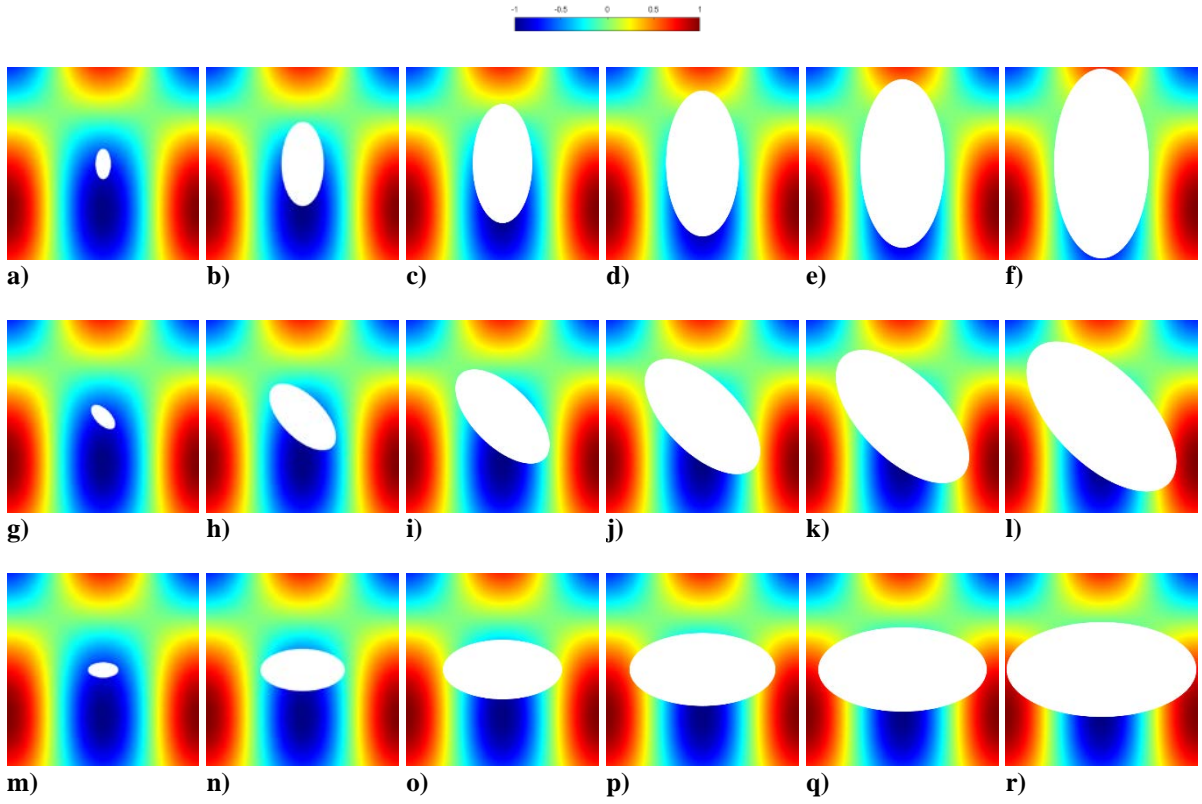
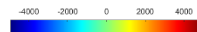


Figure 25. Prescribed MMS temperature field contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$



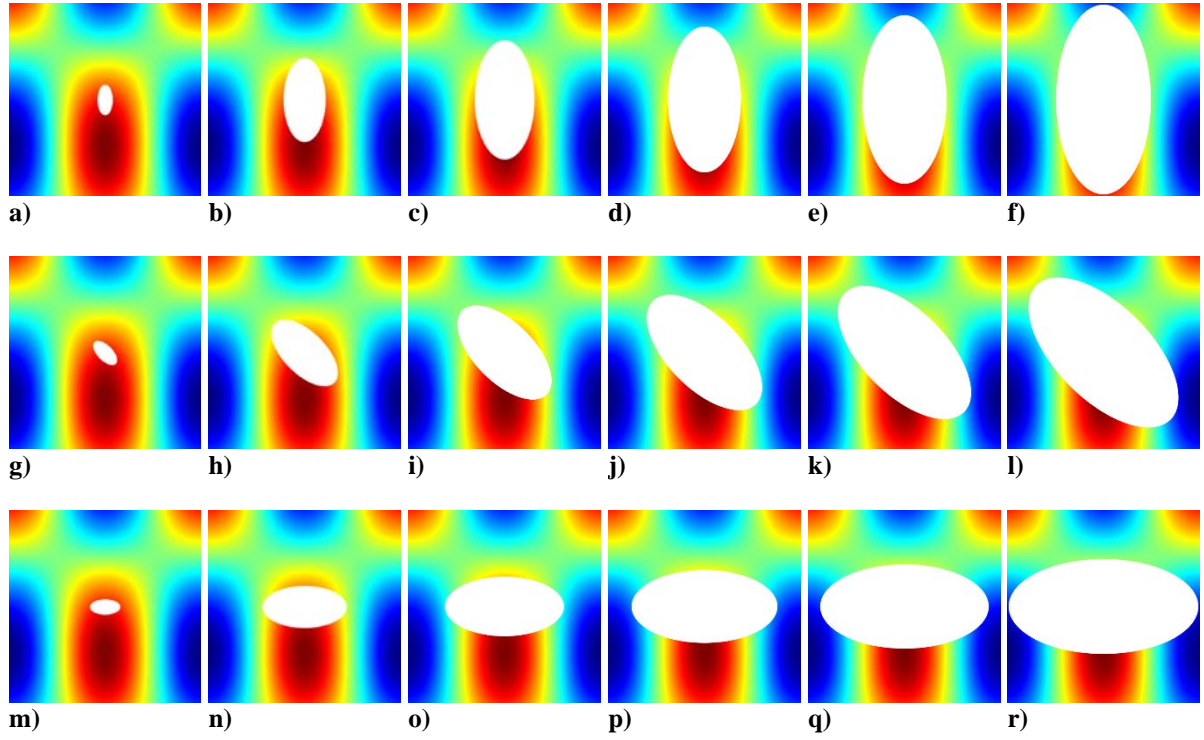


Figure 26. Applied MMS source field contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

4.3 Solution Verification

Solution verification was performed on k^* for the problem of interest. A global deviation grid convergence index (GCI) approach was used to estimate the numerical error, U_{num} , on k^* [45]. A series of five systematically refined meshes was used for each system configuration to evaluate convergence of k^* with mesh refinement. Theory suggests that as a computational mesh is refined, the numerical solution will asymptotically approach some distinct solution. When the solution is in the region of the asymptotic solution, the solution is said to be in the “asymptotic regime.” Conversely, solutions that are not converging systematically near the asymptotic solution are said to be in the “non-asymptotic regime.” The empirically-based global deviation GCI method computes a factor of safety on the finest mesh

k^* solution value based on a quantitative estimation of where the solution falls with respect to the asymptotic solution regime.

To approximate U_{num} with this global deviation GCI method on a fine mesh, first a modified transcendental order of accuracy on the finest of three meshes, p_t , is determined iteratively from

$$p_t = \ln \left[(r_{1,2}^{p_t} - 1) \left(\left| \frac{k_3^* - k_2^*}{k_2^* - k_1^*} \right| \right) + r_{1,2}^{p_t} \right] / \ln(r_{1,2} r_{2,3}), \quad (38)$$

where the subscripts 1, 2, and 3 denote the finest, middle, and coarsest of three meshes, successively refined. Likewise, $r_{i,j}$ is the mesh ratio between two successive meshes, where

$$r_{i,j} = h_j / h_i. \quad (39)$$

For this application, a global order of accuracy deviation value, Δp , is determined by

$$\Delta p = \min(|p_f - p_t|, 4p_f, 0.95p_f). \quad (40)$$

The deviation is used to determine the average global order of accuracy, p^* , defined as

$$p^* = p_f - \Delta p, \quad (41)$$

which in turn is used to compute a factor of safety, FS —based on the inferred proximity of the solution to the asymptotic regime—where,

$$FS = 3.0 - 1.9(p^*/p_f)^8. \quad (42)$$

Figure 27 depicts the behavior and smooth transition of FS with respect to the global order of accuracy deviation for a formally second order accurate method. In the asymptotic

regime, FS approaches 1.1, and in the non-asymptotic regime, FS approaches 3.0. The original GCI method assumed that FS would take on the value of either 1.25 or 3.00 depending on the number of meshes and the observed order of accuracy [46], however the method used here continuously scales FS .

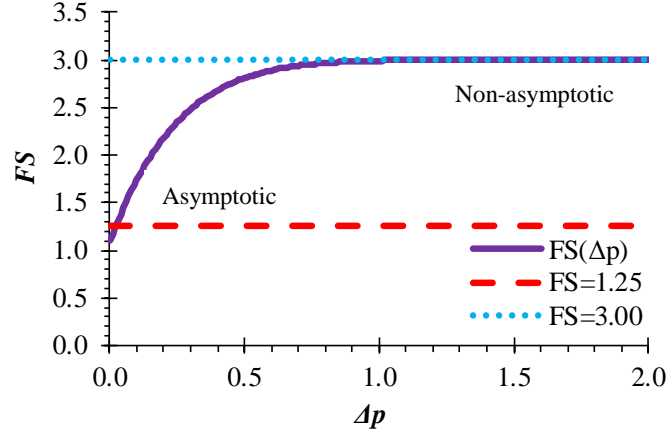


Figure 27. Global deviation estimator factor of safety

An estimate for U_{num} on k^* from the finest of the three systematically refined meshes is

$$U_{num} = FS \left| (k_2^* - k_1^*) / (r_{1,2}^{p^*} - 1) \right|, \quad (43)$$

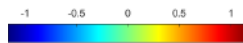
providing a convergence-based uncertainty estimator for discretization-induced error.

5 Results and Discussion

Using the MMS solution described above for all the geometry combinations given in Table 1, the MMS observed order of accuracy is shown to converge to approximately second order for all the analyzed systems, where the computational solution temperature is compared against the prescribed T_{MMS} distribution. Figure 28 shows an example of the computational MMS temperature solution on a triangle pore mesh. The solution was solved with the applied

MMS source term distribution described previously and boundary conditions enforced on the domain consistent in type with the true problem and consistent in value with T_{MMS} . Figure 29 shows the computational MMS temperature solution for fine meshes on a variety of triangle pore system configurations. Likewise, Figure 30 and Figure 31 show computational MMS temperature solutions with mesh refinement and on fine meshes, respectively, for square pore geometries. Figure 32 and Figure 33 show computational MMS temperature solutions with mesh refinement and on fine meshes, respectively, for circle pore geometries. Lastly, Figure 34 and Figure 35 show computational MMS temperature solutions with mesh refinement and on fine meshes, respectively, for ellipse pore system configurations. Note that for each pore type, the numerical temperature distribution approaches the expected, prescribed MMS temperature distribution. Similarly, note that the fine mesh computational MMS solutions are virtually qualitatively indistinguishable from the prescribed MMS temperature solution.

Figure 36 through Figure 39 illustrate the convergence of $p_{O,H}$ with H , where p_f is indicated by the dashed line, suggesting that the numerical method was correctly implemented to solve the heat equation, being second order accurate. Note that L_∞ norm of errors was not computed for five of the circle pore geometries as a minor and relatively inconsequential technical oversight, losing virtually no information about the performance of the system. Order of accuracy results plotted here are shown only for selected geometries to illustrate the maximum spread in results.



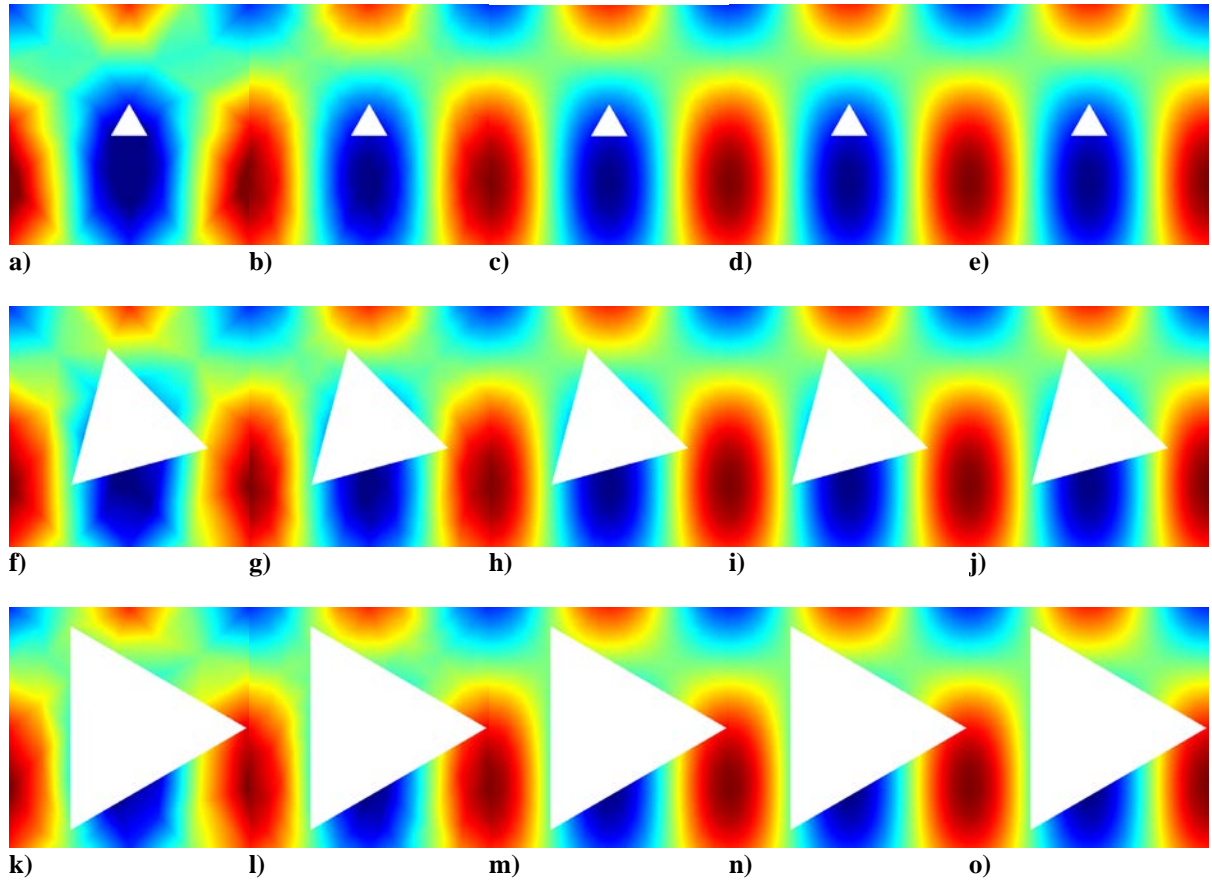
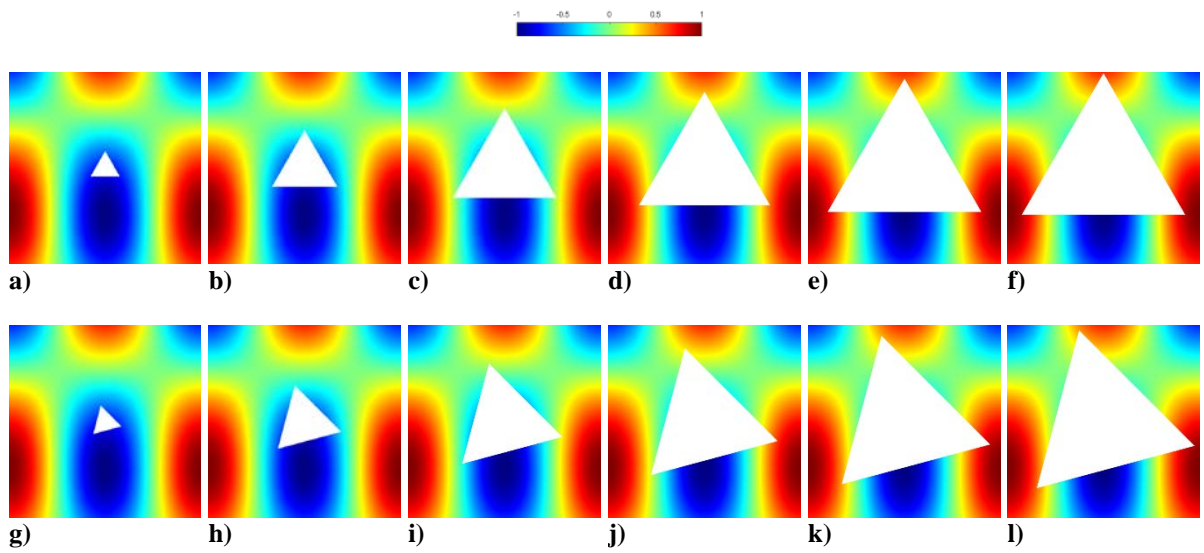


Figure 28. Computational MMS temperature contours for triangle geometries with mesh refinement for $H=5, 4, 3, 2, 1$ with a) through e) $\alpha=1.0\%$ and $\theta_\alpha=0.0^\circ$, f) through j) $\alpha=15.0\%$ and $\theta_\alpha=15.0^\circ$, and k) through o) $\alpha=31.0\%$ and $\theta_\alpha=30.0^\circ$



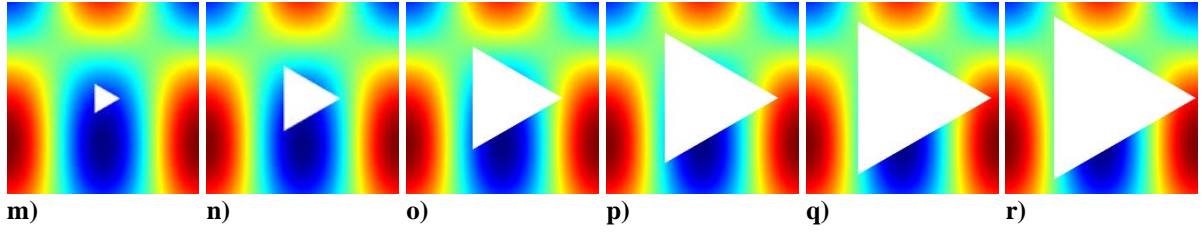


Figure 29. Computational MMS temperature contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=15.0^\circ$, and m) through r) $\theta_\alpha=30.0^\circ$

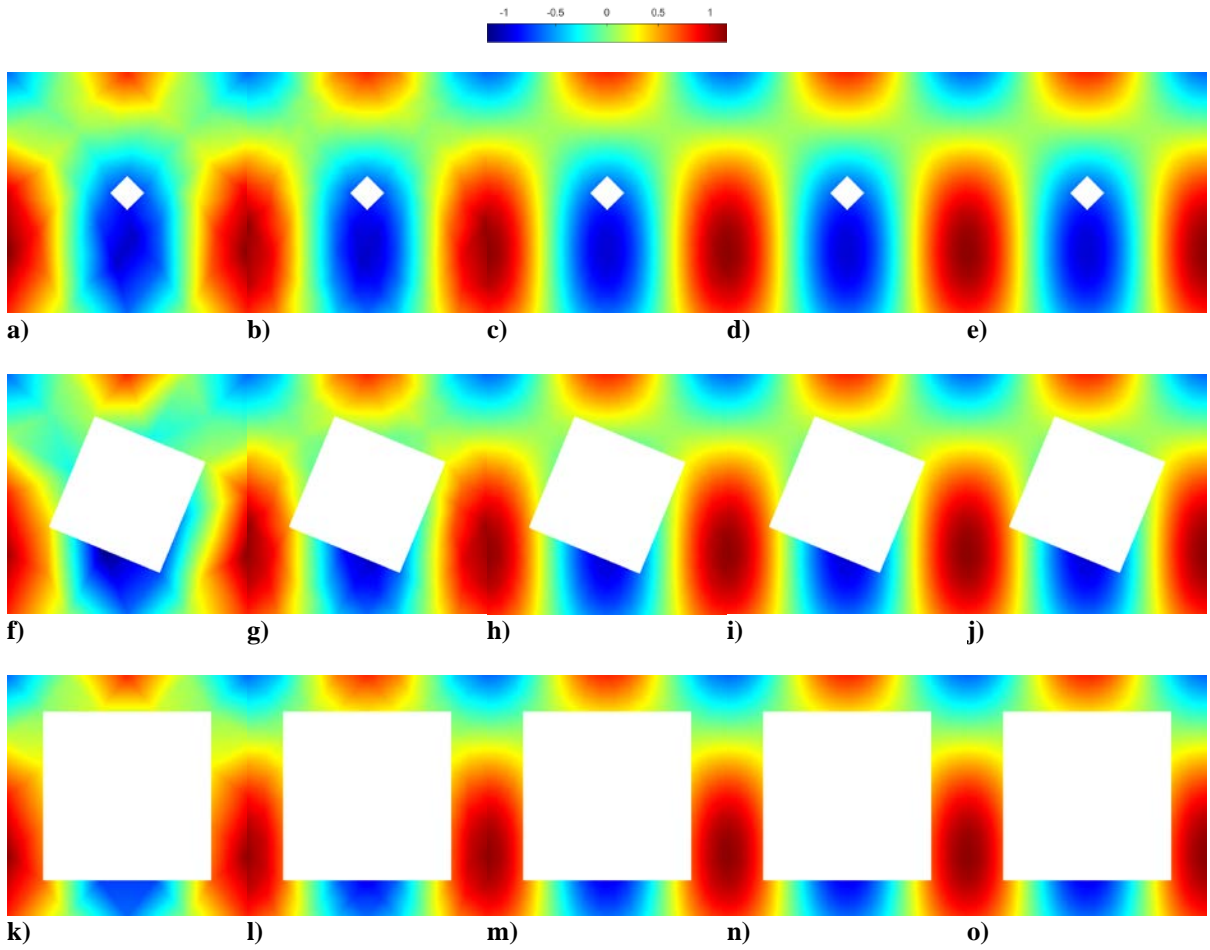


Figure 30. Computational MMS temperature contours for square geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_\alpha=0.0^\circ$, f) through j) $\alpha=25.0\%$ and $\theta_\alpha=22.5^\circ$, and k) through o) $\alpha=49.0\%$ and $\theta_\alpha=45.0^\circ$



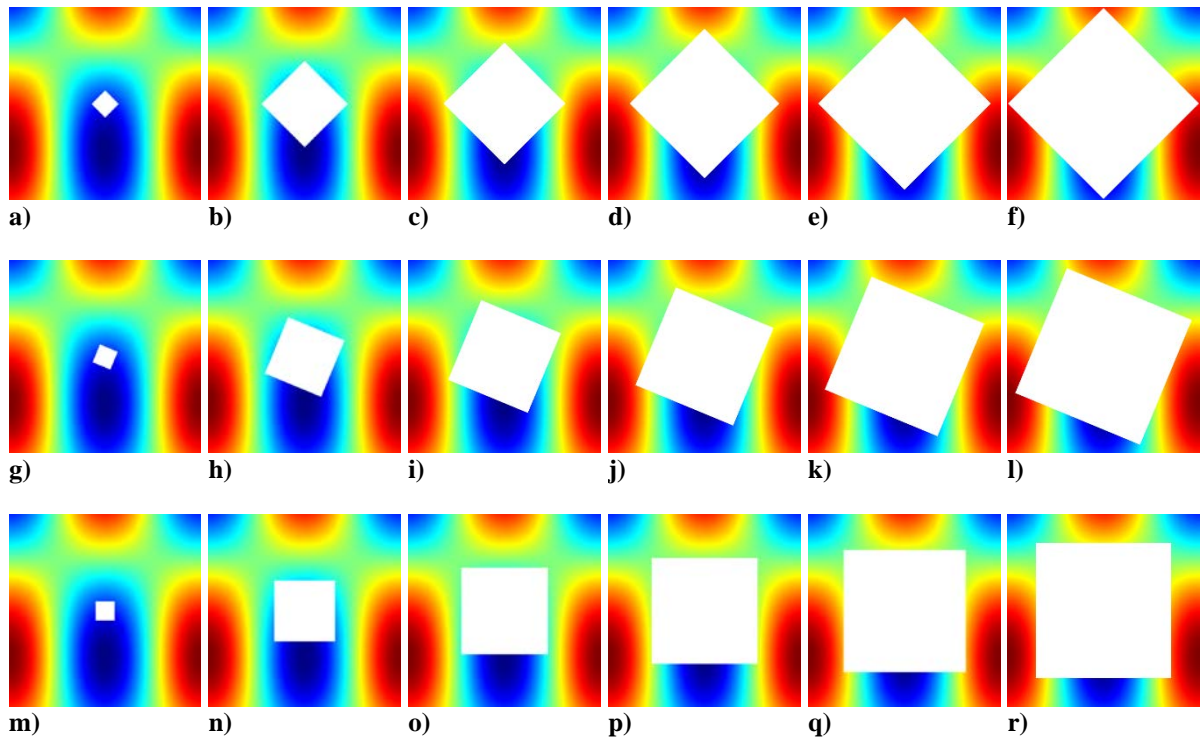
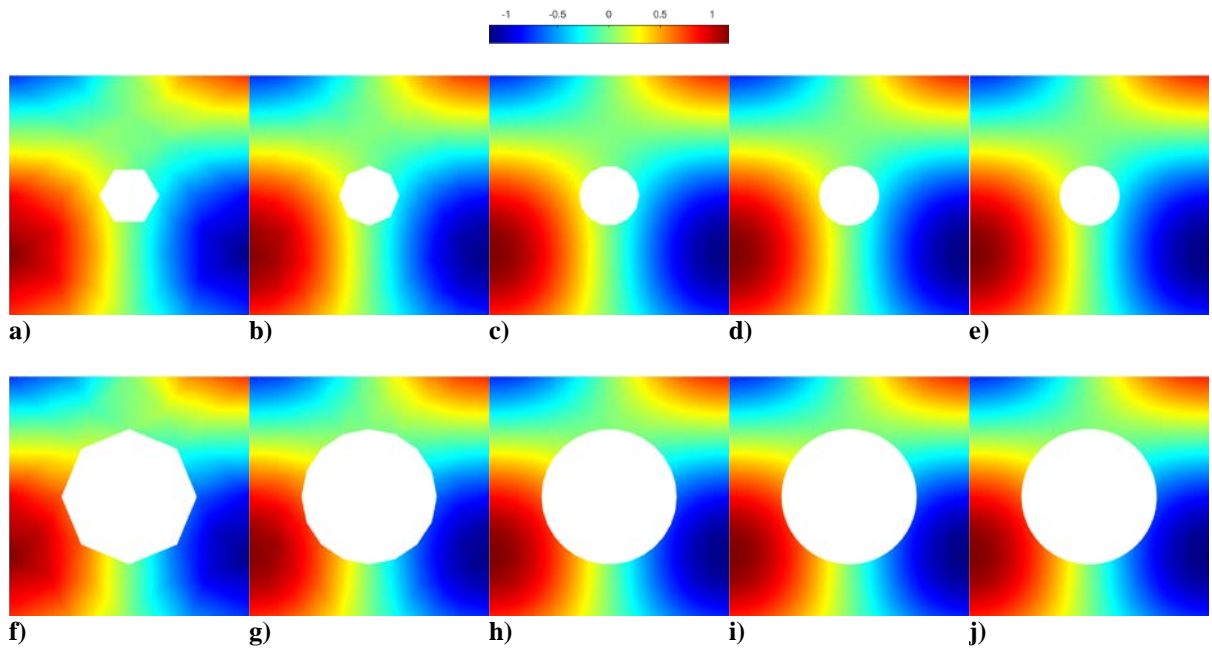


Figure 31. Computational MMS temperature contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$



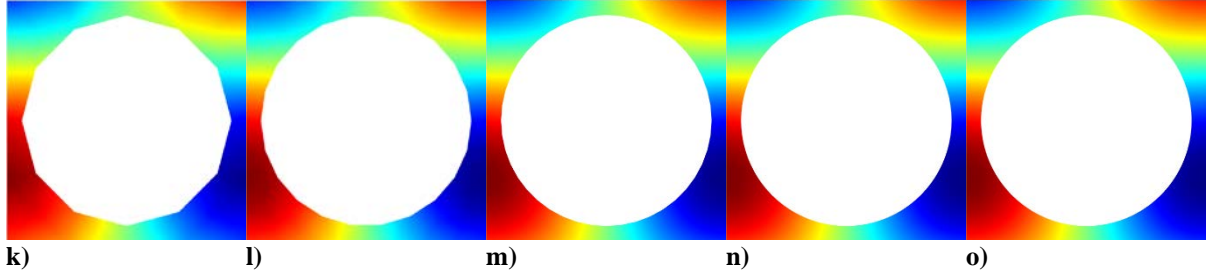


Figure 32. Computational MMS temperature contours for circle geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=4.9\%$, f) through j) $\alpha=24.9\%$, and k) through o) $\alpha=60.1\%$

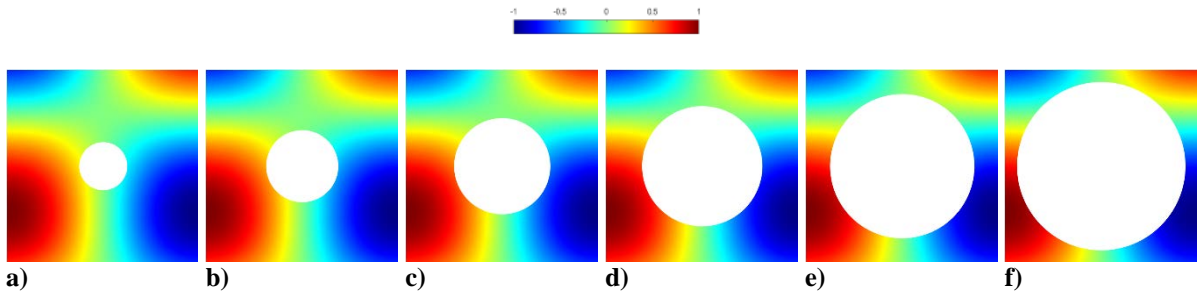
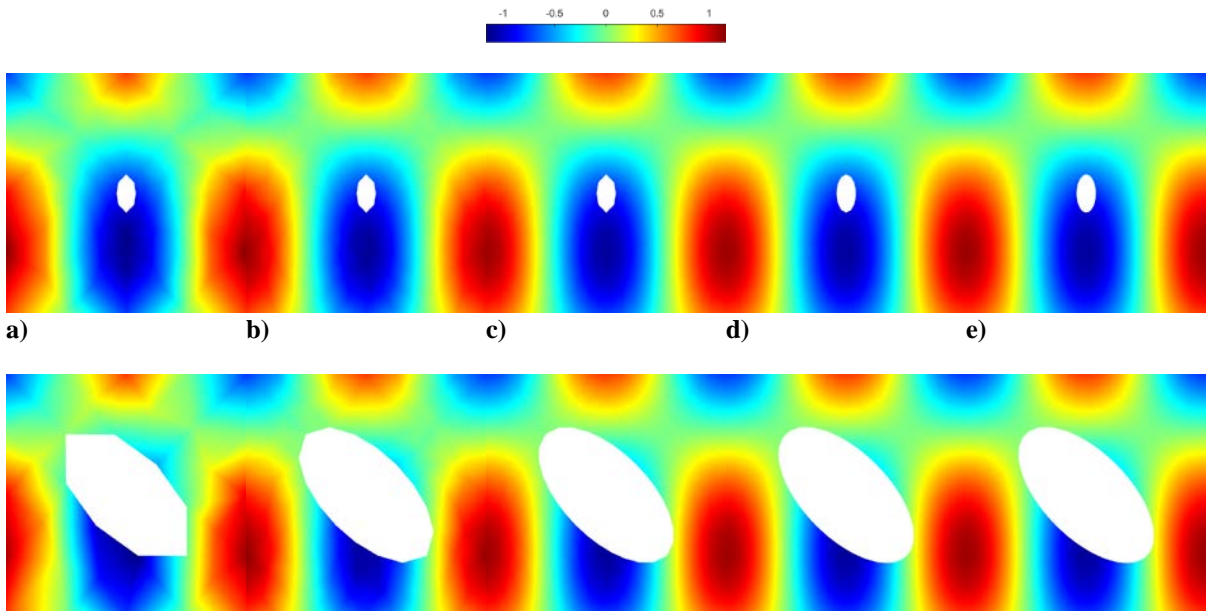


Figure 33. Computational MMS temperature contours for circle geometries with a) $\alpha=4.9\%$, b) $\alpha=11.0\%$, c) $\alpha=19.6\%$, d) $\alpha=30.7\%$, e) $\alpha=44.2\%$, and f) $\alpha=60.1\%$



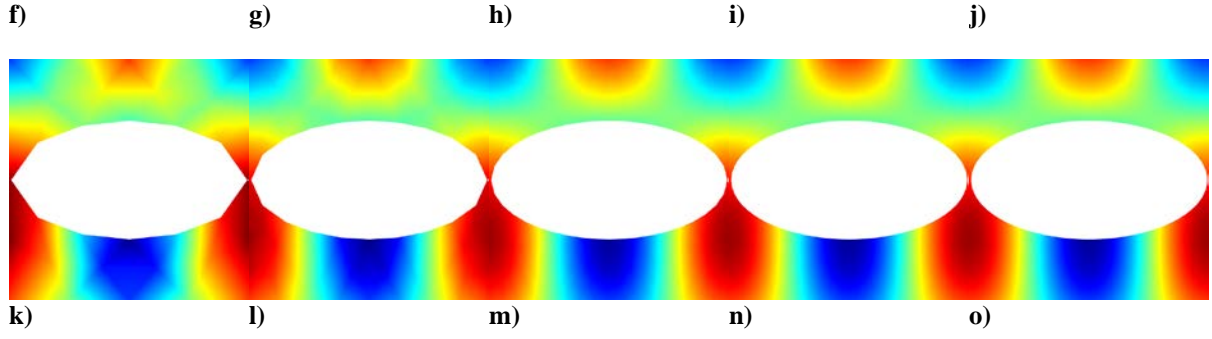


Figure 34. Computational MMS temperature contours for ellipse geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_\alpha=0.0^\circ$, f) through j) $\alpha=20.0\%$ and $\theta_\alpha=45.0^\circ$, and k) through o) $\alpha=38.0\%$ and $\theta_\alpha=90.0^\circ$

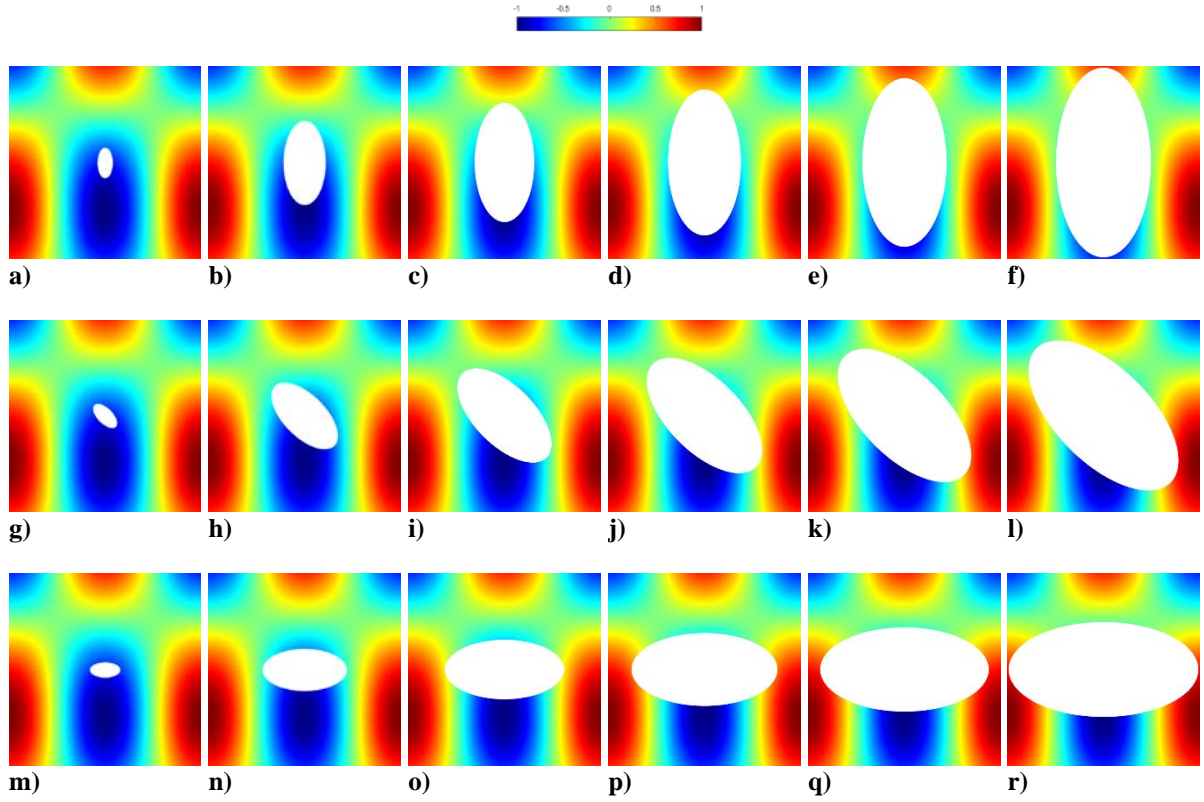


Figure 35. Computational MMS temperature contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0$, and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

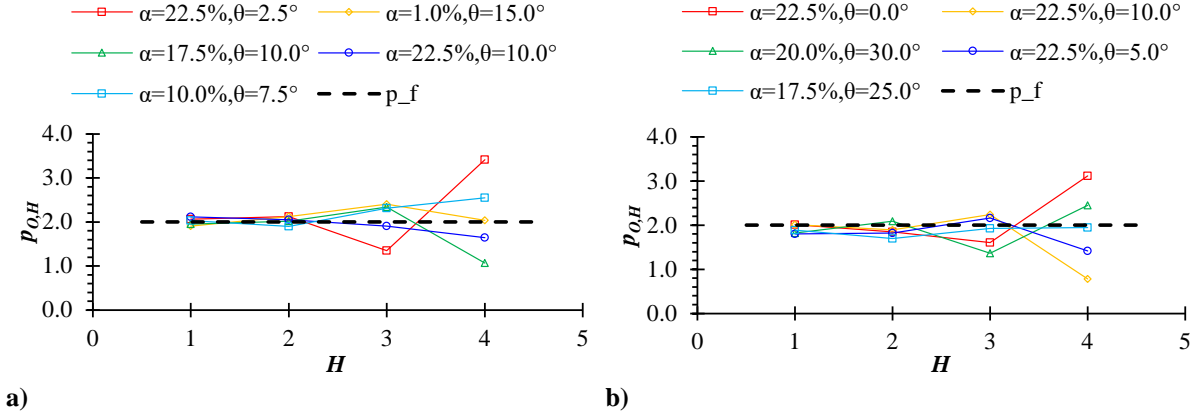


Figure 36. MMS observed order of accuracy convergence with mesh number for select triangle pore systems using a) RMS error and b) L_∞ error

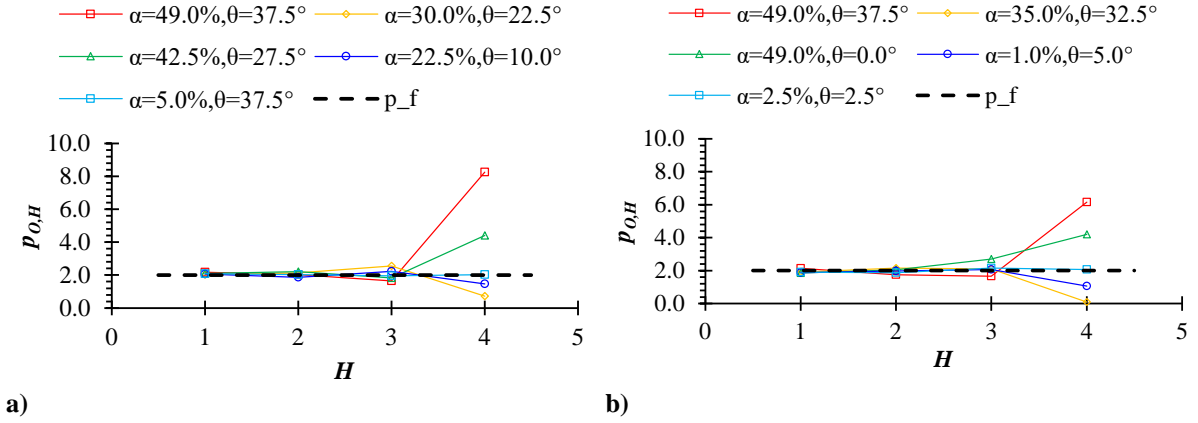


Figure 37. MMS observed order of accuracy convergence with mesh number for select square pore systems using a) RMS error and b) L_∞ error

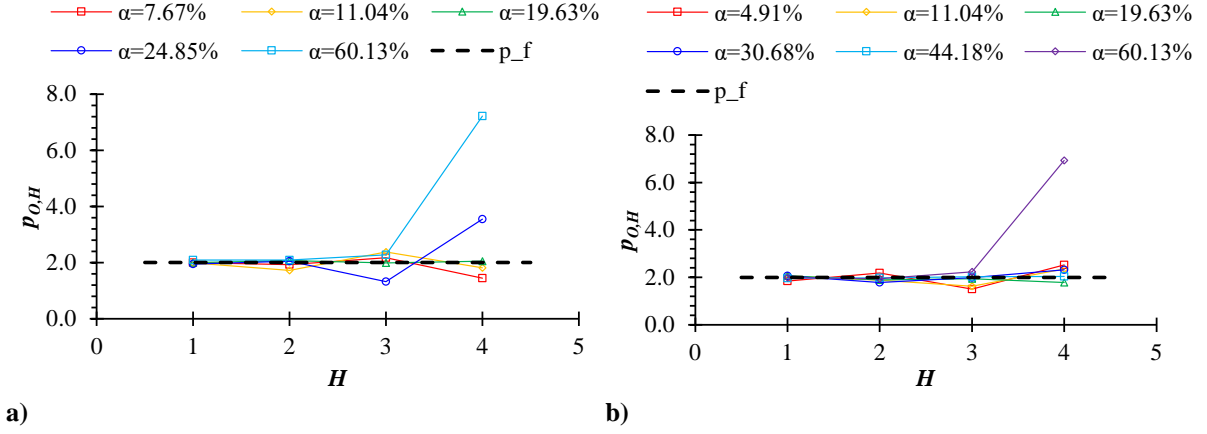


Figure 38. MMS observed order of accuracy convergence with mesh number for select circle pore systems using a) RMS error and b) L_∞ error

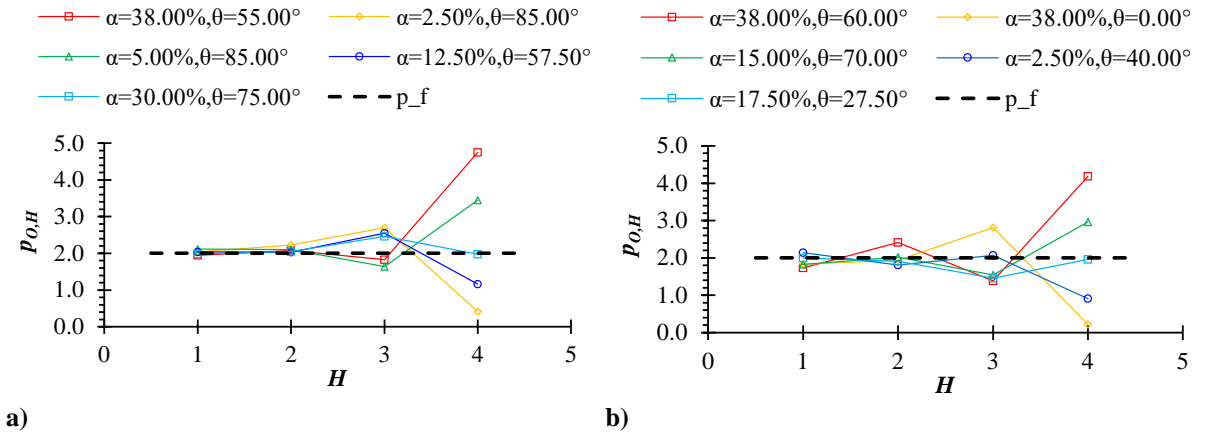


Figure 39. MMS observed order of accuracy convergence with mesh number for select ellipse pore systems using a) RMS error and b) L_∞ error

Systematically refined meshes were used with the GCI method to estimate U_{num} in the computationally-determined k^* for each system configuration. Figure 40, Figure 42, Figure 44, and Figure 46 show resultant temperature distribution contour plots for selected geometry configurations with mesh refinement with qualitative convergence on the temperature distribution solutions. Figure 41, Figure 43, Figure 45, and Figure 47 present contour plots for

the finest meshes of changing α and θ_α for each of the three pore geometries. These figures show the qualitative effects of pore size and pore rotation—where applicable—on the temperature distributions with the unit cell. From the contour plots, it is evident that increasing pore size generally increases temperature gradient near the centerline of the system between the hot and cold boundaries. Rotation of the pores has asymmetric effects on the temperature profiles, with varying effects on the gradients, for the non-circular geometries.

By normalizing clocking angles by 180° , rotation of the pore can be defined on a scale of 0 to 1, where the dimensionless clocking angle, ψ , is simply

$$\psi = \theta_\alpha / 180^\circ. \quad (44)$$

Resulting k^* curve families are shown in Figure 48 for the triangle, square, and ellipse pore geometries as determined by the finest mesh computational solutions. In the plots, the dashed lines are extensions of the computed values based on geometric symmetry. It is evident from these plots that the effects of θ_α on k^* are generally accentuated with increasing α , and k^* decreases monotonically with increased α , as expected. However, k^* appears to have virtually no change with respect to ψ for constant α in the triangle pore systems. The constant k^* values are likely due to the invariable integral of the contraction and restriction profiles in the direction of the domain temperature gradient. Note the relatively disparate behaviors of the different geometry systems.

Figure 49a through Figure 49a show the estimated U_{num} values as computed using the global deviation GCI method on the finest mesh k^* values, normalized by the k^* values at the respective positions. The triangular pore geometry numerical uncertainties are generally constant with changing θ_α . The higher uncertainty levels are manifested for the square pore

geometries when the solid domain is restricted at the adiabatic domain edges near $x=L/2$. However, U_{num} stays below 6.0% for the triangle systems in all but a single point, where the numerical uncertainty is still less than 10.0%. For the square geometries, uncertainties remain below 6.0% in all but four data points. The highest uncertainty value is 37.0% for the highest porosity at 0.0 ° rotation. This is likely due to poor mesh refinement between the tips of the poor and the adiabatic external boundaries where the largest temperature gradients occur due to heat path restrictions. Note that the color legend in Figure 49b was set to a maximum of 10.0% to show the generally low U_{num} distribution for the vast majority of the parameter domain. In the circle geometries, uncertainties are all well below 1.0%. For the ellipse geometries, all uncertainties remain below 4.0%.

Using the normalized angle of reflection, ψ_{ref} , a continuously cyclic normalized clocking angle, ψ_{cyc} , is described by

$$\psi_{cyc} = \psi_{ref} \left(\frac{1}{\pi} \sin^{-1} \left(\sin \left(\pi \left(\frac{\psi}{\psi_{ref}} - 0.5 \right) \right) \right) + 0.5 \right). \quad (45)$$

Using this cyclic angle parameter, the k^* responses within the α ranges given in Table 1 can be characterized using a regression function, f_{reg} , of the form

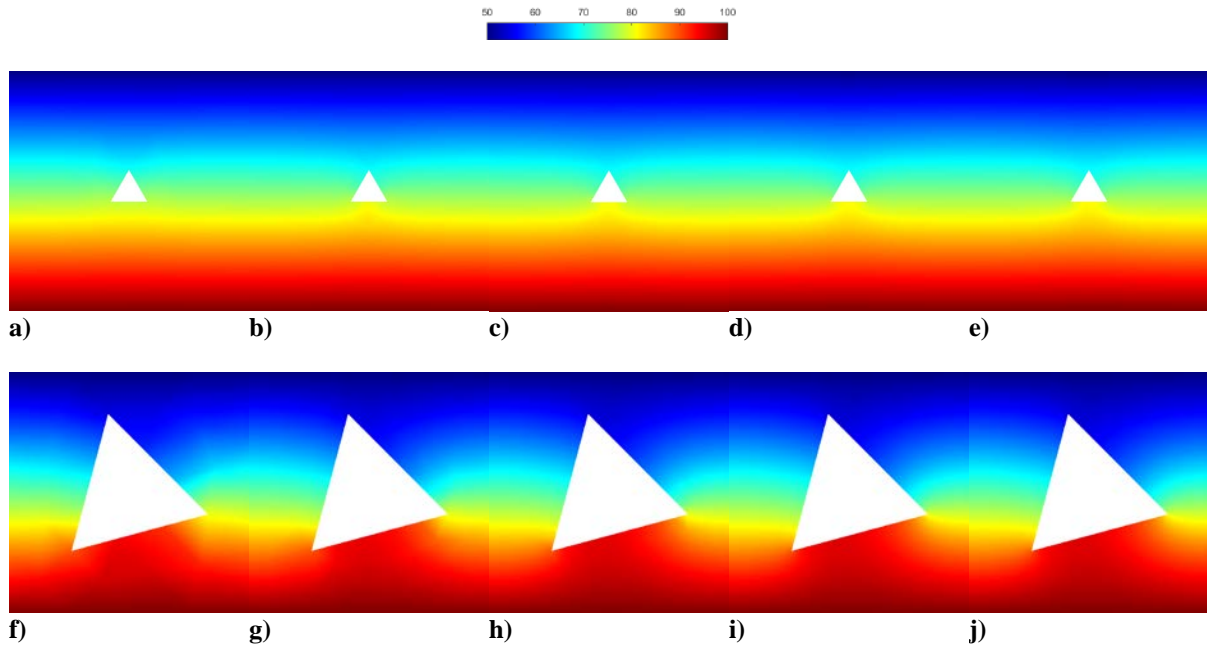
$$\begin{aligned} f_{reg} = & 1 + a_{c,1}\alpha + a_{c,2}\alpha^2 + \dots \\ & a_{g,1}((a_{c,3} - a_{c,1})\alpha + (a_{c,4} - a_{c,2})\alpha^2)\psi_{cyc}^2 + \dots, \\ & a_{g,2}((a_{c,1} - a_{c,3})\alpha + (a_{c,2} - a_{c,4})\alpha^2)\psi_{cyc}^3 \end{aligned} \quad (46)$$

with coefficients $a_{c,1}$ through $a_{c,4}$ and $a_{g,1}$ and $a_{g,2}$. Here, the four a_c coefficients are calibration parameters, tuned to minimize the difference between f_{reg} and the computational k^* values. The a_g coefficients are geometric constants unique to each pore geometry. Table 2 gives the

coefficient values for f_{reg} for each of the pore geometry types. The cyclic angle formulation employed in Equation (46) allows the regression function to be applied to any clocking angle without restriction to the angle ranges shown in Table 1. Figure 50 illustrates the regression function applied to the three different geometries in the same manner as the computational results in Figure 48. The relative difference between the regression evaluations and the simulation results for triangle, square, and ellipse pore geometries are given in Figure 51a through Figure 51d, respectively. The relative difference, δk^* , is defined as

$$\delta k^* = (f_{reg} - k^*)/k^*. \quad (47)$$

The regressions agree with the numerical results to within less than 2.5% for the triangular pore, less than 5.0% for the square pore, less than 4.0% for the circle pore, and less than 6.0% for the elliptical pore.



