

# SANDIA REPORT

SAND2020-3999 - Unlimited Release

Printed April 9, 2020



Sandia  
National  
Laboratories

## Sierra/Aria 4.56 Verification Manual

Brian R. Carnes

Prepared by  
Sandia National Laboratories  
Albuquerque, New Mexico 87185  
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Telephone: (865) 576-8401  
Facsimile: (865) 576-5728  
E-Mail: reports@osti.gov  
Online ordering: <http://www.osti.gov/scitech>

Available to the public from

U.S. Department of Commerce  
National Technical Information Service  
5301 Shawnee Road  
Alexandria, VA 22312

Telephone: (800) 553-6847  
Facsimile: (703) 605-6900  
E-Mail: orders@ntis.gov  
Online order: <https://classic.ntis.gov/help/order-methods>



## **ABSTRACT**

Presented in this document is a portion of the tests that exist in the Sierra Thermal/Fluids verification test suite. Each of these tests is run nightly with the Sierra/TF code suite and the results of the test checked under mesh refinement against the correct analytic result. For each of the tests presented in this document the test setup, derivation of the analytic solution, and comparison of the code results to the analytic solution is provided. This document can be used to confirm that a given code capability is verified or referenced as a compilation of example problems.

# ACKNOWLEDGMENTS

This document's authors acknowledge the help of the Sierra/TF team in creating and maintaining these verification tests.

# CONTENTS

<b>Contents</b>	<b>5</b>
<b>List of Figures</b>	<b>21</b>
<b>List of Tables</b>	<b>23</b>
<b>1. Introduction</b>	<b>27</b>
<b>2. Basic Thermal Tests</b>	<b>28</b>
2.1. Steady Heat Conduction: Hex8 Meshes	28
2.1.1. Features Tested	28
2.1.2. Boundary Conditions	28
2.1.3. Material Parameters	28
2.1.4. Verification of Solution	28
2.2. Steady Heat Conduction: Hex20 Meshes	29
2.2.1. Features Tested	29
2.2.2. Boundary Conditions	30
2.2.3. Material Parameters	30
2.2.4. Verification of Solution	30
2.3. Steady Heat Conduction: Hex27 Meshes	30
2.3.1. Features Tested	31
2.3.2. Boundary Conditions	31
2.3.3. Material Parameters	31
2.3.4. Verification of Solution	32
2.4. Steady Heat Conduction: Tet4 Meshes	33
2.4.1. Features Tested	33

2.4.2.	Boundary Conditions .....	33
2.4.3.	Material Parameters .....	33
2.4.4.	Verification of Solution .....	33
2.5.	Steady Heat Conduction: Tet4Tet10 Meshes .....	34
2.5.1.	Features Tested .....	34
2.5.2.	Boundary Conditions .....	34
2.5.3.	Material Parameters .....	34
2.5.4.	Verification of Solution .....	35
2.6.	Steady Heat Conduction: Tet10 Meshes .....	35
2.6.1.	Features Tested .....	36
2.6.2.	Boundary Conditions .....	36
2.6.3.	Material Parameters .....	36
2.6.4.	Verification of Solution .....	36
2.7.	Transient Heat Conduction: Hex8 Meshes .....	37
2.7.1.	Features Tested .....	37
2.7.2.	Boundary Conditions .....	37
2.7.3.	Material Parameters .....	37
2.7.4.	Verification of Solution .....	37
2.8.	Transient Heat Conduction: Tet4 Meshes .....	38
2.8.1.	Features Tested .....	38
2.8.2.	Boundary Conditions .....	39
2.8.3.	Material Parameters .....	39
2.8.4.	Verification of Solution .....	39
2.9.	Transient Heat Conduction: Tet4Tet10 Meshes .....	40
2.9.1.	Features Tested .....	40
2.9.2.	Boundary Conditions .....	40
2.9.3.	Material Parameters .....	40
2.9.4.	Verification of Solution .....	40
2.10.	Transient Heat Conduction: Tet10 Meshes .....	41
2.10.1.	Features Tested .....	41

2.10.2.	Boundary Conditions .....	41
2.10.3.	Material Parameters .....	41
2.10.4.	Verification of Solution .....	42
2.11.	PostProcess Min/Max .....	42
2.11.1.	Problem Description .....	42
2.11.2.	Features Tested .....	43
2.11.3.	Boundary Conditions .....	43
2.11.4.	Material Parameters .....	43
2.11.5.	Verification of Solution .....	43
2.12.	Adaptivity .....	43
2.12.1.	Features Tested .....	44
2.12.2.	Boundary Conditions .....	44
2.12.3.	Material Parameters .....	44
2.12.4.	Verification of Solution .....	44

### **3. Thermal Boundary Conditions 46**

3.1.	Radiative Heat Flux .....	46
3.1.1.	Features Tested .....	46
3.1.2.	Boundary Conditions .....	46
3.1.3.	Material Parameters .....	46
3.1.4.	Verification of Solution .....	46
3.2.	Radiative Heat Flux From Fortran User Subroutine .....	47
3.2.1.	Features Tested .....	47
3.2.2.	Boundary Conditions .....	47
3.2.3.	Material Parameters .....	48
3.2.4.	Verification of Solution .....	48
3.3.	Convective Heat Flux .....	48
3.3.1.	Features Tested .....	48
3.3.2.	Boundary Conditions .....	48
3.3.3.	Material Parameters .....	48

3.3.4.	Verification of Solution .....	49
3.4.	Thermal Convective Flux (Fortran sub-routine) .....	50
3.4.1.	Problem Description .....	50
3.4.2.	Features Tested .....	50
3.4.3.	Boundary Conditions .....	50
3.4.4.	Material Parameters .....	50
3.4.5.	Verification of Solution .....	50
3.5.	Thermal Convective Flux (User field from Exodus read-in) .....	51
3.5.1.	Problem Description .....	51
3.5.2.	Features Tested .....	51
3.5.3.	Boundary Conditions .....	51
3.5.4.	Material Parameters .....	52
3.5.5.	Verification of Solution .....	52
3.6.	Thermal Heat Flux .....	53
3.6.1.	Thermal Heat Flux (Basic) .....	53
3.6.1.1.	Problem Description .....	53
3.6.1.2.	Features Tested .....	53
3.6.1.3.	Boundary Conditions .....	53
3.6.1.4.	Material Parameters .....	53
3.6.1.5.	Verification of Solution .....	53
3.6.2.	Thermal Heat Flux (Flux node variable user field) .....	54
3.6.2.1.	Problem Description .....	54
3.6.2.2.	Features Tested .....	54
3.6.2.3.	Boundary Conditions .....	54
3.6.2.4.	Material Parameters .....	55
3.6.2.5.	Verification of Solution .....	55
3.6.3.	Thermal Heat Flux (Flux node variable user field) .....	55
3.6.3.1.	Problem Description .....	55
3.6.3.2.	Features Tested .....	56
3.6.3.3.	Boundary Conditions .....	56

3.6.3.4.	Material Parameters .....	56
3.6.3.5.	Verification of Solution .....	56
3.6.4.	Thermal Heat Flux (Fortran Subroutine).....	57
3.6.4.1.	Problem Description .....	57
3.6.4.2.	Features Tested .....	57
3.6.4.3.	Boundary Conditions.....	57
3.6.4.4.	Material Parameters .....	57
3.6.4.5.	Verification of Solution .....	58
3.7.	Thermal Radiative Heat Flux .....	59
3.7.1.	Basic Calore-Style BC .....	59
3.7.1.1.	Problem Description .....	59
3.7.1.2.	Features Tested .....	59
3.7.1.3.	Boundary Conditions.....	59
3.7.1.4.	Material Parameters .....	59
3.7.1.5.	Verification of Solution .....	59
3.7.2.	With Fortran Subroutines .....	60
3.7.2.1.	Problem Description .....	60
3.7.2.2.	Features Tested .....	60
3.7.2.3.	Boundary Conditions.....	60
3.7.2.4.	Material Parameters .....	60
3.7.2.5.	Verification of Solution .....	61
3.7.3.	With User Subroutines .....	61
3.7.3.1.	Problem Description .....	61
3.7.3.2.	Features Tested .....	62
3.7.3.3.	Boundary Conditions.....	62
3.7.3.4.	Material Parameters .....	62
3.7.3.5.	Verification of Solution .....	62
3.8.	Advective Bar .....	62
3.8.1.	Steady Advection-Diffusion .....	62
3.8.2.	Features Tested .....	63

3.8.3.	Boundary Conditions .....	63
3.8.4.	Material Parameters .....	63
3.8.5.	Verification of Solution .....	63
3.8.6.	Transient Advection-Diffusion .....	64
3.8.7.	Features Tested .....	64
3.8.8.	Boundary Conditions .....	65
3.8.9.	Material Parameters .....	65
3.8.10.	Verification of Solution .....	65
3.8.11.	Transient Advection-Diffusion in 2D .....	65
3.8.12.	Features Tested .....	66
3.8.13.	Boundary Conditions .....	66
3.8.14.	Material Parameters .....	66
3.8.15.	Verification of Solution .....	66
3.9.	Solution Verification .....	67
3.9.1.	Features Tested .....	67
3.9.2.	Material Parameters .....	67
3.9.3.	Verification of Solution .....	67

## **4. Thermal Contact 70**

4.1.	1D Flat Contact .....	70
4.1.1.	Features Tested .....	70
4.1.2.	Boundary Conditions .....	70
4.1.3.	Material Parameters .....	70
4.1.4.	Verification of Solution .....	71
4.1.5.	Results: Hex8 Tied .....	71
4.1.6.	Results: Hex8 Resistance .....	71
4.1.7.	Results: Tet4 Tied .....	73
4.1.8.	Results: Tet4 Resistance .....	73
4.1.9.	Results: Hex8-Tet4 Tied .....	75
4.1.10.	Results: Hex8-Tet4 Resistance .....	75

4.2.	3D Curved Contact .....	75
4.2.1.	Features Tested .....	76
4.2.2.	Boundary Conditions .....	76
4.2.3.	Material Parameters .....	76
4.2.4.	Verification of Solution .....	77
4.2.5.	Results: Hex8-Hex8 Contact .....	77
4.2.6.	Results: Tet4-Tet4 Contact .....	79
4.2.7.	Results: Hex8-Tet4 Contact .....	80
4.3.	Steady Hex8 Contact .....	80
4.3.1.	Features Tested .....	80
4.3.2.	Boundary Conditions .....	81
4.3.3.	Material Parameters .....	81
4.3.4.	Verification of Solution .....	81
4.4.	Steady Hex20 Contact .....	81
4.4.1.	Features Tested .....	82
4.4.2.	Boundary Conditions .....	82
4.4.3.	Material Parameters .....	82
4.4.4.	Verification of Solution .....	83
4.5.	Steady Hex27 Contact .....	84
4.5.1.	Features Tested .....	84
4.5.2.	Boundary Conditions .....	84
4.5.3.	Material Parameters .....	84
4.5.4.	Verification of Solution .....	84
4.6.	Steady Tet4 Contact .....	85
4.6.1.	Features Tested .....	85
4.6.2.	Boundary Conditions .....	85
4.6.3.	Material Parameters .....	86
4.6.4.	Verification of Solution .....	86
4.7.	Steady Tet4Tet10 Contact .....	86
4.7.1.	Features Tested .....	87

4.7.2.	Boundary Conditions .....	87
4.7.3.	Material Parameters .....	87
4.7.4.	Verification of Solution .....	87
4.8.	Steady Tet10 Contact .....	88
4.8.1.	Features Tested .....	88
4.8.2.	Boundary Conditions .....	88
4.8.3.	Material Parameters .....	88
4.8.4.	Verification of Solution .....	88
4.9.	Steady Tet10 Dash Contact .....	89
4.9.1.	Features Tested .....	89
4.9.2.	Boundary Conditions .....	89
4.9.3.	Material Parameters .....	89
4.9.4.	Verification of Solution .....	90
4.10.	Transient Tet4Tet10 Contact .....	90
4.10.1.	Features Tested .....	90
4.10.2.	Boundary Conditions .....	91
4.10.3.	Material Parameters .....	91
4.10.4.	Verification of Solution .....	91
4.11.	Transient Tet10 Contact .....	92
4.11.1.	Features Tested .....	92
4.11.2.	Boundary Conditions .....	92
4.11.3.	Material Parameters .....	92
4.11.4.	Verification of Solution .....	92
4.12.	Transient Hex8 Tied Contact .....	93
4.12.1.	Features Tested .....	93
4.12.2.	Boundary Conditions .....	93
4.12.3.	Material Parameters .....	94
4.12.4.	Verification of Solution .....	94
4.13.	Transient Tet4 Tied Contact .....	95
4.13.1.	Features Tested .....	95

4.13.2.	Boundary Conditions .....	95
4.13.3.	Material Parameters .....	95
4.13.4.	Verification of Solution .....	95
<b>5.</b>	<b>Element Death</b>	<b>97</b>
5.1.	CDFEM Element Death (Heat Flux) .....	97
5.1.1.	Features Tested .....	97
5.1.2.	Boundary Conditions .....	97
5.1.3.	Material Parameters .....	97
5.1.4.	Verification of Solution .....	97
5.1.5.	Results: Tri3 .....	98
5.1.6.	Results: Tet4 .....	98
5.2.	3D Spherical Shell Enclosure .....	99
5.2.1.	Problem Description .....	99
5.2.2.	Features Tested .....	99
5.2.3.	Boundary and Initial Conditions .....	99
5.2.4.	Material Parameters .....	100
5.2.5.	Verification of Solution .....	100
5.2.6.	Results .....	101
5.3.	Standard Element Death (Heat Flux) .....	102
5.3.1.	Features Tested .....	102
5.3.2.	Boundary Conditions .....	102
5.3.3.	Material Parameters .....	103
5.3.4.	Verification of Solution .....	103
5.3.5.	Results: 1D Hex8 .....	103
5.3.6.	Results: 1D Quad4 .....	104
5.3.7.	Results: 1D Tri3 .....	104
5.3.8.	Results: 2D Quad4 .....	104
5.3.9.	Features Tested .....	105
5.3.10.	Boundary Conditions .....	105

5.3.11.	Material Parameters .....	105
5.3.12.	Verification of Solution .....	105
5.3.13.	Results: 3D Hex8 .....	106
5.3.14.	Features Tested .....	106
5.3.15.	Boundary Conditions .....	107
5.3.16.	Material Parameters .....	107
5.3.17.	Verification of Solution .....	107

## **6. Time Integration 109**

6.1.	Adaptive Time Integration .....	109
6.1.1.	Features Tested .....	109
6.1.2.	Boundary Conditions .....	109
6.1.3.	Material Parameters .....	109
6.1.4.	Verification of Solution .....	109
6.1.5.	Results: First Order Fixed .....	110
6.1.6.	Results: First Order Adaptive .....	110
6.1.7.	Results: Second Order Fixed .....	112
6.1.8.	Results: Second Order Adaptive .....	112
6.1.9.	Results: BDF2 Fixed .....	114
6.1.10.	Results: BDF2 Adaptive .....	114

## **7. Enclosure Radiation 116**

7.1.	2D Cylindrical Shell Enclosure .....	116
7.1.1.	Problem Description .....	116
7.1.2.	Features Tested .....	116
7.1.3.	Boundary Conditions .....	116
7.1.4.	Material Parameters .....	116
7.1.5.	Verification of Solution .....	117
7.1.6.	Results .....	117
7.2.	2D Annular Enclosure .....	118
7.2.1.	Problem Description .....	118

7.2.2.	Features Tested .....	118
7.2.3.	Boundary Conditions .....	118
7.2.4.	Material Parameters .....	119
7.2.5.	Verification of Solution .....	119
7.3.	3D Spherical Shell Enclosure .....	120
7.3.1.	Problem Description .....	120
7.3.2.	Features Tested .....	120
7.3.3.	Boundary Conditions .....	121
7.3.4.	Material Parameters .....	121
7.3.5.	Verification of Solution .....	121
7.3.6.	Results .....	123
7.4.	3D Spherical Shell Partial Enclosure .....	124
7.4.1.	Problem Description .....	124
7.4.2.	Features Tested .....	124
7.4.3.	Boundary Conditions .....	124
7.4.4.	Material Parameters .....	124
7.4.5.	Verification of Solution .....	125

## **8. Chemistry 126**

8.1.	First Order Reaction (Spatially Varying Temperature) .....	126
8.1.1.	Features Tested .....	126
8.1.2.	Boundary Conditions .....	126
8.1.3.	Material Parameters .....	126
8.1.4.	Verification of Solution .....	126
8.2.	First Order Reaction .....	127
8.2.1.	Features Tested .....	127
8.2.2.	Boundary Conditions .....	128
8.2.3.	Material Parameters .....	128
8.2.4.	Verification of Solution .....	128

8.3.	DAE and Pressure Test .....	128
8.3.1.	Features Tested .....	129
8.3.2.	Boundary Conditions .....	129
8.3.3.	Material Parameters .....	129
8.3.4.	Verification of Solution .....	130
8.4.	PMDI Plugin Test .....	130
8.4.1.	Features Tested .....	130
8.4.2.	Boundary Conditions .....	130
8.4.3.	Material Parameters .....	130
8.4.4.	Verification of Solution .....	130

**9. Miscellaneous 132**

9.1.	Thermal Postprocessing .....	132
9.1.1.	Problem Description .....	132
9.1.2.	Features Tested .....	132
9.1.3.	Boundary Conditions .....	132
9.1.4.	Material Parameters .....	132
9.1.5.	Verification of Solution .....	132
9.2.	Local Coordinates: Cartesian .....	133
9.2.1.	Features Tested .....	134
9.2.2.	Boundary Conditions .....	134
9.2.3.	Material Parameters .....	134
9.2.4.	Verification of Solution .....	134
9.3.	Local Coordinates: Cylindrical .....	134
9.3.1.	Features Tested .....	135
9.3.2.	Boundary Conditions .....	135
9.3.3.	Material Parameters .....	135
9.3.4.	Verification of Solution .....	135

**10. Low-Mach Fluid Flow 137**

**11. How to Build this Document** **138**

**12. Input Decks For Verification Problems** **140**

- 12.1. Basic Thermal Tests ..... 140
  - 12.1.1. Steady Heat Conduction: Hex8 Meshes ..... 140
  - 12.1.2. Steady Heat Conduction: Hex20 Meshes ..... 143
  - 12.1.3. Steady Heat Conduction: Hex27 Meshes ..... 146
  - 12.1.4. Steady Heat Conduction: Tet4 Meshes ..... 149
  - 12.1.5. Steady Heat Conduction: Tet4Tet10 Meshes ..... 152
  - 12.1.6. Steady Heat Conduction: Tet10 Meshes ..... 155
  - 12.1.7. Transient Heat Conduction: Hex8 Meshes ..... 158
  - 12.1.8. Transient Heat Conduction: Tet4 Meshes ..... 161
  - 12.1.9. Transient Heat Conduction: Tet4Tet10 Meshes ..... 165
  - 12.1.10. Transient Heat Conduction: Tet10 Meshes ..... 168
- 12.2. Thermal Boundary Conditions ..... 172
  - 12.2.1. Radiative Heat Flux 3.1 ..... 172
  - 12.2.2. Radiative Heat Flux From Fortran User Subroutine ..... 174
  - 12.2.3. Convective Heat Flux 3.3 ..... 179
- 12.3. Thermal Contact ..... 182
  - 12.3.1. 1D Flat Contact 4.1 ..... 182
    - 12.3.1.1. Hex8 Tied ..... 182
    - 12.3.1.2. Hex8 Resistance ..... 184
    - 12.3.1.3. Tet4 Tied ..... 187
    - 12.3.1.4. Tet4 Resistance ..... 190
    - 12.3.1.5. Hex8-Tet4 Tied ..... 192
    - 12.3.1.6. Hex8-Tet4 Resistance ..... 195
  - 12.3.2. 3D Curved Contact 4.2 ..... 198
    - 12.3.2.1. Hex8-Hex8 Case ..... 198
    - 12.3.2.2. Tet4-Tet4 Case ..... 198
    - 12.3.2.3. Hex8-Tet4 Case ..... 198

12.3.3.	Steady Hex8 Contact .....	198
12.3.4.	Steady Hex20 Contact .....	201
12.3.5.	Steady Hex27 Contact .....	205
12.3.6.	Steady Tet4 Contact .....	208
12.3.7.	Steady Tet4Tet10 Contact .....	211
12.3.8.	Steady Tet10 Contact .....	214
12.3.9.	Steady Tet10 Dash Contact .....	217
12.3.10.	Transient Tet4Tet10 Contact .....	221
12.3.11.	Transient Tet10 Contact .....	224
12.4.	Element Death .....	228
12.4.1.	CDFEM Element Death (Heat Flux) .....	228
12.4.1.1.	Tri3 .....	228
12.4.1.2.	Tet4 .....	231
12.4.2.	3D Spherical Shell Enclosure .....	233
12.5.	Time Integration .....	239
12.5.1.	Adaptive Time Integration .....	239
12.5.1.1.	First Order Fixed .....	239
12.5.1.2.	First Order Adaptive .....	241
12.5.1.3.	Second Order Fixed .....	243
12.5.1.4.	Second Order Adaptive .....	245
12.5.1.5.	BDF2 Fixed .....	247
12.5.1.6.	BDF2 Adaptive .....	249
12.6.	Enclosure Radiation .....	252
12.6.1.	2D Cylindrical Shell Enclosure .....	252
12.6.2.	2D Annular Enclosure .....	254
12.6.3.	3D Spherical Shell Enclosure .....	258
12.6.4.	3D Spherical Shell Partial Enclosure .....	258
12.6.5.	Fully 2D Enclosure Radiation .....	261
12.7.	Chemistry .....	265
12.7.1.	First Order Reaction (Uniform Temperature) .....	265

12.7.2.	First Order Reaction (Spatially Varying Temperature) .....	265
12.7.3.	First Order Reaction .....	267
12.7.4.	DAE and Pressure Test .....	270
12.7.5.	PMDI Plugin Test .....	273
12.8.	Miscellaneous .....	277
12.8.1.	Thermal Postprocessing .....	277
12.8.2.	Postprocess Min/Max .....	282
12.8.3.	Local Coordinates: Cartesian .....	285
12.8.4.	Local Coordinates: Cylindrical .....	287



# LIST OF FIGURES

- 2.1-1. Steady Heat Conduction: Hex8 Meshes ..... 29
- 2.2-1. Steady Heat Conduction: Hex20 Meshes ..... 31
- 2.3-1. Steady Heat Conduction: Hex27 Meshes ..... 32
- 2.4-1. Steady Heat Conduction: Tet4 Meshes ..... 34
- 2.5-1. Steady Heat Conduction: Tet4 Solutions on Tet10 Meshes ..... 35
- 2.6-1. Steady Heat Conduction: Tet10 Meshes ..... 36
- 2.7-1. Transient Heat Conduction: Hex8 Meshes ..... 38
- 2.8-1. Transient Heat Conduction: Tet4 Meshes ..... 39
- 2.9-1. Transient Heat Conduction: Tet4 Solution on Tet10 Meshes ..... 41
- 2.10-1. Transient Heat Conduction: Tet10 Meshes ..... 42
- 2.11-1. Min Max Postprocess ..... 44
- 2.12-1. Steady Heat Conduction: Tet4 Meshes (Adaptive Mesh Refinement) ..... 45
  
- 3.1-1. Radiative Heat Flux ..... 47
- 3.3-1. Convective Heat Flux ..... 49
- 3.4-1. Convergence for 3D thermal steady convective flux BCs. .... 51
- 3.5-1. Convergence for 3D thermal steady convective flux BCs. .... 52
- 3.6-1. Thermal Heat Flux BC ..... 54
- 3.6-2. Thermal Heat Flux BC ..... 55
- 3.6-3. Thermal Heat Flux BC ..... 57
- 3.6-4. Thermal Heat Flux BC ..... 58
- 3.7-1. Thermal Radiative Flux ..... 60
- 3.7-2. Thermal Radiative Flux ..... 61
- 3.7-3. Thermal Radiative Flux ..... 63
- 3.8-1. Steady Advective Conduction: 3D Bar2 Meshes ..... 64

3.8-2. Transient Heat Conduction: 3D Bar2 Meshes .....	65
3.8-3. Transient Heat Conduction: Bar2 Meshes .....	67
3.9-1. Mock AFF Solution Verification .....	68
3.9-2. The convergence rates can vary over time and between QOIs .....	69
4.1-1. 1D Flat Contact: Hex8 Tied .....	71
4.1-2. 1D Flat Contact: Hex8 Resistance .....	72
4.1-3. 1D Flat Contact: Tet4 Tied .....	73
4.1-4. 1D Flat Contact: Tet4 Resistance .....	74
4.1-5. 1D Flat Contact: Hex8-Tet4 Tied .....	75
4.1-6. 1D Flat Contact: Hex8-Tet4 Resistance .....	76
4.2-1. 3D Curved Contact: Hex8-Hex8 Case .....	77
4.2-2. 3D Curved Contact: Tet4-Tet4 Case .....	79
4.2-3. 3D Curved Contact: Hex8-Tet4 Case .....	80
4.3-1. Steady Tied Contact: Hex8 Meshes .....	82
4.4-1. Steady Heat Conduction: Hex20 Meshes .....	83
4.5-1. Steady Heat Conduction: Hex27 Meshes .....	85
4.6-1. Steady Tied Contact: Tet4 Meshes .....	86
4.7-1. Steady Tied Contact: Tet4 Meshes .....	87
4.8-1. Steady Tied Contact: Tet10 Meshes .....	89
4.9-1. Steady Tied Dash Contact: Tet10 Meshes .....	90
4.10-1. Transient Tied Contact: Tet10 Meshes .....	91
4.11-1. Transient Tied Contact: Tet10 Meshes .....	93
4.12-1. Tied Contact Transient Heat Conduction: Hex8 Meshes .....	94
4.13-1. Transient Heat Conduction with Tied Contact: Tet4 Meshes .....	96
5.1-1. CDFEM Element Death (Heat Flux): Tri3 .....	98
5.1-2. CDFEM Element Death (Heat Flux): Tet4 .....	99
5.2-1. Evolution of parameters $r_2$ and $C_o$ .....	102
5.3-1. Element Death (Heat Flux): Hex8 .....	104
5.3-2. Element Death (Heat Flux): Quad4 .....	105

5.3-3. Element Death (Heat Flux): Tri3 .....	106
5.3-4. Element Death (Heat Flux): Quad4 .....	107
5.3-5. Element Death (Heat Flux): Hex8 .....	108
6.1-1. Adaptive Time Integration: Errors for First Order Fixed .....	110
6.1-2. Adaptive Time Integration: Errors for First Order Adaptive .....	111
6.1-3. Adaptive Time Integration: Errors for Second Order Fixed .....	112
6.1-4. Adaptive Time Integration: Errors for Second Order Adaptive .....	113
6.1-5. Adaptive Time Integration: Errors for BDF2 Fixed .....	114
6.1-6. Adaptive Time Integration: Errors for BDF2 Adaptive .....	115
7.1-1. Enclosure Radiation 2D .....	118
7.2-1. 2D Full Enclosure Radiation .....	120
7.3-1. Enclosure Radiation .....	124
7.4-1. Partial Enclosure Radiation .....	125
8.1-1. First Order Reaction (Spatially Varying Temperature) .....	127
8.2-1. First Order Reaction .....	129
9.1-1. Thermal Postprocess .....	133
9.2-1. Local Cartesian Coordinate System .....	135
9.3-1. Local Cylindrical Coordinate System .....	136

## LIST OF TABLES

2.1-1. Steady Heat Conduction: Convergence Rates for Hex8 Meshes .....	29
2.2-1. Steady Heat Conduction: Convergence Rates for Hex20 Meshes .....	30

2.3-1. Steady Heat Conduction: Convergence Rates for Hex27 Meshes .....	32
2.4-1. Steady Heat Conduction: Convergence Rates for Tet4 Meshes .....	33
2.5-1. Steady Heat Conduction: Convergence Rates for Tet4Tet10 Meshes.....	35
2.6-1. Steady Heat Conduction: Convergence Rates for Tet10 Meshes .....	37
2.7-1. Transient Heat Conduction: Convergence Rates for Hex8 Meshes .....	38
2.8-1. Transient Heat Conduction: Convergence Rates for Tet4 Meshes.....	39
2.9-1. Transient Heat Conduction: Convergence Rates for Tet4 Solution on Tet10 Meshes ....	40
2.10-1. Transient Heat Conduction: Convergence Rates for Tet10 Meshes .....	42
2.11-1. Min Max Postprocess: Convergence Rates .....	43
3.1-1. Radiative Heat Flux: Convergence Rates for Hex8 Meshes .....	47
3.3-1. Convective Heat Flux: Convergence Rates for Hex8 Meshes .....	49
3.4-1. Thermal Convective BC: Convergence Rates .....	50
3.5-1. Thermal Convective BC: Convergence Rates .....	52
3.6-1. Thermal Heat Flux BC: Convergence Rates .....	54
3.6-2. Thermal Heat Flux BC: Convergence Rates .....	56
3.6-3. Thermal Heat Flux BC: Convergence Rates .....	56
3.6-4. Thermal Heat Flux BC: Convergence Rates .....	58
3.7-1. Thermal Radiative Flux BC: Convergence Rates .....	59
3.7-2. Thermal Radiative Flux BC: Convergence Rates .....	61
3.7-3. Thermal Radiative Flux BC: Convergence Rates .....	62
3.8-1. Steady Advective Conduction: Convergence Rates for 3D Bar2 Meshes .....	64
3.8-2. Transient Heat Conduction: Convergence Rates for 3D Bar2 Meshes .....	66
3.8-3. Transient Heat Conduction: Convergence Rates for 2D Bar2 Meshes.....	66
4.1-1. 1D Flat Contact: Convergence Rates for Hex8 Tied .....	72
4.1-2. 1D Flat Contact: Convergence Rates for Hex8 Resistance.....	72
4.1-3. 1D Flat Contact: Convergence Rates for Tet4 Tied .....	73
4.1-4. 1D Flat Contact: Convergence Rates for Tet4 Resistance .....	74
4.1-5. 1D Flat Contact: Convergence Rates for Hex8-Tet4 Tied .....	75
4.1-6. 1D Flat Contact: Convergence Rates for Hex8-Tet4 Resistance .....	76

4.2-1. 3D Curved Contact: Convergence Rates for Hex8-Hex8 .....	78
4.2-2. 3D Curved Contact: Convergence Rates for Tet4-Tet4 .....	79
4.2-3. 3D Curved Contact: Convergence Rates for Hex8-Tet4 .....	80
4.3-1. Steady Tied Contact: Convergence Rates for Hex8 Meshes .....	81
4.4-1. Steady Heat Conduction: Convergence Rates for Hex20 Meshes .....	83
4.5-1. Steady Heat Conduction: Convergence Rates for Hex27 Meshes .....	85
4.6-1. Steady Tied Contact: Convergence Rates for Tet4 Meshes .....	86
4.7-1. Steady Tied Contact: Convergence Rates for Tet4 Meshes .....	88
4.8-1. Steady Tied Contact: Convergence Rates for Tet10 Meshes .....	88
4.9-1. Steady Tied DASH Contact: Convergence Rates for Tet10 Meshes .....	90
4.10-1. Transient Tied Contact: Convergence Rates for Tet10 Meshes .....	92
4.11-1. Transient Tied Contact: Convergence Rates for Tet10 Meshes .....	93
4.12-1. Tied Contact Transient Heat Conduction: Convergence Rates for Hex8 Meshes .....	95
4.13-1. Transient Heat Conduction with Tied Contact: Convergence Rates for Tet4 Meshes .....	96
5.1-1. CDFEM Element Death (Heat Flux): Convergence Rates for Tri3 .....	98
5.1-2. CDFEM Element Death (Heat Flux): Convergence Rates for Tet4 .....	98
5.2-1. Dimensions of problem .....	100
5.2-2. Material properties .....	100
5.2-3. Convergence Rates at $t = 0.9$ .....	102
5.3-1. Element Death (Heat Flux): Convergence Rates for Hex8 .....	103
5.3-2. Element Death (Heat Flux): Convergence Rates for Quad4 .....	104
5.3-3. Element Death (Heat Flux): Convergence Rates for Tri3 .....	104
5.3-4. 2D Element Death (Heat Flux): Convergence Rates for Quad4 .....	106
5.3-5. Element Death (Heat Flux): Convergence Rates for Hex8 .....	108
6.1-1. Adaptive Time Integration: Convergence Rates for First Order Fixed .....	110
6.1-2. Adaptive Time Integration: Convergence Rates for First Order Adaptive .....	111
6.1-3. Adaptive Time Integration: Convergence Rates for Second Order Fixed .....	112
6.1-4. Adaptive Time Integration: Convergence Rates for Second Order Adaptive .....	113
6.1-5. Adaptive Time Integration: Convergence Rates for BDF2 Fixed .....	114

6.1-6. Adaptive Time Integration: Convergence Rates for BDF2 Adaptive .....	115
7.1-1. Dimensions of problem .....	116
7.1-2. Material properties .....	117
7.1-3. Enclosure Radiation 2D: Convergence Rates .....	118
7.2-1. 2D Full Enclosure Radiation: Convergence Rates .....	120
7.3-1. Dimensions of problem .....	121
7.3-2. Material properties .....	121
7.3-3. Enclosure Radiation: Convergence Rates .....	123
7.4-1. Partial Enclosure Radiation: Convergence Rates .....	125
8.1-1. First Order Reaction (Spatially Varying Temperature): Convergence Rates for Hex8 Meshes .....	127
8.2-1. First Order Reaction: Convergence Rates for Hex8 Meshes .....	129
8.4-1. PMDI Plugin Test: Initial Conditions .....	131
9.1-1. Thermal Postprocess: Convergence Rates .....	133
9.2-1. Local Cartesian Coordinate System: Convergence Rates .....	134
9.3-1. Local Cylindrical Coordinate System: Convergence Rates .....	136

# 1. INTRODUCTION

The Sierra/TF Verification Manual is divided into chapters based on related capabilities. Each section of a chapter represents a distinct verification test. Some problems that are not yet fully documented are listed at the end of each chapter.

All of these verification tests are run nightly by the development team to continually verify code accuracy under mesh refinement. The graphics and charts in this document are automatically generated by the nightly test runs.

The test files for these problems may be found in the Sierra regression test repository. Most are in the sub-directory called “verification.”

```
aria_rtest/verification
```

All tests are assigned the keyword “verification”. Those that appear in this document also have the keyword “self-documenting”.

For each test, the approximate finite element solution  $T_h$  is compared to the exact solution  $T$  using several global norms, and in some cases using response quantities of interest. This is repeated over a series of uniformly refined meshes (not necessarily nested) with mesh sizes  $\{h_i\}$ , giving a sequence of errors  $\{E_i\}$ . For each pair of meshes, a convergence rate is estimated using the formula

$$r_i \equiv \log(E_i/E_{i-1})/\log(h_i/h_{i-1}). \quad (1.1)$$

The convergence of  $r_i$  to the expected rate is monitored as the mesh is refined. A test passes if all of the estimated convergence rates on the finest pair of meshes are within a given tolerance of the expected rates.

## 2. BASIC THERMAL TESTS

### 2.1. STEADY HEAT CONDUCTION: HEX8 MESHES

This problem tests basic steady state heat conduction in a 3D domain. The geometry consists of a unit cube. A variety of different source terms and boundary conditions are simultaneously applied. The exact solution is a manufactured solution.

#### 2.1.1. Features Tested

Basic heat conduction on Hex8 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

#### 2.1.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

#### 2.1.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

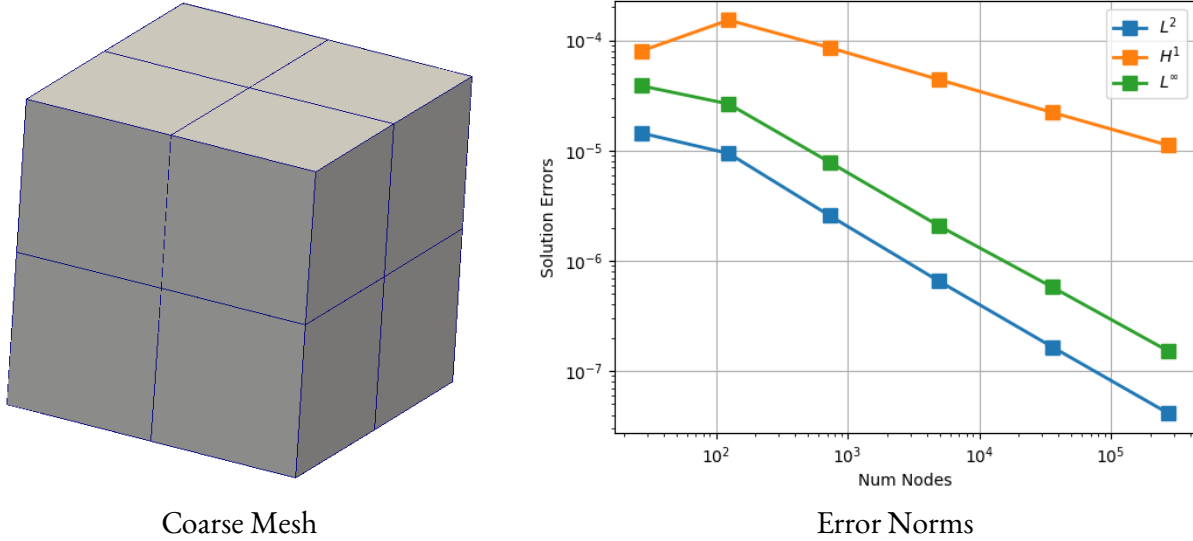
#### 2.1.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, z) = 1 + (x - x^2)^2(y - y^2)^2(z - z^2)^2.$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).



**Figure 2.1-1.. Steady Heat Conduction: Hex8 Meshes**

**Table 2.1-1.. Steady Heat Conduction: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
125	0.83	-1.27	0.74
729	2.20	0.98	2.07
4913	2.15	1.05	2.08
35940	2.08	1.03	1.94
274600	2.05	1.02	1.97

For input decks see Appendix [12.1.1](#).

## 2.2. STEADY HEAT CONDUCTION: HEX20 MESHES

This problem tests basic steady state heat conduction in a 3D domain. The geometry consists of a unit cube. A variety of different source terms and boundary conditions are simultaneously applied. The exact solution is a manufactured solution.

### 2.2.1. Features Tested

Basic heat conduction on Hex20 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

## 2.2.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

## 2.2.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

## 2.2.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, z) = 1 + (x - x^2)^2(y - y^2)^2(z - z^2)^2.$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

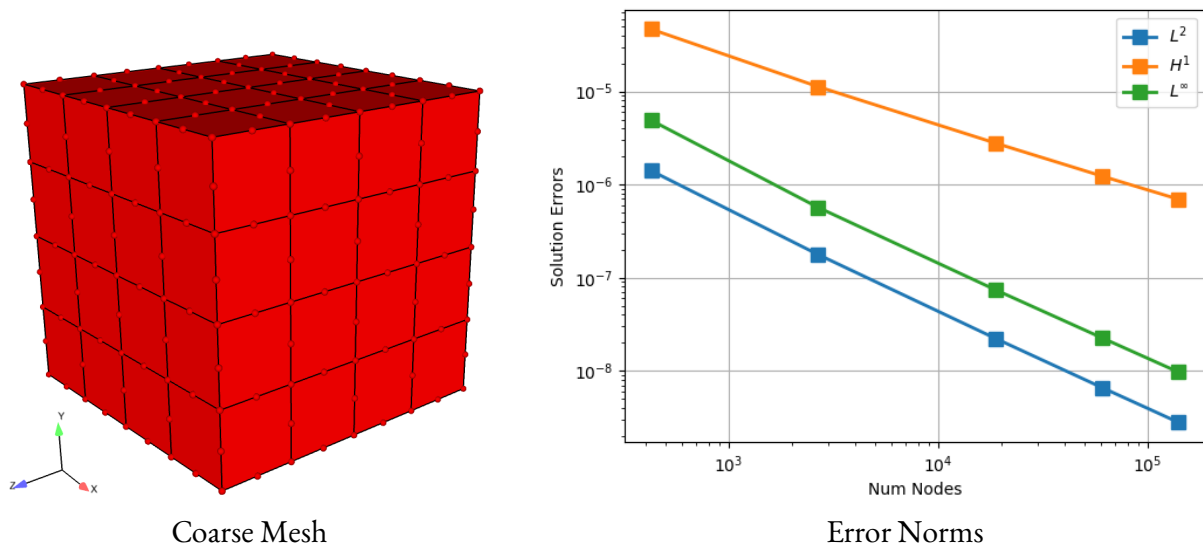
**Table 2.2-1.. Steady Heat Conduction: Convergence Rates for Hex20 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
2673	3.39	2.32	3.52
18780	3.19	2.15	3.13
60620	3.11	2.08	3.04
140500	3.08	2.06	3.01

For input decks see Appendix [12.1.2](#).

## 2.3. STEADY HEAT CONDUCTION: HEX27 MESHES

This problem tests basic steady state heat conduction in a 3D domain. The geometry consists of a unit cube. A variety of different source terms and boundary conditions are simultaneously applied. The exact solution is a manufactured solution.