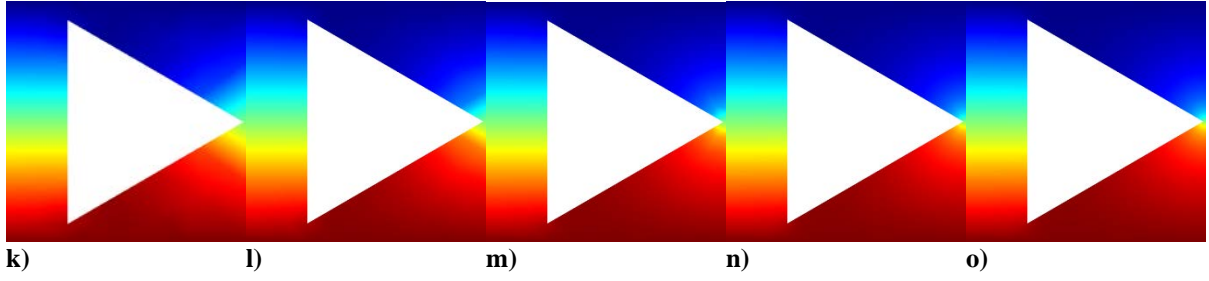


Figure 40. Temperature solution contours for triangle geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_a=0.0^\circ$, f) through j) $\alpha=15.0\%$ and $\theta_a=15.0^\circ$, and k) through o) $\alpha=31.0\%$ and $\theta_a=30.0^\circ$

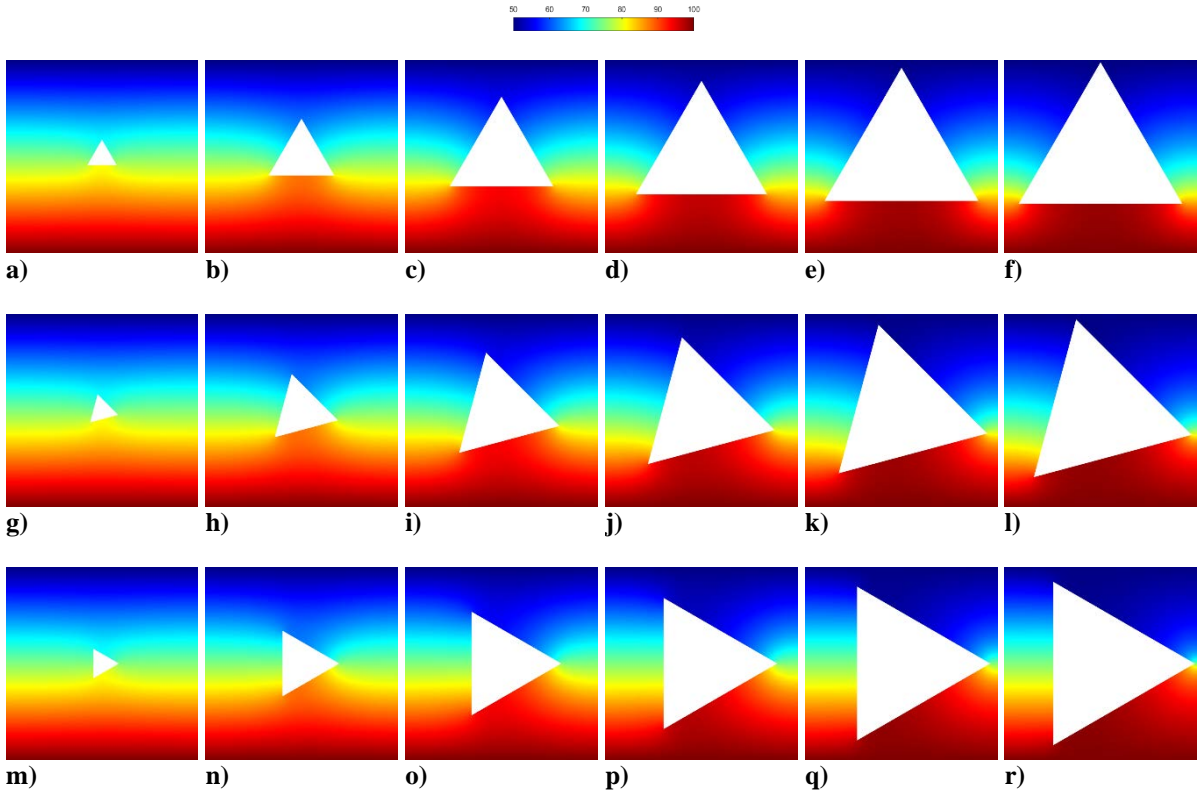


Figure 41. Temperature solution contours for triangle geometries with $\alpha=1.0, 5.0, 12.5, 20.0, 27.5$, and 31.0% and a) through f) $\theta_a=0.0^\circ$, g) through l) $\theta_a=15.0^\circ$, and m) through r) $\theta_a=30.0^\circ$



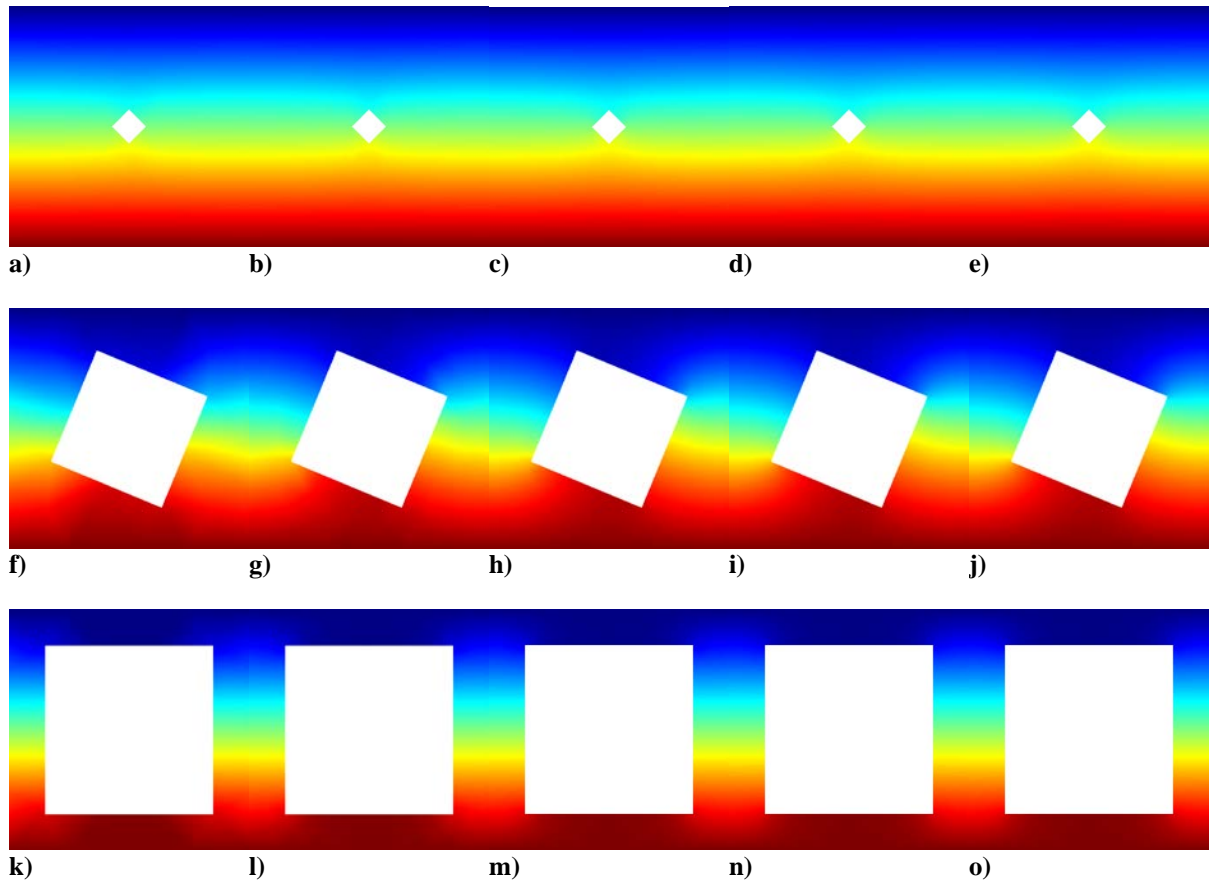
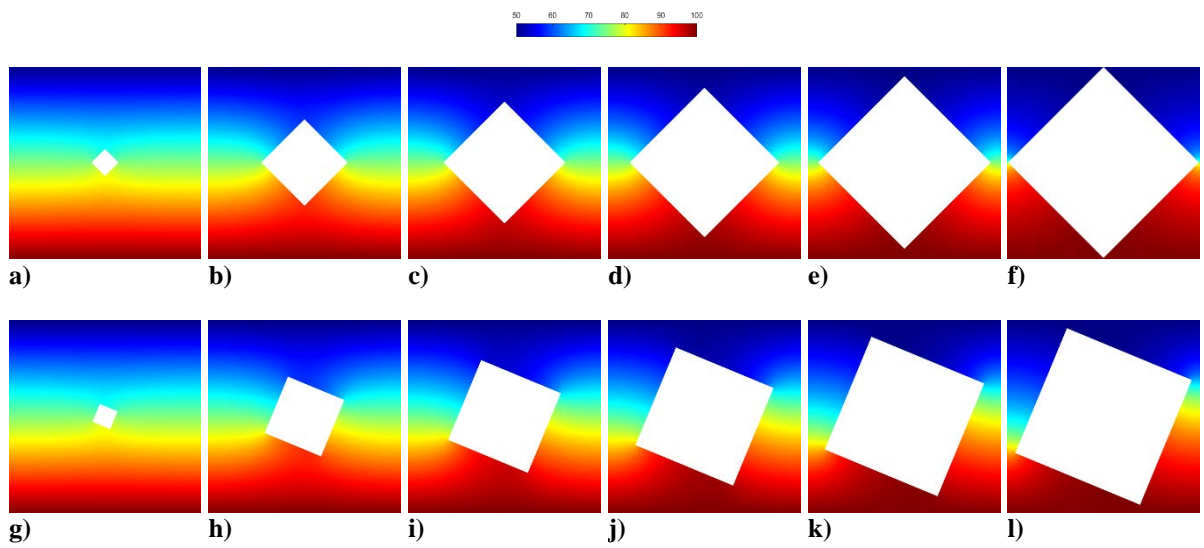


Figure 42. Temperature solution contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$



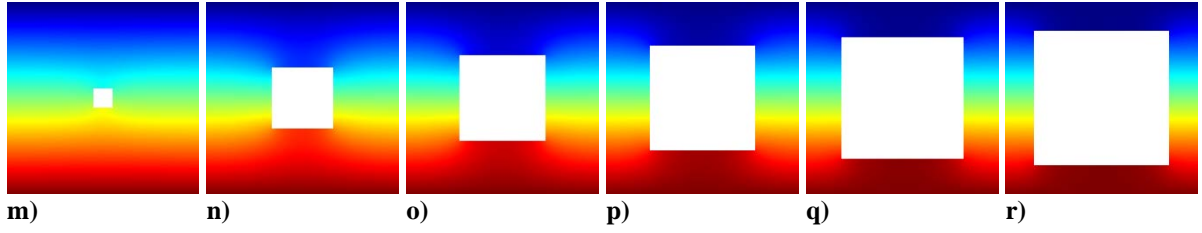


Figure 43. Temperature solution contours for square geometries with $\alpha=1.0, 10.0, 20.0, 30.0, 40.0$, and 49.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=22.5^\circ$, and m) through r) $\theta_\alpha=45.0^\circ$

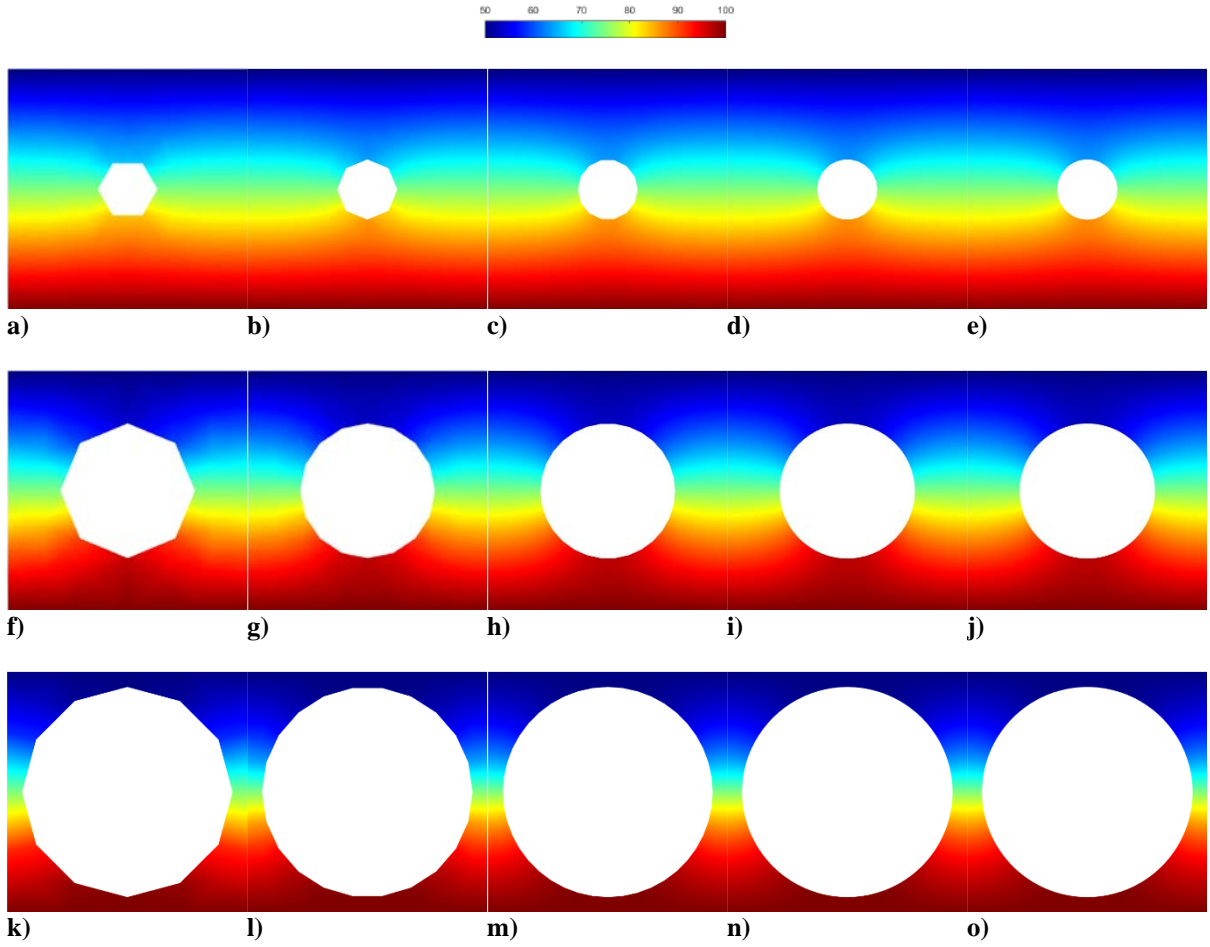


Figure 44. Temperature solution contours for circle geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=4.9\%$, f) through j) $\alpha=24.9\%$, and k) through o) $\alpha=60.1\%$

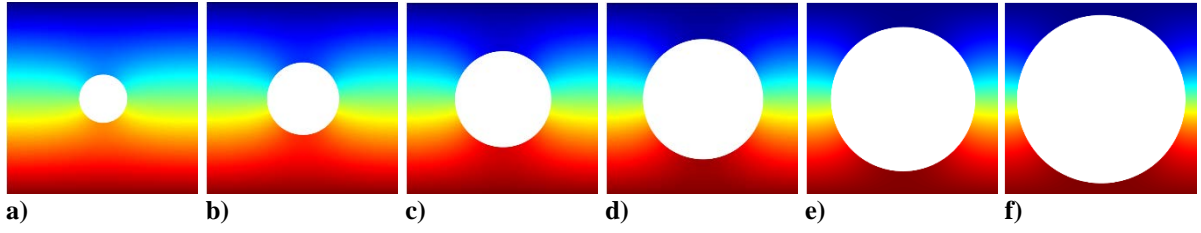


Figure 45. Temperature solution contours for circle geometries with a) $\alpha=4.9\%$, b) $\alpha=11.0\%$, c) $\alpha=19.6\%$, d) $\alpha=30.7\%$, e) $\alpha=44.2\%$, and f) $\alpha=60.1\%$

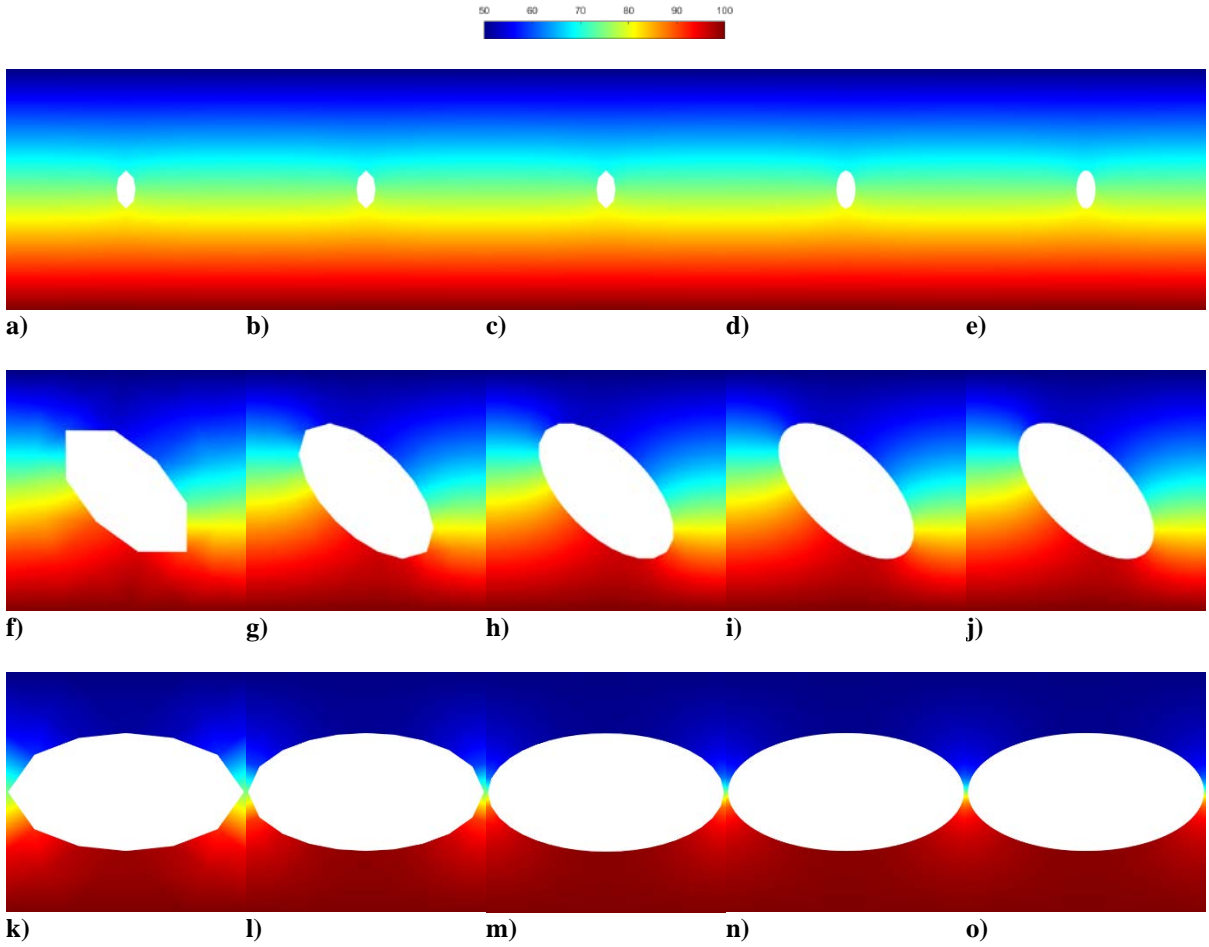


Figure 46. Temperature solution contours for ellipse geometries with mesh refinement for $H=5, 4, 3, 2$, and 1 with a) through e) $\alpha=1.0\%$ and $\theta_a=0.0^\circ$, f) through j) $\alpha=20.0\%$ and $\theta_a=45.0^\circ$, and k) through o) $\alpha=38.0\%$ and $\theta_a=90.0^\circ$



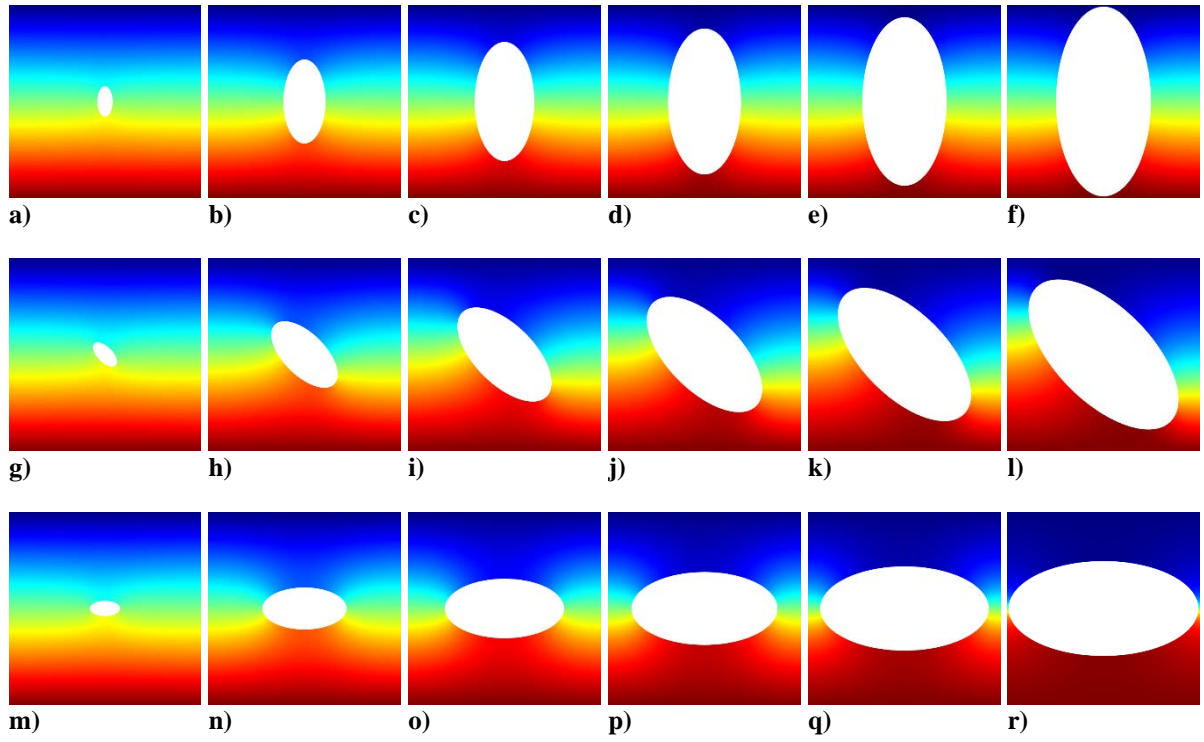


Figure 47. Temperature solution contours for ellipse geometries with $\alpha=1.0, 7.5, 15.0, 22.5, 30.0,$ and 38.0% and a) through f) $\theta_\alpha=0.0^\circ$, g) through l) $\theta_\alpha=45.0^\circ$, and m) through r) $\theta_\alpha=90.0^\circ$

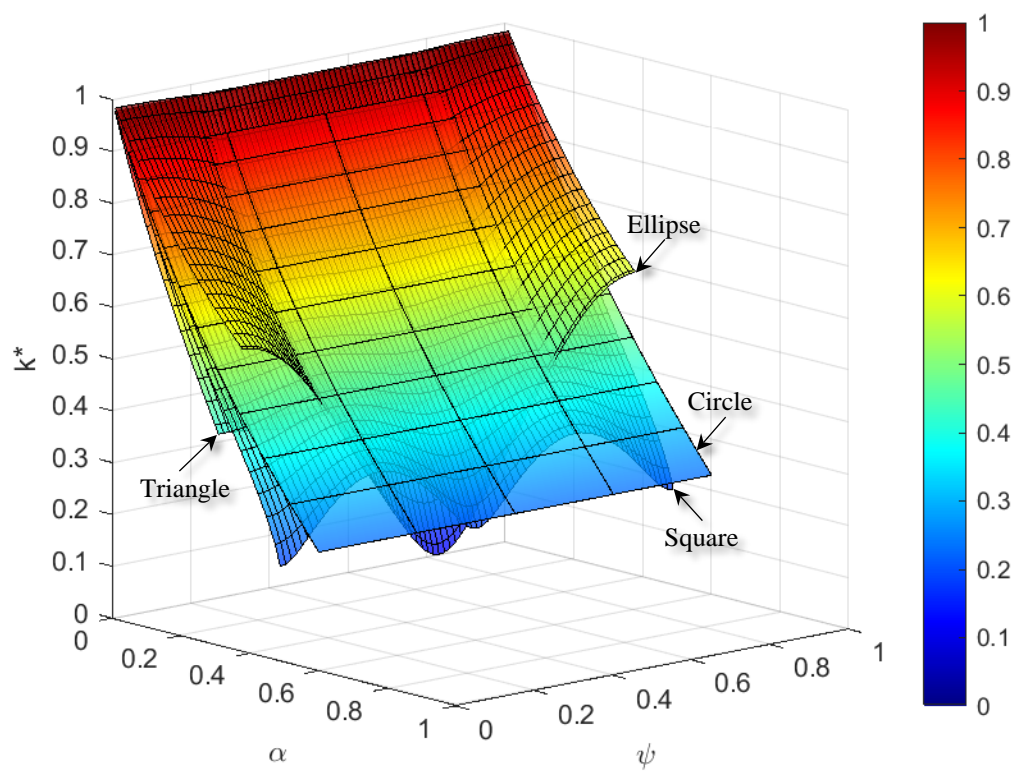
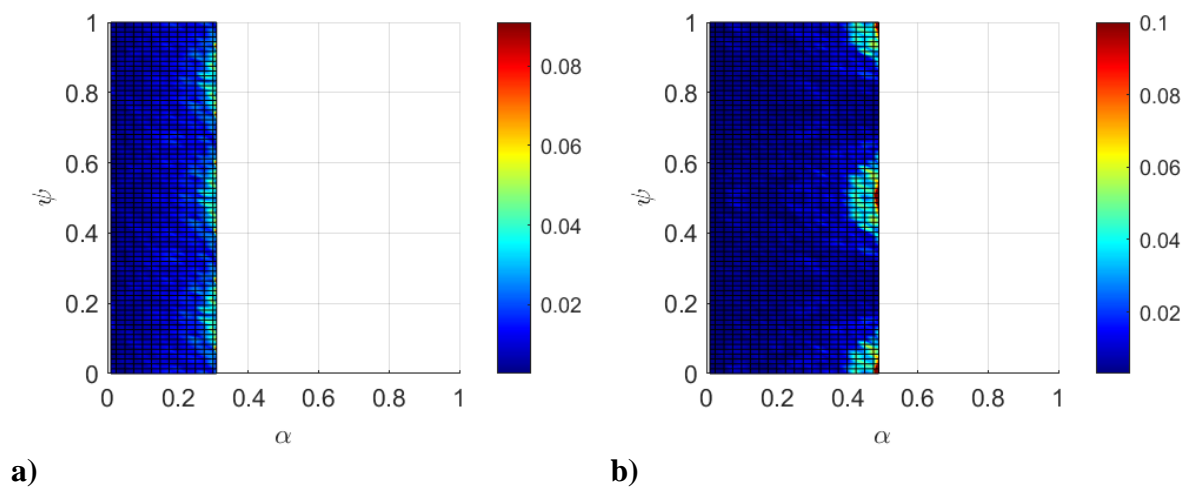


Figure 48. Computational k^* response as a function of porosity and clocking angle for triangle, square, circle, and ellipse pore geometries



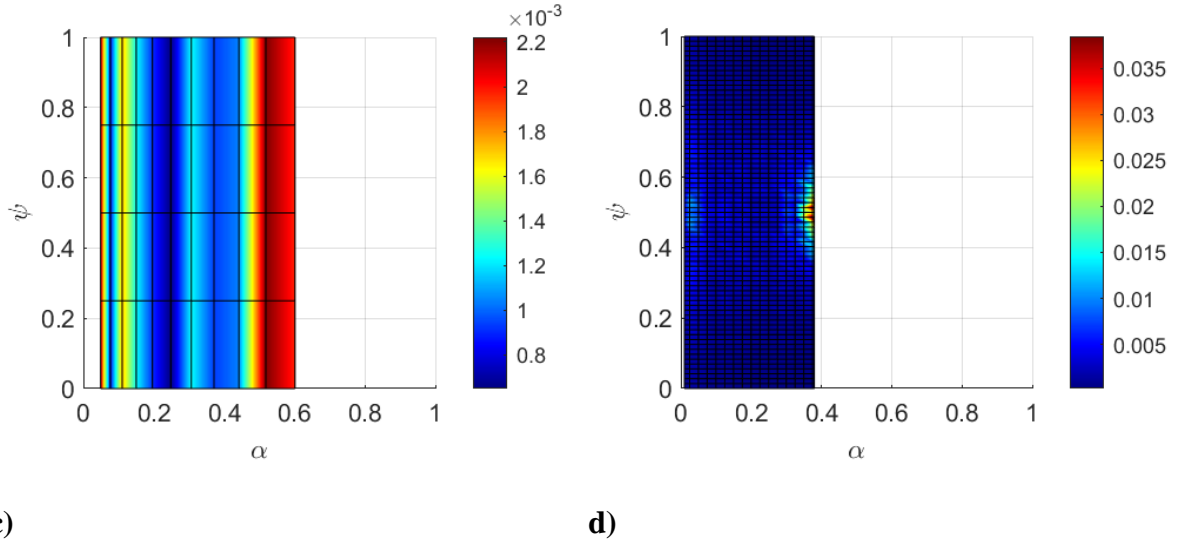


Figure 49. U_{num} estimates for k^* as a function of porosity and clocking angle for a) triangle, b) square, and c) ellipse pore geometries

Table 2. Regression coefficients

	Triangle	Square	Circle	Ellipse
$a_{c,1}$	-2.305720	-1.619511	-1.834454	-1.521327
$a_{c,2}$	1.197482	-0.075605	0.946001	1.482067
$a_{c,3}$	-2.305720	-2.086114	-1.834454	-2.413232
$a_{c,4}$	1.197482	1.518092	0.946001	0.275217
$a_{g,1}$	108.0	48.0	0.0	12.0
$a_{g,2}$	432.0	128.0	0.0	16.0

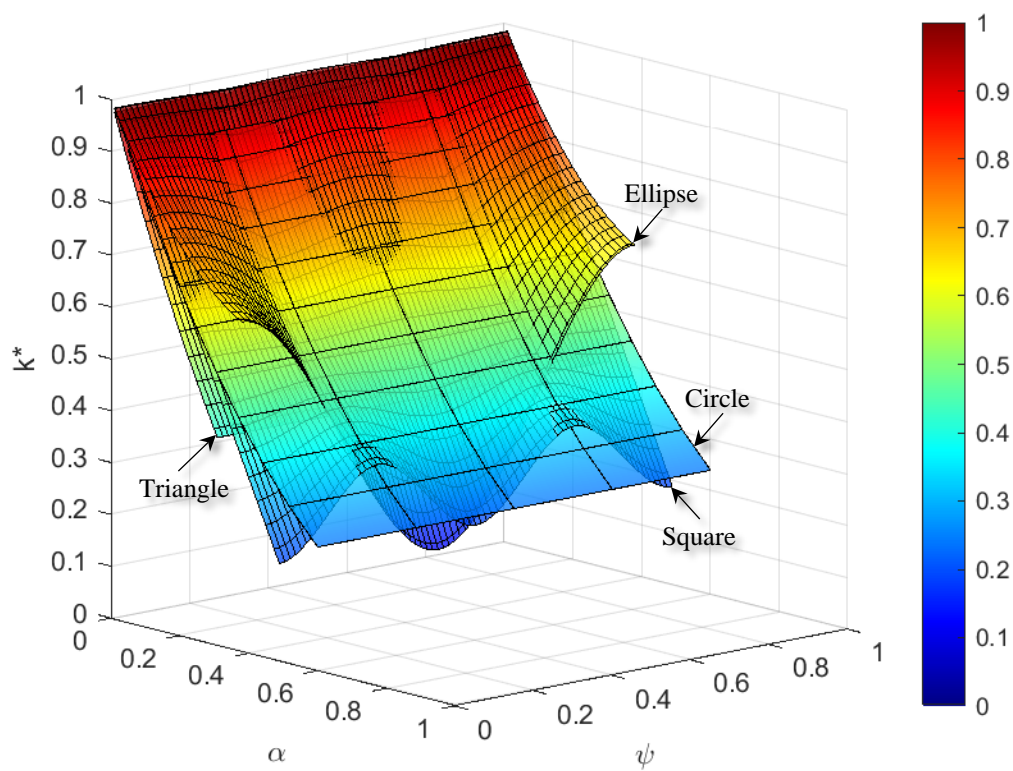
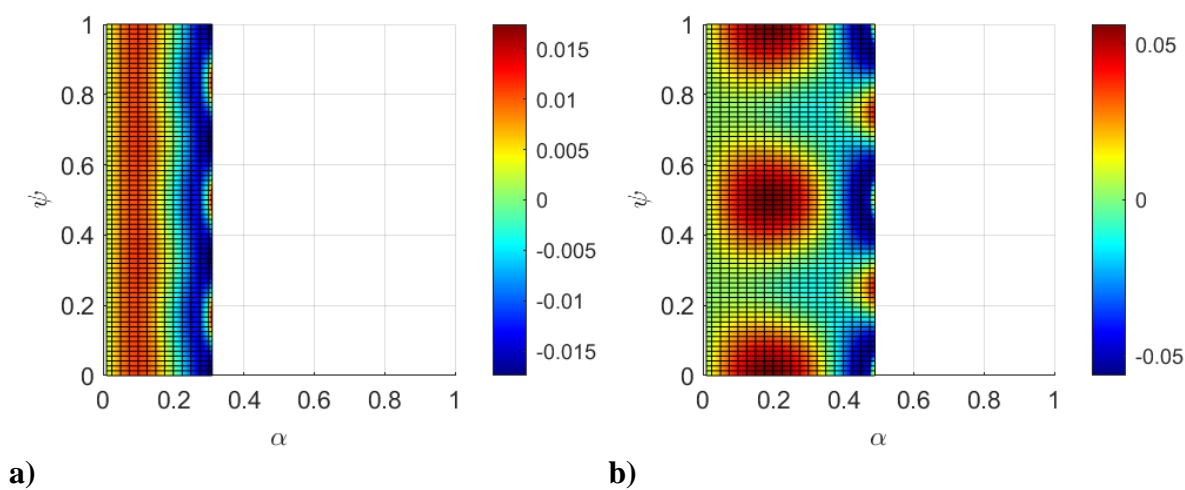


Figure 50. Regression k^* response as a function of porosity and clocking angle for triangle, square, circle, and ellipse pore geometries



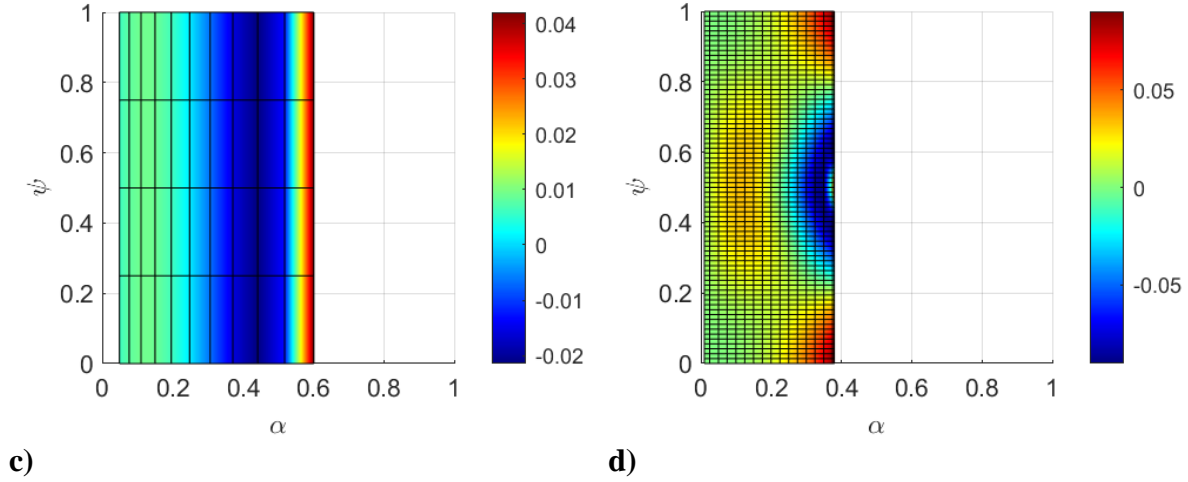


Figure 51. Relative difference in regression k^* from computational k^* with respect to porosity and clocking angle for a) triangle, b) square, and c) ellipse pore geometries

The resultant k^* values as presented here are only valid for system configurations that agree with those used in the system models. That is to say that the effective thermal conductivity is valid in the direction of heat flow and when periodic boundary conditions are assumed on boundaries perpendicular to the heat flow and in assumed two-dimensional systems.

In the literature, a plethora of empirical data revolving around porosity effects on thermal conductivity exists, but data appears sparse that compares directly to the system and analysis provided by this study. However, directly comparable empirical data presented by Wulf et al. measures both porosity, matrix, and effective thermal conductivity for FeCrAl alloy metal foam material using the transient plane source method [47]. The data presented by Wulf were extracted digitally using the WebPlotDigitizer tool [48] and are plotted against the computational results found in this work in Figure 52. The empirical material differs from this study's system in four distinct ways:

1. The empirical system was three-dimensional;
2. The empirical pore geometries were amorphous;
3. The empirical pores were not cubically arranged;
4. The empirical pores frequently contacted one another.

The computational results from Figure 48 of this study are also plotted on the two-dimensional set of axes in Figure 52, where the spread of data for each of the four pore geometry system sets is denoted by the family of solid curves of a single color. In addition, a single dashed curve is shown for each pore shape for a corresponding calibrated regression function trend with the same color as the computational data. Note that the porosity of the empirical FeCrAl alloy was outside of the range of geometries analyzed in this computational study, thus the regression was extended past the computational data regime. Likewise, with the system differences between the computational and empirical studies noted above, the regression functions defined in Equation (46) were calibrated to best agree with the empirical data using a simple least squares optimization, keeping the regression coefficients from Table 2. In Figure 52, it is clearly observed that the circle regression function (dashed green) exhibits obvious error with respect to the computational data (solid green) from which it is derived at around $\alpha=0.6$. The difference is less than 5% error but is exaggerated by the difference in slope at that location. With porosity known, the regression functions' calibration parameter was ψ_{cyc} . Because the triangle and circle regression functions do not change with ψ_{cyc} , only the square and ellipse regression functions were tuned. Note from Figure 52 that the triangle system reduces to $k^*=0$ at a porosity value lower than the empirical data, where $k^*<0$ is physically meaningless and impossible. Table 3 summarizes the errors of the regressions, ε_{k^*} , with respect

to the empirical data along with the calibrated ψ_{cyc} values. Despite the physical impossibility in the triangle regression values, errors are still reported. Errors in the tuned square and ellipse geometries are relatively low—between 2% and 15% error—where circle errors are significantly high.

These results show that not every geometry regression function can well describe empirical data, especially outside of the bounds from which the regression was derived, but the form and range of the regression functions may have the capability of accurately describing real-world k^* values with calibration. Likewise, the bounds of the regression functions given here may provide reasonable bounds of uncertainty in predicting FeCrAl alloy's effective thermal conductivity based solely on known material porosity.

Table 3. Calibrated extended regression comparison to empirical data

Geometry	Triangle	Square	Circle	Ellipse
ψ_{cyc}	---	0.1463	---	0.2339
α	εk^*			
0.850	-0.1314	0.0049	0.0875	0.0045
0.870	-0.1321	0.0026	0.0876	0.0024
0.880	-0.1346	-0.0008	0.0854	-0.0008
0.900	-0.1339	-0.0020	0.0866	-0.0017
0.890	-0.1369	-0.0040	0.0834	-0.0039

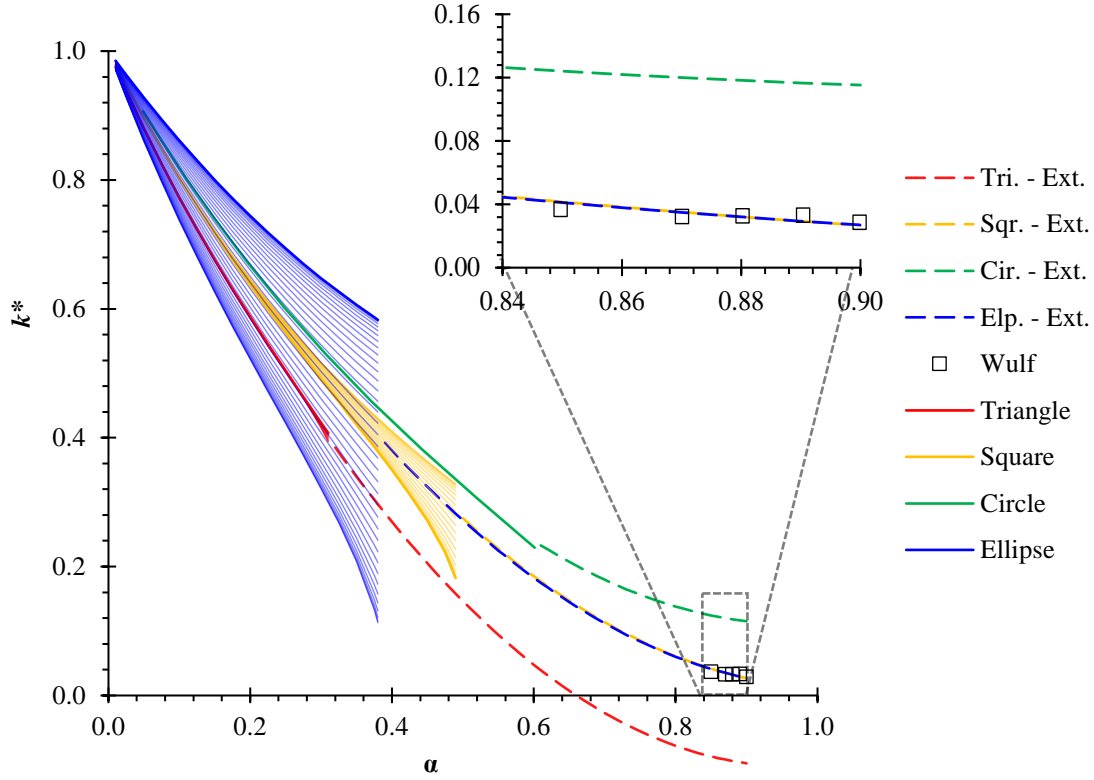


Figure 52. Comparison of computational analysis and regression results with empirical data for FeCrAl alloy material

6 Conclusion and Future Work

The numerical analyses performed in this work have covered a range of two-dimensional heterogeneous, porous material structure geometries, including triangle, square, circle, and ellipse geometries. This paper has presented a numerical approach to characterizing the effective thermal response of two-dimensional heterogeneous structures with respect to material porosity, pore geometry, and pore orientation. Numerical results have been presented for the effective thermal transport response of the porous media.

The method of manufactured solutions was used to perform code verification and showed that an approximately second order accurate numerical method was implemented. The global deviation grid convergence index was used to approximate numerical uncertainty on

effective thermal conductivity of the porous FeCrAl domain using systematic mesh refinement. Numerical uncertainty was estimated to be less than 6.0% for triangle and square geometries (with some exceptions), less than 0.2% for circle geometries, and less than 4.0% for ellipse geometries. A functional regression form was presented to map pore size and orientation to effective thermal conductivity for triangular, square, circular, and elliptical pores to within less than 2.0%, 6.0%, 5.0%, and 10.0% difference, respectively, from the numerical results.

A brief comparison was made between the numerically derived regression functions and empirically measured effective thermal conductivity of porous FeCrAl alloy material. Although the comparison was made outside of the original porosity regime corresponding to the numerical results, a single parameter calibration of the square and ellipse geometry regression functions yielded agreement with the highly porous empirical data to within 14% error. The good agreement on these porosities suggests that a spread of numerically based regression functions could provide reasonable uncertainty bounds in predicting the overall effective thermal conductivity of porous FeCrAl alloy.

Future work should incorporate numerical uncertainty with the effective thermal conductivity correlation to give a prescribed level of overall uncertainty for the correlation. Further dimensionality could be added to the effective thermal conductivity correlation by investigating the effects internal angle ratios in the polygonal pores and the effects of aspect ratio in elliptical pores.

References

- 1 Gu, S., Lu, T. J., Hass, D. D., and Wadley, H. N. G., 2001, "Thermal conductivity of zirconia coatings with zig-zag pore microstructures," *Acta Materialia*, **49**(13), pp. 2539-2547.

- 2 Flores Renteria, A., Saruhan, B., Schulz, U., Raetzer-Scheibe, H.-J., Haug, J., and Wiedenmann, A., 2006, "Effect of morphology on thermal conductivity of EB-PVD PYSZ TBCs," *Surface and Coatings Technology*, **201**(6), pp. 2611-2620.
- 3 Zhu, W., Cai, X. N., Yang, L., Xia, J., Zhou, Y. C., and Pi, Z. P., 2019, "The evolution of pores in thermal barrier coatings under volcanic ash corrosion using X-ray computed tomography," *Surface and Coatings Technology*, **357**, pp. 372-378.
- 4 Irick, K. W. and Fathi, N., 2019, "Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis," *Proceedings of the ASME 2019 Verification and Validation Symposium*, V001T12A002.
- 5 Zhao, Z., Shang, H., Zou, G., Ren, H., Zhuang, W., Liu, L., and Zhou, Y. N., 2019, "A Predictive Model for Thermal Conductivity of Nano-Ag Sintered Interconnect for a SiC Die," *Journal of Electronic Materials*, **48**, pp. 2811-2825.
- 6 Dinesh, B. V. S. and Bhattacharya, A., 2019, "Effect of foam geometry on heat absorption characteristics of PCM-metal foam composite thermal energy storage systems," *International Journal of Heat and Mass Transfer*, **134**, pp. 866-883.
- 7 Rechard, R. P., Hadgu, T., Wang, Y., Sanchez, L. C., McDaniel, P., Skinner, C., and Fathi, N., 2017, "Technical Feasibility of Direct Disposal of Electrefiner Salt Waste," SAND2017-10554, Sandia National Laboratories (SNL-NM), Albuquerque, NM, USA.
- 8 Glass, D. E., Dirling, R., Croop, H., Fry, T. J., and Frank, G. J., 2006, "Materials Development for Hypersonic Flight Vehicles," *Proceedings of the 14th AIAA/AHI Space Planes and Hypersonic Systems and Technologies Conference*, AIAA 2006-8122.
- 9 Tjong, W. C., 2007, "High Temperature Materials for Hypersonic Flight Vehicles," *Journal of The University of New South Wales at the Australian Defence Force Academy*, **1**(1).
- 10 Kasen, S. D., 2013, "Thermal Management at Hypersonic Leading Edges," Dissertation, University of Virginia.
- 11 Finsterle, S., Muller, R. A., Baltzer, R., Payer, J., and Rector, J. W., 2019, "Numerical Evaluation of Thermal Effects from Nuclear Waste Disposed in Horizontal Drillholes," *International High-Level Radioactive Waste Management Conference*.
- 12 Woodside, W., and Messmer, J. H., 1961, "Thermal Conductivity of Porous Media. I. Unconsolidated Sands," *Journal of Applied Physics*, **32**(9), pp. 1688-1699.
- 13 Woodside, W., and Messmer, J. H., 1961, "Thermal Conductivity of Porous Media. II. Consolidated Rocks," *Journal of Applied Physics*, **32**(9), pp. 1699-1706.
- 14 Kiyohashi, H., Sasaki, S., and Masuda, H., 2003, "Effective Thermal Conductivity of Silica Sand as a Filling Material for Crevices," *High-Temperatures-High Pressures*, **35**, pp. 179-192.
- 15 Wallen, B. M., Smits, K. M., Sakaki, T., Howington, S. E., T.K.K., C. D., 2016, "Thermal Conductivity of Binary Sand Mixtures Evaluated through Full Water Content Range," *Soil Science Society of America Journal*, **80**, pp.592 -603.

- 16 Gobetz, Z., Rowen, A. Dickman, C., Meinert, K., and Martukanitz, R., 2016, "Utilization of Additive Manufacturing for Aerospace Exchangers," Office of Naval Research, Final Report.
- 17 Zhang, S., Lane, B., Whiting, J., and Chou, K., 2018, "An Investigation into Metallic Powder Thermal Conductivity in Laser Powder Bed Fusion Additive Manufacturing," *Proceedings of the 19th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference*, pp. 1796-1807.
- 18 Ibrahim, Y., Elkholy, A., Schofield, J. S., Melenka, G. W., and Kempers, R., 2020, "Effective Thermal Conductivity of 3D-printed Continuous Fiber Polymer Composites," *Advance Manufacturing: Polymer & Composites Science*, **6**(1), pp. 17-28.
- 19 Danes, F., Garnier, B., and Dupuis, T., 2003, "Predicting, Measuring, and Tailoring the Transverse Thermal Conductivity of Composites from Polymer Matrix and Metal Filler," *International Journal of Thermophysics*, **24**(3), pp. 727-734.
- 20 Zhao, J.-J., Duan, Y.-Y., Wang X.-D., 2012, "Experimental and analytical analyses of the thermal conductivities and high-temperature characteristics of silica aerogels based on microstructures," *Journal of Physics D: Applied Physics*, **46**, 015304.
- 21 Irick, K. W., 2017, "Effective Thermal Resistance Comparison of Aerogel and Multi-Layer Insulation as Radiative Barriers Using the Single-Sided Guarded Hot Plate Method," *Frontiers in Heat and Mass Transfer*, **8**(2).
- 22 Wu, S., Li, T., Tong, Z., Chao, J., Zhai, T., Xu, J., Yan, T., Wu, M., Xu, Z., Bao, H., Deng, T., and Wang, R., 2019, "High-Performance Thermally Conductive Phase Change Composites by Large-Size Oriented Graphite Sheets for Scalable Thermal Energy Harvesting." *Advanced Materials*, **31**(49), 1905099.
- 23 Pabst, W., Gregorová, Sedlářová, I., and Černý, M., 2011, "Preparation and characterization of porous alumina-zirconia composite ceramics," *Journal of the European Ceramic Society*, **31**(14), pp. 2721-2731.
- 24 Mirabolghasemi, A., Akbarzadeh, A. H., Rodrige, D., and Therriault, D., 2019, "Thermal conductivity of architected cellular metamaterials," *Acta Materialia*, **174**, pp. 61-80.
- 25 Zohuri, Bahman, and Nima Fathi. *Thermal-hydraulic analysis of nuclear reactors*. New York: Springer, 2015.
- 26 World Nuclear Association, 2020, "Nuclear Fuel and its Fabrication," <https://www.world-nuclear.org/information-library/nuclear-fuel-cycle/conversion-enrichment-and-fabrication/fuel-fabrication.aspx>, accessed 18 May 2020.
- 27 Pramanik, D., Hemantha Rao, G. V. S., and Jayaraj, R. N., 2009, "Innovative Process Techniques to Optimize Quality and Microstructure of UO₂ Fuel for PHWRs in India," *Proceedings of International Atomic Energy Agency Technical Meeting on Advanced Fuel Pellet Materials and Fuel Rod Designs for Water Cooled Reactors*, PSI, Villigen, Switzerland, November 2009.

- 28 Park, D. J., Kim, H. G., Jung, Y. I., Park, J. H., Yang, J. H., and Koo, Y. H., 2016, "Behavior of an improved Zr fuel cladding with oxidation resistant coating under loss-of-coolant accident conditions," *Journal of Nuclear Materials*, (482), pp. 75-82.
- 29 Qiu, B., Wang, J., Deng, Y., Wang, M., Wu, Y., and Qiu, S.Z., 2020, "A review on thermohydraulic and mechanical-physical properties of SiC, FeCrAl and Ti₃SiC₂ for ATF cladding," *Nuclear Engineering and Technology*, (52), pp. 1-13.
- 30 Idaho National Laboratory, "BISON," INL-MIS-18-50577-Rev003, 18 May 2020.
- 31 Sandvik Group, 2018, "Kanthal APMT Tube Datasheet".
- 32 MatWeb, 2020, "Schwarzkopf Plansee PM 2000, Sheet Grain Class 6 ODS Iron Alloy Sheet," <http://www.matweb.com/search/datasheet.aspx?matguid=21e9ec9a0de24b47bcf69ab11c375567>, accessed 7 Aug 2020.
- 33 Special Metals Corporation, 2004, "INCOLOY alloy MA956," SMC-008.
- 34 Kim, D. H., Yu, B. Y., Cha, P. R., Yoon, W. Y., Byun, J. Y., and Kim, S. H., 2012, "A study on FeCrAl foam as effective catalyst support under thermal and mechanical stresses," *Surface and Coatings Technology*, (209), pp. 169-176.
- 35 Kämpf, H., and G. Karsten. "Effects of different types of void volumes on the radial temperature distribution of fuel pins." *Nuclear Applications and Technology* 9, no. 3 (1970): 288-300.
- 36 Ibrahim, T. K. and Rahman, M. M., 2013, "Study on effective parameter of the triple-pressure reheat combined cycle performance," *Thermal Science*, **17**(2), pp. 497-508.
- 37 Fathi, N., McDaniel, P., Forsberg, C., and de Oliveira, C., 2018, "Power Cycle Assessment of Nuclear Systems, Providing Energy Storage for Low Carbon Grids," *Journal of Nuclear Engineering and Radiation Science*, **4**(2), 020911.
- 38 Fathi, N., McDaniel, P., Forsberg, C., and de Oliveira, C., 2016, "Nuclear Systems for a Low Carbon Electrical Grid," *In 2016 24th International Conference on Nuclear Engineering*, pp. V001T03A007-V001T03A007.
- 39 Irick, K. W. and Fathi, N., 2018, "Thermal Response of Open-Cell Porous Materials: A Numerical Study and Model Assessment," *Proceedings of the ASME 2018 Verification and Validation Symposium*, V001T03A002.
- 40 Rechard, R. P., Hadgu, T., Wang, Y., Sanchez, L. C., McDaniel, P., Skinner, C., and Fathi, N., 2017, "Technical Feasibility of Direct Disposal of Electrefiner Salt Waste," *SAND2017-10554*, Sandia National Laboratories (SNL-NM), Albuquerque, NM, USA.
- 41 Geuzaine, C. and Remacle, J.-F., 2009, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering* **79**(11), pp. 1309-1331.
- 42 Patankar, S. V., 1980, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publishing Corporation.

- 43 Oberkampf, W. L. and Roy, C. J., 2010, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge, UK.
- 44 Roache, P. J., 2009, *Fundamentals of Verification and Validation*, Hermosa Publishers, Socorro, NM.
- 45 Phillips, T. S. and Roy, C. J., 2016 “A New Extrapolation-Based Uncertainty Estimator for Computational Fluid Dynamics,” *Journal of Verification, Validation and Uncertainty Quantification*, **1**(4), pp. 041006-1, 041006-13.
- 46 Celik, I. B., Ghia, U., Roache, P. J., Freitas, C. J., Coleman, H., and Raad, P. E., 2008, "Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications,” *Journal of Fluids Engineering*, **130**(7), pp. 0780011-0780014.
- 47 Wulf, R., Mendes, M. A.A., Skibina, V., Al-Zoubi, A., Trimis, D., Ray, S., and Gross, U., 2014, “Experimental and numerical determination of effective thermal conductivity of open cell FeCrAl-alloy metal foams,” *International Journal of Thermal Sciences*, (86), pp. 95-103.
- 48 Rohatgi, A., 2020, “WebPlotDigitizer,” Version 4.3, <https://automeris.io/WebPlotDigitizer>.

CLOSING REMARKS

This dissertation has provided an in-depth analysis of fundamental heterogeneous material effective thermal responses by way of high-fidelity computational analyses and data reduction. Regression functions have been developed to correlate characteristics of material system heterogeneity with effective system thermal responses. Both porous and two-material system have been analyzed for a variety of geometries and configurations. The results given should provide cheap, computationally-founded surrogate models and behavioral trends for predicting certain heterogeneous system responses. Users may find that the accuracy of the absolute response of the systems as determined by the computational data is not adequate for reliable predictions, but such a determination does not preclude the use of these results for understanding heterogeneous system sensitivities and expectant behavioral trends. The rigor of code and solution verification used for the presented analyses lends credence to the veracity of the computational data. Such a framework for verification can be used in analyses for different geometries and system configurations.

Despite the seemingly simple system configurations and physics under investigation, the level of complexity and level of rigor used to perform verification on the analysis code and resultant solutions is non-trivial. Although formal verification and validation processes and methodologies do not inherently imply that all analyses require extensive investigation and processing, they do assert that no computationally-obtained results should be accepted blindly or without formal scrutiny. The analyses and evaluations performed within this body of research highlight possible paths that can be taken to generate credibility evidence and quantifiable justification for claims made about computational data.

In harmony with the emphasis placed on technical transparency, communication, documentation, and scrutiny, multiple appendices have been provided to share the means by which the results presented in the preceding chapters were obtained. Specifically, the following items are included:

1. **Appendix A: Galerkin Finite Element Method Description.** Mathematical derivation of the solution method for solving the heat equation.
2. **Appendix B: Solver2D Source Code.** Fortran source code for the program used to generate computational results.
3. **Appendix C: Gmsh Input Files.** Mesh generation scripts used to build the computational domains.
4. **Appendix D: SolutionPlot.** Post-processing script for extracting solution data and presenting data visualization.
5. **Appendix E: Solver2D User Manual.** Guide to using the analysis tools.

Additionally, **Appendix F: Awards and Honors** enumerates academic awards and honors granted Kevin Irick during his pursuit of his Ph.D.

APPENDIX A: GALERKIN FINITE ELEMENT METHOD DESCRIPTION

The Galerkin finite element method used for the computational analyses in this work is a means of discretizing a governing partial differential equation. In this case, the steady-state heat equation was solved. The following sections detail the derivation of equations used in the numerical method, as implemented in the code used for this research.

Generalized Equation

The governing two-dimensional steady-state heat equation looks like

$$\frac{\partial}{\partial x} k_x \frac{\partial T}{\partial x} + \frac{\partial}{\partial y} k_y \frac{\partial T}{\partial y} + Q = 0, \quad (1)$$

where k_x and k_y are the thermal conductivities in the x -direction and y -direction, respectively, T is the temperature field variable, and Q is the energy per unit volume source term. Multiplying Equation (1) by an arbitrary weighting function, ϕ , and integrating over the domain Ω gives

$$\delta \int_{\Omega} \phi \left(\frac{\partial}{\partial x} k_x \frac{\partial T}{\partial x} + \frac{\partial}{\partial y} k_y \frac{\partial T}{\partial y} + Q = 0 \right) d\Omega. \quad (2)$$

The expanded form of Equation (2) looks like

$$\int_{\Omega} \phi \left(\frac{\partial}{\partial x} k_x \frac{\partial T}{\partial x} \right) d\Omega + \int_{\Omega} \phi \left(\frac{\partial}{\partial y} k_y \frac{\partial T}{\partial y} \right) d\Omega + \int_{\Omega} \phi Q d\Omega = 0. \quad (3)$$

Using integration by parts, the first two integrals from Equation (3) can be changed such that

$$\begin{aligned} \int_{\Omega} \phi \left(\frac{\partial}{\partial x} k_x \frac{\partial T}{\partial x} \right) d\Omega &= \int_{\Gamma} \phi \left(k_x \frac{\partial T}{\partial x} \right)_n d\Gamma - \int_{\Omega} k_x \frac{\partial T}{\partial x} \left(\frac{\partial \phi}{\partial x} \right) d\Omega \\ \int_{\Omega} \phi \left(\frac{\partial}{\partial y} k_y \frac{\partial T}{\partial y} \right) d\Omega &= \int_{\Gamma} \phi \left(k_y \frac{\partial T}{\partial y} \right)_n d\Gamma - \int_{\Omega} k_y \frac{\partial T}{\partial y} \left(\frac{\partial \phi}{\partial y} \right) d\Omega \end{aligned}, \quad (4)$$

where Γ is the boundary of Ω , and n denotes the direction normal to Γ (directed outward from Ω). Equation (3) then looks like

$$\int_{\Gamma} \phi \left(k_x \frac{\partial T}{\partial x} \right)_n d\Gamma - \int_{\Omega} k_x \frac{\partial T}{\partial x} \left(\frac{\partial \phi}{\partial x} \right) d\Omega + \int_{\Gamma} \phi \left(k_y \frac{\partial T}{\partial y} \right)_n d\Gamma - \int_{\Omega} k_y \frac{\partial T}{\partial y} \left(\frac{\partial \phi}{\partial y} \right) d\Omega + \int_{\Omega} \phi Q d\Omega = 0. \quad (5)$$

Combining the boundary integrals in Equation (5) reduces it to

$$\int_{\Gamma} \phi \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y} \right)_n d\Gamma - \int_{\Omega} k_x \frac{\partial T}{\partial x} \left(\frac{\partial \phi}{\partial x} \right) d\Omega - \int_{\Omega} k_y \frac{\partial T}{\partial y} \left(\frac{\partial \phi}{\partial y} \right) d\Omega + \int_{\Omega} \phi Q d\Omega = 0. \quad (6)$$

The normal component of the integrand in the first term in Equation (6) is the opposite of definition for the normal heat flux, q_n , at the boundary such that

$$q_n \equiv - \left(k_x \frac{\partial T}{\partial x} + k_y \frac{\partial T}{\partial y} \right)_n. \quad (7)$$

Thus, substitution of Equation (7) into Equation (6) gives

$$- \int_{\Gamma} \phi q_n d\Gamma - \int_{\Omega} k_x \frac{\partial T}{\partial x} \left(\frac{\partial \phi}{\partial x} \right) d\Omega - \int_{\Omega} k_y \frac{\partial T}{\partial y} \left(\frac{\partial \phi}{\partial y} \right) d\Omega + \int_{\Omega} \phi Q d\Omega = 0. \quad (8)$$

Rearrangement of Equation (8) (and dropping parentheses) for convenience yields

$$\int_{\Omega} \frac{\partial \phi}{\partial x} k_x \frac{\partial T}{\partial x} d\Omega + \int_{\Omega} \frac{\partial \phi}{\partial y} k_y \frac{\partial T}{\partial y} d\Omega = \int_{\Omega} \phi Q d\Omega - \int_{\Gamma} \phi q_n d\Gamma. \quad (9)$$

In practical terms, Equation (9) is essentially stating that

$$\text{Net energy change} = \text{Internal heat generation} - \text{Net heat leaving the boundary}$$

This energy balance holds for all Ω with corresponding boundaries, as in each finite element of the discretized domain.

Now, it is assumed that within each Ω (or finite element), the field variable T and the weighting function ϕ are interpolated by smooth functions based on their values at the finite element's vertices. The Galerkin finite element formulation prescribes that both T and ϕ be interpolated using the same function (shape function), denoted here as a vector function N , such that

$$\begin{aligned} T &= T(x, y) = \sum_i^{N_v} N_i(x, y) T_i \\ \phi &= \phi(x, y) = \sum_i^{N_v} N_i(x, y) \phi_i \end{aligned} \quad (10)$$

where T_i , ϕ_i , and N_i are the temperature, weight value, and shape function value at the i^{th} vertex, and N_v is the total number of vertices defining the finite element. The T_i , ϕ_i , and N_i values are elements of vectors \mathbf{T} , $\boldsymbol{\phi}$, and \mathbf{N} , respectively, where

$$\mathbf{T} = \begin{Bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{N_v} \end{Bmatrix}; \quad \boldsymbol{\phi} = \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N_v} \end{Bmatrix}; \quad \text{and} \quad \mathbf{N} = [N_1 \quad N_2 \quad \cdots \quad N_{N_v}] \quad (11)$$

Using these definitions, Equation (10) becomes

$$\begin{aligned} T &= \mathbf{N} \mathbf{T} \\ \phi &= \mathbf{N} \boldsymbol{\phi} \end{aligned} \quad (12)$$

Substitution of Equation (12) into Equation (9) gives the following

$$\int_{\Omega} \frac{\partial \mathbf{N}\boldsymbol{\phi}}{\partial x} k_x \frac{\partial \mathbf{N}\mathbf{T}}{\partial x} d\Omega + \int_{\Omega} \frac{\partial \mathbf{N}\boldsymbol{\phi}}{\partial y} k_y \frac{\partial \mathbf{N}\mathbf{T}}{\partial y} d\Omega = \int_{\Omega} \mathbf{N}\boldsymbol{\phi} Q d\Omega - \int_{\Gamma} \mathbf{N}\boldsymbol{\phi} q_n d\Gamma. \quad (13)$$

A transpose matrix identity becomes useful, where

$$\mathbf{N}\boldsymbol{\phi} = \boldsymbol{\phi} = \boldsymbol{\phi}^T = (\mathbf{N}\boldsymbol{\phi})^T = \boldsymbol{\phi}^T \mathbf{N}^T. \quad (14)$$

Substituting this expression into Equation (13) yields

$$\int_{\Omega} \frac{\partial \boldsymbol{\phi}^T \mathbf{N}^T}{\partial x} k_x \frac{\partial \mathbf{N}\mathbf{T}}{\partial x} d\Omega + \int_{\Omega} \frac{\partial \boldsymbol{\phi}^T \mathbf{N}^T}{\partial y} k_y \frac{\partial \mathbf{N}\mathbf{T}}{\partial y} d\Omega = \int_{\Omega} \boldsymbol{\phi}^T \mathbf{N}^T Q d\Omega - \int_{\Gamma} \boldsymbol{\phi}^T \mathbf{N}^T q_n d\Gamma. \quad (15)$$

Because \mathbf{T} and $\boldsymbol{\phi}$ are constant for a given finite element, they can be pulled out of derivatives and integrals, such that Equation (15) looks like

$$\begin{aligned} \boldsymbol{\phi}^T \int_{\Omega} \frac{\partial \mathbf{N}^T}{\partial x} k_x \frac{\partial \mathbf{N}}{\partial x} d\Omega \mathbf{T} + \boldsymbol{\phi}^T \int_{\Omega} \frac{\partial \mathbf{N}^T}{\partial y} k_y \frac{\partial \mathbf{N}}{\partial y} d\Omega \mathbf{T} = \\ \boldsymbol{\phi}^T \int_{\Omega} \mathbf{N}^T Q d\Omega - \boldsymbol{\phi}^T \int_{\Gamma} \mathbf{N}^T q_n d\Gamma \end{aligned} \quad (16)$$

With $\boldsymbol{\phi}^T$ common to all terms in Equation (16), the equation simplifies to the general two-dimensional finite element discretized governing steady-state heat equation.

$$\int_{\Omega} \frac{\partial \mathbf{N}^T}{\partial x} k_x \frac{\partial \mathbf{N}}{\partial x} d\Omega \mathbf{T} + \int_{\Omega} \frac{\partial \mathbf{N}^T}{\partial y} k_y \frac{\partial \mathbf{N}}{\partial y} d\Omega \mathbf{T} = \int_{\Omega} \mathbf{N}^T Q d\Omega - \int_{\Gamma} \mathbf{N}^T q_n d\Gamma. \quad (17)$$

The single integral with respect to Ω represents a double integral with respect to area A , where

$$d\Omega = dA = dx dy. \quad (18)$$

such that

$$\int_A \frac{\partial \mathbf{N}^T}{\partial x} k_x \frac{\partial \mathbf{N}}{\partial x} dA \mathbf{T} + \int_A \frac{\partial \mathbf{N}^T}{\partial y} k_y \frac{\partial \mathbf{N}}{\partial y} dA \mathbf{T} = \int_A \mathbf{N}^T Q dA - \int_\Gamma \mathbf{N}^T q_n d\Gamma. \quad (19)$$

The matrix integrals corresponding the x - and y -derivatives can be defined such that

$$\begin{aligned} \mathbf{G}_x &= \int_A \frac{\partial \mathbf{N}^T}{\partial x} k_x \frac{\partial \mathbf{N}}{\partial x} dA \\ \mathbf{G}_y &= \int_A \frac{\partial \mathbf{N}^T}{\partial y} k_y \frac{\partial \mathbf{N}}{\partial y} dA \end{aligned} \quad (20)$$

where

$$\mathbf{G} = \mathbf{G}_x + \mathbf{G}_y. \quad (21)$$

Likewise, the entire right-hand side of Equation (19) can be defined as the matrix \mathbf{Q} . Thus, it follows that in the reduced matrix form, Equation (19) becomes

$$\mathbf{GT} = \mathbf{Q}. \quad (22)$$

Linear Triangle Element

For a linear triangle finite element formulation, it is convenient to recast the original cartesian coordinates in terms of natural coordinates (or area coordinates), ξ_1 , ξ_2 , and ξ_3 , where

$$\begin{aligned} x &= x_1 \xi_1 + x_2 \xi_2 + x_3 \xi_3 \\ y &= y_1 \xi_1 + y_2 \xi_2 + y_3 \xi_3 \end{aligned} \quad (23)$$

and

$$\xi_1 + \xi_2 + \xi_3 = 1. \quad (24)$$

Letting the coordinate transformation matrix, \mathbf{J} , (also known as the Jacobian matrix) be defined as

$$\mathbf{J} = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}, \quad (25)$$

it follows that

$$\begin{Bmatrix} 1 \\ x \\ y \end{Bmatrix} = \mathbf{J} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix}, \quad (26)$$

and

$$\mathbf{J}^{-1} = \frac{1}{2A_t} \begin{bmatrix} x_2 y_3 - x_3 y_2 & (y_2 - y_3) & (x_3 - x_2) \\ x_3 y_1 - x_1 y_3 & (y_3 - y_1) & (x_1 - x_3) \\ x_1 y_2 - x_2 y_1 & (y_1 - y_2) & (x_2 - x_1) \end{bmatrix}, \quad (27)$$

where the signed triangle area, A_t , is defined as

$$2A_t = |\mathbf{J}| = (x_2 - x_1)(y_3 - y_1) - (x_1 - x_3)(y_1 - y_2), \quad (28)$$

and

$$\begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} = \mathbf{J}^{-1} \begin{Bmatrix} 1 \\ x \\ y \end{Bmatrix}. \quad (29)$$

If \mathbf{N} is expressed in natural coordinates, then

$$\mathbf{N} = \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix}^T = \frac{1}{2A_t} \begin{Bmatrix} x_2 y_3 - x_3 y_2 + x(y_2 - y_3) + y(x_3 - x_2) \\ x_3 y_1 - x_1 y_3 + x(y_3 - y_1) + y(x_1 - x_3) \\ x_1 y_2 - x_2 y_1 + x(y_1 - y_2) + y(x_2 - x_1) \end{Bmatrix}^T. \quad (30)$$

The partial differentials found in Equation (19) are determined using the chain rule such that

$$\begin{aligned}\partial \mathbf{N} / \partial x &= (\partial \mathbf{N} / \partial \xi_1)(\partial \xi_1 / \partial x) + (\partial \mathbf{N} / \partial \xi_2)(\partial \xi_2 / \partial x) + (\partial \mathbf{N} / \partial \xi_3)(\partial \xi_3 / \partial x) \\ \partial \mathbf{N} / \partial y &= (\partial \mathbf{N} / \partial \xi_1)(\partial \xi_1 / \partial y) + (\partial \mathbf{N} / \partial \xi_2)(\partial \xi_2 / \partial y) + (\partial \mathbf{N} / \partial \xi_3)(\partial \xi_3 / \partial y)\end{aligned}\quad (31)$$

From Equation (29), it can be shown that

$$\begin{aligned}\partial \xi_1 / \partial x &= (y_2 - y_3) / 2A_t \\ \partial \xi_2 / \partial x &= (y_3 - y_1) / 2A_t \\ \partial \xi_3 / \partial x &= (y_1 - y_2) / 2A_t \\ \partial \xi_1 / \partial y &= (x_3 - x_2) / 2A_t \\ \partial \xi_2 / \partial y &= (x_1 - x_3) / 2A_t \\ \partial \xi_3 / \partial y &= (x_2 - x_1) / 2A_t\end{aligned}\quad (32)$$

From Equation (30) and Equation (32), the partial derivatives of the shape function come out to be

$$\begin{aligned}\partial \mathbf{N} / \partial x &= \frac{1}{2A_t} [(y_2 - y_3) \quad (y_3 - y_1) \quad (y_1 - y_2)] \\ \partial \mathbf{N} / \partial y &= \frac{1}{2A_t} [(x_3 - x_2) \quad (x_1 - x_3) \quad (x_2 - x_1)]\end{aligned}\quad (33)$$

In order to evaluate the integrals in Equation (19) using natural coordinates, use the fact that a polynomial function expressed in natural coordinates is integrated over the area by using the formula

$$\int_{A_t} \xi_1^k \xi_2^l \xi_3^m dA_t = 2A_t (k! l! m!) / (2 + k + l + m)!, \quad (34)$$

where k , l , and m are integer exponents to describe the polynomial.

Expressed in natural coordinates, Equation (19) solved for a triangle element becomes

$$\begin{aligned} \int_{A_t} \frac{1}{4A_t^2} \begin{Bmatrix} (y_2 - y_3) \\ (y_3 - y_1) \\ (y_1 - y_2) \end{Bmatrix} k_x \begin{Bmatrix} (y_2 - y_3) \\ (y_3 - y_1) \\ (y_1 - y_2) \end{Bmatrix}^T dA_t \mathbf{T} + \\ \int_{A_t} \frac{1}{4A_t^2} \begin{Bmatrix} (x_3 - x_2) \\ (x_1 - x_3) \\ (x_2 - x_1) \end{Bmatrix} k_y \begin{Bmatrix} (x_3 - x_2) \\ (x_1 - x_3) \\ (x_2 - x_1) \end{Bmatrix}^T dA_t \mathbf{T} = \int_{A_t} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} Q dA_t - \int_{\Gamma} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} q_n d\Gamma \end{aligned} \quad (35)$$

Noting the absence of natural coordinate terms on the left-hand side of Equation (35), the polynomial formula from Equation (34) looks like

$$\int_{A_t} \xi_1^0 \xi_2^0 \xi_3^0 dA_t = 2A_t(0!0!0!)/(2+0+0+0)! = A_t. \quad (36)$$

Thus, Equation (35) becomes

$$\begin{aligned} \frac{1}{4A_t} \begin{Bmatrix} (y_2 - y_3) \\ (y_3 - y_1) \\ (y_1 - y_2) \end{Bmatrix} k_x \begin{Bmatrix} (y_2 - y_3) \\ (y_3 - y_1) \\ (y_1 - y_2) \end{Bmatrix}^T \mathbf{T} + \\ \frac{1}{4A_t} \begin{Bmatrix} (x_3 - x_2) \\ (x_1 - x_3) \\ (x_2 - x_1) \end{Bmatrix} k_y \begin{Bmatrix} (x_3 - x_2) \\ (x_1 - x_3) \\ (x_2 - x_1) \end{Bmatrix}^T \mathbf{T} = \int_{A_t} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} Q dA_t - \int_{\Gamma} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} q_n d\Gamma \end{aligned} \quad (37)$$

Performing the matrix multiplication for the terms associated with \mathbf{T} from Equation (37) yields the directional conductance matrices, \mathbf{G}_x and \mathbf{G}_y , that take on the form

$$\begin{aligned}\mathbf{G}_x &= \frac{k_x}{4A_t} \begin{bmatrix} (y_2 - y_3)^2 & (y_2 - y_3)(y_3 - y_1) & (y_2 - y_3)(y_1 - y_2) \\ (y_3 - y_1)(y_2 - y_3) & (y_3 - y_1)^2 & (y_3 - y_1)(y_1 - y_2) \\ (y_1 - y_2)(y_2 - y_3) & (y_1 - y_2)(y_3 - y_1) & (y_1 - y_2)^2 \end{bmatrix} \\ \mathbf{G}_y &= \frac{k_y}{4A_t} \begin{bmatrix} (x_3 - x_2)^2 & (x_3 - x_2)(x_1 - x_3) & (x_3 - x_2)(x_2 - x_1) \\ (x_1 - x_3)(x_3 - x_2) & (x_1 - x_3)^2 & (x_1 - x_3)(x_2 - x_1) \\ (x_2 - x_1)(x_3 - x_2) & (x_2 - x_1)(x_1 - x_3) & (x_2 - x_1)^2 \end{bmatrix}.\end{aligned}\quad (38)$$

\mathbf{G}_x and \mathbf{G}_y can be combined to yield the conductance matrix, \mathbf{G} , as expressed in Equation (21).

Likewise, the integrals on the right-hand side of Equation (34) can be combined to give

$$\mathbf{Q} = \int_{A_t} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} Q dA_t - \int_{\Gamma} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} q_n d\Gamma. \quad (39)$$

Here, the integrals in \mathbf{Q} are left unsolved, as Q and q_n may take on a variety of forms.

APPENDIX B: SOLVER2D SOURCE CODE

Following is the Fortran source code for the Solver2D program used to perform the computational thermal analyses in this research. Although various versions of this code were used for the different analyses presented in the research, the code presented here is consistent with the most advanced functionality and debugged code used for any of the analyses. The code solves the two-dimensional steady-state heat equation with unstructured triangular meshes. Mesh inputs for the solver were generated using Gmsh⁷, an open-source freeware for mesh generation, and virtually any two-dimensional unstructured triangular mesh generated with Gmsh could be used as a computational domain in Solver2D. Solver2D allows for implementation of user-defined boundary conditions, source terms, and material thermal conductivities. The program also handles multi-material domains and heat load definitions by distinct mesh domains. One distinct feature available in Solver2D is the use of the method of manufactured solutions (MMS). The user can define the necessary MMS equations and can link boundary conditions and heat load definitions from the problem of interest to counterpart MMS equations for quick and consistent MMS simulations. Solver update method options include basic successive over-relaxation and conjugate gradient methods. The code provides easy implementation of multiple analyses in series—i.e., as a batch—with automatic solution output directory naming. Naturally, the following source code could be modified and customized, thus the features mentioned above are highlights of the capabilities currently implemented. For the analyses in this work, the Intel® Fortran compiler was used.

⁷ Geuzaine, C. and Remacle, J.-F., 2009, “Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities,” *International Journal for Numerical Methods in Engineering* **79**(11), pp. 1309-1331.

```

! KEVIN IRICK
! This source code was originally written in Programmer's Notepad, and formatting
! (i.e., tabs and spacing) may become distorted or misaligned if viewed in other
text
! editing environments.]

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
MODULES
module Types
! This module defines custom data types

type AllIntArr1D ! one-dimensional allocatable integer array
integer, allocatable, dimension(:) :: arr
end type AllIntArr1D

type AllRlArr1D ! one-dimensional allocatable real array
real(8), allocatable, dimension(:) :: arr
end type AllRlArr1D

type UserFuncNum ! user function numerics data structure
integer, allocatable, dimension(:) :: intr_ar
real(8), allocatable, dimension(:) :: real_ar
end type

end module Types

module Variables
! This module contains all developer-defined global variables.

use Types

! variables
integer, allocatable, dimension(:) :: bnd_ID ! boundary ID number
character(len=100), allocatable, dimension(:) :: bnd_name ! boundary name
integer, allocatable, dimension(:) :: bnd_type ! boundary type (-1=Intimate
Thermal Contact,1=Temperature/Dirichlet,2=Temperature Gradient/Neumann,3=Heat
Flux)
integer, allocatable, dimension(:) :: bnd_func ! function ID to define boundary
type(AllIntArr1D), allocatable, dimension(:) :: bnd_evrt ! list of edge vertices
on boundary
type(AllIntArr1D), allocatable, dimension(:) :: bnd_tvrt ! list of triangle
vertices on boundary
type(AllIntArr1D), allocatable, dimension(:) :: bnd_edg ! list of edges on
boundary
type(AllIntArr1D), allocatable, dimension(:) :: bnd_tri ! list of triangles on
boundary

integer, allocatable, dimension(:) :: cnt_ID ! list of continuum IDs
integer, allocatable, dimension(:) :: cnt_mat ! continuum material assignment
list
real(8), allocatable, dimension(:) :: cnt_area ! continuum area

```

```

type(AllIntArr1D), allocatable, dimension(:) :: cnt_tri ! list of triangles in
continuum

character(len=200) :: dir_slt ! current solution directory path
character(len=200) :: dir_slt_rt ! current solution directory path root folder
name

integer, allocatable, dimension(:) :: edg_ID ! edge ID numbers
integer, allocatable, dimension(:, :) :: edg_vrt ! edge vertices (v1,v2)
real(8), allocatable, dimension(:, :) :: edg_mdpt ! edge midpoints (x,y,z)
real(8), allocatable, dimension(:) :: edg_lng ! edge lengths
real(8), allocatable, dimension(:, :) :: edg_tngt ! edge direction unit vectors
(dx,dy,dz)
type(AllRlArr1D), allocatable, dimension(:) :: edg_nrmlx ! edge adjacent triangle
edge outward normal unit vectors x-component
type(AllRlArr1D), allocatable, dimension(:) :: edg_nrmlly ! edge adjacent triangle
edge outward normal unit vectors y-component
type(AllRlArr1D), allocatable, dimension(:) :: edg_nrmlz ! edge adjacent triangle
edge outward normal unit vectors z-component
type(AllIntArr1D), allocatable, dimension(:) :: edg_adjvrt ! edge adjacent
triangle opposing vertex IDs
type(AllIntArr1D), allocatable, dimension(:) :: edg_adjtri ! edge adjacent
triangle IDs
type(AllIntArr1D), allocatable, dimension(:) :: edg_adjtet ! edge adjacent
tetrahedron IDs
integer, allocatable, dimension(:) :: edg_bnd ! edge boundary ID association
type(AllIntArr1D), allocatable, dimension(:) :: edg_vox ! edge voxel IDs

integer :: FLAG_disp_inpt ! flag to display input file lines ( 0 = no; 1 = yes)
integer :: FLAG_MMS ! flag to perform MMS ( 0 = no; 1 = yes )
integer :: FLAG_UPDATE_METHOD ! flag for solution update method (1=SOR,
2=Conjugate Gradient)
character(len=100) :: fmtf ! format specification for real values
character(len=100) :: fmte ! format specification for scientific values

real(8), allocatable, dimension(:) :: Ge_x ! edge (vertex-to-vertex) conductance
(x-dimension)
real(8), allocatable, dimension(:) :: Ge_y ! edge (vertex-to-vertex) conductance
(y-dimension)
real(8), allocatable, dimension(:) :: Ge_z ! edge (vertex-to-vertex) conductance
(z-dimension)
real(8), allocatable, dimension(:) :: Gv_x ! vertex (same vertex) conductance (x-
dimension)
real(8), allocatable, dimension(:) :: Gv_y ! vertex (same vertex) conductance (y-
dimension)
real(8), allocatable, dimension(:) :: Gv_z ! vertex (same vertex) conductance (z-
dimension)

real(8) :: h ! overall characteristic mesh size
real(8) :: h1 ! 1-dimensional characteristic mesh size
real(8) :: h2 ! 2-dimensional characteristic mesh size
real(8) :: h3 ! 3-dimensional characteristic mesh size

integer :: i_cas ! index of case number

```

```

integer, allocatable, dimension(:) :: lod_ID ! load ID numbers
character(len=100), allocatable, dimension(:) :: lod_name ! load name
integer, allocatable, dimension(:) :: lod_type ! load type (-
1=unassigned,0=absolute,1=line,2=flux,3=volumetric)
integer, allocatable, dimension(:) :: lod_dist ! load spatial distribution type
(-1=unassigned,0=,1=uniform,2=spatial function)
integer, allocatable, dimension(:) :: lod_distfunc ! load spatial distribution
function ID
integer, allocatable, dimension(:) :: lod_func ! load function ID
integer, allocatable, dimension(:) :: lod_temp ! load temporal behavior type (-
1=unassigned,0=,1=constant,2=time-dependent,3=temperature-dependent,4=time- and
temperature-dependent)
integer, allocatable, dimension(:) :: lod_tempfunc ! load temporal behavior
function ID
type(AllIntArr1D), allocatable, dimension(:) :: lod_cnt ! load application
continuum IDs
type(AllIntArr1D), allocatable, dimension(:) :: lod_cntvrt ! load application
continuum vertex IDs
type(AllR1Arr1D), allocatable, dimension(:) :: lod_cvrtarea ! load application
continuum vertex area
type(AllR1Arr1D), allocatable, dimension(:) :: lod_cvrtlod ! load application
continuum vertex loads
type(AllIntArr1D), allocatable, dimension(:) :: lod_lin ! load application line
IDs
type(AllIntArr1D), allocatable, dimension(:) :: lod_linvrt ! load application
line vertex IDs
type(AllR1Arr1D), allocatable, dimension(:) :: lod_lvrtlod ! load application
line vertex loads

real(8) :: mfe_crt ! critical mfg error (i.e., mfg error used with some threshold
limit)
real(8) :: mfe_max ! maximum mfg error
real(8) :: mfe_min ! minimum mfg error
real(8) :: mfe_rss ! root sum of squares of mfg error
real(8) :: mfe_rms ! root mean squares of mfg error

character(len=500) :: MeshInputFileName ! file location of mesh input file
integer :: Nbar ! number of bars
integer :: Nbnd ! number of boundaries
integer :: Ncas ! number of cases
integer :: Ncnt ! number of continuums
integer :: Nedg ! number of edges
integer :: Nmat ! number of materials
integer :: Nlod ! number of loads
integer :: Ntet ! number of tetrahedra
integer :: Ntri ! number of triangles
integer :: Nvxx ! number of voxels in z-dimension
integer :: Nvxy ! number of voxels in y-dimension
integer :: Nvxz ! number of voxels in x-dimension
integer :: Nvrt ! number of vertices
integer :: Nel1 ! number of 1-dimensional elements
integer :: Nel2 ! number of 2-dimensional elements
integer :: Nel3 ! number of 3-dimensional elements

```



```

real(8), parameter :: pi = 3.14159265358979323846264338327950288419716939937510d0
! pi

character(len=500) pth_wrk_dir ! path of working directory

real(8), allocatable, dimension(:) :: pmult ! conjugate gradient change vector

integer, allocatable, dimension(:) :: tri_ID ! triangle ID numbers
integer, allocatable, dimension(:, :) :: tri_vrt ! triangle vertices (v1,v2,v3)
integer, allocatable, dimension(:, :) :: tri_edg ! triangle edges (e1,e2,e3)
real(8), allocatable, dimension(:) :: tri_area ! triangle area
real(8), allocatable, dimension(:, :) :: tri_cntr ! triangle centroid (x,y,z)
real(8), allocatable, dimension(:, :) :: tri_nrml ! triangle unit normal
(dx,dy,dz)
integer, allocatable, dimension(:) :: tri_cntm ! triangle continuum ID
integer, allocatable, dimension(:) :: tri_mat ! triangle material ID
real(8), allocatable, dimension(:) :: tri_thk ! triangle thickness
type(AllIntArr1D), allocatable, dimension(:) :: tri_vox ! triangle voxel IDs

integer, allocatable, dimension(:) :: vox_ID ! voxel ID
integer, allocatable, dimension(:, :) :: vox_crd ! voxel coordinates (in voxel
dimensions)
real(8), allocatable, dimension(:, :) :: vox_lim_lo ! voxel lower limits (x,y,z)
real(8), allocatable, dimension(:, :) :: vox_lim_hi ! voxel upper limits (x,y,z)
type(AllIntArr1D), allocatable, dimension(:) :: vox_vrt ! voxel contained vertex
IDs
type(AllIntArr1D), allocatable, dimension(:) :: vox_edg ! voxel contained edge
IDs
type(AllIntArr1D), allocatable, dimension(:) :: vox_tri ! voxel contained
triangle IDs

integer, allocatable, dimension(:) :: vrt_ID ! vertex ID number
real(8), allocatable, dimension(:, :) :: vrt_crd ! vertex coordinates (x,y,z)
type(AllIntArr1D), allocatable, dimension(:) :: vrt_adjvrt ! vertex adjacent
vertex IDs
type(AllIntArr1D), allocatable, dimension(:) :: vrt_adjedg ! vertex adjacent edge
IDs
type(AllIntArr1D), allocatable, dimension(:) :: vrt_adjtri ! vertex adjacent
triangle IDs
type(AllIntArr1D), allocatable, dimension(:) :: vrt_adjtet ! vertex adjacent
tetrahedron IDs
type(AllIntArr1D), allocatable, dimension(:) :: vrt_bnd ! vertex boundary ID
association
real(8), allocatable, dimension(:) :: vrt_area ! vertex surface area
real(8), allocatable, dimension(:) :: vrt_vol ! vertex volume
integer, allocatable, dimension(:) :: vrt_vox ! vertex voxel ID
real(8), allocatable, dimension(:) :: vrt_tmp ! vertex temperature
real(8), allocatable, dimension(:, :) :: vrt_flx ! vertex heat flux (qx,qy,qz)
real(8), allocatable, dimension(:, :) :: vrt_src ! (direct,radiative,boundary)
real(8), allocatable, dimension(:) :: vrt_res ! vertex energy residual
real(8), allocatable, dimension(:) :: vrt_tmppm ! vertex manufactured solution
temperature
real(8), allocatable, dimension(:) :: vrt_mfe ! vertex temperature error w.r.t
manufactured solution

```

```

integer, allocatable, dimension(:) :: mat_ID ! material ID number
character(len=100), allocatable, dimension(:) :: mat_name ! material name
real(8), allocatable, dimension(:) :: mat_k ! material thermal conductivity
real(8), allocatable, dimension(:) :: mat_rho ! material mass density
real(8), allocatable, dimension(:) :: mat_cp ! material specific heat
real(8), allocatable, dimension(:) :: mat_mu ! material kinematic viscosity
real(8), allocatable, dimension(:) :: mat_e ! material surface emissivity
real(8), allocatable, dimension(:) :: mat_r ! material surface reflectivity
real(8), allocatable, dimension(:) :: mat_t ! material surface transmissivity

real(8) :: relax ! temperature update relaxation coefficient

real(8) :: res_crt ! critical residual (i.e., residual used with cutoff limit)
real(8) :: res_crt0 ! critical residual (i.e., residual used with cutoff limit)
from previous iteration
real(8) :: res_max ! maximum residual
real(8) :: res_min ! minimum residual
real(8) :: res_rss ! root sum of squares of residuals
real(8) :: res_rms ! root mean squares of residuals
real(8) :: res_lim ! residual cutoff limit

type(AllIntArr1D), allocatable, dimension(:) :: VrtCon ! vertex connectivity list
(precursor to edge list)
integer, allocatable, dimension(:) :: VrtSolv ! vertices to solve list
integer, allocatable, dimension(:) :: VrtNoSolv ! vertices not to solve list
type(AllIntArr1D), allocatable, dimension(:) :: VrtEdg ! vertex edge list
(precursor to edge list)

integer :: i_solve ! solver iteration index
integer :: i_solve_MAX ! maximum solver iterations

character(len=200) :: FILE_OUT_MFGERROR ! manufactured error output file
character(len=200) :: FILE_OUT_MFGTEMPERATURE ! manufactured temperature output
file
character(len=200) :: FILE_OUT_TEMPERATURE ! temperature output file
character(len=200) :: FILE_OUT_RESIDUAL ! residual output file

end module Variables

module UserVariables
! This module contains all user-defined global variables.

use Types

real(8) :: k_eff ! effective thermal conductivity
real(8) :: k_eff_prime ! dimensionless effective thermal conductivity
real(8) :: T_C = 50.0d0 ! TOP COLD boundary condition value
real(8) :: T_H = 100.0d0 ! BOTTOM HOT boundary condition value
real(8) :: kmat = 100.0d0 ! matrix thermal conductivity
real(8) :: kpor = 0.0d0 ! filler thermal conductivity

end module UserVariables

module AllModules
! This module consolidates all variable definition modules.

```

```

use Variables ! include global variables
use UserVariables ! include global user-defined variables

end module AllModules

module Functions
! This module defines custom functions

use Types

contains

! structured user function definitions
type(UserFuncNum) function
UserFunction(fi,N_intr,N_real,N_char,ar_intr,ar_real,ar_char)
! This function is extensible to consolidate custom user functions. To call,
the function
! index must be specified and the appropriate integer, real, and character
inputs must be
! included in the call. The function is returned as a data structure that can
contain
! an array of integers, an array of reals, or both. The size of the function
return
! arrays must be declared by array allocation.

use AllModules
implicit none

! declare variables
integer, intent(in) :: fi ! function index

integer, intent(in) :: N_intr ! number of input integer array elements
integer, intent(in) :: N_real ! number of input real array elements
integer, intent(in) :: N_char ! number of input character array elements

integer, dimension(N_intr), intent(in) :: ar_intr ! input integer array
real(8), dimension(N_real), intent(in) :: ar_real ! input real array
character(len=*), dimension(N_char), intent(in) :: ar_char ! input character
array
integer :: dfi ! diverted function index

! divert function calls for Method of Manufactured Solution
if ( FLAG_MMS == 1 ) then ! if manufactured solution is specified
select case (fi) ! choose the original specified function index
case ( 0 ) ! original null temperature function
dfi = 4 ! divert to manufactured solution temperature function
case ( 1 ) ! original null temperature gradient function
dfi = 5 ! divert to manufactured solution temperature gradient function
case ( 2 ) ! original null heat flux function
dfi = 6 ! divert to manufactured solution temperature flux function
case ( 3 ) ! original null source function
dfi = 7 ! divert to manufactured solution source function
case ( 8 ) ! original top boundary temperature
dfi = 4 ! divert to manufactured solution temperature function

```

```

        case ( 9 ) ! original bottom boundary temperature
            dfi = 4 ! divert to manufactured solution temperature function
        case default ! any other original function
            dfi = fi ! don't divert specified function index
        end select
    else
        dfi = fi ! don't divert specified function index
    end if

! define functions - FUNCTIONS 0-7 MUST REMAIN ASSIGNED AS FOLLOWS:
!   0 = null temperature
!   1 = null temperature gradient
!   2 = null flux
!   3 = null source
!   4 = manufactured solution temperature
!   5 = manufactured solution temperature gradient
!   6 = manufactured solution flux
!   7 = manufactured solution source
select case (dfi) ! choose the diverted function
    case ( 0 ) ! null temperature
        ! This null temperature function definition should not be altered

        allocate(UserFunction%intr_ar(0))
        allocate(UserFunction%real_ar(1))
        ! INPUTS
        ! ar_real(1) = x-coordinate
        ! ar_real(2) = y-coordinate
        ! ar_real(3) = z-coordinate

        ! OUTPUTS
        ! real_ar(1) = temperature

        UserFunction%real_ar(1) = 0.0d0
    case ( 1 ) ! null temperature gradient
        ! This null temperature gradient function definition should not be altered

        allocate(UserFunction%intr_ar(0))
        allocate(UserFunction%real_ar(1:3))
        ! INPUTS
        ! ar_real(1) = x-coordinate
        ! ar_real(2) = y-coordinate
        ! ar_real(3) = z-coordinate

        ! OUTPUTS
        ! real_ar(1) = temperature gradient x-direction
        ! real_ar(2) = temperature gradient y-direction
        ! real_ar(3) = temperature gradient z-direction

        UserFunction%real_ar(1:3) = 0.0d0
    case ( 2 ) ! null heat flux
        ! This null heat flux function definition should not be altered

        allocate(UserFunction%intr_ar(0))
        allocate(UserFunction%real_ar(1:3))
        ! INPUTS

```

```

! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate
! ar_real(4) = thermal conductivity

! OUTPUTS
! real_ar(1) = flux x-direction
! real_ar(2) = flux y-direction
! real_ar(3) = flux z-direction

UserFunction%real_ar(1:3) = 0.0d0
case ( 3 ) ! null source
! This null source function definition should not be altered

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1))
! INPUTS
! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate

! OUTPUTS
! real_ar(1) = volumetric source value

UserFunction%real_ar(1) = 0.0d0
case ( 4 ) ! manufactured solution temperature
! This function defines a user-specified manufactured solution temperature
!
! THIS FUNCTION DEFINITION MUST BE MODIFIED BY THE USER

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1))
! INPUTS
! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate

! OUTPUTS
! real_ar(1) = MMS temperature

UserFunction%real_ar(1) =
dcos(2.0d0*pi*ar_real(1))*dsin(pi*ar_real(2)+0.75d0)
case ( 5 ) ! manufactured solution temperature gradient
! This function defines a user-specified manufactured solution temperature
gradient
!
! THIS FUNCTION DEFINITION MUST BE MODIFIED BY THE USER

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1:3))
! INPUTS
! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate

```

```

! OUTPUTS
! real_ar(1) = MMS temperature gradient x-direction
! real_ar(2) = MMS temperature gradient y-direction
! real_ar(3) = MMS temperature gradient z-direction

UserFunction%real_ar(1) = -
2.0d0*pi*dsin(2.0d0*pi*ar_real(1))*dsin(pi*ar_real(2)+0.75d0)
UserFunction%real_ar(2) =
pi*dcos(2.0d0*pi*ar_real(1))*dcos(pi*ar_real(2)+0.75d0)
UserFunction%real_ar(3) = 0.0d0
case ( 6 ) ! manufactured solution heat flux
! This function defines a user-specified manufactured solution heat flux
!
! THIS FUNCTION DEFINITION MUST BE MODIFIED BY THE USER

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1:3))
! INPUTS
! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate
! ar_real(4) = thermal conductivity

! OUTPUTS
! real_ar(1) = MMS flux x-direction
! real_ar(2) = MMS flux y-direction
! real_ar(3) = MMS flux z-direction

UserFunction%real_ar(1) = 0.0d0
UserFunction%real_ar(2) = 0.0d0
UserFunction%real_ar(3) = 0.0d0
case ( 7 ) ! manufactured solution source term
! This function defines a user-specified manufactured solution source term
!
! THIS FUNCTION DEFINITION MUST BE MODIFIED BY THE USER

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1))
! INPUTS
! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate
! ar_real(4) = thermal conductivity

! OUTPUTS
! real_ar(1) = MMS volumetric source value

UserFunction%real_ar(1) =
5.0d0*ar_real(4)*pi*pi*dcos(2.0d0*pi*ar_real(1))*dsin(pi*ar_real(2)+0.75d0)
case ( 8 )
! cell bottom temperatures

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1))

```

```

        ! OUTPUTS
        ! real_ar(1) = temperature

        UserFunction%real_ar(1) = T_H
case ( 9 )
    ! cell top temperature

    allocate(UserFunction%intr_ar(0))
    allocate(UserFunction%real_ar(1))

    ! OUTPUTS
    ! real_ar(1) = temperature

    UserFunction%real_ar(1) = T_C
    case default ! default function value (null return)
        allocate(UserFunction%intr_ar(0))
        allocate(UserFunction%real_ar(0))
    end select

return

end function UserFunction

end module Functions

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
MAIN PROGRAM
program Solver_3D_Rev_
! This program directs the overall solution algorithm.

use AllModules
use Functions
implicit none

! variables
integer :: i ! index
integer :: j ! index
character(len=50) :: fmt_it ! output format for iteration number
character(len=500) :: string ! dummy string

call DisplayDivision
call DisplayActionStamp('PROGRAM START')

! initialize case variables
call AnalysisDefinitions ! define full analysis set parameters

i_cas = 0 ! index of case

call DisplayDivision
write(string,'(A,i0)') 'TOTAL ANALYSIS CASES: ',Ncas
call DisplayLine01(trim(string))

do i_cas = 1 , Ncas ! loop through all analysis cases
    call DisplayDivision

```

```

write(string,'(A,i0,A)') 'CASE ',i_cas,' START'
call DisplayActionStamp(trim(string))
call DisplayDivision

! initialize variables
i_solve = 0

call CreateSolutionFolder ! create solution folder for current analysis case

call SettingsDefinitions ! define settings for current analysis case

call ImportMesh ! import mesh from input file
call DefineMaterials ! define material thermophysical and thermo-optical
properties
call DefineSurfaceThickness ! define surface element thicknesses
call ComputeVertexVolume ! compute volume of mesh vertices
call ComputeCharacterMeshSize ! compute characteristic mesh size

call BuildVoxels ! builds domain voxels
call FindVoxelContents ! finds contents of voxels

call DefineContinuumMaterial ! assign material to each continuum in mesh
call AssignMaterialsToTriangles ! assign material to each triangle in mesh
! call ComputeVertexRhoCpK

call AssignBoundaryFunctions ! assign boundary definitions to each boundary

call ComputeConductanceMatrix ! compute conductance matrix elements

call InitializeTemperatures ! initialize vertex temperatures
call ApplyBoundaryTemperatures ! fix vertices with defined temperatures

call DefineLoads ! define load characteristics for each load
call GatherLoadVertices ! creates list of vertices with applied loads
call ComputeLoadValues ! creates list of vertex heat loads

call IdentifyConstantVertices ! create lists of vertices to be solved and not
solved

call ComputeSourceTerms ! compute source terms for each vertex

call ComputeResiduals

call EstablishTemperatureFile
call EstablishResidualFile

call PrintModelInformation
call PrintSolutionInformation

if ( FLAG_MMS == 1 ) then
    call ComputeManufacturedTemperatures

```



```

        call EstablishManufacturedTemperatureFile
        call EstablishManufacturedErrorFile
        call PrintManufacturedInformation
    end if

! store original temperature and residual values
call EstablishConjugateGradientStructures
    pmult = vrt_res
    pmult(VrtNoSolv(:)) = 0.0d0
    res_crt = 0.0d0

! iterate solution
call DisplayDivision
call DisplayActionStamp('ITERATING START')

write(fmt_it,'(A,i0)') 'i',floor(log10(real(i_solve_MAX,8)))+1

do i_solve = 1, i_solve_MAX
    call UpdateSolution
    call ComputeTemperatureMfgError

    !call PrintSolutionInformation
    !call PrintManufacturedInformation

    if ( mod(i_solve,100) == 0 ) then
        write(string,'(A, '//fmt_it//',2(A, '//fmte//'))') 'It.: ',i_solve,' |
Res.: ',res_crt,' | Mfg. Error: ',mfe_crt
        call DisplayLine01(trim(string))
    end if

    if ( res_crt < res_lim ) then
        write(string,'(A, '//fmt_it//',2(A, '//fmte//'))') 'It.: ',i_solve,' |
Res.: ',res_crt,' | Mfg. Error: ',mfe_crt
        call DisplayLine01(trim(string))
        exit
    end if
end do

call DisplayActionStamp('ITERATING END')

call PrintSolutionInformation

if ( FLAG_MMS == 1 ) then
    call PrintManufacturedInformation
end if

call UserPostProcessing
call PrintUserPostSolution

call DeallocateArrays

call DisplayDivision
write(string,'(A,i0,A)') 'CASE ',i_cas,' END'
call DisplayActionStamp(trim(string))
end do

```

```

call DisplayDivision
call DisplayActionStamp('PROGRAM END')
call DisplayDivision

end program Solver_3D_Rev_

subroutine AnalysisDefinitions
! This subroutine defines the full analysis set parameters.
!
! THIS SUBROUTINE MUST BE MODIFIED BY THE USER

use AllModules
implicit none

Ncas = 1 ! number of analysis cases

end subroutine AnalysisDefinitions

subroutine SettingsDefinitions
! This subroutine defines settings for the current analysis case.
!
! THIS SUBROUTINE MUST BE MODIFIED BY THE USER

use AllModules
implicit none

FLAG_MMS = 0 ! MMS
FLAG_UPDATE_METHOD = 2 ! solution update method (1=SOR, 2=Conjugate Gradient)
fmtf = "f0.16" ! format specification for real values
i_solve_MAX = 10000000 ! maximum solver iterations
res_lim = 0.00000001 ! residual cutoff limit

if ( i_cas == 1) MeshInputFileName =
'Triangle_Porosity_10000_Rot_0_Mesh_250000.bdfgm'

fmte = "e13.6" ! format specification for scientific values
relax = 1.0d0 ! temperature update relaxation coefficient
FLAG_disp_inpt = 0 ! flag to display input file lines

end subroutine SettingsDefinitions

! ----- USER ROUTINES
subroutine UserPostProcessing
use AllModules
implicit none

call DisplayDivision
call DisplayActionStamp('USER POST-PROCESSING START')

call EffectiveThermalConductivity

call DisplayActionStamp('USER POST-PROCESSING END')

```

```

end subroutine UserPostProcessing

subroutine EffectiveThermalConductivity
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: COM_FLAG ! flag indicating common vertex has been found
real(8) :: res_sum_1 ! sum of residuals of vertices on boundary 1
real(8) :: res_sum_3 ! sum of residuals of vertices on boundary 3
integer :: bnd_no_top ! boundary number for top of domain ( laminate = 4, pore = 6)
integer :: bnd_no_bot ! boundary number for bottom of domain ( laminate = 1, pore = 1)
character(len=500) :: string ! dummy string

integer, allocatable, dimension(:) :: bnd_vrt_list ! list of boundary vertices

i = 0
res_sum_1 = 0.0d0
res_sum_3 = 0.0d0
k_eff = 0.0d0
k_eff_prime = 0.0d0

! get list of top boundary vertices
allocate(bnd_vrt_list(0))
bnd_vrt_list = (/ bnd_vrt_list(:) , bnd_evrt(5)%arr(:) /)

do i = 1, size(bnd_evrt(6)%arr)
  COM_FLAG = 0
  do j = 1, size(bnd_vrt_list)
    if ( bnd_evrt(6)%arr(i) .eq. bnd_vrt_list(j) ) then
      COM_FLAG = 1
      exit
    end if
  end do
  if ( COM_FLAG .eq. 0 ) then
    bnd_vrt_list = (/ bnd_vrt_list , bnd_evrt(6)%arr(i) /)
  end if
end do

! sum residuals across top boundaries
do i = 1, size(bnd_vrt_list)
  res_sum_1 = res_sum_1 + vrt_res(bnd_vrt_list(i))
end do

! get list of bottom boundary vertices
deallocate(bnd_vrt_list)
allocate(bnd_vrt_list(0))
bnd_vrt_list = (/ bnd_vrt_list(:) , bnd_evrt(1)%arr(:) /)

```

```

do i = 1, size(bnd_evrt(2)%arr)
  COM_FLAG = 0
  do j = 1, size(bnd_vrt_list)
    if ( bnd_evrt(2)%arr(i) .eq. bnd_vrt_list(j) ) then
      COM_FLAG = 1
      exit
    end if
  end do
  if ( COM_FLAG .eq. 0 ) then
    bnd_vrt_list = (/ bnd_vrt_list , bnd_evrt(2)%arr(i) /)
  end if
end do

! sum residuals across top boundaries
do i = 1, size(bnd_vrt_list)
  res_sum_3 = res_sum_3 + vrt_res(bnd_vrt_list(i))
end do

k_eff = dabs(( ( res_sum_1 - res_sum_3 ) / 2.0d0 ) * (1.0d0) / (T_H - T_C) /
(1.0d0))

k_eff_prime = k_eff / mat_k(cnt_mat(1))

write(string,'(A, '//fmtf//')') 'Effective Thermal Conductivity          : ',
k_eff
call DisplayLine01(trim(string))
write(string,'(A, '//fmtf//')') 'Dimensionless Effective Thermal Conductivity: ',
k_eff_prime
call DisplayLine01(trim(string))

end subroutine EffectiveThermalConductivity

subroutine PrintUserPostSolution
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_Verification'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

```

```

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! header
write(1,'(A)') "Mesh
Filename,Porosity,h,h1,h2,h3,V1,V2,k1,k2,keff,k',R_L2,R_Linf,ln(h),ln(R_L2),ln(R_
Linf),e_L2,e_Linf,ln(e_L2),ln(e_Linf)"

! data
write(1,'(A,A)',advance='no') trim(MeshInputFileName),','
if ( Ncnt .eq. 2 ) then
    write(1,'(///trim(fmtf)//',A)',advance='no') cnt_area(2)/(cnt_area(1) +
cnt_area(2)),','
else
    write(1,'(///trim(fmtf)//',A)',advance='no') 0.0d0,','
end if
write(1,'(///trim(fmtf)//',A)',advance='no') h,','
write(1,'(///trim(fmtf)//',A)',advance='no') h1,','
write(1,'(///trim(fmtf)//',A)',advance='no') h2,','
write(1,'(///trim(fmtf)//',A)',advance='no') h3,','
write(1,'(///trim(fmtf)//',A)',advance='no') cnt_area(1),','
if ( Ncnt .eq. 2 ) then
    write(1,'(///trim(fmtf)//',A)',advance='no') cnt_area(2),','
else
    write(1,'(///trim(fmtf)//',A)',advance='no') 0.0d0,','
end if
write(1,'(///trim(fmtf)//',A)',advance='no') mat_k(cnt_mat(1)),','
if ( Ncnt .eq. 2 ) then
    write(1,'(///trim(fmtf)//',A)',advance='no') mat_k(cnt_mat(2)),','
else
    write(1,'(///trim(fmtf)//',A)',advance='no') 0.0d0,','
end if
write(1,'(///trim(fmtf)//',A)',advance='no') k_eff,','
write(1,'(///trim(fmtf)//',A)',advance='no') k_eff_prime,','
write(1,'(///trim(fmtf)//',A)',advance='no') res_rms,','
write(1,'(///trim(fmtf)//',A)',advance='no') res_max,','
write(1,'(///trim(fmtf)//',A)',advance='no') dlog(h),','
write(1,'(///trim(fmtf)//',A)',advance='no') dlog(res_rms),','
write(1,'(///trim(fmtf)//',A)',advance='no') dlog(res_max),','
write(1,'(///trim(fmtf)//',A)',advance='no') mfe_rms,','
write(1,'(///trim(fmtf)//',A)',advance='no') mfe_max,','
write(1,'(///trim(fmtf)//',A)',advance='no') dlog(mfe_rms),','
write(1,'(///trim(fmtf)//',A)',advance='no') dlog(mfe_max),','
write(1,'(A)') ''

close(1)

end subroutine PrintUserPostSolution

! ----- PROGRAM ROUTINES
subroutine ApplyBoundaryTemperatures
! This subroutine enforces Dirichlet boundary conditions and any defined
! vertex temperatures on all associated vertices.