**Figure 2.2-1.. Steady Heat Conduction: Hex20 Meshes**

### 2.3.1. Features Tested

Basic heat conduction on Hex27 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 2.3.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

### 2.3.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

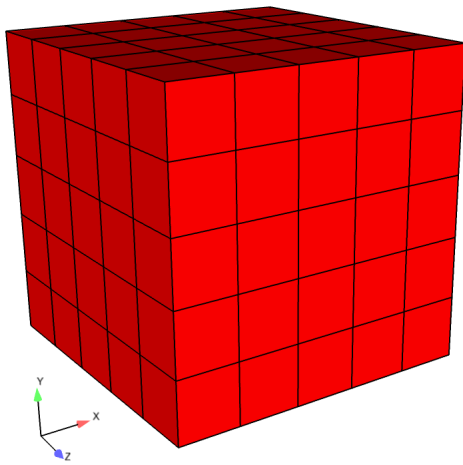
### 2.3.4. Verification of Solution

A manufactured solution is chosen as

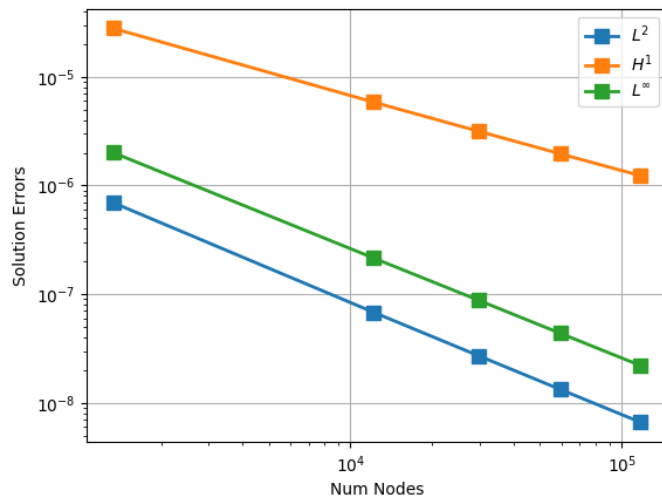
$$T(x, y, z) = 1 + (x - x^2)^2(y - y^2)^2(z - z^2)^2.$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).



Coarse Mesh



Error Norms

**Figure 2.3-1.. Steady Heat Conduction: Hex27 Meshes**

**Table 2.3-1.. Steady Heat Conduction: Convergence Rates for Hex27 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
12170	3.15	2.12	3.03
29790	3.10	2.07	3.01
59320	3.08	2.06	3.02
117600	3.07	2.05	3.01

For input decks see Appendix [12.1.3](#).

## 2.4. STEADY HEAT CONDUCTION: TET4 MESHES

This problem is identical to the one in Section 2.1 except that unstructured Tet4 meshes are used instead. The meshes are obtained from Cubit.

### 2.4.1. Features Tested

Basic heat conduction on Tet4 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 2.4.2. Boundary Conditions

Same as in Section 2.1.

### 2.4.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 2.4.4. Verification of Solution

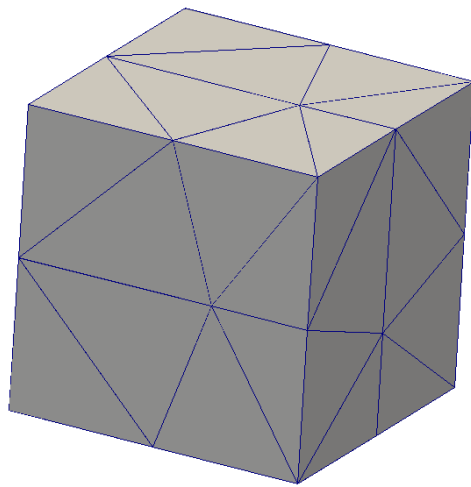
Same as in Section 2.1.

Unlike the Hex8 case, we have observed in many cases, and in this test, that the convergence rate for the temperature in the  $L^\infty$  norm is somewhat less than 2, in this case about 1.9. The exact reason for this behavior is unclear.

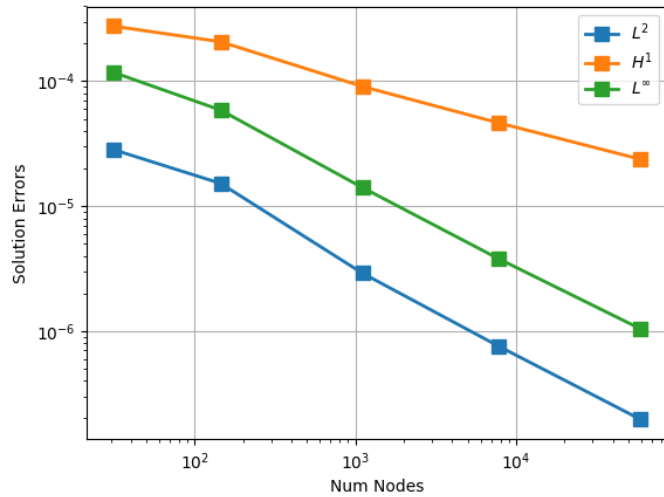
**Table 2.4-1.. Steady Heat Conduction: Convergence Rates for Tet4 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
145	1.21	0.56	1.36
1104	2.45	1.22	2.10
7725	2.07	1.03	2.02
59640	1.99	0.99	1.91

For input decks see Appendix 12.1.4.



Coarse Mesh



Error Norms

Figure 2.4-1.. Steady Heat Conduction: Tet4 Meshes

## 2.5. STEADY HEAT CONDUCTION: TET4TET10 MESHES

This problem is identical to the one in Section 2.1 with the exception of constant thermal conductivity and use of unstructured Tet10 meshes. The meshes are obtained from Cubit.

### 2.5.1. Features Tested

Basic heat conduction with Tet4 solution on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 2.5.2. Boundary Conditions

Same as in Section 2.1.

### 2.5.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

## 2.5.4. Verification of Solution

Same as in Section 2.1.

Unlike the Hex8 case, we have observed in many cases, and in this test, that the convergence rate for the temperature in the  $L^\infty$  norm is somewhat less than 2, in this case about 1.9. The exact reason for this behavior is unclear.

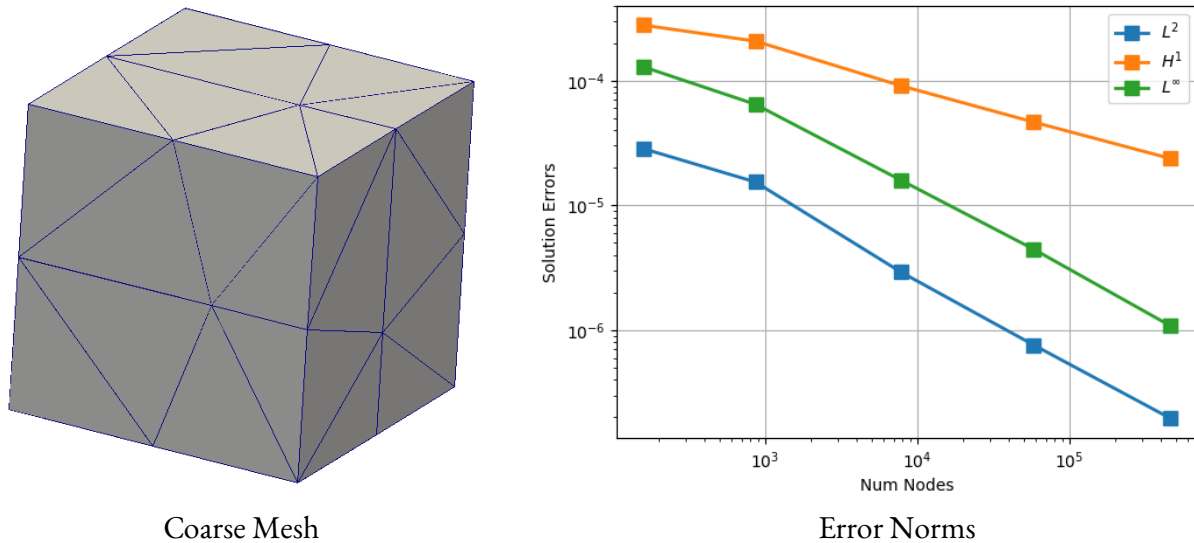


Figure 2.5-1.. Steady Heat Conduction: Tet4 Solutions on Tet10 Meshes

Table 2.5-1.. Steady Heat Conduction: Convergence Rates for Tet4Tet10 Meshes

Num Dofs	$L^2$	$H^1$	$L^\infty$
865	1.09	0.52	1.22
7831	2.26	1.12	1.90
58210	2.01	1.00	1.90
464400	1.96	0.98	2.05

For input decks see Appendix 12.1.5.

## 2.6. STEADY HEAT CONDUCTION: TET10 MESHES

This problem is identical to the one in Section 2.1 except that unstructured Tet10 meshes are used instead. The meshes are obtained from Cubit.

## 2.6.1. Features Tested

Basic heat conduction on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

## 2.6.2. Boundary Conditions

Same as in Section 2.1.

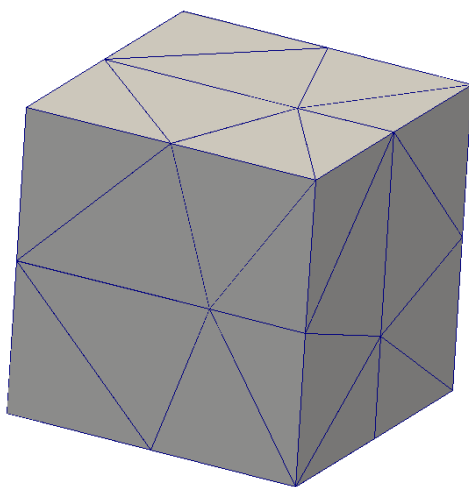
## 2.6.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

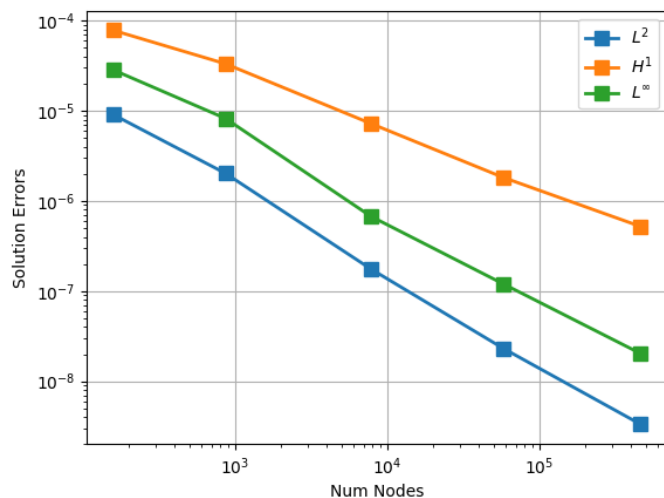
## 2.6.4. Verification of Solution

Same as in Section 2.1.

Unlike the Hex8 case, we have observed in many cases, and in this test, that the convergence rate for the temperature in the  $L^\infty$  norm is somewhat less than 3, in this case about 2.7. The exact reason for this behavior is unclear.



Coarse Mesh



Error Norms

**Figure 2.6-1.. Steady Heat Conduction: Tet10 Meshes**

For input decks see Appendix 12.1.6.

**Table 2.6-1.. Steady Heat Conduction: Convergence Rates for Tet10 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
865	2.65	1.51	2.19
7831	3.32	2.08	3.41
58210	3.04	2.07	2.58
464400	2.80	1.80	2.57

## 2.7. TRANSIENT HEAT CONDUCTION: HEX8 MESHES

This problem tests basic transient heat conduction in a 3D domain. The geometry consists of a unit cube.

### 2.7.1. Features Tested

Basic transient heat conduction on Hex8 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 2.7.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

### 2.7.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

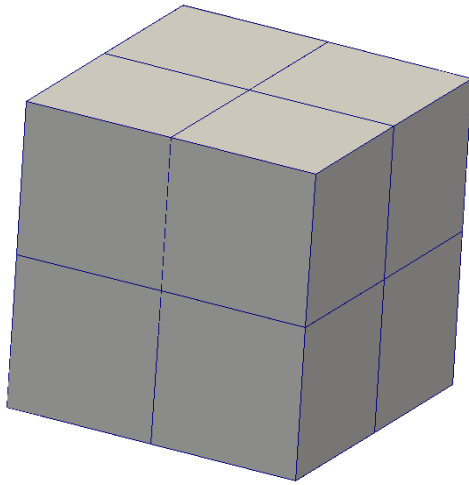
### 2.7.4. Verification of Solution

A manufactured solution is chosen as

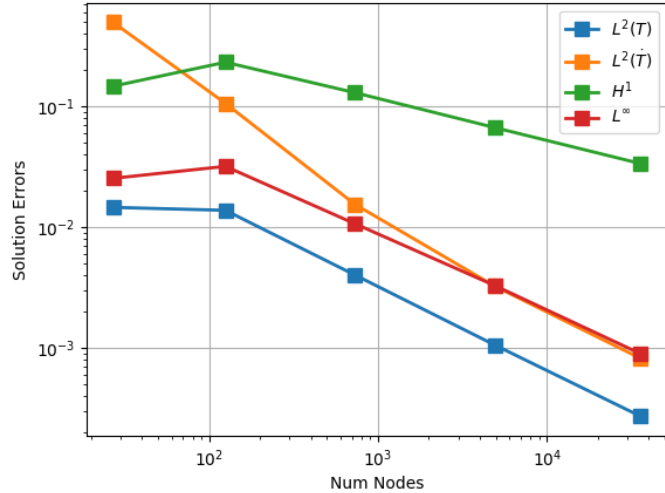
$$T(x, y, z, t) = (x - x^2)^2 (y - y^2)^2 (z - z^2)^2 m(t) + 1,$$
$$m(t) = 10^4 (1 - \exp(-t) + t \exp(-(t - 1)^2))$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution at final time are computed in the  $L^2$  norm of  $T$  and  $\dot{T}$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, 2 and 1, respectively (within a tolerance).



Coarse Mesh



Error Norms

**Figure 2.7-1.. Transient Heat Conduction: Hex8 Meshes**

**Table 2.7-1.. Transient Heat Conduction: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
125	0.12	3.02	-0.89	-0.45
729	2.09	3.28	0.98	1.85
4913	2.09	2.46	1.05	1.86
35940	2.06	2.07	1.04	1.96

For input decks see Appendix [12.1.7](#).

## 2.8. TRANSIENT HEAT CONDUCTION: TET4 MESHES

This problem tests basic transient heat conduction in a 3D domain as in Section [2.7](#). The geometry consists of a unit cube and a single bulk fluid element.

### 2.8.1. Features Tested

Basic transient heat conduction on Tet4 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; bulk fluid element; heat flux and source term from Encore user subroutines.

## 2.8.2. Boundary Conditions

Identical to Section 2.7 except one convective flux boundary condition is now connected to a bulk fluid element.

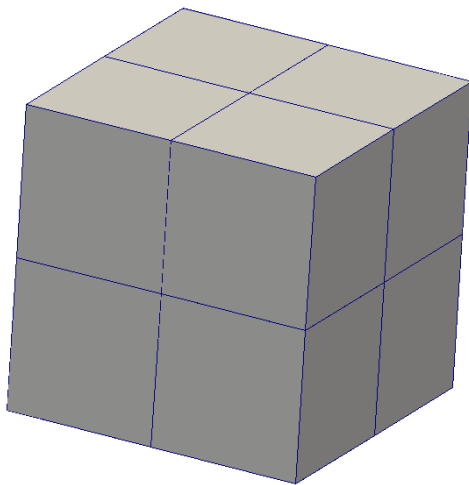
## 2.8.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

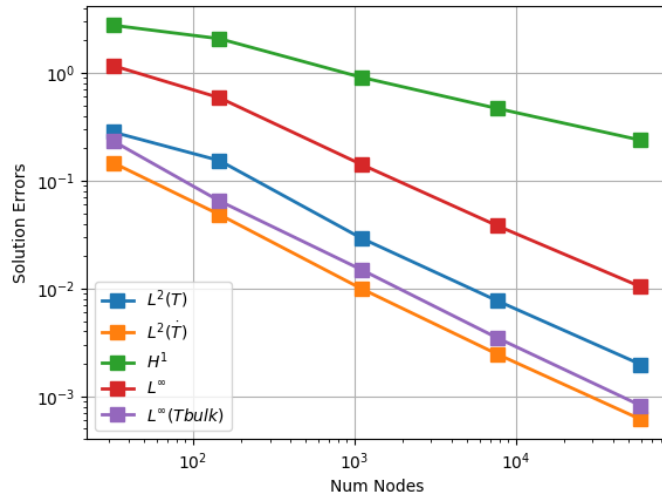
## 2.8.4. Verification of Solution

A manufactured solution is chosen as in Section 2.7.

For each mesh, the errors in the temperature solution at final time are computed in the  $L^2$  norm of  $T$  and  $\dot{T}$ ,  $L^\infty$  and  $H^1$  norms. As in Section 2.4, we see convergence rates for  $L^\infty$  that are slightly less than 2.



Coarse Mesh



Error Norms

Figure 2.8-1.. Transient Heat Conduction: Tet4 Meshes

Table 2.8-1.. Transient Heat Conduction: Convergence Rates for Tet4 Meshes

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$	$L^\infty(T_{bulk})$
146	1.20	2.19	0.57	1.36	2.53
1105	2.45	2.34	1.22	2.11	2.17
7726	2.07	2.16	1.03	2.02	2.26
59640	1.99	2.04	0.99	1.91	2.12

For input decks see Appendix [12.1.8](#).

## 2.9. TRANSIENT HEAT CONDUCTION: TET4TET10 MESHES

This problem tests basic transient heat conduction in a 3D domain as in Section [2.8](#). The geometry consists of a unit cube.

### 2.9.1. Features Tested

Basic transient heat conduction Tet4 analysis on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 2.9.2. Boundary Conditions

Identical to Section [2.8](#)

### 2.9.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 2.9.4. Verification of Solution

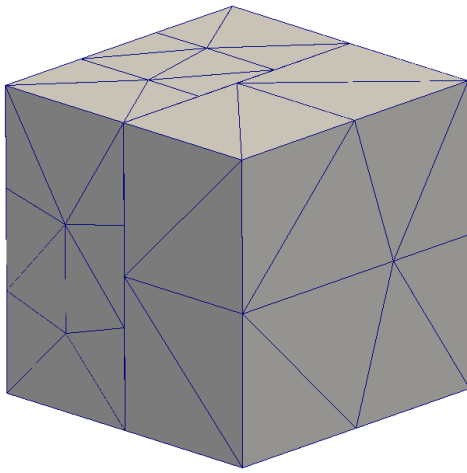
A manufactured solution is chosen as in Section [2.8](#).

For each mesh, the errors in the temperature solution at final time are computed in the  $L^2$  norm of  $T$  and  $\dot{T}$ ,  $L^\infty$  and  $H^1$  norms. As in Section [2.8](#), we see convergence rates for  $L^\infty$  that are slightly less than 2.

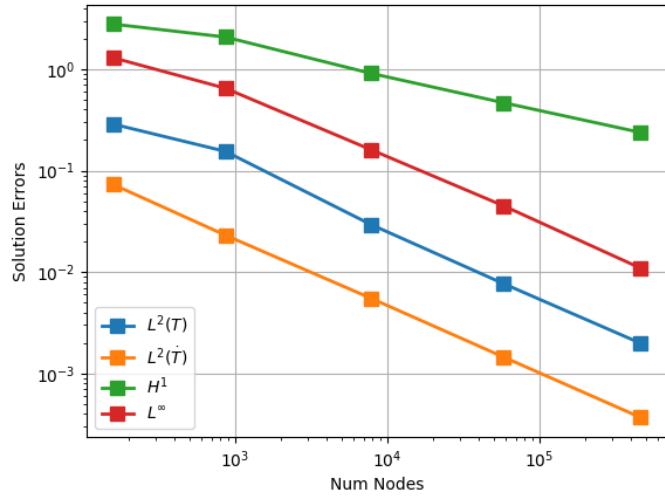
**Table 2.9-1.. Transient Heat Conduction: Convergence Rates for Tet4 Solution on Tet10 Meshes**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
865	1.09	2.03	0.52	1.22
7831	2.26	1.95	1.12	1.90
58210	2.01	2.00	1.00	1.90
464400	1.96	1.98	0.98	2.05

For input decks see Appendix [12.1.9](#).



Coarse Mesh



Error Norms

Figure 2.9-1.. Transient Heat Conduction: Tet4 Solution on Tet10 Meshes

## 2.10. TRANSIENT HEAT CONDUCTION: TET10 MESHES

This problem tests basic transient heat conduction in a 3D domain as in Section 2.7. The geometry consists of a unit cube.

### 2.10.1. Features Tested

Basic transient heat conduction on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 2.10.2. Boundary Conditions

Identical to Section 2.7

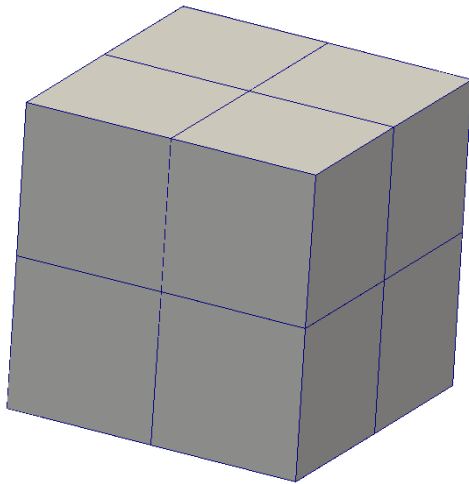
### 2.10.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

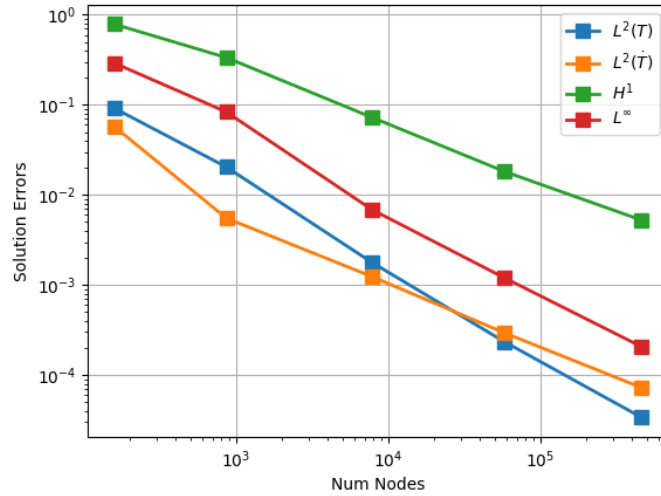
## 2.10.4. Verification of Solution

A manufactured solution is chosen as in Section 2.7.

For each mesh, the errors in the temperature solution at final time are computed in the  $L^2$  norm of  $T$  and  $\dot{T}$ ,  $L^\infty$  and  $H^1$  norms. As in Section 2.6, we see convergence rates for  $L^\infty$  that are slightly less than 2.



Coarse Mesh



Error Norms

Figure 2.10-1.. Transient Heat Conduction: Tet10 Meshes

Table 2.10-1.. Transient Heat Conduction: Convergence Rates for Tet10 Meshes

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
865	2.66	4.15	1.51	2.21
7831	3.32	2.04	2.08	3.40
58210	3.03	2.15	2.07	2.61
464400	2.79	2.02	1.80	2.55

For input decks see Appendix 12.1.10.

## 2.11. POSTPROCESS MIN/MAX

### 2.11.1. Problem Description

This problem tests the min/max postprocessors in Aria.

## 2.11.2. Features Tested

min max postprocessors

## 2.11.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surfaces 1-4.

A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

## 2.11.4. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant.

## 2.11.5. Verification of Solution

The manufactured solution is

$$\sin(7x) \sin(8y).$$

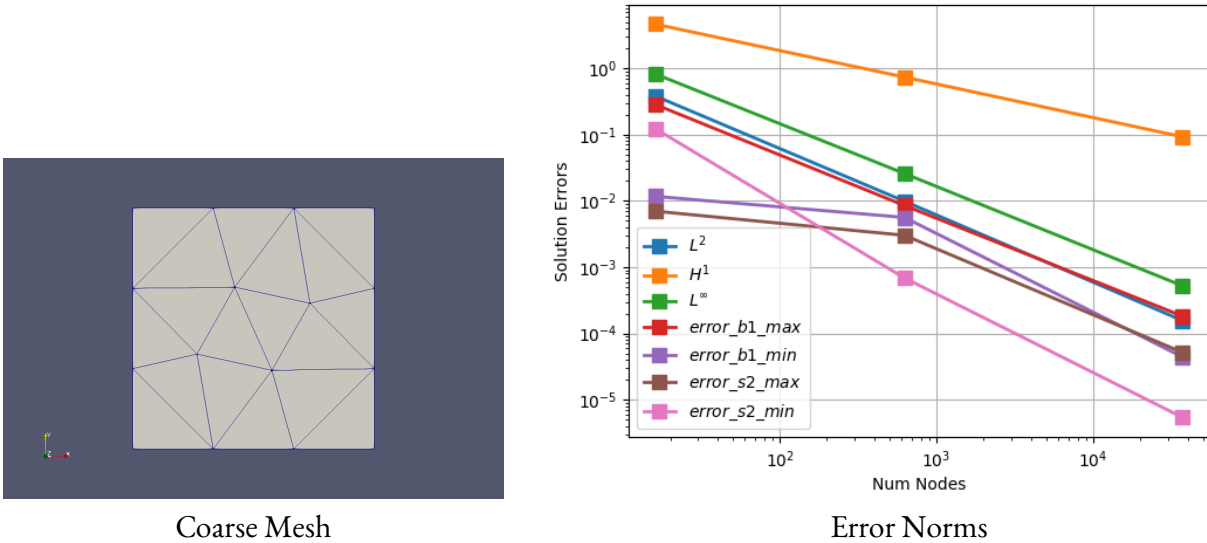
For each uniformly refined mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms and for various postprocessors. Additionally, the nodal maximum and minimum values on both block 1 and surface 2 are computed using Encore postprocessors and the convergence of these values is compared as well. Since the maximum and minimums are nodal, the location of the nodes will reflect the max/min values produced for a given mesh. Provided that the mesh is uniformly refined (without smoothing that may shift the nodal locations), every mesh refinement will produce a better result, dependent on how much closer to the maximum/minimum true solution the new nodes are.

**Table 2.11-1.. Min Max Postprocess: Convergence Rates**

Num Dofs	$L^2$	$H^1$	$L^\infty$	$error\_b1\_max$	$error\_b1\_min$	$error\_s2\_max$	$error\_s2\_min$
625	2.00	1.00	1.89	1.92	0.40	0.46	2.83
37249	2.03	1.02	1.91	1.88	2.37	1.99	2.37

## 2.12. ADAPTIVITY

This problem is identical to the one in Section 2.4 except that we use adaptive mesh refinement to refine from a coarse base mesh obtained from Cubit.



**Figure 2.11-1.. Min Max Postprocess**

### 2.12.1. Features Tested

Basic heat conduction on Tet4 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines; adaptive mesh refinement; local error indicators based on jump in heat flux.

### 2.12.2. Boundary Conditions

Same as in Section 2.1.

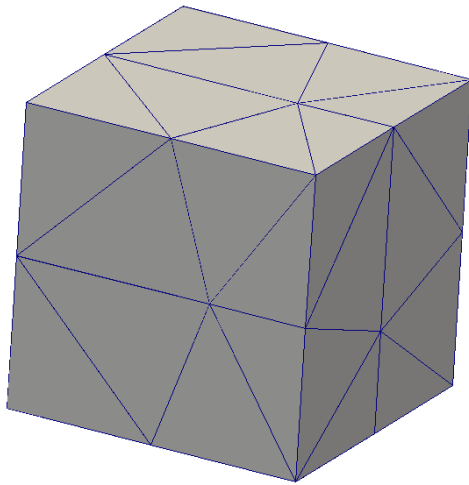
### 2.12.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

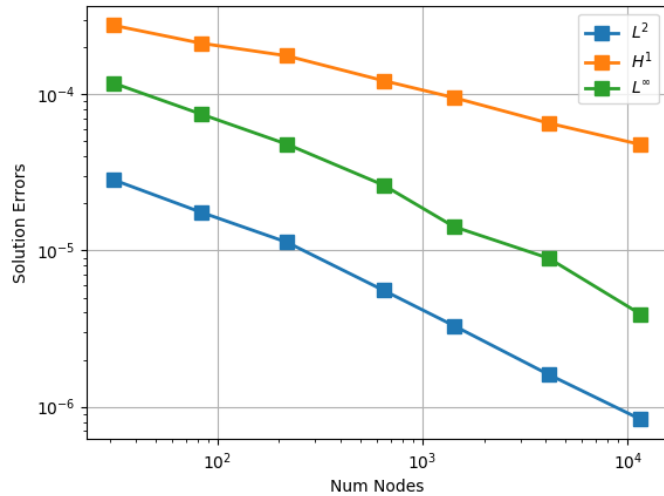
### 2.12.4. Verification of Solution

The mesh is adapted using code from Sierra/Percept that refines tetrahedral meshes without any hanging nodes (conformal meshes only). The element error indicator is computed using a residual-based error indicator in Encore, that computes the integrated jump in the normal heat flux across inter-element faces. The input file is configured to refine elements so that the sum of the error in the refined elements is approximately 75% of the total error in all elements.

Because of variability in the meshes, we expect the error reduction to be noisy. In this case, we use linear least squares to estimate the slope of the error on a log-log plot against mesh size. Since the solution is smooth we also expect the meshes to eventually refine everywhere. We estimate convergence in the usual error norms and observe rates close to the theoretical ones (second order convergence for the  $L^2$  and  $L^\infty$  norms and first order convergence for the  $H^1$  norm). Mesh size is estimated using the formula  $h \approx N^{-1/3}$ , where  $N$  is the number of nodes in the mesh.



Coarse Mesh



Error Norms

**Figure 2.12-1.. Steady Heat Conduction: Tet4 Meshes (Adaptive Mesh Refinement)**

Documentation for the following tests is in progress:

```

1 nlin_verify1/1dnonlin_verify1.test|np8
2 o_2d/aniso_2d.test|np8
3 o_3d/aniso_3d.test|np8
4 shell_2d/cyl_shell_2d.test|np8
5 shell_3d/cyl_shell_3d.test|np8
6 in_C_fi/nonlin_C_fi.test|np1
7 in_C_trap/nonlin_C_trap.test|np1
8 ce_parab/source_parab.test|np1
9 ce_parab_2d/source_parab_2d.test|np1
10 shell_axi/sph_shell_axi.test|np1
11 rical_shell/spherical_shell.test|np4
12 11_nonlin/x11b11_nonlin.test|np1

```

# 3. THERMAL BOUNDARY CONDITIONS

## 3.1. RADIATIVE HEAT FLUX

This problem tests the radiative flux boundary condition under steady state heat conduction in a 2D domain. The geometry consists of a unit square.

### 3.1.1. Features Tested

Basic heat conduction on Quad4 meshes; radiative flux boundary conditions with constant emissivity and reference temperature; radiation form factor from C-style user subroutine; temperature boundary conditions from C-style user subroutine and constant values.

### 3.1.2. Boundary Conditions

At surface 3, the temperature is prescribed from a C-style user subroutine. On surfaces 2 and 4, a constant temperature boundary condition is used. On surface 1, a radiative heat flux condition is prescribed. No source term is needed.

### 3.1.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant.

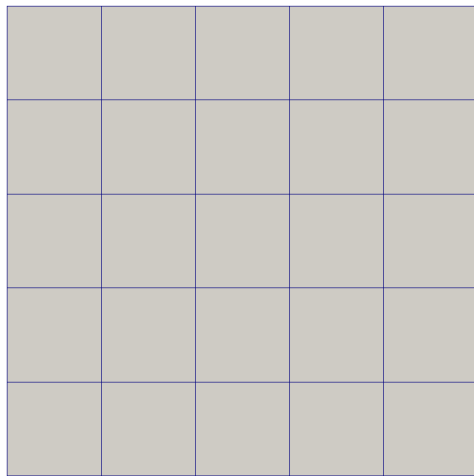
### 3.1.4. Verification of Solution

A manufactured solution is chosen as

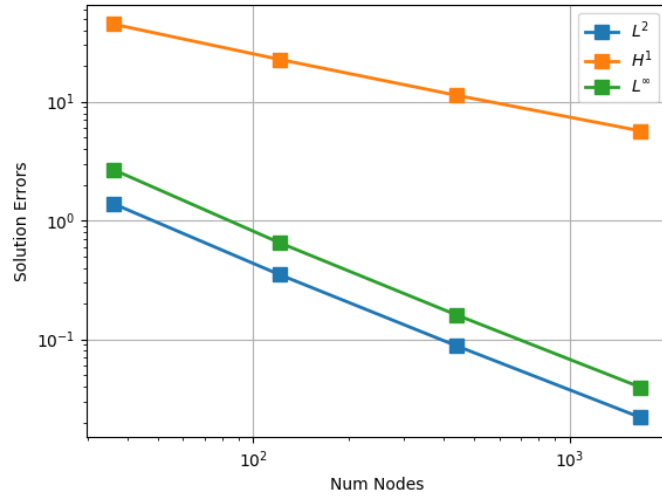
$$T(x, y) = 200 \exp(-\pi y) \sin(\pi x) + 600$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$  and  $L^\infty$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 1, and 2, respectively (within a tolerance).

For input decks see Appendix [12.2.1](#).



Coarse Mesh



Error Norms

**Figure 3.1-1.. Radiative Heat Flux**

**Table 3.1-1.. Radiative Heat Flux: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
121	2.27	1.13	2.34
441	2.14	1.07	2.17
1681	2.07	1.04	2.09

## 3.2. RADIATIVE HEAT FLUX FROM FORTRAN USER SUBROUTINE

This test verifies that a user-supplied subroutine for convective coefficient and reference temperature (restricted to a surface patch) produces the same results as the equivalent input syntax with constant values. The user subroutine is applied to the entire exterior surface, while the case using constant values must be applied only to specific sidesets that span a portion of the exterior surface.

### 3.2.1. Features Tested

Basic heat conduction on a Hex8 mesh; convective and radiative flux BCs, Fortran user subroutines.

### 3.2.2. Boundary Conditions

Convective and radiative flux BCs are applied to the exterior boundary.

### **3.2.3. Material Parameters**

The values of density, thermal conductivity, emissivity and specific heat are all constant.

### **3.2.4. Verification of Solution**

The test compares Exodus output between two input files. The first does not use any user subroutines and instead relies on sidesets to apply the correct convective and radiative boundary conditions with constant coefficients. The second uses a single convective boundary condition with user subroutines for both the convective coefficient and reference temperature. The two input files produce results that agree to the default tolerances in the exodiff script.

For input decks see Appendix [12.2.2](#).

## **3.3. CONVECTIVE HEAT FLUX**

This problem tests the convective flux boundary condition under transient heat conduction in a 2D domain. The geometry consists of a unit square.

### **3.3.1. Features Tested**

Transient heat conduction on Quad4 meshes; convective flux boundary conditions with user subroutines for convective coefficient and reference temperature; temperature boundary conditions from C-style user subroutine and constant values.

### **3.3.2. Boundary Conditions**

At surface 3, the temperature is prescribed from a C-style user subroutine. On surfaces 2 and 4, a constant temperature boundary condition is used. On surface 1, a convective heat flux condition is prescribed. No source term is needed. The initial condition is provided by a C-style user subroutine

### **3.3.3. Material Parameters**

The values of density, thermal conductivity, and specific heat are all constant.

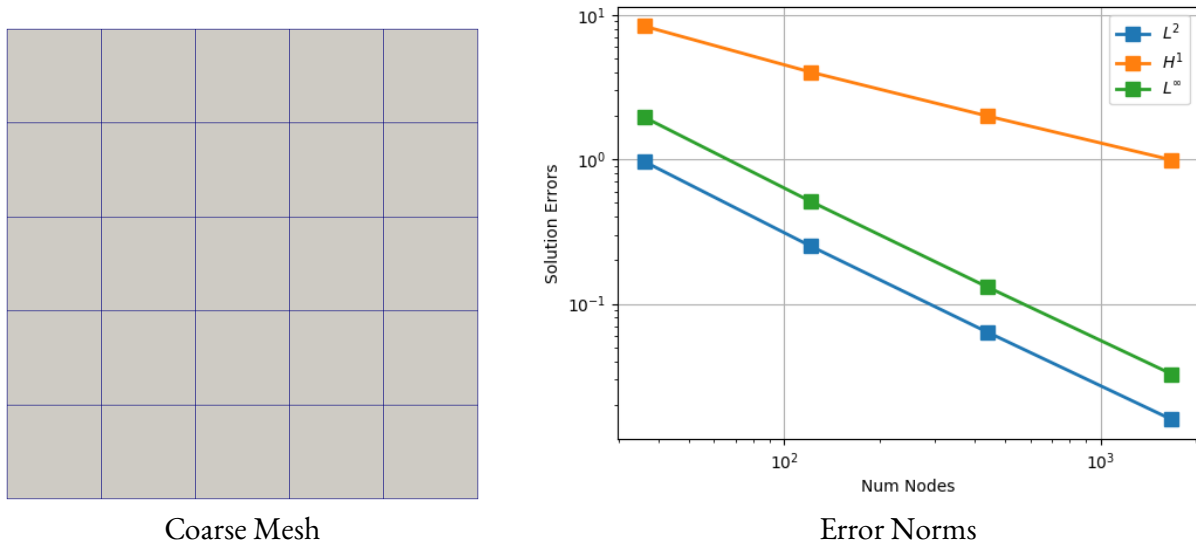
### 3.3.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, t) = 100 \exp(-2\pi^2 t) \sin(\pi x) (\cos(\pi y) + \sin(\pi y))$$

Because the solution is based on eigenfunctions, it satisfies the heat equation with no source term.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).



**Figure 3.3-1.. Convective Heat Flux**

**Table 3.3-1.. Convective Heat Flux: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
121	2.22	1.21	2.22
441	2.13	1.09	2.12
1681	2.07	1.04	2.07

For input decks see Appendix [12.2.3](#).

## 3.4. THERMAL CONVECTIVE FLUX (FORTRAN SUB-ROUTINE)

### 3.4.1. Problem Description

This problem tests the convective flux boundary condition with a convective coefficient Fortran subroutine for a steady thermal problem in a 3D domain whose geometry consists of a unit-sized cube.

### 3.4.2. Features Tested

Convective Flux BC, Convective Coefficient Fortran Subroutine, user subroutine, integrated flux, integrated power

### 3.4.3. Boundary Conditions

Convective flux boundary conditions are imposed on surfaces 1 and 2. Dirichlet BCs are specified using the exact solution on surfaces 3-6. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

### 3.4.4. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant.

### 3.4.5. Verification of Solution

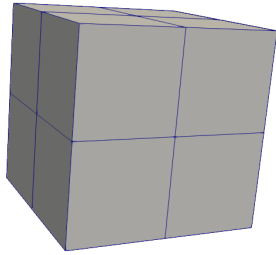
The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + (z + z^2).$$

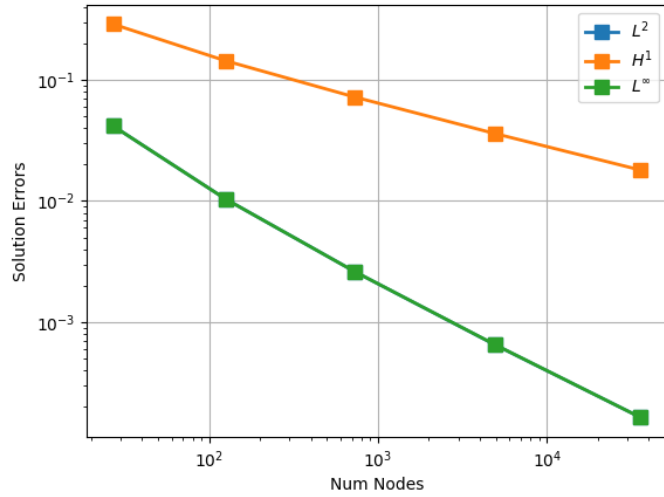
For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms.

**Table 3.4-1.. Thermal Convective BC: Convergence Rates**

Num Dofs	$L^2$	$H^1$	$L^\infty$
125	2.71	1.36	2.71
729	2.36	1.18	2.36
4913	2.18	1.09	2.18
35940	2.09	1.04	2.09



Coarse Mesh



Error Norms

Figure 3.4-1.. Convergence for 3D thermal steady convective flux BCs.

## 3.5. THERMAL CONVECTIVE FLUX (USER FIELD FROM EXODUS READ-IN)

### 3.5.1. Problem Description

This problem evaluates a convective flux boundary condition with a convective coefficient and a reference temperature from an exodus file for a steady thermal problem in a 3D domain whose geometry consists of a unit-sized cube.

### 3.5.2. Features Tested

Convective Flux BC, Convective Coefficient, transfers, user subroutine, integrated flux, integrated power

### 3.5.3. Boundary Conditions

Convective flux boundary conditions are imposed on surfaces 1 and 2. Dirichlet BCs are specified using the exact solution on surfaces 3-6. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

### 3.5.4. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant.

### 3.5.5. Verification of Solution

The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + (z^2 + z).$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms.