```

```

use AllModules
use functions
implicit none

! declare variables
integer :: i ! index
integer :: j ! index
integer, allocatable, dimension(:) :: IntAr ! placeholder integer array
real(8), allocatable, dimension(:) :: RlAr ! placeholder real array
character(len=200), allocatable, dimension(:) :: CharAr ! placeholder character
array

type(UserFuncNum) :: dummy

! enforce Dirichlet conditions
do i = 1, Nbnd ! loop through all boundaries
  if ( bnd_type(i) .eq. 1 ) then ! if boundary is Dirichlet/temperature condition
    ! allocate placeholder arrays
    allocate(IntAr(0))
    allocate(RlAr(3))
    allocate(CharAr(0))

    ! initialize placeholder arrays

    do j = 1, size(bnd_evrt(i)%arr(:)) ! edge boundaries
      if ( bnd_evrt(i)%arr(j) .eq. 0 ) then ! if no vertices are on boundary
        exit ! stop checking boundary
      end if

      !~dummy = UserFunction(bnd_func(i),0,0,0,IntAr,RlAr,CharAr)
      !~vrt_tmp(bnd_evrt(i)%arr(j)) = dummy%real_ar(1)

      RlAr(1:3) =
(/vrt_crd(1,bnd_evrt(i)%arr(j)),vrt_crd(2,bnd_evrt(i)%arr(j)),vrt_crd(3,bnd_evrt(
i)%arr(j))/) ! store coordinates of boundary vertex
      dummy = UserFunction(bnd_func(i),0,3,0,IntAr,RlAr,CharAr) ! define user
function
      vrt_tmp(bnd_evrt(i)%arr(j)) = dummy%real_ar(1) ! assign boundary
temperature to vertex based on assigned boundary function
    end do

    do j = 1, size(bnd_tvrt(i)%arr(:)) ! triangle boundaries
      if ( bnd_tvrt(i)%arr(j) .eq. 0 ) then ! if no triangles are on boundary
        exit ! stop checking boundary
      end if
    end do

    ! deallocate placeholder arrays
    deallocate(IntAr)
    deallocate(RlAr)
    deallocate(CharAr)
  end if
end do

end subroutine ApplyBoundaryTemperatures

```

```

subroutine AssignBoundaryFunctions
! This subroutine assigns both boundary condition type IDs and
! function IDs to each boundary.

use AllModules
implicit none

integer :: i ! index

! NOTE: Boundary Types
!      -1 = Intimate Thermal Contact
!      1 = Temperature/Dirichlet
!      2 = Temperature Gradient/Neumann
!      3 = Heat Flux
do i = 1, Nbnd ! loop through all boundaries
  select case ( i ) ! each case block defines the boundary type ID and function
    ID for a unique boundary
    case ( 1 )
      bnd_type(i) = 1
      bnd_func(i) = 8
    case ( 2 )
      bnd_type(i) = 1
      bnd_func(i) = 8
    case ( 3 )
      bnd_type(i) = 2
      bnd_func(i) = 1
    case ( 4 )
      bnd_type(i) = 2
      bnd_func(i) = 1
    case ( 5 )
      bnd_type(i) = 1
      bnd_func(i) = 9
    case ( 6 )
      bnd_type(i) = 1
      bnd_func(i) = 9
    case ( 7 )
      bnd_type(i) = 2
      bnd_func(i) = 1
    case ( 8 )
      bnd_type(i) = 2
      bnd_func(i) = 1
    case ( 9 )
      bnd_type(i) = 2
      bnd_func(i) = 1
    case ( 10 )
      bnd_type(i) = 2
      bnd_func(i) = 1
    case ( 11 )
      bnd_type(i) = 2
      bnd_func(i) = 1
  end select
end do

end subroutine AssignBoundaryFunctions

```

```

subroutine AssignMaterialsToTriangles
! This subroutine assigns a material to each triangle based on
! the material assigned to the continuum to which the triangles
! belong.

use AllModules
implicit none

! variables
integer :: i ! index

! get material ID for each triangle based on continuum ID
do i = 1, Ntri ! loop through all triangles
    tri_mat(i) = cnt_mat(tri_cntm(i))
end do

end subroutine AssignMaterialsToTriangles

subroutine BuildVoxels
! This subroutine divides the computational domain into three-
! dimensional voxels based on the set of vertices.

use AllModules
implicit none

! declare variables
real(8) :: dim_min_x ! minimum x-dimension value
real(8) :: dim_max_x ! maximum x-dimension value
real(8) :: dim_min_y ! minimum y-dimension value
real(8) :: dim_max_y ! maximum y-dimension value
real(8) :: dim_min_z ! minimum z-dimension value
real(8) :: dim_max_z ! maximum z-dimension value
integer :: i ! index
integer :: j ! index
integer :: m ! index
integer :: Nvox ! nominal number of voxels in single dimension
real(8), dimension(1:3) :: vox_wid ! voxel width (x,y,z)

! initialize variables [1]
dim_min_x = 0.0d0
dim_max_x = 0.0d0
dim_min_y = 0.0d0
dim_max_y = 0.0d0
dim_min_z = 0.0d0
dim_max_z = 0.0d0
i = 0
j = 0
m = 0
Nvox = 0
Nvxx = 0
Nvxy = 0
Nvxz = 0
vox_wid(1:3) = 0.0d0

```



```

! determine nominal number of voxels in each dimension
Nvox = 1 + nint(dlog(real(Nvrt,8))/dlog(2.0d0))
Nvox = 1*Nvox

! find limits in x-dimension
dim_min_x = minval(vrt_crd(1,1:Nvrt))
dim_max_x = maxval(vrt_crd(1,1:Nvrt))

! determine number of voxels in x-dimension
if ( dabs(dim_min_x - dim_max_x) .eq. 0.0d0 ) then
  Nvxx = 1
  vox_wid(1) = 1.0d0
else
  Nvxx = Nvox
  vox_wid(1) = 1.01d0*(dim_max_x-dim_min_x)/real(Nvxx,8)
end if

! find limits in y-dimension
dim_min_y = minval(vrt_crd(2,1:Nvrt))
dim_max_y = maxval(vrt_crd(2,1:Nvrt))

! determine number of voxels in y-dimension
if ( dabs(dim_min_y - dim_max_y) .eq. 0.0d0 ) then
  Nvxy = 1
  vox_wid(2) = 1.0d0
else
  Nvxy = Nvox
  vox_wid(2) = 1.01d0*(dim_max_y-dim_min_y)/real(Nvxy,8)
end if

! find limits in z-dimension
dim_min_z = minval(vrt_crd(3,1:Nvrt))
dim_max_z = maxval(vrt_crd(3,1:Nvrt))

! determine number of voxels in z-dimension
if ( dabs(dim_min_z - dim_max_z) .eq. 0.0d0 ) then
  Nvxz = 1
  vox_wid(3) = 1.0d0
else
  Nvxz = Nvox
  vox_wid(3) = 1.01d0*(dim_max_z-dim_min_z)/real(Nvxz,8)
end if

! allocate voxel data structures
allocate(vox_ID(1:Nvxx*Nvxy*Nvxz))
allocate(vox_crd(1:3,1:Nvxx*Nvxy*Nvxz))
allocate(vox_lim_lo(1:3,1:Nvxx*Nvxy*Nvxz))
allocate(vox_lim_hi(1:3,1:Nvxx*Nvxy*Nvxz))
allocate(vox_vrt(1:Nvxx*Nvxy*Nvxz))
allocate(vox_edg(1:Nvxx*Nvxy*Nvxz))
allocate(vox_tri(1:Nvxx*Nvxy*Nvxz))
do i = 1, Nvxx*Nvxy*Nvxz
  allocate(vox_vrt(i)%arr(0))
  allocate(vox_edg(i)%arr(0))
  allocate(vox_tri(i)%arr(0))

```

```

    vox_ID(i) = i
end do

! initialize voxel coordinates
vox_crd(1:3,1:Nvxx*Nvxy*Nvxz) = 0.0d0

! initialize voxel limits
vox_lim_lo(1:3,1:Nvxx*Nvxy*Nvxz) = 0.0d0
vox_lim_hi(1:3,1:Nvxx*Nvxy*Nvxz) = 0.0d0

! define voxel coordinates (in voxel dimensions)
do i = 1, Nvxx
  do j = 1, Nvxy
    do m = 1, Nvxz
      vox_crd(1:3,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = (/i,j,m/)
    end do
  end do
end do

! define voxel lower limits
vox_lim_lo(1,1) = (dim_min_x + dim_max_x)/2.0d0 - real(Nvxx,8)*vox_wid(1)/2.0d0
vox_lim_lo(2,1) = (dim_min_y + dim_max_y)/2.0d0 - real(Nvxy,8)*vox_wid(2)/2.0d0
vox_lim_lo(3,1) = (dim_min_z + dim_max_z)/2.0d0 - real(Nvxz,8)*vox_wid(3)/2.0d0
do i = 1, Nvxx
  do j = 1, Nvxy
    do m = 1, Nvxz
      vox_lim_lo(1,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_lo(1,1) +
vox_wid(1)*real(i-1,8)
      vox_lim_lo(2,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_lo(2,1) +
vox_wid(2)*real(j-1,8)
      vox_lim_lo(3,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_lo(3,1) +
vox_wid(3)*real(m-1,8)
    end do
  end do
end do

! define voxel upper limits
do i = 1, Nvxx-1
  do j = 1, Nvxy
    do m = 1, Nvxz
      vox_lim_hi(1,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_lo(1,Nvxx*Nvxy*(m-
1)+Nvxx*(j-1)+i+1)
    end do
  end do
end do

do i = 1, Nvxx
  do j = 1, Nvxy-1
    do m = 1, Nvxz
      vox_lim_hi(2,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_lo(2,Nvxx*Nvxy*(m-
1)+Nvxx*(j)+i)
    end do
  end do
end do

```

```

do i = 1, Nvxx
  do j = 1, Nvxy
    do m = 1, Nvxz-1
      vox_lim_hi(3,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) =
vox_lim_lo(3,Nvxx*Nvxy*(m)+Nvxx*(j-1)+i)
    end do
  end do
end do

vox_lim_hi(1,Nvxx*Nvxy*Nvxz) = (dim_min_x + dim_max_x)/2.0d0 +
real(Nvxx,8)*vox_wid(1)/2.0d0
do i = Nvxx, Nvxx
  do j = 1, Nvxy
    do m = 1, Nvxz
      vox_lim_hi(1,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_hi(1,Nvxx*Nvxy*Nvxz)
    end do
  end do
end do

vox_lim_hi(2,Nvxx*Nvxy*Nvxz) = (dim_min_y + dim_max_y)/2.0d0 +
real(Nvxy,8)*vox_wid(2)/2.0d0
do i = 1, Nvxx
  do j = Nvxy, Nvxy
    do m = 1, Nvxz
      vox_lim_hi(2,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_hi(2,Nvxx*Nvxy*Nvxz)
    end do
  end do
end do

vox_lim_hi(3,Nvxx*Nvxy*Nvxz) = (dim_min_z + dim_max_z)/2.0d0 +
real(Nvxz,8)*vox_wid(3)/2.0d0
do i = 1, Nvxx
  do j = 1, Nvxy
    do m = Nvxz, Nvxz
      vox_lim_hi(3,Nvxx*Nvxy*(m-1)+Nvxx*(j-1)+i) = vox_lim_hi(3,Nvxx*Nvxy*Nvxz)
    end do
  end do
end do

end subroutine BuildVoxels

subroutine ComputeConductanceMatrix
! This subroutine computes the elements of the conductance matrix
! relating vertex temperatures to vertex heat loads.

use AllModules
implicit none

! declare variables
integer :: i ! index

! allocate arrays
allocate(Ge_x(1:Nedg))
allocate(Ge_y(1:Nedg))
allocate(Ge_z(1:Nedg))

```

```

allocate(Gv_x(1:Nvrt))
allocate(Gv_y(1:Nvrt))
allocate(Gv_z(1:Nvrt))

! initialize variables
Ge_x(:) = 0.0d0
Ge_y(:) = 0.0d0
Ge_z(:) = 0.0d0
Gv_x(:) = 0.0d0
Gv_y(:) = 0.0d0
Gv_z(:) = 0.0d0

! loop through each triangle and accumulate conductance contributions based on
edges
do i = 1, Ntri ! loop through each triangle
  ! x conductance
  Gv_x(tri_vrt(1,i)) = Gv_x(tri_vrt(1,i)) +
mat_k(tri_mat(i))*(vrt_crd(2,tri_vrt(2,i))-
vrt_crd(2,tri_vrt(3,i)))*(vrt_crd(2,tri_vrt(2,i))-
vrt_crd(2,tri_vrt(3,i)))/4.0d0/tri_area(i) ! vertex 1
  Gv_x(tri_vrt(2,i)) = Gv_x(tri_vrt(2,i)) +
mat_k(tri_mat(i))*(vrt_crd(2,tri_vrt(3,i))-
vrt_crd(2,tri_vrt(1,i)))*(vrt_crd(2,tri_vrt(3,i))-
vrt_crd(2,tri_vrt(1,i)))/4.0d0/tri_area(i) ! vertex 2
  Gv_x(tri_vrt(3,i)) = Gv_x(tri_vrt(3,i)) +
mat_k(tri_mat(i))*(vrt_crd(2,tri_vrt(1,i))-
vrt_crd(2,tri_vrt(2,i)))*(vrt_crd(2,tri_vrt(1,i))-
vrt_crd(2,tri_vrt(2,i)))/4.0d0/tri_area(i) ! vertex 3

  Ge_x(tri_edg(1,i)) = Ge_x(tri_edg(1,i)) +
mat_k(tri_mat(i))*(vrt_crd(2,tri_vrt(2,i))-
vrt_crd(2,tri_vrt(3,i)))*(vrt_crd(2,tri_vrt(3,i))-
vrt_crd(2,tri_vrt(1,i)))/4.0d0/tri_area(i) ! edge 1
  Ge_x(tri_edg(2,i)) = Ge_x(tri_edg(2,i)) +
mat_k(tri_mat(i))*(vrt_crd(2,tri_vrt(3,i))-
vrt_crd(2,tri_vrt(1,i)))*(vrt_crd(2,tri_vrt(1,i))-
vrt_crd(2,tri_vrt(2,i)))/4.0d0/tri_area(i) ! edge 2
  Ge_x(tri_edg(3,i)) = Ge_x(tri_edg(3,i)) +
mat_k(tri_mat(i))*(vrt_crd(2,tri_vrt(2,i))-
vrt_crd(2,tri_vrt(3,i)))*(vrt_crd(2,tri_vrt(1,i))-
vrt_crd(2,tri_vrt(2,i)))/4.0d0/tri_area(i) ! edge 3

  ! y conductance
  Gv_y(tri_vrt(1,i)) = Gv_y(tri_vrt(1,i)) +
mat_k(tri_mat(i))*(vrt_crd(1,tri_vrt(2,i))-
vrt_crd(1,tri_vrt(3,i)))*(vrt_crd(1,tri_vrt(2,i))-
vrt_crd(1,tri_vrt(3,i)))/4.0d0/tri_area(i) ! vertex 1
  Gv_y(tri_vrt(2,i)) = Gv_y(tri_vrt(2,i)) +
mat_k(tri_mat(i))*(vrt_crd(1,tri_vrt(3,i))-
vrt_crd(1,tri_vrt(1,i)))*(vrt_crd(1,tri_vrt(3,i))-
vrt_crd(1,tri_vrt(1,i)))/4.0d0/tri_area(i) ! vertex 2
  Gv_y(tri_vrt(3,i)) = Gv_y(tri_vrt(3,i)) +
mat_k(tri_mat(i))*(vrt_crd(1,tri_vrt(1,i))-
vrt_crd(1,tri_vrt(2,i)))*(vrt_crd(1,tri_vrt(1,i))-
vrt_crd(1,tri_vrt(2,i)))/4.0d0/tri_area(i) ! vertex 3

```

```

      Ge_y(tri_edg(1,i)) = Ge_y(tri_edg(1,i)) +
mat_k(tri_mat(i))*(vrt_crd(1,tri_vrt(2,i))-
vrt_crd(1,tri_vrt(3,i)))*(vrt_crd(1,tri_vrt(3,i))-
vrt_crd(1,tri_vrt(1,i)))/4.0d0/tri_area(i) ! edge 1
      Ge_y(tri_edg(2,i)) = Ge_y(tri_edg(2,i)) +
mat_k(tri_mat(i))*(vrt_crd(1,tri_vrt(3,i))-
vrt_crd(1,tri_vrt(1,i)))*(vrt_crd(1,tri_vrt(1,i))-
vrt_crd(1,tri_vrt(2,i)))/4.0d0/tri_area(i) ! edge 2
      Ge_y(tri_edg(3,i)) = Ge_y(tri_edg(3,i)) +
mat_k(tri_mat(i))*(vrt_crd(1,tri_vrt(2,i))-
vrt_crd(1,tri_vrt(3,i)))*(vrt_crd(1,tri_vrt(1,i))-
vrt_crd(1,tri_vrt(2,i)))/4.0d0/tri_area(i) ! edge 3

end do

end subroutine ComputeConductanceMatrix

subroutine ComputeLoadValues
! This subroutine makes a list of all load values for vertices with an applied
load,
! currently only addressing uniform, constant flux loads.

use AllModules
use functions
implicit none

! variables
real(8) :: flux_val ! flux value
integer :: i ! index
integer :: j ! index
integer :: j_vrt ! current vertex index
integer, allocatable, dimension(:) :: IntAr ! placeholder integer array
real(8), allocatable, dimension(:) :: RlAr ! placeholder real array
character(len=200), allocatable, dimension(:) :: CharAr ! placeholder character
array

type(UserFuncNum) :: dummy

! initialize variables [1]
flux_val = 0.0d0
i = 0
j = 0
j_vrt = 0

! preallocate arrays

! initialize variables [2]

! NOTE: Load Types
!      -1 = Unassigned
!      0 = Absolute heat load, q (i.e., W)
!      1 = Line load, q' (i.e., W/m)
!      2 = Heat Flux, q'' (i.e., W/m2)
!      3 = Volumetricload, q''' (i.e., W/m3)

```

```

! NOTE: Load Spatial Distribution Types
!       -1 = Unassigned
!       0 =
!       1 = Uniform
!       2 = Spatial function
! NOTE: Load Temporal Behavior Types
!       -1 = Unassigned
!       0 =
!       1 = Constant
!       2 = Time-dependent
!       3 = Temperature-dependent
!       4 = Time- and temperature-dependent
! loop through all loads
do i = 1, Nlod
  select case ( lod_dist(i) ) ! determine spatial distribution
    case ( -1 ) ! unassigned
    case ( 0 ) !
    case ( 1 ) ! uniform
      select case ( lod_type(i) ) ! determine load type
        case ( -1 ) ! unassigned
        case ( 0 ) ! absolute
        case ( 1 ) ! line
        case ( 2 ) ! flux
          ! allocate placeholder arrays
          allocate(IntAr(0))
          allocate(RlAr(5))
          allocate(CharAr(0))

          ! loop through all load vertices
          do j = 1, size(lod_cntvrt(i)%arr(:))
            ! store current vertex
            j_vrt = lod_cntvrt(i)%arr(j)

            ! get flux
            RlAr(1:5) =
(/vrt_crd(1,j_vrt),vrt_crd(2,j_vrt),vrt_crd(3,j_vrt),0.0d0,vrt_tmp(j_vrt)/) !
store coordinates of boundary vertex
            dummy = UserFunction(lod_func(i),0,5,0,IntAr,RlAr,CharAr) ! define
user function
            flux_val = dummy%real_ar(1) ! define vertex flux value

            ! multiply flux by vertex area
            lod_cvrtlod(i)%arr = flux_val*lod_cvrtarea(i)%arr
          end do

          ! deallocate placeholder arrays
          deallocate(IntAr)
          deallocate(RlAr)
          deallocate(CharAr)
        case ( 3 ) ! volumetric
      end select
    case ( 2 ) ! spatial function
  end select
end do
end do

```

```
end subroutine ComputeLoadValues
```

```
subroutine ComputeResiduals
```

```
use AllModules
```

```
use Functions
```

```
implicit none
```

```
! computes initial source terms
```

```
! declare variables
```

```
integer :: i ! index
```

```
integer :: j ! index
```

```
integer :: vi ! vertex index
```

```
real(8) :: GeTsum
```

```
GeTsum = 0.0d0
```

```
vrt_res(1:Nvrt) = 0.0d0
```

```
res_crt = 0.0d0 ! critical residual
```

```
res_max = 0.0d0 ! maximum residual
```

```
res_min = 0.0d0 ! minimum residual
```

```
res_rss = 0.0d0 ! root sum of squares of residuals
```

```
res_rms = 0.0d0 ! root mean squares of residuals
```

```
! compute residual for first vertex
```

```
do i = 1, 1
```

```
    vi = VrtSolv(i)
```

```
    GeTsum = 0.0d0
```

```
    ! compute sum of T*Ge for each vertex
```

```
    do j = 1, size(vrt_adjedg(vi)%arr(:))
```

```
        GeTsum = GeTsum +
```

```
        vrt_tmp(vrt_adjvrt(vi)%arr(j))*(Ge_x(vrt_adjedg(vi)%arr(j))+Ge_y(vrt_adjedg(vi)%a  
rr(j))+Ge_z(vrt_adjedg(vi)%arr(j)))
```

```
    end do
```

```
        vrt_res(vi) = (vrt_src(1,vi) + vrt_src(3,vi)) -
```

```
        vrt_tmp(vi)*(Gv_x(vi)+Gv_y(vi)+Gv_z(vi)) - GeTsum
```

```
        res_max = maxval((/dabs(vrt_res(vi)),res_max/))
```

```
        res_min = vrt_res(vi)
```

```
        res_rss = res_rss + vrt_res(vi)*vrt_res(vi)
```

```
end do
```

```
! compute residual for each solution vertex
```

```
do i = 2, size(VrtSolv(:))
```

```
    vi = VrtSolv(i)
```

```
    GeTsum = 0.0d0
```

```
    ! compute sum of T*Ge for each vertex
```

```
    do j = 1, size(vrt_adjedg(vi)%arr(:))
```

```
        GeTsum = GeTsum +
```

```
        vrt_tmp(vrt_adjvrt(vi)%arr(j))*(Ge_x(vrt_adjedg(vi)%arr(j))+Ge_y(vrt_adjedg(vi)%a  
rr(j))+Ge_z(vrt_adjedg(vi)%arr(j)))
```

```
    end do
```

```

        vrt_res(vi) = (vrt_src(1,vi) + vrt_src(3,vi)) -
vrt_tmp(vi)*(Gv_x(vi)+Gv_y(vi)+Gv_z(vi)) - GeTsum

        res_max = maxval((/dabs(vrt_res(vi)),res_max/))
        res_min = minval((/dabs(vrt_res(vi)),res_min/))
        res_rss = res_rss + vrt_res(vi)*vrt_res(vi)
    end do
    res_rms = dsqrt(res_rss/real(size(VrtSolv(:)),8))
    res_rss = dsqrt(res_rss)

    ! compute residual for each non-solution vertex
    do i = 1, size(VrtNoSolv(:))
        vi = VrtNoSolv(i)
        GeTsum = 0.0d0
        ! compute sum of T*Ge for each vertex
        do j = 1, size(vrt_adjedg(vi)%arr(:))
            GeTsum = GeTsum +
vrt_tmp(vrt_adjvrt(vi)%arr(j))*(Ge_x(vrt_adjedg(vi)%arr(j))+Ge_y(vrt_adjedg(vi)%
rr(j))+Ge_z(vrt_adjedg(vi)%arr(j)))
        end do

        vrt_res(vi) = (vrt_src(1,vi) + vrt_src(3,vi)) -
vrt_tmp(vi)*(Gv_x(vi)+Gv_y(vi)+Gv_z(vi)) - GeTsum
    end do

    res_crt = res_rms

end subroutine ComputeResiduals

subroutine ComputeSourceTerms
! This subroutine computes source terms for each vertex in the mesh.

use AllModules
use Functions
implicit none
! computes initial source terms

! declare variables
integer :: i ! index
integer :: j ! index
real(8), dimension(1:3) :: t_grad1 ! temperature gradient 1 (xyz-components)
type(UserFuncNum) :: func_ret ! function return structure
integer, allocatable, dimension(:) :: IntAr ! placeholder integer array
real(8), allocatable, dimension(:) :: RlAr ! placeholder real array
character(len=200), allocatable, dimension(:) :: CharAr ! placeholder character
array
real(8) :: source ! source term
integer :: ivrt ! vertex index

! allocate arrays
allocate(vrt_src(1:3,1:Nvrt))
vrt_src(1:3,1:Nvrt) = 0.0d0

! initialize variables
t_grad1(:) = 0.0d0

```



```

! boundary source terms
! loop through all boundaries
! source term is accumulated if boundary is temperature gradient or heat flux
do i = 1, Nbnd ! loop through all boundaries
  select case ( bnd_type(i) ) ! each case block addresses a unique boundary
  condition type
    case ( 2 ) ! temperature gradient
      ! allocate placeholder arrays
      allocate(IntAr(0))
      allocate(RlAr(4))
      allocate(CharAr(0))

      ! initialize placeholder arrays

      ! loop through all edges on boundary
      do j = 1, size(bnd_edg(i)%arr(:))
        ! get gradient value at edge first vertex
        RlAr = (/vrt_crd(1,edg_vrt(1,bnd_edg(i)%arr(j))) ,
vrt_crd(2,edg_vrt(1,bnd_edg(i)%arr(j))) ,
vrt_crd(3,edg_vrt(1,bnd_edg(i)%arr(j))),mat_k(tri_mat(edg_adjtri(bnd_edg(i)%arr(j)
))%arr(1)))/)
        func_ret = UserFunction(bnd_func(i),0,4,0,IntAr,RlAr,CharAr)
        t_grad1(1:3) = func_ret%real_ar(1:3)

        ! compute net boundary source on vertex 1
        vrt_src(3,edg_vrt(1,bnd_edg(i)%arr(j))) =
vrt_src(3,edg_vrt(1,bnd_edg(i)%arr(j))) +
(t_grad1(1)*edg_nrm1x(bnd_edg(i)%arr(j))%arr(1) +
t_grad1(2)*edg_nrm1y(bnd_edg(i)%arr(j))%arr(1) +
t_grad1(3)*edg_nrm1z(bnd_edg(i)%arr(j))%arr(1))*edg_lng(bnd_edg(i)%arr(j))*mat_k(
tri_mat(edg_adjtri(bnd_edg(i)%arr(j))%arr(1))*tri_thk(edg_adjtri(bnd_edg(i)%arr(
j))%arr(1))/2.0d0

        ! get gradient value at edge second vertex
        RlAr =
(/vrt_crd(1,edg_vrt(2,bnd_edg(i)%arr(j))),vrt_crd(2,edg_vrt(2,bnd_edg(i)%arr(j)))
,vrt_crd(3,edg_vrt(2,bnd_edg(i)%arr(j))),mat_k(tri_mat(edg_adjtri(bnd_edg(i)%arr(
j))%arr(1)))/)
        func_ret = UserFunction(bnd_func(i),0,4,0,IntAr,RlAr,CharAr)
        t_grad1(1:3) = func_ret%real_ar(1:3)

        ! compute net boundary source on vertex 2
        vrt_src(3,edg_vrt(2,bnd_edg(i)%arr(j))) =
vrt_src(3,edg_vrt(2,bnd_edg(i)%arr(j))) +
(t_grad1(1)*edg_nrm1x(bnd_edg(i)%arr(j))%arr(1) +
t_grad1(2)*edg_nrm1y(bnd_edg(i)%arr(j))%arr(1) +
t_grad1(3)*edg_nrm1z(bnd_edg(i)%arr(j))%arr(1))*edg_lng(bnd_edg(i)%arr(j))*mat_k(
tri_mat(edg_adjtri(bnd_edg(i)%arr(j))%arr(1))*tri_thk(edg_adjtri(bnd_edg(i)%arr(
j))%arr(1))/2.0d0
      end do

      ! deallocate placeholder arrays
      deallocate(IntAr)
      deallocate(RlAr)
    end case
  end select
end do

```

```

        deallocate(CharAr)
        case ( 3 ) ! heat flux
            ! loop through all edges on boundary and apply the functional heat flux
to each vertex
        end select
    end do

    ! for all boundary edges
    ! S_b_e = 0.0d0

    ! for all boundary triangles
    ! S_b_t = 0.0d0

    ! direct source terms
    ! add heat load to each vertex as defined by direct heat loads or MMS
do i = 1, Nlod ! loop through all loads
    select case ( lod_type(i) ) ! determine load type
        case ( -1 ) ! unassigned
        case ( 0 ) ! absolute
        case ( 1 ) ! line
        case ( 2 ) ! flux
            do j = 1, size(lod_cntvrt(i)%arr(:)) ! loop through all vertices in load
                ivrt = lod_cntvrt(i)%arr(j) ! store current vertex ID

                vrt_src(1,ivrt) = vrt_src(1,ivrt) + lod_cvrtlod(i)%arr(j) ! add load
contribution to vertex source term
            end do
        case ( 3 ) ! volumetric
    end select
end do

do i = 1 , size(VrtSolv(:)) ! loop through all vertices to be solved
    ! allocate placeholder arrays
    allocate(IntAr(0))
    allocate(RlAr(4))
    allocate(CharAr(0))

    ivrt = VrtSolv(i) ! store ID of vertex to be solved

    ! get source term contribution from each surrounding triangle
    RlAr = (/vrt_crd(1,ivrt),vrt_crd(2,ivrt),vrt_crd(3,ivrt),0.0d0/)
    do j = 1 , size(vrt_adjtri(ivrt)%arr(:)) ! loop through all triangles
surrounding current vertex
        RlAr(4) = mat_k(tri_mat(vrt_adjtri(ivrt)%arr(j))) ! store thermal
conductivity of adjacent triangle
        func_ret = UserFunction(3,0,4,0,IntAr,RlAr,CharAr) ! define user function

        vrt_src(1,ivrt) = vrt_src(1,ivrt) + func_ret%real_ar(1) *
(tri_area(vrt_adjtri(ivrt)%arr(j))/3.0d0) ! compute source term contribution
    end do

    ! deallocate placeholder arrays
    deallocate(IntAr)
    deallocate(RlAr)

```

```

    deallocate(CharAr)
end do

end subroutine ComputeSourceTerms

subroutine ComputeTemperatureMfgError
use AllModules
use Functions
implicit none

integer :: i ! index
integer :: vi ! vertex index

vrt_mfe(1:Nvrt) = 0.0d0
mfe_crt = 0.0d0 ! critical mfg error
mfe_max = 0.0d0 ! maximum mfg error
mfe_min = vrt_tmp(VrtSolv(1)) - vrt_tmpm(VrtSolv(1)) ! minimum mfg error
mfe_rss = 0.0d0 ! root sum of squares of mfg error
mfe_rms = 0.0d0 ! root mean squares of mfg error

! compute mfg error for each solution vertex
do i = 1, Nvrt!size(VrtSolv(:))
    vi = i!VrtSolv(i)

    vrt_mfe(vi) = vrt_tmp(vi) - vrt_tmpm(vi)

    mfe_max = maxval((/dabs(vrt_mfe(vi)),mfe_max/))
    mfe_min = minval((/dabs(vrt_mfe(vi)),mfe_min/))
    mfe_rss = mfe_rss + vrt_mfe(vi)*vrt_mfe(vi)
end do
mfe_rms = dsqrt(mfe_rss/real(Nvrt,8))
mfe_rss = dsqrt(mfe_rss)

! compute mfg error for each non-solution vertex
do i = 1, size(VrtNoSolv(:))
    vi = VrtNoSolv(i)

    vrt_mfe(vi) = vrt_tmp(vi) - vrt_tmpm(vi)
end do

mfe_crt = mfe_rms

end subroutine ComputeTemperatureMfgError

subroutine ComputeVertexVolume
! This subroutine computes the total volume for each vertex.

use AllModules
implicit none

integer :: i ! index

! add triangle volume contributions to each vertex
do i = 1, Ntri ! loop through all triangles
    ! add triangle contribution to triangle first vertex

```

```

vrt_vol(tri_vrt(1,i)) = vrt_vol(tri_vrt(1,i)) + tri_thk(i)*tri_area(i)/3.0d0

! add triangle contribution to triangle second vertex
vrt_vol(tri_vrt(2,i)) = vrt_vol(tri_vrt(2,i)) + tri_thk(i)*tri_area(i)/3.0d0

! add triangle contribution to triangle thrid vertex
vrt_vol(tri_vrt(3,i)) = vrt_vol(tri_vrt(3,i)) + tri_thk(i)*tri_area(i)/3.0d0
end do

end subroutine ComputeVertexVolume

subroutine ComputeCharacterMeshSize
! This subroutine computes the characteristic mesh size of the
! computational domain for 1D, 2D, and 3D elements.

use AllModules
implicit none

integer :: i ! inded
character(len=500) :: string ! dummy string

h = 0.0d0
h1 = 0.0d0
h2 = 0.0d0
h3 = 0.0d0
i = 0

! compute 2D characteristic mesh size
do i = 1, Nel2 ! loop through all 2D elements
    h2 = h2 + tri_area(i) ! accumulate areas of 2D elements
end do
if ( Nel2 > 0 ) h2 = (h2 / real(Nel2,8))**(1.0d0/2.0d0) ! take the root of the
average area

! compute 3D characteristic mesh size
!~do i = 1, Nel3
!~    h3 = h3 + tet_vol(i)
!~end do
!~if ( Nel3 > 0 ) h3 = (h3 / real(Nel3,8))**(1.0d0/3.0d0)

! compute overall characteristic mesh size by weighted average
h = ( real(Nel1,8)*h1 + real(Nel2,8)*h2 + real(Nel3,8)*h3 ) / ( real(Nel1,8) +
real(Nel2,8) + real(Nel3,8) )

! display results to user
write(string,'(A)') "Characteristic Mesh Sizes:"
call DisplayLine01(trim(string))
write(string,'(A, '//fmtf//)') "    1D: ",h1
call DisplayLine02(trim(string))
write(string,'(A, '//fmtf//)') "    2D: ",h2
call DisplayLine02(trim(string))
write(string,'(A, '//fmtf//)') "    3D: ",h3
call DisplayLine02(trim(string))
write(string,'(A, '//fmtf//)') "Overall: ",h
call DisplayLine02(trim(string))

```

```

end subroutine ComputeCharacterMeshSize

subroutine ComputeManufacturedTemperatures
use AllModules
use Functions
implicit none

integer :: i ! index
type(UserFuncNum) :: func_ret ! function return structure
integer, allocatable, dimension(:) :: IntAr ! placeholder integer array
real(8), allocatable, dimension(:) :: RlAr ! placeholder real array
character(len=200), allocatable, dimension(:) :: CharAr ! placeholder character
array

! allocate placeholder arrays
allocate(IntAr(0))
allocate(RlAr(3))
allocate(CharAr(0))

do i = 1 , Nvrt
    RlAr(1:3) = (/vrt_crd(1,i),vrt_crd(2,i),vrt_crd(3,i)/)
    func_ret = UserFunction(4,0,3,0,IntAr,RlAr,CharAr)
    vrt_tmpm(i) = func_ret%real_ar(1)
end do

! deallocate placeholder arrays
deallocate(IntAr)
deallocate(RlAr)
deallocate(CharAr)

end subroutine ComputeManufacturedTemperatures

subroutine CreateSolutionFolder
! This subroutine creates a new directory in the current working folder
! as a destination for all case output files.

use AllModules
use ifport
implicit none

! initialize variables
character(len=50) dir_app ! directory root appendix
character(len=50) fmt_app ! appendix number format
integer :: istat ! status flag for working directory
logical :: result ! success flag of making new solution directory

call DisplayActionStamp('SOLUTION PATH BUILD START')

dir_slt_rt = "Solution" ! solution directory root name

istat = getcwd(pth_wrk_dir) ! get current working directory path

```

```

write(fmt_app,'(A,i0)') 'i0.',floor(log10(real(Ncas,8)))+2 ! get number of digits
for solution folder appendices (appendix format)
write(dir_app,'(//fmt_app//)') i_cas ! store solution appendix with correct
number of digits (leading zeros)

dir_slt =
adjustl(trim(pth_wrk_dir))//'\ '//'adjustl(trim(dir_slt_rt))//'_ '//'adjustl(trim(dir
_app)) ! build path of solution folder

! display information to user
call DisplayLine01('Solution folder path:')
call DisplayLine02(trim(dir_slt))

! make solution directory
result = makedirqq(dir_slt)

call DisplayActionStamp('SOLUTION PATH BUILD END')

end subroutine CreateSolutionFolder

subroutine CrossProduct(v1,v2,xprdct)
use AllModules
implicit none
! computes the cross product of two vectors

! variables
real(8), dimension(1:3), intent(in) :: v1 ! first vector (x,y,z components)
real(8), dimension(1:3), intent(in) :: v2 ! second vector (x,y,z components)
real(8), dimension(1:3), intent(out) :: xprdct ! cross product (x,y,z components)

! initialize variables
xprdct(1:3) = 0.0d0

xprdct(1) = v1(2)*v2(3)-v1(3)*v2(2)
xprdct(2) = v1(3)*v2(1)-v1(1)*v2(3)
xprdct(3) = v1(1)*v2(2)-v1(2)*v2(1)

end subroutine CrossProduct

subroutine DeallocateArrays
use AllModules
implicit none

deallocate(bnd_ID) ! boundary ID number
deallocate(bnd_name) ! boundary name
deallocate(bnd_type) ! boundary type (-1=Intimate Thermal
Contact,1=Temperature/Dirichlet,2=Temperature Gradient/Neumann,3=Heat Flux)
deallocate(bnd_func) ! function ID to define boundary
deallocate(bnd_evrt) ! list of edge vertices on boundary
deallocate(bnd_tvrt) ! list of triangle vertices on boundary
deallocate(bnd_edg) ! list of edges on boundary
deallocate(cnt_ID) ! list of continuum IDs
deallocate(cnt_mat) ! continuum material assignment list
deallocate(cnt_area) ! continuum area
deallocate(cnt_tri) ! list of triangles in continuum

```

```

deallocate(edg_ID)
deallocate(edg_vrt)
deallocate(edg_mdpt)
deallocate(edg_lng)
deallocate(edg_tngt)
deallocate(edg_nrmlx)
deallocate(edg_nrmlly)
deallocate(edg_nrmlz)
deallocate(edg_adjvrt)
deallocate(edg_adjtri)
deallocate(edg_vox)
deallocate(edg_bnd)
deallocate(Ge_x) ! edge (vertex-to-vertex) conductance (x-dimension)
deallocate(Ge_y) ! edge (vertex-to-vertex) conductance (y-dimension)
deallocate(Ge_z) ! edge (vertex-to-vertex) conductance (z-dimension)
deallocate(Gv_x) ! vertex (same vertex) conductance (x-dimension)
deallocate(Gv_y) ! vertex (same vertex) conductance (y-dimension)
deallocate(Gv_z) ! vertex (same vertex) conductance (z-dimension)
deallocate(lod_ID)
deallocate(lod_name)
deallocate(lod_type)
deallocate(lod_dist)
deallocate(lod_distfunc)
deallocate(lod_func)
deallocate(lod_temp)
deallocate(lod_tempfunc)
deallocate(lod_cnt)
deallocate(lod_cntvrt)
deallocate(lod_cvrtarea)
deallocate(lod_cvrtlod)
deallocate(lod_lin)
deallocate(lod_linvrt)
deallocate(lod_lvrtlod)
deallocate(tri_ID)
deallocate(tri_vrt)
deallocate(tri_edg)
deallocate(tri_area)
deallocate(tri_cntr)
deallocate(tri_nrml)
deallocate(tri_cntm)
deallocate(tri_mat)
deallocate(tri_thk)
deallocate(tri_vox)
deallocate(vox_ID)
deallocate(vox_crd)
deallocate(vox_lim_lo)
deallocate(vox_lim_hi)
deallocate(vox_vrt)
deallocate(vox_edg)
deallocate(vox_tri)
deallocate(vrt_ID)
deallocate(vrt_crd)
deallocate(vrt_adjvrt)
deallocate(vrt_adjedg)
deallocate(vrt_adjtri)

```

```

deallocate(vrt_bnd)
deallocate(vrt_area)
deallocate(vrt_mfe)
deallocate(vrt_vol)
deallocate(vrt_vox)
deallocate(vrt_tmp)
deallocate(vrt_tmpm)
deallocate(pmult)
deallocate(vrt_src)
deallocate(vrt_res)
deallocate(mat_ID)
deallocate(mat_name)
deallocate(mat_k)
deallocate(mat_rho)
deallocate(mat_cp)
deallocate(mat_mu)
deallocate(mat_e)
deallocate(mat_r)
deallocate(mat_t)
deallocate(VrtCon) ! vertex connectivity list (precursor to edge list)
deallocate(VrtSolv) ! vertices to solve list
deallocate(VrtNoSolv) ! vertices not to solve list
deallocate(VrtEdg)

end subroutine DeallocateArrays

subroutine DefineContinuumMaterial
! This subroutine assigns a material to each mesh continuum.

use AllModules
implicit none

integer :: i ! index

! preallocate arrays
allocate(cnt_mat(1:Ncnt))

! initialize variables
cnt_mat = 0

! assign material ID to each continuum
do i = 1 , Ncnt ! loop through all continuums
  select case ( i ) ! each case block defines material for unique continuum
    case ( 1 )
      cnt_mat(i) = 1
    case ( 2 )
      cnt_mat(i) = 2
  end select
end do

end subroutine DefineContinuumMaterial

subroutine DefineSurfaceThickness
! This subroutine defines the thickness of each surface element
! in the mesh.

```



```

use AllModules
implicit none

integer :: i ! index

call DisplayActionStamp('ELEMENTS PROCESSING START')

do i = 1, Ntri ! loop through all triangles
    tri_thk(i) = 1.0d0
end do

call DisplayActionStamp('ELEMENTS PROCESSING END')

end subroutine DefineSurfaceThickness

subroutine DefineMaterials
! This subroutine defines a database of materials with their
! associated thermophysical and themoptical properties.

use AllModules
implicit none

! variables
integer :: i ! index

call DisplayActionStamp('MATERIALS PROCESSING START')

! initialize variables [1]
Nmat = 1

! preallocate arrays
allocate(mat_ID(1:Nmat))
allocate(mat_name(1:Nmat))
allocate(mat_k(1:Nmat))
allocate(mat_rho(1:Nmat))
allocate(mat_cp(1:Nmat))
allocate(mat_mu(1:Nmat))
allocate(mat_e(1:Nmat))
allocate(mat_r(1:Nmat))
allocate(mat_t(1:Nmat))

! initialize variables [2]
do i = 1, Nmat
    mat_ID(i) = i
    write(mat_name(i), '(A,i0)') 'Material_', i
    mat_k(i) = 1.0d0
    mat_rho(i) = 1.0d0
    mat_cp(i) = 1.0d0
    mat_mu(i) = 1.0d0
    mat_e(i) = 1.0d0
    mat_r(i) = 0.0d0
    mat_t(i) = 0.0d0
end do

```

```

! assign material properties to each material
do i = 1, Nmat ! loop through all materials
  select case ( i ) ! each case block defines a unique material
    case ( 1 )
      ! MATERIAL 1
      mat_name(i) = 'Matrix Material'
      mat_k(i) = kmat
      mat_rho(i) = 1.0d0
      mat_cp(i) = 1.0d0
      mat_mu(i) = 1.0d0
      mat_e(i) = 1.0d0
      mat_r(i) = 0.0d0
      mat_t(i) = 0.0d0
    end select
  end do

call DisplayActionStamp('MATERIALS PROCESSING END')

end subroutine DefineMaterials

subroutine Defineloads
! This subroutine defines a database of thermal loads with their
! associated characteristics.

use AllModules
implicit none

! variables
integer :: i ! index

! initialize variables [1]
Nlod = 0

! preallocate arrays
allocate(lod_ID(1:Nlod))
allocate(lod_name(1:Nlod))
allocate(lod_type(1:Nlod))
allocate(lod_dist(1:Nlod))
allocate(lod_distfunc(1:Nlod))
allocate(lod_func(1:Nlod))
allocate(lod_temp(1:Nlod))
allocate(lod_tempfunc(1:Nlod))
allocate(lod_cnt(1:Nlod))
allocate(lod_cntvrt(1:Nlod))
allocate(lod_cvrtarea(1:Nlod))
allocate(lod_cvrtlod(1:Nlod))
allocate(lod_lin(1:Nlod))
allocate(lod_linvrt(1:Nlod))
allocate(lod_lvrtlod(1:Nlod))

! initialize variables [2]
do i = 1, Nlod
  lod_ID(i) = i
  write(lod_name(i), '(A,i0)') 'Load_', i
  lod_type(i) = -1

```

```

    lod_dist(i) = -1
    lod_distfunc(i) = 0
    lod_func(i) = 0
    lod_temp(i) = -1
    lod_tempfunc(i) = 0
    allocate(lod_cnt(i)%arr(0))
    allocate(lod_lin(i)%arr(0))
end do

! assign load characteristics to each load
! NOTE: Load Types
!     -1 = Unassigned
!     0 = Absolute heat load, q (i.e., W)
!     1 = Line load, q' (i.e., W/m)
!     2 = Heat Flux, q'' (i.e., W/m2)
!     3 = Volumetricload, q''' (i.e., W/m3)
! NOTE: Load Spatial Distribution Types
!     -1 = Unassigned
!     0 =
!     1 = Uniform
!     2 = Spatial function
! NOTE: Load Temporal Behavior Types
!     -1 = Unassigned
!     0 =
!     1 = Constant
!     2 = Time-dependent
!     3 = Temperature-dependent
!     4 = Time- and temperature-dependent
do i = 1, Nlod ! loop through all loads
    select case ( i ) ! each case block defines a unique lod
        case ( 1 )
            ! LOAD 1
            lod_name(i) = 'Load 1'
            lod_type(i) = 2
            lod_dist(i) = 1
            lod_distfunc(i) = 0
            lod_func(i) = 9
            lod_temp(i) = 1
            lod_tempfunc(i) = 0
            lod_cnt(i)%arr = (/lod_cnt(i)%arr,1/)
            lod_lin(i)%arr = (/lod_lin(i)%arr/)
        end select
    end do

end subroutine DefineLoads

subroutine Determinant2D(v1,v2,dtrm)
use AllModules
implicit none
! computes the determinant of a two vectors after transforming them into
! an xy plane

! variables
real(8), intent(out) :: dtrm ! determinant
real(8), dimension(1:3), intent(in) :: v1 ! first vector (x,y,z components)

```

```

real(8), dimension(1:2) :: v1xy ! first vector rotated to xy plane (x,y
components)
real(8), dimension(1:3), intent(in) :: v2 ! second vector (x,y,z components)
real(8), dimension(1:2) :: v2xy ! second vector rotated to xy plane (x,y
components)

! initialize variables
dtrm = 0.0d0
v1xy(1:2) = 0.0d0
v2xy(1:2) = 0.0d0

! transform vectors to xy plane
! get vector perpendicular to plane
! call CrossProduct2D(v1,v2,z1)
! build rotation matrix
v1xy = v1(1:2)
v2xy = v2(1:2)

! compute determinant
dtrm = v1xy(1)*v2xy(2) - v1xy(2)*v2xy(1)

end subroutine Determinant2D

subroutine EdgeDirectionCode(vec_dir,edg_dir_code)
use AllModules
implicit none
! defines the directionality code of a 3D vector

! variables
real(8), dimension(1:3), intent(in) :: vec_dir
integer, dimension(1:3), intent(out) :: edg_dir_code

! initialize
edg_dir_code = (/0,0,0/) ! initialize code

! check x-dimension
if ( vec_dir(1) < 0.0d0 ) edg_dir_code(1) = -1
if ( vec_dir(1) > 0.0d0 ) edg_dir_code(1) = 1

! check y-dimension
if ( vec_dir(2) < 0.0d0 ) edg_dir_code(2) = -1
if ( vec_dir(2) > 0.0d0 ) edg_dir_code(2) = 1

! check z-dimension
if ( vec_dir(3) < 0.0d0 ) edg_dir_code(3) = -1
if ( vec_dir(3) > 0.0d0 ) edg_dir_code(3) = 1

if ( sum(abs(edg_dir_code(1:3))) .eq. 0 ) then
  write(*,*) "ERROR: Vector has no directionality."
  stop
end if

end subroutine EdgeDirectionCode

subroutine EstablishConjugateGradientStructures

```

```

! This subroutine initializes conjugate gradient solution update
! structures.

use AllModules
implicit none

pmult = vrt_res
pmult(VrtNoSolv(:)) = 0.0d0

end subroutine EstablishConjugateGradientStructures

subroutine FindVoxelContents
! This subroutine makes list of all vertices, edges, and triangles
! found in each voxel.

use AllModules
implicit none

! variables
integer, dimension(1:3) :: edg_dir_code ! edge directionality code (x,y,z; -
1=negative,0=zero,1=positive)
integer, dimension(1:3) :: edg_dir_code_srch ! vector directionality code (x,y,z;
-1=negative,0=zero,1=positive)
integer :: i ! index
integer :: j ! index
integer :: m ! index
integer :: n ! index
integer :: i_x ! index placeholder
integer :: i_y ! index placeholder
integer :: i_z ! index placeholder
integer :: vox_start ! intial voxel ID
integer :: vox_stop ! terminal voxel ID
integer :: vox_curr ! current voxel ID
integer :: vox_next ! next voxel ID
integer :: int_type ! line-plane intersection type (0=none, 1=point, 2=linear)
real(8), dimension(1:3) :: pnv ! plane normal vector (x,y,z)
real(8), dimension(1:3) :: pp0 ! point on plane (x,y,z)
integer :: FLAG_LPX_yz ! line-yz plane intersection flag (0=no intersection,
1=point intersection, 2=linear intersection)
integer :: FLAG_LPX_zx ! line-zx plane intersection flag (0=no intersection,
1=point intersection, 2=linear intersection)
integer :: FLAG_LPX_xy ! line-xy plane intersection flag (0=no intersection,
1=point intersection, 2=linear intersection)
integer :: int_plane ! plane of edge intersection (0=none,1=yz,2=zx,3=xy)
real(8), dimension(1:3) :: int_yz ! point of intersection on yz plane (x,y,z)
real(8), dimension(1:3) :: int_zx ! point of intersection on zx plane (x,y,z)
real(8), dimension(1:3) :: int_xy ! point of intersection on xy plane (x,y,z)
real(8), dimension(1:3) :: int_pnt ! point of intersection (x,y,z)
real(8), dimension(1:3) :: vec_yz ! vector from edge vertex to point of
intersection on yz plane (x,y,z)
real(8), dimension(1:3) :: vec_zx ! vector from edge vertex to point of
intersection on zx plane (x,y,z)
real(8), dimension(1:3) :: vec_xy ! vector from edge vertex to point of
intersection on xy plane (x,y,z)
real(8) :: int_dist_yz ! distance at point of edge-yz plane intersection

```

```

real(8) :: int_dist_zx ! distance at point of edge-zx plane intersection
real(8) :: int_dist_xy ! distance at point of edge-xy plane intersection
real(8) :: int_dist ! distance at point of edge-plane intersection
real(8), dimension(1:3) :: vec_srch ! voxel search direction vector (x,y,z)
integer :: vox_dif ! difference between two voxel IDs
real(8), dimension(1:3) :: ray_start ! starting coordinates for voxel search ray
(x,y,z)

! initialize variables [1]
edg_dir_code(1:3) = (/0,0,0/)
edg_dir_code_srch(1:3) = (/0,0,0/)
i = 0
j = 0
m = 0
n = 0
i_x = 0
i_y = 0
i_z = 0
int_type = 0
int_plane = 0
vox_start = 0
vox_stop = 0
vox_curr = 0
vox_next = 0
pnt = (/0.0d0,0.0d0,0.0d0/)
pp0 = (/0.0d0,0.0d0,0.0d0/)
FLAG_LPX_yz = 0
FLAG_LPX_zx = 0
FLAG_LPX_xy = 0
int_yz = (/0.0d0,0.0d0,0.0d0/)
int_zx = (/0.0d0,0.0d0,0.0d0/)
int_xy = (/0.0d0,0.0d0,0.0d0/)
int_pnt = (/0.0d0,0.0d0,0.0d0/)
vec_yz = (/0.0d0,0.0d0,0.0d0/)
vec_zx = (/0.0d0,0.0d0,0.0d0/)
vec_xy = (/0.0d0,0.0d0,0.0d0/)
int_dist = 0.0d0
vec_srch = (/0.0d0,0.0d0,0.0d0/)
vox_dif = 0
ray_start = (/0.0d0,0.0d0,0.0d0/)

! preallocate arrays

! initialize variables [2]

! VERTICES
! locate voxel associated with each vertex
do i = 1, Nvrt ! loop through all vertices
  i_x = 1
  i_y = 1
  i_z = 1
  do j = 2, Nvxx ! loop through all x-voxels
    if ( vrt_crd(1,i) .ge. vox_lim_lo(1,j) ) then ! if the vertex coordinate is
greater than the voxel lower limit
      ! store the voxel x-index

```

```

        i_x = j
    end if
end do

do j = 2, Nvxy ! loop through all y-voxels
    if ( vrt_crd(2,i) .ge. vox_lim_lo(2,Nvxx*(j-1)+1) ) then ! if the vertex
coordinate is greater than the voxel lower limit
        ! store the voxel y-index
        i_y = j
    end if
end do

do j = 2, Nvxz ! loop through all z-voxels
    if ( vrt_crd(3,i) .ge. vox_lim_lo(3,Nvxx*Nvxy*(j-1)+1) ) then ! if the vertex
coordinate is greater than the voxel lower limit
        ! store the voxel z-index
        i_z = j
    end if
end do

! store the vertex ID in the voxel content list
vox_vrt(Nvxx*Nvxy*(i_z-1)+Nvxx*(i_y-1)+i_x)%arr = (/vox_vrt(Nvxx*Nvxy*(i_z-
1)+Nvxx*(i_y-1)+i_x)%arr,i/)

! assign voxel ID to vertex
vrt_vox(i) = Nvxx*Nvxy*(i_z-1)+Nvxx*(i_y-1)+i_x
end do

! EDGES
! locate voxels associated with each edge
do i = 1, Nedg
    ! get edge directionality code
    call EdgeDirectionCode(edg_tngt(1:3,i),edg_dir_code)

    ! determine start and stop voxel IDs
    vox_start = vrt_vox(edg_vrt(1,i))
    vox_stop = vrt_vox(edg_vrt(2,i))

    ! initialize current voxel
    vox_curr = vox_start

    ! append current voxel to current edge's voxel list
    edg_vox(i)%arr = (/edg_vox(i)%arr,vox_curr/)

    ! append current edge to current voxel's edge list
    vox_edg(vox_curr)%arr = (/vox_edg(vox_curr)%arr,i/)

    ! loop through voxels to get voxel list
    do while ( vox_curr .ne. vox_stop )
        call
RayVoxelIntersectionSearch(vrt_crd(1:3,edg_vrt(1,i)),vox_curr,edg_tngt(1:3,i),edg
_dir_code,vox_next)

        ! store intersected voxel as current
        vox_curr = vox_next
    end do
end do

```

```

        ! append current voxel to current edge's voxel list
        edg_vox(i)%arr = (/edg_vox(i)%arr,vox_curr/)

        ! append current edge to current voxel's edge list
        vox_edg(vox_curr)%arr = (/vox_edg(vox_curr)%arr,i/)
    end do
end do

! TRIANGLES
! locate voxels associated with each triangle
do i = 1, Ntri
    ! initialize triangle voxel list based on triangle edges
    tri_vox(i)%arr = edg_vox(tri_edg(1,i))%arr ! initialize triangle voxel list as
    first edge voxel list

    ! add additional voxels from second edge
    do j = 1, size(edg_vox(tri_edg(2,i))%arr(:)) ! loop through second edge voxels
        int_type = 0 ! flag that voxel doesn't exist in triangle list
        do m = 1, size(tri_vox(i)%arr(:)) ! loop through triangle voxels
            if ( tri_vox(i)%arr(m) .eq. edg_vox(tri_edg(2,i))%arr(j) ) then ! if
voxel exists
                int_type = 1
                exit
            end if
        end do

        if ( int_type .eq. 0 ) then ! if edge voxel isn't in triangle voxel list
            tri_vox(i)%arr = (/tri_vox(i)%arr,edg_vox(tri_edg(2,i))%arr(j)/) ! append
edge voxel to triangle voxel
        end if
    end do

    ! add additional voxels from third edge
    do j = 1, size(edg_vox(tri_edg(3,i))%arr(:)) ! loop through third edge voxels
        int_type = 0 ! flag that voxel doesn't exist in triangle list
        do m = 1, size(tri_vox(i)%arr(:)) ! loop through triangle voxels
            if ( tri_vox(i)%arr(m) .eq. edg_vox(tri_edg(3,i))%arr(j) ) then ! if
voxel exists
                int_type = 1
                exit
            end if
        end do

        if ( int_type .eq. 0 ) then ! if edge voxel isn't in triangle voxel list
            tri_vox(i)%arr = (/tri_vox(i)%arr,edg_vox(tri_edg(3,i))%arr(j)/) ! append
edge voxel to triangle voxel
        end if
    end do

    ! get search direction vector (parallel to second edge and towards triangle)
    if ( edg_vrt(2,tri_edg(2,i)) .eq. tri_vrt(3,i) ) then
        vec_srch(1:3) = edg_tngt(1:3,tri_edg(2,i)) ! edge 2 tangent vector
    else
        vec_srch(1:3) = -edg_tngt(1:3,tri_edg(2,i)) ! opposite edge 2 tangent vector
    end if
end do

```



```

end if

! get first edge directionality code
call EdgeDirectionCode(edg_tngt(1:3,tri_edg(1,i)),edg_dir_code)

! get search vector directionality code
call EdgeDirectionCode(vec_srch,edg_dir_code_srch)

! accumulate voxels in search direction
do j = 1, size(edg_vox(tri_edg(1,i))%arr(:)) - 1 ! loop through all edge 1
intersections with voxel limits
! store current edge 1 voxel
vox_curr = edg_vox(tri_edg(1,i))%arr(j)

pnv = (/0.0d0,0.0d0,0.0d0/)
pp0 = (/0.0d0,0.0d0,0.0d0/)

! determine which interface (x, y, or z) is intersected by edge 1 current and
next voxel
vox_dif = edg_vox(tri_edg(1,i))%arr(j+1) - vox_curr ! difference between
edge voxel IDs

if ( ( abs(vox_dif)/(Nvxx*Nvxy) .ge. 1 ) .and. (
dmod(real(abs(vox_dif),8),real(Nvxx*Nvxy,8)) .eq. 0.0d0 ) ) then ! move is in z-
dimension
pnv = (/0.0d0,0.0d0,1.0d0/) ! define normal vector to xy plane

! check z-directionality of edge code: edg_dir_code(3)
if ( edg_dir_code(3) .gt. 0 ) then ! if positive current voxel hi limit
pp0 = (/0.0d0,0.0d0,vox_lim_hi(3,vox_curr)/)
elseif ( edg_dir_code(3) .lt. 0 ) then ! if negative current voxel lo limit
pp0 = (/0.0d0,0.0d0,vox_lim_lo(3,vox_curr)/)
else
write(*,*) "ERROR: Search direction has no defined directionality."
stop
end if
elseif ( ( abs(vox_dif)/Nvxx .ge. 1 ) .and. (
dmod(real(abs(vox_dif),8),real(Nvxx,8)) .eq. 0.0d0 ) ) then ! move is in y-
dimension
pnv = (/0.0d0,1.0d0,0.0d0/) ! define normal vector to zx plane

! check y-directionality of edge code: edg_dir_code(2)
if ( edg_dir_code(2) .gt. 0 ) then ! if positive current voxel hi limit
pp0 = (/0.0d0,vox_lim_hi(2,vox_curr),0.0d0/)
elseif ( edg_dir_code(2) .lt. 0 ) then ! if negative current voxel lo limit
pp0 = (/0.0d0,vox_lim_lo(2,vox_curr),0.0d0/)
else
write(*,*) "ERROR: Search direction has no defined directionality."
stop
end if
else ! move is in x-dimension
pnv = (/1.0d0,0.0d0,0.0d0/) ! define normal vector to yz plane

! check x-directionality of edge code: edg_dir_code(1)
if ( edg_dir_code(1) .gt. 0 ) then ! if positive current voxel hi limit

```

```

        pp0 = (/vox_lim_hi(1,vox_curr),0.0d0,0.0d0/)
    elseif ( edg_dir_code(1) .lt. 0 ) then ! if negative current voxel lo limit
        pp0 = (/vox_lim_lo(1,vox_curr),0.0d0,0.0d0/)
    else
        write(*,*) "ERROR: Search direction has no defined directionality."
        stop
    end if
end if

call
LinePlaneIntersection(edg_tngt(1:3,tri_edg(1,i)),vrt_crd(1:3,edg_vrt(1,tri_edg(1,
i))),pntv,pp0,FLAG_LPX_xy,ray_start) ! get edge-plane intersection point

! search for voxels until edge 3 voxel is hit
int_type = 0 ! initialize flag to indicate edge 3 voxel is intersected

! check if edge 3 is in edge 1 start voxel
do m = 1, size(vox_edg(vox_curr)%arr(:)) ! loop through all edges in
current voxel
    if ( tri_edg(3,i) .eq. vox_edg(vox_curr)%arr(m) ) then ! if voxel edge is
same as edge 3
        int_type = 1 ! flag that edge 3 voxel has been found
        exit
    end if
end do

do while ( int_type .eq. 0 ) ! search voxels in edge 2 direction
    call
RayVoxelIntersectionSearch(ray_start,vox_curr,vec_srch,edg_dir_code_srch,vox_next
) ! get next voxel in search direction

    vox_curr = vox_next ! store next voxel as current voxel

    ! check if next voxel is in edge 3
    do m = 1, size(edg_vox(tri_edg(3,i))%arr(:)) ! loop through all voxels of
edge 3
        if ( edg_vox(tri_edg(3,i))%arr(m) .eq. vox_curr ) then ! if in edge 3,
stop searching
            int_type = 1 ! flag as in edge 3
            exit
        end if
    end do

    if ( int_type .eq. 0 ) then ! if voxel not in edge 3, check if in
triangle
        FLAG_LPX_xy = 0 ! initialize as not in triangle

        do n = 1, size(tri_vox(i)%arr(:)) ! loop through all voxels in triangle
            if ( vox_curr .eq. tri_vox(i)%arr(n) ) then ! if voxel is already
in triangle, stop looping
                FLAG_LPX_xy = 1 ! flag as in triangle
                exit ! stop looping
            end if
        end do
    end if
end do

```

```

        if ( FLAG_LPX_xy .eq. 0 ) then ! if not in triangle, add voxel to
triangle list
            tri_vox(i)%arr = (/tri_vox(i)%arr,vox_curr/) ! add voxel to
triangle list
        end if
    end if
end do

! add triangle to voxels
do j = 1, size(tri_vox(i)%arr(:))
    vox_tri(tri_vox(i)%arr(j))%arr = (/vox_tri(tri_vox(i)%arr(j))%arr,tri_ID(i)/)
end do

end do

end subroutine FindVoxelContents

subroutine GatherLoadVertices
! This subroutine makes a list of all vertices with an applied load, currently
only addressing
! flux loads.

use AllModules
implicit none

! variables
real(8) :: area_tri3 ! 1/3 triangle area
integer :: FLAG_FOUND ! flag for finding vertex location (0 = not found, 1 =
found)
integer :: i ! index
integer :: j ! index
integer :: j_cnt ! continuum ID
integer :: m ! index
integer :: m_tri ! triangle ID
integer :: n ! index
integer :: n_prev ! index placeholder
integer :: vrt_1 ! triangle lowest vertex ID
integer, dimension(1:3) :: vrt_123 ! triangle ordered vertex IDs
integer :: vrt_123_curr ! placeholder for current ordered vertex position
integer :: vrt_2 ! triangle median vertex ID
integer :: vrt_3 ! triangle highest vertex ID

! initialize variables [1]
area_tri3 = 0.0d0
FLAG_FOUND = 0
i = 0
j = 0
j_cnt = 0
n = 0
m_tri = 0
vrt_1 = 0
vrt_123 = (/0,0,0/)
vrt_123_curr = 0
vrt_2 = 0

```

```

vrt_3 = 0

! preallocate arrays
do i = 1, Nlod
  allocate(lod_cntvrt(i)%arr(0))
  allocate(lod_cvrtarea(i)%arr(0))
  allocate(lod_cvrtlod(i)%arr(0))
  allocate(lod_linvrt(i)%arr(0))
  allocate(lod_lvrtlod(i)%arr(0))
end do

! initialize variables [2]

! gather vertices
! loop through all defined loads
do i = 1, Nlod

  ! gather vertices depending on load type
  select case ( lod_type(i) )
    case ( -1 ) ! unassigned
    case ( 0 ) ! absolute, (i.e., W)
    case ( 1 ) ! line, (i.e., W/m)
    case ( 2 ) ! flux, (i.e., W/m2)

    ! loop through receiving continuums
    do j = 1, size(lod_cnt(i)%arr(:))
      j_cnt = lod_cnt(i)%arr(j) ! store continuum ID

      ! loop through triangles in continuum
      do m = 1, size(cnt_tri(j_cnt)%arr(:))
        m_tri = cnt_tri(j_cnt)%arr(m) ! store continuum current triangle
ID
        area_tri3 = tri_area(m_tri)/3.0d0 ! store 1/3 current triangle
area
        vrt_1 = minval( tri_vrt(1:3,m_tri) ) ! triangle vertex with lowest
ID
        vrt_3 = maxval( tri_vrt(1:3,m_tri) ) ! triangle vertex with
highest ID

        ! find the triangle vertex with the median ID
        if ( ( tri_vrt(1,m_tri) > vrt_1 ) .and. ( tri_vrt(1,m_tri) < vrt_3
) ) then
          vrt_2 = tri_vrt(1,m_tri)
        else if ( ( tri_vrt(2,m_tri) > vrt_1 ) .and. ( tri_vrt(2,m_tri) <
vrt_3 ) ) then
          vrt_2 = tri_vrt(2,m_tri)
        else if ( ( tri_vrt(3,m_tri) > vrt_1 ) .and. ( tri_vrt(3,m_tri) <
vrt_3 ) ) then
          vrt_2 = tri_vrt(3,m_tri)
        end if
      end do
    end do
  end select
end do

```

```

vrt_123 = (/vrt_1,vrt_2,vrt_3/) ! store ordered triangle vertex
IDs

    if ( size(lod_cntvrt(i)%arr(:)) .eq. 0 ) then ! continuum first
triangle - manually assign vertex IDs and vertex areas if vertex list is empty
        ! manually build the continuum vertex ID list
        lod_cntvrt(i)%arr = (/lod_cntvrt(i)%arr,vrt_1,vrt_2,vrt_3/)

        ! manually build the continuum vertex area list
        lod_cvrtarea(i)%arr =
(/lod_cvrtarea(i)%arr,area_tri3,area_tri3,area_tri3/)

    else ! otherwise assign vertex IDs and vertex areas automatically
        ! loop through existing vertex list to find where
        FLAG_FOUND = 0
        n_prev = 0
        vrt_123_curr = 1

        n = 1 ! look at first vertex in list
        if ( vrt_1 < lod_cntvrt(i)%arr(n) ) then ! if lowest vertex ID
is lower than current vertex in list then insert ID and area
            ! insert ID
            lod_cntvrt(i)%arr = (/vrt_1,lod_cntvrt(i)%arr/)

            ! insert area
            lod_cvrtarea(i)%arr = (/area_tri3,lod_cvrtarea(i)%arr/)

            n_prev = n ! store the location of the new vertex
            vrt_123_curr = 2 ! start looking at median vertex ID
        else if ( vrt_1 .eq. lod_cntvrt(i)%arr(n) ) then ! if lowest
vertex ID is equal to current vertex in list then add area only
            ! add area
            lod_cvrtarea(i)%arr(n) = lod_cvrtarea(i)%arr(n) + area_tri3

            n_prev = n ! store the location of the new vertex
            vrt_123_curr = 2 ! start looking at median vertex ID
        end if

        ! remaining vertices in list
        do while ( vrt_123_curr < 4 )
            FLAG_FOUND = 0

            do n = n_prev + 1, size(lod_cntvrt(i)%arr(:)) ! loop through
remaing vertices in list
                if ( vrt_123(vrt_123_curr) < lod_cntvrt(i)%arr(n) ) then !
if lowest vertex ID is lower than current vertex in list then insert ID and area
                    ! insert ID
                    lod_cntvrt(i)%arr = (/lod_cntvrt(i)%arr(1:n-
1),vrt_123(vrt_123_curr),lod_cntvrt(i)%arr(n:size(lod_cntvrt(i)%arr(:))))/)

                    ! insert area
                    lod_cvrtarea(i)%arr = (/lod_cvrtarea(i)%arr(1:n-
1),area_tri3,lod_cvrtarea(i)%arr(n:size(lod_cvrtarea(i)%arr(:))))/)

                    n_prev = n ! store the location of the new vertex

```

```

                                vrt_123_curr = vrt_123_curr + 1 ! start looking at
next vertex ID
                                FLAG_FOUND = 1
                                exit ! stop searching
                                else if ( vrt_123(vrt_123_curr) .eq. lod_cntvrt(i)%arr(n)
) then ! if lowest vertex ID is equal to current vertex in list then add area
only
                                ! add area
                                lod_cvrtarea(i)%arr(n) = lod_cvrtarea(i)%arr(n) +
area_tri3

                                n_prev = n ! store the location of the new vertex
                                vrt_123_curr = vrt_123_curr + 1 ! start looking at
next vertex ID
                                FLAG_FOUND = 1
                                exit ! stop searching
                                end if
                                end do

                                if ( FLAG_FOUND .eq. 1 ) cycle ! continue looking at median
vertices

                                n = size(lod_cntvrt(i)%arr(:)) ! last vertex in list
                                if ( vrt_123(vrt_123_curr) > lod_cntvrt(i)%arr(n) ) then ! if
lowest vertex ID is higher than current vertex in list then append ID and area
                                ! append ID
                                lod_cntvrt(i)%arr =
(/lod_cntvrt(i)%arr,vrt_123(vrt_123_curr)/)

                                ! append area
                                lod_cvrtarea(i)%arr = (/lod_cvrtarea(i)%arr,area_tri3/)

                                FLAG_FOUND = 1 ! indicate location has been found
                                n_prev = n + 1 ! store the location of the new vertex
                                vrt_123_curr = vrt_123_curr + 1 ! start looking at next
vertex ID

                                else ! vertex hasn't been assigned. This case should never
happen.

                                ! ERROR
                                write(*,*) "ERROR! : CONTINUUM VERTEX NOT ASSIGNED FOR
LOAD, TYPE FLUX"

                                stop ! stop the program
                                end if

                                end do
                                end if
                                end do
                                end do
                                case ( 3 ) ! volumetric, (i.e., W/m3)
                                end select
                                end do

                                !~! debug display
                                !~do i = 1, Nlod
                                !~ call Displayload(lod_ID(i))

```

```

!~end do

end subroutine GatherLoadVertices

subroutine IdentifyConstantVertices
! This subroutine makes a list of all vertices with defined or unchanging
temperatures, and
! it makes a list of all vertices with temperatures that must be solved.

use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: cyc_flag ! cycle flag

allocate(VrtSolv(0))
allocate(VrtNoSolv(0))

do i = 1, Nvrt ! loop through all vertices
    cyc_flag = 0

    if ( Gv_x(i) + Gv_y(i) + Gv_z(i) .eq. 0.0d0 ) then ! if vertex has 0 selfsame
conductance, don't solve
        ! if vertex gets filtered, append ID to no solve list
        VrtNoSolv = (/VrtNoSolv,i/)
        cycle
    end if

    if ( size(vrt_bnd(i)%arr(:)) > 0 ) then ! if vertex is on one or more
boundaries
        do j = 1, size(vrt_bnd(i)%arr(:)) ! loop through all boundaries associated
with vertex
            if ( bnd_type(vrt_bnd(i)%arr(j)) .eq. 1 ) then ! if vertex is on Dirichlet
boundary, don't solve
                cyc_flag = 1
                ! if vertex gets filtered, append ID to no solve list
                VrtNoSolv = (/VrtNoSolv,i/)
                exit
            end if
        end do
    end if
    if ( cyc_flag .eq. 1 ) then
        cycle
    end if

    ! if vertex passes filters, append ID to solve list
    VrtSolv = (/VrtSolv,i/)
end do

end subroutine IdentifyConstantVertices

subroutine ImportMesh
! This subroutine imports mesh information from Gmsh NASTRAN formatting.

```

```

use AllModules
implicit none
! import NASTRAN mesh from Gmsh
! NOTE: When importing from a Gmsh NASTRAN file, it is possible
! to obtain boundaries and continuums besides vertices, edges,
! triangles, and tetrahedra

! can probably crudely link MATLAB or Python to code using text files for inputs

! user-defined data structures

! variables
integer :: bndID ! input boundary ID
integer, allocatable, dimension(:) :: BndOrgID ! original boundary ID from input
source
integer :: bndv1 ! input boundary first vertex ID
integer :: bndv2 ! input boundary second vertex ID
integer :: cntmID ! input triangle continuum ID
integer :: cpID ! input vertex coordinate system
integer :: FLAG_APPEND ! flag to append new boundary (0=don't append; 1=append)
real(8) :: dx ! edge x length
real(8) :: dy ! edge y length
real(8) :: dz ! edge z length
integer :: elID ! input geometric element source element ID
integer :: i ! index
character(len=500) :: InputLine ! input file line text
integer :: ios ! I/O status flag
integer :: j ! index
integer :: LineLength ! number of characters in input file line
integer :: m ! index
integer :: maxID ! maximum vertex ID from triangle vertex pair
integer :: minID ! minimum vertex ID from triangle vertex pair
real(8) :: SumTriA ! sum of triangle areas
integer, allocatable, dimension(:) :: TriCntmID ! list of original continuum IDs
integer, dimension(1:5) :: TriVrtPr ! triangle vertex pairing list
real(8), dimension(1:3) :: vec1 ! first vector (xyz components)
real(8), dimension(1:3) :: vec2 ! second vector (xyz components)
real(8), dimension(1:3) :: vec3 ! third vector (xyz components)
integer :: vrtID_elist ! first edges vertex ID marker
integer :: vrtID1 ! input triangle first vertex ID
integer :: vrtID2 ! input triangle second vertex ID
integer :: vrtID3 ! input triangle third vertex ID
integer, allocatable, dimension(:) :: VrtOrgID ! original vertex ID from input
source
real(8) :: xcrd ! input vertex x-coordinate
real(8), dimension(1:3) :: xprdc ! vector cross product
real(8) :: ycrd ! input vertex y-coordinate
real(8) :: zcrd ! input vertex z-coordinate
integer, allocatable, dimension(:) :: zzzTrshVrtIdLst ! temporary vertex
connectivity ID list
integer :: zzzTrshInt ! temporary integer placeholder
integer, allocatable, dimension(:) :: zzzTrshIntArray ! temporary integer array
placeholder
real(8) :: zzzTrshRl ! temporary real placeholder

```



```

real(8), allocatable, dimension(:, :) :: zzzTrshRlArry2D ! temporary 2D real array
placeholder
integer, allocatable, dimension(:, :) :: zzzTrshIntArry2D ! temporary 2D integer
array placeholder
type(AllIntArr1D), allocatable, dimension(:) :: zzzTrshAllIntArry1D ! temporary
1D array of integer allocatable placeholder
type(AllRlArr1D), allocatable, dimension(:) :: zzzTrshAllRlArry1D ! temporary 1D
array of real allocatable placeholder

character(len=500) :: trashfile ! trash output file
character(len=500) :: string ! dummy string

call DisplayActionStamp('IMPORT MESH START')

! intialize variables [1]
bndID = 0
bndv1 = 0
bndv2 = 0
cntmID = 0
cpID = 0
dx = 0.0d0
dy = 0.0d0
dz = 0.0d0
elID = 0
FLAG_APPEND = 0
i = 0
InputLine = ""
ios = 0
j = 0
LineLength = 0
m = 0
maxID = 0
minID = 0
Nbar = 0
Nbnd = 0
Ncnt = 0
Nedg = 0
Nel1 = 0
Nel2 = 0
Nel3 = 0
Ntet = 0
Ntri = 0
Nvrt = 0
SumTriA = 0.0d0
trashfile = "TrashFile.txt"
TriVrtPr = (/1,2,3,1,2/)
vec1 = 0.0d0
vec2 = 0.0d0
vec3 = 0.0d0
vrtID_estart = 0
vrtID1 = 0
vrtID2 = 0
vrtID3 = 0
xcrd = 0.0d0
ycrd = 0.0d0

```

```

zcrd = 0.0d0
zzzTrshRl = 0.0d0

! preallocate arrays

allocate(bnd_ID(0))
allocate(bnd_evrt(0))
allocate(bnd_name(0))

allocate(BndOrgID(0))

allocate(cnt_tri(0))
allocate(cnt_ID(0))

allocate(tri_ID(0))
allocate(tri_vrt(1:3,0))
allocate(tri_cntm(0))

allocate(TriCntmID(0))

allocate(vrt_crd(1:3,0))

allocate(VrtCon(0))
allocate(VrtEdg(0))

allocate(vrt_ID(0))

allocate(VrtOrgID(0))
allocate(zzzTrshVrtIdLst(1:1))
allocate(zzzTrshIntArray(1:1))

! initialize variables [2]
zzzTrshIntArray = 0

! read each line of the input file
open(unit=1,file=MeshInputFileName,action='read',iostat=ios) ! open geometry
input file

! take action, check to make sure file opens correctly

write(string,'(A,i0)') "Mesh Input File I/O Error Status Flag: ",ios
call DisplayLine01(trim(string))
if ( ios .ne. 0 ) then ! if error status is not 0
    write(string,'(A,i0)') "Mesh Input File I/O Error Status Flag: ",ios
    call DisplayLine01(trim(string))
    stop ! end
end if

InputLine = "" ! ensure input line content is cleared before reading from file
ios = 0 ! ensure I/O flag initialized before importing geometry from file
do while ( ios .eq. 0 )
    if ( FLAG_disp_inpt == 1 ) then
        write(*,*) trim(InputLine)
    end if
    LineLength = len(trim(InputLine)) ! get length of current input file line

```

```

    if ( LineLength .gt. 6 ) then ! make sure line has enough characters to be
valid
    !      GRID      GRID      GRID      GRID      GRID      GRID      GRID      GRID
GRID
    if ( InputLine(1:4) .eq. "GRID" ) then ! line is a vertex
        if ( FLAG_disp_inpt == 1 ) then
            write(*,*) InputLine(1:4)
        end if
        read(InputLine(6:LineLength),'(2i,3f)') elID,cpID,xcrd,ycrd,zcrd ! read in
input line data
        Nvrt = Nvrt + 1 ! increase vertex count

        ! append the new/original vertex IDs to repsective lists
        vrt_ID = (/vrt_ID,Nvrt/)
        VrtOrgID = (/VrtOrgID,elID/)

        ! save existing vertex coordinates
        allocate(zzzTrshRlArry2D(1:3,Nvrt-1))
        zzzTrshRlArry2D = vrt_crd

        ! resize vertex coordinates
        deallocate(vrt_crd)
        allocate(vrt_crd(1:3,1:Nvrt))
        vrt_crd(1:3,1:Nvrt-1) = zzzTrshRlArry2D
        vrt_crd(1:3,Nvrt) = (/xcrd,ycrd,zcrd/)

        ! clear vertex coordinate placeholde
        deallocate(zzzTrshRlArry2D)

    !      CBAR      CBAR      CBAR      CBAR      CBAR      CBAR      CBAR      CBAR
CBAR
    else if ( InputLine(1:4) .eq. "CBAR" ) then ! line is a boundary
        read(InputLine(6:LineLength),'(4i,3f)')
        elID,bndID,bndv1,bndv2,xcrd,ycrd,zcrd ! read in input line data

        FLAG_APPEND = 1 ! assume need to append
        do i = 1, Nbnd ! loop through existing boundary IDs
            if ( bndID < bnd_ID(i) ) then ! if new ID is less than current, insert
and exit
                Nbnd = Nbnd + 1 ! increment boundary count

                bnd_ID = (/bnd_ID,bndID/) ! temporarily append boundary ID to end
of ID list
                bnd_ID(i:Nbnd) = (/bndID,bnd_ID(i:Nbnd-1)/) ! exchange the
original latter portion of the boundary ID list with the new boundary ID

                allocate(zzzTrshAllIntArry1D(1:Nbnd-1))
                zzzTrshAllIntArry1D = bnd_evrt

                deallocate(bnd_evrt)
                allocate(bnd_evrt(1:Nbnd))

                bnd_evrt(1:Nbnd-1) = zzzTrshAllIntArry1D
                bnd_evrt(Nbnd)%arr =
(/minval((/bndv1,bndv2/)),maxval((/bndv1,bndv2/)))

```

```

        bnd_evrt(i:Nbnd) = (/bnd_evrt(Nbnd),bnd_evrt(i:Nbnd-1)/)

        deallocate(zzzTrshAllIntArray1D)

        FLAG_APPEND = 0 ! don't need to append
        exit
    else if ( bndID .eq. bnd_ID(i) ) then ! else if equal, retain vertice
IDs, exit
        bnd_evrt(i)%arr =
(/bnd_evrt(i)%arr,minval(/bndv1,bndv2/)),maxval(/bndv1,bndv2/)/)

        FLAG_APPEND = 0 ! don't need to append
        exit
    end if
end do

if ( FLAG_APPEND .eq. 1 ) then ! if need to append, append
Nbnd = Nbnd + 1 ! increment boundary count
bnd_ID = (/bnd_ID,bndID/) ! append boundary ID to end of ID list

allocate(zzzTrshAllIntArray1D(1:Nbnd-1))
zzzTrshAllIntArray1D = bnd_evrt

deallocate(bnd_evrt)
allocate(bnd_evrt(1:Nbnd))

bnd_evrt(1:Nbnd-1) = zzzTrshAllIntArray1D
bnd_evrt(Nbnd)%arr =
(/minval(/bndv1,bndv2/)),maxval(/bndv1,bndv2/)/)

deallocate(zzzTrshAllIntArray1D)
end if

!      CTRIA3      CTRIA3      CTRIA3      CTRIA3      CTRIA3      CTRIA3
CTRIA3      CTRIA3
    else if ( InputLine(1:6) .eq. "CTRIA3" ) then ! line is a triangle
        read(InputLine(8:LineLength),'(5i)') elID,cntmID,vrtID1,vrtID2,vrtID3 !
read in input line data
        Ntri = Ntri + 1 ! increase triangle count

        ! append the new triangle IDs to list
        tri_ID = (/tri_ID,Ntri/)

        ! append the new triangle continuum to list
        tri_cntm = (/tri_cntm,cntmID/)

        ! save existing triangle vertices
        allocate(zzzTrshIntArray2D(1:3,Ntri-1))
        zzzTrshIntArray2D = tri_vrt

        ! resize triangle vertices
        deallocate(tri_vrt)
        allocate(tri_vrt(1:3,1:Ntri))
        tri_vrt(1:3,1:Ntri-1) = zzzTrshIntArray2D

```

```

tri_vrt(1:3,Ntri) = (/vrtID1,vrtID2,vrtID3/)

! clear vertex coordinate placeholder
deallocate(zzzTrshIntArray2D)

FLAG_APPEND = 1 ! assume need to append
do i = 1, Ncnt ! loop through existing continuum IDs
  if ( cntmID < TriCntmID(i) ) then ! if new ID is less than current,
insert and exit
    Ncnt = Ncnt + 1 ! increment continuum count
    TriCntmID = (/TriCntmID,cntmID/) ! temporarily append continuum ID
to end of ID list
    TriCntmID(i:Ncnt) = (/cntmID,TriCntmID(i:Ncnt-1)/) ! exchange the
original latter portion of the continuum ID list with the new continuum ID

    FLAG_APPEND = 0 ! don't need to append
    exit
  else if ( cntmID .eq. TriCntmID(i) ) then ! else if equal, exit
    FLAG_APPEND = 0 ! don't need to append
    exit
  end if
end do

if ( FLAG_APPEND .eq. 1 ) then ! if need to append, append
  Ncnt = Ncnt + 1 ! increment continuum count
  TriCntmID = (/TriCntmID,cntmID/) ! append cointinuum ID to end of ID
list
end if
end if
end if
read(1,'(A)',iostat=ios) InputLine ! get file line
end do

close(1) ! close geometry input file

allocate(bnd_tvrt(0))
allocate(bnd_type(1:Nbnd))
allocate(bnd_func(1:Nbnd))
bnd_type(1:Nbnd) = 0
bnd_func(1:Nbnd) = 0

! change vertex ID in triangle list
do i = 1, Nvrt
  where( tri_vrt(1:3,1:Ntri) .eq. VrtOrgID(i) )
    tri_vrt(1:3,1:Ntri) = -i
  end where
end do

tri_vrt(1:3,1:Ntri) = -tri_vrt(1:3,1:Ntri)

! change vertex ID in boundary list
do i = 1, Nvrt
  do j = 1, Nbnd
    where( bnd_evrt(j)%arr(:) .eq. VrtOrgID(i) )
      bnd_evrt(j)%arr(:) = -i
    end where
  end do
end do

```

```

        end where
    end do
end do
do i = 1, Nbnd
    bnd_evrt(i)%arr(:) = -bnd_evrt(i)%arr(:)
end do

! get original boundary ID list and relabel ID
deallocate(BndOrgID)
allocate(BndOrgID(1:Nbnd))
deallocate(bnd_name)
allocate(bnd_name(1:Nbnd))
do i = 1, Nbnd
    BndOrgID(i) = bnd_ID(i)
    bnd_ID(i) = i
    write(bnd_name(i), '(A,i0)') 'Boundary_', bnd_ID(i)
end do

! change continuum ID in triangle list
do i = 1, Ncnt
    where ( tri_cntm(1:Ntri) .eq. TriCntmID(i) )
        tri_cntm(1:Ntri) = -i
    end where
end do
tri_cntm(1:Ntri) = -tri_cntm(1:Ntri)

! create vertex connectivity list (precursor to edge list)
deallocate(VrtCon)
allocate(VrtCon(1:Nvrt))
do i = 1, Nvrt
    allocate(VrtCon(i)%arr(0))
end do

do i = 1, Ntri

    !~write(*,*) 'TRIANGLE ', i
    do m = 1, 3
        !~ write(*,*) ' SIDE ', m
        minID = minval(tri_vrt(TriVrtPr(m:m+1),i)) ! store the lowest vertex ID from
the first pair
        maxID = maxval(tri_vrt(TriVrtPr(m:m+1),i)) ! store the highest vertex ID from
the first pair

        !~write(*,*) 'IDs ', minID, '/', maxID
        !~write(*,*) ' size of iter ', size(VrtCon(minID)%arr(:))

        FLAG_APPEND = 1 ! assume need to append
        do j = 1, size(VrtCon(minID)%arr(:))
            if ( maxID < VrtCon(minID)%arr(j) ) then ! if new ID is less than
current, insert and exit
                !~ write(*,*) 'maxID ', maxID, ' less than ', VrtCon(minID)%arr(j)

                allocate(zzzTrshAllIntArray1D(1))
                allocate(zzzTrshAllIntArray1D(1)%arr(1:size(VrtCon(minID)%arr(:))))

```

```

      zzzTrshAllIntArray1D(1)%arr(:) = VrtCon(minID)%arr(:)

      deallocate(VrtCon(minID)%arr)
      allocate(VrtCon(minID)%arr(1:size(zzzTrshAllIntArray1D(1)%arr(:))+1))

      VrtCon(minID)%arr(1:j-1) = zzzTrshAllIntArray1D(1)%arr(1:j-1)

      VrtCon(minID)%arr(j) = maxID

      VrtCon(minID)%arr(j+1:size(zzzTrshAllIntArray1D(1)%arr(:))+1) =
      zzzTrshAllIntArray1D(1)%arr(j:size(zzzTrshAllIntArray1D(1)%arr(:)))

      deallocate(zzzTrshAllIntArray1D)

      FLAG_APPEND = 0 ! don't need to append
      exit
    else if ( maxID .eq. VrtCon(minID)%arr(j) ) then ! else if equal, exit
      FLAG_APPEND = 0 ! don't need to append
      exit
    end if
  end do

  if ( FLAG_APPEND .eq. 1 ) then ! if need to append, append

    allocate(zzzTrshAllIntArray1D(1))
    allocate(zzzTrshAllIntArray1D(1)%arr(1:size(VrtCon(minID)%arr(:))))

    zzzTrshAllIntArray1D(1)%arr(:) = VrtCon(minID)%arr(:)

    deallocate(VrtCon(minID)%arr)
    allocate(VrtCon(minID)%arr(1:size(zzzTrshAllIntArray1D(1)%arr(:))+1))

    VrtCon(minID)%arr(1:size(zzzTrshAllIntArray1D(1)%arr(:))) =
    zzzTrshAllIntArray1D(1)%arr(:) ! append max ID to end of ID list
    VrtCon(minID)%arr(size(zzzTrshAllIntArray1D(1)%arr(:))+1) = maxID

    deallocate(zzzTrshAllIntArray1D)
  end if
end do

! set sizes of vertex edge list
VrtEdg = VrtCon

! allocate size of edge structures
do i = 1, Nvrt
  do j = 1, size(VrtCon(i)%arr(:))
    VrtEdg(i)%arr(j) = Nedg + j
  end do
  Nedg = Nedg + size(VrtCon(i)%arr(:))
end do
allocate(edg_ID(1:Nedg))
allocate(edg_vrt(1:2,1:Nedg))
allocate(edg_bnd(1:Nedg))
allocate(edg_adjvrt(1:Nedg))

```

```

allocate(edg_adjtri(1:Nedg))
allocate(edg_nrmlx(1:Nedg))
allocate(edg_nrmlly(1:Nedg))
allocate(edg_nrmlz(1:Nedg))
allocate(edg_vox(1:Nedg))
do i = 1, Nedg
    allocate(edg_nrmlx(i)%arr(0))
    allocate(edg_nrmlly(i)%arr(0))
    allocate(edg_nrmlz(i)%arr(0))
    allocate(edg_vox(i)%arr(0))
end do

! allocate size of vertex structures
allocate(vrt_adjvrt(1:Nvrt))
allocate(vrt_adjedg(1:Nvrt))
allocate(vrt_adjtri(1:Nvrt))
do i = 1, Nvrt
    allocate(vrt_adjvrt(i)%arr(0))
    allocate(vrt_adjedg(i)%arr(0))
    allocate(vrt_adjtri(i)%arr(0))
end do
allocate(vrt_res(1:Nvrt))
allocate(vrt_mfe(1:Nvrt))
allocate(vrt_area(1:Nvrt))
vrt_area(1:Nvrt) = 0.0d0
allocate(vrt_vol(1:Nvrt))
vrt_vol(1:Nvrt) = 0.0d0
allocate(vrt_vox(1:Nvrt))
allocate(vrt_tmp(1:Nvrt))
allocate(vrt_tmppm(1:Nvrt))
allocate(pmuilt(1:Nvrt))
vrt_tmp(1:Nvrt) = -273.0d0
vrt_tmppm(1:Nvrt) = -273.0d0
vrt_vox(1:Nvrt) = 0

! find edges from vertex connectivity list
do i = 1, Nvrt
    do j = 1, size(VrtCon(i)%arr(:))
        ! define edges
        edg_vrt(1:2,VrtEdg(i)%arr(j)) = (/vrt_ID(i),VrtCon(i)%arr(j)/) ! define edge
vertices based on connectivity list
        edg_ID(VrtEdg(i)%arr(j)) = VrtEdg(i)%arr(j) ! defined edge ID

        ! add vertices to vertex adjacent vertices list
        vrt_adjvrt(i)%arr = (/vrt_adjvrt(i)%arr,VrtCon(i)%arr(j)/)
        vrt_adjvrt(VrtCon(i)%arr(j))%arr =
(/vrt_adjvrt(VrtCon(i)%arr(j))%arr,vrt_ID(i)/)

        ! add edges to vertex adjacent edges list
        vrt_adjedg(i)%arr = (/vrt_adjedg(i)%arr,edg_ID(VrtEdg(i)%arr(j))/)
        vrt_adjedg(VrtCon(i)%arr(j))%arr =
(/vrt_adjedg(VrtCon(i)%arr(j))%arr,edg_ID(VrtEdg(i)%arr(j))/)
    end do
end do

```



```

! allocate boundary edge lists
allocate(bnd_edg(1:Nbnd))

! get boundary edges from boundary edge vertex list
do i = 1, Nbnd
  allocate(bnd_edg(i)%arr(0)) ! allocate boundary edge list

  do j = 1, size(bnd_evrt(i)%arr(:)), 2 ! loop through vertices in boundary edge
vertices list
    do m = 1, size(VrtCon(bnd_evrt(i)%arr(j))%arr(:)) ! loop through vertices
connected to current edge vertex
      if ( bnd_evrt(i)%arr(j+1) .eq. VrtCon(bnd_evrt(i)%arr(j))%arr(m) ) then !
if next vertex in boundary edge vertex list is same as edge connectivity
        bnd_edg(i)%arr = (/bnd_edg(i)%arr,VrtEdg(bnd_evrt(i)%arr(j))%arr(m)/) !
add associated edge from VrtEdg (connectivity list) to bnd_edg
        exit
      end if
    end do
  end do
end do

! initialize edge boundary identifier
edg_bnd(1:Nedg) = 0

! reorder boundary edge list
do i = 1, Nbnd
  !write(*,*) 'Boundary ', i
  !write(*,*) ' N edges ', size(bnd(i)%edg(:))
  do j = 1, size(bnd_edg(i)%arr(:))-1
    !write(*,*) ' Edge element ', j
    do m = j + 1, size(bnd_edg(i)%arr(:))
      ! if mth element is smaller than jth element, insert mth element and
shift
      if ( bnd_edg(i)%arr(m) < bnd_edg(i)%arr(j) ) then
        zzzTrshInt = bnd_edg(i)%arr(m) ! store mth element

        bnd_edg(i)%arr(j+1:m) = bnd_edg(i)%arr(j:m-1) ! shift jth to m-1th
subarray
        bnd_edg(i)%arr(j) = zzzTrshInt ! insert mth element at jth position
      end if
    end do
  end do

  ! assign boundary identifier to edges
  do j = 1, size(bnd_edg(i)%arr(:))
    edg_bnd(bnd_edg(i)%arr(j)) = i
  end do
end do

! allocate vertex boundary ID list
allocate(vrt_bnd(1:Nvrt))
do i = 1, Nvrt
  allocate(vrt_bnd(i)%arr(0))
end do
deallocate(bnd_tvrt)

```

```

allocate(bnd_tvrt(1:Nbnd))
do i = 1, Nbnd
  allocate(bnd_tvrt(i)%arr(0))
end do

! reorder boundary vertex list (and remove repeated vertices)
do i = 1, Nbnd
  !write(*,*) 'Boundary ', i
  !write(*,*) ' N edges ', size(bnd(i)%edg(:))
  do j = 1, size(bnd_evrt(i)%arr(:))-1
    !write(*,*) ' Edge element ', j
    do m = j + 1, size(bnd_evrt(i)%arr(:))
      ! if mth element is smaller than jth element, insert mth element and
shift
      if ( bnd_evrt(i)%arr(m) < bnd_evrt(i)%arr(j) ) then
        zzzTrshInt = bnd_evrt(i)%arr(m) ! store mth element
        bnd_evrt(i)%arr(j+1:m) = bnd_evrt(i)%arr(j:m-1) ! shift jth to m-1th
subarray
        bnd_evrt(i)%arr(j) = zzzTrshInt ! insert mth element at jth position
      end if
    end do
  end do

  j = 1
  do while ( j < size(bnd_evrt(i)%arr(:)) )

    m = j + 1

    do while ( m <= size(bnd_evrt(i)%arr(:)) )
      if ( bnd_evrt(i)%arr(j) .eq. bnd_evrt(i)%arr(m) ) then
        if ( m .ne. size(bnd_evrt(i)%arr(:)) ) then ! if m is not last element,
shift
          bnd_evrt(i)%arr =
(/bnd_evrt(i)%arr(1:j),bnd_evrt(i)%arr(m+1:size(bnd_evrt(i)%arr(:))))/
        else ! if m is last element, delete
          bnd_evrt(i)%arr = bnd_evrt(i)%arr(1:j)
        end if
      else
        exit
      end if
    end do

    j = m ! increment j index
  end do

  ! assign boundary identifier to vertices
  do j = 1, size(bnd_evrt(i)%arr(:))
    vrt_bnd(bnd_evrt(i)%arr(j))%arr = (/vrt_bnd(bnd_evrt(i)%arr(j))%arr(:),i/)
  end do
end do

! allocate edge structures
allocate(edg_mdpt(1:3,1:Nedg))
allocate(edg_lng(1:Nedg))
allocate(edg_tngt(1:3,1:Nedg))

```

```

edg_mdpt(1:3,1:Nedg) = 0.0d0
edg_lng(1:Nedg) = 0.0d0
edg_tngt(1:3,1:Nedg) = 0.0d0
do i = 1, Nedg
    allocate(edg_adjvrt(i)%arr(0))
    allocate(edg_adjtri(i)%arr(0))
end do

! compute edges midpoint, tangent vector, length, and normal vector
do i = 1, Nedg
    ! compute midpoint
    edg_mdpt(1:3,i) = (vrt_crd(1:3,edg_vrt(1,i)) + vrt_crd(1:3,edg_vrt(2,i)))/2.0d0
    ! compute length
    dx = vrt_crd(1,edg_vrt(2,i)) - vrt_crd(1,edg_vrt(1,i))
    dy = vrt_crd(2,edg_vrt(2,i)) - vrt_crd(2,edg_vrt(1,i))
    dz = vrt_crd(3,edg_vrt(2,i)) - vrt_crd(3,edg_vrt(1,i))
    edg_lng(i) = dsqrt( dx*dx + dy*dy + dz*dz )
    ! compute tangent vector
    edg_tngt(1:3,i) = (/dx,dy,dz/)/edg_lng(i)
end do

! allocate triangle structures
allocate(tri_edg(1:3,1:Ntri))
tri_edg(1:3,1:Ntri) = 0
allocate(tri_cntr(1:3,1:Ntri))
allocate(tri_area(1:Ntri))
allocate(tri_nrml(1:3,1:Ntri))
tri_nrml(1:3,1:Ntri) = 0.0d0
tri_cntr(1:3,1:Ntri) = 0.0d0
tri_area(1:Ntri) = 0.0d0
allocate(tri_thk(1:Ntri))
tri_thk(1:Ntri) = 0.0d0
allocate(tri_mat(1:Ntri))
tri_mat(1:Ntri) = 0
allocate(tri_vox(1:Ntri))
do i = 1, Ntri
    allocate(tri_vox(i)%arr(0))
end do

! allocate continuum structures
deallocate(cnt_tri)
allocate(cnt_tri(1:Ncnt))
do i = 1, Ncnt
    allocate(cnt_tri(i)%arr(0))
end do
deallocate(cnt_ID)
allocate(cnt_area(1:Ncnt))
allocate(cnt_ID(1:Ncnt))
do i = 1, Ncnt
    cnt_ID(i) = i
    cnt_area(i) = 0.0d0
end do

! find triangle edges
! compute triangle areas and add to vertex areas

```

```

! create list of triangles for each continuum
do i = 1, Ntri
  ! find first edge
  do j = 1, size(VrtCon(minval(tri_vrt((/1,2/),i)))%arr)
    if ( VrtCon(minval(tri_vrt((/1,2/),i)))%arr(j) .eq.
maxval(tri_vrt((/1,2/),i)) ) then
      tri_edg(1,i) = VrtEdg(minval(tri_vrt((/1,2/),i)))%arr(j)
    end if
  end do
  ! find second edge
  do j = 1, size(VrtCon(minval(tri_vrt((/2,3/),i)))%arr)
    if ( VrtCon(minval(tri_vrt((/2,3/),i)))%arr(j) .eq.
maxval(tri_vrt((/2,3/),i)) ) then
      tri_edg(2,i) = VrtEdg(minval(tri_vrt((/2,3/),i)))%arr(j)
    end if
  end do
  ! find third edge
  do j = 1, size(VrtCon(minval(tri_vrt((/3,1/),i)))%arr)
    if ( VrtCon(minval(tri_vrt((/3,1/),i)))%arr(j) .eq.
maxval(tri_vrt((/3,1/),i)) ) then
      tri_edg(3,i) = VrtEdg(minval(tri_vrt((/3,1/),i)))%arr(j)
    end if
  end do

  ! add triangle to edge/vertex adjacent triangles list and compute triangle edge
normal
  do j = 1, 3
    ! add triangle to edge adjacent triangles list
    edg_adjtri(tri_edg(j,i))%arr = (/edg_adjtri(tri_edg(j,i))%arr,tri_ID(i)/)

    ! add vertex to edge adjacent vertex list
    edg_adjvrt(tri_edg(j,i))%arr =
(/edg_adjvrt(tri_edg(j,i))%arr,vrt_ID(tri_vrt(TriVrtPr(j+2),i))/)

    ! add triangle to vertex adjacent triangles list
    vrt_adjtri(tri_vrt(j,i))%arr = (/vrt_adjtri(tri_vrt(j,i))%arr,tri_ID(i)/)

    ! triangle edge normal
    ! use TriVrtPr, vec1, & vec2
    ! vp = j x j + 1
    vec1(:) = vrt_crd(1:3,tri_vrt(TriVrtPr(j+1),i)) -
vrt_crd(1:3,tri_vrt(TriVrtPr(j),i)) ! vj+1 - vj
    vec2(:) = vrt_crd(1:3,tri_vrt(TriVrtPr(j+2),i)) -
vrt_crd(1:3,tri_vrt(TriVrtPr(j),i)) ! vj+2 - vj
    call CrossProduct(vec1,vec2,vec3)

    ! nrml = (j x vp)
    call CrossProduct(vec1,vec3,vec2)
    ! nrml = |nrml|
    call VectorMagnitude(vec2,3,zzzTrshR1)
    vec2 = vec2 / zzzTrshR1
    edg_nrmlx(tri_edg(j,i))%arr = (/edg_nrmlx(tri_edg(j,i))%arr,vec2(1)/)
    edg_nrmlly(tri_edg(j,i))%arr = (/edg_nrmlly(tri_edg(j,i))%arr,vec2(2)/)
    edg_nrmlz(tri_edg(j,i))%arr = (/edg_nrmlz(tri_edg(j,i))%arr,vec2(3)/)
  end do
end do

```

```

! vec3 is triangle normal, so normalize and store
call VectorMagnitude(vec3,3,zzzTrshR1)
vec3 = vec3 / zzzTrshR1
tri_nrml(1:3,i) = vec3

! compute triangle area
vec1 = vrt_crd(1:3,tri_vrt(2,i))-vrt_crd(1:3,tri_vrt(1,i)) ! triangle side
vector between vertices 1 and 2
vec2 = vrt_crd(1:3,tri_vrt(3,i))-vrt_crd(1:3,tri_vrt(1,i)) ! triangle side
vector between vertices 1 and 3
call CrossProduct(vec1,vec2,xprdct)
call VectorMagnitude(xprdct,size(xprdct),tri_area(i))
tri_area(i) = tri_area(i) / 2.0d0 ! triangle area is half of cross product
magnitude
SumTriA = SumTriA + tri_area(i) ! add triangle area to triangle area sum

! accumulate vertex surface area
vrt_area(tri_vrt(1,i)) = vrt_area(tri_vrt(1,i)) + tri_area(i)/3.0d0
vrt_area(tri_vrt(2,i)) = vrt_area(tri_vrt(2,i)) + tri_area(i)/3.0d0
vrt_area(tri_vrt(3,i)) = vrt_area(tri_vrt(3,i)) + tri_area(i)/3.0d0

! compute triangle centroid
tri_cntr(1:3,i) = ( vrt_crd(1:3,tri_vrt(1,i)) + vrt_crd(1:3,tri_vrt(2,i)) +
vrt_crd(1:3,tri_vrt(3,i)) )/3.0d0

! build continuum triangle list
cnt_tri(tri_cntm(i))%arr = (/cnt_tri(tri_cntm(i))%arr,tri_ID(i)/)

! accumulate continuum area
cnt_area(tri_cntm(i)) = cnt_area(tri_cntm(i)) + tri_area(i)
end do

! compute total number of d-dimensional elements
Nel1 = Nbar
Nel2 = Ntri
Nel3 = Ntet

call DisplayActionStamp('IMPORT MESH END')

end subroutine ImportMesh

subroutine InitializeTemperatures
! This subroutine initializes the temperature of each vertex in
! the mesh.
!
! THIS SUBROUTINE MUST BE MODIFIED BY THE USER

use AllModules
use functions
implicit none

! declare variables
integer :: i ! index
integer, allocatable, dimension(:) :: IntAr ! placeholder integer array

```

```

real(8), allocatable, dimension(:) :: RlAr ! placeholder real array
character(len=200), allocatable, dimension(:) :: CharAr ! placeholder character
array
type(UserFuncNum) :: func_ret ! function return structure

! allocate arrays
allocate(IntAr(0))
allocate(RlAr(3))
allocate(CharAr(0))

! initialize temperatures
do i = 1, Nvrt ! loop through all vertices
    RlAr(1:3) = (/vrt_crd(1,i),vrt_crd(2,i),vrt_crd(3,i)/) ! store vertex
coordinates
    func_ret = UserFunction(8,0,3,0,IntAr,RlAr,CharAr) ! define user function
    vrt_tmp(i) = func_ret%real_ar(1) ! store initial temperature based on user
function
end do

deallocate(IntAr)
deallocate(RlAr)
deallocate(CharAr)

end subroutine InitializeTemperatures

subroutine LinePlaneIntersection(lv,lp0,pnv,pp0,FLAG_LPX,lpp)
use AllModules
implicit none
! computes the intersection point of a line with a plane

! variables
real(8), dimension(1:3), intent(in) :: lv ! line vector (x,y,z)
real(8), dimension(1:3), intent(in) :: lp0 ! point on line (x,y,z)
real(8), dimension(1:3), intent(in) :: pnv ! plane normal vector (x,y,z)
real(8), dimension(1:3), intent(in) :: pp0 ! point on plane (x,y,z)
integer, intent(out) :: FLAG_LPX ! line-plane intersection flag (0=no
intersection, 1=point intersection, 2=linear intersection)
real(8), dimension(1:3), intent(out) :: lpp ! point of intersection (x,y,z)
real(8) :: d ! scaled length along line of intersection
real(8) :: lp_dot ! dot product of line and plane normal
real(8) :: lpn_dot ! dot product of line-plane-points vector and plane normal

! initialize variables
d = 0.0d0
FLAG_LPX = 0
lpp(1:3) = (/0.0d0,0.0d0,0.0d0/)
lp_dot = 0.0d0
lpn_dot = 0.0d0

! dot product of line and plane normal
lp_dot = lv(1)*pnv(1) + lv(2)*pnv(2) + lv(3)*pnv(3)

! dot product of line-plane-points vector and plane normal
lpn_dot = (pp0(1)-lp0(1))*pnv(1) + (pp0(2)-lp0(2))*pnv(2) + (pp0(3)-
lp0(3))*pnv(3)

```

```

! check if dot product is 0
if ( lp_dot .eq. 0.0d0 ) then ! if zero, check if line is in the plane

    ! check dot product
    if ( lpn_dot .eq. 0.0d0 ) then ! if zero, line is in plane
        FLAG_LPX = 2 ! continuous intersection
    end if
else ! if not zero, compute intersection point
    d = lpn_dot/lp_dot

    lpp = d*lv + lp0 ! instersection point

    FLAG_LPX = 1 ! point intersection
end if

end subroutine LinePlaneIntersection

subroutine PrintManufacturedSolutionInformation
use AllModules

call PrintManufacturedTemperatureInfo

end subroutine PrintManufacturedSolutionInformation

subroutine PrintModelInformation
use AllModules
implicit none

call PrintInputFileInfo
call PrintVertexInfo
call PrintVertexAssociationInfo
call PrintEdgeInfo
call PrintEdgeAssociationInfo
call PrintTriangleInfo
call PrintTriangleAssociationInfo
call PrintSolveVertexInfo
call PrintNoSolveVertexInfo
call PrintContinuumInfo
call PrintMaterialInfo
call PrintConductanceMatrixOffAxisInfo
call PrintConductanceMatrixDiagonalInfo
call PrintSourceInfo
call PrintLoadInfo
call PrintVoxelInfo
call PrintVoxelAssociationInfo

end subroutine PrintModelInformation

subroutine PrintSolutionInformation
use AllModules

call PrintTemperatureInfo
call PrintResidualInfo

```

```

end subroutine PrintSolutionInformation

subroutine PrintManufacturedInformation
use AllModules

call PrintManufacturedTemperatureInfo
call PrintManufacturedErrorInfo

end subroutine PrintManufacturedInformation

subroutine VectorMagnitude(vec,VecLen,mag)
use AllModules
implicit none

! variables
integer :: i ! index
real(8), intent(out) :: mag ! vector magnitude
integer, intent(in) :: VecLen ! vector length
real(8), dimension(1:VecLen), intent(in) :: vec ! vector

! initialize variables
mag = 0.0d0

! sum the square of the vector components
do i = 1, VecLen
    mag = mag + vec(i)*vec(i)
end do

! get the magnitude of the vector
mag = dsqrt( mag )

end subroutine VectorMagnitude

subroutine
RayVoxelIntersectionSearch(ray_start,vox_curr,vec_srch,edg_dir_code,vox_next)
use AllModules
implicit none
! Finds next voxel along vector direction

! variables
real(8), dimension(1:3), intent(in) :: ray_start ! ray origin coordinates (x,y,z)
integer, intent(in) :: vox_curr ! initial voxel ID
real(8), dimension(1:3), intent(in) :: vec_srch ! search direction unit vector
(x,y,z)
integer, dimension(3), intent(in) :: edg_dir_code ! search vector directionality
code
integer, intent(out) :: vox_next ! next voxel ID
integer :: FLAG_LPX_yz ! line-yz plane intersection flag (0=no intersection,
1=point intersection, 2=linear intersection)
integer :: FLAG_LPX_zx ! line-zx plane intersection flag (0=no intersection,
1=point intersection, 2=linear intersection)
integer :: FLAG_LPX_xy ! line-xy plane intersection flag (0=no intersection,
1=point intersection, 2=linear intersection)
integer :: int_plane ! plane of edge intersection (0=none,1=yz,2=zx,3=xy)
real(8) :: int_dist ! distance at point of edge-plane intersection

```



```

real(8), dimension(1:3) :: int_pnt ! point of intersection (x,y,z)
real(8), dimension(1:3) :: pnv ! plane normal vector (x,y,z)
real(8), dimension(1:3) :: pp0 ! point on plane (x,y,z)
real(8), dimension(1:3) :: int_yz ! point of intersection on yz plane (x,y,z)
real(8), dimension(1:3) :: int_zx ! point of intersection on zx plane (x,y,z)
real(8), dimension(1:3) :: int_xy ! point of intersection on xy plane (x,y,z)
real(8), dimension(1:3) :: vec_yz ! vector from edge vertex to point of
intersection on yz plane (x,y,z)
real(8), dimension(1:3) :: vec_zx ! vector from edge vertex to point of
intersection on zx plane (x,y,z)
real(8), dimension(1:3) :: vec_xy ! vector from edge vertex to point of
intersection on xy plane (x,y,z)
real(8) :: int_dist_yz ! distance at point of edge-yz plane intersection
real(8) :: int_dist_zx ! distance at point of edge-zx plane intersection
real(8) :: int_dist_xy ! distance at point of edge-xy plane intersection

! initialize variables [1]
FLAG_LPX_yz = 0
FLAG_LPX_zx = 0
FLAG_LPX_xy = 0
vox_next = 0
int_plane = 0
int_dist = 0.0d0
int_pnt = (/0.0d0,0.0d0,0.0d0/)
pnv = (/0.0d0,0.0d0,0.0d0/)
pp0 = (/0.0d0,0.0d0,0.0d0/)
int_yz = (/0.0d0,0.0d0,0.0d0/)
int_zx = (/0.0d0,0.0d0,0.0d0/)
int_xy = (/0.0d0,0.0d0,0.0d0/)
vec_yz = (/0.0d0,0.0d0,0.0d0/)
vec_zx = (/0.0d0,0.0d0,0.0d0/)
vec_xy = (/0.0d0,0.0d0,0.0d0/)
int_dist_yz = 0.0d0
int_dist_zx = 0.0d0
int_dist_xy = 0.0d0

! check voxel boundary plane in x-direction
if ( edg_dir_code(1) .ne. 0 ) then ! if x code is non-zero, check yz-plane
intersection (int_yz)
    pnv = (/1.0d0,0.0d0,0.0d0/)

    if ( edg_dir_code(1) .eq. -1 ) then
        pp0 = (/vox_lim_lo(1,vox_curr),ray_start(2),ray_start(3)/)
    else
        pp0 = (/vox_lim_hi(1,vox_curr),ray_start(2),ray_start(3)/)
    end if

    call LinePlaneIntersection(vec_srch,ray_start,pnv,pp0,FLAG_LPX_yz,int_yz)

! determine if yz intersection is closest
if ( FLAG_LPX_yz .eq. 1 ) then
    int_plane = 1
    vec_yz = int_yz-ray_start
    call VectorMagnitude(vec_yz,3,int_dist)
    int_pnt = int_yz

```

```

    end if
end if

! check voxel boundary plane in y-direction
if ( edg_dir_code(2) .ne. 0 ) then ! if y code is non-zero, check zx-plane
intersection (int_zx)
    pnv = (/0.0d0,1.0d0,0.0d0/)

    if ( edg_dir_code(2) .eq. -1 ) then
        pp0 = (/ray_start(1),vox_lim_lo(2,vox_curr),ray_start(3)/)
    else
        pp0 = (/ray_start(1),vox_lim_hi(2,vox_curr),ray_start(3)/)
    end if

    call LinePlaneIntersection(vec_srch,ray_start,pnv,pp0,FLAG_LPX_zx,int_zx)

    ! determine if zx intersection is closest
    if ( FLAG_LPX_zx .eq. 1 ) then
        vec_zx = int_zx-ray_start
        call VectorMagnitude(vec_zx,3,int_dist_zx)

        if ( int_plane .eq. 0 ) then
            int_plane = 2
            int_pnt = int_zx
        else if ( int_dist_zx < int_dist ) then
            int_plane = 2
            int_pnt = int_zx
        end if
    end if
end if

! check voxel boundary plane in z-direction
if ( edg_dir_code(3) .ne. 0 ) then ! if z code is non-zero, check xy-plane
intersection (int_xy)
    pnv = (/0.0d0,0.0d0,1.0d0/)

    if ( edg_dir_code(3) .eq. -1 ) then
        pp0 = (/ray_start(1),ray_start(2),vox_lim_lo(3,vox_curr)/)
    else
        pp0 = (/ray_start(1),ray_start(2),vox_lim_hi(3,vox_curr)/)
    end if

    call LinePlaneIntersection(vec_srch,ray_start,pnv,pp0,FLAG_LPX_xy,int_xy)

    ! determine if xy intersection is closest
    if ( FLAG_LPX_xy .eq. 1 ) then
        vec_xy = int_xy-ray_start
        call VectorMagnitude(vec_xy,3,int_dist_xy)

        if ( int_plane .eq. 0 ) then
            int_plane = 3
            int_pnt = int_xy
        else if ( int_dist_xy < int_dist ) then
            int_plane = 3
            int_pnt = int_xy
        end if
    end if
end if

```

```

        end if
    end if
end if

! step into next voxel
select case ( int_plane )
    case ( 0 ) ! no intersection
        write(*,*) "ERROR: Vector does not cross voxel."
        stop
    case ( 1 ) ! yz plane
        vox_next = vox_curr + edg_dir_code(1) ! step to next voxel x-direction
    case ( 2 ) ! zx plane
        vox_next = vox_curr + edg_dir_code(2)*Nvxx ! step to next voxel y-direction
    case ( 3 ) ! xy plane
        vox_next = vox_curr + edg_dir_code(3)*Nvxx*Nvxy ! step to next voxel z-
direction
end select

end subroutine RayVoxelIntersectionSearch

subroutine UpdateSolution
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: vi ! vertex index
real(8) :: GeTsum
real(8), dimension (1:Nvrt) :: Gp_vec ! conjugate gradient conductance change
vector
real(8), dimension(1:Nvrt) :: tmp_new
real(8) :: pGp ! conjugate gradient conductance change vector square
real(8) :: res_ss ! sum of squares of current iteration residuals
real(8) :: res_ss0 ! sum of squares of previous iteration residuals
real(8) :: cgsscale ! conjugate gradient change scaler

GeTsum = 0.0d0
tmp_new = vrt_tmp(:)
res_ss = 0.0d0
res_ss0 = 0.0d0
Gp_vec = 0.0d0
pGp = 0.0d0

select case (FLAG_UPDATE_METHOD)
    case ( 1 ) ! SOR
        ! compute sum of T*Ge for each vertex
        ! Need list of non-Dirichlet and zero conductance vertices
        do i = 1, size(VrtSolv(:))
            vi = VrtSolv(i)
            GeTsum = 0.0d0
            do j = 1, size(vrt_adjedg(vi)%arr(:))
                GeTsum = GeTsum +
(Ge_x(vrt_adjedg(vi)%arr(j))+Ge_y(vrt_adjedg(vi)%arr(j))+Ge_z(vrt_adjedg(vi)%arr(
j)))*vrt_tmp(vrt_adjvrt(vi)%arr(j))
            end do

```

```

        tmp_new(vi) = ((vrt_src(1,vi) + vrt_src(3,vi)) - GeTsum) /
(Gv_x(vi)+Gv_y(vi)+Gv_z(vi))
    end do

    vrt_tmp(:) = relax*(tmp_new(:) - vrt_tmp(:)) + vrt_tmp(:)

    call ComputeResiduals
case ( 2 ) ! Conjugate Gradient
! Gp_vec
    Gp_vec(1:Nvrt) = 0.0d0
    do i = 1, Nvrt
        do j = 1, size(vrt_adjedg(i)%arr(:))
            Gp_vec(i) = Gp_vec(i) +
(Ge_x(vrt_adjedg(i)%arr(j))+Ge_y(vrt_adjedg(i)%arr(j))+Ge_z(vrt_adjedg(i)%arr(j))
)*pmult(vrt_adjvrt(i)%arr(j))
        end do
        Gp_vec(i) = Gp_vec(i) + (Gv_x(i)+Gv_y(i)+Gv_z(i))*pmult(i)
    end do

    ! set no solve vertex Gp elements to 0
    Gp_vec(VrtNoSolv(:)) = 0.0d0

    ! pGp
    pGp = 0.0d0
    do i = 1, Nvrt
        pGp = pGp + pmult(i)*Gp_vec(i)
    end do

    ! sum of the residual squares
    res_ss = res_rss*res_rss

    cgsscale = res_ss/pGp ! determine change scale

    ! update temperatures
    vrt_tmp(:) = vrt_tmp(:) + cgsscale*pmult(:)

    ! compute residuals
    call ComputeResiduals

    res_ss0 = res_ss ! store old critical residual
    res_ss = res_rss*res_rss

    ! update change vector
    pmult(:) = vrt_res(:) + pmult(:)*res_ss/res_ss0
    pmult(VrtNoSolv(:)) = 0.0d0
end select

end subroutine UpdateSolution

! ----- DISPLAY ROUTINES
subroutine DisplayVertex(vi)
use AllModules
implicit none

```

```

integer, intent(in) :: vi ! vertex index
character(len=200) :: out_str_1 ! output string
character(len=200) :: out_str_2 ! output string
integer :: i ! index

write(*,'(A)') 'CARD: VERTEX-----'
write(out_str_1,'(i0)') vrt_ID(vi)
write(*,'(A,1x,A)') '          ID              (-)'
: ',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') vrt_crd(1,vi)
write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),", "
write(out_str_2,'(f0.6)') vrt_crd(2,vi)
write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
write(out_str_2,'(f0.6)') vrt_crd(3,vi)
write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
write(*,'(A,1x,A)') '          Coordinates      (m)'
: ',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') vrt_adjvrt(vi)%arr(1)
out_str_2 = ''
do i = 2, size(vrt_adjvrt(vi)%arr(:))
    write(out_str_1,'(A,A,A)')
    trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
    write(out_str_2,'(i0)') vrt_adjvrt(vi)%arr(i)
    write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
    out_str_2 = ''
end do
write(*,'(A,1x,A)') '          Adjacent Vertices  (-)'
: ',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') vrt_adjedg(vi)%arr(1)
out_str_2 = ''
do i = 2, size(vrt_adjedg(vi)%arr(:))
    write(out_str_1,'(A,A,A)')
    trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
    write(out_str_2,'(i0)') vrt_adjedg(vi)%arr(i)
    write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
    out_str_2 = ''
end do
write(*,'(A,1x,A)') '          Adjacent Edges      (-)'
: ',trim(adjustl(out_str_1))

if ( size(vrt_adjtri(vi)%arr(:)) > 0 ) then
    write(out_str_1,'(i0)') vrt_adjtri(vi)%arr(1)
    out_str_2 = ''
    do i = 2, size(vrt_adjtri(vi)%arr(:))
        write(out_str_1,'(A,A,A)')
        trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
        write(out_str_2,'(i0)') vrt_adjtri(vi)%arr(i)
        write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
        out_str_2 = ''
    end do
else