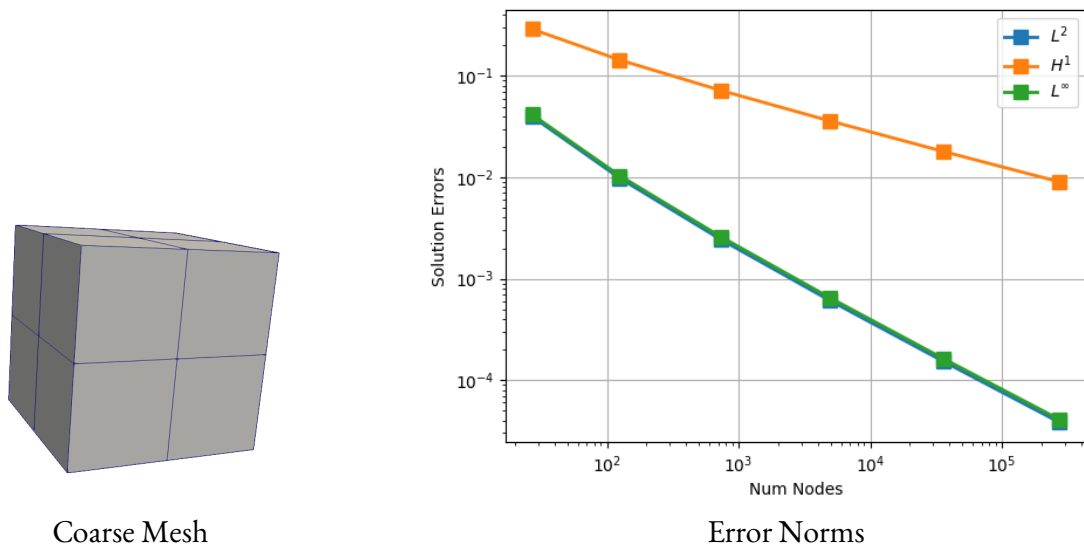


Figure 3.5-1.. Convergence for 3D thermal steady convective flux BCs.

Table 3.5-1.. Thermal Convective BC: Convergence Rates

Num Dofs	$L^2$	$H^1$	$L^\infty$
125	2.74	1.36	2.71
729	2.36	1.18	2.36
4913	2.18	1.09	2.18
35940	2.09	1.05	2.09
274600	2.05	1.02	2.05

## 3.6. THERMAL HEAT FLUX

### 3.6.1. Thermal Heat Flux (Basic)

#### 3.6.1.1. *Problem Description*

This problem tests a steady thermal solution on a unit cube with heat flux boundary conditions imposed on one of the six surfaces. The mesh uses Hex8 elements.

#### 3.6.1.2. *Features Tested*

Basic heat conduction, Calore style heat flux BCs, Integrated Flux Output, Integrated Power Output, Hex8 meshes, user functions.

#### 3.6.1.3. *Boundary Conditions*

Dirichlet BCs are specified using the exact solution on surfaces 2-6. On surface 1, a heat flux BC is specified, using a heat flux of  $2 - \exp(1)$ . A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator. The integrated flux and power are calculated and output as global variables, which should both be equal for a surface with area of 1 and equal to  $2 - \exp(1)$  for all meshes considered.

#### 3.6.1.4. *Material Parameters*

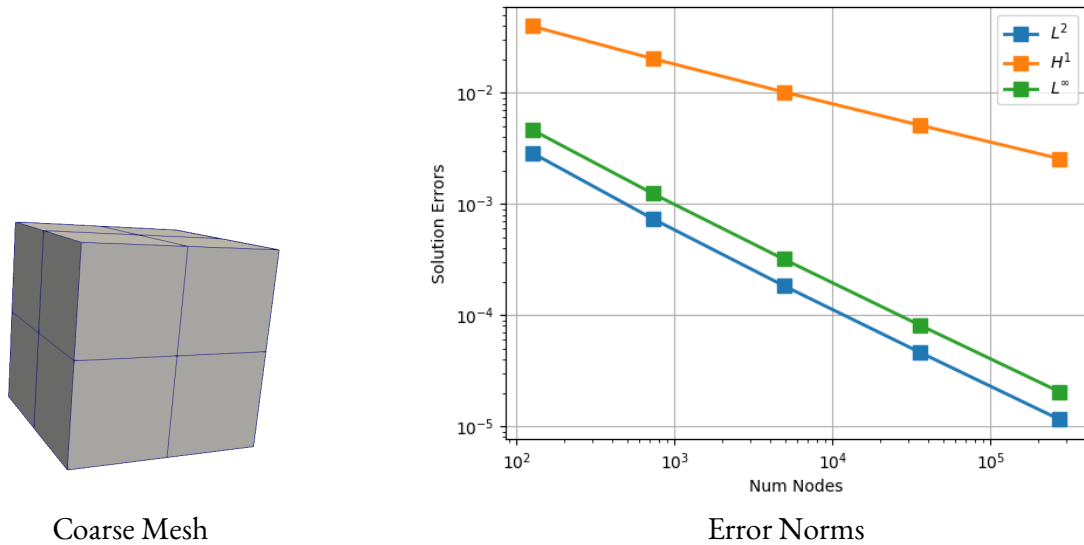
The values of density, thermal conductivity, and specific heat are all constant with the same value for both blocks.

#### 3.6.1.5. *Verification of Solution*

The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + z^2 \exp(z).$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The test passes, only if the observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).



**Figure 3.6-1.. Thermal Heat Flux BC**

**Table 3.6-1.. Thermal Heat Flux BC: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
729	2.33	1.15	2.25
4913	2.17	1.08	2.13
35940	2.09	1.04	2.07
274600	2.04	1.02	2.04

## 3.6.2. Thermal Heat Flux (Flux node variable user field)

### 3.6.2.1. Problem Description

This problem tests a steady thermal solution on a unit cube with heat flux boundary conditions imposed on one of the six surfaces. The mesh uses Hex8 elements.

### 3.6.2.2. Features Tested

Basic heat conduction, Calore style heat flux BCs, Flux Node Variable, User field, Field Scaling, Hex8 meshes, user functions, transfer.

### 3.6.2.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surfaces 1 and 3-6. On surface 2, a heat flux BC is specified, using a flux node variable user field. A source term is applied within all blocks based on

substituting the exact solution into the heat conduction operator. Transfers are specified at the surface.

### 3.6.2.4. Material Parameters

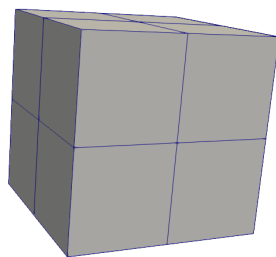
The values of density, thermal conductivity, and specific heat are all constant with the same value for both blocks.

### 3.6.2.5. Verification of Solution

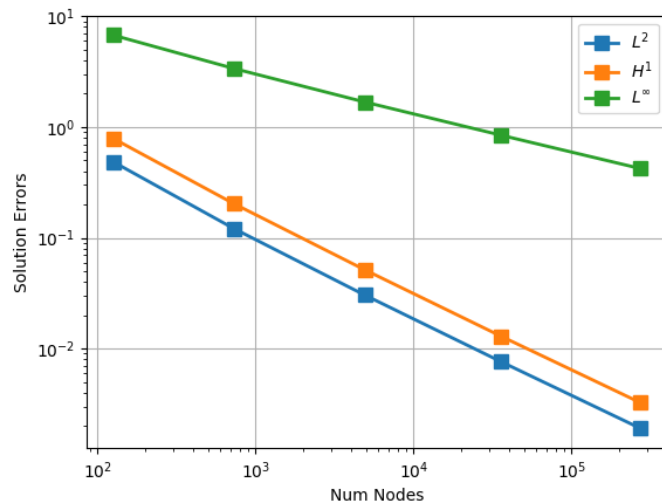
The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + 20 * (z^2 - z) * (1 + x + y + xy).$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The test passes, only if the observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).



Coarse Mesh



Error Norms

Figure 3.6-2.. Thermal Heat Flux BC

## 3.6.3. Thermal Heat Flux (Flux node variable user field)

### 3.6.3.1. Problem Description

This problem tests a steady thermal solution on a unit cube with heat flux boundary conditions imposed on one of the six surfaces. The mesh uses Hex8 elements.

**Table 3.6-2.. Thermal Heat Flux BC: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
729	2.36	2.31	1.18
4913	2.18	2.16	1.09
35940	2.09	2.08	1.04
274600	2.05	2.04	1.02

**3.6.3.2. Features Tested**

Basic heat conduction, Calore style heat flux BCs, User field real nodal vector, Hex8 meshes, user functions, transfers.

**3.6.3.3. Boundary Conditions**

Dirichlet BCs are specified using the exact solution on surfaces 1 and 3-6. On surface 2, a heat flux BC is specified, using a flux vector node variable defined as a user field. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

**3.6.3.4. Material Parameters**

The values of density, thermal conductivity, and specific heat are all constant with the same value for both blocks.

**3.6.3.5. Verification of Solution**

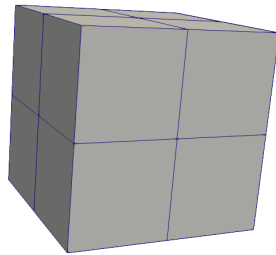
The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + 20 * (z^2 - z) * (1 + x + y + xy).$$

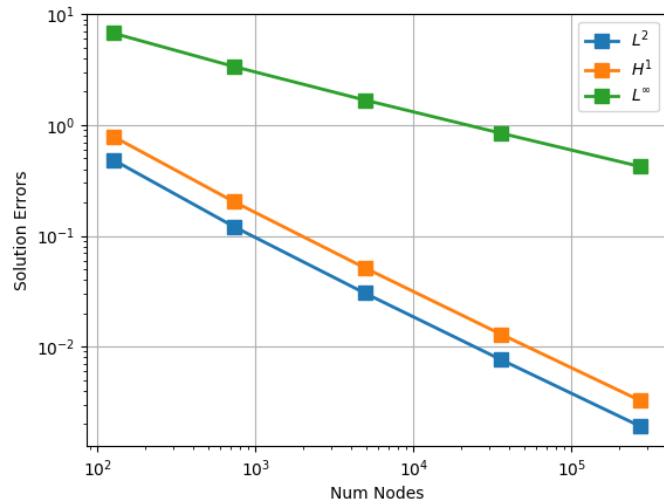
For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The test passes, only if the observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).

**Table 3.6-3.. Thermal Heat Flux BC: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
729	2.36	2.31	1.18
4913	2.18	2.16	1.09
35940	2.09	2.08	1.04
274600	2.05	2.04	1.02



Coarse Mesh



Error Norms

Figure 3.6-3.. Thermal Heat Flux BC

### 3.6.4. Thermal Heat Flux (Fortran Subroutine)

#### 3.6.4.1. Problem Description

This problem tests a steady thermal solution on a unit cube with heat flux boundary conditions imposed on one of the six surfaces. The mesh uses Hex8 elements.

#### 3.6.4.2. Features Tested

Basic heat conduction, Calore style heat flux BCs, Integrated Flux Output, Integrated Power Output, Fortran subroutine, Hex8 meshes, user plugin.

#### 3.6.4.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surfaces 3-6. On surfaces 1 and 2, heat flux BCs are specified, using Fortran subroutines. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

#### 3.6.4.4. Material Parameters

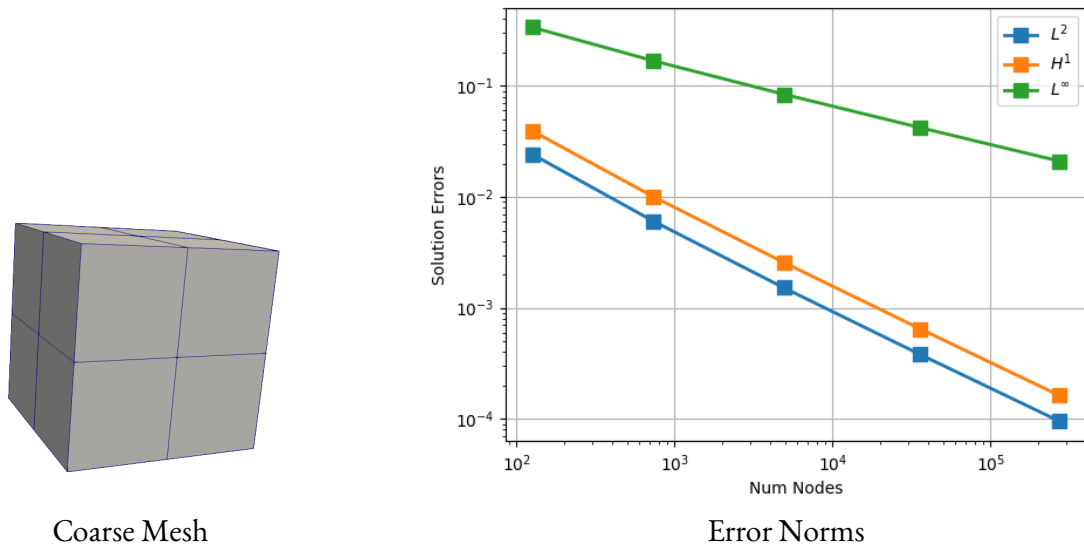
The values of density, thermal conductivity, and specific heat are all constant with the same value for both blocks.

### 3.6.4.5. Verification of Solution

The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + (z^2 + z) * (1 + x + y + xy).$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The test passes, only if the observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).



**Figure 3.6-4.. Thermal Heat Flux BC**

**Table 3.6-4.. Thermal Heat Flux BC: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
729	2.36	2.31	1.18
4913	2.18	2.16	1.09
35940	2.09	2.08	1.05
274600	2.05	2.04	1.02

## 3.7. THERMAL RADIATIVE HEAT FLUX

### 3.7.1. Basic Calore-Style BC

#### 3.7.1.1. Problem Description

This problem evaluates a steady thermal solution with radiative heat flux boundary conditions on a 3D unit cube domain.

#### 3.7.1.2. Features Tested

Basic heat conduction, Calore style radiative heat flux BCs, Integrated Flux Output, Integrated Power Output, Hex8 meshes.

#### 3.7.1.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surfaces 2-6. On surface 1, a radiative heat flux BC is specified with constant emissivity and a radiation form factor of 0.2. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

#### 3.7.1.4. Material Parameters

The values of density, thermal conductivity, specific heat, and emissivity are all constant values.

#### 3.7.1.5. Verification of Solution

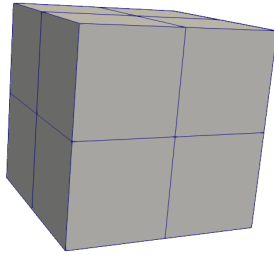
The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + T - \frac{\partial T}{\partial n}(z^2 - z).$$

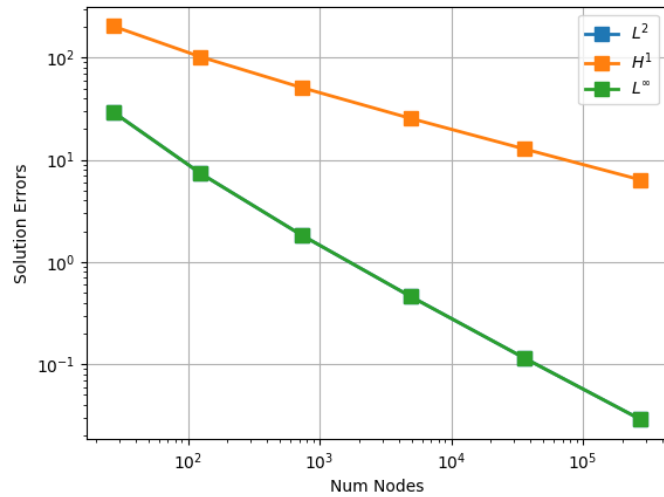
For each discretization, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).

**Table 3.7-1.. Thermal Radiative Flux BC: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
125	2.71	1.36	2.71
729	2.36	1.18	2.36
4913	2.18	1.09	2.18
35940	2.09	1.04	2.09
274600	2.05	1.02	2.05



Coarse Mesh



Error Norms

**Figure 3.7-1.. Thermal Radiative Flux**

## 3.7.2. With Fortran Subroutines

### 3.7.2.1. Problem Description

This problem evaluates a steady thermal solution with radiative heat flux boundary conditions using Fortran subroutines on a 3D unit cube domain.

### 3.7.2.2. Features Tested

Basic heat conduction, Calore style radiative heat flux BCs, Integrated Flux Output, Integrated Power Output, Hex8 meshes, Fortran subroutines.

### 3.7.2.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surfaces 2-6. On surface 1, a radiative heat flux BC is specified with emissivity, reference temperature, and radiation form factor of provided by Fortran subroutines. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

### 3.7.2.4. Material Parameters

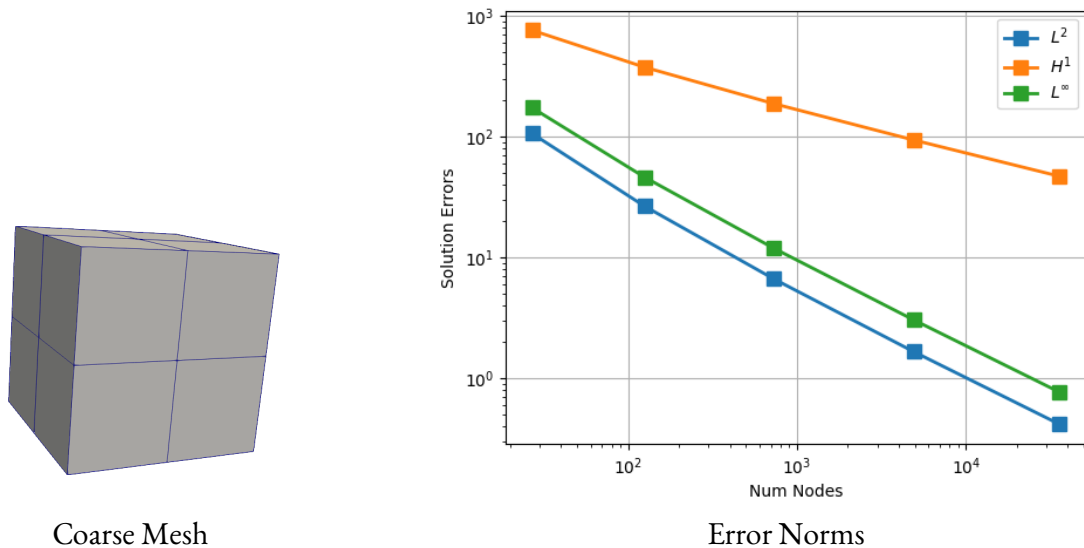
The values of density, thermal conductivity, specific heat, and emissivity are all constant values.

### 3.7.2.5. Verification of Solution

The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + T - \frac{\partial T}{\partial n}(z^2 - z).$$

For each discretization, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).



**Figure 3.7-2.. Thermal Radiative Flux**

**Table 3.7-2.. Thermal Radiative Flux BC: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
125	2.72	1.38	2.59
729	2.36	1.18	2.30
4913	2.18	1.09	2.15
35940	2.09	1.05	2.08

### 3.7.3. With User Subroutines

#### 3.7.3.1. Problem Description

This problem evaluates a steady thermal solution with radiative heat flux boundary conditions with user subroutines on a 3D unit cube domain.

### 3.7.3.2. Features Tested

Basic heat conduction, Calore style radiative heat flux BCs, Integrated Flux Output, Integrated Power Output, Hex8 meshes, user subroutines.

### 3.7.3.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surfaces 2-6. On surface 1, a radiative heat flux BC is specified with emissivity, reference temperature and radiation form factor provided by user subroutines. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

### 3.7.3.4. Material Parameters

The values of density, thermal conductivity, specific heat, and emissivity are all constant values.

### 3.7.3.5. Verification of Solution

The manufactured solution is

$$T(x, y, z) = (x - x^2)^2(y - y^2)^2(z - z^2)^2 + T - \frac{\partial T}{\partial n}(z^2 - z).$$

For each discretization, the errors in the temperature solution are computed in the  $L^2$ ,  $H^1$ , and  $L^\infty$  norms. The observed rates of convergence are 2 (except for the  $L^\infty$  norm, with convergence order 1).

**Table 3.7-3.. Thermal Radiative Flux BC: Convergence Rates**

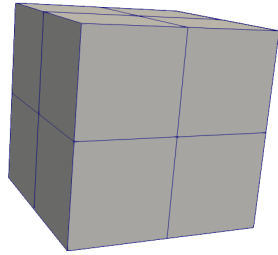
Num Dofs	$L^2$	$L^\infty$	$H^1$
125	2.72	1.38	2.59
729	2.36	1.18	2.30
4913	2.18	1.09	2.15
35940	2.09	1.05	2.08

## 3.8. ADVECTIVE BAR

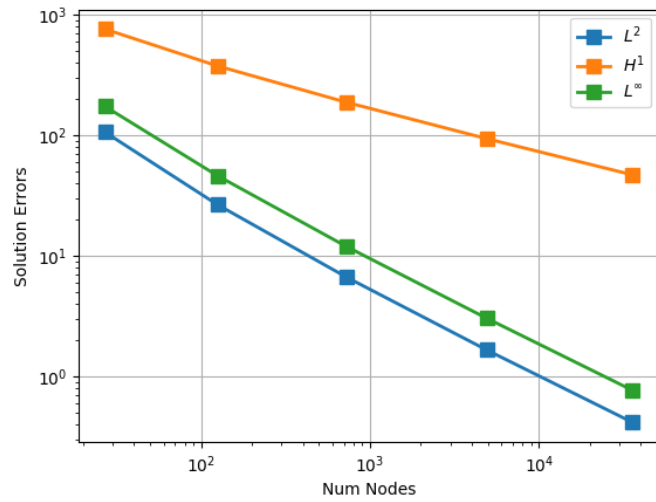
Advective bar model verification tests.

### 3.8.1. Steady Advection-Diffusion

The three dimensional Bar2 meshes of one element block are generated in Cubit.



Coarse Mesh



Error Norms

Figure 3.7-3.. Thermal Radiative Flux

### 3.8.2. Features Tested

Steady heat conduction on 3D Bar2 meshes, Dirichlet boundary conditions, constant source term, advection and SUPG stabilization.

### 3.8.3. Boundary Conditions

$$T(0) = T(1) = 0$$

### 3.8.4. Material Parameters

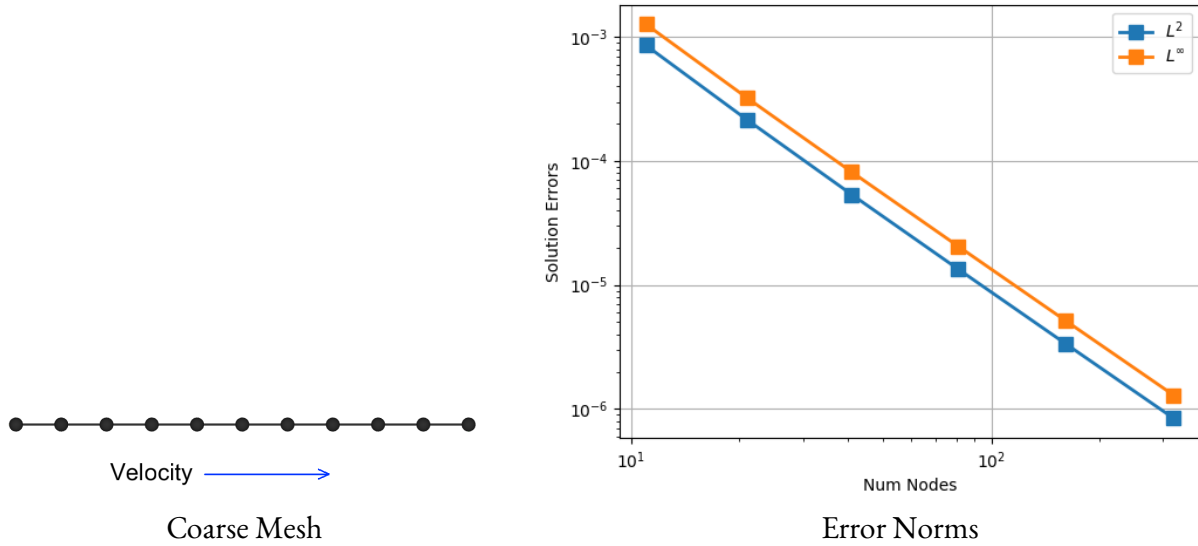
The values of density, thermal conductivity, and specific heat are all constant, equal to one in the block.

### 3.8.5. Verification of Solution

Solution verification is carried out by computing the error in the numerical solution based upon comparison with the analytic solution.

$$T(x) = \frac{1}{\rho CV} \left[ x - \frac{1 - \exp(x\gamma)}{1 - \exp(\gamma)} \right]$$

where  $\gamma = \rho CV/k$  where  $\rho$  is the density,  $C$  is specific heat,  $k$  is the thermal conductivity and  $V$  is the advection velocity. In this test, we find that the convergence rate for the temperature in the  $L^\infty$  and  $L^2$  norms are 2.



**Figure 3.8-1.. Steady Advective Conduction: 3D Bar2 Meshes**

**Table 3.8-1.. Steady Advective Conduction: Convergence Rates for 3D Bar2 Meshes**

Num Dofs	$L^2$	$L^\infty$
21	2.14	2.11
41	2.07	2.06
81	2.04	2.03
161	2.02	2.01
321	2.01	2.01

### 3.8.6. Transient Advection-Diffusion

The three dimensional Bar2 meshes of one element block are generated in Cubit.

### 3.8.7. Features Tested

Transient heat conduction on 3D Bar2 meshes, Dirichlet boundary conditions and Encore function source term.

### 3.8.8. Boundary Conditions

Dirichlet boundary conditions on the bar ends based upon the manufactured solution  $T(x)$

$$T(0) = T(1) = T_i$$

### 3.8.9. Material Parameters

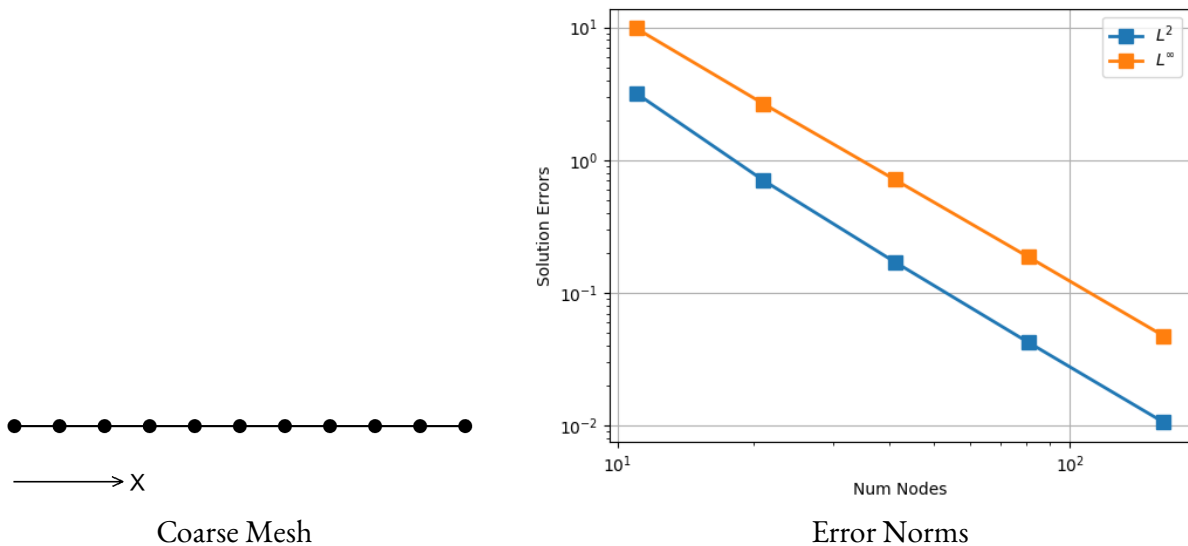
The values of density, thermal conductivity, and specific heat are all constant, equal to one in the bar block.

### 3.8.10. Verification of Solution

Solution verification is carried out by computing the error in the numerical solution based upon comparison with the analytic solution.

$$T(x) = T_i + Atx(x - 1) \exp(-Bt) \exp(-Bx)$$

In this test, we find that the convergence rate for the temperature in the  $L^\infty$  and  $L^2$  norms are 2.



**Figure 3.8-2.. Transient Heat Conduction: 3D Bar2 Meshes**

### 3.8.11. Transient Advection-Diffusion in 2D

The two dimensional Bar2 meshes of one element block are generated in Cubit.

**Table 3.8-2.. Transient Heat Conduction: Convergence Rates for 3D Bar2 Meshes**

Num Dofs	$L^2$	$L^\infty$
21	2.33	2.02
41	2.12	1.97
81	2.05	1.98
161	2.02	1.99

### 3.8.12. Features Tested

Transient heat conduction on 2D Bar2 meshes, Dirichlet boundary conditions and Encore function source term.

### 3.8.13. Boundary Conditions

Dirichlet boundary conditions on the bar ends

$$T(0) = T(1) = T_i$$

### 3.8.14. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in the bar block.

### 3.8.15. Verification of Solution

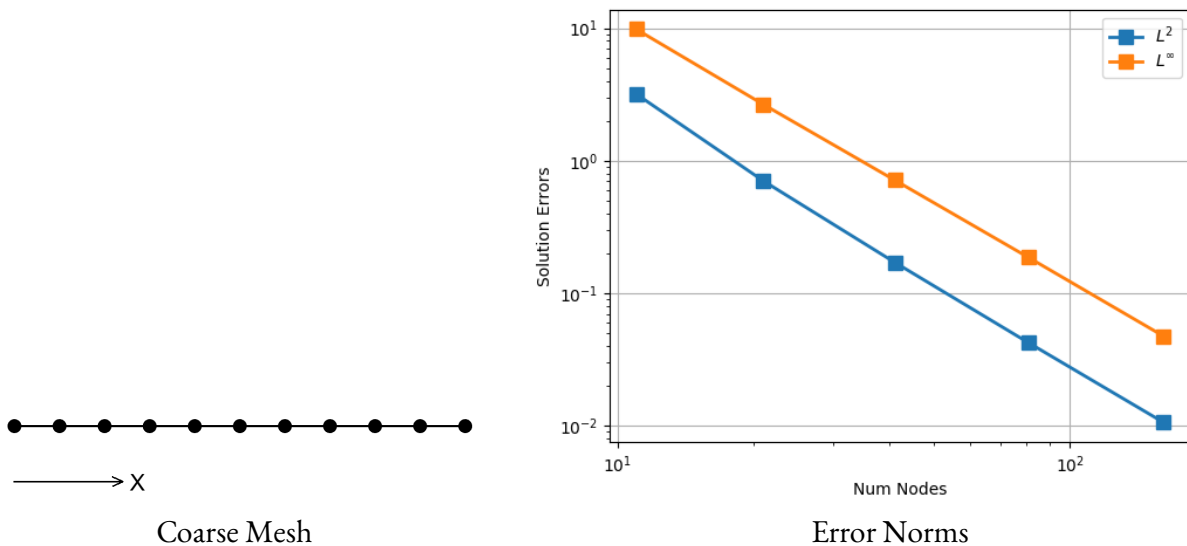
Solution verification is carried out by computing the error in the numerical solution based upon comparison with the analytic solution.

$$T(x) = T_i + Atx(x - 1) \exp(-Bt) \exp(-Bx)$$

In this test, we find that the convergence rate for the temperature in the  $L^\infty$  and  $L^2$  norms are 2.

**Table 3.8-3.. Transient Heat Conduction: Convergence Rates for 2D Bar2 Meshes**

Num Dofs	$L^2$	$L^\infty$
21	2.33	2.02
41	2.12	1.97
81	2.05	1.98
161	2.02	1.99



**Figure 3.8-3.. Transient Heat Conduction: Bar2 Meshes**

## 3.9. SOLUTION VERIFICATION

This test is for a Mock AFF (including a metal case, foam, mock components, and temperature-dependent properties) that uses extrapolation to determine an approximation to the exact solution as a function of the results from three levels of meshes.

### 3.9.1. Features Tested

Extrapolation, Radiative flux boundary condition

### 3.9.2. Material Parameters

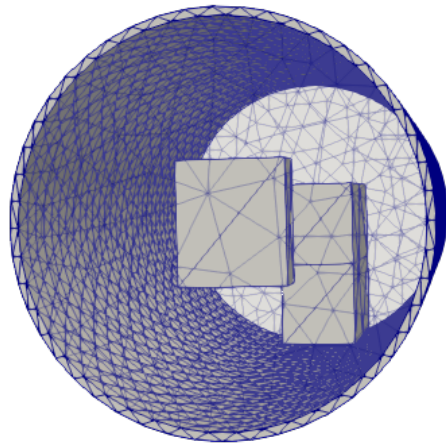
Constant density, emissivity. Temperature dependent user functions for specific heat and thermal conductivity.

### 3.9.3. Verification of Solution

Quantities of interest are the maximum, minimum, and average temperatures on both blocks and points. There is no manufactured solution in this case, instead an extrapolated solution is calculated and used to measure convergence and approximate the absolute error for a given mesh resolution.

Documentation for the following tests is in progress:

```
nic_material_decomposition/organic_material_decomposition.test | np4
```



**Figure 3.9-1.. Mock AFF Solution Verification**

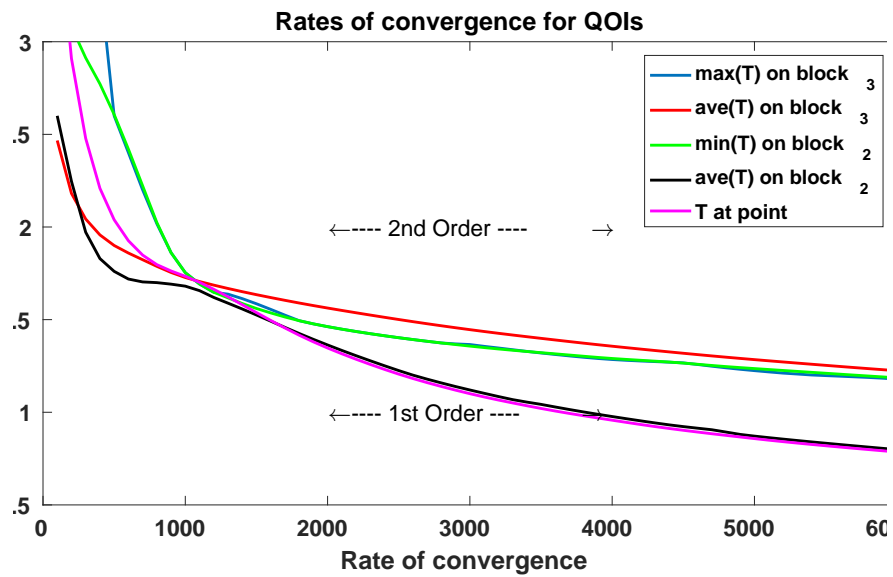


Figure 3.9-2.. The convergence rates can vary over time and between QOIs

# 4. THERMAL CONTACT

## 4.1. 1D FLAT CONTACT

This problem tests thermal contact along a flat surface using 3D domains. The geometry consists of two thick blocks, which are in contact along a common flat surface. The mesh nodes on either side of the contact surface are not aligned in general.

In this problem we observe sub-optimal convergence rates in the  $L^\infty$  norm when using Tet elements. This is a known issue with unknown cause.

The contact search tolerances are fixed for all meshes, with a zero tangential and normal tolerances.

### 4.1.1. Features Tested

Basic heat conduction, tied and resistance thermal contact between non-matching meshes (Hex-Hex, Tet-Tet, Hex-Tet).

### 4.1.2. Boundary Conditions

The interface between the two blocks is a thermal contact boundary condition. Both tied contact and resistance contact (with finite contact resistance) are tested. The left and right boundary conditions are prescribed using constant values. The remaining boundary conditions are adiabatic. A constant source term is applied in each block (with different signs).

### 4.1.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

#### 4.1.4. Verification of Solution

A manufactured solution is chosen based on the contact interface at  $x = 0$ :

$$T(x, y, z) = \begin{cases} \frac{1}{2}(1+x)(\gamma+x), & x < 0, \\ 1 + \frac{1}{2}(1-x)(-\gamma+x), & x > 0 \end{cases}$$

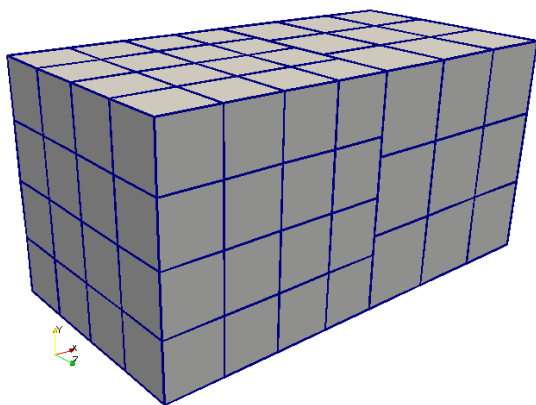
where  $\gamma = (2 - R)/(2 + R)$  is a constant depending on the thermal contact resistance  $R$ . Here  $R$  is the inverse of the contact conductance that is provided as a code input. In the case of tied contact,  $R = 0$  and therefore  $\gamma = 1$ . We note that when  $R > 0$ , this exact solution exhibits a jump in temperature across the contact interface.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

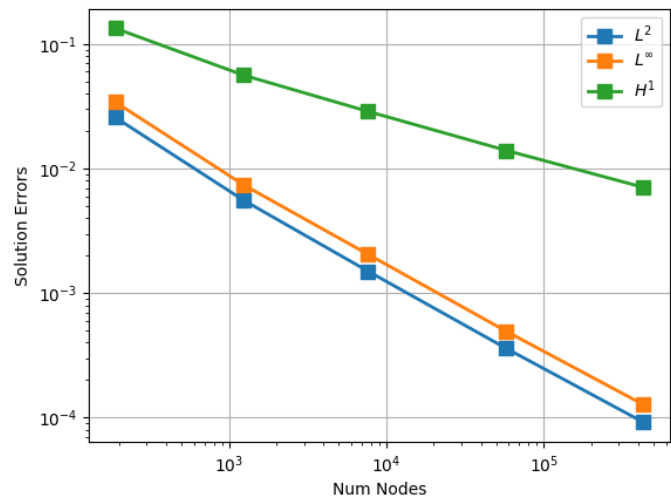
These rates are observed for the Hex-Hex case; however, both of the cases involving Tet meshes exhibit a reduced order of convergence in the  $L^\infty$  norms (convergence rate about 1.7).

For input decks see Appendix 12.3.1.