```

```

    out_str_1 = ''
end if
write(*,'(A,1x,A)') '      Adjacent Triangles (-)
:',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') vrt_bnd(vi)%arr(1)
out_str_2 = ''
do i = 2, size(vrt_bnd(vi)%arr(:))
    write(out_str_1,'(A,A,A)')
    trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
    write(out_str_2,'(i0)') vrt_bnd(vi)%arr(i)
    write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
    out_str_2 = ''
end do
write(*,'(A,1x,A)') '      Boundary IDs          (-)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') vrt_area(vi)
write(*,'(A,1x,A)') '      Surface Area          (m2)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') vrt_vol(vi)
write(*,'(A,1x,A)') '      Volume              (m3)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') vrt_tmp(vi)
write(*,'(A,1x,A)') '      Temperature          (K)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') vrt_src(1,vi)
out_str_2 = ''
do i = 2, size(vrt_src(:,vi))
    write(out_str_1,'(A,A,A)')
    trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
    write(out_str_2,'(f0.6)') vrt_src(i,vi)
    write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
    out_str_2 = ''
end do
write(*,'(A,1x,A)') '      Heat Source          (W)
:',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') vrt_vox(vi)
write(*,'(A,1x,A)') '      Voxel              (-)
:',trim(adjustl(out_str_1))

end subroutine DisplayVertex

subroutine DisplayEdge(ei)
use AllModules
implicit none

integer, intent(in) :: ei ! edge index
character(len=200) :: out_str_1 ! output string
character(len=200) :: out_str_2 ! output string

```

```

integer :: i ! index

write(*,'(A)') 'CARD: EDGE-----'
write(out_str_1,'(i0)') edg_ID(ei)
write(*,'(A,1x,A)') '          ID              (-):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') edg_vrt(1,ei)
write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),", "
write(out_str_2,'(i0)') edg_vrt(2,ei)
write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
write(*,'(A,1x,A)') '          Vertices          (-):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') edg_lng(ei)
write(*,'(A,1x,A)') '          Length              (m):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') edg_mdpt(1,ei)
out_str_2 = ''
do i = 2, 3
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
  write(out_str_2,'(f0.6)') edg_mdpt(i,ei)
  write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  out_str_2 = ''
end do
write(*,'(A,1x,A)') '          Midpoint              (m):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') edg_tngt(1,ei)
out_str_2 = ''
do i = 2, 3
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
  write(out_str_2,'(f0.6)') edg_tngt(i,ei)
  write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  out_str_2 = ''
end do
write(*,'(A,1x,A)') '          Tangent Vector          (-):',trim(adjustl(out_str_1))

if ( size(edg_adjtri(ei)%arr(:)) > 0 ) then
  write(out_str_1,'(i0)') edg_adjtri(ei)%arr(1)
  out_str_2 = ''
  do i = 2, size(edg_adjtri(ei)%arr(:))
    write(out_str_1,'(A,A,A)')
    trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
    write(out_str_2,'(i0)') edg_adjtri(ei)%arr(i)
    write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
    out_str_2 = ''
  end do
else
  out_str_1 = ''
end if
write(*,'(A,1x,A)') '          Adjacent Triangles (-):',trim(adjustl(out_str_1))

if ( size(edg_nrmlx(ei)%arr(:)) > 0 ) then
  write(out_str_1,'(f0.6)') edg_nrmlx(ei)%arr(1)

```

```

        out_str_2 = ''
        do i = 2, size(edg_nrmlx(ei)%arr(:))
            write(out_str_1,'(A,A,A)')
            trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
            write(out_str_2,'(f0.6)') edg_nrmlx(ei)%arr(i)
            write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
            out_str_2 = ''
        end do
    else
        out_str_1 = ''
    end if
    write(*,'(A,1x,A)') '          Normal Vector x      (-):',trim(adjustl(out_str_1))

    if ( size(edg_nrmlly(ei)%arr(:)) > 0 ) then
        write(out_str_1,'(f0.6)') edg_nrmlly(ei)%arr(1)
        out_str_2 = ''
        do i = 2, size(edg_nrmlly(ei)%arr(:))
            write(out_str_1,'(A,A,A)')
            trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
            write(out_str_2,'(f0.6)') edg_nrmlly(ei)%arr(i)
            write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
            out_str_2 = ''
        end do
    else
        out_str_1 = ''
    end if
    write(*,'(A,1x,A)') '          Normal Vector y      (-):',trim(adjustl(out_str_1))

    if ( size(edg_nrmlz(ei)%arr(:)) > 0 ) then
        write(out_str_1,'(f0.6)') edg_nrmlz(ei)%arr(1)
        out_str_2 = ''
        do i = 2, size(edg_nrmlz(ei)%arr(:))
            write(out_str_1,'(A,A,A)')
            trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
            write(out_str_2,'(f0.6)') edg_nrmlz(ei)%arr(i)
            write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
            out_str_2 = ''
        end do
    else
        out_str_1 = ''
    end if
    write(*,'(A,1x,A)') '          Normal Vector z      (-):',trim(adjustl(out_str_1))

    if ( size(edg_adjvrt(ei)%arr(:)) > 0 ) then
        write(out_str_1,'(i0)') edg_adjvrt(ei)%arr(1)
        out_str_2 = ''
        do i = 2, size(edg_adjvrt(ei)%arr(:))
            write(out_str_1,'(A,A,A)')
            trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
            write(out_str_2,'(i0)') edg_adjvrt(ei)%arr(i)
            write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
            out_str_2 = ''
        end do
    else
        out_str_1 = ''
    end if

```



```

end if
write(*,'(A,1x,A)') '      Adjacent Vertices      (-):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') edg_bnd(ei)
write(*,'(A,1x,A)') '      Boundary ID              (-):',trim(adjustl(out_str_1))

end subroutine DisplayEdge

subroutine DisplayTriangle(ti)
use AllModules
implicit none

integer, intent(in) :: ti ! triangle index
character(len=200) :: out_str_1 ! output string
character(len=200) :: out_str_2 ! output string
integer :: i ! index

write(*,'(A)') 'CARD: TRIANGLE-----'
write(out_str_1,'(i0)') tri_ID(ti)
write(*,'(A,1x,A)') '      ID              (-) :',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') tri_vrt(1,ti)
out_str_2 = ''
do i = 2, 3
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),""
  write(out_str_2,'(i0)') tri_vrt(i,ti)
  write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  out_str_2 = ''
end do
write(*,'(A,1x,A)') '      Vertices      (-) :',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') tri_edg(1,ti)
out_str_2 = ''
do i = 2, 3
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),""
  write(out_str_2,'(i0)') tri_edg(i,ti)
  write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  out_str_2 = ''
end do
write(*,'(A,1x,A)') '      Edges      (-) :',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') tri_area(ti)
write(*,'(A,1x,A)') '      Area      (m2):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') tri_cntr(1,ti)
out_str_2 = ''
do i = 2, 3
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),""
  write(out_str_2,'(f0.6)') tri_cntr(i,ti)
  write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  out_str_2 = ''
end do

```

```

write(*,'(A,1x,A)') '          Centroid      (m) :',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') tri_thk(ti)
write(*,'(A,1x,A)') '          Thickness      (m) :',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') tri_nrml(1,ti)
out_str_2 = ''
do i = 2, 3
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
  write(out_str_2,'(f0.6)') tri_nrml(i,ti)
  write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  out_str_2 = ''
end do
write(*,'(A,1x,A)') '          Normal Vector (-) :',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') tri_cntm(ti)
write(*,'(A,1x,A)') '          Continuum      (-) :',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') tri_mat(ti)
write(*,'(A,1x,A)') '          Material      (-) :',trim(adjustl(out_str_1))

end subroutine DisplayTriangle

subroutine DisplayBoundary(bi)
use AllModules
implicit none

integer, intent(in) :: bi ! boundary index
character(len=200) :: out_str_1 ! output string
character(len=200) :: out_str_2 ! output string
integer :: i ! index

write(*,'(A)') 'CARD: BOUNDARY-----'
write(out_str_1,'(i0)') bnd_ID(bi)
write(*,'(A,1x,A)') '          ID              (-):',trim(adjustl(out_str_1))

write(out_str_1,'(A)') trim(adjustl(bnd_name(bi)))
write(*,'(A,1x,A)') '          Name              (-):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') size(bnd_evrt(bi)%arr(:))
write(*,'(A,1x,A)') '          Edge Vertices      (-):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') size(bnd_edg(bi)%arr(:))
write(*,'(A,1x,A)') '          Edges              (-):',trim(adjustl(out_str_1))

select case ( bnd_type(bi) )
  case ( -1 ) ! intimate thermal contact
    write(out_str_1,'(A)') 'Intimate Thermal Contact'
  case ( 0 ) ! ERROR: NO ASSIGNMENT!
    write(out_str_1,'(A)') 'ERROR: NO ASSIGNMENT!'
  case ( 1 ) ! Dirichlet (temperature)
    write(out_str_1,'(A)') 'Temperature'
  case ( 2 ) ! Neumann (temperature gradient)
    write(out_str_1,'(A)') 'Temperature Gradient'

```

```

    case ( 3 ) ! Heat Flux
        write(out_str_1,'(A)') 'Heat Flux'
    end select
write(*,'(A,1x,A)') '      Type                      (-):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') bnd_func(bi)
write(*,'(A,1x,A)') '      Function                  (-):',trim(adjustl(out_str_1))

end subroutine DisplayBoundary

subroutine DisplayLoad(li)
use AllModules
implicit none

integer, intent(in) :: li ! load index
character(len=200) :: out_str_1 ! output string
character(len=200) :: out_str_2 ! output string
integer :: i ! index

write(*,'(A)') 'CARD: LOAD-----'
write(out_str_1,'(i0)') lod_ID(li)
write(*,'(A,1x,A)') '      ID                      (-
):',trim(adjustl(out_str_1))

write(out_str_1,'(A)') trim(adjustl(lod_name(li)))
write(*,'(A,1x,A)') '      Name                      (-
):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') lod_type(li)
write(*,'(A,1x,A)') '      Type                      (-
):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') size(lod_cnt(li)%arr(:))
write(out_str_1,'(i0)') lod_cnt(li)%arr(1)
out_str_2 = ''
do i = 2, size(lod_cnt(li)%arr(:))
    write(out_str_1,'(A,A,A)')
    trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),", "
    write(out_str_2,'(i0)') lod_cnt(li)%arr(i)
    write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
    out_str_2 = ''
end do
write(*,'(A,1x,A)') '      Load Continuums                      (-
):',trim(adjustl(out_str_1))

write(out_str_1,'(i0)') size(lod_cntvrt(li)%arr(:))
write(*,'(A,1x,A)') '      Load Continuum Vertices                      (-
):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.10)') sum(lod_cvrtarea(li)%arr(:))
write(*,'(A,1x,A)') '      Total Load Area
(m2):',trim(adjustl(out_str_1))

end subroutine DisplayLoad

```

```

subroutine DisplayMaterial(mi)
use AllModules
implicit none

integer, intent(in) :: mi ! material index
character(len=200) :: out_str_1 ! output string
character(len=200) :: out_str_2 ! output string
integer :: i ! index

write(*,'(A)') 'CARD: MATERIAL-----'
write(out_str_1,'(i0)') mat_ID(mi)
write(*,'(A,1x,A)') '          ID          (-)
:',trim(adjustl(out_str_1))

write(out_str_1,'(A)') mat_name(mi)
write(*,'(A,1x,A)') '          Name          (-)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') mat_k(mi)
write(*,'(A,1x,A)') '          Thermal Conductivity (W/m-K)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') mat_rho(mi)
write(*,'(A,1x,A)') '          Density          (kg/m3)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') mat_cp(mi)
write(*,'(A,1x,A)') '          Specific Heat          (J/kg-
K):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') mat_mu(mi)
write(*,'(A,1x,A)') '          Dynamic Viscosity          (Pa-s)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') mat_e(mi)
write(*,'(A,1x,A)') '          Emissivity          (-)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') mat_r(mi)
write(*,'(A,1x,A)') '          Reflectivity          (-)
:',trim(adjustl(out_str_1))

write(out_str_1,'(f9.6)') mat_t(mi)
write(*,'(A,1x,A)') '          Transmissivity          (-)
:',trim(adjustl(out_str_1))

end subroutine DisplayMaterial

subroutine DisplayConductanceMatrix(vi)
use AllModules
implicit none

! declare variables
integer, intent(in) :: vi ! vertex index
character(len=200) :: out_str_1 ! output string

```

```

character(len=200) :: out_str_2 ! output string
character(len=200) :: out_str_3 ! output string
integer :: i ! index
real(8) :: Gsum_x ! conductance summation (x-direction)
real(8) :: Gsum_y ! conductance summation (x-direction)

! initialize variables
Gsum_x = 0.0d0
Gsum_y = 0.0d0

write(*,'(A)') 'CARD: CONDUCTANCE MATRIX-----'
write(out_str_1,'(i0)') vrt_ID(vi)
write(*,'(A,1x,A)') '          Vertex ID                      (-)
:',trim(adjustl(out_str_1))

do i = 1, size(vrt_adjvrt(vi)%arr(:))
  write(out_str_1,'(i0)') vrt_adjedg(vi)%arr(i)
  out_str_2 = ''
  write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)), "/"
  write(out_str_2,'(i0)') vrt_adjvrt(vi)%arr(i)
  write(out_str_1,'(A,A,A)')
  trim(adjustl(out_str_1)),trim(adjustl(out_str_2)), '/'
  if ( vrt_adjedg(vi)%arr(i) .ne. 0 ) then
    write(out_str_2,'(f0.6)') Ge_x(vrt_adjedg(vi)%arr(i))
    write(out_str_2,'(A,A)') trim(adjustl(out_str_2)), '/'
    write(out_str_3,'(f0.6)') Ge_y(vrt_adjedg(vi)%arr(i))
    write(out_str_2,'(A,A)') trim(adjustl(out_str_2)),trim(adjustl(out_str_3))
    Gsum_x = Gsum_x + Ge_x(vrt_adjedg(vi)%arr(i))
    Gsum_y = Gsum_y + Ge_y(vrt_adjedg(vi)%arr(i))
  else
    write(out_str_2,'(f0.6)') 0.0
  end if
  write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
  write(*,'(A,1x,A)') '          Edge/Connected Vertex ID/Edge Conductance x/y
(W/K):',trim(adjustl(out_str_1))
end do

write(out_str_1,'(f0.6)') Gsum_x
out_str_2 = ''
write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2)), "/"
write(out_str_2,'(f0.6)') Gsum_y
write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
write(*,'(A,1x,A)') '          Sum of Edge Conductances x/y
(W/K):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') Gv_x(vi)
out_str_2 = ''
write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2)), "/"
write(out_str_2,'(f0.6)') Gv_y(vi)
write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
write(*,'(A,1x,A)') '          Vertex Conductance x/y
(W/K):',trim(adjustl(out_str_1))

write(out_str_1,'(f0.6)') dabs(Gv_x(vi)+Gsum_x)
out_str_2 = ''

```

```

write(out_str_1,'(A,A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2)),"/"
write(out_str_2,'(f0.6)') dabs(Gv_y(vi)+Gsum_y)
write(out_str_1,'(A,A)') trim(adjustl(out_str_1)),trim(adjustl(out_str_2))
write(*,'(A,1x,A)') '          Discrepancy x/y
(W/K):',trim(adjustl(out_str_1))

end subroutine DisplayConductanceMatrix

subroutine DisplayVertexSolveList
use AllModules
implicit none

integer :: i ! index
character(len=200) :: out_str_1 ! output string

write(*,'(A)') 'CARD: VERTEX SOLVE LIST-----'
do i = 1, size(VrtSolv)
    write(out_str_1,'(i0)') vrt_ID(VrtSolv(i))
    write(*,'(A,1x,A)') '          Vertex ID (-):',trim(adjustl(out_str_1))
end do

end subroutine DisplayVertexSolveList

subroutine DisplayVertexNoSolveList
use AllModules
implicit none

integer :: i ! index
character(len=200) :: out_str_1 ! output string

write(*,'(A)') 'CARD: VERTEX NO SOLVE LIST-----'
do i = 1, size(VrtNoSolv)
    write(out_str_1,'(i0)') vrt_ID(VrtNoSolv(i))
    write(*,'(A,1x,A)') '          Vertex ID (-):',trim(adjustl(out_str_1))
end do

end subroutine DisplayVertexNoSolveList

subroutine DisplayActionStamp(line_action)
use AllModules
implicit none

! variables
character(*), intent(in) :: line_action
character(len=8) :: date_str
character(len=10) :: time_str
character(len=5) :: zone_str
integer, dimension(1:8) :: values_arr

! get date and time data
call date_and_time(date_str,time_str,zone_str,values_arr)

! display line action with timestamp
write(*,'(A)') '['//date_str//
'//time_str(1:2)//':'//time_str(3:4)//':'//time_str(5:10)//'] | '//line_action

```

```

end subroutine DisplayActionStamp

subroutine DisplayLine01(line_string)
use AllModules
implicit none

! variables
character(*), intent(in) :: line_string

! display line string
write(*,'(A)') ' | '//line_string

end subroutine DisplayLine01

subroutine DisplayLine02(line_string)
use AllModules
implicit none

! variables
character(*), intent(in) :: line_string

! display line string
write(*,'(A)') ' | '//line_string

end subroutine DisplayLine02

subroutine DisplayDivision
use AllModules
implicit none

! variables

! display division
write(*,'(A)') '-----|-----'
'-----'

end subroutine DisplayDivision

! ----- OUTPUT FILE ROUTINES
subroutine PrintVertexInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

```

```

FILE_NAME = 'Model_VertexInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//'trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

!~do while ( fe )
!~ i = i + 1
!~ write(*,'(A,A,A,A,i0,A)') trim(FILE_OUT),' already exists. Checking for
',trim(FILE_NAME),'_',i,trim(FILE_EXT)
!~ write(FILE_OUT,'(A,A,i0,A)') trim(FILE_NAME),'_',i,trim(FILE_EXT)
!~ INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)
!~end do

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! write(*,'(A,i0)') trim(FILE_OUT)//' write flag: ',ios

!ID, x, y, z, area, vol, adjvrt, adjedg, adjtri, bnd
! header
write(1,'(A)') 'ID,x,y,z,area,volume,voxel'

! data
do i = 1, Nvrt
  write(1,'(i0,A)',advance='no') vrt_ID(i),', '
  write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_crd(1,i),', '
  write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_crd(2,i),', '
  write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_crd(3,i),', '
  write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_area(i),', '
  write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_vol(i),', '
  write(1,'(i0,A)',advance='no') vrt_vox(i),', '
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintVertexInfo

subroutine PrintVertexAssociationInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_VertexAssociationInfo'

```



```

FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! write(*,'(A,i0)') trim(FILE_OUT)//' write flag: ',ios

do i = 1, Nvrt
  write(1,'(i0,A)',advance='no') vrt_ID(i),'_adjacent_vertices,'
  do j = 1, size(vrt_adjvrt(i)%arr)
    write(1,'(i0,A)',advance='no') vrt_adjvrt(i)%arr(j),','
  end do
  write(1,'(A)') ''

  write(1,'(i0,A)',advance='no') vrt_ID(i),'_adjacent_edges,'
  do j = 1, size(vrt_adjedg(i)%arr)
    write(1,'(i0,A)',advance='no') vrt_adjedg(i)%arr(j),','
  end do
  write(1,'(A)') ''

  write(1,'(i0,A)',advance='no') vrt_ID(i),'_adjacent_triangles,'
  do j = 1, size(vrt_adjtri(i)%arr)
    write(1,'(i0,A)',advance='no') vrt_adjtri(i)%arr(j),','
  end do
  write(1,'(A)') ''

  write(1,'(i0,A)',advance='no') vrt_ID(i),'_associated_boundaries,'
  do j = 1, size(vrt_bnd(i)%arr)
    write(1,'(i0,A)',advance='no') vrt_bnd(i)%arr(j),','
  end do
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintVertexAssociationInfo

subroutine PrintEdgeInfo
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_EdgeInfo'

```

```

FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! write(*,'(A,i0)') trim(FILE_OUT)//' write flag: ',ios

! ID, vrt1, vrt2, mdpt_x, mdpt_y, mdpt_z, length, tngt_x, tngt_y, tngt_z,
boundary
! header
write(1,'(A)')
'ID,vrt1,vrt2,mdpt_x,mdpt_y,mdpt_z,length,tngt_x,tngt_y,tngt_z,boundary'

! data
do i = 1, Nedg
  write(1,'(i0,A)',advance='no') edg_ID(i),',',
  write(1,'(i0,A)',advance='no') edg_vrt(1,i),',',
  write(1,'(i0,A)',advance='no') edg_vrt(2,i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_mdpt(1,i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_mdpt(2,i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_mdpt(3,i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_lng(i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_tngt(1,i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_tngt(2,i),',',
  write(1,'( '//trim(fmtf)//',A)',advance='no') edg_tngt(3,i),',',
  write(1,'(i0,A)',advance='no') edg_bnd(i),',',
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintEdgeInfo

subroutine PrintInputFileInfo
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_MeshInputFileInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

```

```

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! write(*,'(A,i0)') trim(FILE_OUT)//' write flag: ',ios

! ID, vrt1, vrt2, mdpt_x, mdpt_y, mdpt_z, length, tngt_x, tngt_y, tngt_z,
boundary
! header
write(1,'(A)') 'Mesh_File_Path'

! data
write(1,'(A)',advance='no') MeshInputFileName

close(1)

end subroutine PrintInputFileInfo

subroutine PrintEdgeAssociationInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_EdgeAssociationInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

do i = 1, Nedg
  write(1,'(i0,A)',advance='no') edg_ID(i),'_adjacent_vertices,'
  do j = 1, size(edg_adjvrt(i)%arr)
    write(1,'(i0,A)',advance='no') edg_adjvrt(i)%arr(j),','
  end do
  write(1,'(A)') ''

  write(1,'(i0,A)',advance='no') edg_ID(i),'_adjacent_triangles,'
  do j = 1, size(edg_adjtri(i)%arr)
    write(1,'(i0,A)',advance='no') edg_adjtri(i)%arr(j),','
  end do
  write(1,'(A)') ''

```

```

write(1,'(i0,A)',advance='no') edg_ID(i),'_normal_x,'
do j = 1, size(edg_nrmlx(i)%arr)
  write(1,'(//trim(fmtf)//',A)',advance='no') edg_nrmlx(i)%arr(j),','
end do
write(1,'(A)') ''

write(1,'(i0,A)',advance='no') edg_ID(i),'_normal_y,'
do j = 1, size(edg_nrml_y(i)%arr)
  write(1,'(//trim(fmtf)//',A)',advance='no') edg_nrml_y(i)%arr(j),','
end do
write(1,'(A)') ''

write(1,'(i0,A)',advance='no') edg_ID(i),'_normal_z,'
do j = 1, size(edg_nrml_z(i)%arr)
  write(1,'(//trim(fmtf)//',A)',advance='no') edg_nrml_z(i)%arr(j),','
end do
write(1,'(A)') ''

write(1,'(i0,A)',advance='no') edg_ID(i),'_voxels,'
do j = 1, size(edg_vox(i)%arr)
  write(1,'(i0,A)',advance='no') edg_vox(i)%arr(j),','
end do
write(1,'(A)') ''
end do

close(1)

end subroutine PrintEdgeAssociationInfo

subroutine PrintTriangleInfo
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_TriangleInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! ID, vrt1, vrt2, vrt3, edg1, edg2, edg3, cntr_x, cntr_y, cntr_z, area, nrml_x,
nrml_y, nrml_z, thickness, continuum, material

```

```

! header
write(1,'(A)')
'ID,vrt1,vrt2,vrt3,edg1,edg2,edg3,center_x,center_y,center_z,area,normal_x,normal
_y,normal_z,thickness,continuum,material'

! data
do i = 1, Ntri
  write(1,'(i0,A)',advance='no') tri_ID(i),', '
  write(1,'(i0,A)',advance='no') tri_vrt(1,i),', '
  write(1,'(i0,A)',advance='no') tri_vrt(2,i),', '
  write(1,'(i0,A)',advance='no') tri_vrt(3,i),', '
  write(1,'(i0,A)',advance='no') tri_edg(1,i),', '
  write(1,'(i0,A)',advance='no') tri_edg(2,i),', '
  write(1,'(i0,A)',advance='no') tri_edg(3,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_cntr(1,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_cntr(2,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_cntr(3,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_area(i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_nrml(1,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_nrml(2,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_nrml(3,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') tri_thk(i),', '
  write(1,'(i0,A)',advance='no') tri_cntm(i),', '
  write(1,'(i0,A)',advance='no') tri_mat(i),', '
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintTriangleInfo

subroutine PrintTriangleAssociationInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_TriangleAssociationInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

```

```

! write(*,'(A,i0)') trim(FILE_OUT)//' write flag: ',ios

do i = 1, Ntri
  write(1,'(i0,A)',advance='no') tri_ID(i),'_voxels,'
  do j = 1, size(tri_vox(i)%arr)
    write(1,'(i0,A)',advance='no') tri_vox(i)%arr(j),' '
  end do
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintTriangleAssociationInfo

subroutine PrintContinuumInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_ContinuumInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0
j = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! write(*,'(A,i0)') trim(FILE_OUT)//' write flag: ',ios

do j = 1, Ncnt
  write(1,'(i0,A)',advance='no') cnt_ID(j),' '
  write(1,'(i0,A)',advance='no') cnt_mat(j),' '
  write(1,'(//fmtf//',A)',advance='no') cnt_area(j),' '
  do i = 1, size(cnt_tri(j)%arr(:))
    write(1,'(i0,A)',advance='no') cnt_tri(j)%arr(i),' '
  end do
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintContinuumInfo

```

```

subroutine PrintLoadInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_LoadInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0
j = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

do j = 1, Nlod
  write(1,'(i0,A)',advance='no') lod_ID(j),', '
  write(1,'(i0,A)',advance='no') lod_type(j),', '
  write(1,'(i0,A)',advance='no') lod_func(j),', '
  write(1,'( '//fmtf//',A)',advance='no') sum(lod_cvrtarea(j)%arr(:)),', '
  write(1,'( '//fmtf//',A)',advance='no') sum(lod_cvrtlod(j)%arr(:)),', '
  do i = 1 , size(lod_cnt(j)%arr(:))
    write(1,'(i0,A)',advance='no') lod_cnt(j)%arr(i),', '
  end do
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintLoadInfo

subroutine PrintManufacturedTemperatureInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

```

```

FILE_OUT = FILE_OUT_MFGTEMPERATURE

open(unit=1,file=trim(FILE_OUT),access='append',action='write',status='old',iostat=ios)

! Iteration,
! header
if ( i_solve .eq. 0 ) then
  write(1,'(A)',advance='no') 'Iteration,'
  do i = 1, Nvrt
    write(1,'(i0,A)',advance='no') vrt_ID(i),', '
  end do
  write(1,'(A)') ''
end if

! data
write(1,'(i0,A)',advance='no') i_solve,', '
do i = 1, Nvrt
  write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_tmpm(i),', '
end do
write(1,'(A)') ''

close(1)

end subroutine PrintManufacturedTemperatureInfo

subroutine PrintMaterialInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_MaterialInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0
j = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! ID, name, k, rho, cp, mu, e, r, t
! header

```



```

write(1,'(A)')
'ID,Name,Thermal_Conductivity,Mass_Density,Specific_Heat,Dynamic_Viscosity,Emissi
vity,Reflectivity,Transmissivity'

! data
do i = 1, Nmat
  write(1,'(i0,A)',advance='no') mat_ID(i),', '
  write(1,'(A,A)',advance='no') mat_name(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_k(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_rho(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_cp(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_mu(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_e(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_r(i),', '
  write(1,'(//trim(fmtf)//',A)',advance='no') mat_t(i),', '
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintMaterialInfo

subroutine PrintSolveVertexInfo
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_SolveVertexInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

do i = 1 , size(VrtSolv)
  write(1,'(i0,A)') vrt_ID(VrtSolv(i)),', '
end do

close(1)

end subroutine PrintSolveVertexInfo

subroutine PrintNoSolveVertexInfo
use AllModules

```

```

implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_NoSolveVertexInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

do i = 1 , size(VrtNoSolv)
    write(1,'(i0,A)') vrt_ID(VrtNoSolv(i)),','
end do

close(1)

end subroutine PrintNoSolveVertexInfo

subroutine PrintTemperatureInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_OUT = FILE_OUT_TEMPERATURE

open(unit=1,file=trim(FILE_OUT),access='append',action='write',status='old',iostat=ios)

! Iteration,
! header
if ( i_solve .eq. 0 ) then
    write(1,'(A)',advance='no') 'Iteration,'
    do i = 1, Nvrt
        write(1,'(i0,A)',advance='no') vrt_ID(i),','
    end do

```

```

        write(1,'(A)') ''
    end if

    ! data
    write(1,'(i0,A)',advance='no') i_solve,', '
    do i = 1, Nvrt
        write(1,'( '//trim(fmtf)//',A)',advance='no') vrt_tmp(i),', '
    end do
    write(1,'(A)') ''

    close(1)

end subroutine PrintTemperatureInfo

subroutine PrintVoxelAssociationInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: m ! index
integer :: n ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_VoxelAssociationInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! header

! data
do i = 1, Nvxx*Nvxy*Nvxz
    write(1,'(i0,A)',advance='no') vox_ID(i),'_vertices,'
    do j = 1, size(vox_vrt(i)%arr)
        write(1,'(i0,A)',advance='no') vox_vrt(i)%arr(j),', '
    end do
    write(1,'(A)') ''

    write(1,'(i0,A)',advance='no') vox_ID(i),'_edges,'
    do j = 1, size(vox_edg(i)%arr)
        write(1,'(i0,A)',advance='no') vox_edg(i)%arr(j),', '
    end do
    write(1,'(A)') ''

```

```

        write(1,'(i0,A)',advance='no') vox_ID(i),'_triangles,'
        do j = 1, size(vox_tri(i)%arr)
            write(1,'(i0,A)',advance='no') vox_tri(i)%arr(j),','
        end do
        write(1,'(A)') ''
    end do

close(1)

end subroutine PrintVoxelAssociationInfo

subroutine PrintVoxelInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_VoxelInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! header
write(1,'(A)') 'Voxel,x,y,z,x_lo,x_hi,y_lo,y_hi,z_lo,z_hi'

! data
do i = 1, Nvxx*Nvxy*Nvxz
    write(1,'(i0,A)',advance='no') vox_ID(i),','
    write(1,'(i0,A)',advance='no') vox_crd(1,i),','
    write(1,'(i0,A)',advance='no') vox_crd(2,i),','
    write(1,'(i0,A)',advance='no') vox_crd(3,i),','
    write(1,'( '//trim(fmtf)//',A)',advance='no') vox_lim_lo(1,i),','
    write(1,'( '//trim(fmtf)//',A)',advance='no') vox_lim_hi(1,i),','
    write(1,'( '//trim(fmtf)//',A)',advance='no') vox_lim_lo(2,i),','
    write(1,'( '//trim(fmtf)//',A)',advance='no') vox_lim_hi(2,i),','
    write(1,'( '//trim(fmtf)//',A)',advance='no') vox_lim_lo(3,i),','
    write(1,'( '//trim(fmtf)//',A)',advance='no') vox_lim_hi(3,i),','
    write(1,'(A)') ''
end do

close(1)

```

```

end subroutine PrintVoxelInfo

subroutine PrintConductanceMatrixOffAxisInfo
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_ConductanceMatrixOffAxisInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slr)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! Edge ID, Ge_x, Ge_y, Ge_z, Ge_tot
! header
write(1,'(A)') 'Edge_ID,Gx,Gy,Gz,Gtot'

! data
do i = 1, Nedg
  write(1,'(i0,A)',advance='no') edg_ID(i),','
  write(1,'( '//trim(fmtf)//',A)',advance='no') Ge_x(i),','
  write(1,'( '//trim(fmtf)//',A)',advance='no') Ge_y(i),','
  write(1,'( '//trim(fmtf)//',A)',advance='no') Ge_z(i),','
  write(1,'( '//trim(fmtf)//',A)',advance='no') (Ge_x(i)+Ge_y(i)+Ge_z(i)),','
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintConductanceMatrixOffAxisInfo

subroutine PrintConductanceMatrixDiagonalInfo
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

```

```

ios = 0

FILE_NAME = 'Model_ConductanceMatrixDiagonalInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

! Edge ID, Gv_x, Gv_y, Gv_z, Gv_tot
! header
write(1,'(A)') 'Vertex_ID,Gx,Gy,Gz,Gtot'

! data
do i = 1, Nvrt
  write(1,'(i0,A)',advance='no') vrt_ID(i),',',
  write(1,'(' '//trim(fmtf)//',A)',advance='no') Gv_x(i),',',
  write(1,'(' '//trim(fmtf)//',A)',advance='no') Gv_y(i),',',
  write(1,'(' '//trim(fmtf)//',A)',advance='no') Gv_z(i),',',
  write(1,'(' '//trim(fmtf)//',A)',advance='no') (Gv_x(i)+Gv_y(i)+Gv_z(i)),',',
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintConductanceMatrixDiagonalInfo

subroutine PrintSourceInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_SourceInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slt)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

open(unit=1,file=trim(FILE_OUT),action='write',status='replace',iostat=ios)

!Vrt_ID, Direct Source, Radiative Source, Boundary Source, Total Source

```

```

! header
write(1,'(A)')
'Vertex_ID,Direct_Source,Radiative_Source,Boundary_Source,Total_Source'

! data
do i = 1, Nvrt
  write(1,'(i0,A)',advance='no') vrt_ID(i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') vrt_src(1,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') vrt_src(2,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no') vrt_src(3,i),', '
  write(1,'(///trim(fmtf)//',A)',advance='no')
(vrt_src(1,i)+vrt_src(2,i)+vrt_src(3,i)),', '
  write(1,'(A)') ''
end do

close(1)

end subroutine PrintSourceInfo

subroutine PrintResidualInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_OUT = FILE_OUT_RESIDUAL

open(unit=1,file=trim(FILE_OUT),access='append',action='write',status='old',iosta
t=ios)

! Iteration,
! header
if ( i_solve .eq. 0 ) then
  write(1,'(A)',advance='no') 'Iteration,'
  do i = 1, Nvrt
    write(1,'(i0,A)',advance='no') vrt_ID(i),', '
  end do
  write(1,'(A)') ''
end if

! data
write(1,'(i0,A)',advance='no') i_solve,', '
do i = 1, Nvrt
  write(1,'(///fmtf//',A)',advance='no') vrt_res(i),', '
end do
write(1,'(A)') ''

```

```

close(1)

end subroutine PrintResidualInfo

subroutine PrintManufacturedErrorInfo
use AllModules
implicit none

integer :: i ! index
integer :: j ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_OUT = FILE_OUT_MFGERROR

open(unit=1,file=trim(FILE_OUT),access='append',action='write',status='old',iosta
t=ios)

! Iteration,
! header
if ( i_solve .eq. 0 ) then
  write(1,'(A)',advance='no') 'Iteration,'
  do i = 1, Nvrt
    write(1,'(i0,A)',advance='no') vrt_ID(i),', '
  end do
  write(1,'(A)') ''
end if

! data
write(1,'(i0,A)',advance='no') i_solve,', '
do i = 1, Nvrt
  write(1,'(///fmte///,A)',advance='no') vrt_mfe(i),', '
end do
write(1,'(A)') ''

close(1)

end subroutine PrintManufacturedErrorInfo

subroutine EstablishTemperatureFile
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

```



```

ios = 0

FILE_NAME = 'Model_TemperatureInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

FILE_OUT_TEMPERATURE = FILE_OUT

open(unit=1,file=trim(FILE_OUT_TEMPERATURE),action='write',status='replace',iostat=ios)

close(1)

end subroutine EstablishTemperatureFile

subroutine EstablishManufacturedTemperatureFile
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_ManufacturedTemperatureInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

FILE_OUT_MFGTEMPERATURE = FILE_OUT

open(unit=1,file=trim(FILE_OUT_MFGTEMPERATURE),action='write',status='replace',iostat=ios)

close(1)

end subroutine EstablishManufacturedTemperatureFile

subroutine EstablishManufacturedErrorFile
use AllModules
implicit none

integer :: i ! index

```

```

integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_ManufacturedErrorInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

FILE_OUT_MFGERROR = FILE_OUT

open(unit=1,file=trim(FILE_OUT_MFGERROR),action='write',status='replace',iostat=i
os)

close(1)

end subroutine EstablishManufacturedErrorFile

subroutine EstablishResidualFile
use AllModules
implicit none

integer :: i ! index
integer :: ios ! I/O flag
character(len=200) :: FILE_NAME
character(len=200) :: FILE_EXT
character(len=200) :: FILE_OUT
logical :: fe ! file exists flag

ios = 0

FILE_NAME = 'Model_ResidualInfo'
FILE_EXT = '.txt'

FILE_OUT = trim(dir_slst)//'\ '//trim(FILE_NAME)//trim(FILE_EXT)
i = 0

INQUIRE(FILE=trim(FILE_OUT), EXIST=fe)

FILE_OUT_RESIDUAL = FILE_OUT

open(unit=1,file=trim(FILE_OUT_RESIDUAL),action='write',status='replace',iostat=i
os)

close(1)

end subroutine EstablishResidualFile

```

APPENDIX C: GMSH INPUT FILES

In order to systematically generate the meshes used for the studies in this body of research, input files were created for the Gmsh⁸ mesh generation tool. Following are input files that were used to generate the meshes used in the thermal analyses.

Circle Pore

```
// mesh size (m)
// h = 0.00390625 ;

// cell side length (m)
s = 1.0 ;

// pore radius (m)
// r = 0.1250 ;

// square vertices
Point(1) = {0.0, 0.0, 0.0, h} ;
Point(2) = {s, 0.0, 0.0, h} ;
Point(3) = {s, s, 0.0, h} ;
Point(4) = {0.0, s, 0.0, h} ;

// pore vertices
Point(5) = {s/2.0-r, s/2.0, 0.0, h} ;
Point(6) = {s/2.0, s/2.0, 0.0, h} ;
Point(7) = {s/2.0+r, s/2.0, 0.0, h} ;

// square edges
Line(8) = {1,2} ;
Line(9) = {2,3} ;
Line(10) = {3,4} ;
Line(11) = {4,1} ;

// circle arcs
Circle(12) = {5,6,7} ;
Circle(13) = {7,6,5} ;

// square boundary
Line Loop(14) = {8,9,10,11} ;

// circle boundary
Line Loop(15) = {12,13} ;
```

⁸ Geuzaine, C. and Remacle, J.-F., 2009, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering* **79**(11), pp. 1309-1331.

```
// porous domain
Plane Surface(16) = {14,15} ;
```

Circle Filler

```
// mesh sizes (m)
ms[] = {0.25 ,0.125,0.0625,0.03125,0.015625} ;

// porosities
porosity = { 0.02, 0.05 , 0.15 , 0.25 , 0.35 , 0.45 , 0.55 , 0.65, 0.70, 0.75,
0.78 } ;

// cell side length (m)
L = 1.0 ;

For m In {1:11} // porosity
For k In {1:5} // mesh size

// mesh size (m)
h = ms[k-1] ;

// porosity (m2/m2)
alpha = porosity[m-1] ;

// pore radius (m)
r = Sqrt( L*L*alpha/Pi ) ;

NewModel ;

// number of geometries
N_GEOM = 0;
N_POINTS = 0;
N_LINES = 0;
N_LOOPS = 0;
N_SURFACES = 0;

// define vertices
N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { 0.0 , 0.0 , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { L/2.0 , 0.0 , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { L/2.0 , L/2.0 - r , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { L/2.0 , L/2.0 , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { L/2.0 , L/2.0 + r , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { L/2.0 , L , 0.0 , h };
```

```

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { 0.0 , L , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { 0.0 , L/2.0 , 0.0 , h };

N_GEOM = N_GEOM + 1;
Point(N_GEOM) = { L/2.0 - r , L/2.0 , 0.0 , h };

N_POINTS = N_GEOM;

// edges
N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 1 , 2 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 2 , 3 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 3 , 4 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 4 , 5 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 5 , 6 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 6 , 7 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 7 , 8 } ;

N_GEOM = N_GEOM + 1 ;
Line(N_GEOM) = { 8 , 1 } ;

N_GEOM = N_GEOM + 1 ;
Circle(N_GEOM) = { 3 , 4 , 9 } ;

N_GEOM = N_GEOM + 1 ;
Circle(N_GEOM) = { 9 , 4 , 5 } ;

N_EDGES = N_GEOM - N_POINTS;

// regions
N_GEOM = N_GEOM + 1 ;
Line Loop(N_GEOM) = { 10 , 11 , 18 , 19 , 14 , 15 , 16 , 17 } ;

N_GEOM = N_GEOM + 1 ;
Line Loop(N_GEOM) = { 12 , 13 , -19 , -18 } ;

N_LOOPS = N_GEOM - N_POINTS - N_EDGES;

// surfaces

```

```

N_GEOM = N_GEOM + 1 ;
Plane Surface(N_GEOM) = {20} ;

N_GEOM = N_GEOM + 1 ;
Plane Surface(N_GEOM) = {21} ;

N_SURFACES = N_GEOM - N_POINTS - N_EDGES - N_LOOPS;

Mesh 2 ;

Mesh.Format = 31 ;
Mesh.BdfFieldFormat = 0 ;
Mesh.LabelType = 1 ;

fn = Sprintf('2-
Phase_Pore_Circle_Centered_Porosity_%g_Mesh_%g.bdfgm',alpha*1000000,h*1000000);
// baseline output file name
Save StrReplace(fn,'_000000_P','_P') ;

EndFor
EndFor

```

Triangle Pore

```

// mesh sizes (m)
ms[] = {0.25,0.125,0.0625,0.03125,0.015625} ;

// porosities
porosity = { 0.01 , 0.025 , 0.05 , 0.075 , 0.1 , 0.125 , 0.15 , 0.175 , 0.2 ,
0.225 , 0.25 , 0.275 , 0.3 , 0.31 } ;

// angles
angle = { 0 , 2.5 , 5 , 7.5 , 10 , 12.5 , 15 , 17.5 , 20 , 22.5 , 25 , 27.5 , 30
} ;

// cell side length (m)
L = 1.0 ;

For m In {1:14} // porosity 6
For j In {1:13} // angle 6
For k In {1:5} // mesh size 5

// mesh size (m)
h = ms[k-1] ;

// porosity (m2/m2)
alpha = porosity[m-1] ;

// number of sides on polygon
N_SIDES = 3 ;

// clocking (degrees)
CLOCK = angle[j-1] ;
Clock_c = -Cos((CLOCK+180)*Pi/180.0);

```

```

Clock_s = Sin((CLOCK+180)*Pi/180.0);

// centroid
cx = 0.5 ;
cy = 0.5 ;

// pore radius (m)
r = Sqrt( L*L*alpha/(N_SIDES*Sin(Pi/N_SIDES)*Cos(Pi/N_SIDES)) ) ;

// base
B = Sqrt(2.0*alpha/Sin(60.0*Pi/180.0));

// height
H = B*Sin(60.0*Pi/180.0);

NewModel ;

// define vertices
Point(1) = { 0.0 , 0.0 , 0.0 , h }; // 1
Point(2) = { L/2.0 , 0.0 , 0.0 , h };
Point(3) = { L , 0.0 , 0.0 , h }; // 2
Point(4) = { L , L/2.0 , 0.0 , h };
Point(5) = { L , L , 0.0 , h }; // 3
Point(6) = { L/2.0 , L , 0.0 , h };
Point(7) = { 0.0 , L , 0.0 , h }; // 4
Point(8) = { 0.0 , L/2.0 , 0.0 , h };

Point(9) = { cx , cy , 0.0 , h };

Point(10) = { Clock_c*(B/2.0) + Clock_s*(-H/3.0) + cx, -Clock_s*(B/2.0) +
Clock_c*(-H/3.0) + cy , 0.0 , h };
Point(11) = { Clock_c*(-B/2.0) + Clock_s*(-H/3.0) + cx, -Clock_s*(-B/2.0) +
Clock_c*(-H/3.0) + cy , 0.0 , h };
Point(12) = { Clock_c*(0.0) + Clock_s*(2.0*H/3.0) + cx, -Clock_s*(0.0) +
Clock_c*(2.0*H/3.0) + cy , 0.0 , h };

// edges
Line(1) = { 1 , 2 } ; // 1
Line(2) = { 2 , 3 } ; // 2
Line(3) = { 3 , 4 } ; // 3
Line(4) = { 4 , 5 } ; // 4
Line(5) = { 5 , 6 } ; // 1
Line(6) = { 6 , 7 } ; // 2
Line(7) = { 7 , 8 } ; // 3
Line(8) = { 8 , 1 } ; // 4

Line(9) = { 10 , 11 } ; // 5
Line(10) = { 11 , 12 } ; // 6
Line(11) = { 12 , 10 } ; // 7

// regions
Line Loop(1) = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ;
Line Loop(2) = { 9 , 10 , 11 } ;

// surfaces

```

```

Plane Surface(1) = {1 , 2} ;

Mesh 2 ;

Mesh.Format = 31 ;
Mesh.BdfFieldFormat = 0 ;
Mesh.LabelType = 1 ;

fn =
Sprintf('Triangle_Porosity_%g_Rot_%g_Mesh_%g.bdfgm',alpha*1000000,CLOCK*10,h*1000
000); // baseline output file name
Save StrReplace(fn,'_000000_P','_P') ;

EndFor
EndFor
EndFor

```

Square Pore

```

// mesh sizes (m)
ms[] = {0.25,0.125,0.0625,0.03125,0.015625} ;

// porosities
porosity = { 0.01 , 0.025 , 0.05 , 0.075 , 0.1 , 0.125 , 0.15 , 0.175 , 0.2 ,
0.225 , 0.25 , 0.275 , 0.3 , 0.325 , 0.35 , 0.375 , 0.4 , 0.425 , 0.45 , 0.475 ,
0.49 } ;

// angles
angle = { 0 , 2.5 , 5 , 7.5 , 10 , 12.5 , 15 , 17.5 , 20 , 22.5 , 25 , 27.5 , 30
, 32.5 , 35 , 37.5 , 40 , 42.5 , 45 } ;

// cell side length (m)
L = 1.0 ;

For m In {1:21} // porosity 1:9
For j In {1:19} // angle 1:10
For k In {1:5} // mesh size 1:5

// mesh size (m)
h = ms[k-1] ;

// porosity (m2/m2)
alpha = porosity[m-1] ;

// clocking (degrees)
CLOCK = angle[j-1] ;
Clock_c = -Cos((CLOCK-45)*Pi/180.0);
Clock_s = Sin((CLOCK-45)*Pi/180.0);

// number of sides on polygon
N_SIDES = 4 ;

// number of geometries
N_GEOM = 0;

```



```

// centroid
cx = 0.5 ;
cy = 0.5 ;

// pore radius (m)
r = Sqrt( L*L*alpha/(N_SIDES*Sin(Pi/N_SIDES)*Cos(Pi/N_SIDES)) ) ;

// base
B = Sqrt(alpha*L*L);

NewModel ;

// define vertices
Point(1) = { 0.0 , 0.0 , 0.0 , h }; // 1
Point(2) = { L/2.0 , 0.0 , 0.0 , h };
Point(3) = { L , 0.0 , 0.0 , h }; // 2
Point(4) = { L , L/2.0 , 0.0 , h };
Point(5) = { L , L , 0.0 , h }; // 3
Point(6) = { L/2.0 , L , 0.0 , h };
Point(7) = { 0.0 , L , 0.0 , h }; // 4
Point(8) = { 0.0 , L/2.0 , 0.0 , h };

Point(9) = { cx , cy , 0.0 , h };

Point(10) = { Clock_c*(B/2.0) + Clock_s*(-B/2.0) + cx, -Clock_s*(B/2.0) +
Clock_c*(-B/2.0) + cy , 0.0 , h }; // 5
Point(11) = { Clock_c*(-B/2.0) + Clock_s*(-B/2.0) + cx, -Clock_s*(-B/2.0) +
Clock_c*(-B/2.0) + cy , 0.0 , h }; // 6
Point(12) = { Clock_c*(-B/2.0) + Clock_s*(B/2.0) + cx, -Clock_s*(-B/2.0) +
Clock_c*(B/2.0) + cy , 0.0 , h }; // 7
Point(13) = { Clock_c*(B/2.0) + Clock_s*(B/2.0) + cx, -Clock_s*(B/2.0) +
Clock_c*(B/2.0) + cy , 0.0 , h }; // 8

// edges
Line(1) = { 1 , 2 } ; // 1
Line(2) = { 2 , 3 } ; // 2
Line(3) = { 3 , 4 } ; // 3
Line(4) = { 4 , 5 } ; // 4
Line(5) = { 5 , 6 } ; // 1
Line(6) = { 6 , 7 } ; // 2
Line(7) = { 7 , 8 } ; // 3
Line(8) = { 8 , 1 } ; // 4
Line(9) = { 10 , 11 } ; // 5
Line(10) = { 11 , 12 } ; // 6
Line(11) = { 12 , 13 } ; // 7
Line(12) = { 13 , 10 } ; // 8

// regions
Line Loop(1) = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ;
Line Loop(2) = { 9 , 10 , 11 , 12 } ;

// surfaces
Plane Surface(1) = { 1 , 2 } ; // 1

```

```

Mesh 2 ;

Mesh.Format = 31 ;
Mesh.BdfFieldFormat = 0 ;
Mesh.LabelType = 1 ;

fn =
Sprintf('Square_Porosity_%g_Rot_%g_Mesh_%g.bdfgm',alpha*1000000,CLOCK*10,h*100000
0); // baseline output file name
Save StrReplace(fn,'_000000_P','_P') ;

EndFor
EndFor
EndFor

```

Ellipse Pore

```

// mesh sizes (m)
ms[] = {0.25,0.125,0.0625,0.03125,0.015625} ;

// porosities
porosity = { 0.01 , 0.025 , 0.05 , 0.075 , 0.1 , 0.125 , 0.15 , 0.175 , 0.2 ,
0.225 , 0.25 , 0.275 , 0.3 , 0.325 , 0.35 , 0.375 , 0.38 } ;

// angles
angle = { 0 , 2.5 , 5 , 7.5 , 10 , 12.5 , 15 , 17.5 , 20 , 22.5 , 25 , 27.5 , 30
, 32.5 , 35 , 37.5 , 40 , 42.5 , 45 , 47.5 , 50 , 52.5 , 55 , 57.5 , 60 , 62.5 ,
65 , 67.5 , 70 , 72.5 , 75 , 77.5 , 80 , 82.5 , 85 , 87.5 , 90 } ;

// cell side length (m)
L = 1.0 ;

// aspect ratio (major radius/minor radius)
aspect_ratio = 2.0 ;

For m In {1:17} // porosity
For j In {1:37} // angle
For k In {1:5} // mesh size

// mesh size (m)
h = ms[k-1] ;

// porosity (m2/m2)
alpha = porosity[m-1] ;

// clocking (degrees)
CLOCK = angle[j-1] ;
Clock_c = -Cos(CLOCK*Pi/180.0);
Clock_s = Sin(CLOCK*Pi/180.0);

// centroid
cx = 0.5 ;
cy = 0.5 ;

```

```

// pore radii (m)
rp = Sqrt( L*L*alpha/(2.0*Pi) ) ;
ra = aspect_ratio*rp;

NewModel ;

// define vertices
Point(1) = { 0.0 , 0.0 , 0.0 , h };
Point(2) = { L/2.0 , 0.0 , 0.0 , h };
Point(3) = { L , 0.0 , 0.0 , h };
Point(4) = { L , L/2.0 , 0.0 , h };
Point(5) = { L , L , 0.0 , h };
Point(6) = { L/2.0 , L , 0.0 , h };
Point(7) = { 0.0 , L , 0.0 , h };
Point(8) = { 0.0 , L/2.0 , 0.0 , h };

Point(9) = { cx , cy , 0.0 , h };

Point(10) = { Clock_c*(0.0) + Clock_s*(-ra) + cx, -Clock_s*(0.0) + Clock_c*(-ra)
+ cy , 0.0 , h };
Point(11) = { Clock_c*(-rp) + Clock_s*(0.0) + cx, -Clock_s*(-rp) + Clock_c*(0.0)
+ cy , 0.0 , h };
Point(12) = { Clock_c*(0.0) + Clock_s*(ra) + cx, -Clock_s*(0.0) + Clock_c*(ra) +
cy , 0.0 , h };
Point(13) = { Clock_c*(rp) + Clock_s*(0.0) + cx, -Clock_s*(rp) + Clock_c*(0.0) +
cy , 0.0 , h };

// edges
Line(1) = { 1 , 2 } ;
Line(2) = { 2 , 3 } ;
Line(3) = { 3 , 4 } ;
Line(4) = { 4 , 5 } ;
Line(5) = { 5 , 6 } ;
Line(6) = { 6 , 7 } ;
Line(7) = { 7 , 8 } ;
Line(8) = { 8 , 1 } ;
Ellipse(9) = { 10 , 9 , 11 , 11 } ;
Ellipse(10) = { 11 , 9 , 11 , 12 } ;
Ellipse(11) = { 12 , 9 , 11 , 13 } ;
Ellipse(12) = { 13 , 9 , 11 , 10 } ;

// regions
Line Loop(1) = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ;
Line Loop(2) = { 9 , 10 , 11 , 12 } ;

// surfaces
Plane Surface(20) = { 1 , 2 } ;

Mesh 2 ;

Mesh.Format = 31 ;
Mesh.BdfFieldFormat = 0 ;
Mesh.LabelType = 1 ;

```

```
fn =  
Sprintf('Ellipse_Porosity_%g_Arat_%g_Rot_%g_Msh_%g.bdfgm',alpha*1000000,aspect_ra  
tio*10,CLOCK*10,h*1000000); // baseline output file name  
Save StrReplace(fn,'_000000_P','_P') ;  
  
EndFor  
EndFor  
EndFor
```