#### 4.1.5. Results: Hex8 Tied



Coarse Mesh



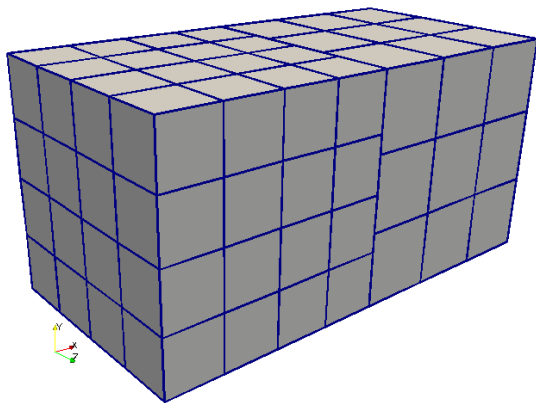
Error Norms

Figure 4.1-1.. 1D Flat Contact: Hex8 Tied

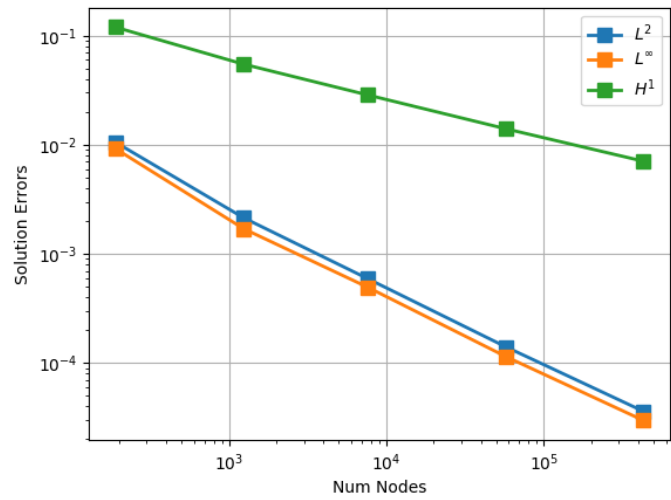
#### 4.1.6. Results: Hex8 Resistance

**Table 4.1-1.. 1D Flat Contact: Convergence Rates for Hex8 Tied**

Num Dofs	$L^2$	$L^\infty$	$H^1$
1241	2.45	2.44	1.38
7657	2.17	2.12	1.10
57890	2.11	2.11	1.07
432100	2.04	2.03	1.02



Coarse Mesh



Error Norms

**Figure 4.1-2.. 1D Flat Contact: Hex8 Resistance**

**Table 4.1-2.. 1D Flat Contact: Convergence Rates for Hex8 Resistance**

Num Dofs	$L^2$	$L^\infty$	$H^1$
1241	2.55	2.70	1.25
7657	2.12	2.04	1.07
57890	2.13	2.17	1.06
432100	2.03	2.01	1.02

### 4.1.7. Results: Tet4 Tied

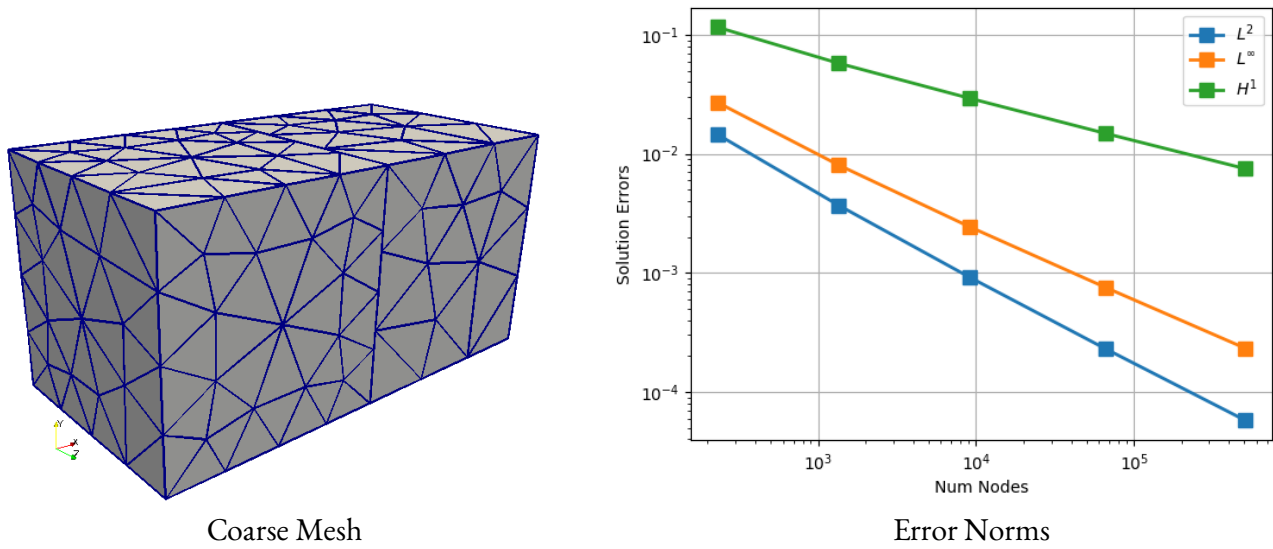
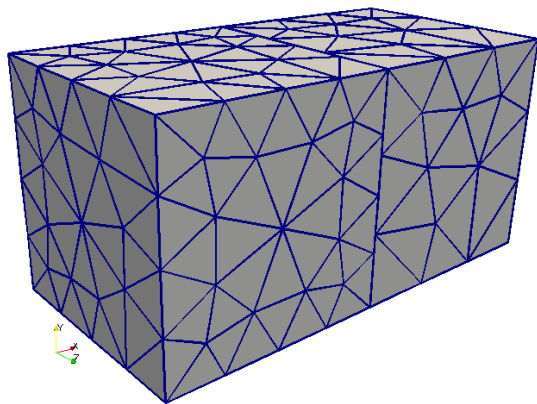


Figure 4.1-3.. 1D Flat Contact: Tet4 Tied

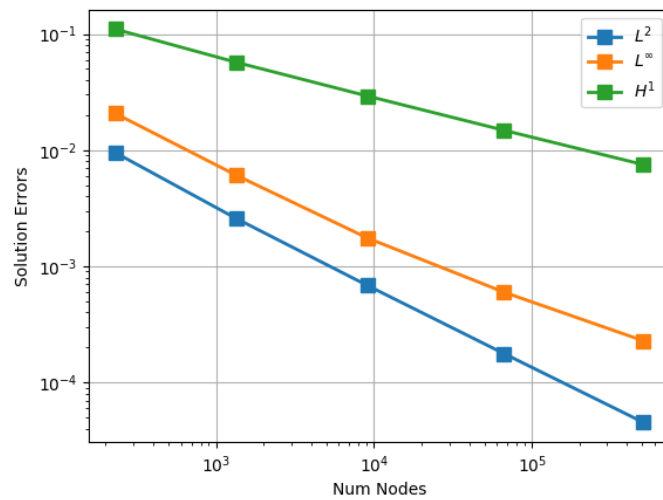
Table 4.1-3.. 1D Flat Contact: Convergence Rates for Tet4 Tied

Num Dofs	$L^2$	$L^\infty$	$H^1$
1348	2.33	2.06	1.18
9102	2.19	1.88	1.06
66620	2.09	1.78	1.03
509200	2.04	1.73	1.01

### 4.1.8. Results: Tet4 Resistance



Coarse Mesh



Error Norms

**Figure 4.1-4.. 1D Flat Contact: Tet4 Resistance**

**Table 4.1-4.. 1D Flat Contact: Convergence Rates for Tet4 Resistance**

Num Dofs	$L^2$	$L^\infty$	$H^1$
1348	2.22	2.09	1.12
9102	2.08	1.95	1.05
66620	2.04	1.62	1.02
509200	2.01	1.42	1.01

### 4.1.9. Results: Hex8-Tet4 Tied

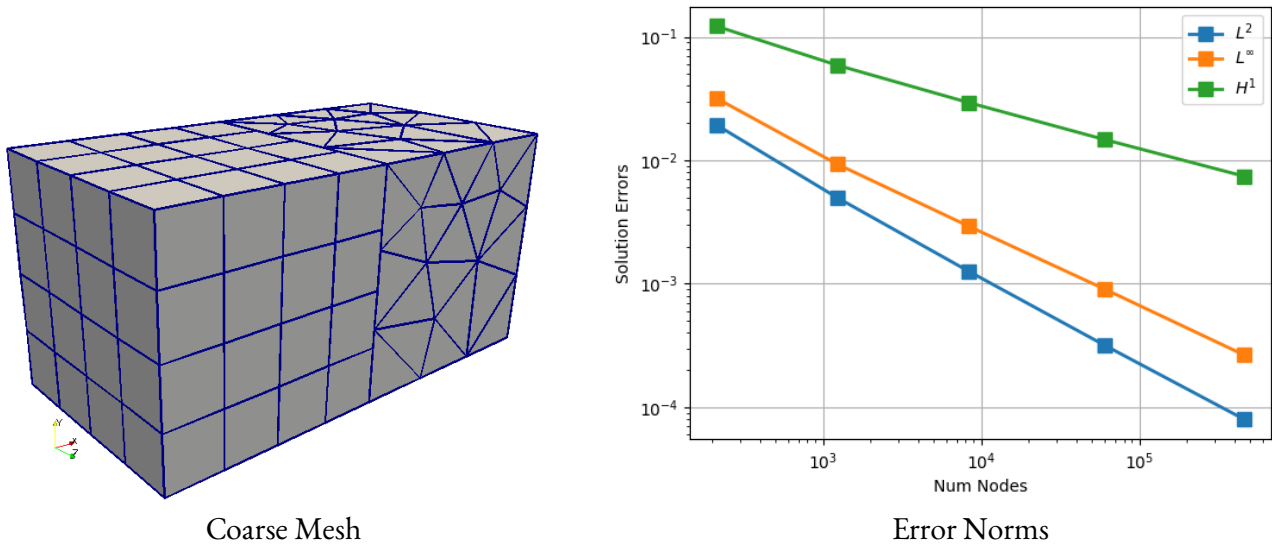


Figure 4.1-5.. 1D Flat Contact: Hex8-Tet4 Tied

Table 4.1-5.. 1D Flat Contact: Convergence Rates for Hex8-Tet4 Tied

Num Dofs	$L^2$	$L^\infty$	$H^1$
1231	2.32	2.10	1.24
8284	2.16	1.82	1.09
60570	2.09	1.78	1.04
462800	2.04	1.82	1.02

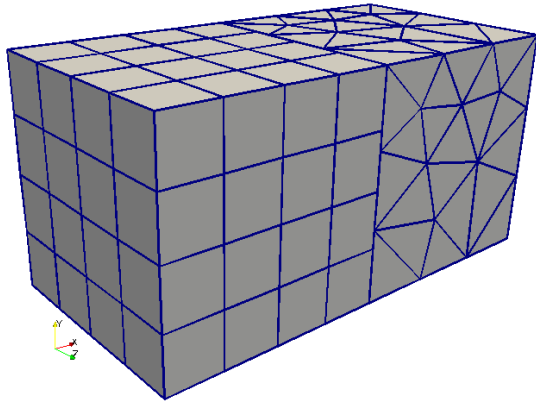
### 4.1.10. Results: Hex8-Tet4 Resistance

## 4.2. 3D CURVED CONTACT

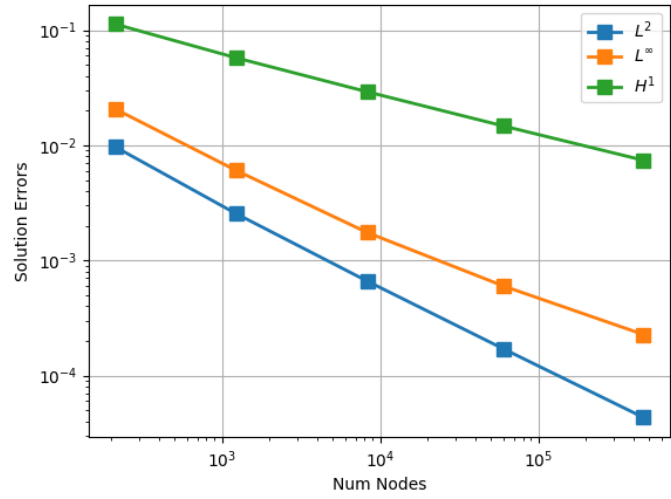
This problem tests thermal contact along a curved surface in 3D. The geometry consists of two thick spherical shells, which are in contact along a shared surface. The mesh nodes on either side of the contact surface are not aligned in general.

In this problem we observe sub-optimal convergence rates in the  $L^\infty$  norm when using tet elements. This is a known issue with unknown cause.

The contact search tolerances are fixed for all meshes, with a zero tangential tolerance and a normal tolerance large enough to insure a proper contact search on the coarsest mesh.



Coarse Mesh



Error Norms

**Figure 4.1-6.. 1D Flat Contact: Hex8-Tet4 Resistance**

**Table 4.1-6.. 1D Flat Contact: Convergence Rates for Hex8-Tet4 Resistance**

Num Dofs	$L^2$	$L^\infty$	$H^1$
1231	2.28	2.10	1.15
8284	2.12	1.95	1.07
60570	2.05	1.62	1.03
462800	2.02	1.43	1.01

### 4.2.1. Features Tested

Basic heat conduction, tied thermal contact between non-matching meshes (hex-hex, tet-tet, hex-tet).

### 4.2.2. Boundary Conditions

The interface between the two blocks is a tied thermal contact boundary condition. The outer and inner boundary conditions are prescribed at the nodes using the analytic solution.

### 4.2.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

## 4.2.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, z) = -3x^2z - 3y^2z + 2z^3$$

This solution is harmonic, implying that no source term is needed for the steady state heat equation with constant conductivity.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

These rates are observed for the hex-hex case; however, both of the cases involving tet meshes exhibit a reduced order of convergence in the  $L^\infty$  norms (convergence rate about 1.7).

For input decks see Appendix 12.3.2.

## 4.2.5. Results: Hex8-Hex8 Contact

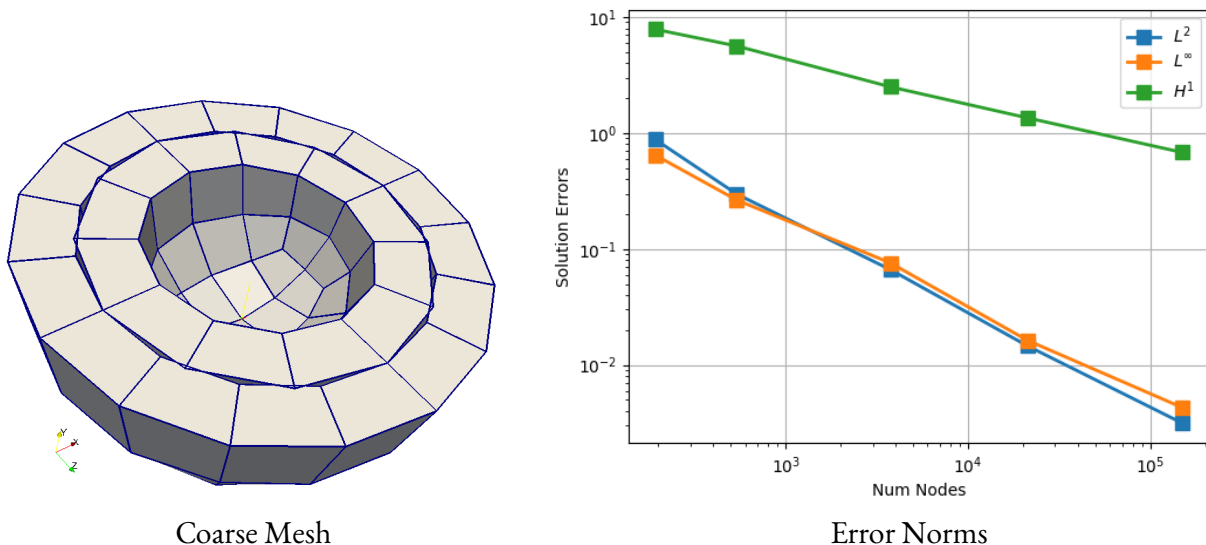
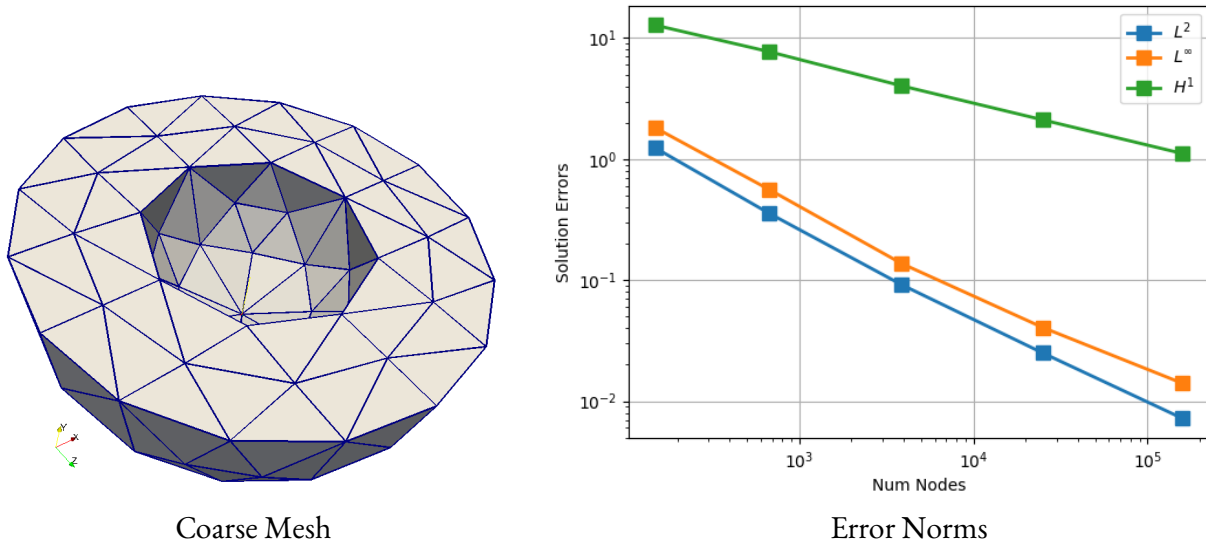


Figure 4.2-1.. 3D Curved Contact: Hex8-Hex8 Case

**Table 4.2-1.. 3D Curved Contact: Convergence Rates for Hex8-Hex8**

Num Dofs	$L^2$	$L^\infty$	$H^1$
540	3.16	2.60	0.96
3752	2.32	1.93	1.25
21220	2.62	2.66	1.07
150700	2.35	2.05	1.06

## 4.2.6. Results: Tet4-Tet4 Contact



**Figure 4.2-2.. 3D Curved Contact: Tet4-Tet4 Case**

**Table 4.2-2.. 3D Curved Contact: Convergence Rates for Tet4-Tet4**

Num Dofs	$L^2$	$L^\infty$	$H^1$
674	2.47	2.34	1.00
3881	2.33	2.41	1.11
25010	2.09	1.96	1.04
159100	2.02	1.73	1.04

## 4.2.7. Results: Hex8-Tet4 Contact

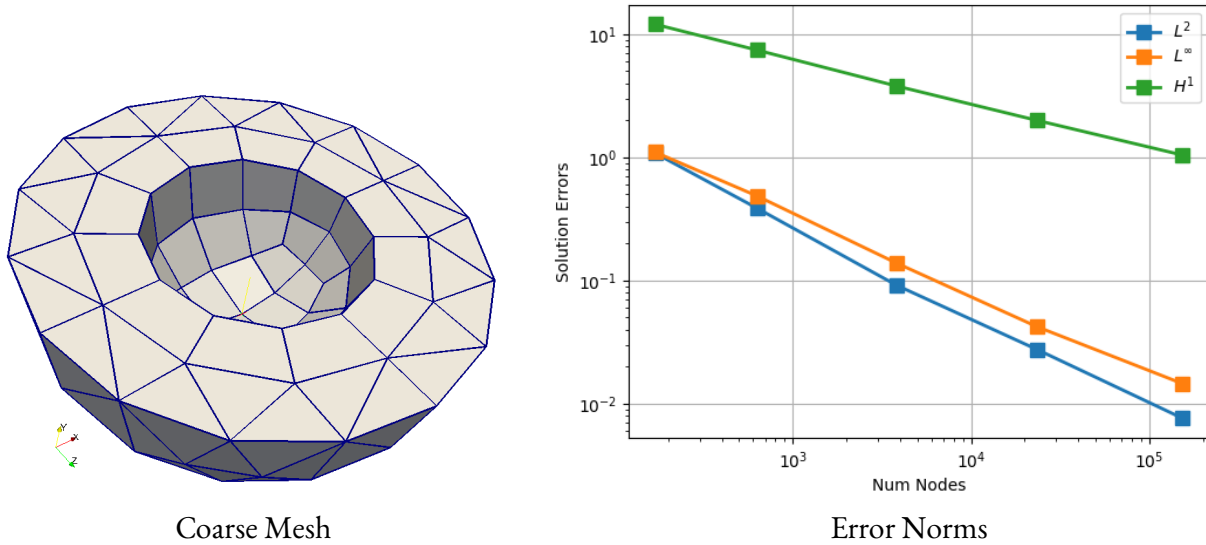


Figure 4.2-3.. 3D Curved Contact: Hex8-Tet4 Case

Table 4.2-3.. 3D Curved Contact: Convergence Rates for Hex8-Tet4

Num Dofs	$L^2$	$L^\infty$	$H^1$
630	2.34	1.87	1.11
3830	2.40	2.09	1.12
23420	1.98	1.97	1.06
153700	2.06	1.70	1.04

## 4.3. STEADY HEX8 CONTACT

This problem tests basic steady state heat conduction in a 3D domain. The geometry consists of a unit cube.

### 4.3.1. Features Tested

Basic heat conduction on Hex8 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.3.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

### 4.3.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 4.3.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, z) = 1 + (x - x^2)^2(y - y^2)^2(z - z^2)^2.$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

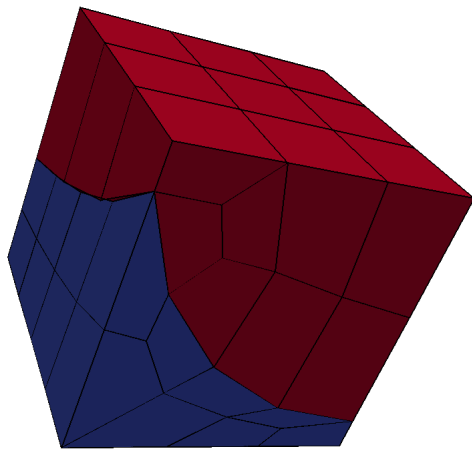
**Table 4.3-1.. Steady Tied Contact: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
192	0.86	-0.44	0.60
982	2.22	0.97	2.39
6419	2.31	1.07	2.28
46280	2.06	1.04	1.71
350600	1.95	1.02	2.08

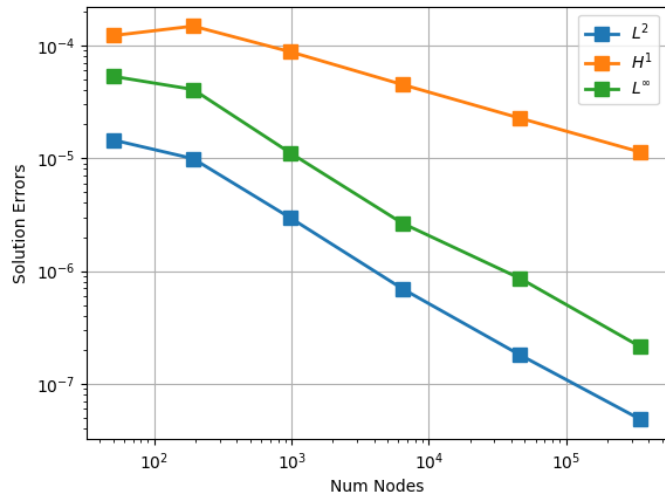
For input decks see Appendix [12.3.3](#).

## 4.4. STEADY HEX20 CONTACT

This problem tests basic steady state heat conduction in a 3D domain. The geometry consists of a unit cube. A variety of different source terms and boundary conditions are simultaneously applied. The exact solution is a manufactured solution.



Coarse Mesh



Error Norms

Figure 4.3-1.. Steady Tied Contact: Hex8 Meshes

#### 4.4.1. Features Tested

Basic heat conduction on Hex20 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

#### 4.4.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

#### 4.4.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

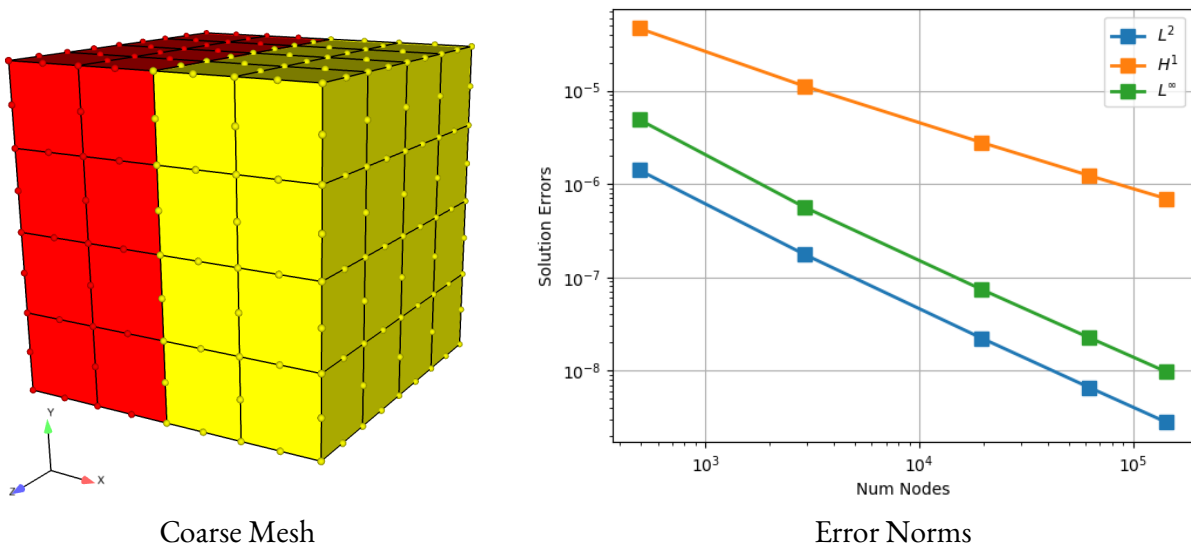
#### 4.4.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, z) = 1 + (x - x^2)^2(y - y^2)^2(z - z^2)^2.$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).



**Figure 4.4-1.. Steady Heat Conduction: Hex20 Meshes**

**Table 4.4-1.. Steady Heat Conduction: Convergence Rates for Hex20 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
2898	3.50	2.40	3.64
19620	3.25	2.19	3.19
62450	3.15	2.10	3.08
143700	3.10	2.07	3.04

For input decks see Appendix 12.3.4.

## 4.5. STEADY HEX27 CONTACT

This problem tests basic steady state heat conduction in a 3D domain. The geometry consists of a unit cube. A variety of different source terms and boundary conditions are simultaneously applied. The exact solution is a manufactured solution.

### 4.5.1. Features Tested

Basic heat conduction on Hex27 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.5.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine).

### 4.5.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 4.5.4. Verification of Solution

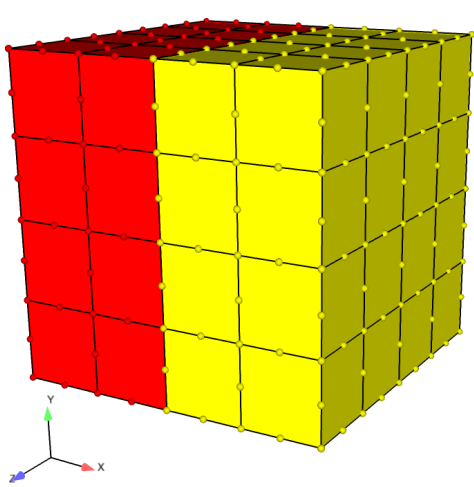
A manufactured solution is chosen as

$$T(x, y, z) = 1 + (x - x^2)^2(y - y^2)^2(z - z^2)^2.$$

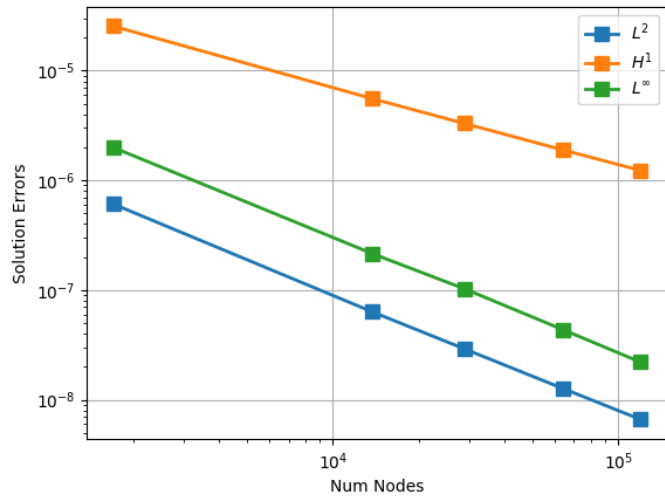
The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

For input decks see Appendix [12.3.5](#).



Coarse Mesh



Error Norms

**Figure 4.5-1.. Steady Heat Conduction: Hex27 Meshes**

**Table 4.5-1.. Steady Heat Conduction: Convergence Rates for Hex27 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
13750	3.25	2.18	3.19
28830	3.13	2.10	2.97
63880	3.14	2.09	3.25
120000	3.11	2.07	3.26

## 4.6. STEADY TET4 CONTACT

This problem is identical to the one in Section 2.1 except that unstructured Tet4 meshes are used instead.

### 4.6.1. Features Tested

Basic heat conduction on Tet4 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.6.2. Boundary Conditions

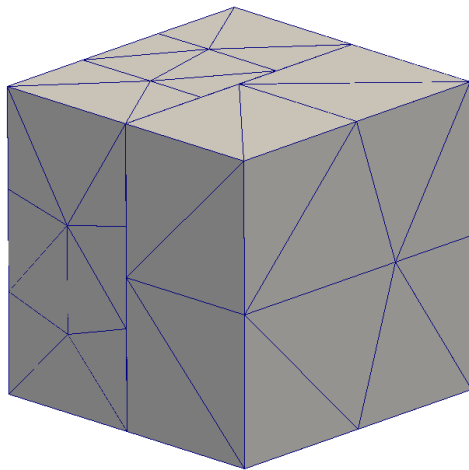
Same as in Section 2.1.

### 4.6.3. Material Parameters

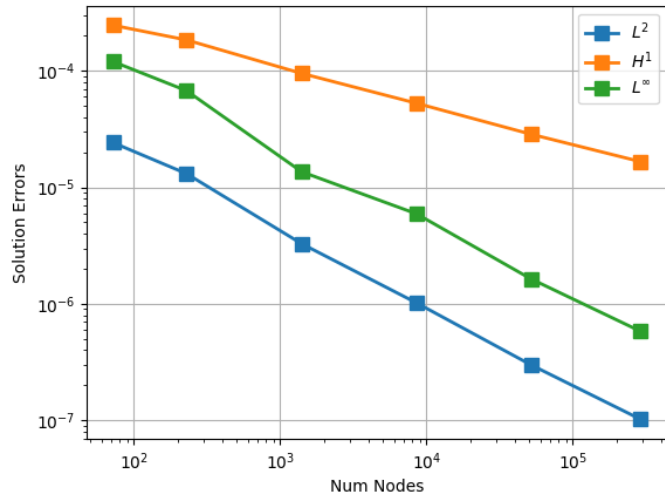
The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 4.6.4. Verification of Solution

Same as in Section 2.1.



Coarse Mesh



Error Norms

Figure 4.6-1.. Steady Tied Contact: Tet4 Meshes

Table 4.6-1.. Steady Tied Contact: Convergence Rates for Tet4 Meshes

Num Dofs	$L^2$	$H^1$	$L^\infty$
229	1.60	0.76	1.48
1402	2.29	1.09	2.65
8535	1.93	0.98	1.38
51620	2.05	1.02	2.14
291200	1.88	0.94	1.81

For input decks see Appendix 12.3.6.

## 4.7. STEADY TET4TET10 CONTACT

This problem is identical to the one in Section 2.1 except that unstructured Tet4 meshes are used instead.

### 4.7.1. Features Tested

Basic heat conduction on Tet4 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.7.2. Boundary Conditions

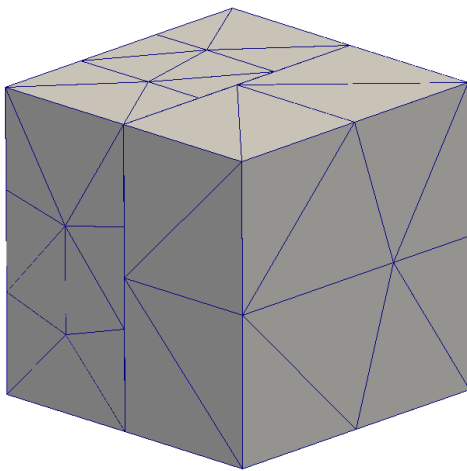
Same as in Section 2.1.

### 4.7.3. Material Parameters

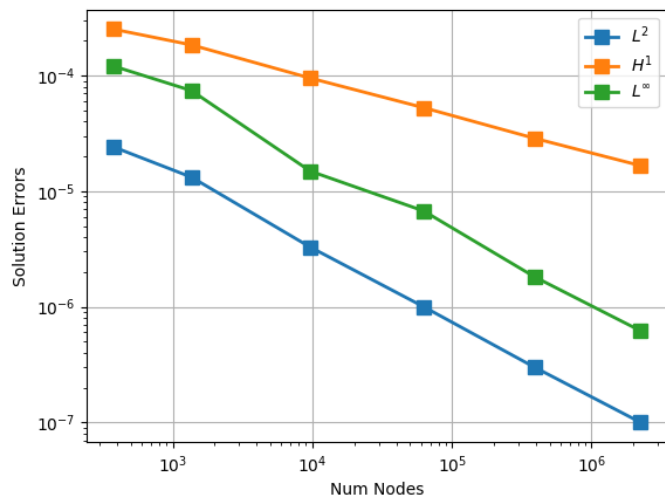
The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 4.7.4. Verification of Solution

Same as in Section 2.1.



Coarse Mesh



Error Norms

Figure 4.7-1.. Steady Tied Contact: Tet4 Meshes

For input decks see Appendix 12.3.7.

**Table 4.7-1.. Steady Tied Contact: Convergence Rates for Tet4 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
1364	1.40	0.72	1.13
9663	2.14	1.02	2.46
62720	1.90	0.94	1.27
392000	1.98	1.00	2.16
2250000	1.89	0.93	1.84

## 4.8. STEADY TET10 CONTACT

This problem is identical to the one in Section 2.1 except that unstructured Tet10 meshes are used instead.

### 4.8.1. Features Tested

Basic heat conduction on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.8.2. Boundary Conditions

Same as in Section 2.1.

### 4.8.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

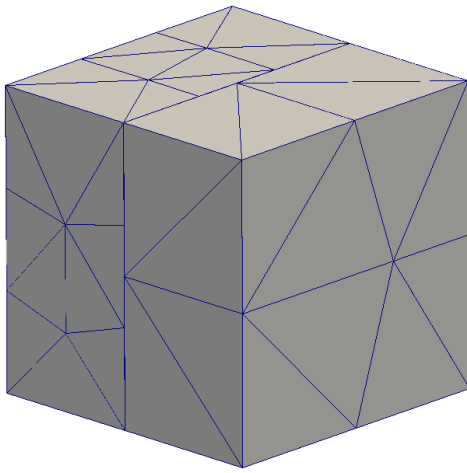
### 4.8.4. Verification of Solution

Same as in Section 2.1.

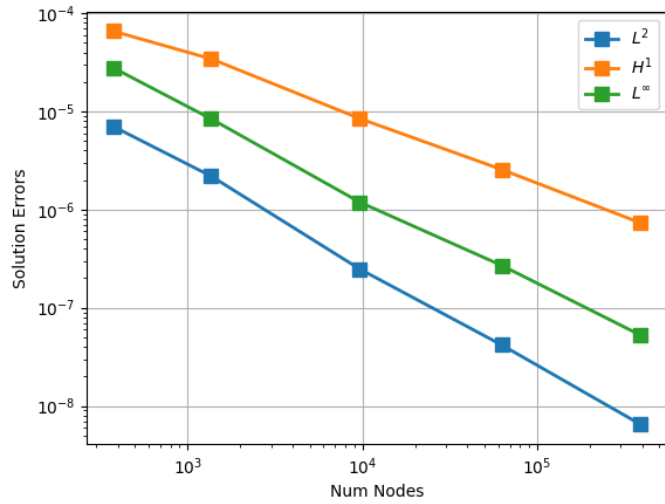
**Table 4.8-1.. Steady Tied Contact: Convergence Rates for Tet10 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
1364	2.71	1.51	2.78
9663	3.34	2.14	2.99
62720	2.86	1.92	2.39
392000	3.06	2.05	2.68

For input decks see Appendix 12.3.8.



Coarse Mesh



Error Norms

Figure 4.8-1.. Steady Tied Contact: Tet10 Meshes

## 4.9. STEADY TET10 DASH CONTACT

This problem is identical to the one in Section 2.1 except that unstructured Tet10 meshes are used instead.

### 4.9.1. Features Tested

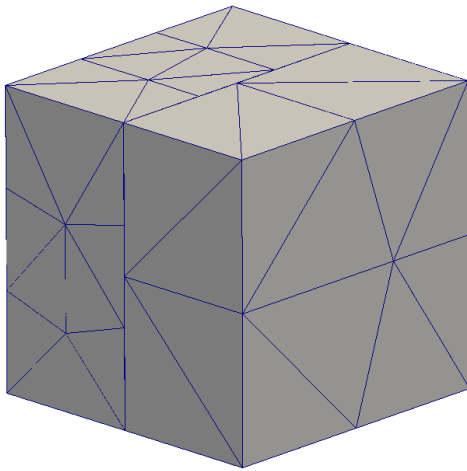
Basic heat conduction on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.9.2. Boundary Conditions

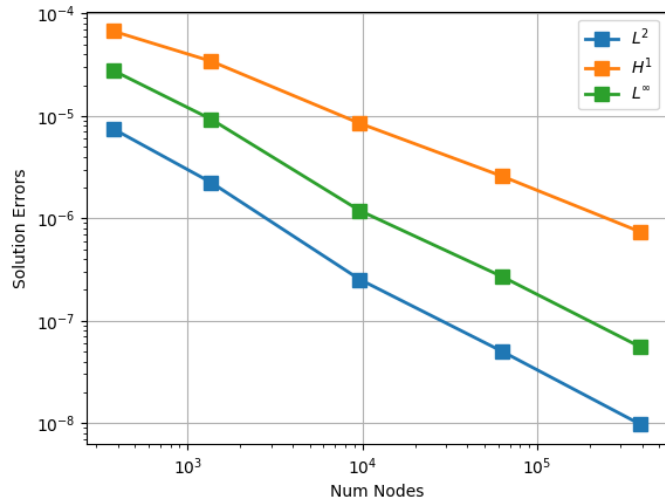
Same as in Section 2.1.

### 4.9.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.



Coarse Mesh



Error Norms

**Figure 4.9-1.. Steady Tied Dash Contact: Tet10 Meshes**

**Table 4.9-1.. Steady Tied DASH Contact: Convergence Rates for Tet10 Meshes**

Num Dofs	$L^2$	$H^1$	$L^\infty$
1364	2.80	1.57	2.55
9663	3.34	2.14	3.14
62720	2.59	1.91	2.38
392000	2.70	2.06	2.61

#### 4.9.4. Verification of Solution

Same as in Section 2.1.

For input decks see Appendix 12.3.9.

## 4.10. TRANSIENT TET4TET10 CONTACT

This problem tests basic transient heat conduction with contact in a 3D domain. The geometry consists of two halves of a unit cube meshed with Tet10 elements. The problem is solved using Tet4 interpolation and applying thermal contact at the common interface between the two domains.

### 4.10.1. Features Tested

Basic heat conduction on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

## 4.10.2. Boundary Conditions

Using a manufactured solution, Dirichlet boundary conditions are applied on all the non-contact exposed faces.

## 4.10.3. Material Parameters

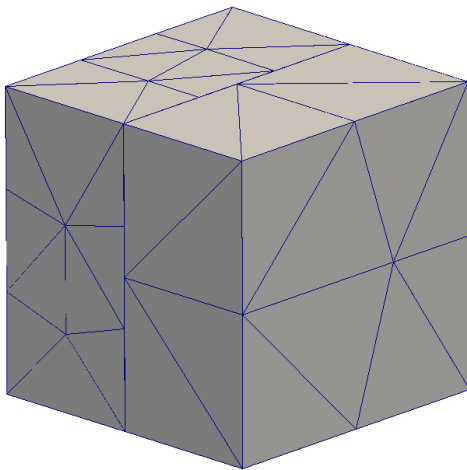
The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

## 4.10.4. Verification of Solution

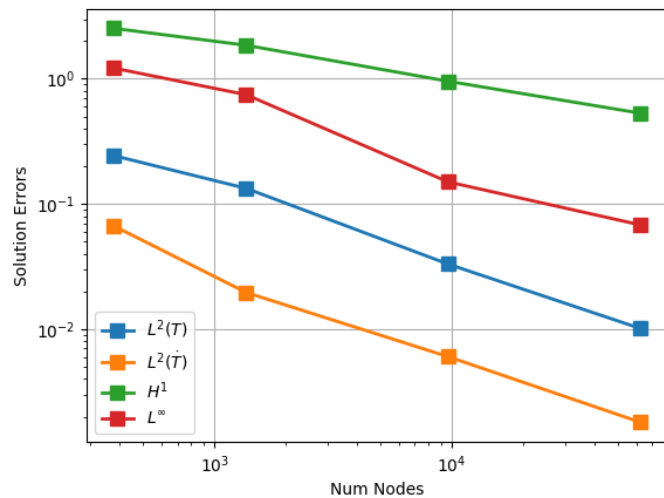
A manufactured solution is chosen as

$$T(x, y, z, t) = (x - x^2)^2 (y - y^2)^2 (z - z^2)^2 m(t) + 1,$$
$$m(t) = 10^4 [1. - \exp(-t) + t * \exp(-(t - 1.0) * (t - 1.0))];$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.



Coarse Mesh



Error Norms

**Figure 4.10-1.. Transient Tied Contact: Tet10 Meshes**

For input decks see Appendix [12.3.10](#).

**Table 4.10-1.. Transient Tied Contact: Convergence Rates for Tet10 Meshes**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
1364	1.41	2.85	0.72	1.14
9663	2.13	1.81	1.02	2.46
62720	1.90	1.94	0.94	1.27

## 4.11. TRANSIENT TET10 CONTACT

This problem tests basic transient heat conduction with contact in a 3D domain. The geometry consists of two halves of a unit cube meshed with Tet10 elements. The problem is solved by applying thermal contact at the common interface between the two domains.

### 4.11.1. Features Tested

Basic heat conduction on Tet10 meshes; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.11.2. Boundary Conditions

Using a manufactured solution, Dirichlet boundary conditions are applied on all the non-contact exposed faces.

### 4.11.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

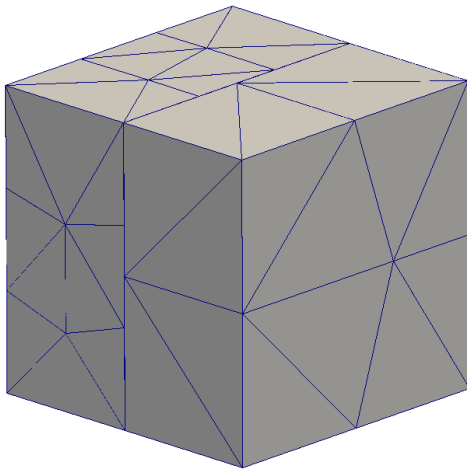
### 4.11.4. Verification of Solution

A manufactured solution is chosen as

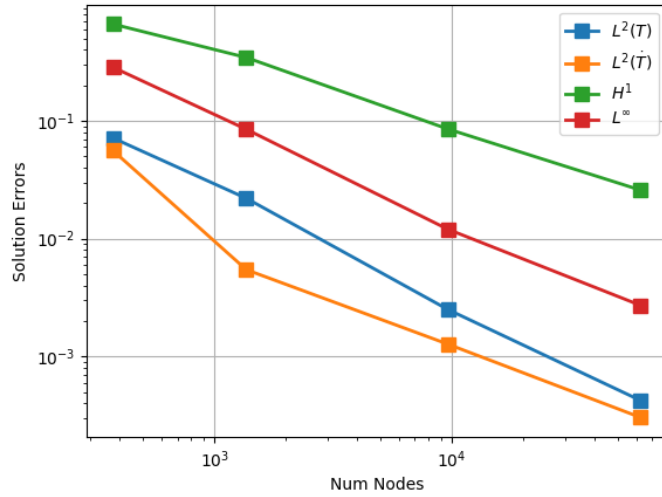
$$T(x, y, z, t) = (x - x^2)^2 (y - y^2)^2 (z - z^2)^2 m(t) + 1,$$
$$m(t) = 10^4 [1. - \exp(-t) + t * \exp(-(t - 1.0) * (t - 1.0))];$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For input decks see Appendix [12.3.11](#).



Coarse Mesh



Error Norms

**Figure 4.11-1.. Transient Tied Contact: Tet10 Meshes**

**Table 4.11-1.. Transient Tied Contact: Convergence Rates for Tet10 Meshes**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
1364	2.73	5.40	1.51	2.81
9663	3.33	2.23	2.14	3.00
62720	2.85	2.30	1.92	2.39

## 4.12. TRANSIENT HEX8 TIED CONTACT

This problem tests transient heat conduction on a 3D domains with a nonconformal mesh between two blocks. Tied temperature (generalized contact) is used for matching the energy equation between nonconformal blocks. The geometry consists of a unit cube.

### 4.12.1. Features Tested

Transient heat conduction on Hex8 meshes; dirichlet, heat flux, and convective flux boundary conditions, Tied Contact, Nonconformal; constant source terms; heat flux and source term from Encore user subroutines.

### 4.12.2. Boundary Conditions

At surfaces 4 and 6, the temperature is prescribed as a constant value. On surfaces 3 and 5, a heat flux condition is prescribed using a sum of a constant heat flux and a heat flux from an Encore function (user subroutine). On surfaces 1 and 2, heat flux condition is prescribed using a sum of a convective flux

boundary condition (with constant flux and convective coefficient) and a heat flux from an Encore function (user subroutine). Within the domain a source term is prescribed using a sum of a constant source and an Encore function (user subroutine). On the two interior surfaces connecting the nonconformal blocks (surfaces 7 and 8), a contact definition is defined as tied temperature.

### 4.12.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 4.12.4. Verification of Solution

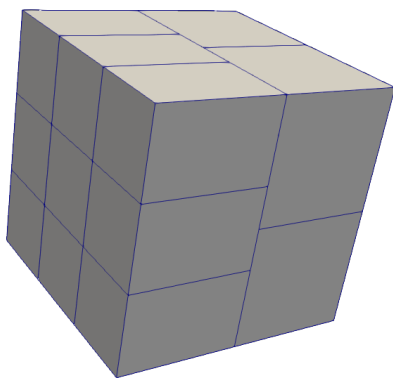
A manufactured solution is chosen as

$$T(x, y, z, t) = (x - x^2)^2 (y - y^2)^2 (z - z^2)^2 m(t) + 1,$$

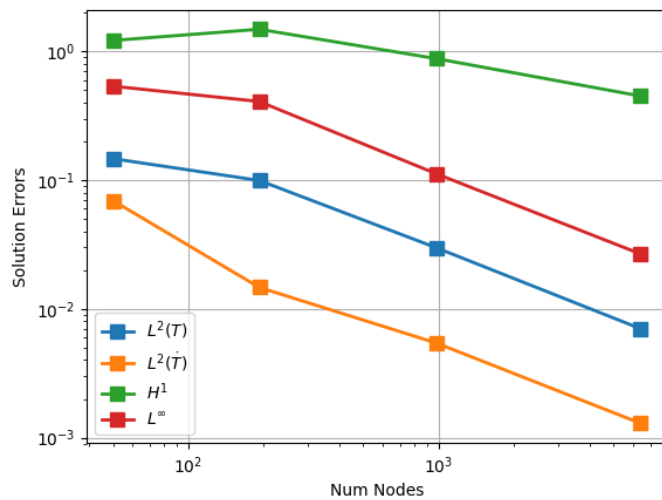
$$m(t) = 10^4 (1 - \exp(-t) + t \exp(-(t - 1)^2))$$

The source and heat flux user subroutines are chosen so that the solution satisfies the heat equation with the correct boundary conditions.

For each mesh, the errors in the temperature solution at final time are computed in the  $L^2$  norm of  $T$  and  $\dot{T}$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, 2 and 1, respectively (within a tolerance).



Coarse Mesh



Error Norms

Figure 4.12-1.. Tied Contact Transient Heat Conduction: Hex8 Meshes

**Table 4.12-1.. Tied Contact Transient Heat Conduction: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
192	0.88	3.46	-0.44	0.61
982	2.22	1.84	0.97	2.39
6419	2.31	2.30	1.07	2.29

## 4.13. TRANSIENT TET4 TIED CONTACT

This problem tests transient heat conduction and tied thermal contact in a 3D domain as in Section 2.7. The geometry consists of a unit cube that is split along the plane at  $x = 0.5$ .

### 4.13.1. Features Tested

Basic transient heat conduction on Tet4 meshes; non-conformal tied thermal contact; dirichlet, heat flux, and convective flux boundary conditions; constant source terms; heat flux and source term from Encore user subroutines.

### 4.13.2. Boundary Conditions

Identical to Section 2.7.

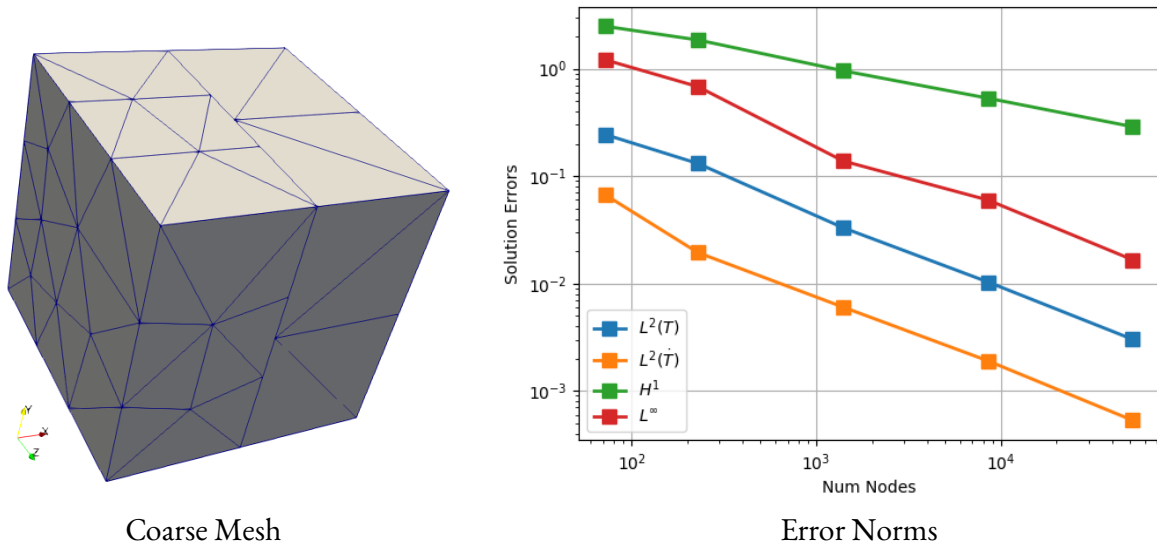
### 4.13.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 4.13.4. Verification of Solution

A manufactured solution is chosen as in Section 2.7.

For each mesh, the errors in the temperature solution at final time are computed in the  $L^2$  norm of  $T$  and  $\dot{T}$ ,  $L^\infty$  and  $H^1$  norms. We see convergence rates for  $\dot{T}$  that are slightly greater than two.



**Figure 4.13-1.. Transient Heat Conduction with Tied Contact: Tet4 Meshes**

**Table 4.13-1.. Transient Heat Conduction with Tied Contact: Convergence Rates for Tet4 Meshes**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
229	1.61	3.22	0.76	1.49
1402	2.29	1.94	1.10	2.64
8535	1.93	1.91	0.98	1.39
51620	2.05	2.12	1.02	2.14

# 5. ELEMENT DEATH

## 5.1. CDFEM ELEMENT DEATH (HEAT FLUX)

This problem tests transient conduction and CDFEM element death using 2D and 3D domains. The geometry consists of a thick 1/4 cylindrical or 1/8 spherical shell.

### 5.1.1. Features Tested

Transient heat conduction, adaptive second order time integration (BDF<sub>2</sub>), CDFEM element death, temperature and heat flux boundary conditions, Tri<sub>3</sub> and Tet<sub>4</sub> meshes.

### 5.1.2. Boundary Conditions

On one surface, the exact solution is used to specify a time-varying temperature. At the other surface, an analytic heat flux is applied using the exact solution. The erosion of the volume from CDFEM element death causes the surface with the heat flux BC to gradually recede as the material is removed.

### 5.1.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 5.1.4. Verification of Solution

A manufactured solution  $T$  and exact source term  $S$  are chosen in 2D to be:

$$T(r, t) = \frac{\ln(r)}{\ln(2-t)}, \quad S(r, t) = \frac{\ln(r)}{(\ln(2-t))^2(2-t)}$$

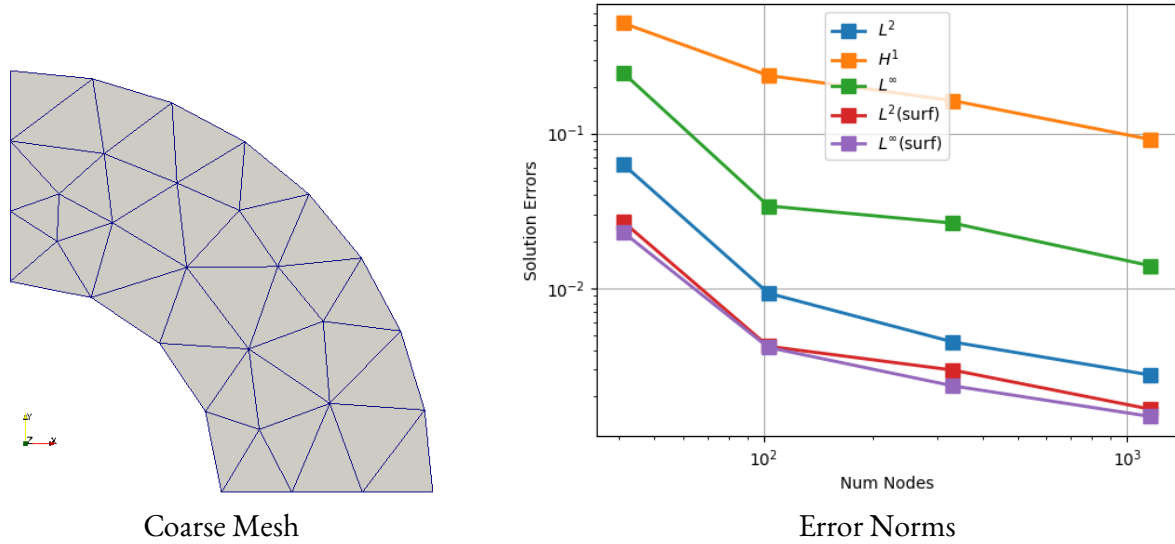
and in 3D to be:

$$T(r, t) = (1+t)/r, \quad S(r, t) = 1/r.$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms over the volume, and in the  $L^2$  and  $L^\infty$  norms over the outer surface. The test passes, only if the observed

rates of convergence in these norms are one (within a tolerance). First order convergence is expected in this case, due to the nature of the coupling of the CDFEM mesh decomposition and the heat conduction solve.

### 5.1.5. Results: Tri3



**Figure 5.1-1.. CDFEM Element Death (Heat Flux): Tri3**

**Table 5.1-1.. CDFEM Element Death (Heat Flux): Convergence Rates for Tri3**

Num Dofs	$L^2$	$H^1$	$L^\infty$	$L^2(\text{surf})$	$L^\infty(\text{surf})$
103	4.16	1.69	4.30	4.03	3.73
332	1.24	0.64	0.44	0.60	0.98
1163	0.78	0.93	1.01	0.93	0.73

### 5.1.6. Results: Tet4

**Table 5.1-2.. CDFEM Element Death (Heat Flux): Convergence Rates for Tet4**

Num Dofs	$L^2$	$H^1$	$L^\infty$	$L^2(\text{surf})$	$L^\infty(\text{surf})$
1024	0.63	0.39	0.76	0.60	0.83
5470	1.66	1.40	1.71	1.57	1.68
32588	1.27	1.19	1.34	1.27	1.34

For input decks see Appendix 12.4.1.

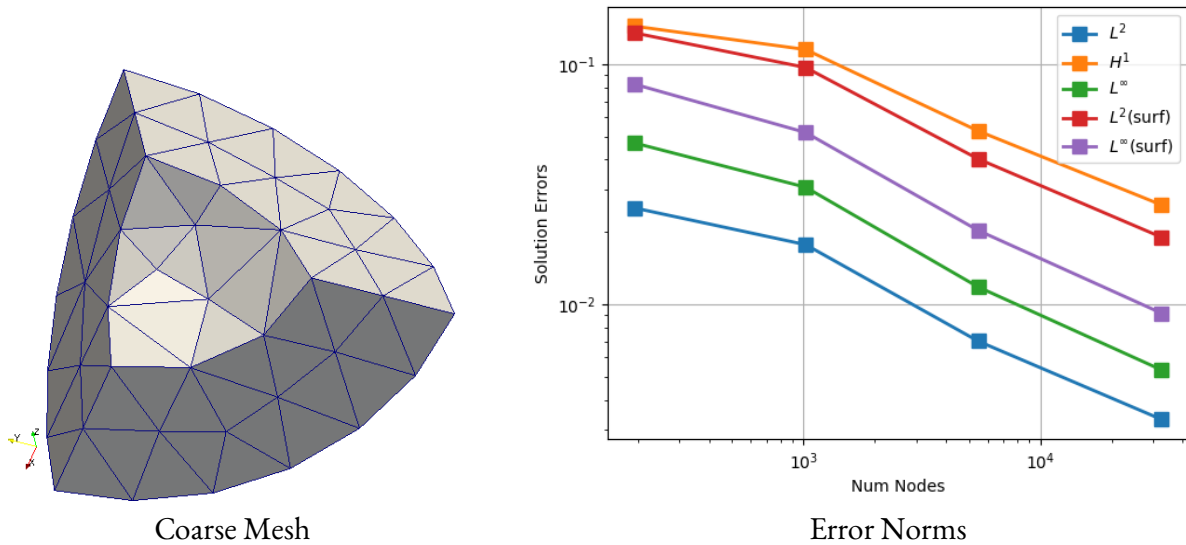


Figure 5.1-2.. CDFEM Element Death (Heat Flux): Tet4

## 5.2. 3D SPHERICAL SHELL ENCLOSURE

### 5.2.1. Problem Description

This problem tests transient conduction, enclosure radiation, and CDFEM element death. The initial geometry of this problem is a hollow sphere (block 2) inside and in contact with a second hollow sphere (block 1). The geometry is such that the solution maintains radial symmetry. The inner sphere decomposes at a specific failure temperature, resulting in a changing enclosure geometry.

### 5.2.2. Features Tested

Transient heat conduction, enclosure radiation, CDFEM element death, Tet4 meshes.

### 5.2.3. Boundary and Initial Conditions

The initial condition is a piecewise steady state temperature distribution defined below in (5.1). The boundary conditions specify the temperature  $T_4$  at the outer surface (4) of the outer sphere and  $T_1$  at the inner surface (1) of the inner sphere. The inner temperature  $T_1$  will be gradually increased, while  $T_4$  remains constant in time.

An enclosure is defined initially using the outer surface of the inner volume (surface 2 of block 2) and the inner surface of the outer volume (surface 3 of block 1). The erosion of the inner volume (block 2) from CDFEM element death causes surface 2 to gradually recede as the material within block 2 is removed.

Dimensions are defined in Table 5.2-1.

**Table 5.2-1.. Dimensions of problem**

radius of surface_1	$r_1$	0.01
radius of surface_2	$r_2$	0.02
radius of surface_3	$r_3$	0.03
radius of surface_4	$r_4$	0.04

## 5.2.4. Material Parameters

Material properties are shown in Table 5.2-2.

**Table 5.2-2.. Material properties**

Thermal conductivity	$\kappa$	1.0
Density	$\rho$	7682.0
Specific heat	$C_p$	10.0
emissivity (inner)	$\epsilon_2$	0.6
emissivity (outer)	$\epsilon_3$	0.7
Stefan-Boltzmann constant	$\sigma$	5.6704e-8
failure temperature (block 2)	$T_c$	867.011674920813

## 5.2.5. Verification of Solution

The solution after failure occurs is specified using inner and outer temperature solutions of the form:

$$T_i(r) \equiv T_1 + (T_c - T_1) \frac{1/r - 1/r_1}{1/r_2 - 1/r_1}, \quad r_1 \leq r \leq r_2, \quad (5.1)$$

$$T_o(r) \equiv C_o + (T_4 - C_o) \frac{1/r - 1/r_3}{1/r_4 - 1/r_3}, \quad r_3 \leq r \leq r_4 \quad (5.2)$$

Here all parameters are known except  $r_2$  and  $C_o$ , which will vary with time. The initial value of  $r_2$  is given in Table 5.2-1; the initial value of  $C_o$  is chosen to satisfy the enclosure radiation equilibrium equations below.

To complete the solution, we now derive a system of two nonlinear equations to solve for  $r_2$  and  $C_o$ . These are the energy balances on the outer and inner enclosure surfaces, given by

$$R_2 \equiv q_2 - \sigma \epsilon_2 T_2^4 + \epsilon_2 (F_{22} J_2 + F_{23} J_3) \quad (5.3)$$

$$R_3 \equiv -q_3 - \sigma \epsilon_3 T_3^4 + \epsilon_3 (F_{32} J_2 + F_{33} J_3) \quad (5.4)$$

where the three terms in each equation represent fluxes from conduction, radiative emission, and radiative reflection. The conductive fluxes are defined by Fourier's law as

$$q_2 = -\kappa_2 \frac{\partial T_i}{\partial r} \Big|_{r=r_2} = \kappa_2 \frac{T_c - T_1}{r_1^2(1/r_2 - 1/r_1)} \quad (5.5)$$

$$q_3 = -\kappa_3 \frac{\partial T_o}{\partial r} \Big|_{r=r_3} = \kappa_3 \frac{T_4 - C_o}{r_3^2(1/r_4 - 1/r_3)} \quad (5.6)$$

The surface temperatures are

$$T_2 \equiv T_i|_{r=r_2} = T_c, \quad T_3 \equiv T_o|_{r=r_3} = C_o$$

The radiosities are obtained by solving the linear system for enclosure radiation

$$\begin{bmatrix} 1 - (1 - \epsilon_2)F_{22} & -(1 - \epsilon_2)F_{23} \\ -(1 - \epsilon_3)F_{32} & 1 - (1 - \epsilon_3)F_{33} \end{bmatrix} \begin{bmatrix} J_2 \\ J_3 \end{bmatrix} = \begin{bmatrix} \sigma \epsilon_2 T_2^4 \\ \sigma \epsilon_3 T_3^4 \end{bmatrix}$$

to obtain

$$\begin{bmatrix} J_2 \\ J_3 \end{bmatrix} = \frac{1}{a} \begin{bmatrix} 1 - (1 - \epsilon_3)F_{33} & (1 - \epsilon_2)F_{23} \\ (1 - \epsilon_3)F_{32} & 1 - (1 - \epsilon_2)F_{22} \end{bmatrix} \begin{bmatrix} \sigma \epsilon_2 T_2^4 \\ \sigma \epsilon_3 T_3^4 \end{bmatrix}$$

where  $a$  is the determinant

$$a = (1 - (1 - \epsilon_2)F_{22})(1 - (1 - \epsilon_3)F_{33}) - (1 - \epsilon_2)F_{23}(1 - \epsilon_3)F_{32}$$

The viewfactor coefficients  $F_{ij}$  are given by

$$F_{22} = 0, \quad F_{23} = 1, \quad F_{32} = (r_2/r_3)^2, \quad F_{33} = 1 - F_{32}$$

The specific function we choose for  $T_1(t)$  is

$$T_1(t) \equiv T_1 + 400(1 - \cos(\pi t))/2$$

The time histories of  $r_2$  and  $C_o$  are shown in Figure 5.2-1.

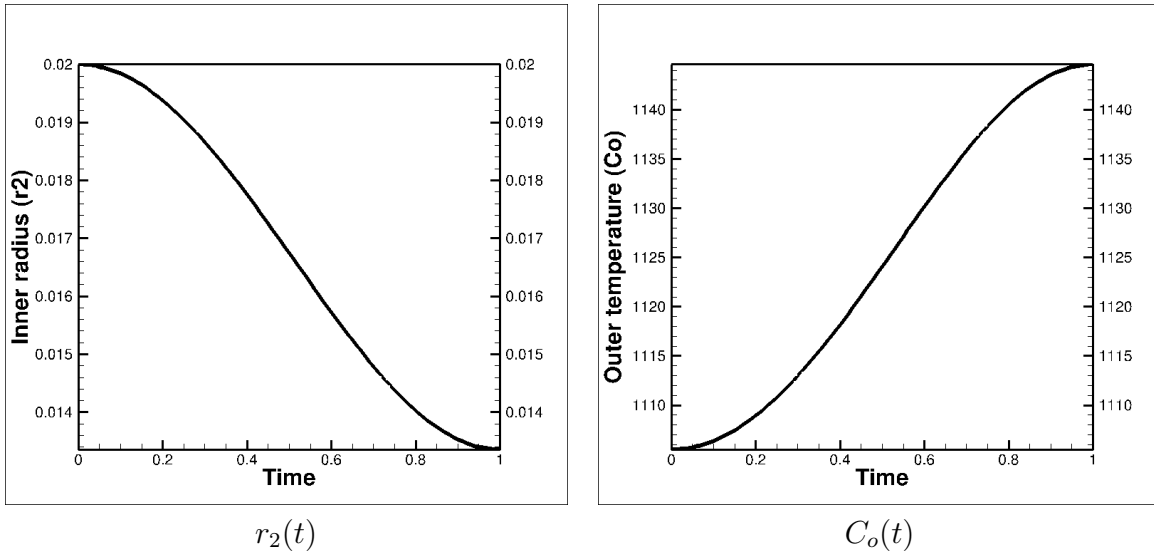
In order to derive the source term, the time derivatives of  $r_2$  and  $C_o$  are computed once the pair of nonlinear equations is solved using Newton's method. Since the spatial part of the piecewise solution is harmonic, the source terms become just  $\rho c_p \partial_t T$ , where

$$\partial_t T_i \equiv \dot{T}_1 + (T_c - \dot{T}_1) \frac{1/r - 1/r_1}{1/r_2 - 1/r_1} + (T_c - T_1) \frac{\dot{r}_2(1/r - 1/r_1)}{r_2^2(1/r_2 - 1/r_1)^2}, \quad r_1 \leq r \leq r_2, \quad (5.7)$$

$$\partial_t T_o \equiv \dot{C}_o \left(1 - \frac{1/r - 1/r_3}{1/r_4 - 1/r_3}\right), \quad r_3 \leq r \leq r_4 \quad (5.8)$$

## 5.2.6. Results

Results are presented running the problem on three meshes up to time  $t = 0.9$ .



**Figure 5.2-1.. Evolution of parameters  $r_2$  and  $C_o$ .**

**Table 5.2-3.. Convergence Rates at  $t = 0.9$**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$L^\infty$	$H^1$
4307	2.11	2.09	0.83	1.03
25590	2.25	1.52	2.15	1.05

For input decks see Appendix [12.4.2](#).

### 5.3. STANDARD ELEMENT DEATH (HEAT FLUX)

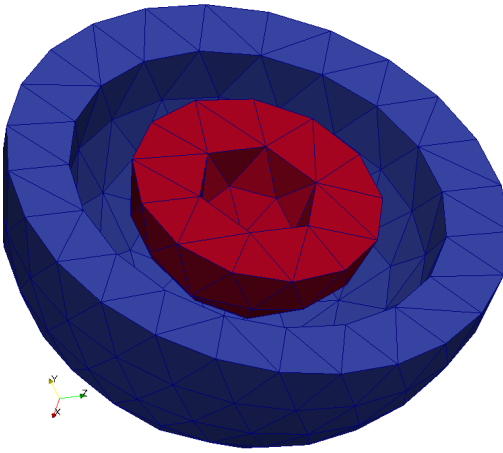
This problem tests transient conduction with standard element death on a 2D square domain than is essentially a 1D problem.

#### 5.3.1. Features Tested

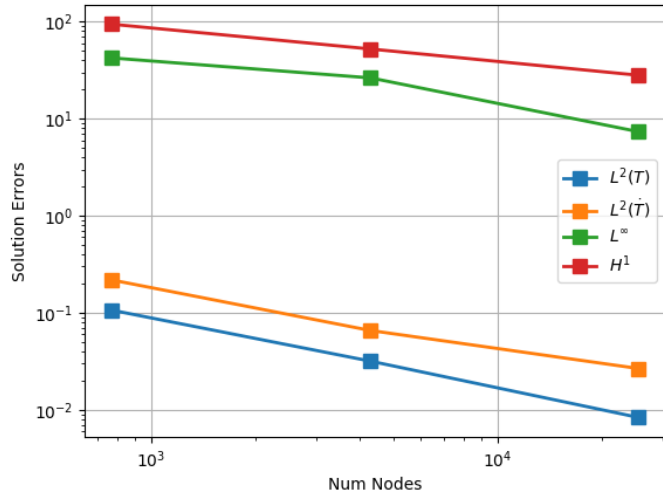
Transient heat conduction, adaptive second order time integration (BDF2), standard element death, temperature and heat flux boundary conditions, Tri3, Hex8 and Quad4 meshes.

#### 5.3.2. Boundary Conditions

On one surface, the exact solution is used to specify a time-varying temperature. At the other surface, an analytic heat flux is applied using the exact solution. The erosion of the volume from element death causes the surface with the heat flux BC to recede element by element as the material is removed.



Coarse Mesh



Error Norms ( $t = 0.9$ )

### 5.3.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 5.3.4. Verification of Solution

A manufactured solution  $T$  and exact source term  $S$  are chosen to be:

$$T(r, t) = \exp(t - x).$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms over the area. The test passes, only if the observed rates of convergence in these norms are one (within a tolerance). First order convergence is expected in this case.

### 5.3.5. Results: 1D Hex8

Table 5.3-1.. Element Death (Heat Flux): Convergence Rates for Hex8

Num Dofs	Var1	Var2	Var3
40	nan	nan	nan
72	1.69	1.59	1.62
144	0.93	0.96	0.93
272	1.21	1.14	1.17

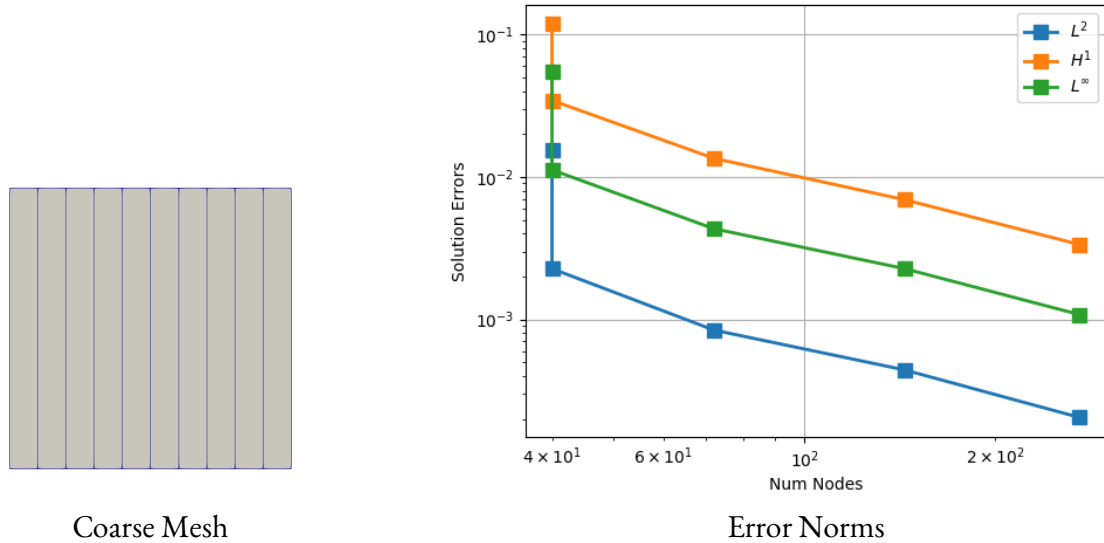


Figure 5.3-1.. Element Death (Heat Flux): Hex8

### 5.3.6. Results: 1D Quad4

Table 5.3-2.. Element Death (Heat Flux): Convergence Rates for Quad4

Num Dofs	Var1	Var2	Var3
22	1.38	1.38	1.34
36	1.85	1.52	1.66
68	1.16	1.12	1.13
134	1.09	1.09	1.09

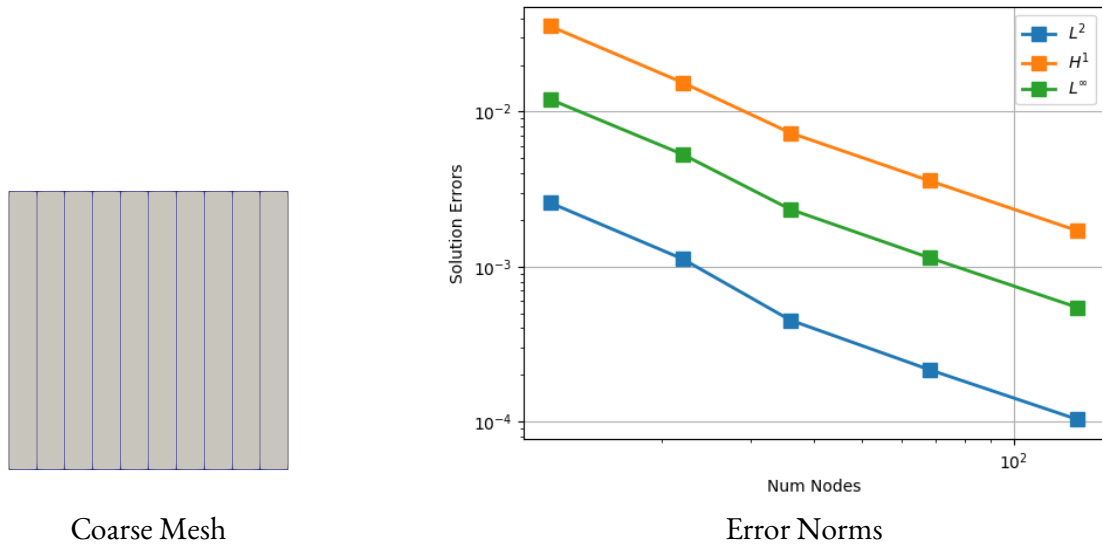
### 5.3.7. Results: 1D Tri3

Table 5.3-3.. Element Death (Heat Flux): Convergence Rates for Tri3

Num Dofs	Var1	Var2	Var3
20	nan	nan	nan
38	1.43	1.44	1.40
73	1.09	1.04	1.04
138	1.19	1.12	1.17

### 5.3.8. Results: 2D Quad4

This problem tests transient conduction with standard element death on a 2D quarter slice of an annulus.



**Figure 5.3-2.. Element Death (Heat Flux): Quad4**

### 5.3.9. Features Tested

Transient heat conduction, fixed first order time integration, standard element death, Quad4 mesh.

### 5.3.10. Boundary Conditions

On one surface, the exact solution is used to specify a time-varying temperature. On the other surfaces, the exact source solution is provided as the flux boundary condition. The erosion of the volume from element death is caused by having a minimum nodal value of temperature less than 1.

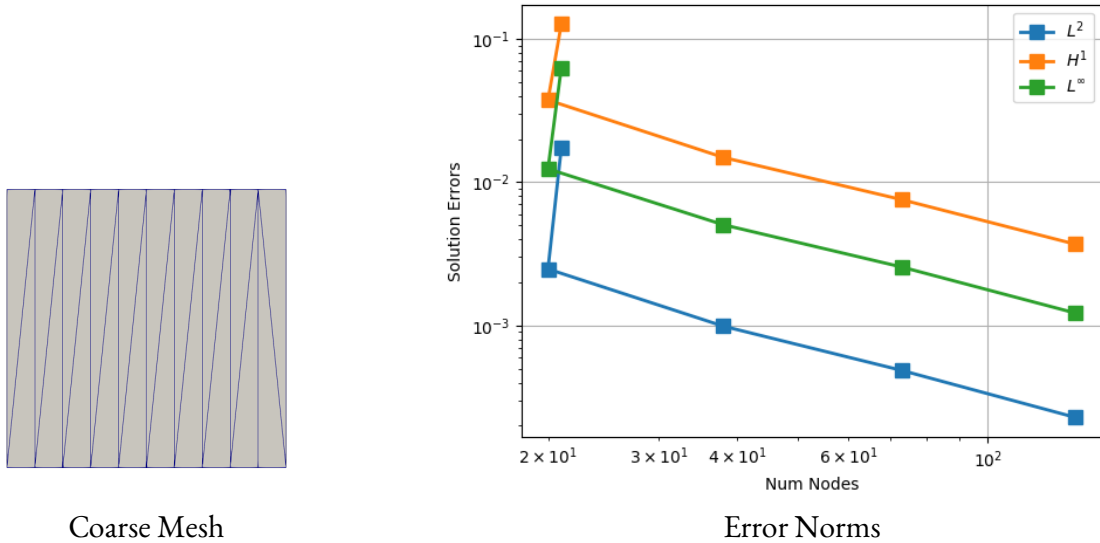
### 5.3.11. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 5.3.12. Verification of Solution

A manufactured solution  $T$  and exact source term  $S$  are chosen to be:

$$T(r, t) = \ln(\sqrt{x^2 + y^2})(1/\ln(2 - t)).$$



**Figure 5.3-3.. Element Death (Heat Flux): Tri3**

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms over the area. The test passes, only if the observed rates of convergence in these norms are one (within a tolerance). First order convergence is expected in this case.

**Table 5.3-4.. 2D Element Death (Heat Flux): Convergence Rates for Quad4**

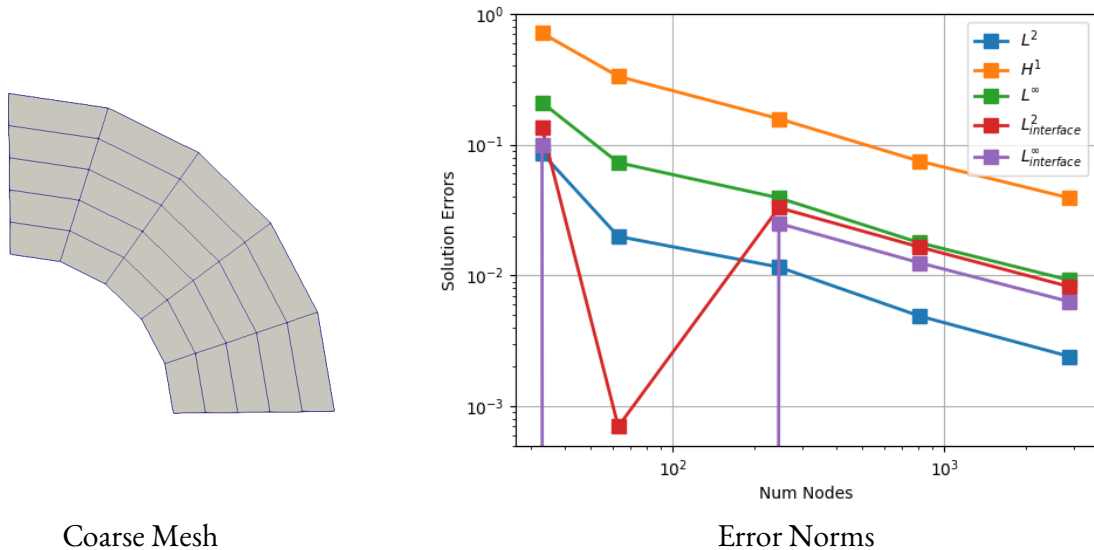
Num Dofs	Var1	Var2	Var3	Var4	Var5
63	4.53	2.35	3.28	16.25	nan
246	0.79	1.10	0.91	-5.66	nan
810	1.45	1.25	1.32	1.16	1.16
2898	1.12	1.02	1.04	1.10	1.08

### 5.3.13. Results: 3D Hex8

This problem evaluates transient conduction with standard element death on a 3D quarter of a hollow sphere geometry.

### 5.3.14. Features Tested

Transient heat conduction, fixed first order time integration, standard element death, Hex8 mesh.



**Figure 5.3-4.. Element Death (Heat Flux): Quad4**

### 5.3.15. Boundary Conditions

On surface 2, the exact solution is used to specify a time-varying temperature. On all remaining surfaces, a heat flux boundary condition is imposed with a flux time function specified. The erosion of the volume from element death is caused by having a maximum nodal value of temperature greater than  $I$ .

### 5.3.16. Material Parameters

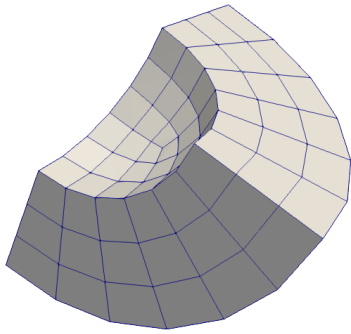
The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks.

### 5.3.17. Verification of Solution

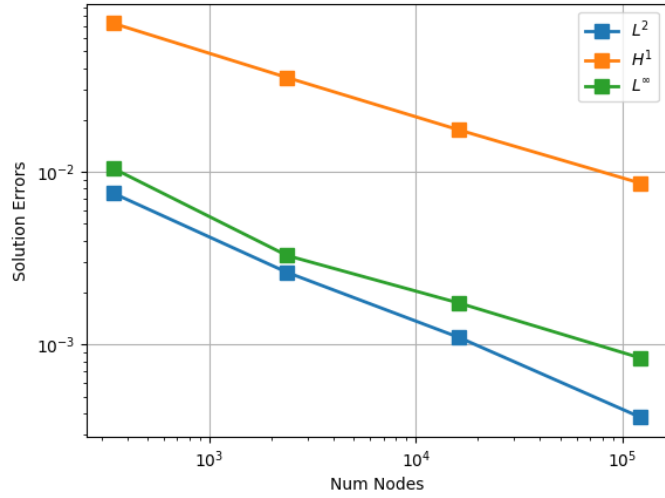
A manufactured solution  $T$  and exact source term  $S$  are chosen to be:

$$T(r, t) = \frac{1 + t}{\sqrt{x^2 + y^2 + z^2}}.$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms over the area. The observed rates of convergence in these norms are one (within a tolerance). First order convergence is expected in this case.



Coarse Mesh



Error Norms

**Figure 5.3-5.. Element Death (Heat Flux): Hex8**

**Table 5.3-5.. Element Death (Heat Flux): Convergence Rates for Hex8**

Num Dofs	$L^2$	$H^1$	$L^\infty$
2382	1.64	1.12	1.80
16214	1.36	1.10	1.00
122892	1.57	1.05	1.08

# 6. TIME INTEGRATION

## 6.1. ADAPTIVE TIME INTEGRATION

This problem tests the various implicit time integrators using both fixed and adaptive time stepping. The integrators are first order (Backward Euler), second order (Crank-Nicolson) and BDF2. The geometry is a 2D square.

### 6.1.1. Features Tested

Transient heat conduction, time integrators, adaptive time stepping, polynomial temperature dependence of density and thermal conductivity.

### 6.1.2. Boundary Conditions

The boundary conditions are prescribed at the nodes using the analytic solution. The initial condition is specified using an Encore function evaluated at the nodes.