APPENDIX D: SOLUTIONPLOT

SolutionPlot is a MATLAB® script written for the extraction and visualization of the results produced by a Solver2D simulation (or series of simulations). The script reads in the Solver2D solution files, extracting mesh and computational system characteristics, and provides multiple options for visualization. Although various versions of SolutionPlot were used for analyzing the simulation results in this research, the script presented here provides all the basic functionality for analyzing results generated by the Solver2D code included in this dissertation. Note that at times, SolutionPlot needed to be modified to meet specific needs, for example in cases of plotting heat flux vectors on temperature contour plots or generating figures for specific publication formatting requirements.

```
clear;
clc;

N_CASES = 3145;
i_CASE_START = 1;
i_CASE_STOP = 3145;
i_CASE_SET = [5:5:3145];

FOLDERNAME_SOLUTION_FILES_ROOT = "C:\Analyses\Thermal\Solution";

FILENAME_VERTEX_INFO = 'Model_VertexInfo.txt' % vertex file
FILENAME_TEMPERATURE_INFO = 'Model_TemperatureInfo.txt' % temperature file
FILENAME_MFGTEMPERATURE_INFO = 'Model_ManufacturedTemperatureInfo.txt' %
temperature file
FILENAME_SOURCE_INFO = 'Model_SourceInfo.txt' % temperature file
FILENAME_VOXEL_INFO = 'Model_VoxelInfo.txt' % voxel file
FILENAME_LOAD_INFO = 'Model_LoadInfo.txt' % load file
FILENAME_RESIDUAL_INFO = 'Model_ResidualInfo.txt' % temperature file
FILENAME_MFGERROR_INFO = 'Model_ManufacturedErrorInfo.txt' % temperature file
FILENAME_EDGE_INFO = 'Model_EdgeInfo.txt' % edge file
FILENAME_EDGEASSC_INFO = 'Model_EdgeAssociationInfo.txt' % edge association file
FILENAME_TRIANGLE_INFO = 'Model_TriangleInfo.txt' % triangle file
FILENAME_SOLVE_VERTEX_INFO = 'Model_SolveVertexInfo.txt' % solution vertex file
FILENAME_NO_SOLVE_VERTEX_INFO = 'Model_NoSolveVertexInfo.txt' % no solution
vertex file
FILENAME_CONTINUUM_INFO = 'Model_ContinuumInfo.txt' % continuum file
FILENAME_VERIFICATION_INFO = 'Model_Verification.txt' % verification data

FLAG_SOLUTION_LIST = 0; % 0 = no list just range; 1 = specified list
FLAG_MMS = 0; % 0 = non-MMS, 1 = MMS
```

```

FLAG_PLOT = 0; % 1 = plot, 0 = no plot
FLAG_SAVE_PLOT = 1; % 1 = save plot, 0 = no save plot
FLAG_CLOSE_PLOT = 0; % 1 = close plot after save, 0 = no close plot after save
FLAG_PLOT_MESH = 0; % 1 = plot mesh, 0 = no mesh
FLAG_PLOT_VOXEL = 0; % 1 = plot voxels, 0 = no voxels
FLAG_PLOT_EDGE_NORMAL = 0; % 1 = plot edge normal, 0 = no plot
FLAG_PLOT_SOLUTION = 1; % 1 = plot solution, 0 = no solution
FLAG_SOLUTION_TYPE = 'TMP'; % XCD = x-coordinate, YCD = y-coordinate, ZCD = z-
coordinate, TMP = temperature, SRC = source, RES = residual, MMS = manufactured
solution, MSR = manufactured source, MFE = error to manufactured solution, CNT =
continuum
FLAG_LABEL_SOLVE_VERTEX = 0; % 1 = show solve vertex IDs, 0 = no labels
FLAG_PLOT_SOLVE_VERTEX = 0; % 1 = show solve vertex, 0 = no show
FLAG_PLOT_NO_SOLVE_VERTEX = 0; % 1 = show no solve vertex, 0 = no show
FLAG_PLOT_LOAD = 0; % 1 = mark load entities, 0 = no mark
FLAG_LABEL_NO_SOLVE_VERTEX = 0; % 1 = show no solve vertex IDs, 0 = no labels
FLAG_SELECT_CONTINUUM = 0; % 0 = plot all continuums, 1 = plot selected
continuums
FLAG_SHOW_AXIS = 1; % 1 = include axes on plot; 2 = remove axes from plot
FLAG_LABEL_BOUNDARY = 0; % 1 = show boundary edge boundary IDs; 0 = no labels
FLAG_SET_COLOR_LIMITS = 1; % 1 = define contour value limits; 1 = no limit
definition
FLAG_COLORBAR = 1; % 1 = show colorbar; 0 = no colorbar
FLAG_TRIM_FIGURE = 0; % 0 = no trim, 1 = trim

% other parameters
cntm_slct = [1];
cntm_clr = [[1;0;0],[1;.75;0],[0;1;0],[0;0;1],[0;.8;1],[.00; .69;
.94],[.44;.19;.63]];

cntr_lim = [-1,1];

fig_scl = 1;
fig_type = '.png';
ticks_x = [];
ticks_y = [];

% initialized variables
vrf = {};
tmp_limits = [];
c_limits = [];

if ( FLAG_SOLUTION_LIST == 1 )
    i_CASE_SET = i_CASE_SET;
else
    i_CASE_SET = i_CASE_START : i_CASE_STOP;
end

i_COUNT = 0;
for i_CASE = i_CASE_SET
    i_COUNT = i_COUNT + 1;

FOLDERNAME_SOLUTION_FILES_APPENDIX =
sprintf(strcat('%0',num2str(floor(log10(N_CASES))+2),'i'),i_CASE)

```

```

FOLDERNAME_SOLUTION_FILES_FOLDER =
strcat(FOLDERNAME_SOLUTION_FILES_ROOT, '_', FOLDERNAME_SOLUTION_FILES_APPENDIX, '\')

if ( i_COUNT == 1 )
    vrf_temp =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_VERIFICATION_INFO));
    vrf = strsplit(vrf_temp.textdata{1,1}, ',');
    vrf{2,1} = cellstr(vrf_temp.textdata{2,1});
    for i = 2 : size(vrf,2)
        vrf{2,i} = vrf_temp.data(1,i-1);
    end
    clear vrf_temp;
else
    vrf_temp =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_VERIFICATION_INFO));
    vrf{i_COUNT + 1,1} = cellstr(vrf_temp.textdata{2,1});
    for i = 2 : size(vrf,2)
        vrf{i_COUNT + 1,i} = vrf_temp.data(1,i-1);
    end
    clear vrf_temp;
end

vrt = importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_VERTEX_INFO));
tmp =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_TEMPERATURE_INFO));
if ( FLAG_MMS == 1 )
    tmpm =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_MFGTEMPERATURE_INFO))
;
end
src = importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_SOURCE_INFO));
lod = importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_LOAD_INFO));
res =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_RESIDUAL_INFO));
if ( FLAG_MMS == 1 )
    mfe =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_MFGERROR_INFO));
end
edg = importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_EDGE_INFO));
edga =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_EDGEASSC_INFO));
tri =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_TRIANGLE_INFO));
vox = importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_VOXEL_INFO));
vrt_slv =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_SOLVE_VERTEX_INFO));
vrt_no_slv =
importdata(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_NO_SOLVE_VERTEX_INFO)
);
cntm = csvread(strcat(FOLDERNAME_SOLUTION_FILES_FOLDER, FILENAME_CONTINUUM_INFO));
cntm(cntm==0) = NaN;
msh_file_nm = cell2mat(vrf{size(vrf,1),1});
lod(lod==0) = NaN;

```

```

vrt = vrt.data;
vox = vox.data;
tmp = tmp.data;
if ( FLAG_MMS == 1 )
    tmpm = tmpm.data;
end
src = src.data;
if ( FLAG_MMS == 1 )
    srcm = src(:,1:2);
    for i_vrt = 1 : size(src,1)
        srcm(i_vrt,2) = -
500.0*pi*pi*cos(2.0*pi*vrt(i_vrt,2))*sin(pi*vrt(i_vrt,3)+0.75);
    end
end
res = res.data;
if ( FLAG_MMS == 1 )
    mfe = mfe.data;
end
edg = edg.data;
edga = edga.data;
tri = tri.data;

if ( FLAG_PLOT == 1 )
    % choose continuums to plot
    if ( FLAG_SELECT_CONTINUUM == 1 )
        cntm_slct = cntm(cntm_slct,1);
    else
        cntm_slct = cntm(:,1);
    end

    % create figure
    fig(i_CASE) = figure();

    % draw solution
    switch ( FLAG_SOLUTION_TYPE )
        case ( 'XCD' )
            vrt_col = 2;
        case ( 'YCD' )
            vrt_col = 3;
        case ( 'ZCD' )
            vrt_col = 4;
        case ( 'TMP' )
            vrt_col = 5;
        case ( 'RES' )
            vrt_col = 6;
        case ( 'SRC' )
            vrt_col = 7;
        otherwise
            vrt_col = 5;
    end
    if ( FLAG_PLOT_SOLUTION == 1 )
        for j = cntm_slct'
            disp(['Continuum ',num2str(j)])
            for i = cntm(j,4:size(cntm(j,:),2))
                if ( isnan(i) )

```



```

        continue
    end
    x = vrt(tri(i,2:4),2);
    y = vrt(tri(i,2:4),3);
    switch ( FLAG_SOLUTION_TYPE )
        case ( 'RES' )
            c = abs( res(size(res,1),tri(i,2:4)) );
        case ( 'TMP' )
            c = tmp(size(tmp,1),tri(i,2:4));
        case ( 'SRC' )
            c = src(tri(i,2:4),5);
        case ( 'MMS' )
            c = tmpm(size(tmpm,1),tri(i,2:4));
        case ( 'MSR' )
            c = srcm(tri(i,2:4),2);
        case ( 'MFE' )
            c = mfe(size(mfe,1),tri(i,2:4));
        case ( 'CNT' )
            c = cntm_clr(:,j);
        otherwise
            c = vrt(tri(i,2:4),vrt_col);
    end
    switch ( FLAG_SOLUTION_TYPE )
        case ( 'CNT' )
            if ( FLAG_PLOT_MESH == 1 )
                MESH_COLOR = 'k';
            else
                MESH_COLOR = 'none';
            end
        otherwise
            patch(x,y,c,'LineStyle','none');
    end
    hold on;
end
end

if ( FLAG_SET_COLOR_LIMITS == 1 )
    caxis(cntr_lim);
end

colorbar
colormap jet
end

% draw edges
if ( FLAG_PLOT_MESH == 1 )
    for i = 1:size(edg,1)
        plot(vrt(edg(i,2:3),2),vrt(edg(i,2:3),3),'k');
        hold on;
    end
end
end

```

```

% draw edge normals
if ( FLAG_PLOT_EDGE_NORMAL == 1 )
    for i = 1:size(edg,1)
        scatter(edg(i,4),edg(i,5),'g','filled');
        hold on;
        nrmlx = edga((i-1)*5+3,1)*edg(i,7);
        nrmlly = edga((i-1)*5+4,1)*edg(i,7);
        plot([edg(i,4),edg(i,4)+nrmlx],[edg(i,5),edg(i,5)+nrmlly],'g');
    end
end

% plot solve vertices
if ( FLAG_PLOT_SOLVE_VERTEX == 1 )
    for i = 1:size(vrt_slv,1)
        plot(vrt(vrt_slv(i),2),vrt(vrt_slv(i),3),'MarkerEdgeColor',[1 0
0],'LineStyle','none','Marker','.')
        hold on;
    end
end

% label solve vertices
if ( FLAG_LABEL_SOLVE_VERTEX == 1 )
    for i = 1:size(vrt_slv,1)
        text(vrt(vrt_slv(i),2),vrt(vrt_slv(i),3),[num2str(vrt_slv(i))],'Color',[1 0 0])
        hold on;
    end
end

% plot no solve vertices
if ( FLAG_PLOT_NO_SOLVE_VERTEX == 1 )
    for i = 1:size(vrt_no_slv,1)
        plot(vrt(vrt_no_slv(i),2),vrt(vrt_no_slv(i),3),'MarkerEdgeColor',[0 0
0],'LineStyle','none','Marker','.')
        hold on;
    end
end

% label no solve vertices
if ( FLAG_LABEL_NO_SOLVE_VERTEX == 1 )
    for i = 1:size(vrt_no_slv,1)
        text(vrt(vrt_no_slv(i),2),vrt(vrt_no_slv(i),3),[num2str(vrt_no_slv(i))],'Color',[
0 0 0])
        hold on;
    end
end

% label boundary edges
if ( FLAG_LABEL_BOUNDARY == 1 )
    for i = 1:size(edg,1)
        if edg(i,11) > 0
            text(edg(i,4),edg(i,5),[num2str(edg(i,11))],'Color',[0 0 0])
            hold on;
        end
    end
end

```

```

        end
    end

    % show voxel boundaries
    if ( FLAG_PLOT_VOXEL == 1 )
        for i = 1 : size(vox,1)
            plot([vox(i,2),vox(i,2)],[vox(1,3),vox(size(vox,1),3)],'Color',[.5 .5
.5]);
            plot([vox(1,2),vox(size(vox,1),2)],[vox(i,3),vox(i,3)],'Color',[.5 .5
.5]);
            hold on;
        end
    end

    % mark load entities
    if ( FLAG_PLOT_LOAD == 1 )
        for i = 1:size(lod,1)
            for j = lod(i,6:size(lod,2))
                if ( isnan(j) )
                    continue
                else
                    for k = cntm(j,4:size(cntm,2))
                        if ( isnan(k) )
                            continue
                        else
                            % mark all triangle in triangle (columns 8 and 9)
scatter(tri(k,8),tri(k,9),'MarkerEdgeColor','k','MarkerFaceColor','r');
                            hold on;
                        end
                    end
                end
            end
        end
    end

    axis equal
    axis tight
    set(gca,'TickDir','out');
    xticks(ticks_x);
    yticks(ticks_y);
    if ( FLAG_COLORBAR == 0 )
        colorbar off;
    end
    pos = get(gcf,"Position");
    set(gcf,'Position',[pos(1) fig_scl*pos(2) pos(3) fig_scl*pos(4)]);
    set(get(gca,"XAxis"),"FontSize",fig_scl*10);
    set(get(gca,"YAxis"),"FontSize",fig_scl*10);
    x_lim = xlim;
    y_lim = ylim;

    if ( FLAG_TRIM_FIGURE == 1 )
        set(gca,"Units","inches");
        set(gcf,"Units","inches");
        pos = get(gca,"Position");

```

```

        ax_pos = [0 0 abs(x_lim(2)-x_lim(1))/abs(y_lim(2)-y_lim(1))*pos(4)
pos(4)];
        fig_pos = [2 2 ax_pos(3) ax_pos(4)];
        end

        if ( FLAG_SHOW_AXIS == 0 )
            set(gca,'XColor','none');
            set(gca,'YColor','none');
        else
            if ( FLAG_TRIM_FIGURE == 1 )
                if max(size(ticks_x,2),size(ticks_y)) > 0
                    ax_pos = [fig_scl*(0.2/0.59) fig_scl*(0.15/0.59) abs(x_lim(2)-
x_lim(1))/abs(y_lim(2)-y_lim(1))*pos(4) pos(4)];
                    fig_pos = [2 2 abs(x_lim(2)-x_lim(1))/abs(y_lim(2)-
y_lim(1))*pos(4)+fig_scl*(0.2/0.59)+0.5*fig_scl*(0.15/0.59)
pos(4)+1.5*fig_scl*(0.15/0.59)];
                end
            end
        end

        if ( FLAG_TRIM_FIGURE == 1 )
            set(gca,"Position",ax_pos);
            set(gcf,"Position",fig_pos);
        end

        drawnow

        % save plot
        if ( FLAG_SAVE_PLOT == 1 )
            FIGURE_TYPE = FLAG_SOLUTION_TYPE;
            if ( FLAG_PLOT_SOLUTION == 0 )
                FIGURE_TYPE = "MSH";
            end
            FILENAME_FIGURE =
strcat('PLOT_',FIGURE_TYPE,'_',FOLDERNAME_SOLUTION_FILES_APPENDIX,'_',msh_file_nm
(1:find(msh_file_nm=='-1')-1),fig_type)
            saveas(gcf,FILENAME_FIGURE);
            if ( FLAG_CLOSE_PLOT == 1 )
                close(gcf);
            end
        end

        % computer temperature limits
        tmp_limits = [tmp_limits;[min(tmp(size(tmp,1),:)),max(tmp(size(tmp,1),:))]];

        switch ( FLAG_SOLUTION_TYPE )
            case ( 'RES' )
                c_all = abs( res(size(res,1),tri(:,2:4)) )';
            case ( 'TMP' )
                c_all = tmp(size(tmp,1),tri(:,2:4))';
            case ( 'SRC' )
                c_all = src(tri(:,2:4),5);
            case ( 'MMS' )
                c_all = tmpm(size(tmpm,1),tri(:,2:4))';
        end

```

```

    case ( 'MSR' )
        c_all = srcm(tri(:,2:4),2);
    case ( 'MFE' )
        c_all = mfe(size(mfe,1),tri(:,2:4));
    otherwise
        c_all = vrt(tri(:,2:4),vrt_col);
end
c_limits = [c_limits;[min(c_all),max(c_all)]];
c_lim_min = min(c_limits(:,1));
c_lim_max = max(c_limits(:,2));

end

```

APPENDIX E: SOLVER2D USER MANUAL

Following is an abridged version of the user manual produced to for an end-to-end thermal analysis as performed in this research. The workflow covered includes mesh generation with Gmsh, computational simulation with Solver2D, and post-processing using SolutionPlot. Some of the information provided here may be redundant to information given in other appendices. Note that the user manual is for useful guidance and may not exactly match the Solver2D code and SolutionPlot script provided in other appendices.

1 Introduction

This section is intended to give a high-level description of three primary facets of the Solver2D program. First is a brief description of the computational context for the program, then is a description of the mesh generation software used, and finally is the purpose of the Solver2D program itself.

1.1 Finite Element Method

The finite element method (FEM) is used in Solve2D as a means of solving for the temperature distribution within a defined domain (or system) where heat is transferred via thermal conduction. This is accomplished by dividing the system into geometric elements (or regions), as seen in Figure 1, within which the mathematical equation describing the heat transfer process, the “heat equation,” can be solved. The process of dividing the original domain into the set of elements is referred to as “spatial discretization,” and the set of elements is referred to as the “mesh.” Specifically, the Galerkin FEM is used in Solver2D, which provides specific stipulations on how the governing heat transfer equation is solved.

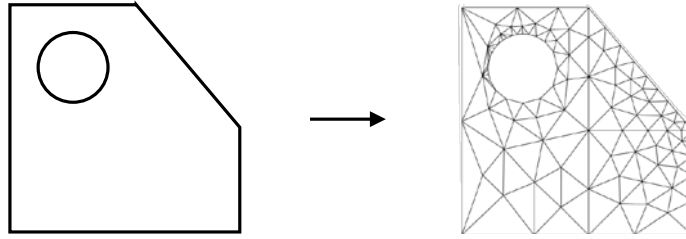


Figure 1. Finite element illustration

1.2 Gmsh Mesh Generation

Gmsh is a mesh generation software that provides convenient means for building a mesh and providing data structures to describe that mesh by either a graphical user interface (GUI) approach or a script-based approach [1]. The software is available for download at <http://gmsh.info/>. In this software environment, the user can build and define a domain for meshing as well as define specific meshing parameters that can be tailored for a specific analysis.

1.3 Solver2D Program

The Solver2D program is a processing and computational program, written in Fortran, that solves the heat equation across the domain using user-defined parameters and settings. This program configures input mesh information into compatible data structures in order to systematically perform solution calculations. The user defines mathematical expressions, material properties, and solution settings that are all systematically incorporated into the solution algorithm to solve the heat equation on the mesh based on the user-prescribed domain and program configuration.

2 Detailed Description

This section presents a detailed description of the three aspects of the Solver2D program described above, namely the Galerkin FEM approach, Gmsh mesh generation, and the Solver2D program. Figure 2 illustrates the architectural approach employed in using Solver2D to solve the heat equation on the computational domain. In general, the architecture is divided into two sections: configuration and solver.

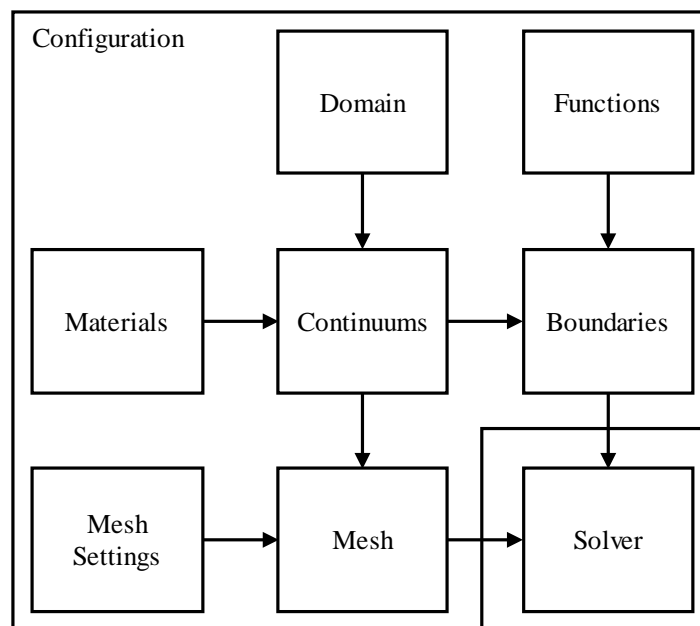


Figure 2. Architectural approach to computational heat transfer

The configuration section is the flow of the set of definitions and algorithms that is employed, fully defining the problem the user would like to solve. Required user inputs are definitions of the domain, mesh settings, materials, and functions. These all flow via Gmsh and Solver2D to produce continuums, boundaries, and a mesh. Ultimately, the mesh defines the discretization of the domain over which a thermal solution is sought, and the boundaries

constrain the problem solution. This information is then fed into the solver section which is found fully in the Solver2D program.

The illustrations shown in Figure 3 dissect the progression from geometric domain definition to thermal solution using the architecture described in Figure 2, as follows:

- a. Figure 3a illustrates an example of a domain defined by the user.
- b. The domain can be divided into various regions, referred to as “continuum,” as shown in Figure 3b, where each continuum represents a region of the domain to be meshed uniquely with a consistent material definition within each continuum.
- c. Figure 3c illustrates the various boundaries that are generated by the defined domain and continuums. Boundaries represent the free curves that define the shape of the domain, as well as the curves that define the interface of two different continuums.
- d. Figure 3d illustrates a resultant mesh over the entire domain, where each continuum is mesh uniquely within the enclosed region defined by boundaries.
- e. With mathematical functions (conditions) defined for each of the boundaries and material properties defined for each of the continuums, a thermal solution is found, an example contour plot of which is given in Figure 3e.

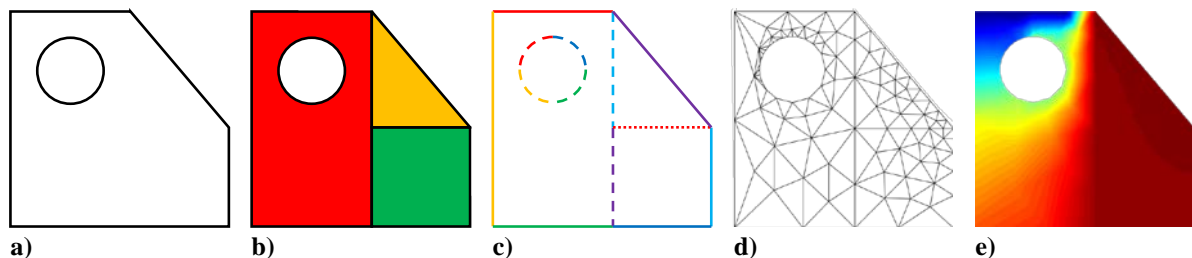


Figure 3. Computational domain progression to solution

The following sections describe necessary elements in the process described above.

2.1 Galerkin Finite Element Method

The purpose of Solver2D is to solve the governing two-dimensional steady-state heat equation. A detailed description of the Galerkin FEM formulation is given in a separate appendix.

2.2 Element Types and Shape Functions

Although many different finite element types exist, Solver2D only employs a single finite element type, the linear triangle. The following section describes the linear triangle element type.

2.2.1 Linear Triangle

A linear triangle element, like the one shown in Figure 4, assumed to lie in the xy -plane, has area A_t and is comprised of three vertices, with each i^{th} vertex of the triangle located at coordinates $(x_{t,i}, y_{t,i})$ with temperature $T_{t,i}$. The temperature field within the element is interpolated linearly between the three bounding vertex temperatures.

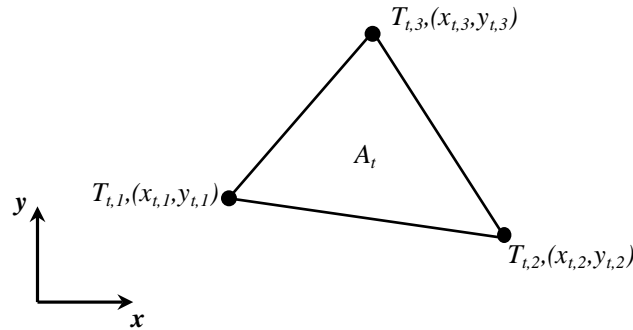


Figure 4. Notional linear triangle element diagram

2.3 Gmsh Mesh Generation

Gmsh is open-source freeware used for succinct mesh generation [1]. Gmsh allows the user to define the geometries over which mesh generation must be made, defining mesh parameters that are clear and easily manipulated. Likewise, Gmsh allows for user scripting, facilitating systemic geometry and mesh alterations that are consistent and automatic. Thorough documentation is available online for the complete use of Gmsh. In this context, a brief overview of the functionality of Gmsh is given to provide insight into the basic utility of the software with respect to Solver2D. The description of Gmsh given here is not an official documentation of the software but is merely an interpretation of the software description intended to facilitate contextual clarity for its use in conjunction with Solver2D.

2.3.1 *Domain Construction*

Gmsh allows for the physical construction of a domain, referred to as “physical groups” within Gmsh, using points, lines, surfaces, and volumes. The four physical groups can be built within three-dimensional space, where points represent discrete positions, lines are one-dimensional geometries (having length), surfaces are two-dimensional geometries (having area), and volumes are three-dimensional geometries (having volume). In Gmsh, the general construction of domains flows up serially by order of physical dimension, beginning with the zero-dimensional points up to the three-dimensional volumes, as illustrated in Figure 5. Points, lines, and surfaces are needed for construction of two-dimensional domains (like the domain shown in Figure 3a). Points, lines, surfaces, and volumes are needed for construction of three-dimensional domains. Note that surfaces are defined by closed line loops, where lines are linked continuously with no gaps.

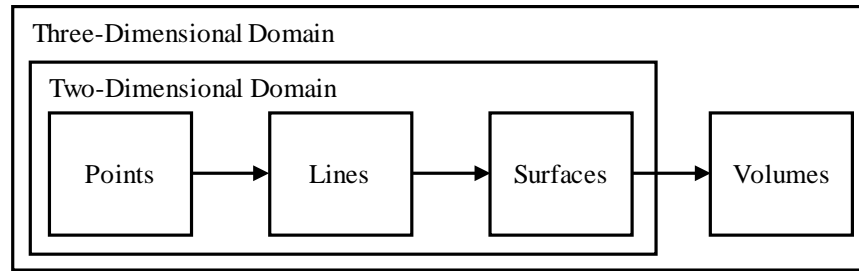


Figure 5. Gmsh domain construction order

In Gmsh, physical entities are indexed in order of creation, as illustrated in Figure 6. Figure 6a illustrates the creation of all domain-defining control points. Figure 6b shows the lines created by connecting points. Figure 6c shows the intermediary step of defining closed line loops by connecting physically-continuous lines. Figure 6d shows the resulting surfaces defined by the line loops. Subsequently, volumes could be further defined in a like manner.

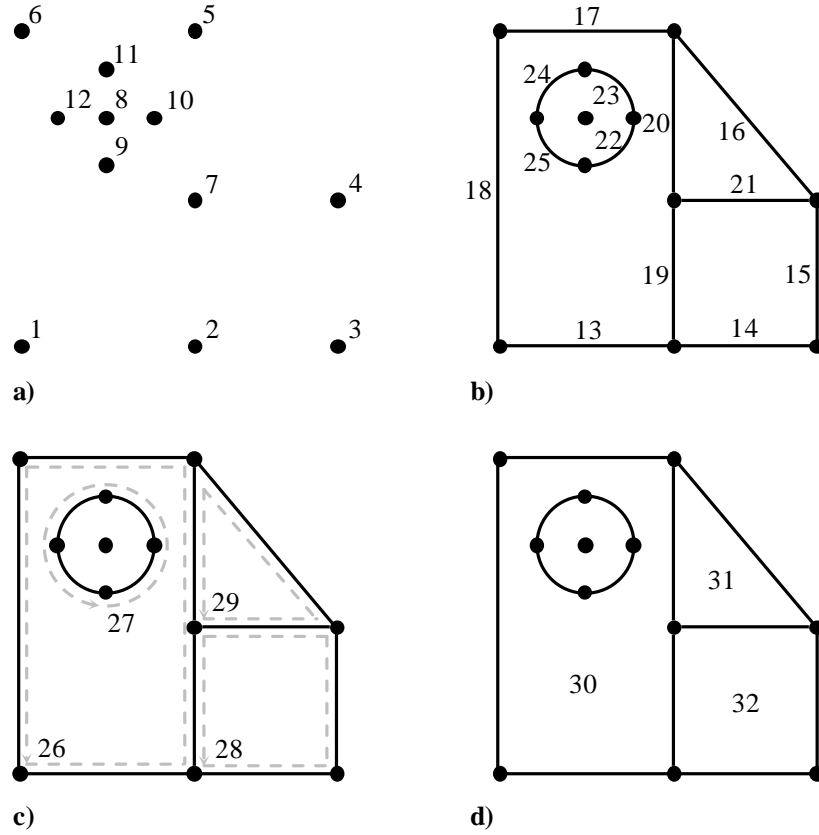


Figure 6. Gmsh physical entity indexing

2.3.2 Meshing

The second phase in Gmsh is meshing. As with the creation of the domain, a mesh is created sequentially through the four dimensions. When points are specified in Gmsh, the user must also specify a characteristic one-dimensional mesh size associated with the point. A point is specified as (x, y, z, h_p) , where x , y , and z are the location coordinates, and h_p is the point characteristic mesh size. For example, a point may be specified as $(1, 1, 0, 0.125)$, meaning the point is located at $x=1$, $y=1$, and $z=0$ with $h_p=0.125$. Once the domain is specified as described above, the domain is meshed along one-dimensional structures (lines). Figure 7 shows an example of the effects of point specification on one-dimensional meshing, where the black and

grey points are the specified control points and the resulting generated points, respectively. Notice that the spacing of the generated points in the region of the specified control points closely matches the specified characteristic mesh size of the control point. Likewise, one-dimensional meshing is followed by two-dimensional meshing, which is driven by the preceding one-dimensional meshing results. An example of a resultant two-dimensional mesh is shown in Figure 8, where the generated internal points are white with black outlines, and the grey lines connecting all points form an unstructured triangular mesh. As with the one-dimensional meshing, the separation of the generated internal points near the specified points is approximately the size of the specified h_p value of the specified point. Again, the same pattern follows for three-dimensional meshing.

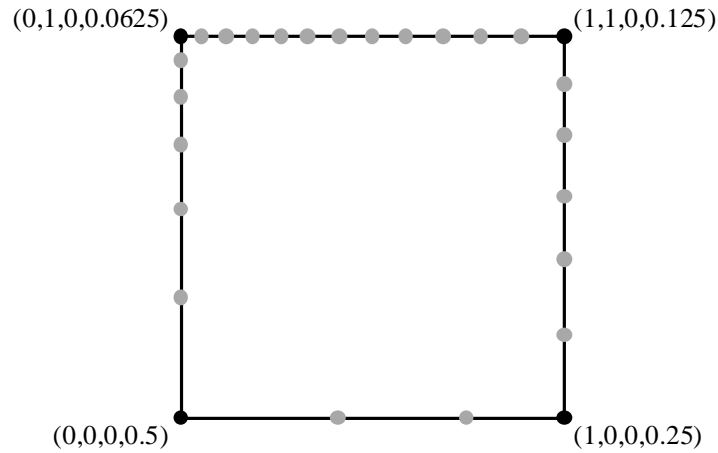


Figure 7. One-dimensional meshing example

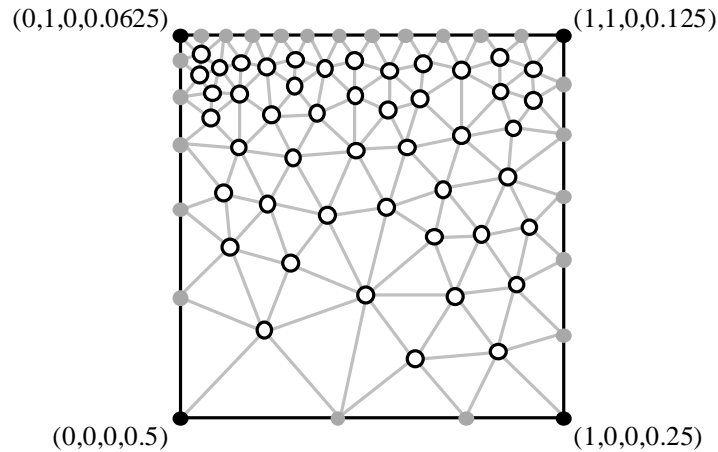


Figure 8. Two-dimensional meshing example

2.3.3 *Mesh Export File*

The last action required within Gmsh is exporting the resultant mesh. In its given state, Solver2D only accepts one file structure type, whereas Gmsh has the capability to output the mesh in a variety of file formats. The mesh must be exported in a text-based NASTRAN Bulk Data File using free field formatting and elementary entity element tags (not ignoring physical groups). Following this mesh export scheme, the exported mesh file can be read and processed by Solver2D. The Gmsh mesh export file will begin with the line:

\$ Created by Gmsh

The export file will end with the line:

ENDDATA

Each line in the body of the export file corresponds to some physical structure from the mesh domain construction and mesh generation steps, including vertices (points), edges (lines), and triangles. Portions of the output line data structures are used by Solver2D to conveniently and correctly build the thermal model. The following sections describe the output line data

structures of the mesh file and how the information is used, stored, and/or interpreted by Solver2D.

2.3.3.1 Vertex

1	2	3	4	5	6
---	---	---	---	---	---

1. Element type (character string)
2. Vertex element ID number (integer)
3. Unused (integer)
4. Vertex x-coordinate (real)
5. Vertex y-coordinate (real)
6. Vertex z-coordinate (real)

Example:

Line:

GRID,19,0,2.020000,0.525913,0.351703

Breakout:

GRID	19	0	2.020000	0.525913	0.351703
------	----	---	----------	----------	----------

Values:

1. Element type = GRID
2. Vertex ID number = 19
3. Unused
4. Vertex x-coordinate = 2.020000
5. Vertex y-coordinate = 0.525913
6. Vertex z-coordinate = 0.351703

2.3.3.2 Edge

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

1. Element type (character string)

2. Edge ID number (integer)
3. Edge assigned boundary ID number (integer)
4. Edge 1st vertex ID number (integer)
5. Edge 2nd vertex ID number (integer)
6. Unused
7. Unused
8. Unused

Example:

Line:

CBAR,78,5,5,18,0.000000,0.000000,0.000000

Breakout:

CBAR	78	5	5	18	0.000000	0.000000	0.000000
------	----	---	---	----	----------	----------	----------

Values:

1. Element type = CBAR
2. Edge ID number = 78
3. Edge assigned boundary ID number = 5
4. Edge 1st vertex ID number = 5
5. Edge 2nd vertex ID number = 18

2.3.3.3 Triangle

1	2	3	4	5	6
---	---	---	---	---	---

1. Element type (character string)
2. Triangle ID number (integer)
3. Triangle assigned continuum ID number (integer)
4. Triangle 1st vertex ID number (integer)
5. Triangle 2nd vertex ID number (integer)
9. Triangle 3rd vertex ID number (integer)

Example:

Line:

CTRIA3,1087,2,32,29,3

Breakout:

CTRIA3	1087	2	32	29	3
--------	------	---	----	----	---

Values:

1. Element type = CTRIA3
2. Triangle ID number = 1087
3. Triangle assigned continuum ID number = 2
4. Triangle 1st vertex ID number = 32
5. Triangle 1nd vertex ID number = 29
6. Triangle 3rd vertex ID number = 3

2.3.4 Summary

Once the use of Gmsh is complete, portions of the Configuration architecture shown in Figure 2 are fulfilled. The Domain is fully defined, the physical partitioning of Continuums is defined, as illustrated in Figure 6d, physical Boundaries are defined, as seen in Figure 6b, the Mesh Settings have been used, and a Mesh has been generated.

2.4 Solver2D Program

Solver2D follows a workflow that closes out incomplete portions of the Configuration architecture shown in Figure 2 and then executes the Solver portion according to the nominal process flow illustrated in Figure 9. In this routine, a “case” refers to a single analysis, comprised of all settings, mesh structures, material properties, and boundary conditions. Thus, N_{case} is the total number of cases run in a batch, and i_{case} is the index of a single case within that batch, ranging from 1 to N_{case} . Also, R_{lim} is the heat equation energy balance residual threshold value used as criteria for defining when the analysis on a given case is complete, and R_{crit} is the critical heat equation energy balance residual determined from the current solution

iteration. This section is dedicated to detailing the utility of the Solver2D code from a user perspective in order to solve the heat equation over the defined domain given user conditions.

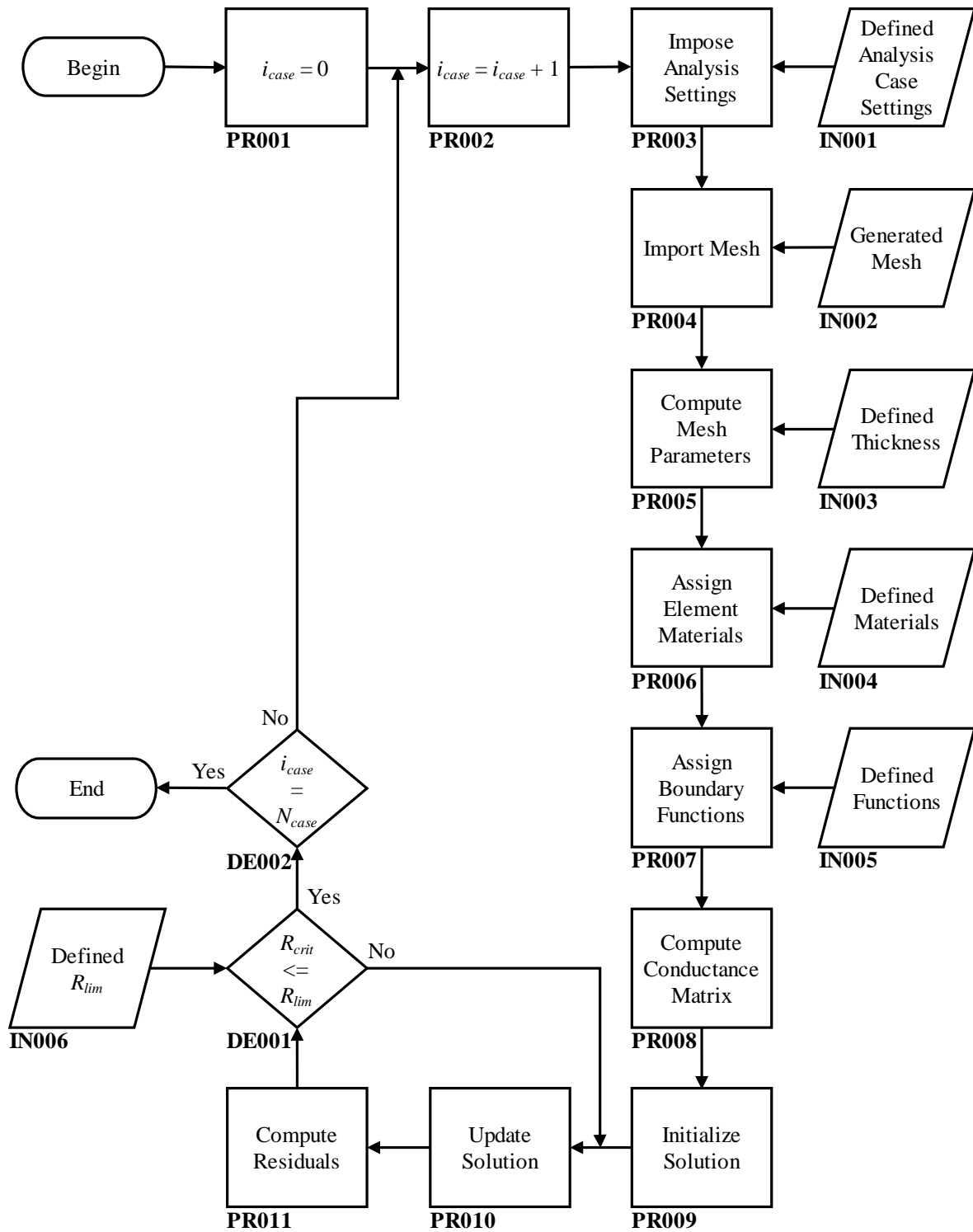


Figure 9. Solver2D nominal process flow

2.4.1 *Data Types and Variables*

All traditional Fortran data types are acceptable in Solver2D; however, custom data types and structures may be defined by the developer or user for convenience or special functionality. Thus, a module named *Types* exists in Solver2D for a centralized database of custom data types. Any developer-defined data types are visible in the *Types* module and should not be altered by the user. Likewise, the user may add its own data types to this module for use elsewhere in the code.

Two modules are in place for the declaration of global variables. The first module, *Variables*, contains developer-defined variables and should not be altered. The second module, *UserVariables*, is available for the user to declare new variables for use elsewhere in the code. When declaring a user variable, make sure that the name is not already used for any other user-defined or developer-defined variable. Note that the *Types* module is used in both the *Variables* and *UserVariables* modules, so any developer-defined or user-defined data types may be used when declaring a new variable.

2.4.2 *Functions*

Functions is a module used to store any developer-defined and user-defined functions so that the functions may be accessible globally within the code. In Solver2D, functions are mathematical input-output definitions that can be accessed by other routines and algorithms within the program. These defined functions satisfy the Inputs portion of the architecture in Figure 2 as well as input **IN005** set in Figure 9. *Functions* contains a function called *UserFunction* that has a structured input/return format so as to maintain uniformity for future development and to provide dynamic utility. The declaration of *UserFunction* looks like:

```

type(UserFuncNum) function
UserFunction(fi,N_intr,N_real,N_char,ar_intr,ar_real,ar_char)

```

UserFunction is of developer-defined data type *UserFuncNum*, which includes two structures, *intr_ar* and *real_ar*, where *intr_ar* and *real_ar* are independent, allocatable, and one-dimensional arrays of type integer and type double-precision real, respectively. This means that *UserFunction* can return a dynamically-sized array of integer values and/or a dynamically-sized array of double precision-real values. Inputs to *UserFunction* are tedious but simply structured. Table 1 details the inputs to *UserFunction*, where real(8) refers to the double-precision real data type.

Table 1. UserFunction input variable descriptions

Input Variable	Type	Description	Use/Purpose
<i>fi</i>	Integer	Function index	Point to selection of operations case
<i>N_intr</i>	Integer	Number of input integer array elements	Allocate size of dynamic input integer array
<i>N_real</i>	Integer	Number of input real array elements	Allocate size of dynamic input real array
<i>N_char</i>	Integer	Number of input character array elements	Allocate size of dynamic input character array
<i>ar_intr</i>	1D allocatable integer array	Input integer array	Store all necessary input integer parameters for selected operations case
<i>ar_real</i>	1D allocatable real(8) array	Input real array	Store all necessary input real parameters for selected operations case
<i>ar_char</i>	1D allocatable character array	Input character array	Store all necessary input character parameters for

The user can call *UserFunction* with arrays of integers, reals, and characters, each of independent lengths (including length zero). Even if no integer, real, or character inputs are required for the function operations, the user must still input at least empty arrays of the

specified data type. Likewise, the user must input the number of elements in each array, including the number 0 for zero-length arrays. In this manner, a function call to *UserFunction* is structured for dynamic use.

Once inside *UserFunction*, the routine uses *fi* to find the desired set of operations for the function call. The operations are segregated in a select case block, where each case block defines a unique set of operations (or function) for the function call. It is within each unique case block that the actual return values of *UserFunction* are allocated and defined. Each case block, including both developer-defined and user-defined blocks, should begin by allocating the *UserFunction%intr_ar* and *UserFunction%real_ar* structures built into the *UserFuncNum* data type assigned to the return value of *UserFunction*. Thus, each case block defines the number of integer elements and the number of real elements in the function integer and real arrays, respectively. Once the return arrays are allocated, the operations for that functional case block are defined, and values are assigned to the elements of the return arrays.

Although user-defined functions can be programmed, eight function definitions (numbered 0 through 7) are set aside for specific uses as follows:

0. **Null temperature.** This function returns a single real value of 0 that is intended to correspond to a scalar temperature value of 0.
1. **Null temperature gradient.** This function returns a 3-element real array, with elements equal to 0, that is intended to correspond to 0-valued temperature gradients in the x-direction, y-direction, and z-direction, respectively.
2. **Null heat flux.** This function returns a 3-element real array, with elements equal to 0, that is intended to correspond to 0-valued heat flux in the x-direction, y-direction, and z-direction, respectively.
3. **Null source term.** This function returns a single real value of 0 that is intended to correspond to a scalar source term value of 0.
4. **Manufactured solution temperature.** This function returns a single scalar value, corresponding to a manufactured solution temperature distribution definition.
5. **Manufactured solution temperature gradient.** This function returns a 3-element real array corresponding to the x-direction, y-direction, and z-direction spatial temperature gradients of the manufactured solution temperature distribution definition.

6. **Manufactured solution heat flux.** This function returns a 3-element real array corresponding to the x -direction, y -direction, and z -direction heat fluxes of the manufactured solution temperature distribution definition.
7. **Manufactured solution heat source.** This function returns a single real value corresponding to the source term from the manufactured solution temperature distribution definition.

Cases 0 through 3 should not be modified by the user. Cases 4 through 7 should be modified by the user in that the user should alter the operations but not the size of return arrays. Any additional cases should be written and modified by the user as needed, respecting the structure of the function call and return style.

Prior to defining the functional select case, a case diversion select case block is used for systematically diverting selected functions depending on if the analysis is specified as a manufactured solution analysis or as a traditional solution analysis. For clarification, if a problem is set up with a defined function for a temperature boundary value for the real analysis, a manufactured solution on the same domain would want the temperature boundary to be defined based on the manufactured solution temperature distribution. Thus, by changing a single analysis case flag variable, this *UserFunction* routine would divert from the real analysis operations block to the manufactured temperature operations block, all based on user specification.

Functions are intended to be centrally-located for easy access and convenient reference to the user as well as to code developers. Likewise, this allows for any function to be universally accessible to any part of the program analysis, regardless of its original intended use.

2.4.3 *Analysis Case Settings*

Analysis case settings can cover any number of parameters and variables in the code. The analysis case settings input **IN001** and the imposing of those settings in process **PR003**

from Figure 9 are covered by subroutine *SettingsDefinitions* but is also supported by subroutine *AnalysisDefinitions*. *AnalysisDefinitions* defines the N_{case} value as well as any other variables/parameters the user deems persistent across all analysis cases. *SettingsDefinitions* is a routine intended to specify unique settings for a given analysis. Settings and parameters that must be defined in either *AnalysisDefinitions* or in *SettingsDefinitions* are described in Table 2. Other variables or parameters that the user wishes to allocate or define between cases can be altered in the *SettingsDefinitions* subroutine. For example, a user may want to declare a thermal conductivity value that is used in a user-defined function, but the user may also want to change that thermal conductivity value from case to case. The variable can be declared in *UserVariables*, be used for computation in *UserFunction*, and be dynamically defined prior a given analysis case in *SettingsDefinitions*.

Table 2. Essential analysis case settings

Variable	Type	Description	Use/Purpose
<i>FLAG_MMS</i>	Integer	MMS flag	Indicate analysis is using MMS and to incorporate functions or operations relating to the manufactured solution analysis
<i>fntf</i>	Character	Format specification for output of real values	Control the output style of real values
<i>i_solve_MAX</i>	Integer	Maximum number of solution iterations	Limit the number of iterations allowed for a single analysis case
<i>res_lim</i>	Real(8)	Cutoff residual value, R_{lim}	Control the extent to which the solver resolves the heat equation energy balance
<i>MeshInputFileName</i>	Character	Name of mesh input file	Point to mesh file as generated in Gmsh
<i>fnte</i>	Character	Format specification for output of real scientific	Control the output style of real scientific (exponential) values

		(exponential) values	
<i>relax</i>	Real(8)	Solution update relaxation coefficient	Scale the updated temperature solution at each iteration
<i>FLAG_disp_inpt</i>	Integer	Input display flag	Control the display of the input file line text to the user

2.4.4 *Mesh Import*

The *ImportMesh* subroutine fulfills process **PR004** by reading in the mesh file generated in Gmsh and specified in the analysis case settings of input **IN001**. This subroutine converts the Gmsh NASTRAN data structures into data structures used by Solver2D. *ImportMesh* generates vertices, edges, and triangles from the mesh file, determines continuum segregations, identifies vertices and edges lying on boundaries, and identifies continuums with which triangle elements are associated. The function of the mesh should be transparent to the user, with the exception of information the routine provides to the user. The mesh file should be located in the same directory as the Solver2D program executable.

2.4.5 *Materials*

A material is a set of parameters defining the thermophysical and/or thermo-optical properties of some domain or boundary in the problem. The user may specify as many materials as it desires, N_{mat} . Specifically, a single material consists of multiple properties. These properties are stored in one-dimensional arrays by property, the lengths of which are defined by the total number of materials defined by the user. Table 3 describes material variables.

Table 3. Material property variables

Variable	Type	Description
N_{mat}	Integer	Number of defined materials, N_{mat}
mat_ID	1D allocatable real(8) array	Material ID

<i>mat_name</i>	1D allocatable character array	Name of material
<i>mat_k</i>	1D allocatable real(8) array	Thermal conductivity
<i>mat_rho</i>	1D allocatable real(8) array	Mass density
<i>mat_cp</i>	1D allocatable real(8) array	Specific heat
<i>mat_mu</i>	1D allocatable real(8) array	Kinematic viscosity
<i>mat_e</i>	1D allocatable real(8) array	Surface emissivity
<i>mat_r</i>	1D allocatable real(8) array	Surface reflectivity
<i>mat_t</i>	1D allocatable real(8) array	Surface transmissivity

The above-described parameters drive the thermal behavior of the computational solution of the system of interest. Material definitions are databased by the user in the *DefineMaterials* subroutine. The user must first define N_{mat} . The subroutine automatically allocates the property array variables and initializes the material properties. However, the user must manipulate a select case block to assign the material property values, where the case value is selected by the ID number of the material. This subroutine fulfills the Materials segment of Figure 2 as well as input **IN004** of Figure 9.

2.4.6 Continuum

Continuums are originally constructed in Gmsh, such as is shown by Figure 6d. However, the *DefineContinuumMaterials* subroutine assigns materials to the continuums, thus completing the Continuums portion of Figure 2 and process **PR006** of Figure 9. *DefineContinuumMaterials* loops through the total number of continuums, N_{cnt} , as created in Gmsh, and assigns defined materials to each unique continuum. This is performed using a select case block in which the user must ensure that each continuum in the mesh receives a

material assignment. Note that a single material may be assigned to more than one continuum, but no continuum may be without a material assignment. The assignment is made by using the desired material's ID. Continuum hold multiple parameters, as outlined in Table 4. The finite elements that comprise a given continuum are further assigned the associated continuum material in subroutine *AssignMaterialsToTriangles*.

Table 4. Continuum structure variables

Variable	Type	Description
<i>cnt_ID</i>	1D allocatable integer array	Continuum ID
<i>cnt_mat</i>	1D allocatable integer array	ID of assigned material
<i>cnt_area</i>	1D allocatable real(8) array	Total area of elements comprising continuum
<i>cnt_tri</i>	1D allocatable AllIntArr1D array	List of triangle element IDs comprising continuum

It is important to note that the continuum IDs are automatically and sequentially generated in *ImportMesh*, starting with 1. The IDs for the continua generated in Solver2D are based on the numerical order of the surface IDs designated in Gmsh. For example, the surfaces indexed in Figure 6d are 30, 31, and 32, thus the corresponding IDs in Solver2D would be 1, 2, and 3, respectively. This is also true for Gmsh-designated IDs that are not in consecutive order, such as 12, 14, and 7, which would receive the Solver2D-assigned IDs of 2, 3, and 1, respectively.

Lastly, the out-of-plane thickness of surface elements must also be defined. This definition is performed for each surface element in subroutine *DefineSurfaceThickness*. The user may define the thickness of the two-dimensional domain, satisfying input **IN003** from Figure 9.

2.4.7 Boundaries

Boundaries are determined by line definitions as constructed in Gmsh, such as are indexed in Figure 6b. Thus, any vertices (points) or edges that are generated coincidentally on a given line are determined to be associated with that boundary. Boundaries hold multiple parameters, as outlined in Table 5.

Table 5. Boundary structure variables

Variable	Type	Description
<i>bnd_ID</i>	1D allocatable integer array	Boundary ID
<i>bnd_name</i>	1D allocatable character array	Name of boundary
<i>bnd_type</i>	1D allocatable integer array	ID of boundary type
<i>bnd_func</i>	1D allocatable integer array	ID of assigned boundary function
<i>bnd_evrt</i>	1D allocatable AllIntArr1D array	List of vertex IDs coincident boundary
<i>bnd_edg</i>	1D allocatable AllIntArr1D array	List of edge IDs coincident boundary

The boundary descriptions are designated in subroutine *AssignBoundaryFunctions*.

Four different boundary type designations are available, indexed as follows:

- 1. **Intimate thermal contact.** This implies no special treatment is given to vertices or edges on a boundary.
1. **Temperature.** This indicates that the temperatures of vertices on a boundary are not solved but are defined exactly by user definition.
2. **Temperature gradient.** This indicates that the spatial temperature gradient at edges on a boundary is defined by the user.
3. **Heat flux.** This indicates that the directional heat flux across an edge on a boundary is defined by the user.

Values for the definitions of boundary types 1 through 3 must be defined using a user function, employed in the *UserFunction* subroutine. Thus, in *AssignBoundaryFunctions* a

select case block is used to assign both the boundary type index and the user function index that describes each boundary, where each boundary must be assigned a boundary type index and a user function. The combination *AssignBoundaryFunctions* and *UserFunction* completes the Boundary portion of Figure 2 and process **PR007** from Figure 9.

It is important to note that the boundary IDs are automatically and sequentially generated in *ImportMesh*, starting with 1. The IDs for the boundaries generated in Solver2D are based on the numerical order of the line IDs designated in Gmsh. For example, three lines indexed in Figure 6b are 13, 19, and 22. Thus, the corresponding boundary IDs in Solver2D would be 1, 7, and 10, respectively.

2.4.8 *User Subroutines*

Users may add subroutines to the source code but should beware of accidentally modifying or deleting other routines. To use all of the global data types, variables, and functions, make sure to use *AllModules* and *Functions* in the subroutine, where *AllModules* is a module that uses *Variables*, *UserVariables*, and *Types*.

2.4.9 *Compile and Run*

If modifications are made to the source code, as will be the case for any analysis setup, the source code must be saved. The code is written in the Fortran programming language, thus the code must be compiled using an acceptable Fortran compiler. The Intel® Fortran compiler was used in the development of the code, thus the Intel® Fortran Compiler is the recommended compiler for Solver2D.

Once an executable file is generated for the Solver2D program, the executable can be run to perform the programmed analyses.

Solver2D will output solution files for each analysis case in a unique folder titled “Solution_0#”, where # is replaced with the i_{case} index for the corresponding analysis case.

Output files include but are not limited to the following files:

1. **Model_VertexInfo.txt.** This file contains information for all model vertices, including vertex ID, coordinates, area, and volume.
2. **Model_EdgeInfo.txt.** This file contains information for all model edges, including edge ID, vertex IDs, midpoint coordinates, length, tangent unit vector, and assigned boundary ID.
3. **Model_TriangleInfo.txt.** This file contains information for all model triangle elements, including triangle ID, vertex IDs, edge IDs, centroid coordinates, area, surface normal vector, thickness, assigned continuum ID, and assigned material ID.
4. **Model_ContinuumInfo.txt.** This file contains information for each continuum in the model, including the continuum ID, assigned material ID, total area, and comprising triangle elements ID list.
5. **Model_TemperatureInfo.txt.** This file contains vertex temperature values for all model vertices at different solution iterations, minimally before the initial solver iteration and at the end of the final solution iteration.
6. **Model_SourceInfo.txt.** This file contains vertex source term values for all model vertices with direct, radiative, boundary, and total source term values enumerated.
7. **Model_ResidualInfo.txt.** This file contains vertex heat equation energy balance residual values for all model vertices at different solution iterations, minimally before the initial solver iteration and at the end of the final solution iteration.

2.4.10 Summary

Solver2D is a Fortran code that combines mesh information with user inputs to solve the heat equation. The program process mesh file data structures and systematically incorporates boundary conditions to solve for the temperature profile of a system.

3 SolutionPlot

SolutionPlot is a convenient tool used for visualization of the analysis solution, built specifically to compliment Solver2D and its data structures. SolutionPlot is an editable MATLAB® script that reads in Solver2D output files, extracts the data according to the

defined data structures and file formats, and plots various aspects of the corresponding analysis. The script has the ability to display the mesh or individual mesh structures, finite elements according to continuum assignment, continuous temperature distribution, continuous residual distribution, and continuous source distribution. This script is simple and easily modifiable to accommodate user needs.

4 Example Application

This section walks through an example two-dimensional application case, following the general process and tools describe in this manual, beginning with the domain description, followed by mesh generation in Gmsh, analysis setup in Solver2D, and solution visualization in SolutionPlot. For the mesh generation steps, screenshots are given to show the process using the graphical user interface, but text is also given to show how the user may employ script-based generation.

4.1 Domain Definition

This step establishes a reference system to be used for generating a mesh in Gmsh, finding a thermal solution in Solver2D, and performing visualization in SolutionPlot.

1. Define the system of interest.

Because of the various steps involved in coming to an analysis solution for a system, it can be useful to create a reference diagram (or simple visual model) of the domain of interest. The diagram shown in Figure 10, illustrates the dimensions and boundary conditions on the system that will be built and analyzed through the remainder of this example application.