### 6.1.3. Material Parameters

The specific heat is constant. The density and thermal conductivity are linear polynomials in the temperature.

### 6.1.4. Verification of Solution

A manufactured solution is chosen as

$$T(x, y, t) = \sin(C_1 t) + 2x \cos(C_2 t) + 3y \sin(C_3 t) + 4xy \cos(C_4 t) + 5x^2 \sin(C_5 t) + 6y^2 \cos(C_6 t)$$

which requires a source term. This solution is designed to have a non-trivial time-dependence using constants:

$$C_1 = \pi, \quad C_2 = 2\pi, \quad C_3 = 3\pi, \quad C_4 = \pi, \quad C_5 = 2.5\pi, \quad C_6 = 0.5\pi$$

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The  $L^2$  error in the temperature time derivative is also computed. The test passes, only if the observed rates of convergence in these norms are 1 for  $H^1$  and 2 for all other norms (within a tolerance).

Because the adaptive meshes use less time steps, we use time step size instead of mesh size for estimation of the convergence rates. We also include the  $L^2$  error in the time derivative of the temperature.

For input decks see Appendix [12.5.1](#).

### 6.1.5. Results: First Order Fixed

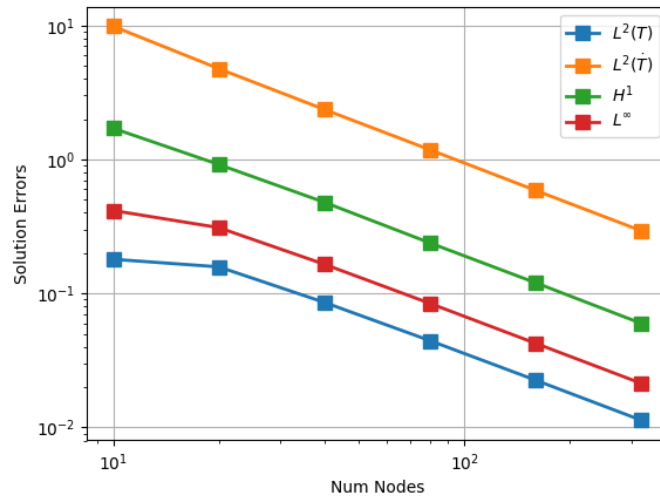


Figure 6.1-1.. Adaptive Time Integration: Errors for First Order Fixed

Table 6.1-1.. Adaptive Time Integration: Convergence Rates for First Order Fixed

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
20	0.18	1.05	0.89	0.43
40	0.89	1.01	0.94	0.91
80	0.95	1.01	1.01	0.97
160	0.98	1.00	0.99	0.99
320	0.99	1.00	1.00	0.99

### 6.1.6. Results: First Order Adaptive

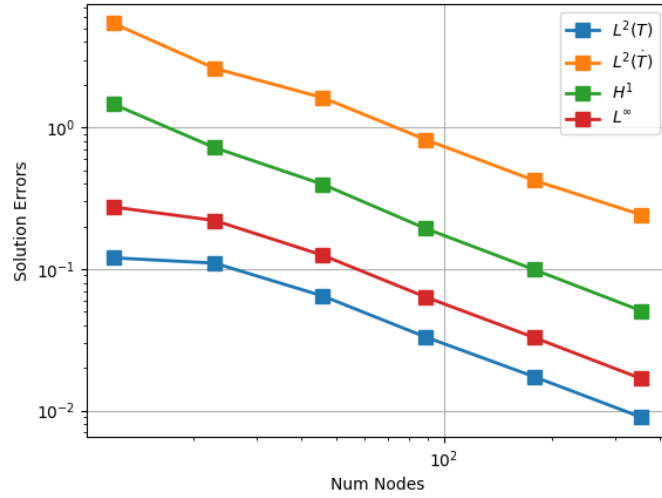


Figure 6.1-2.. Adaptive Time Integration: Errors for First Order Adaptive

Table 6.1-2.. Adaptive Time Integration: Convergence Rates for First Order Adaptive

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
23	0.13	1.13	1.09	0.34
46	0.77	0.68	0.86	0.81
89	1.01	1.04	1.09	1.03
178	0.93	0.95	0.96	0.95
355	0.96	0.81	0.98	0.97

### 6.1.7. Results: Second Order Fixed

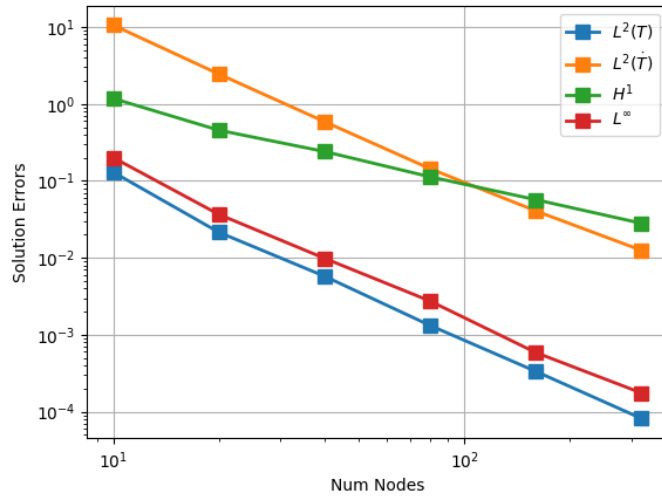
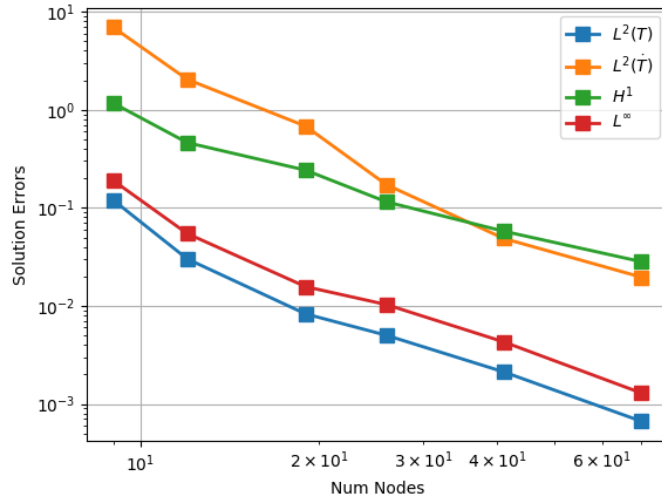


Figure 6.1-3.. Adaptive Time Integration: Errors for Second Order Fixed

Table 6.1-3.. Adaptive Time Integration: Convergence Rates for Second Order Fixed

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
20	2.59	2.11	1.37	2.46
40	1.90	2.06	0.92	1.90
80	2.13	2.03	1.10	1.85
160	1.98	1.82	0.99	2.22
320	2.03	1.71	1.02	1.75

### 6.1.8. Results: Second Order Adaptive



**Figure 6.1-4.. Adaptive Time Integration: Errors for Second Order Adaptive**

**Table 6.1-4.. Adaptive Time Integration: Convergence Rates for Second Order Adaptive**

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
12	4.81	4.26	3.24	4.34
19	2.81	2.39	1.41	2.71
26	1.61	4.38	2.38	1.35
41	1.88	2.77	1.51	1.92
70	2.17	1.70	1.34	2.24

### 6.1.9. Results: BDF2 Fixed

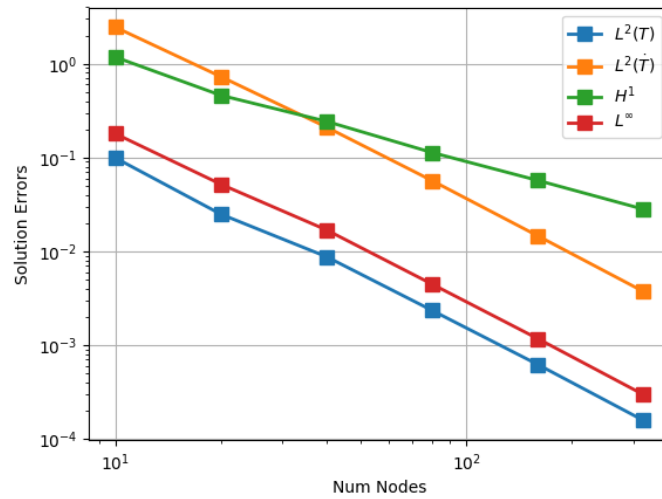


Figure 6.1-5.. Adaptive Time Integration: Errors for BDF2 Fixed

Table 6.1-5.. Adaptive Time Integration: Convergence Rates for BDF2 Fixed

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
20	2.00	1.75	1.36	1.80
40	1.51	1.79	0.92	1.61
80	1.90	1.90	1.10	1.93
160	1.92	1.95	0.99	1.93
320	1.98	1.98	1.02	1.98

### 6.1.10. Results: BDF2 Adaptive

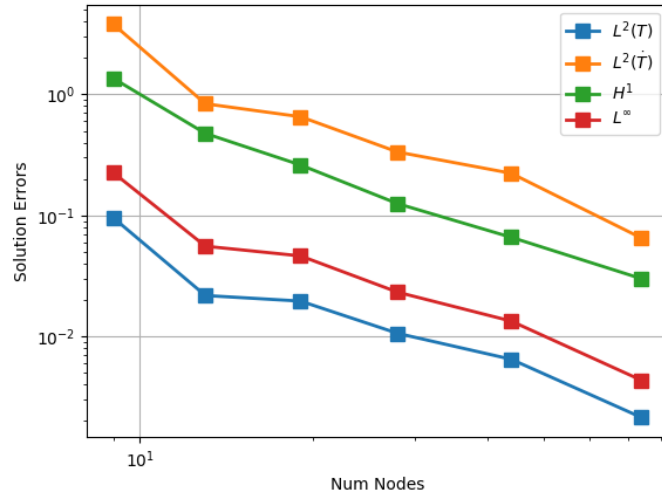


Figure 6.1-6.. Adaptive Time Integration: Errors for BDF2 Adaptive

Table 6.1-6.. Adaptive Time Integration: Convergence Rates for BDF2 Adaptive

Num Dofs	$L^2(T)$	$L^2(\dot{T})$	$H^1$	$L^\infty$
13	4.03	4.12	2.86	3.80
19	0.27	0.65	1.58	0.49
28	1.59	1.73	1.89	1.78
44	1.09	0.90	1.42	1.22
74	2.13	2.37	1.51	2.17

# 7. ENCLOSURE RADIATION

## 7.1. 2D CYLINDRICAL SHELL ENCLOSURE

### 7.1.1. Problem Description

This problem tests steady state coupled conduction and enclosure radiation. The geometry of this problem is a hollow cylinder (block 2) inside a second hollow cylinder (block 1), which is a radially symmetric problem.

### 7.1.2. Features Tested

Basic heat conduction, enclosure radiation, Quad<sub>4</sub>/Tri<sub>3</sub> meshes.

### 7.1.3. Boundary Conditions

The boundary conditions specify the temperature of the outer surface of the outer sphere ( $T(r_4) = T_4$ ) and the inner surface of the inner sphere ( $T(r_1) = T_1$ ).

The problem is steady state but is initialized with a constant temperature of 300 in both blocks. The inner surface temperature  $T_1$  is set to 300. The outer surface temperature  $T_4$  is set to 1300.

Dimensions are defined in Table 7.1-1.

**Table 7.1-1.. Dimensions of problem**

radius of surface_1	$r_1$	0.01
radius of surface_2	$r_2$	0.02
radius of surface_3	$r_3$	0.03
radius of surface_4	$r_4$	0.04

### 7.1.4. Material Parameters

Material properties are shown in Table 7.1-2.

**Table 7.1-2.. Material properties**

Thermal conductivity (block_1)	$\kappa_1$	2.0
Thermal conductivity (block_2)	$\kappa_2$	0.35
Density	$\rho$	1.0
Specific heat	$C_p$	1.0
emissivity (surface_2)	$\epsilon_2$	0.50
emissivity (surface_3)	$\epsilon_3$	0.55
Stefan-Boltzmann constant	$\sigma$	5.6704e-8

### 7.1.5. Verification of Solution

In cylindrical coordinates, the temperature is independent of  $\theta$  and  $z$ . Integrating this equation twice with respect to the radius  $r$ , we obtain the general solution in either hollow cylinder to be

$$T(r) = C_1 \log(r) + C_2,$$

for arbitrary constants  $C_1$  and  $C_2$ . We will use  $r_i, i = 1, \dots, 4$  to denote the location of the four surfaces of constant  $r$ , numbered from inside to outside. Unless specified otherwise, we will use these subscripts for other quantities which are evaluated at one of the four surfaces.

Including the boundary conditions into the solution allows us to eliminate two constants and gives

$$T_{inner}(r) = T_1 + c_I \log(r/r_1) \text{ for } r_1 < r < r_2 \quad (7.1)$$

$$T_{outer}(r) = T_4 + c_O \log(r/r_4) \text{ for } r_3 < r < r_4 \quad (7.2)$$

To solve for  $c_I$  and  $c_O$  we compute the temperatures at the enclosure surfaces  $r_2$  and  $r_3$ , defined as  $T_2 = T_{inner}(r_2)$  and  $T_3 = T_{outer}(r_3)$ :

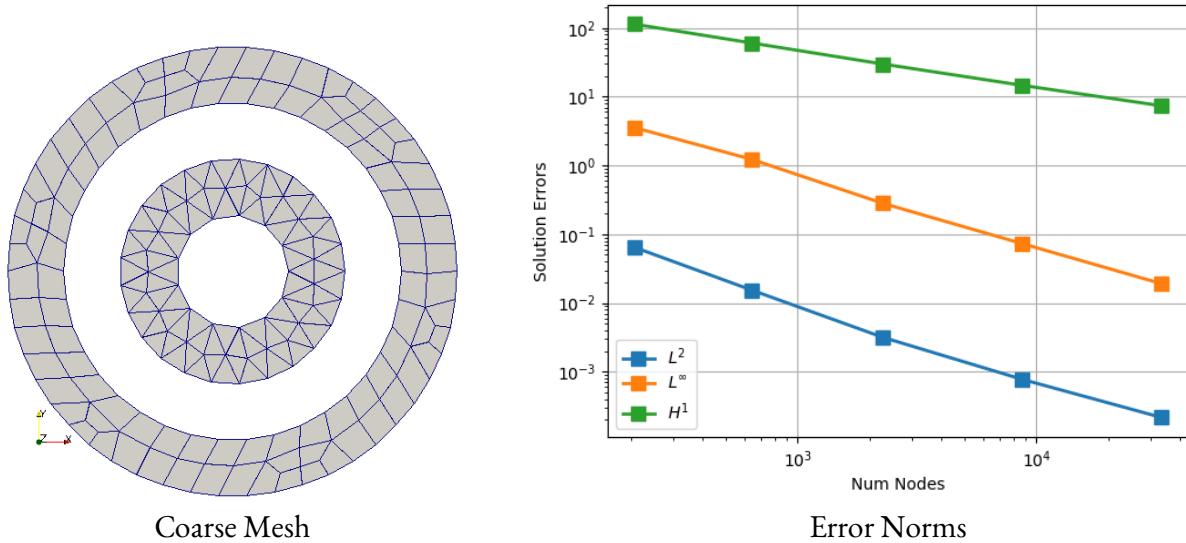
### 7.1.6. Results

The exact temperatures at the enclosure surfaces (to six digits precision) are  $T_2 = 444.7977$  and  $T_3 = 956.5915$ . From these values we can compute the values of  $c_O$  and  $c_I$  and thus the exact solution.

For each mesh, the errors in the temperature solution are computed in the  $L^2, L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

These optimal rates are observed in this test.

For input decks see Appendix [12.6.1](#).



**Figure 7.1-1.. Enclosure Radiation 2D**

**Table 7.1-3.. Enclosure Radiation 2D: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
640	2.54	1.88	1.11
2276	2.50	2.33	1.11
8673	2.10	2.01	1.07
33500	1.90	2.00	1.02

## 7.2. 2D ANNULAR ENCLOSURE

### 7.2.1. Problem Description

This problem tests steady state coupled conduction and enclosure radiation. The geometry is an annulus with a crack.

### 7.2.2. Features Tested

Basic heat conduction, enclosure radiation, Tri3 mesh.

### 7.2.3. Boundary Conditions

The outer and crack boundary conditions are prescribed at the nodes using the analytic solution. The inner boundary uses an enclosure boundary condition.

## 7.2.4. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant.

## 7.2.5. Verification of Solution

The manufactured solution is

$$\begin{aligned}
 J(\theta) &= k_1 + k_2 \sqrt{\varepsilon} \cos\left(\frac{\sqrt{\varepsilon}\theta}{2}\right) / \sin\left(\sqrt{\frac{\varepsilon}{\pi}} 2\right), \\
 H(\theta) &= k_1 + k_2 \left(\frac{\sqrt{\varepsilon}}{1-\varepsilon}\right) \left(\cos\left(\frac{\sqrt{\varepsilon}\theta}{2}\right) / \sin\left(\frac{\sqrt{\varepsilon}\pi}{2}\right) - \sqrt{\varepsilon} \cos\left(\frac{\theta}{2}\right)\right), \\
 q(\theta) &= J(\theta) - H(\theta), \\
 \beta(\theta) &= \left(\frac{k_1 + k_2 \cos\left(\frac{\theta}{2}\right)}{\sigma}\right)^{1/4}, \\
 T(r, \theta) &= r\beta(\theta) + (r - r_{\text{cyl}}) \left(\frac{q(\theta)}{\kappa} - \beta(\theta)\right),
 \end{aligned}$$

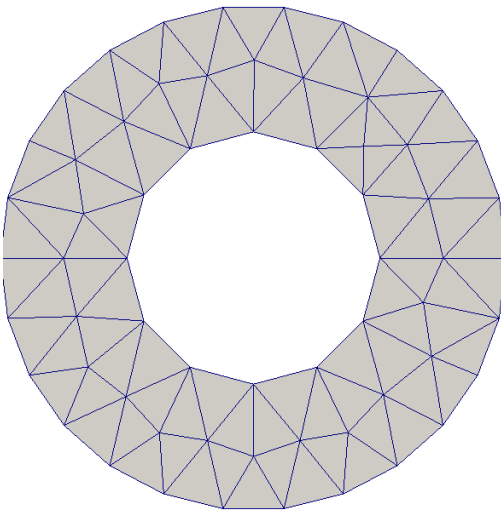
where  $J$  is the radiosity,  $H$  is the irradiance,  $q$  is the flux, and

$$\begin{aligned}
 \sigma &= 5.6704 \times 10^{-8}, \\
 \kappa &= 1, \\
 r_{\text{cyl}} &= 1, \\
 \varepsilon &= 0.9, \\
 k_1 &= 8000, \\
 k_2 &= 400.
 \end{aligned}$$

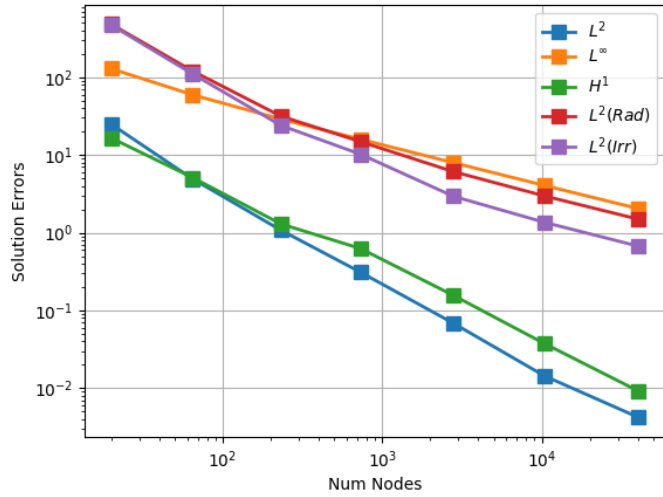
For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance). Additionally, the errors in the radiosity and irradiance are computed in the  $L^2$  norms and be 1 (within a tolerance).

These optimal rates are observed in this test.

For input decks see Appendix [12.6.2](#).



Coarse Mesh



Error Norms

**Figure 7.2-1.. 2D Full Enclosure Radiation**

**Table 7.2-1.. 2D Full Enclosure Radiation: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$	$L^2(Rad)$	$L^2(Irr)$
65	2.76	1.31	2.00	2.36	2.46
232	2.38	1.13	2.15	2.10	2.40
734	2.18	1.04	1.27	1.32	1.49
2788	2.26	1.03	2.07	1.32	1.85
10420	2.37	1.03	2.17	1.09	1.18
40530	1.81	1.01	2.08	1.01	1.04

## 7.3. 3D SPHERICAL SHELL ENCLOSURE

### 7.3.1. Problem Description

This problem tests steady state coupled conduction and enclosure radiation. The geometry of this problem is a hollow sphere (block 2) inside a second hollow sphere (block 1), which is a radially symmetric problem.

### 7.3.2. Features Tested

Basic heat conduction, enclosure radiation, Hex8 meshes.

### 7.3.3. Boundary Conditions

The boundary conditions specify the temperature of the outer surface of the outer sphere ( $T(r_4) = T_4$ ) and the inner surface of the inner sphere ( $T(r_1) = T_1$ ).

The problem is steady state but is initialized with a constant temperature of 300 in both blocks. The inner surface temperature  $T_1$  is set to 300. The outer surface temperature  $T_4$  is set to 1300.

Dimensions are defined in Table 7.3-1.

**Table 7.3-1.. Dimensions of problem**

radius of surface_1	$r_1$	0.01
radius of surface_2	$r_2$	0.02
radius of surface_3	$r_3$	0.03
radius of surface_4	$r_4$	0.04

### 7.3.4. Material Parameters

Material properties are shown in Table 7.3-2.

**Table 7.3-2.. Material properties**

Thermal conductivity (block_1)	$\kappa_1$	2.0
Thermal conductivity (block_2)	$\kappa_2$	0.35
Density	$\rho$	1.0
Specific heat	$C_p$	1.0
emissivity (surface_2)	$\epsilon_2$	0.50
emissivity (surface_3)	$\epsilon_3$	0.55
Stefan-Boltzmann constant	$\sigma$	5.6704e-8

### 7.3.5. Verification of Solution

In spherical coordinates, the temperature is independent of  $\theta$  and  $\phi$ . Integrating this equation twice with respect to the radius  $r$ , we obtain the general solution in either hollow sphere to be

$$T(r) = C_1 r^{-1} + C_2,$$

for arbitrary constants  $C_1$  and  $C_2$ . We will use  $r_i, i = 1, \dots, 4$  to denote the location of the four surfaces of constant  $r$ , numbered from inside to outside. Unless specified otherwise, we will use these subscripts for other quantities which are evaluated at one of the four surfaces.

Including the boundary conditions into the solution allows us to eliminate two constants and gives

$$T_{inner}(r) = T_1 + c_I \left( \frac{1}{r} - \frac{1}{r_4} \right) \text{ for } r_1 < r < r_2 \quad (7.3)$$

$$T_{outer}(r) = T_4 + c_O \left( \frac{1}{r} - \frac{1}{r_1} \right) \text{ for } r_3 < r < r_4 \quad (7.4)$$

To solve for  $c_I$  and  $c_O$  we compute the temperatures at the enclosure surfaces  $r_2$  and  $r_3$ , defined as  $T_2 = T_{inner}(r_2)$  and  $T_3 = T_{outer}(r_3)$ :

$$T_2 = T_1 + c_I \left( \frac{1}{r_2} - \frac{1}{r_4} \right) \quad (7.5)$$

$$T_3 = T_4 + c_O \left( \frac{1}{r_3} - \frac{1}{r_1} \right) \quad (7.6)$$

The fluxes at the surfaces between the two hollow spheres are

$$q_2 = \left( -\kappa \frac{\partial T}{\partial r} \Big|_{r=r_3} \right) \cdot \mathbf{n} = \frac{\kappa_1 c_I}{r_2^2}$$

$$q_3 = \left( -\kappa \frac{\partial T}{\partial r} \Big|_{r=r_2} \right) \cdot \mathbf{n} = \frac{\kappa_2 c_O}{r_3^2}$$

Here we have used  $\kappa_1$  and  $\kappa_2$  to denote the thermal conductivity of the inner and outer blocks, respectively.

These normal conductive fluxes are included in the total energy balance at the enclosure surfaces using the radiative transport equations (for grey diffuse surfaces):

$$q_2 = \sigma \epsilon_2 T_2^4 - \epsilon_2 \sum_j F_{2j} J_j$$

$$q_3 = \sigma \epsilon_3 T_3^4 - \epsilon_3 \sum_j F_{3j} J_j$$

where  $\sigma$  is the Stefan Boltzmann constant,  $\epsilon$  is the emissivity,  $F_{ij}$  is the geometric viewfactor of surface  $i$  with respect to surface  $j$  and  $J_j$  is the radiosity for surface  $j$ .

The viewfactor coefficient  $F_{ij}$  is the fraction of energy that leaves surface  $i$  and arrives at surface  $j$ . For this geometric setup, no point on the inner surface at  $r_2$  can “see” itself (no straight line can be drawn from a point on its surface onto itself) and so  $F_{22} = 0$ . By viewfactor reciprocity

$$\sum_j F_{ij} = 1$$

we must have  $F_{23} = 1$ . The outer-to-inner view factor  $F_{32}$  can be computed analytically to be

$$F_{32} = \frac{r_2^2}{r_3^2}$$

and again by viewfactor reciprocity

$$F_{33} = 1 - F_{32} = 1 - \frac{r_2^2}{r_3^2}$$

The system of equations that must be solved for the radiosities at the inner and outer surfaces is given by

$$J_2 = \epsilon_2 \sigma T_2^4 + (1 - \epsilon_2)[F_{22}J_2 + F_{23}J_3]$$

$$J_3 = \epsilon_3 \sigma T_3^4 + (1 - \epsilon_3)[F_{32}J_2 + F_{33}J_3]$$

Solving this system of equations, we can write  $J_2$  and  $J_3$  in terms of temperature, and plug this back into the equation for the surface flux. We then get a system of two nonlinear equations to solve for  $T_2$  and  $T_3$ , the temperatures of the adjacent surfaces without Dirichlet boundary conditions. For our given set of parameters, these equations are solved iteratively in Matlab using the `fsolve` function.

### 7.3.6. Results

The exact temperatures at the enclosure surfaces (to six digits precision) are  $T_2 = 564.783$  and  $T_3 = 1047.825$ . From these values we can compute the values of  $c_O$  and  $c_I$  and thus the exact solution.

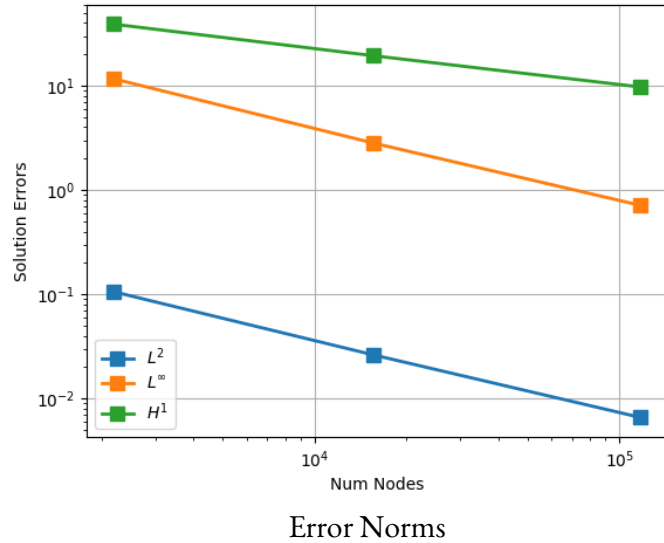
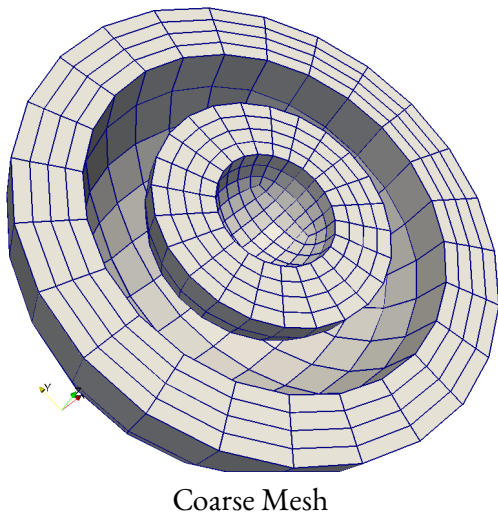
For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

These optimal rates are observed in this test.

**Table 7.3-3.. Enclosure Radiation: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
15590	2.14	2.17	1.06
117600	2.05	2.05	1.03

For input decks see Appendix [12.6.3](#).



**Figure 7.3-1.. Enclosure Radiation**

## 7.4. 3D SPHERICAL SHELL PARTIAL ENCLOSURE

### 7.4.1. Problem Description

This problem tests coupled conduction and enclosure radiation with a partial enclosure. The geometry consists of two thick spherical shells separated by a gap. The outer shell has a section removed so that the enclosure is only partial.

### 7.4.2. Features Tested

Basic heat conduction, enclosure radiation with partial enclosure, Hex8 meshes.

### 7.4.3. Boundary Conditions

The outer and inner boundary conditions are prescribed at the nodes using the analytic solution. The analytic solution is used to set the boundary conditions on the cutaway face near the opening in the outer shell.

### 7.4.4. Material Parameters

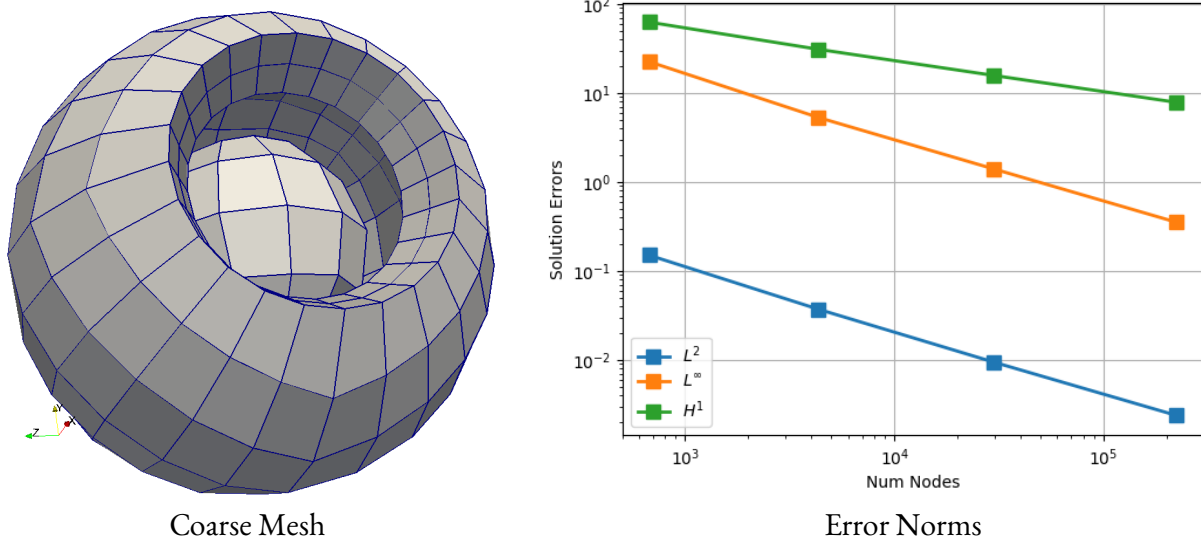
The values of density, thermal conductivity, and specific heat are all constant within each element block; however, the values differ between blocks.

### 7.4.5. Verification of Solution

The analytic solution is identical to Section 7.3. The area for the partial enclosure is computed analytically.

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 2, 2, and 1, respectively (within a tolerance).

These optimal rates are observed in this test.



**Figure 7.4-1.. Partial Enclosure Radiation**

**Table 7.4-1.. Partial Enclosure Radiation: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$	$H^1$
4338	2.26	2.32	1.13
29690	2.13	2.07	1.06
223200	2.06	2.06	1.03

For input decks see Appendix 12.6.4.

## 8. CHEMISTRY

### 8.1. FIRST ORDER REACTION (SPATIALLY VARYING TEMPERATURE)

This problem tests the interface to the CHEMEQ solver under the assumption that the temperature remains variable in space but remains constant in time. The geometry consists of a unit cube meshed with Hex8 elements refined only in one direction ( $x$ ).

#### 8.1.1. Features Tested

CHEMEQ solver; source term from chemistry; nonlinear solver; second order time integrator with fixed time steps; constant initial temperature; constant temperature boundary condition.

#### 8.1.2. Boundary Conditions

A constant temperature is applied on surface 1. The initial temperature is provided by an Encore user subroutine and the initial species values are  $A = 1$  and  $B = 0$ .

#### 8.1.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks. The CHEMEQ parameters are chosen to model a single first order reaction  $A \rightarrow B$  with constant values of pre-exponential factor and activation energy.

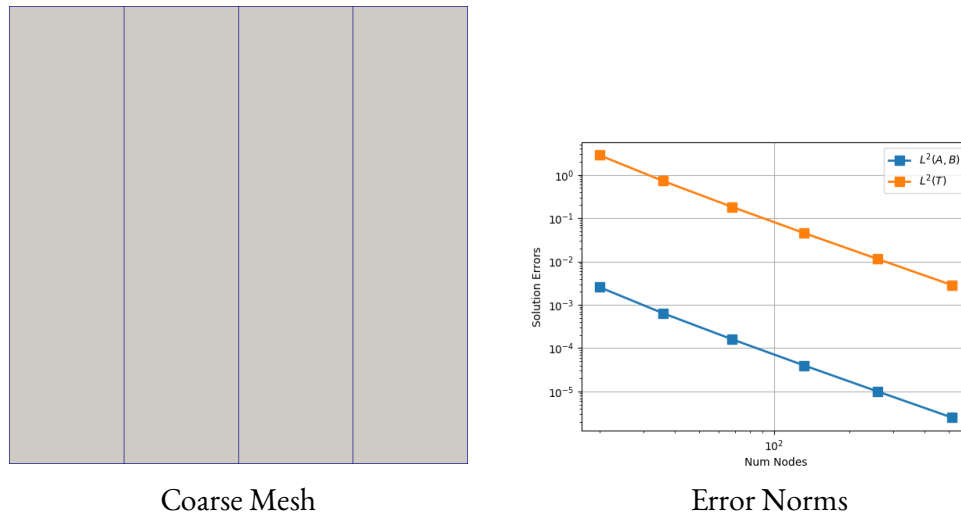
#### 8.1.4. Verification of Solution

A manufactured solution is chosen as

$$\begin{aligned} T(x) &= 400 (1 + 0.2 \cos(\pi x)), \\ A(x, t) &= \exp \left\{ -\exp(5) \exp\left(-\frac{1000}{RT(x)}\right) t \right\}, \\ B(x, t) &= 1 - A(x, t) \end{aligned}$$

where  $R = 1.9872$  is the ideal gas constant. A source term is used to insure that the temperature does not vary in time.

For each mesh, the errors in the temperature and species  $A$  and  $B$  are computed in the  $L^2$  norm. The test passes, only if the observed rates of convergence in these norms are 2 (within a tolerance).



**Figure 8.1-1.. First Order Reaction (Spatially Varying Temperature)**

**Table 8.1-1.. First Order Reaction (Spatially Varying Temperature): Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2(A, B)$	$L^2(T)$
36	2.37	2.34
68	2.18	2.18
132	2.09	2.09
260	2.04	2.04
516	2.02	2.02

For input decks see Appendix [12.7.2](#).

## 8.2. FIRST ORDER REACTION

This problem tests the interface to the CHEMEQ solver under a temperature field that is variable in space and time. The geometry consists of a unit cube meshed with Hex8 elements refined only in one direction.

### 8.2.1. Features Tested

CHEMEQ solver; source term from chemistry; nonlinear solver; second order time integrator with fixed time steps; initial temperature from user sub; constant temperature boundary condition.

## 8.2.2. Boundary Conditions

The initial temperature and the temperature boundary condition on surface 1 are provided by an Encore user subroutine and the initial species values are  $A = 1$  and  $B = 0$ .

## 8.2.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant, equal to one in both blocks. The CHEMEQ parameters are chosen to model a single first order reaction  $A \rightarrow B$  with constant values of pre-exponential factor and activation energy.

## 8.2.4. Verification of Solution

A manufactured solution is chosen as

$$\begin{aligned}\phi(x, t) &= \exp(a - E/(RT_0))(1 + 0.1 \sin(x)) \exp(t), \\ \Phi(x, t) &= \exp(a - E/(RT_0))(1 + 0.1 \sin(x))(\exp(t) - 1), \\ T(x, t) &= (E/R)/(a - \ln(\phi(x, t))), \\ A(x, t) &= \exp(-\Phi(x, t)), \\ B(x, t) &= 1 - A(x, t)\end{aligned}$$

where  $a$  is the log pre-exponential factor,  $R$  is the ideal gas constant,  $E$  is the activation energy, and  $T_0$  is a reference temperature value. The form of the solution is contrived so that

$$\begin{aligned}\partial_t A(x, t) &= -\partial_t B(x, t) = -\phi(x, t) A(x, t) \\ \phi(x, t) &= \exp(a) \exp\left(-\frac{E}{RT(x, t)}\right)\end{aligned}$$

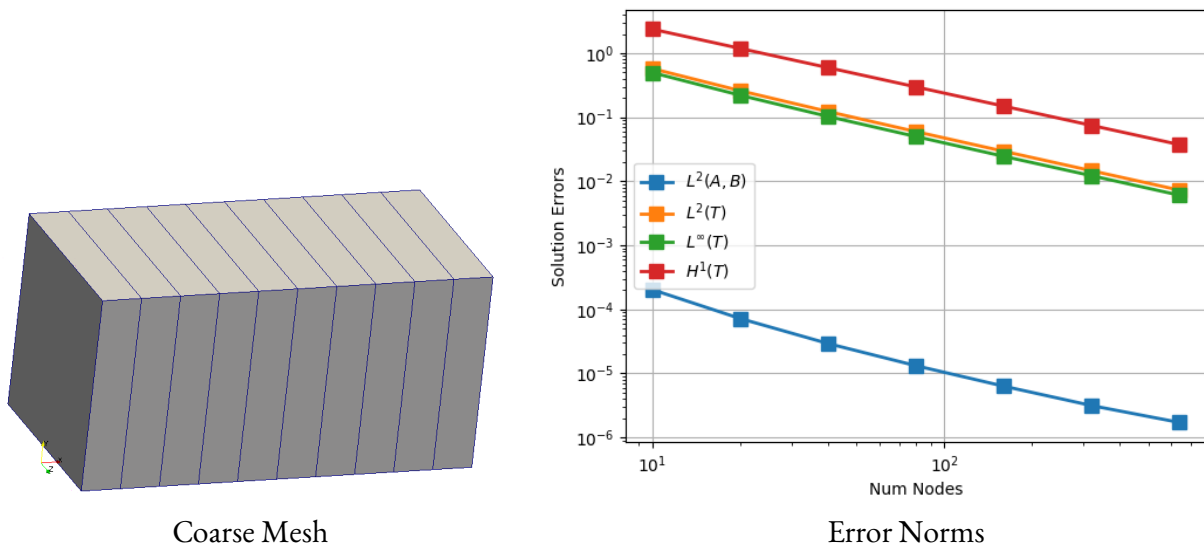
This allows the chemistry ODEs to be satisfied exactly, but a source term is needed in the energy equation.

For each mesh, the errors in the temperature and species  $A$  and  $B$  are computed in the  $L^2$  norm. The test passes, only if the observed rates of convergence in these norms are 1 (within a tolerance). Currently it is not clear why the convergence rates are only first order.

For input decks see Appendix [12.7.3](#).

## 8.3. DAE AND PRESSURE TEST

This test runs CHEMEQ with a kinetics model that includes both pressure dependence and distributed activation energy for a single element mesh with uniform temperature and pressure.



**Figure 8.2-1.. First Order Reaction**

**Table 8.2-1.. First Order Reaction: Convergence Rates for Hex8 Meshes**

Num Dofs	$L^2(A, B)$	$L^2(T)$	$L^\infty(T)$	$H^1(T)$
20	1.50	1.15	1.17	1.00
40	1.30	1.08	1.10	1.00
80	1.15	1.04	1.05	1.00
160	1.07	1.02	1.03	1.00
320	1.00	1.01	1.01	1.00
640	0.88	1.01	1.01	1.00

### 8.3.1. Features Tested

Basic heat conduction on a Hex8 mesh; CHEMEQ solver with pressure dependence and distributed activation energy.

### 8.3.2. Boundary Conditions

No boundary conditions are prescribed, resulting in an adiabatic flux BC.

### 8.3.3. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant.

### 8.3.4. Verification of Solution

The analytic solution for the concentration of species A as a function of time for the constant values used in this test case is

$$A(t) = \frac{1}{2} \operatorname{erfc}\left(\frac{1}{6}(\sqrt{2} - 6 \operatorname{erf}^{-1}(1 - 45t/(2 \exp(\frac{61}{18}))))\right)$$

The test compares the temperature errors against a gold file of the error at each time step. The exact solution for the concentration of A is also output to the exodus file and a comparison plotting that and the solved for concentration as a function of time has them lying on top of one another.

For input decks see Appendix [12.7.4](#).

## 8.4. PMDI PLUGIN TEST

This test verifies that the PMDI plugin calculates the correct pressure and effective conductivity based on the auxiliary variable values.

### 8.4.1. Features Tested

Basic heat conduction on a Hex8 mesh; CHEMEQ solver with pressure dependence and a user plugin to model a seven-species PMDI foam decomposition reaction.

### 8.4.2. Boundary Conditions

No boundary conditions are prescribed, resulting in an adiabatic flux BC.

### 8.4.3. Material Parameters

The values of density, emissivity and specific heat are all constant. The thermal conductivity is computed using a C-style user subroutine contained within the foam model.

### 8.4.4. Verification of Solution

The initial conditions are specified as follows: The test includes a Mathematica notebook file (ExpectedSolution.nb) for calculation of expected pressure which is 1.15125e7 Pa or 1669.75 psi.

For input decks see Appendix [12.7.5](#).

**Table 8.4-1.. PMDI Plugin Test: Initial Conditions**

Variable	Value	Units
Bulk Density	321.4432249	$kg/m^3$
Initial condensed density	1500	
Initial porosity	0.786301	
Mass fraction of all ChemEQ species	1/7	
Temperature	599.8	
Initial gas pressure (N <sub>2</sub> )	101325	
Initial gas temperature	299.9	

# 9. MISCELLANEOUS

## 9.1. THERMAL POSTPROCESSING

### 9.1.1. Problem Description

This problem tests basic thermal postprocessors in Aria.

### 9.1.2. Features Tested

Basic heat conduction, thermal postprocessors, Hex8 meshes.

### 9.1.3. Boundary Conditions

Dirichlet BCs are specified using the exact solution on surface 1. On surface 2, a natural convection BC is specified, using the exact solution as the reference temperature and a constant heat transfer coefficient. Similarly, a radiative flux BC is applied on surface 3, with constant values of emissivity and radiation form factor. A source term is applied within all blocks based on substituting the exact solution into the heat conduction operator.

### 9.1.4. Material Parameters

The values of density, thermal conductivity, and specific heat are all constant with the same value for both blocks.

### 9.1.5. Verification of Solution

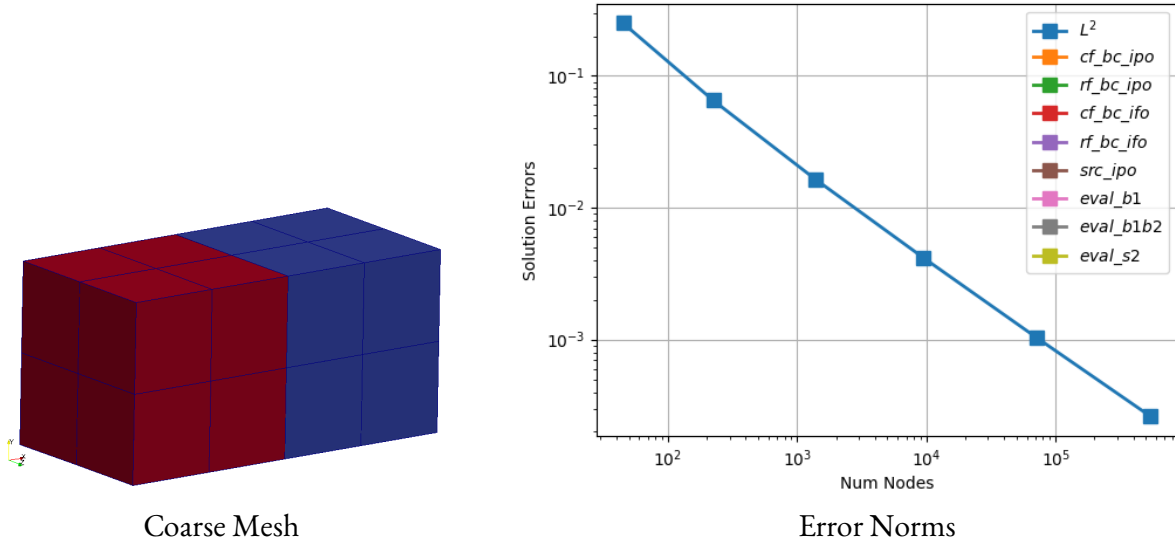
The manufactured solution is

$$T_0 + \exp(C_0(x^2 - 1) + C_1(y^2 - 0.25) + C_2(z^2 - 0.25) + C_3t).$$

Postprocessors are computed for the integrated power output for convective and radiative BCs (cf\_bc\_ipo, rf\_bc\_ipo), the integrated flux output for convective and radiative BCs (cf\_bc\_ifo, rf\_bc\_ifo), the integrated power output for volume source terms (src\_ipo), and several point evaluations (eval\_b1, eval\_b1b2, eval\_s2).

For each mesh, the errors in the temperature solution are computed in the  $L^2$  norm and for various postprocessors. The test passes, only if the observed rates of convergence are 2 (except for the integrated power output for source terms, which convergences with order 4).

These optimal rates are observed in this test clearly in most cases. However, for the point evaluation cases, a large amount of variability exists in the convergence rates.



**Figure 9.1-1.. Thermal Postprocess**

**Table 9.1-1.. Thermal Postprocess: Convergence Rates**

Num Dofs	$L^2$	<i>cf_bc_ipo</i>	<i>rf_bc_ipo</i>	<i>cf_bc_ifo</i>	<i>rf_bc_ifo</i>	<i>src_ipo</i>	<i>eval_b1</i>	<i>eval_b1b2</i>	<i>eval_s2</i>
225	2.52	2.88	2.83	2.88	2.83	4.62	2.21	1.94	2.56
1377	2.27	2.52	2.48	2.52	2.48	4.44	2.83	3.47	5.75
9537	2.14	2.27	2.25	2.27	2.25	4.26	1.71	1.98	-0.16
70785	2.07	2.13	2.12	2.13	2.12	4.14	2.15	2.05	2.37
545025	2.04	2.06	2.05	2.06	2.05	4.07	2.20	1.57	2.44

For input decks see Appendix 12.8.1.

## 9.2. LOCAL COORDINATES: CARTESIAN

This problem tests the use of a local Cartesian coordinate system in a material model. The geometry is a 3D cube that has been rotated.

### 9.2.1. Features Tested

Steady heat conduction, time integrators, tensor thermal conductivity, local Cartesian coordinates in a material model.

### 9.2.2. Boundary Conditions

The boundary conditions are prescribed at the nodes using the analytic solution. The initial condition is specified using an Encore function evaluated at the nodes.

### 9.2.3. Material Parameters

The specific heat is constant. The density and thermal conductivity are constant, with a diagonal (tensor) thermal conductivity in the local coordinate space of the material.

### 9.2.4. Verification of Solution

A manufactured solution is chosen as

$$T(X, Y, Z) = T_0 + T_1 \cos(x_k X) \cos(y_k Y) \cos(z_k Z)$$

where  $(X, Y, Z)$  are the local material coordinates, which are related to the Cartesian coordinates  $(x, y, z)$  by a rotation matrix consisting of a product of rotations (22.5 deg around the  $z$ -axis and 45 deg around the  $x$ -axis).

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 1 for  $H^1$  and 2 for all other norms (within a tolerance).

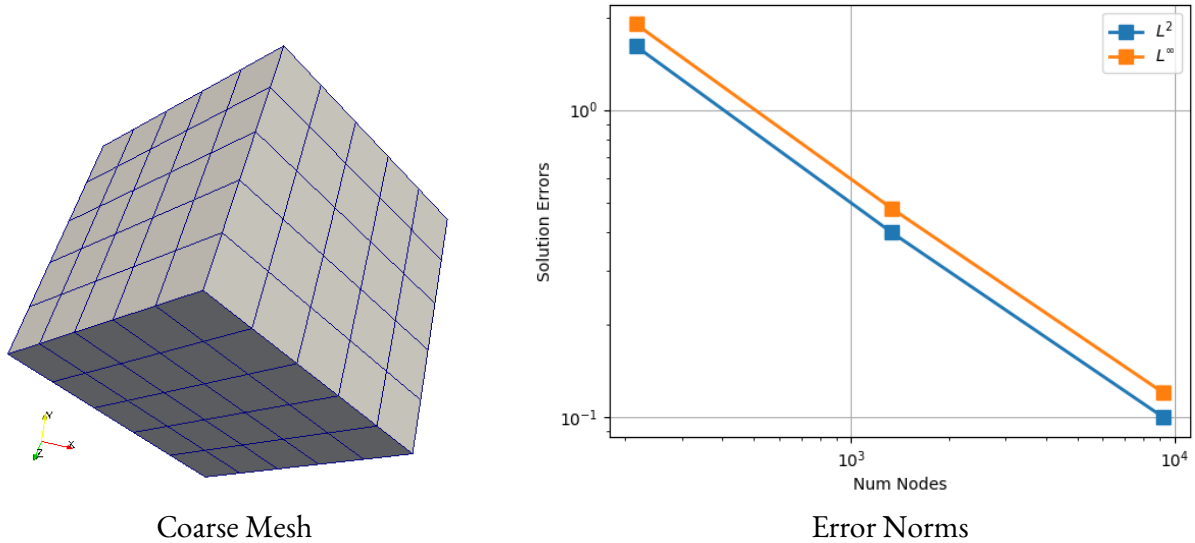
**Table 9.2-1.. Local Cartesian Coordinate System: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$
1331	2.29	2.27
9261	2.15	2.14

For input decks see Appendix [12.8.3](#).

## 9.3. LOCAL COORDINATES: CYLINDRICAL

This problem tests the use of a local cylindrical coordinate system in a material model. The geometry is a 3D cube that has been rotated.



**Figure 9.2-1.. Local Cartesian Coordinate System**

### 9.3.1. Features Tested

Steady heat conduction, time integrators, tensor thermal conductivity, local coordinates in a material model.

### 9.3.2. Boundary Conditions

The boundary conditions are prescribed at the nodes using the analytic solution. The initial condition is specified using an Encore function evaluated at the nodes.

### 9.3.3. Material Parameters

The specific heat and density are constant. The diagonal (tensor) components of the thermal conductivity are specified using constant values in the local coordinate space of the material.

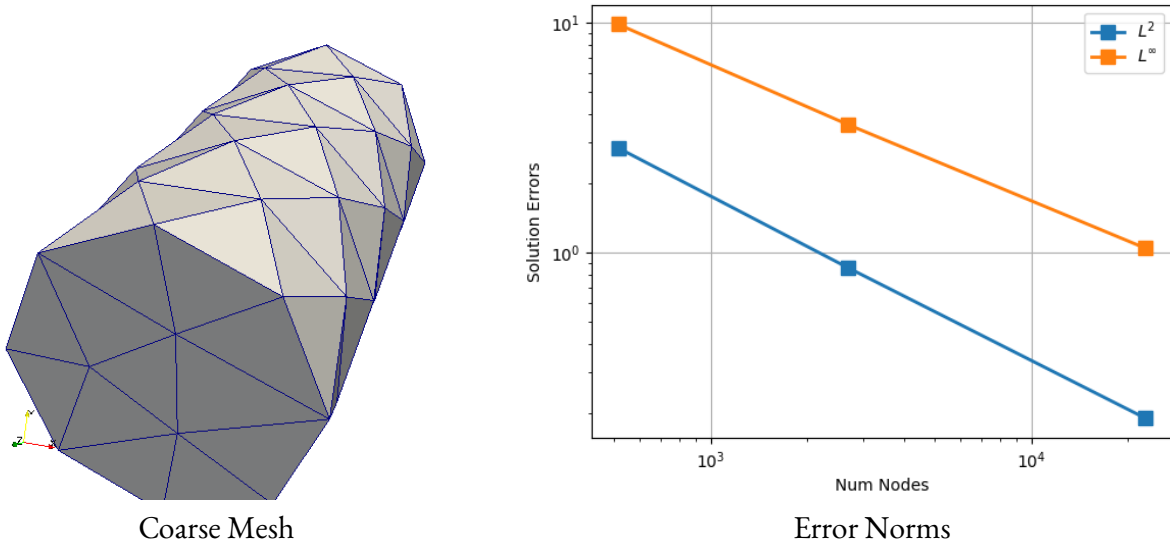
### 9.3.4. Verification of Solution

A manufactured solution is chosen as

$$T(X, Y, Z) = T_0 + T_1(2R)^3 \cos(\theta) \cos(z_k Z)$$

where  $(R, \Theta, Z)$  are the local cylindrical material coordinates, which are related to the standard cylindrical coordinates  $(x, y, z)$  by a rotation matrix consisting of a product of rotations (22.5 deg around the  $z$ -axis and 45 deg around the  $x$ -axis).

For each mesh, the errors in the temperature solution are computed in the  $L^2$ ,  $L^\infty$  and  $H^1$  norms. The test passes, only if the observed rates of convergence in these norms are 1 for  $H^1$  and 2 for all other norms (within a tolerance).



**Figure 9.3-1.. Local Cylindrical Coordinate System**

**Table 9.3-1.. Local Cylindrical Coordinate System: Convergence Rates**

Num Dofs	$L^2$	$L^\infty$
2692	2.19	1.83
22723	2.12	1.74

For input decks see Appendix [12.8.4](#).

# 10. LOW-MACH FLUID FLOW

Documentation for the following tests is in progress:

```
1 _rtest/aria/cvfemConvTaylorVortex/cvfemConvTaylorVortex.test|np4
2 _rtest/aria/gfemConvTaylorVortex/gfemConvTaylorVortex.test|np4
3 _rtest/aria/hfemConvTaylorVortex/hfemConvTaylorVortex.test|np4
4 mConvTaylorVortex/cvfemConvTaylorVortex.test|np8
5 mSteadyTaylorVortex/cvfemSteadyTaylorVortex.test|np8
6 mSteadyTaylorVortexKeps/cvfemSteadyTaylorVortexKeps.test|np8
7 m_couette_flow/cdfem_couette_flow.test|cdfem_couette_flow_tri3
8 m_couette_flow/cdfem_couette_flow.test|cdfem_couette_flow_tri6
9 ConvTaylorVortex/gfemConvTaylorVortex.test|np8
10 SteadyTaylorVortex/gfemSteadyTaylorVortex.test|np8
```

# 11. HOW TO BUILD THIS DOCUMENT

You need to have Sierra developer access (through WebCars). Then you should clone the Sierra Git repository containing the tests to a location with adequate memory (currently more than 80GB), using a command like this:

```
git clone sierra-git:/git/tests
```

Then you need to assign the verification tests, running the following command from your local tests repository:

```
assign --path aria_rtest/verification
```

This will produce a text file called `assigned.tests` containing the list of all tests to run. You should edit the second line of this file to indicate the remote location (accessible from the HPC machine where you will run the tests). For example, I might have something like this:

```
# Created by assign at Fri Sep 19 09:52:09 2014
#@ /gscratch1/bcarnes/TESTS
aria_rtest/verification/1dnonlin_verify1/1dnonlin_verify1.test|np8
aria_rtest/verification/cyl_shell_2d/cyl_shell_2d.test|np8
aria_rtest/verification/cyl_shell_3d/cyl_shell_3d.test|np8
...
```

Next you need to copy the test files and the `assigned.test` file to the remote location (here it is “/gscratch1/bcarnes/TESTS/”):

```
rsync -azv aria_rtest/verification redsky:/gscratch1/bcarnes/TESTS/aria_rtest
scp assigned.tests redsky:/gscratch1/bcarnes/TESTS/
```

Here I am only copying the verification test sub-directory, since I do not want to run any other tests.

On the HPC machine, you will need to load a pre-built version of the code such as the nightly master build:

```
module load sierra/master
```

To see where the executables are located, you can run something like:

```
[bcarnes@redsky-login9 ~]$ which aria
/projects/sierra/redsky/install/master/bin/aria
```

Finally, to run the tests, you use the `testrun` script, with a few additional arguments. The first locates the source code needed to compile the various user subroutines (which we just found from running “which aria”), the second enables tests to run as long as needed, the third uses the queue, and the fourth saves the results so you can use them in the manual.

```
testrun --user sourcedir=/projects/sierra/tlcc2/install/master/ \
        --allow-multipliers=time \
        --queued \
        --save-all-results
```

It may take 1-2 hours to run all the tests. Note that if the tests start to fail with an error associated with the `ACCOUNT` not being set, you may need to set it using your `WCID`:

```
export ACCOUNT=fyXXXXXX
```

To view your available `WCIDs`, run the following command:

```
mywcid
```

To build this manual, you should clone the Sierra Git repository containing the documentation files using a command like this:

```
git clone sierra-git:/git/docs
```

Then go to the directory within your local repository containing the Aria Verification Manual files:

```
cd aria/doc/verification_manual
```

Once the tests have all ran successfully, you should sync the results from the remote location back to this directory:

```
rsync -azv redsky:/gscratch1/bcarnes/TESTS/results .
```

Then run the a script to execute any local postprocessing needed to create the plots for the tests:

```
python ariaPostprocess.py
```

Finally you can create the manual using `pdflatex`:

```
pdflatex Aria_Verification_Manual.tex
```

which should create a new PDF output file.

# 12. INPUT DECKS FOR VERIFICATION PROBLEMS

## 12.1. BASIC THERMAL TESTS

### 12.1.1. Steady Heat Conduction: Hex8 Meshes

```
BEGIN SIERRA myJob

begin definition for function kxx
  type = piecewise linear
  begin values
    0.0 0.5
    1.0 2.0
    2.0 8.0
  end values
end

begin definition for function kyy
  type = piecewise linear
  begin values
    0.0 0.2
    1.0 1.2
    2.0 2.1
    20.0 20.2
  end values
end

begin definition for function kzz
  type = piecewise linear
  begin values
    0.0 1.0
    1.0 2.0
    2.0 3.0
    20.0 21.0
  end values
  scale by 2.0
end

BEGIN ARIA MATERIAL Kryptonite
  Density = Constant rho=1
  tensor thermal conductivity = user_function X = temperature name_xx = kxx name_yy = kyy name_zz = kzz
  Specific Heat = Constant cp=1
  heat conduction = Generalized
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
```

```

    Begin Trilinos Equation Solver Direct_Solver
      Solution Method = amesos-superlu
    End
  {endif}

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
      BEGIN CG SOLVER
        BEGIN JACOBI PRECONDITIONER
          END
          MAXIMUM ITERATIONS = 1000
          RESIDUAL SCALING = NONE
          CONVERGENCE TOLERANCE = 1.000000e-14
        END
      END TPETRA EQUATION SOLVER
    {else}
      Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = CG
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-14
      End
    {endif}

    BEGIN FINITE ELEMENT MODEL cube
      database name = cube_h{N}_hex8.e
      coordinate system is cartesian
      decomposition method = rcb

      BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
      END PARAMETERS FOR BLOCK block_1

    END FINITE ELEMENT MODEL cube

    Begin User Function exact_soln
      Load From File ./exact.so Using Function registerExactSoln
    End

    Begin User Function exact_src
      Load From File ./exact.so Using Function registerExactSrc
    End

    Begin User Function flux_surface_3
      Load From File ./exact.so Using Function registerFlux_Surface_3
    End

    Begin User Function flux_surface_5
      Load From File ./exact.so Using Function registerFlux_Surface_5
    End

    Begin User Function flux_surface_1
      Load From File ./exact.so Using Function registerFlux_Surface_1
    End

    Begin User Function flux_surface_2
      Load From File ./exact.so Using Function registerFlux_Surface_2
    End

    Begin Norm Postprocessor l2
      Use Function exact_soln
      Subtract Function nonlinear_solution->TEMPERATURE
      Compute Norms L2
    End

    Begin Norm Postprocessor h1
      Use Function exact_soln

```

```

    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors_thermal_steady_hex8_h{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Sequential The_Time_Block
        Advance myRegion
    End
        Simulation Start Time = 0
        Simulation Termination Time = 1
End
    End

    BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

```

```

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

    Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = thermal_steady_hex8_h{N}.e
    at step 1, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

    END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.2. Steady Heat Conduction: Hex20 Meshes

```

BEGIN SIERRA myJob

    load user plugin file ./exact.so

    begin definition for function kxx
        type = piecewise linear
        begin values
            0.0 0.5
            1.0 2.0
            2.0 8.0
        end values
    end

    begin definition for function kyy
        type = piecewise linear
        begin values
            0.0 0.2
            1.0 1.2
            2.0 2.1
            20.0 20.2
        end values
    end

    begin definition for function kzz
        type = piecewise linear
        begin values
            0.0 1.0
            1.0 2.0
            2.0 3.0
            20.0 21.0
        end values
        scale by 2.0
    end

    BEGIN ARIA MATERIAL Kryptonite
        Density = Constant rho=1

```

```

        tensor thermal conductivity = user_function X = temperature name_xx = kxx name_yy = kyy name_zz = kzz
        Specific Heat      = Constant cp=1
        heat conduction    = Generalized
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN SUPERLU SOLVER
        END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
    End
{endif}

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN CG SOLVER
            BEGIN JACOBI PRECONDITIONER
            END
            MAXIMUM ITERATIONS = 1000
            RESIDUAL SCALING = NONE
            CONVERGENCE TOLERANCE = 1.000000e-14
        END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = CG
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-14
    End
{endif}

BEGIN FINITE ELEMENT MODEL cube
    database name = cube_h{N}_hex20.g
    coordinate system is cartesian
    decomposition method = rcb

    BEGIN PARAMETERS FOR BLOCK block_1
        material Kryptonite
    END PARAMETERS FOR BLOCK block_1

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
    Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_src
    Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
    Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
    Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
    Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2

```

```

    Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
Write To File errors_thermal_steady_hex20_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential The_Time_Block
        Advance myRegion
      End
      Simulation Start Time = 0
      Simulation Termination Time = 1
    End
  End

  BEGIN ARIA REGION myRegion

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

    use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

    EQ ENERGY for TEMPERATURE on block_1 using Q2S with DIFF SRC

    # surface_4: x=0
    # surface_6: x=1

    # surface_3: y=0
    # surface_5: y=1

    # surface_1: z=1
    # surface_2: z=0

    # const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

```

```

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_hex20_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

### 12.1.3. Steady Heat Conduction: Hex27 Meshes

```

BEGIN SIERRA myJob

  load user plugin file ./exact.so

begin definition for function kxx
  type = piecewise linear
  begin values
    0.0 0.5
    1.0 2.0
    2.0 8.0
  end values
end

begin definition for function kyy
  type = piecewise linear
  begin values
    0.0 0.2
    1.0 1.2
    2.0 2.1
    20.0 20.2
  end values
end

begin definition for function kzz
  type = piecewise linear

```

```

begin values
  0.0 1.0
  1.0 2.0
  2.0 3.0
  20.0 21.0
end values
scale by 2.0
end

BEGIN ARIA MATERIAL Kryptonite
  Density = Constant rho=1
  tensor thermal conductivity = user_function X = temperature name_xx = kxx name_yy = kyy name_zz = kzz
  Specific Heat = Constant cp=1
  heat conduction = Generalized
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Direct_Solver
  Solution Method = amesos-superlu
  End
{endif}

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
  BEGIN CG SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 1000
  RESIDUAL SCALING = NONE
  CONVERGENCE TOLERANCE = 1.000000e-14
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Iterative_Solver
  Solution Method = CG
  Preconditioning Method = Jacobi
  Maximum Iterations = 1000
  Residual Norm Scaling = None
  Residual Norm Tolerance = 1.0e-14
  End
{endif}

BEGIN FINITE ELEMENT MODEL cube
  database name = cube_h{N}_hex27.g
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
  material Kryptonite
  END PARAMETERS FOR BLOCK block_1

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3

```

```

End

Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
Write To File errors_thermal_steady_hex27_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential The_Time_Block
        Advance myRegion
      End
      Simulation Start Time = 0
      Simulation Termination Time = 1
    End
  End

  BEGIN ARIA REGION myRegion

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

    use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

    EQ ENERGY for TEMPERATURE on block_1 using Q2 with DIFF SRC

    # surface_4: x=0
    # surface_6: x=1

```

```

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
BC const dirichlet at surface_4 Temperature = 1.0
BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_hex27_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.4. Steady Heat Conduction: Tet4 Meshes

```

BEGIN SIERRA myJob

load user plugin file ./exact.so

BEGIN ARIA MATERIAL Kryptonite
  Density          = Constant rho=1
  Thermal Conductivity = constant k=1
  Specific Heat    = Constant cp=1
  heat conduction  = basic
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER

```

```

        END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
    End
{endif}

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
    BEGIN CG SOLVER
        BEGIN JACOBI PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 1000
        RESIDUAL SCALING = NONE
        CONVERGENCE TOLERANCE = 1.000000e-14
    END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = CG
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-14
    End
{endif}

    BEGIN FINITE ELEMENT MODEL cube
        database name = cube_h{N}_tet4.e
        coordinate system is cartesian
        decomposition method = rcb

        BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
        END PARAMETERS FOR BLOCK block_1

    END FINITE ELEMENT MODEL cube

    Begin User Function exact_soln
        Load From File ./exact.so Using Function registerExactSoln
    End

    Begin User Function exact_soln_dot
        Load From File ./exact.so Using Function registerExactSolnDot
    End

    Begin User Function exact_src
        Load From File ./exact.so Using Function registerExactSrc
    End

    Begin User Function flux_surface_3
        Load From File ./exact.so Using Function registerFlux_Surface_3
    End

    Begin User Function flux_surface_5
        Load From File ./exact.so Using Function registerFlux_Surface_5
    End

    Begin User Function flux_surface_1
        Load From File ./exact.so Using Function registerFlux_Surface_1
    End

    Begin User Function flux_surface_2
        Load From File ./exact.so Using Function registerFlux_Surface_2
    End

    Begin Norm Postprocessor 12

```

```

    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
End

Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
Write To File errors_h{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Sequential The_Time_Block
        Advance myRegion
    End
    Simulation Start Time = 0
    Simulation Termination Time = 1
End
    End

    BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

```

```

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

    Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = thermal_steady_tet4_h{N}.e
    at step 1, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

    END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.5. Steady Heat Conduction: Tet4Tet10 Meshes

```

BEGIN SIERRA myJob

    load user plugin file ./exact.so

    BEGIN ARIA MATERIAL Kryptonite
        Density          = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat     = Constant cp=1
        heat conduction   = basic
    END ARIA MATERIAL Kryptonite

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN SUPERLU SOLVER
        END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
        End
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN CG SOLVER
        BEGIN JACOBI PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 1000
        RESIDUAL SCALING = NONE
    {endif}

```

```

        CONVERGENCE TOLERANCE = 1.000000e-14
    END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = CG
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-14
    End
{endif}

    BEGIN FINITE ELEMENT MODEL cube
        database name = cube_h{N}_tet10.e
        coordinate system is cartesian
        decomposition method = rcb

        BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
        END PARAMETERS FOR BLOCK block_1

    END FINITE ELEMENT MODEL cube

    Begin User Function exact_soln
        Load From File ./exact.so Using Function registerExactSoln
    End

    Begin User Function exact_soln_dot
        Load From File ./exact.so Using Function registerExactSolnDot
    End

    Begin User Function exact_src
        Load From File ./exact.so Using Function registerExactSrc
    End

    Begin User Function flux_surface_3
        Load From File ./exact.so Using Function registerFlux_Surface_3
    End

    Begin User Function flux_surface_5
        Load From File ./exact.so Using Function registerFlux_Surface_5
    End

    Begin User Function flux_surface_1
        Load From File ./exact.so Using Function registerFlux_Surface_1
    End

    Begin User Function flux_surface_2
        Load From File ./exact.so Using Function registerFlux_Surface_2
    End

    Begin Norm Postprocessor l2
        Use Function exact_soln
        Subtract Function nonlinear_solution->TEMPERATURE
        Compute Norms L2
    End

    Begin Norm Postprocessor h1
        Use Function exact_soln
        Subtract Function nonlinear_solution->TEMPERATURE
        Compute Norms H1
    End

    Begin Norm Postprocessor linf
        Use Function exact_soln
        Subtract Function nonlinear_solution->TEMPERATURE
        Compute Norms LInfinity

```

```

End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
Write To File errors_thermal_steady_tet4_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
  BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

```

```

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_tet4_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

  END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.6. Steady Heat Conduction: Tet10 Meshes

```

BEGIN SIERRA myJob

  load user plugin file ./exact.so

  BEGIN ARIA MATERIAL Kryptonite
    Density = Constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat = Constant cp=1
    heat conduction = basic
  END ARIA MATERIAL Kryptonite

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
      BEGIN SUPERLU SOLVER
      END
    END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Direct_Solver
    Solution Method = amesos-superlu
    End
  {endif}

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
      BEGIN CG SOLVER
        BEGIN JACOBI PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 1000
        RESIDUAL SCALING = NONE
        CONVERGENCE TOLERANCE = 1.000000e-14
      END
    END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Iterative_Solver
    Solution Method = CG
    Preconditioning Method = Jacobi
    Maximum Iterations = 1000
    Residual Norm Scaling = None
    Residual Norm Tolerance = 1.0e-14
    End
  {endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = cube_h{N}_tet10.e

```

```

coordinate system is cartesian
decomposition method = rcb

BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
END PARAMETERS FOR BLOCK block_1

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./exact.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
Write To File errors_thermal_steady_tet10_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block

```

```

    Advance myRegion
End
    Simulation Start Time = 0
    Simulation Termination Time = 1
End
    End

    BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q2 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

    Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = thermal_steady_tet10_h{N}.e
    at step 1, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

```

```
END PROCEDURE myAriaProcedure
```

```
END SIERRA myJob
```

## 12.1.7. Transient Heat Conduction: Hex8 Meshes

```
BEGIN SIERRA myJob
```

```
load user plugin file ./exact_transient.so
```

```
# L = { L = 10 }
```

```
# rho = { rho = 1 }
```

```
# Cp = { Cp = 1 }
```

```
BEGIN ARIA MATERIAL Kryptonite  
Density = Constant rho={rho}  
Thermal Conductivity = constant k=1  
Specific Heat = Constant cp={Cp}  
heat conduction = basic  
latent heat = constant value={L}  
END ARIA MATERIAL Kryptonite
```

```
{if(useTpetra)}
```

```
BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER  
BEGIN SUPERLU SOLVER  
END
```

```
END TPETRA EQUATION SOLVER
```

```
{else}
```

```
Begin Trilinos Equation Solver Direct_Solver  
Solution Method = amesos-superlu  
End
```

```
{endif}
```

```
{if(useTpetra)}
```

```
BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER  
BEGIN CG SOLVER  
BEGIN JACOBI PRECONDITIONER  
END  
MAXIMUM ITERATIONS = 1000  
RESIDUAL SCALING = NONE  
CONVERGENCE TOLERANCE = 1.000000e-12  
END
```

```
END TPETRA EQUATION SOLVER
```

```
{else}
```

```
Begin Trilinos Equation Solver Iterative_Solver  
Solution Method = CG  
Preconditioning Method = Jacobi  
Maximum Iterations = 1000  
Residual Norm Scaling = None  
Residual Norm Tolerance = 1.0e-12  
End
```

```
{endif}
```

```
BEGIN FINITE ELEMENT MODEL cube  
database name = cube_h{N}_hex8.e  
coordinate system is cartesian  
decomposition method = rcb
```

```
BEGIN PARAMETERS FOR BLOCK block_1
```

```
material Kryptonite
```

```
END PARAMETERS FOR BLOCK block_1
```

```
END FINITE ELEMENT MODEL cube
```

```
Begin User Function exact_soln
```

```
Load From File ./exact_transient.so Using Function registerExactSoln
```

```

End

Begin User Function exact_soln_dot
  Load From File ./exact_transient.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./exact_transient.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact_transient.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact_transient.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact_transient.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact_transient.so Using Function registerFlux_Surface_2
End

# Ts = { Ts = 0.5 }
# Tl = { Tl = 1.5 }
# Tm = { Tm = 0.5 * (Ts + Tl) }
# sigma = { sigma = 0.429858 * (Tm - Ts) }

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Transient The_Time_Block

```

```

    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 3
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.5/2**N}
    Time Integration Method = Second_Order
    Time Step Variation = fixed
  End
End
  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

  IC for temperature on block_1 = encore_function name=exact_soln

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3

  Begin Heat Flux Boundary Condition hfbc2
  Add Surface surface_5
    Flux = -5
  End

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1

  Begin Convective Flux Boundary Condition cfbc2
  Add Surface surface_2
    Convective Coefficient = 2
    Reference Temperature = 2
  End

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
  Begin Volume Heating vh1

```

```

        Add Volume block_1
        Value = 1
    End

    Source for Energy on block_1 = melting Ts={Ts} Tl={Tl}

Source For ENERGY on block_1 = Encore_Function Name=exact_src

    Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = output{N}.e
    at step 0, increment = {2**N}
    #at step 0, increment = 1
    title Aria cube test
    nodal variables = solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

    END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.8. Transient Heat Conduction: Tet4 Meshes

```

BEGIN SIERRA myJob

    load user plugin file ./exact_transient.so

    BEGIN ARIA MATERIAL Kryptonite
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END ARIA MATERIAL Kryptonite

    BEGIN ARIA MATERIAL Air
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END ARIA MATERIAL

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
            BEGIN SUPERLU SOLVER
            END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
        End
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
            BEGIN CG SOLVER
            BEGIN JACOBI PRECONDITIONER
            END
        END
    {endif}

```

```

        MAXIMUM ITERATIONS = 1000
        RESIDUAL SCALING = NONE
        CONVERGENCE TOLERANCE = 1.000000e-12
    END
END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = CG
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-12
    End
{endif}

BEGIN FINITE ELEMENT MODEL cube
    database name = cube_h{N}_tet4.e
    coordinate system is cartesian
    decomposition method = rcb

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
    Load From File ./exact_transient.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
    Load From File ./exact_transient.so Using Function registerExactSolnDot
End

Begin User Function exact_src
    Load From File ./exact_transient.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
    Load From File ./exact_transient.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
    Load From File ./exact_transient.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
    Load From File ./exact_transient.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
    Load From File ./exact_transient.so Using Function registerFlux_Surface_2
End

# T0 = { T0 = 2 }
# h = { h = 2 }
# rho = { rho = 1 }
# cp = { cp = 1 }
# omega = { omega = PI }
# bn_vol = { bn_vol = 0.5 }
Begin String Function bulk_node_exact_solution
    Value is "{T0} * (sin({omega} * t) + 1)"
End
Begin String Function bulk_node_source
    use function bulk_node_exact_solution as Tb
    Value is "{rho * cp * omega * T0} * cos({omega} * t) - ({h} * (1 - Tb))/{bn_vol}"
End
Begin String Function bulk_node_flux_bc_corr

```

```

        use function bulk_node_exact_solution as Tb
        Value is "({h} * (Tb - 2))"
    End

Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
    volumes block_1
End

Begin Norm Postprocessor l2_dot
    Use Function exact_soln_dot
    Subtract Function time_derivative_at_time->TEMPERATURE
    Compute Norms L2
    volumes block_1
End

Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
    volumes block_1
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
    volumes block_1
End

Begin Norm Postprocessor linf_bulk_node
    Use Function bulk_node_exact_solution
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms Nodal LInfinity
    volumes block_for_abulknode
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Transient The_Time_Block
        Advance myRegion
    End
        Simulation Start Time = 0
        Simulation Termination Time = 3
End
Begin Parameters For Transient The_Time_Block
    Begin Parameters For Aria Region myRegion
        Initial Time Step Size = {0.5/2**N}
        Time Integration Method = Second_Order
        Time Step Variation = fixed
    End
End
End

    BEGIN ARIA REGION myRegion

```

```

Nonlinear Solution Strategy = Newton
  Minimum Nonlinear Solves = 1
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  nonlinear residual minimum convergence rate = 0.999 number of steps = 3
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

  IC const on block_1 temperature = 1.0

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1

  Begin Bulk Fluid Element aBulkNode
    material = Air
    bulk element volume = constant v = {bn_vol}
    initial temperature = {T0}
    bulk eq energy for temperature using p0 with mass src
    bulk source for energy = encore_function name=bulk_node_source
  End
  Begin Convective Flux Boundary Condition bulk_flux
    add surface surface_2
    use bulk element aBulkNode
    convective coefficient = {h}
  End
  BC Flux for Energy on surface_2 = encore_function name=bulk_node_flux_bc_corr

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

  Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf
Evaluate Postprocessor linf_bulk_node

```

```

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = output{N}.e
    at step 0, increment = {2**N}
    title Aria cube test
    nodal variables = solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
    global variables = abulknode_T
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.9. Transient Heat Conduction: Tet4Tet10 Meshes

```

BEGIN SIERRA myJob

    load user plugin file ./exact_transient.so

    BEGIN ARIA MATERIAL Kryptonite
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END ARIA MATERIAL Kryptonite

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
            BEGIN SUPERLU SOLVER
                END
            END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
        End
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
            BEGIN CG SOLVER
                BEGIN JACOBI PRECONDITIONER
                    END
                MAXIMUM ITERATIONS = 1000
                RESIDUAL SCALING = NONE
                CONVERGENCE TOLERANCE = 1.000000e-12
            END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = CG
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-12
        End
    {endif}

    BEGIN FINITE ELEMENT MODEL cube
        database name = cube_h{N}_tet10.e
        coordinate system is cartesian
        decomposition method = rcb

        BEGIN PARAMETERS FOR BLOCK block_1

```

```

material Kryptonite
  END PARAMETERS FOR BLOCK block_1

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact_transient.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./exact_transient.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./exact_transient.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact_transient.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact_transient.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact_transient.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact_transient.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main

```

```

Begin System Main
  Begin Transient The_Time_Block
    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 3
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.5/2**N}
    Time Integration Method = Second_Order
    Time Step Variation = fixed
  End
End
  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

  IC const on block_1 temperature = 1.0

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
  BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

  Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot

```

```

Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = output{N}.e
    at step 0, increment = {2**N}
    title Aria cube test
    nodal variables = solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.1.10. Transient Heat Conduction: Tet10 Meshes

```

BEGIN SIERRA myJob

    load user plugin file ./exact_transient.so

    BEGIN ARIA MATERIAL Kryptonite
        Density          = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat     = Constant cp=1
        heat conduction   = basic
    END ARIA MATERIAL Kryptonite

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
            BEGIN SUPERLU SOLVER
                END
            END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Direct_Solver
            Solution Method = amesos-superlu
        End
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
            BEGIN CG SOLVER
                BEGIN JACOBI PRECONDITIONER
                    END
                MAXIMUM ITERATIONS = 1000
                RESIDUAL SCALING = NONE
                CONVERGENCE TOLERANCE = 1.000000e-12
            END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Iterative_Solver
            Solution Method = CG
            Preconditioning Method = Jacobi
            Maximum Iterations = 1000
            Residual Norm Scaling = None
            Residual Norm Tolerance = 1.0e-12
        End
    {endif}

    BEGIN FINITE ELEMENT MODEL cube
        database name = cube_h{N}_tet10.e
        coordinate system is cartesian
        decomposition method = rcb

```

```

        BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
        END PARAMETERS FOR BLOCK block_1

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact_transient.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./exact_transient.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./exact_transient.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact_transient.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact_transient.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact_transient.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact_transient.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description

```

```

Use System Main
Begin System Main
  Begin Transient The_Time_Block
    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 3
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.5/2**N}
    Time Integration Method = Second_Order
    Time Step Variation = fixed
  End
End
End

      BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q2 with MASS DIFF SRC

  IC const on block_1 temperature = 1.0

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
  BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

  Output Number of Nodes

Evaluate Postprocessor 12

```

```
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = output{N}.e
    at step 0, increment = {2**N}
    title Aria cube test
    nodal variables = solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

    END PROCEDURE myAriaProcedure

END SIERRA myJob
```

## 12.2. THERMAL BOUNDARY CONDITIONS

### 12.2.1. Radiative Heat Flux 3.1

```
BEGIN SIERRA Aria

Title Radiation Form Factor Flux User_Sub

Begin Global Constants
  Stefan Boltzmann constant = 5.6704E-8
End Global Constants

load user plugin file ./FormFactor.so
load user plugin file ./DirichletBC.so

Begin User Function exact_soln
  Load From File ./Exact_Solution.so Using Function registerExactSoln
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

Begin Aria Material mat1
  density = constant rho = 0.1
  thermal conductivity = constant k = 1.0
  specific heat = constant cp = 1.0
  heat conduction = basic
End Aria Material mat1

Begin Aria Material mat_s1
  emissivity = constant e = 0.8
  bc rad reference temperature = constant t_ref = 500
  radiation form factor = calore_user_sub name = form_factor type = element
End

Begin Finite Element Model myModel
  Database Name = mesh{N}.g
  Coordinate System = Cartesian
  decomposition method = rcb
  Database Type = EXODUSII
  Use Material mat1 for block_1
  Use Material mat_s1 for surface_1
End Finite Element Model myModel
```

```

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
    BEGIN CG SOLVER
      BEGIN JACOBI PRECONDITIONER
      END
      MAXIMUM ITERATIONS = 1000
      RESIDUAL SCALING = R0
      CONVERGENCE TOLERANCE = 1.000000e-12
    END
  END TPETRA EQUATION SOLVER
{else}
  begin aztec equation solver solve_temperature
    solution method = cg
    preconditioning method = jacobi
    maximum iterations = 1000
    residual norm tolerance = 1.0e-12
    residual norm scaling = r0
  end aztec equation solver solve_temperature
{endif}

begin procedure myProcedure

  Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential MySolveBlock
    Advance myRegion
  End
  End
End

begin Aria region myRegion

  Use Finite Element Model myModel
  Use Linear Solver solve_temperature

  Nonlinear Solution Strategy = Newton

  Maximum nonlinear iterations = 10
  Nonlinear residual tolerance = 1.0e-10
  Nonlinear correction tolerance = 1.0e-10
  Nonlinear relaxation factor = 1.0

  EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF

  BC dirichlet for temperature on surface_3 = calore_user_sub name = localCoord_bc type=node

#   Begin Radiative Flux Boundary Condition fraction
#   Add surface surface_1
#   Emissivity = 0.8
#   Reference Temperature = 500.0
#   Radiation Form Factor Subroutine = form_factor
#   End

  BC Flux for Energy at surface_1 = generalized_rad

  BC const DIRICHLET at surface_2 temperature = 600.0
  BC const DIRICHLET at surface_4 temperature = 600.0

  Output Number of Nodes
  Evaluate Postprocessor l2
  Evaluate Postprocessor h1
  Evaluate Postprocessor linf

  Begin Results Output Label diffusion output
  database Name = output{N}.e
  At Step 1, Increment = 1
  Timestep Adjustment Interval = 1