This example system consists of multiple exterior and interior domain boundaries, multiple materials, and different types of boundary conditions.

This diagram could be referenced and/or annotated at various stages to make sure the right conditions are referenced. This is especially useful for complex domains when traversing from Gmsh to Solver2D.

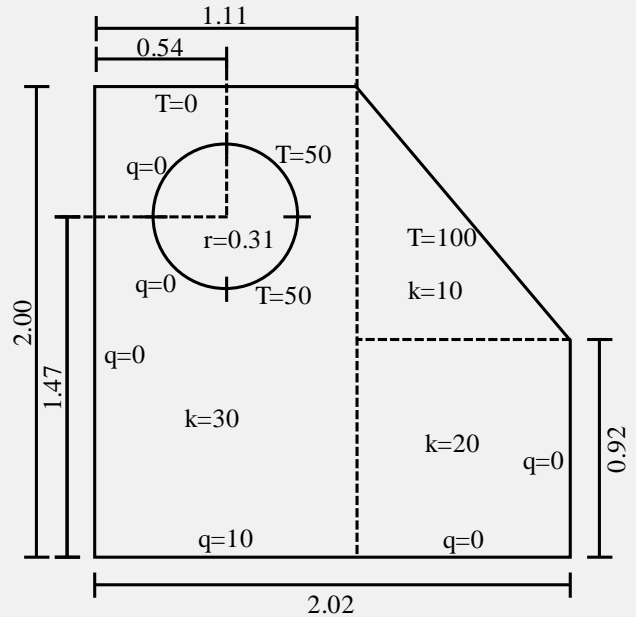


Figure 10. Example system diagram

4.2 Mesh Generation

The following steps walk through an example mesh generation process using Gmsh:

1. Open a new file in Gmsh.

If the geometry window is not blank, as shown in Figure 11, click File→Clear.

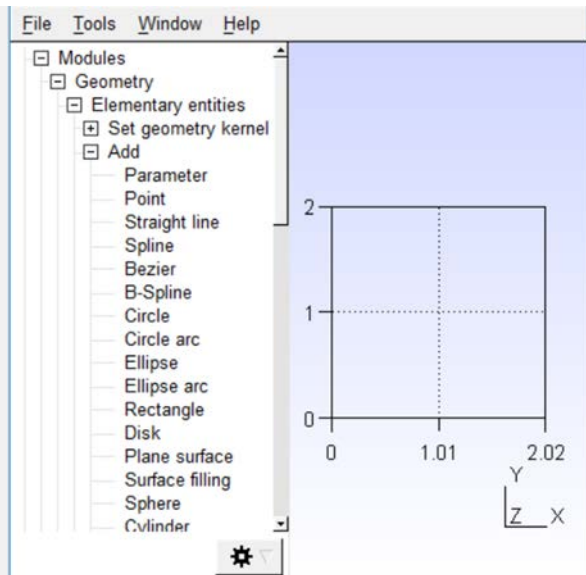


Figure 11. New Gmsh geometry window

2. Create 12 new points to define the domain.

Click Modules→Geometry→Elementary entities→Add→Point in the program tree on the left side of the window.

In the Elementary Entity Context window, enter “0” for the X-coordinate, “0” for the Y-coordinate, “0” for the Z-coordinate, and “1” for the prescribed mesh size at the point, as shown in Figure 12.

Click the Add button. This will add a point in the geometry window, but the Elementary Entity Context window will not close.

Enter the following 11 XYZ-coordinates and prescribed mesh sizes, clicking Add after entering each set:

2. 1.11 , 0.00 , 0.00 , 1.000
3. 2.02 , 0.00 , 0.00 , 0.500
4. 2.02 , 0.92 , 0.00 , 0.100
5. 1.11 , 2.00 , 0.00 , 0.200
6. 0.00 , 2.00 , 0.00 , 1.000
7. 1.11 , 0.92 , 0.00 , 0.500
8. 0.54 , 1.47 , 0.00 , 1.000
9. 0.54 , 1.16 , 0.00 , 0.200
10. 0.85 , 1.47 , 0.00 , 0.100
11. 0.54 , 1.78 , 0.00 , 0.050
12. 0.23 , 1.47 , 0.00 , 0.025

Once the last point has been added, close the Elementary Entity Context window, then press “q” to abort the addition of points. The 12 added points should be visible in the geometry window, as shown in Figure 13.

In the Gmsh script file, the language that performs this step is:

```
// Generate control points
Point(1) = {0, 0, 0, 1};
Point(2) = {1.11, 0, 0, 1};
Point(3) = {2.02, 0, 0, 0.5};
Point(4) = {2.02, 0.92, 0, 0.1};
Point(5) = {1.11, 2, 0, 0.2};
Point(6) = {0, 2, 0, 1};
Point(7) = {1.11, 0.92, 0, 0.5};
Point(8) = {0.54, 1.47, 0, 1};
Point(9) = {0.54, 1.16, 0, 0.2};
Point(10) = {0.85, 1.47, 0, 0.1};
Point(11) = {0.54, 1.78, 0, 0.05};
Point(12) = {0.23, 1.47, 0, 0.025};
```

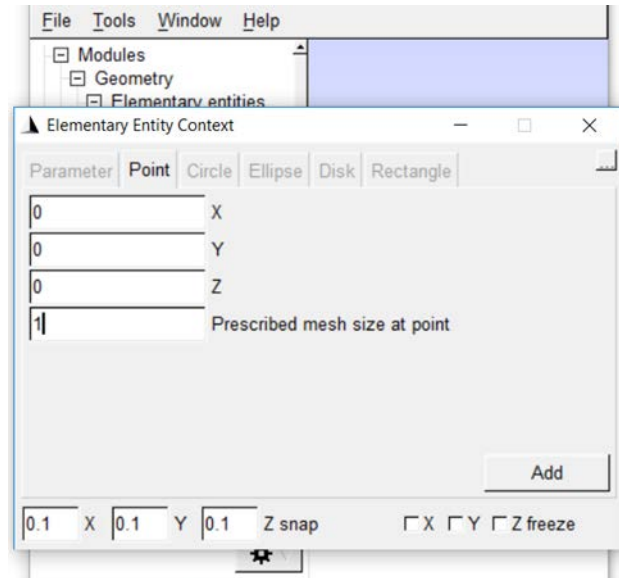


Figure 12. Gmsh Elementary Entity Context window

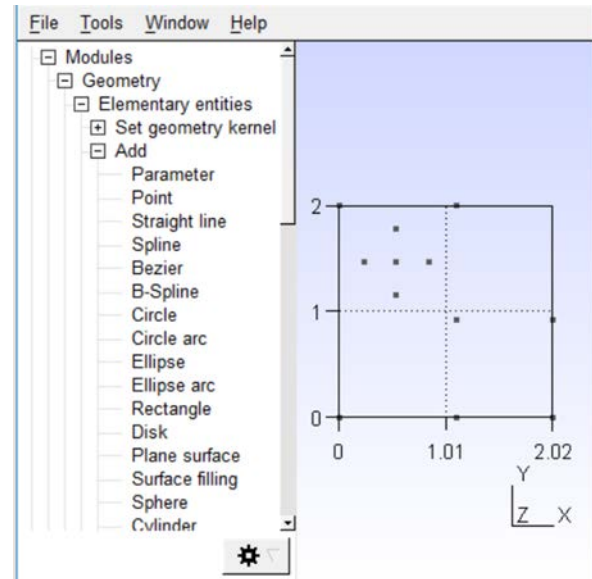


Figure 13. Gmsh added points

3. Create eight straight lines to define the domain.

Click Modules→Geometry→Elementary entities→Add→Straight line in the program tree on the left side of the window.

When Gmsh gives the prompt to select start point, as shown in Figure 14, click Point 1, located at (0.00,0.00,0.00).

The 1st point will become highlighted, and Gmsh will give the prompt to select end point, as shown in Figure 15.

Click Point 2. This will generate Line 1, which appears in blue, as shown in Figure 16, then Gmsh will prompt to select start point for another line.

Create the following eight straight lines, connecting points in the order shown:

2. 2 , 3
3. 3 , 4
4. 4 , 5
5. 5 , 6
6. 6 , 1
7. 2 , 7
8. 7 , 5
9. 7 , 4

Once the last straight line has been generated, press “q” to abort the addition of straight lines. The geometry should appear as seen in Figure 17.

The eight straight lines will each represent a unique boundary, identified and indexed by Solver2D in the order they were created here in Gmsh.

In the Gmsh script file, the language that performs this step is:

```
// Generate straight lines
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 5};
Line(5) = {5, 6};
Line(6) = {6, 1};
Line(7) = {2, 7};
Line(8) = {7, 5};
Line(9) = {7, 4};
```

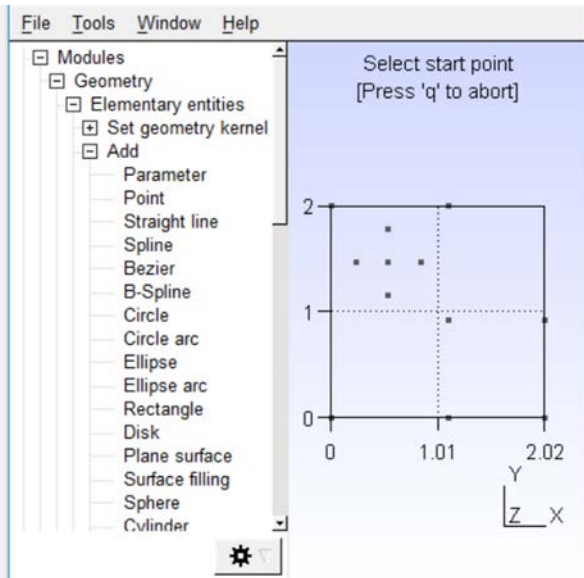


Figure 14. Gmsh straight line begin prompt

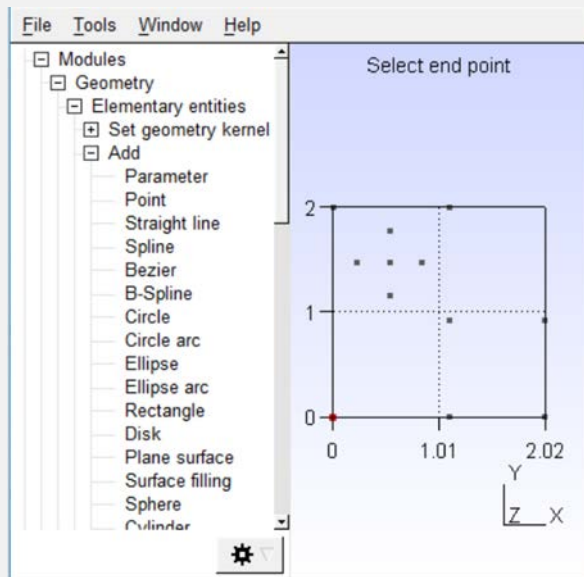


Figure 15. Gmsh straight line end prompt

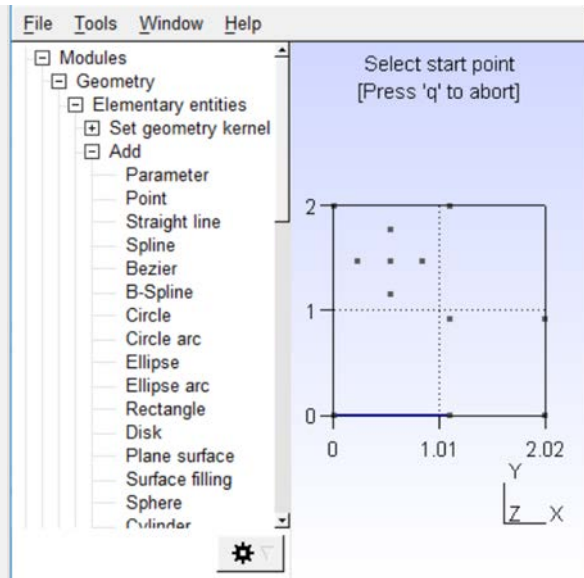


Figure 16. Gmsh first added straight line

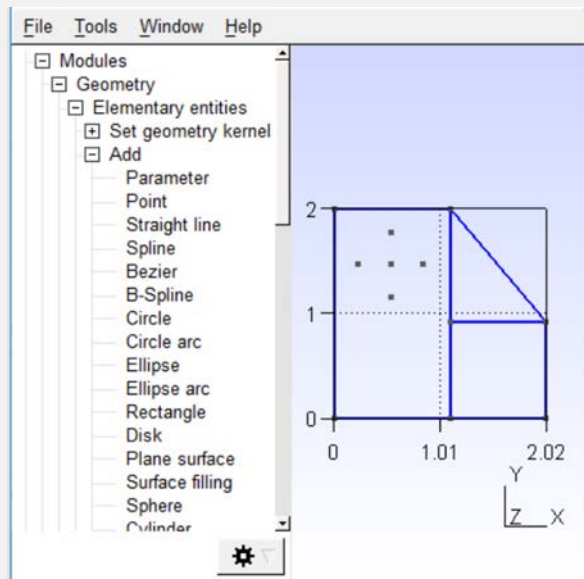


Figure 17. Gmsh added straight lines

4. Create four circle arcs to define the domain.

Click Modules→Geometry→Elementary entities→Add→Circle arc in the program tree on the left side of the window.

When Gmsh gives the prompt to select start point, as shown in Figure 18, click Point 9, located at (0.54,1.16,0.00).

The 9th point will become highlighted, and Gmsh will give the prompt to select center point, as shown in Figure 19.

Click Point 8. The 8th point will become highlighted, and Gmsh will give the prompt to select end point, as shown in Figure 20.

This will generate Circle 10, which appears in blue, as shown in Figure 21, then Gmsh will prompt to select start point for another circle arc. Note that straight lines and circle arcs are both variations of a general curve, hence the sequential indexing between the added straight lines and circle arc.

Create the following three circle arcs, connecting points in the order shown:

2. 10 , 8 , 11
3. 11 , 8 , 12
4. 12 , 8 , 9

Once the last circle arc has been generated, press “q” to abort the addition of circle arcs. The geometry should appear as seen in Figure 22.

The four circle arcs will each represent a unique boundary, identified and indexed by Solver2D in the order they were created here in Gmsh in sequence with the straight lines created in Step 3.

In the Gmsh script file, the language that performs this step is:

```
// Generate circle arcs
Circle(10) = {9, 8, 10};
Circle(11) = {10, 8, 11};
Circle(12) = {11, 8, 12};
Circle(13) = {12, 8, 9};
```

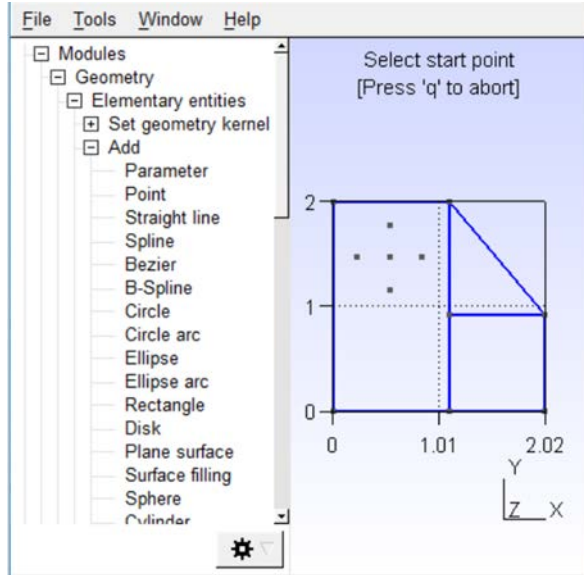


Figure 18. Gmsh circle arc begin point

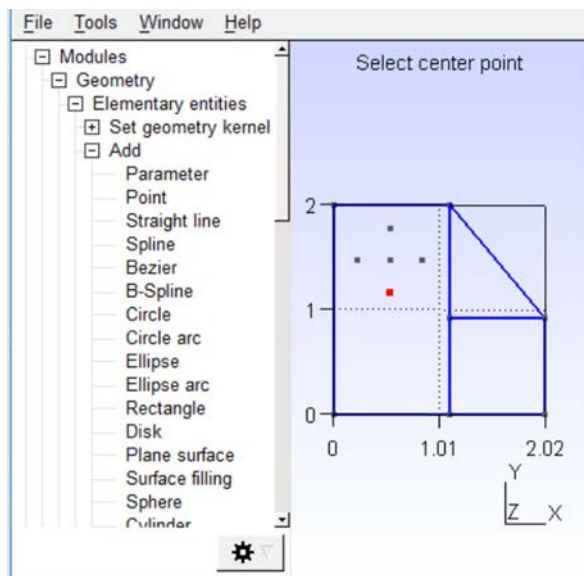


Figure 19. Gmsh circle arc center point

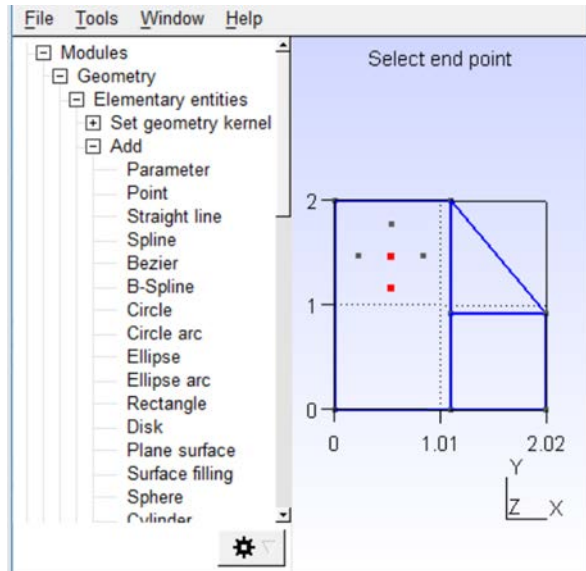


Figure 20. Gmsh circle arc end point

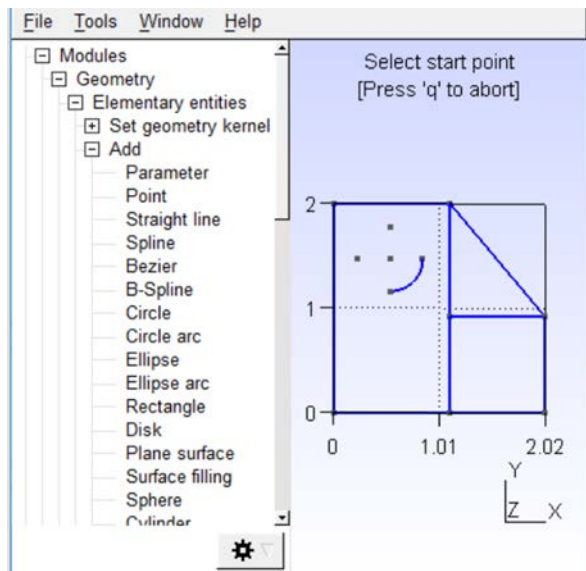


Figure 21. Gmsh first added circle arc

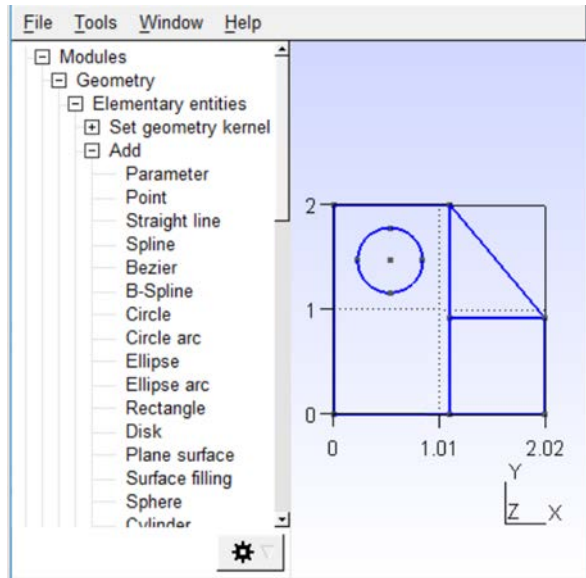


Figure 22. Gmsh added circle arcs

5. Create three plane surfaces to segregate the domain.

Click Modules→Geometry→Elementary entities→Add→Plane surface in the program tree on the left side of the window.

When Gmsh gives the prompt to select surface boundary, as shown in Figure 23, click Line 1, connecting Point 1 and Point 2.

The 1st, 5th, and 6th lines will become highlighted, and Gmsh will continue giving the prompt to select surface boundary, as shown in Figure 24.

Continue forming a full line loop by clicking Line 7 then Line 8. The 7th and 8th lines will become highlighted, as shown in Figure 25. Gmsh will also give the prompt to select hole boundaries, although the prompt is not visible in Figure 25.

As was done previously with the straight lines, click Circle 10, Circle 11, Circle 12, and Circle 13, forming a complete line loop hole. The four circle arcs will become highlighted in Gmsh after each selection, as shown in Figure 26.

Press “e” to end boundary selection and generate Plane 10. Gmsh will show dashed lines through the center of the generated surface and then prompt to select surface boundary for the next plane surface, as shown in Figure 27.

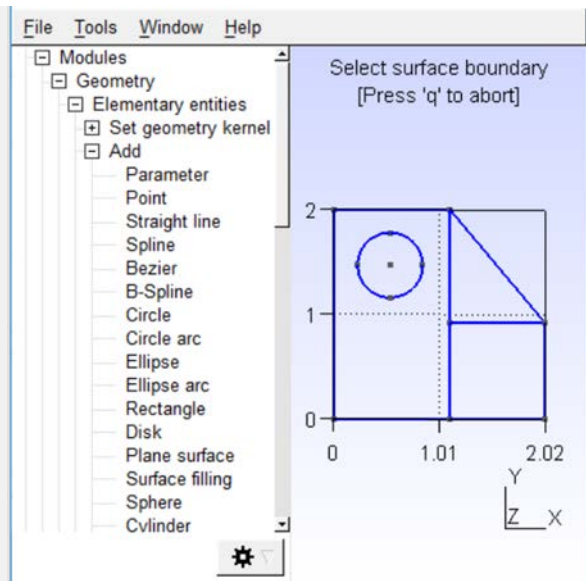


Figure 23. Gmsh plane surface begin surface boundary

Create the following two plane surfaces, clicking the straight lines in the order shown and pressing “e” after the sequence of line selections:

2. 2, 9, 7
3. 9, 4, 8

Once the last plane surface has been generated, press “q” to abort the addition of plane surfaces. The geometry should appear as seen in Figure 28.

The three plane surfaces will each represent a unique continuum, identified and indexed by Solver2D in the order they were created here in Gmsh.

In the Gmsh script file, the language that performs this step is:

```
// Generate plane surfaces
Line Loop(1) = {1, 7, 8, 5, 6};
Line Loop(2) = {10, 11, 12, 13};
Plane Surface(1) = {1, 2};
Line Loop(3) = {2, 3, -9, -7};
Plane Surface(2) = {3};
Line Loop(4) = {9, 4, -8};
Plane Surface(3) = {4};
```

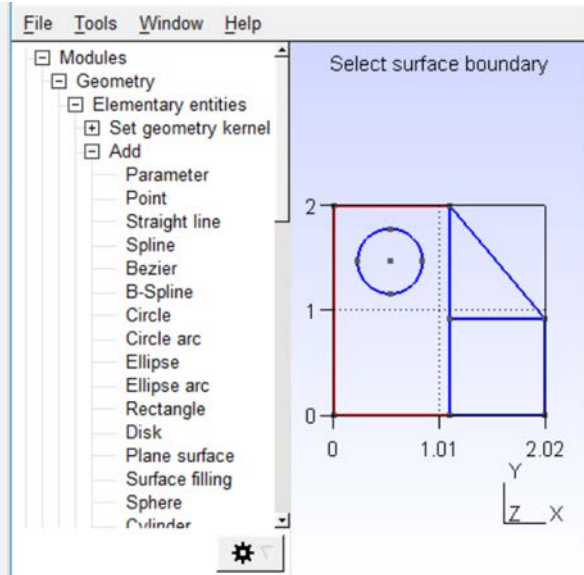


Figure 24. Gmsh selected surface boundary

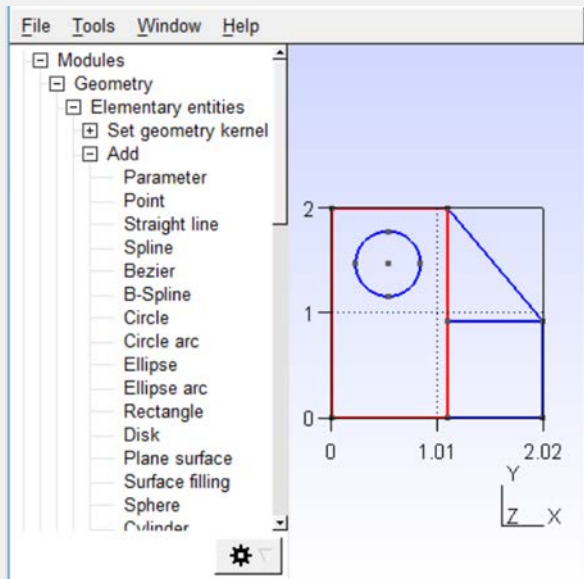


Figure 25. Gmsh hole boundary

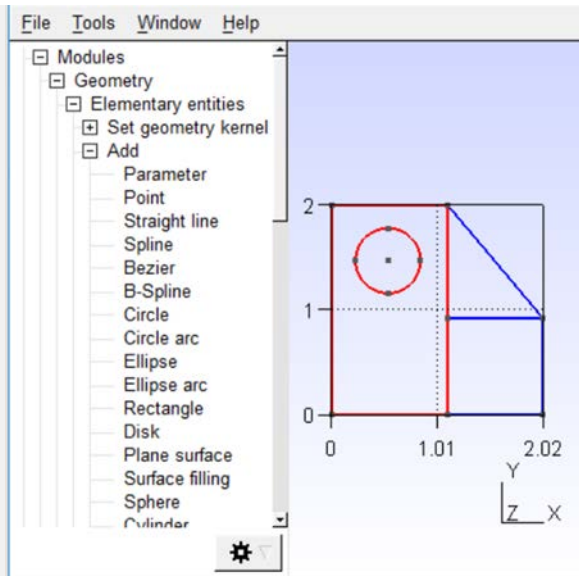


Figure 26. Gmsh selected hole boundaries

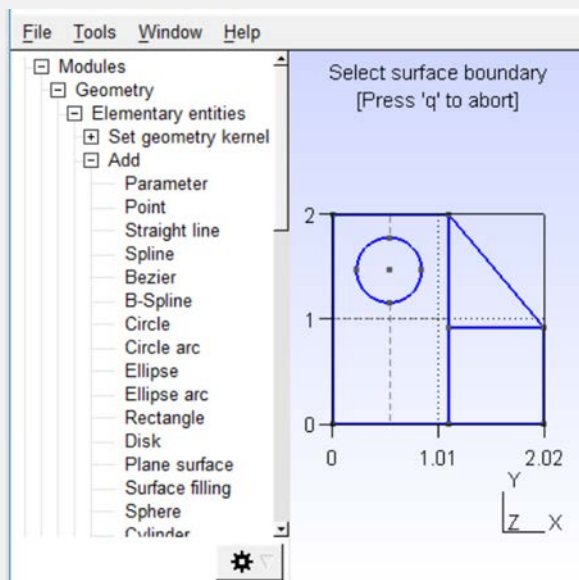


Figure 27. Gmsh plane surface end surface boundary

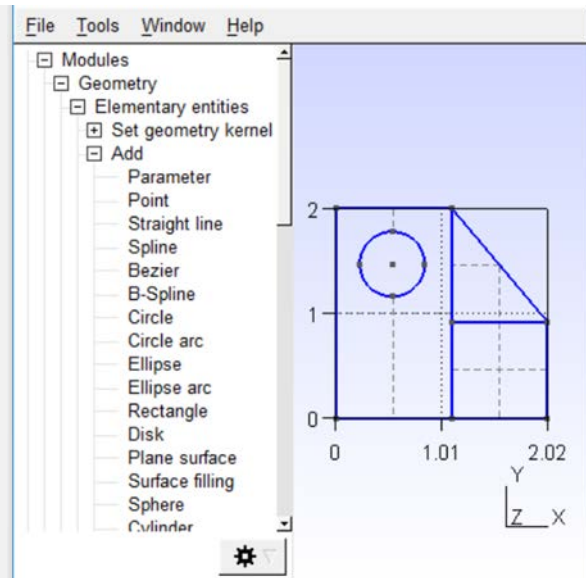


Figure 28. Gmsh added plane surface

6. Mesh the domain.

Click Modules→Mesh→1D in the program tree on the left side of the window.

No visible changes will appear in the geometry window, but Gmsh will subdivide each curve according to the prescribed mesh sizes at the points defining each curve, respectively.

Click Modules→Mesh→2D in the program tree on the left side of the window.

Gmsh will generate all non-boundary points, triangulating the entire point set within each surface according to the meshed curves and the prescribed mesh sizes at the user-defined control points, as shown in Figure 29.

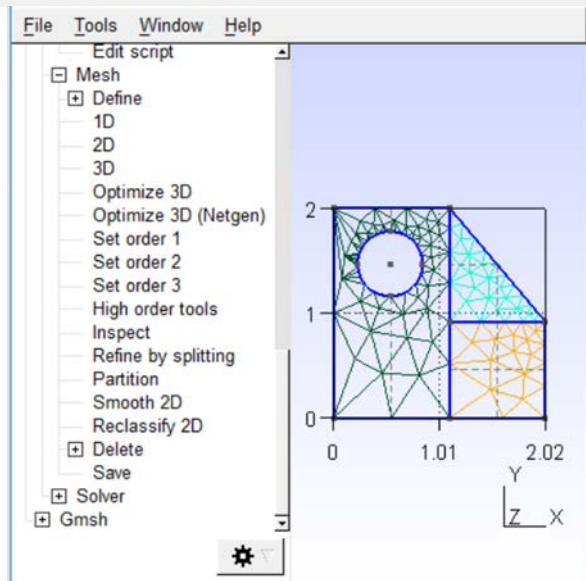


Figure 29. Gmsh meshed domain

The generated mesh will provide information for all vertices, boundary edges, and triangles used in Solver2D, indicating to which continuum each triangle belongs.

In the Gmsh script file, the language that performs this step is:

```
// Mesh domain
Mesh 2 ;
```

7. Export the mesh to a text file.

Click File→Export in the window ribbon at the top of the window.

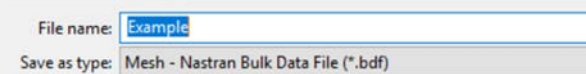


Figure 30. Gmsh mesh export file type

In the Export window, name the file “Example,” and select the file type as “Mesh – Nastran Bulk Data File (*.bdf),” as shown in Figure 30.

Click the Save button, and the BDF Options window will appear, as seen in Figure 31. Select the “Free field” format and the “Elementary entity” Element tag options. Ensure the Save all option is unchecked, then click the OK button.

Gmsh will create the mesh file in the path specified. Change the extension of the created file from “bdf” to bdfgm,” which is the accepted file name extension in Solver2D.

In the Gmsh script file, the language that performs this step is:

```
// Export mesh to file
Mesh.Format = 31 ;
Mesh.BdfFieldFormat = 0 ;
Mesh.LabelType = 1 ;
Save 'Example.bdfgm' ;
```

This completes the mesh generation purpose of Gmsh.

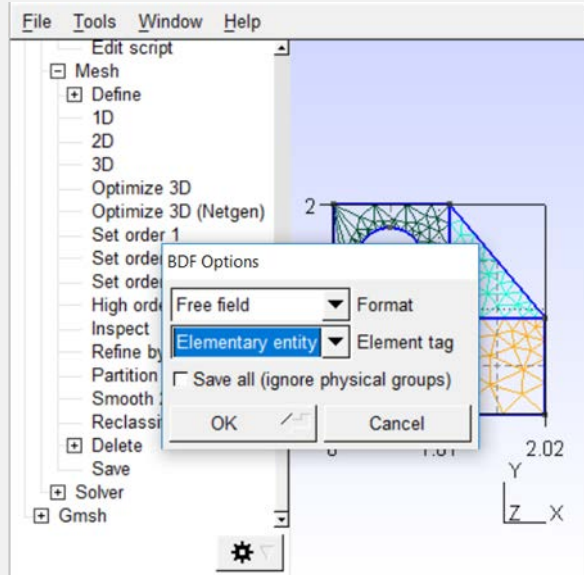


Figure 31. Gmsh NASTRAN export options

8. The full Gmsh script file for this example is:

```
//////////////////// POINTS
// Generate control points
Point(1) = {0, 0, 0, 1};
Point(2) = {1.11, 0, 0, 1};
Point(3) = {2.02, 0, 0, 0.5};
Point(4) = {2.02, 0.92, 0, 0.1};
Point(5) = {1.11, 2, 0, 0.2};
Point(6) = {0, 2, 0, 1};
Point(7) = {1.11, 0.92, 0, 0.5};
Point(8) = {0.54, 1.47, 0, 1};
Point(9) = {0.54, 1.16, 0, 0.2};
Point(10) = {0.85, 1.47, 0, 0.1};
Point(11) = {0.54, 1.78, 0, 0.05};
Point(12) = {0.23, 1.47, 0, 0.025};

//////////////////// CURVES
// Generate straight lines
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 5};
Line(5) = {5, 6};
Line(6) = {6, 1};
Line(7) = {2, 7};
Line(8) = {7, 5};
Line(9) = {7, 4};

// Generate circle arcs
Circle(10) = {9, 8, 10};
Circle(11) = {10, 8, 11};
Circle(12) = {11, 8, 12};
Circle(13) = {12, 8, 9};

// Generate plane surfaces
Line Loop(1) = {1, 7, 8, 5, 6};
Line Loop(2) = {10, 11, 12, 13};
Plane Surface(1) = {1, 2};
Line Loop(3) = {2, 3, -9, -7};
Plane Surface(2) = {3};
Line Loop(4) = {9, 4, -8};
Plane Surface(3) = {4};
```

```
// Mesh domain
Mesh 2 ;

// Export mesh to file
Mesh.Format = 31 ;
Mesh.BdfFieldFormat = 0 ;
Mesh.LabelType = 1 ;
Save 'Example.bdfgm' ;
```

4.3 Analysis

The following steps walk through finding a thermal solution using Solver2D.

1. Define user variables.

Navigate in the Solver2D source code to the *UserVariables* module. The module may be empty, as shown in Figure 32, or it may already have content.

```
module UserVariables
! This module contains all user-defined global variables.

use Types

end module UserVariables
```

Figure 32. Solver2D *UserVariables* module

The following three global variables (with declared data type and initial value) will be added to the source code:

1. *T_HOT*, double, 100
2. *T_WRM*, double, 50
3. *q_bnd*, double, 10

```
module UserVariables
! This module contains all user-defined global variables.

use Types

real(8) :: T_HOT = 100.0d0
real(8) :: T_WRM = 50.0d0
real(8) :: q_bnd = 10.0d0

end module UserVariables
```

Figure 33. Solver2D added user variables

Below any use statements in the module, enter the following lines of code:

```
real(8) :: T_HOT = 100.0d0
real(8) :: T_WRM = 50.0d0
real(8) :: q_bnd = 10.0d0
```

These added variables will be used to define some functions and boundary conditions in later steps. The module should then appear as it is shown in Figure 33.

2. Define user functions.

Navigate in the Solver2D source code to the *Functions* module. Scroll down to the bottom of the *UserFunction* function definition where the diverted function select case block ends, as shown in Figure 34. The select case block should have definitions for Solver2D occupied cases 0 through 7 and a default case, a shown here, or cases beyond 0 through 7 may exist.

The following three function cases (with description) will be added to the source code:

1. Case 8, hot temperature boundary function
2. Case 9, warm temperature boundary function
3. Case 10, non-zero heat flux boundary function

Below the last indexed case block (but before the default case block) in the function, enter the following lines of code:

```
case ( 8 )
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(1))
  UserFunction%real_ar(1) = T_HOT
case ( 9 )
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(1))
  UserFunction%real_ar(1) = T_WRM
case ( 10 )
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(3))
  UserFunction%real_ar(1) = 0.0d0
  UserFunction%real_ar(2) = q_bnd
  UserFunction%real_ar(3) = 0.0d0
```

These added function cases are used to define boundary conditions in later steps. The end of the function select case block should then appear as it is shown in Figure 35.

```
! real_ar(3) = PMS flux z-direction
UserFunction%real_ar(1) = 0.0d0
UserFunction%real_ar(2) = 0.0d0
UserFunction%real_ar(3) = 0.0d0
case ( 7 ) ! manufactured solution source term
! This function defines a user-specified manufactured solution source term
!
! THIS FUNCTION DEFINITION MUST BE MODIFIED BY THE USER

allocate(UserFunction%intr_ar(0))
allocate(UserFunction%real_ar(1))
! INPUTS
! ar_real(1) = x-coordinate
! ar_real(2) = y-coordinate
! ar_real(3) = z-coordinate
! ar_real(4) = thermal conductivity

! OUTPUTS
! real_ar(1) = PMS volumetric source value
UserFunction%real_ar(1) = 5.0d0*ar_real(4)*pi*pi*cos(2.0d0*pi*ar_real(1))*dsin(pi*a
case default ! default function value (null return)
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(0))
end select

return

end function UserFunction

end module Functions
```

Figure 34. Solver2D *Functions* module

```
! OUTPUTS
! real_ar(1) = PMS volumetric source value
UserFunction%real_ar(1) = 5.0d0*ar_real(4)*pi*pi*cos(2.0d0*pi*ar_real(1))*dsin(pi*a
case ( 8 )
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(1))
  UserFunction%real_ar(1) = T_HOT
case ( 9 )
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(1))
  UserFunction%real_ar(1) = T_WRM
case ( 10 )
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(3))
  UserFunction%real_ar(1) = 0.0d0
  UserFunction%real_ar(2) = q_bnd
  UserFunction%real_ar(3) = 0.0d0
case default ! default function value (null return)
  allocate(UserFunction%intr_ar(0))
  allocate(UserFunction%real_ar(0))
end select

return

end function UserFunction

end module Functions
```

Figure 35. Solver2D added user functions

3. Define materials.

Navigate in the Solver2D source code to the *DefineMaterials* subroutine. Scroll to the segment of the subroutine between variable declarations and array preallocations as shown in Figure 36. The subroutine should have some definition for Nmat, the number of materials defined in the code.

Change the definition of Nmat to three materials by changing the line to:

Nmat = 3

```
subroutine DefineMaterials
! This subroutine defines a database of materials with their
! associated thermophysical and thermoptical properties.

use AllModules
implicit none

! variables
integer :: i ! index

! initialize variables [1]
Nmat = 2

! preallocate arrays
allocate(mat_ID(1:Nmat))
allocate(mat_name(1:Nmat))
allocate(mat_k(1:Nmat))
allocate(mat_rho(1:Nmat))
allocate(mat_cp(1:Nmat))
allocate(mat_mu(1:Nmat))
allocate(mat_e(1:Nmat))
allocate(mat_r(1:Nmat))
allocate(mat_t(1:Nmat))
```

Figure 36. Solver2D *DefineMaterials* subroutine

This tells Solver2D to store three different materials in its database. The code should appear as it is given in Figure 37.

Scroll down to the material property assignment loop as shown in Figure 38. A select case block is used to define each of the materials with their associated properties.

The following three material definitions will be added to the source code:

Property	Material		
ID	1	2	3
mat_name	Mat_1	Mat_2	Mat_3
mat_k	20.0	30.0	10.0
mat_rho	1.0	2.0	3.0
mat_cp	10.0	20.0	30.0
mat_mu	0.1	0.2	0.3
mat_e	0.1	0.2	0.3
mat_r	0.5	0.5	0.5
mat_t	0.4	0.3	0.2

Within the select case block, enter the following lines of code:

```
case ( 1 )
  ! MATERIAL 1
  mat_name(i) = 'Mat_1'
  mat_k(i) = 20.0d0
  mat_rho(i) = 1.0d0
  mat_cp(i) = 10.0d0
  mat_mu(i) = 0.1d0
  mat_e(i) = 0.1d0
  mat_r(i) = 0.5d0
  mat_t(i) = 0.4d0
case ( 2 )
  ! MATERIAL 2
  mat_name(i) = 'Mat_2'
  mat_k(i) = 30.0d0
  mat_rho(i) = 2.0d0
  mat_cp(i) = 20.0d0
  mat_mu(i) = 0.2d0
  mat_e(i) = 0.2d0
  mat_r(i) = 0.5d0
  mat_t(i) = 0.3d0
case ( 3 )
  ! MATERIAL 3
  mat_name(i) = 'Mat_3'
  mat_k(i) = 10.0d0
  mat_rho(i) = 3.0d0
  mat_cp(i) = 30.0d0
  mat_mu(i) = 0.3d0
  mat_e(i) = 0.3d0
  mat_r(i) = 0.5d0
  mat_t(i) = 0.2d0
```

These added material definitions are used to define continuum properties in later steps. The end of the subroutine should then appear as it is shown in Figure 39.

```
subroutine DefineMaterials
! This subroutine defines a database of materials with their
! associated thermophysical and themoptical properties.

use AllModules
implicit none

! variables
integer :: i ! index

! initialize variables [1]
Nmat = 3

! preallocate arrays
allocate(mat_ID(1:Nmat))
allocate(mat_name(1:Nmat))
allocate(mat_k(1:Nmat))
allocate(mat_rho(1:Nmat))
allocate(mat_cp(1:Nmat))
allocate(mat_mu(1:Nmat))
allocate(mat_e(1:Nmat))
allocate(mat_r(1:Nmat))
allocate(mat_t(1:Nmat))
```

Figure 37. Solver2D added materials

```
! assign material properties to each material
do i = 1, Nmat ! loop through all materials
  select case ( i ) ! each case block defines a unique material

  end select
end do

end subroutine DefineMaterials
```

Figure 38. Solver2D blank materials definition

```
! assign material properties to each material
do i = 1, Nmat ! loop through all materials
  select case ( i ) ! each case block defines a unique material
    case ( 1 )
      ! MATERIAL 1
      mat_name(i) = 'Mat_1'
      mat_k(i) = 20.0d0
      mat_rho(i) = 1.0d0
      mat_cp(i) = 10.0d0
      mat_mu(i) = 0.1d0
      mat_e(i) = 0.1d0
      mat_r(i) = 0.5d0
      mat_t(i) = 0.4d0
    case ( 2 )
      ! MATERIAL 2
      mat_name(i) = 'Mat_2'
      mat_k(i) = 30.0d0
      mat_rho(i) = 2.0d0
      mat_cp(i) = 20.0d0
      mat_mu(i) = 0.2d0
      mat_e(i) = 0.2d0
      mat_r(i) = 0.5d0
      mat_t(i) = 0.3d0
    case ( 3 )
      ! MATERIAL 3
      mat_name(i) = 'Mat_3'
      mat_k(i) = 10.0d0
      mat_rho(i) = 3.0d0
      mat_cp(i) = 30.0d0
      mat_mu(i) = 0.3d0
      mat_e(i) = 0.3d0
      mat_r(i) = 0.5d0
      mat_t(i) = 0.2d0
  end select
end do

end subroutine DefineMaterials
```

Figure 39. Solver2D added materials definitions

4. Define continuums.

Materials will be assigned to continuums as described by the system definition shown in Figure 10.

Navigate to the *DefineContinuumMaterial* subroutine. The subroutine should contain a do loop with a select case block for continuum material ID assignment. The select case block may be empty, as seen in Figure 40.

The three continuums that were generated in Gmsh will be assigned materials as follows:

1. Material ID 2
2. Material ID 3
3. Material ID 1

Within the select case block, enter the following lines of code:

```
case ( 1 )
  cnt_mat(i) = 2
case ( 2 )
  cnt_mat(i) = 3
case ( 3 )
  cnt_mat(i) = 1
```

These added continuum material assignments are used to define finite element material properties. The subroutine should then appear as it is shown in Figure 41.

```
subroutine DefineContinuumMaterial
! This subroutine assigns a material to each mesh continuum.

use AllModules
implicit none

integer :: i ! index

! preallocate arrays
allocate(cnt_mat(1:Ncnt))

! initialize variables
cnt_mat = 0

! assign material ID to each continuum
do i = 1 , Ncnt ! loop through all continuums
  select case ( i ) ! each case block defines material for unique continuum
  end select
end do

end subroutine DefineContinuumMaterial
```

Figure 40. Solver2D blank continuums definition

```
subroutine DefineContinuumMaterial
! This subroutine assigns a material to each mesh continuum.

use AllModules
implicit none

integer :: i ! index

! preallocate arrays
allocate(cnt_mat(1:Ncnt))

! initialize variables
cnt_mat = 0

! assign material ID to each continuum
do i = 1 , Ncnt ! loop through all continuums
  select case ( i ) ! each case block defines material for unique continuum
    case ( 1 )
      cnt_mat(i) = 1
    case ( 2 )
      cnt_mat(i) = 2
  end select
end do

end subroutine DefineContinuumMaterial
```

Figure 41. Solver2D added continuum material assignments

5. Define boundaries.

Boundary types definitions and defined functions will be assigned to each boundary as shown in Figure 10, where 13 different boundaries were generated.

Navigate to the *AssignBoundaryFunctions* subroutine. The subroutine should contain a do loop with a select case block for boundary type definitions and boundary function assignments. The select case block may be empty, as seen in Figure 42.

The 13 boundaries that were generated by definition in Gmsh will be assigned a boundary type and function assignment pair (type,function) based on the function definitions provided in the *Functions* module that match the boundary conditions given in Figure 10 as follows:

1. 3,10
2. 3,2

```
subroutine AssignBoundaryFunctions
! This subroutine assigns both boundary condition type IDs and
! function IDs to each boundary.

use AllModules
implicit none

integer :: i ! index

! NOTE: Boundary Types
! -1 = Intimate Thermal Contact
! 1 = Temperature/Dirichlet
! 2 = Temperature Gradient/Neumann
! 3 = Heat Flux

do i = 1, Nbnid ! loop through all boundaries
  select case ( i ) ! each case block defines the boundary type ID and function ID for a unid
  end select
end do

end subroutine AssignBoundaryFunctions
```

Figure 42. Solver2D blank boundary definitions

```

3. 3,2
4. 1,8
5. 1,0
6. 3,2
7. -1,0
8. -1,0
9. -1,0
10. 1,9
11. 1,9
12. 3,2
13. 3,2

```

Within the select case block, enter the following lines of code:

```

case ( 1 )
    bnd_type(i) = 3
    bnd_func(i) = 10
case ( 2 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 3 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 4 )
    bnd_type(i) = 1
    bnd_func(i) = 8
case ( 5 )
    bnd_type(i) = 1
    bnd_func(i) = 0
case ( 6 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 7 )
    bnd_type(i) = -1
    bnd_func(i) = 0
case ( 8 )
    bnd_type(i) = -1
    bnd_func(i) = 0
case ( 9 )
    bnd_type(i) = -1
    bnd_func(i) = 0
case ( 10 )
    bnd_type(i) = 1
    bnd_func(i) = 9
case ( 11 )
    bnd_type(i) = 1
    bnd_func(i) = 9
case ( 12 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 13 )
    bnd_type(i) = 3
    bnd_func(i) = 2

```

```

select case ( i ) ! each case block defines the boundary type ID and function ID for a ur
case ( 1 )
    bnd_type(i) = 3
    bnd_func(i) = 10
case ( 2 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 3 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 4 )
    bnd_type(i) = 1
    bnd_func(i) = 8
case ( 5 )
    bnd_type(i) = 1
    bnd_func(i) = 0
case ( 6 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 7 )
    bnd_type(i) = -1
    bnd_func(i) = 0
case ( 8 )
    bnd_type(i) = -1
    bnd_func(i) = 0
case ( 9 )
    bnd_type(i) = -1
    bnd_func(i) = 0
case ( 10 )
    bnd_type(i) = 1
    bnd_func(i) = 9
case ( 11 )
    bnd_type(i) = 1
    bnd_func(i) = 9
case ( 12 )
    bnd_type(i) = 3
    bnd_func(i) = 2
case ( 13 )
    bnd_type(i) = 3
    bnd_func(i) = 2
end select

```

Figure 43. Solver2D added boundary definitions

These added boundary definitions are used to constrain the behavior of the thermal solution. The subroutine should then appear as it is shown in Figure 43.

14. Define surface thickness.

Navigate to the *DefineSurfaceThickness* subroutine. This routine nominally loops through all triangles in the mesh and assigns each triangle a thickness, *tri_thk*. Make sure the subroutine assigns a thickness of 1.0 to all triangles, as shown in Figure 44.

```
subroutine DefineSurfaceThickness
! This subroutine defines the thickness of each surface element
! in the mesh.

use AllModules
implicit none

integer :: i ! index

do i = 1, Ntri ! loop through all triangles
  tri_thk(i) = 1.0d0
end do

end subroutine DefineSurfaceThickness
```

Figure 44. Solver2D defined surface thickness

15. Define analysis case settings.

Navigate to the *AnalysisDefinitions* subroutine.

This routine can be used to set parameters that define the full analysis set and that are constant between analysis cases. This example will only run one analysis case, so set *Ncas* to 1, as shown in Figure 45.

```
subroutine AnalysisDefinitions
! This subroutine defines the full analysis set parameters.
!
! THIS SUBROUTINE MUST BE MODIFIED BY THE USER

use AllModules
implicit none

Ncas = 1 ! number of analysis cases

end subroutine AnalysisDefinitions
```

Figure 45. Solver2D defined analysis set parameters

Next, navigate to the *SettingsDefinitions* subroutine.

Here, analysis case parameters and settings can be defined for individual analysis cases. The following settings will be defined:

- Method of Manufactured Solutions operation: False
- Output format for type real values: 16 digits of precision
- Maximum number of solver iterations: 1,000,000
- Residual cutoff limit: 0.00000001
- Mesh file name: Example.bdfgm
- Output format for scientific values: Field width of 13, 6 digits of precision
- Solver relaxation coefficient: 1.0
- Flag to display mesh file lines to user: False

```
subroutine SettingsDefinitions
! This subroutine defines settings for the current analysis case.
!
! THIS SUBROUTINE MUST BE MODIFIED BY THE USER

use AllModules
implicit none

FLAG_MMS = 0 ! MMS
fmtf = "f0.16" ! format specification for real values
i_solve_MAX = 10000000 ! maximum solver iterations
res_lim = 0.00000001 ! residual cutoff limit
if ( i_cas == 1 ) then
  MeshInputFileName = 'Example.bdfgm'
end if
fmtf = "e13.6" ! format specification for scientific values
relax = 1.0d0 ! temperature update relaxation coefficient
FLAG_disp_inpt = 1 ! flag to display input file lines

end subroutine SettingsDefinitions
```

Figure 46. Solver2D defined analysis case parameters

In order to specify these parameters, type the following code into the subroutine:

```
FLAG_MMS = 0 ! MMS
fmtf = "f0.16" ! format specification for
real values
i_solve_MAX = 10000000 ! maximum solver
iterations
res_lim = 0.00000001 ! residual cutoff
limit
if ( i_cas == 1 ) then
  MeshInputFileName = 'Example.bdfgm'
end if
fmtf = "e13.6" ! format specification for
scientific values
relax = 1.0d0 ! temperature update
relaxation coefficient
FLAG_disp_inpt = 0 ! flag to display input
file lines
```

The subroutine should then look as seen in Figure 46.

16. Compile and run Solver2D.

Solver2D was developed using Intel® Fortran, thus this process walks through commands for Intel® Fortran.

Open a command line window.

Navigate to the location of the Solver2D source code.

Enter the following command:
`ifort Solver2D.f90 -traceback -check`

After a successful compile, enter the command:
`Solver2D.exe`

Solver2D will output information to the screen similar to what is shown in Figure 47.

A folder labeled as “Solution_01” will be created in the same directory as Solver2D.exe and will contain multiple output files which contain information about the mesh and the simulation. These files can be used later for visualization of the system.

```
importing mesh
Mesh Input File I/O Error Status Flag:          0
Characteristic Mesh Sizes:
  1D: .0000000000000000
  2D: .1315103122746646
  3D: .0000000000000000
Overall: .1315103122746646
It.:   100 | Res.: 0.766805E+01 | Mfg. Error: 0.327710E+03
It.:   200 | Res.: 0.763543E+00 | Mfg. Error: 0.329010E+03
It.:   300 | Res.: 0.748352E-01 | Mfg. Error: 0.329128E+03
It.:   400 | Res.: 0.732455E-02 | Mfg. Error: 0.329140E+03
It.:   500 | Res.: 0.716825E-03 | Mfg. Error: 0.329141E+03
It.:   600 | Res.: 0.701524E-04 | Mfg. Error: 0.329141E+03
It.:   700 | Res.: 0.686550E-05 | Mfg. Error: 0.329141E+03
It.:   800 | Res.: 0.671895E-06 | Mfg. Error: 0.329141E+03
It.:   900 | Res.: 0.657554E-07 | Mfg. Error: 0.329141E+03
It.:  982 | Res.: 0.977783E-08 | Mfg. Error: 0.329141E+03
```

Figure 47. Solver2D output

4.4 Visualization

The following steps walk through interrogating a thermal solution using SolutionPlot.

1. Define analysis set.

In SolutionPlot, set N_CASES to 1 (matching the total number of cases run in an analysis set in Solver2D).

Set both i_CASE_START and i_CASE_STOP to 1 to analyze case 1 through case 1. (If N_CASES were greater than 1, i_CASE_START and i_CASE_STOP could be chosen to be any range bounded by 1 and N_CASES.)

Although many options are available for plotting in SolutionPlot, turn on these options:

- Plot
- Plot mesh
- Plot solution
- Solution type: temperature

Choosing these options will tell SolutionPlot to

- Generate a plot
- Plot the FE mesh
- Plot some distribution on the mesh
- Plot temperature as the distribution variable

2. Run the interrogation.

In MATLAB®, run the SolutionPlot script. SolutionPlot will generate a figure according to the options chosen, as seen in Figure 48.

Explore different settings in SolutionPlot or make changes to the script to achieve desired interrogations and visualization.

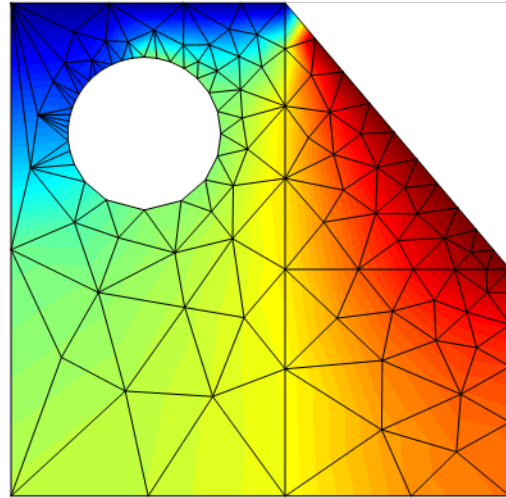


Figure 48. SolutionPlot temperature contour

References

- 1 Geuzaine, C. and Remacle, J.-F., 2009, "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering* 79(11), pp. 1309-1331.

APPENDIX F: AWARDS, HONORS, AND PUBLICATIONS

Following is a list of academic awards and honors bestowed upon Kevin Irick during his tenure in the Ph.D. program:

1. Graduate Success Scholarship, Nominee – Fall 2020
2. Carter Endowed Scholarship – Fall 2019/Spring 2020
3. Charlotte and William Kraft Graduate Fellowship – Spring 2020
4. Graduate Success Scholarship – Fall 2018/Spring 2019
5. Arthur J. Harvey Foundation Endowed Fellowship – Fall 2018

Publications as of November 2020:

1. Irick, Kevin, and Nima Fathi. "Computational Evaluation of Thermal Response of Open-cell Foam With Circular Pore." *Journal of Verification, Validation and Uncertainty Quantification*, accepted, 2020.
2. Irick, Kevin, and Nima Fathi. "Evaluation of Pore Geometry Effects on Porous Cell Thermal Behavior." In *Verification and Validation*, vol. 83594, p. V001T08A002. American Society of Mechanical Engineers, 2020.
3. Irick, Kevin, and Nima Fathi. "High-Fidelity Calculation of Effective Thermal Response of Composite Media With Heat Generation Source." In *Verification and Validation*, vol. 83594, p. V001T08A003. American Society of Mechanical Engineers, 2020.
4. Irick, Kevin, and Nima Fathi. "Computational Evaluation of Thermal Barrier Coatings: Two-Phase Thermal Transport Analysis." In *Verification and Validation*, vol. 41174, p. V001T12A002. American Society of Mechanical Engineers, 2019.
5. Irick, Kevin, and Nima Fathi. "Thermal Response of Open-Cell Porous Materials: A Numerical Study and Model Assessment." In *Verification and Validation*, vol. 40795, p. V001T03A002. American Society of Mechanical Engineers, 2018.