```

```

        Title Radiative Flux BC User Sub Test
        Nodal Variables = solution->temperature as T
        End Results Output Label diffusion output

    end Aria region myRegion

end procedure myProcedure

end sierra Aria

```

## 12.2.2. Radiative Heat Flux From Fortran User Subroutine

```

begin sierra FandI_VnVtest

    title Verification of Fire and Ice BC subroutine, AKA Directed Heating User Sub \$
    Simplified model with sidesets to check that BCs are applied to faces specified \$

Load User Plugin File ./FireAndIceBC.so USING function conv_subs_register

#####
##### Material property definitions #####
#####

begin aria material VnVmat
    heat conduction = basic
    density = constant rho = 8000.0 # Approximate value for VnV study
    emissivity = constant e = 0.30 # Approximate value for VnV study
    specific heat = constant cp = 550 # Approximate value for VnV study
    thermal conductivity = constant k = 20 # Approximate value for VnV study
end aria material VnVmat

#####
##### UPDATE THE FINITE ELEMENT MODEL #####
#####

begin finite element model fem
    database name = VnVmesh2.g
    database type = exodusII
    use material VnVmat for block_1

    # - Block id 10 had name 10
    use material VnVmat for block_10

    # - Block id 11 had name 11
    use material VnVmat for block_11

    # - Block id 12 had name 12
    use material VnVmat for block_12

    # - Block id 13 had name 13
    use material VnVmat for block_13

    # - Block id 2 had name 2
    use material VnVmat for block_2

    # - Block id 3 had name 3
    use material VnVmat for block_3

    # - Block id 4 had name 4
    use material VnVmat for block_4

    # - Block id 5 had name 5
    use material VnVmat for block_5

    # - Block id 6 had name 6
    use material VnVmat for block_6

```

```

# - Block id 7 had name 7
use material VnVmat for block_7

# - Block id 8 had name 8
use material VnVmat for block_8

# - Block id 9 had name 9
use material VnVmat for block_9
end finite element model fem

#####

begin global constants
  stefan boltzmann constant = 5.67e-8
end global constants

#####

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER  TRILINOS SOLVE
  BEGIN GMRES SOLVER
  BEGIN DD-ILUT PRECONDITIONER
  DROP TOLERANCE = 0
  FILL FRACTION = 5.000000e+00
  END
  MAXIMUM ITERATIONS = 1000
  RESTART ITERATIONS = 100
  RESIDUAL SCALING = R0
  CONVERGENCE TOLERANCE = 1.000000e-12
  END
  MATRIX SCALING = ONE_NORM
END TPETRA EQUATION SOLVER
{else}
  Begin TRILINOS Equation Solver trilinos_solve
  Solution Method = GMRES
  Preconditioning Method = DD-ILUT
  Maximum Iterations = 1000
  Matrix Scaling = row-sum
  Residual Norm Tolerance = 1.0e-12
  Residual Norm Scaling = R0
  Restart Iterations is 100
  preconditioning steps is 1
  Param-Real AZ_ilut_fill value 5.0
  polynomial order = 300
  ilu threshold = 1.0e-6
  End TRILINOS Equation Solver trilinos_solve
{endif}

begin procedure aria_procedure

  begin solution control description

    Use System Main

    Begin System Main
      Simulation Max Global Iterations = 10000
      Simulation Start Time           = 0.0
      #Simulation Termination Time     = 3600.0

    Begin Transient Main
      Advance myRegion
    End

  End

  Begin Parameters For Transient Main
    Start Time = 0.0

```

```

Termination Time = 0.1

Begin Parameters for Aria Region myRegion
# -----#
# Time Integration Method = Second_Order #Second_Order
# -----#
Time Step Variation      = Adaptive
Initial Time Step Size  = 0.01
Minimum Time Step Size  = 0.01
Maximum Time Step Size  = 0.01
#Minimum Resolved Time Step Size = 0.001
Predictor-Corrector Tolerance = 1.0e-08
End
End

End #solution control

#####

begin aria region myRegion
# solve energy equation for temperature at the nodes (1st order) with diffusion
EQ ENERGY for TEMPERATURE on all_blocks using Q1 with lumped_mass Diff

use finite element model fem

nonlinear solution strategy = newton
use dof averaged nonlinear residual
accept solution after maximum nonlinear iterations = true
use linear solver trilinos_solve
nonlinear relaxation factor = 1.0

nonlinear residual tolerance = 1.0e-10 # for transient
maximum nonlinear iterations = 10 # for transient
#nonlinear residual tolerance = 1.0e-15 # for steady state
#maximum nonlinear iterations = 150 # for steady state

#####
##### Initial Conditions #####
#####

IC Const on all_blocks temperature = 300.0

#####
##### Convective Boundary Conditions #####
#####
#
begin convective flux boundary condition FireAndIceBC

add surface surface_1 surface_2 surface_3 surface_4
add surface surface_5 surface_6 surface_7 surface_8
add surface surface_9 surface_10 surface_11 surface_12
add surface surface_13 surface_14 surface_15 surface_16
add surface surface_17 surface_18 surface_19 surface_20
add surface surface_21 surface_22 surface_23 surface_24
add surface surface_25 # All external surfaces

# User Sub Integer Input Constants:
# cosdistA=idat(1), for x < xA
# cosdistAB=idat(2), for xA <= x <= xB
# cosdistB=idat(3), for x > xB

# User Sub Real Input Constants:
# xoffset=rdat(1), such that abs(xnosetip-xoffset)=0
# xA=rdat(2), distance from nosetip to position A (xA>0)
# xB=rdat(3), distance from nosetip to position B (xB>0)
# hA1=rdat(4), convective htc for x < xA for azimuthal section 1
# hAB1=rdat(5), convective htc for xA <= x <= xB for section 1
# hB1=rdat(6), convective htc for x > xB for azimuthal section 1

```

```

# TrefA1=rdat(7), Tref for x < xA for azimuthal section 1
# TrefAB1=rdat(8), Tref for xA <= x <= xB for azimuthal section 1
# TrefB1=rdat(9), Tref for x > xB for azimuthal section 1
# emisA1=rdat(10), emis for x < xA for azimuthal section 1
# emisAB1=rdat(11), emis for xA <= x <= xB for azimuthal section 1
# emisB1=rdat(12), emis for x > xB for azimuthal section 1
# hA2=rdat(13), convective htc for x < xA for azimuthal section 2
# hAB2=rdat(14), convective htc for xA <= x <= xB for azimuthal section 2
# hB2=rdat(15), convective htc for x > xB for azimuthal section 2
# TrefA2=rdat(16), Tref for x < xA for azimuthal section 2
# TrefAB2=rdat(17), Tref for xA <= x <= xB for azimuthal section 2
# TrefB2=rdat(18), Tref for x > xB for azimuthal section 2
# emisA2=rdat(19), emis for x < xA for azimuthal section 2
# emisAB2=rdat(20), emis for xA <= x <= xB for azimuthal section 2
# emisB2=rdat(21), emis for x > xB for azimuthal section 2
# thetaA=rdat(22), azimuthal reference angle (degrees) for section 1 of region A
# thetaAB=rdat(23), azimuthal reference angle (degrees) for section 1 of region AB
# thetaB=rdat(24), azimuthal reference angle (degrees) for section 1 of region B
# dphiA=rdat(25), subtended angle (degrees) for section 1 of region A
# dphiAB=rdat(26), subtended angle (degrees) for section 1 of region AB
# dphiB=rdat(27), subtended angle (degrees) for section 1 of region B
# notes:
# if cosdistA (or AB,or B) set to 1 then impose cosine distribution on radiative htc
# otherwise, distribution is uniform (convective distribution is always uniform)
# x coordinate assumed to lie on centerline of bomb
# htc_total = convective htc + effective radiative htc
# effective radiative htc = sigma*emis*(Twall+Tref)*(Twall^2+Tref^2)
# emissivities should be the same as associated material emissivities, but
# can be set to zero to eliminate radiative heat transfer from a region
# set convective htc to zero to eliminate convective heat transfer from a region
# set both convective htc and emis to zero for an adiabatic (insulated) surface
# a zero-degree angle corresponds to the y axis
# angle is positive in clockwise direction when looking in the positive x axis direction
# theta is the angle of the center of the azimuthal section and dphi is the delta angle
# with section 1 extending from theta-dphi/2 to theta+dphi/2, centered on theta
# section 2 is opposite section 1 and can be empty if dphi = 360 degrees.

convective coefficient fortran subroutine is coef_directed_angle
reference temperature fortran subroutine is tref_directed_angle
integer data 0 0 0
# xoffset xA xB
# hA1 hAB1 hB1
# TrefA1 TrefAB1 TrefB1
# emisA1 emisAB1 emisB1
# hA2 hAB2 hB2
# TrefA2 TrefAB2 TrefB2
# emisA2 emisAB2 emisB2
# thetaA thetaAB thetaB
# dphiA dphiAB dphiB
real data -1.5 0.5 1.0 \$$
0.0 50.0 0 \$$
300.0 1000.0 300.0 \$$
0.0 0.8 0.0 \$$
0.0 0.0 100.0 \$$
300.0 300.0 900.0 \$$
0.0 0.0 0.0 \$$
0.0 300.0 240. \$$
0.0 120.0 240.0

#
# xA xB azimuthal section 1 -dphi/2 ^ ^ |
# A | AB | B axial (x axis) regions | y| | V
# ===== surfaces | _ _ > x | _ _ > z
# | | |
# xA xB azimuthal section 2 dphi/2 z out of page x into page
#

integrated power output qFireIce
integrated flux output fluxFireIce

```

```

end convective flux boundary condition FireAndIceBC

#####
##### Results #####
#####
#####

Begin user variable HTC
type is face real length = 1
initial value = 0.0
add part surface_1 surface_2 surface_3 surface_4
add part surface_5 surface_6 surface_7 surface_8
add part surface_9 surface_10 surface_11 surface_12
add part surface_13 surface_14 surface_15 surface_16
add part surface_17 surface_18 surface_19 surface_20
add part surface_21 surface_22 surface_23 surface_24
add part surface_25
End

Begin user variable SurfFlux
type is face real length = 1
initial value = 0.0
add part surface_1 surface_2 surface_3 surface_4
add part surface_5 surface_6 surface_7 surface_8
add part surface_9 surface_10 surface_11 surface_12
add part surface_13 surface_14 surface_15 surface_16
add part surface_17 surface_18 surface_19 surface_20
add part surface_21 surface_22 surface_23 surface_24
add part surface_25
End

Begin user variable Tref #for checking only
type is face real length = 1
initial value = 0.0
add part surface_1 surface_2 surface_3 surface_4
add part surface_5 surface_6 surface_7 surface_8
add part surface_9 surface_10 surface_11 surface_12
add part surface_13 surface_14 surface_15 surface_16
add part surface_17 surface_18 surface_19 surface_20
add part surface_21 surface_22 surface_23 surface_24
add part surface_25
End

Begin Results Output NodalTdata
Title VnVtest Nodal Temperature Data
database name = VnVinputTest.e
Nodal Variables = solution->temperature as T
#nodal variables = temperatureredot as Tdot
#Face Variables = HTC SurfFlux Tref #costheta
Global Variables = time_step as timestep

# Global Variables = PEinterior_T as PEinterior_T

Timestep Adjustment Interval is 1
At Time 0.0, Increment = 0.01 #

End

#####

end aria region myRegion

end procedure aria_procedure

end sierra FandI_VnVtest

```

## 12.2.3. Convective Heat Flux 3.3

```
Begin SIERRA Aria

  load user plugin file ./Exact_Solution.so

  Begin User Function exact_soln
    Load From File ./Exact_Solution.so Using Function registerExactSoln
  End

  Begin Norm Postprocessor l2_norm
    Use Function exact_soln
    Subtract Function nonlinear_solution->temperature
    Compute Norms L2
  End

  Begin Norm Postprocessor h1_norm
    Use Function exact_soln
    Subtract Function nonlinear_solution->temperature
    Compute Norms H1
  End

  Begin Norm Postprocessor linf_norm
    Use Function exact_soln
    Subtract Function nonlinear_solution->temperature
    Compute Norms LInfinity
  End

  Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
  End

  load user plugin file ./FluxBC.so
  load user plugin file ./DirichletBC.so
  load user plugin file ./Init.so

  Begin Aria Material M_Block
    density          = constant rho = 1.
    specific heat    = constant cp = 1.
    heat conduction  = basic
    Thermal conductivity = constant k = 1.
  End

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
      BEGIN GMRES SOLVER
        BEGIN DD-ILU PRECONDITIONER
          END
          MAXIMUM ITERATIONS = 1000
          RESIDUAL SCALING = NONE
          CONVERGENCE TOLERANCE = 1.000000e-12
        END
      END TPETRA EQUATION SOLVER
  {else}
    Begin AZTEC Equation Solver solve_temperature
      Solution Method = gmres
      Preconditioning Method = dd-ilu
      Maximum Iterations = 1000
      Residual Norm Tolerance = 1e-12
      Residual Norm Scaling = NONE
    End
  {endif}

  Begin Finite Element Model myModel
```

```

Database Name = mesh{N}.g
Coordinate System = cartesian
decomposition method = rcb
Use Material M_Block for block_1
End

Begin procedure myProcedure

Begin solution control description
Use System Main
Begin System Main
Simulation Start Time          = 0.0
Simulation Termination Time    = 0.1
Simulation Max Global Iterations = 100
Begin Transient Time_Block
advance myRegion
End
End System Main

Begin parameters for transient Time_Block
Start Time = 0.0
Begin parameters for aria Region myRegion
time step variation = fixed # adaptive
initial time step size = {0.008*0.5**(N)}
time integration method = second_order
predictor-corrector tolerance = 1.0E-5
End
End
End Solution Control Description

begin aria region myRegion

Use Finite Element Model myModel
Use Linear Solver solve_temperature

nonlinear solution strategy = newton
nonlinear residual tolerance = 1.0e-10
maximum nonlinear iterations = 10
Nonlinear Relaxation Factor = 1.0

EQ energy for temperature on block_1 using Q1 with diff mass

BC const dirichlet on surface_2 Temperature = 0.0

Begin Temperature Boundary Condition s4
Add Surface surface_4
Temperature = 0.0
End

BC dirichlet for temperature on surface_3 = calore_user_sub name = localCoord_bc type=node
IC for temperature on block_1 = calore_user_sub name = localCoord_ic type=node

Output Number of Nodes
Evaluate Postprocessor l2_norm
Evaluate Postprocessor h1_norm
Evaluate Postprocessor linf_norm

Begin Convective Flux Boundary Condition internal
Add Surface surface_1 #y=1, 0<x<1, normal=(0,1)
Reference Temperature Subroutine = tref_coeff
Convective Coefficient Subroutine = convec_coeff
End

Begin Results Output output
Database Name = output{N}.e
AT STEP 0, INCREMENT = {2**(N)}
TITLE Aria Heat Convective Flux BC Condition
Nodal Variables = nonlinear_solution->temperature as T

```

```
End  
end aria region myRegion  
End procedure myProcedure  
End sierra Aria
```

## 12.3. THERMAL CONTACT

### 12.3.1. 1D Flat Contact 4.1

#### 12.3.1.1. Hex8 Tied

```
#{R=0.0}
begin sierra Aria

  title Adaptive Square

  load user plugin file ./Exact_solution.so

  BEGIN ARIA MATERIAL M1
    Density          = Constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat     = Constant cp=1
    heat conduction   = basic
  END

  BEGIN ARIA MATERIAL M2
    Density          = Constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat     = Constant cp=1
    heat conduction   = basic
  END

  Begin Finite Element Model bar
    Database Name = 2blocks_contact_unaligned_hex8_h{N}.g
    Begin parameters for block block_1
      material M1
    End
    Begin parameters for block block_2
      material M2
    End
  End Finite Element Model bar

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER  DIRECT_SOLVER
    BEGIN UMFPAK SOLVER
    END
    END TPETRA EQUATION SOLVER
  {else}
    begin trilinos equation solver direct_solver
    solution method = amesos-umfpack
    end trilinos equation solver direct_solver
  {endif}

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER  ITERATIVE_SOLVER
    BEGIN GMRES SOLVER
    BEGIN JACOBI PRECONDITIONER
    END
    MAXIMUM ITERATIONS = 20000
    RESIDUAL SCALING = R0
    CONVERGENCE TOLERANCE = 1.000000e-12
    END
    END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Iterative_Solver
    Solution Method = GMRES
    Preconditioning Method = Jacobi
    Maximum Iterations = 20000
    Residual Norm Scaling = R0
    Residual Norm Tolerance = 1.0e-12
    End
  }
```

```

{endif}

Begin Field Function ffunc
  Use Nodal Field solution->temperature As Value
End

Begin User Function exact_soln
  Load From File ./Exact_solution.so Using Function registerExactSolution
  Parameter R = {R}
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms L2
  Store in l2_err
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms LInfinity
  Store in linf_err
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms H1
  Store in h1_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_tied_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

begin procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential The_Time_Block
        Advance myRegion
      End
      Simulation Start Time = 0
      Simulation Termination Time = 1
    End
  End

  begin Aria region myRegion

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

    use finite element model bar
    use linear solver Iterative_Solver #Direct_Solver #

    EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

    Begin TEMPERATURE BOUNDARY CONDITION xm1
      add surface surface_1
    End
  end
end

```

```

    TEMPERATURE = 0.
End

Begin TEMPERATURE BOUNDARY CONDITION xp1
    add surface surface_2
    TEMPERATURE = 1.
End

Begin VOLUME HEATING sm
    add volume block_1
    VALUE = -1.
End

Begin VOLUME HEATING sp
    add volume block_2
    VALUE = 1.
End

begin contact definition res1
    contact surface surf_1 contains surface_3
    contact surface surf_2 contains surface_4

    begin interaction inter_1
        surfaces = surf_1 surf_2
    end interaction inter_1

    begin enforcement enf_1
        Enforcement for Energy = Tied_Temperature
    end enforcement

end contact definition res1

Output Number of Nodes
Evaluate Postprocessor l2
Evaluate Postprocessor linf
Evaluate Postprocessor h1

Begin Results Output Label diffusion output
    database Name = 2blocks_tied_h{N}.e
    At Step 1, Increment = 1
    Title Calore Two Blocks
    Nodal Variables = solution->temperature as T
    Element Variables = l2_err linf_err h1_err
End

end

end

end

```

### 12.3.1.2. *Hex8 Resistance*

```

#{R=4.0}
begin sierra Aria

    title Adaptive Square

    BEGIN ARIA MATERIAL M1
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END

    BEGIN ARIA MATERIAL M2

```

```

        Density          = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat      = Constant cp=1
        heat conduction    = basic
    END

    Begin Finite Element Model bar
        Database Name = 2blocks_contact_unaligned_hex8_h{N}.g
        Begin parameters for block block_1
            material M1
        End
        Begin parameters for block block_2
            material M2
        End
    End Finite Element Model bar

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN UMFPACK SOLVER
        END
    END TPETRA EQUATION SOLVER
    {else}
        begin trilinos equation solver direct_solver
        solution method = amesos-umfpack
        end trilinos equation solver direct_solver
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN GMRES SOLVER
        BEGIN JACOBI PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 20000
        RESIDUAL SCALING = R0
        CONVERGENCE TOLERANCE = 1.000000e-12
        END
    END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = GMRES
        Preconditioning Method = Jacobi
        Maximum Iterations = 20000
        Residual Norm Scaling = R0
        Residual Norm Tolerance = 1.0e-12
        End
    {endif}

    Begin Field Function ffunc
        Use Nodal Field solution->temperature As Value
    End

    Begin User Function exact_soln
        Load From File ./Exact_solution.so Using Function registerExactSolution
        Parameter R = {R}
    End

    Begin Norm Postprocessor l2
        Use Function exact_soln
        Subtract Function ffunc
        Compute Norms L2
        Store in l2_err
    End

    Begin Norm Postprocessor linf
        Use Function exact_soln
        Subtract Function ffunc
        Compute Norms LInfinity
        Store in linf_err

```

```

End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms H1
  Store in h1_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_res_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

begin procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential The_Time_Block
        Advance myRegion
      End
      Simulation Start Time = 0
      Simulation Termination Time = 1
    End
  End

  begin Aria region myRegion

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

    use finite element model bar
    use linear solver Iterative_Solver #Direct_Solver #

    EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

    Begin TEMPERATURE BOUNDARY CONDITION xm1
      add surface surface_1
      TEMPERATURE = 0.
    End

    Begin TEMPERATURE BOUNDARY CONDITION xp1
      add surface surface_2
      TEMPERATURE = 1.
    End

    Begin VOLUME HEATING sm
      add volume block_1
      VALUE = -1.
    End

    Begin VOLUME HEATING sp
      add volume block_2
      VALUE = 1.
    End

    begin contact definition res1
      contact surface surf_1 contains surface_3
      contact surface surf_2 contains surface_4

      begin interaction inter_1

```

```

        surfaces = surf_1 surf_2
    end interaction inter_1

    begin enforcement enf_1
        conductance coefficient= {1.0/R}
        Enforcement for Energy = gap_conductance
    end enforcement

end contact definition res1

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor linf
Evaluate Postprocessor h1

Begin Results Output Label diffusion output
    database Name = 2blocks_res_h{N}.e
    At Step 1, Increment = 1
    Title Calore Two Blocks
    Nodal Variables = solution->temperature as T
    Element Variables = l2_err linf_err h1_err
End

end

end

end

```

### 12.3.1.3. Tet4 Tied

```

#{R=0.0}
begin sierra Aria

    title Adaptive Square

    load user plugin file ./Exact_solution.so

    BEGIN ARIA MATERIAL M1
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END

    BEGIN ARIA MATERIAL M2
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END

    Begin Finite Element Model bar
        Database Name = 2blocks_contact_unaligned_tet4_h{N}.g
        Begin parameters for block block_1
            material M1
        End
        Begin parameters for block block_2
            material M2
        End
    End Finite Element Model bar

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN UMFPACK SOLVER
    }

```

```

        END
    END TPETRA EQUATION SOLVER
{else}
    begin trilinos equation solver direct_solver
        solution method = amesos-umfpack
    end trilinos equation solver direct_solver
{endif}

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN GMRES SOLVER
            BEGIN DD-ILU PRECONDITIONER
                END
                MAXIMUM ITERATIONS = 200
                RESIDUAL SCALING = R0
                CONVERGENCE TOLERANCE = 1.000000e-12
            END
        END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = GMRES
        Preconditioning Method = dd-ilu
        Maximum Iterations = 200
        Residual Norm Scaling = R0
        Residual Norm Tolerance = 1.0e-12
    End
{endif}

    Begin Field Function ffunc
        Use Nodal Field solution->temperature As Value
    End

    Begin User Function exact_soln
        Load From File ./Exact_solution.so Using Function registerExactSolution
        Parameter R = {R}
    End

    Begin Norm Postprocessor l2
        Use Function exact_soln
        Subtract Function ffunc
        Compute Norms L2
        Store in l2_err
    End

    Begin Norm Postprocessor linf
        Use Function exact_soln
        Subtract Function ffunc
        Compute Norms LInfinity
        Store in linf_err
    End

    Begin Norm Postprocessor h1
        Use Function exact_soln
        Subtract Function ffunc
        Compute Norms H1
        Store in h1_err
    End

    Begin Postprocessor Output Control pp_out
        Comment Character Is %
        Write To File errors_tied_h{N}.dat
        Floating Point Precision Is 3
        Floating Point Format Is Scientific
    End

    begin procedure myProcedure

        Begin Solution Control Description

```

```

Use System Main
Begin System Main
  Begin Sequential The_Time_Block
    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
End

begin Aria region myRegion

  Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

  use finite element model bar
  use linear solver Iterative_Solver #Direct_Solver #

  EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

  Begin TEMPERATURE BOUNDARY CONDITION xm1
    add surface surface_1
    TEMPERATURE = 0.
  End

  Begin TEMPERATURE BOUNDARY CONDITION xp1
    add surface surface_2
    TEMPERATURE = 1.
  End

  Begin VOLUME HEATING sm
    add volume block_1
    VALUE = -1.
  End

  Begin VOLUME HEATING sp
    add volume block_2
    VALUE = 1.
  End

  begin contact definition res1
    contact surface surf_1 contains surface_3
    contact surface surf_2 contains surface_4

    begin interaction inter_1
      surfaces = surf_1 surf_2
    end interaction inter_1

    begin enforcement enf_1
      Enforcement for Energy = Tied_Temperature
    end enforcement

  end contact definition res1

  Output Number of Nodes

  Evaluate Postprocessor l2
  Evaluate Postprocessor linf
  Evaluate Postprocessor h1

  Begin Results Output Label diffusion output
    database Name = 2blocks_tied_h{N}.e
    At Step 1, Increment = 1
    Title Calore Two Blocks

```

```

        Nodal Variables = solution->temperature as T
        Element Variables = l2_err linf_err h1_err
    End

    end

end

end

```

#### 12.3.1.4. *Tet4 Resistance*

```

#{R=4.0}
begin sierra Aria

    title Adaptive Square

    BEGIN ARIA MATERIAL M1
        Density          = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat     = Constant cp=1
        heat conduction   = basic
    END

    BEGIN ARIA MATERIAL M2
        Density          = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat     = Constant cp=1
        heat conduction   = basic
    END

    Begin Finite Element Model bar
        Database Name = 2blocks_contact_unaligned_tet4_h{N}.g
        Begin parameters for block block_1
            material M1
        End
        Begin parameters for block block_2
            material M2
        End
    End Finite Element Model bar

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN UMFPACK SOLVER
        END
        END TPETRA EQUATION SOLVER
    {else}
        begin trilinos equation solver direct_solver
        solution method = amesos-umfpack
        end trilinos equation solver direct_solver
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN GMRES SOLVER
        BEGIN DD-ILU PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 200
        RESIDUAL SCALING = R0
        CONVERGENCE TOLERANCE = 1.000000e-12
        END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = GMRES
        Preconditioning Method = dd-ilu
    }
}

```

```

        Maximum Iterations = 200
        Residual Norm Scaling = R0
        Residual Norm Tolerance = 1.0e-12
    End
{endif}

Begin Field Function ffunc
    Use Nodal Field solution->temperature As Value
End

Begin User Function exact_soln
    Load From File ./Exact_solution.so Using Function registerExactSolution
    Parameter R = {R}
End

Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function ffunc
    Compute Norms L2
    Store in l2_err
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function ffunc
    Compute Norms LInfinity
    Store in linf_err
End

Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function ffunc
    Compute Norms H1
    Store in h1_err
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors_res_h{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

begin procedure myProcedure

    Begin Solution Control Description
        Use System Main
        Begin System Main
            Begin Sequential The_Time_Block
                Advance myRegion
            End
            Simulation Start Time = 0
            Simulation Termination Time = 1
        End
    End

begin Aria region myRegion

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

    use finite element model bar
    use linear solver Iterative_Solver #Direct_Solver #

```

```

EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

Begin TEMPERATURE BOUNDARY CONDITION xm1
  add surface surface_1
  TEMPERATURE = 0.
End

Begin TEMPERATURE BOUNDARY CONDITION xp1
  add surface surface_2
  TEMPERATURE = 1.
End

Begin VOLUME HEATING sm
  add volume block_1
  VALUE = -1.
End

Begin VOLUME HEATING sp
  add volume block_2
  VALUE = 1.
End

begin contact definition res1
  contact surface surf_1 contains surface_3
  contact surface surf_2 contains surface_4

  begin interaction inter_1
    surfaces = surf_1 surf_2
  end interaction inter_1

  begin enforcement enf_1
    conductance coefficient= {1.0/R}
    Enforcement for Energy = gap_conductance
  end enforcement

end contact definition res1

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor linf
Evaluate Postprocessor h1

Begin Results Output Label diffusion output
  database Name = 2blocks_res_h{N}.e
  At Step 1, Increment = 1
  Title Calore Two Blocks
  Nodal Variables = solution->temperature as T
  Element Variables = l2_err linf_err h1_err
End

end

end

end

```

### 12.3.1.5. *Hex8-Tet4 Tied*

```

#{R=0.0}
begin sierra Aria

  title Adaptive Square

  load user plugin file ./Exact_solution.so

```

```

BEGIN ARIA MATERIAL M1
  Density          = Constant rho=1
  Thermal Conductivity = constant k=1
  Specific Heat    = Constant cp=1
  heat conduction  = basic
END

BEGIN ARIA MATERIAL M2
  Density          = Constant rho=1
  Thermal Conductivity = constant k=1
  Specific Heat    = Constant cp=1
  heat conduction  = basic
END

Begin Finite Element Model bar
  Database Name = 2blocks_contact_unaligned_hex8_tet4_h{N}.g
  Begin parameters for block block_1
    material M1
  End
  Begin parameters for block block_2
    material M2
  End
End Finite Element Model bar

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER  DIRECT_SOLVER
  BEGIN UMFPAK SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
  begin trilinos equation solver direct_solver
  solution method = amesos-umfpack
  end trilinos equation solver direct_solver
{endif}

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER  ITERATIVE_SOLVER
  BEGIN GMRES SOLVER
  BEGIN DD-ILU PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 200
  RESIDUAL SCALING = R0
  CONVERGENCE TOLERANCE = 1.000000e-12
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Iterative_Solver
  Solution Method = GMRES
  Preconditioning Method = dd-ilu
  Maximum Iterations = 200
  Residual Norm Scaling = R0
  Residual Norm Tolerance = 1.0e-12
  End
{endif}

Begin Field Function ffunc
  Use Nodal Field solution->temperature As Value
End

Begin User Function exact_soln
  Load From File ./Exact_solution.so Using Function registerExactSolution
  Parameter R = {R}
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms L2

```

```

    Store in l2_err
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms LInfinity
  Store in linf_err
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms H1
  Store in h1_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_tied_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

begin procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential The_Time_Block
        Advance myRegion
      End
      Simulation Start Time = 0
      Simulation Termination Time = 1
    End
  End

begin Aria region myRegion

  Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

  use finite element model bar
  use linear solver Iterative_Solver #Direct_Solver #

  EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

  Begin TEMPERATURE BOUNDARY CONDITION xm1
    add surface surface_1
    TEMPERATURE = 0.
  End

  Begin TEMPERATURE BOUNDARY CONDITION xp1
    add surface surface_2
    TEMPERATURE = 1.
  End

  Begin VOLUME HEATING sm
    add volume block_1
    VALUE = -1.
  End

  Begin VOLUME HEATING sp
    add volume block_2

```

```

        VALUE = 1.
    End

    begin contact definition res1
        contact surface surf_1 contains surface_3
        contact surface surf_2 contains surface_4

        begin interaction inter_1
            surfaces = surf_1 surf_2
        end interaction inter_1

        begin enforcement enf_1
            Enforcement for Energy = Tied_Temperature
        end enforcement

    end contact definition res1

    Output Number of Nodes

    Evaluate Postprocessor l2
    Evaluate Postprocessor linf
    Evaluate Postprocessor h1

    Begin Results Output Label diffusion output
        database Name = 2blocks_tied_h{N}.e
        At Step 1, Increment = 1
        Title Calore Two Blocks
        Nodal Variables = solution->temperature as T
        Element Variables = l2_err linf_err h1_err
    End

end

end

end

```

### 12.3.1.6. *Hex8-Tet4 Resistance*

```

#{R=4.0}
begin sierra Aria

    title Adaptive Square

    BEGIN ARIA MATERIAL M1
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END

    BEGIN ARIA MATERIAL M2
        Density = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END

    Begin Finite Element Model bar
        Database Name = 2blocks_contact_unaligned_hex8_tet4_h{N}.g
        Begin parameters for block block_1
            material M1
        End
        Begin parameters for block block_2
            material M2
        End
    End

```

```

End Finite Element Model bar

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN UMFPACK SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
  begin trilinos equation solver direct_solver
  solution method = amesos-umfpack
  end trilinos equation solver direct_solver
{endif}

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
  BEGIN GMRES SOLVER
  BEGIN DD-ILU PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 200
  RESIDUAL SCALING = R0
  CONVERGENCE TOLERANCE = 1.000000e-12
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Iterative_Solver
  Solution Method = GMRES
  Preconditioning Method = dd-ilu
  Maximum Iterations = 200
  Residual Norm Scaling = R0
  Residual Norm Tolerance = 1.0e-12
  End
{endif}

Begin Field Function ffunc
  Use Nodal Field solution->temperature As Value
End

Begin User Function exact_soln
  Load From File ./Exact_solution.so Using Function registerExactSolution
  Parameter R = {R}
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms L2
  Store in l2_err
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms LInfinity
  Store in linf_err
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function ffunc
  Compute Norms H1
  Store in h1_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_res_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific

```

```

End

begin procedure myProcedure

  Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential The_Time_Block
      Advance myRegion
    End
    Simulation Start Time = 0
    Simulation Termination Time = 1
  End
End

begin Aria region myRegion

  Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

  use finite element model bar
  use linear solver Iterative_Solver #Direct_Solver #

  EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

  Begin TEMPERATURE BOUNDARY CONDITION xm1
    add surface surface_1
    TEMPERATURE = 0.
  End

  Begin TEMPERATURE BOUNDARY CONDITION xp1
    add surface surface_2
    TEMPERATURE = 1.
  End

  Begin VOLUME HEATING sm
    add volume block_1
    VALUE = -1.
  End

  Begin VOLUME HEATING sp
    add volume block_2
    VALUE = 1.
  End

  begin contact definition res1
    contact surface surf_1 contains surface_3
    contact surface surf_2 contains surface_4

    begin interaction inter_1
      surfaces = surf_1 surf_2
    end interaction inter_1

    begin enforcement enf_1
      conductance coefficient= {1.0/R}
      Enforcement for Energy = gap_conductance
    end enforcement

  end contact definition res1

  Output Number of Nodes

  Evaluate Postprocessor l2
  Evaluate Postprocessor linf

```

```

        Evaluate Postprocessor h1

        Begin Results Output Label diffusion output
        database Name = 2blocks_res_h{N}.e
        At Step 1, Increment = 1
        Title Calore Two Blocks
        Nodal Variables = solution->temperature as T
        Element Variables = l2_err linf_err h1_err
    End

end

end

end

```

## 12.3.2. 3D Curved Contact 4.2

12.3.2.1. *Hex8-Hex8 Case*

12.3.2.2. *Tet4-Tet4 Case*

12.3.2.3. *Hex8-Tet4 Case*

## 12.3.3. Steady Hex8 Contact

```

#N={N=5}
BEGIN SIERRA myJob

    BEGIN ARIA MATERIAL Kryptonite
        Density          = Constant rho=1
        Thermal Conductivity = constant k=1
        Specific Heat     = Constant cp=1
        heat conduction   = basic
    END ARIA MATERIAL Kryptonite

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN SUPERLU SOLVER
        END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
        End
    {endif}

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN GMRES SOLVER
        BEGIN JACOBI PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 1000
        RESIDUAL SCALING = NONE
        CONVERGENCE TOLERANCE = 1.000000e-15
        END
        END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = GMRES
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
    }

```

```

        Residual Norm Tolerance = 1.0e-15
    End
{endif}

BEGIN FINITE ELEMENT MODEL cube
    database name = cube_contact_h{N}_hex8.e
    coordinate system is cartesian
    decomposition method = rcb

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

    BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
    END PARAMETERS FOR BLOCK block_2

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
    Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
    Load From File ./exact.so Using Function registerExactSolnDot
End

Begin User Function exact_src
    Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
    Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
    Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
    Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
    Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
    store in l2_error
End

Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
    store in h1_error
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
    store in linf_error
End

```

```

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC
EQ ENERGY for TEMPERATURE on block_2 using Q1 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
  BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

```

```

begin contact definition mpc1
  contact surface surf_1 contains surface_7
  contact surface surf_2 contains surface_8

  output rule = summary

  begin interaction inter_1
    surfaces = surf_1 surf_2
    normal tolerance = 0.01
  end
  begin enforcement enf_1
    Enforcement for Energy = Tied_Temperature
  end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_hex8_tied_contact_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
  element variables = l2_error h2_error linf_error
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.3.4. Steady Hex20 Contact

```

BEGIN SIERRA myJob

  load user plugin file ./exact.so

  begin definition for function kxx
    type = piecewise linear
    begin values
      0.0 0.5
      1.0 2.0
      2.0 8.0
    end values
  end

  begin definition for function kyy
    type = piecewise linear
    begin values
      0.0 0.2
      1.0 1.2
      2.0 2.1
      20.0 20.2
    end values
  end

  begin definition for function kzz
    type = piecewise linear
    begin values
      0.0 1.0

```

```

    1.0 2.0
    2.0 3.0
    20.0 21.0
end values
scale by 2.0
end

BEGIN ARIA MATERIAL Kryptonite
Density          = Constant rho=1
tensor thermal conductivity = user_function X = temperature name_xx = kxx name_yy = kyy name_zz = kzz
Specific Heat    = Constant cp=1
heat conduction  = Generalized
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
END TPETRA EQUATION SOLVER
{else}
Begin Trilinos Equation Solver Direct_Solver
Solution Method = amesos-superlu
End
{endif}

{if(useTpetra)}
BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
  BEGIN GMRES SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 1000
  RESIDUAL SCALING = NONE
  CONVERGENCE TOLERANCE = 1.000000e-14
END
END TPETRA EQUATION SOLVER
{else}
Begin Trilinos Equation Solver Iterative_Solver
Solution Method = GMRES
Preconditioning Method = Jacobi
Maximum Iterations = 1000
Residual Norm Scaling = None
Residual Norm Tolerance = 1.0e-14
End
{endif}

BEGIN FINITE ELEMENT MODEL cube
database name = cube_contact_h{N}_hex20.g
coordinate system is cartesian
decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
  material Kryptonite
  END PARAMETERS FOR BLOCK block_1

  BEGIN PARAMETERS FOR BLOCK block_2
  material Kryptonite
  END PARAMETERS FOR BLOCK block_2

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_src
Load From File ./exact.so Using Function registerExactSrc
End

```

```

Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
Write To File errors_thermal_steady_hex20_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential The_Time_Block
        Advance myRegion
      End
      Simulation Start Time = 0
      Simulation Termination Time = 1
    End
  End

  BEGIN ARIA REGION myRegion

    Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

    use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

    EQ ENERGY for TEMPERATURE on block_1 using Q2S with DIFF SRC
    EQ ENERGY for TEMPERATURE on block_2 using Q2S with DIFF SRC

```

```

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
BC const dirichlet at surface_4 Temperature = 1.0
BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
  contact surface surf_1 contains surface_7
  contact surface surf_2 contains surface_8

  begin interaction inter_1
    surfaces = surf_1 surf_2
  end
  begin enforcement enf_1
    Enforcement for Energy = Tied_Temperature
  end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_hex20_tied_contact_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

```

```
END SIERRA myJob
```

## 12.3.5. Steady Hex27 Contact

```
BEGIN SIERRA myJob
```

```
  load user plugin file ./exact.so
```

```
begin definition for function kxx
```

```
  type = piecewise linear
```

```
  begin values
```

```
    0.0 0.5
```

```
    1.0 2.0
```

```
    2.0 8.0
```

```
  end values
```

```
end
```

```
begin definition for function kyy
```

```
  type = piecewise linear
```

```
  begin values
```

```
    0.0 0.2
```

```
    1.0 1.2
```

```
    2.0 2.1
```

```
    20.0 20.2
```

```
  end values
```

```
end
```

```
begin definition for function kzz
```

```
  type = piecewise linear
```

```
  begin values
```

```
    0.0 1.0
```

```
    1.0 2.0
```

```
    2.0 3.0
```

```
    20.0 21.0
```

```
  end values
```

```
  scale by 2.0
```

```
end
```

```
BEGIN ARIA MATERIAL Kryptonite
```

```
  Density = Constant rho=1
```

```
  tensor thermal conductivity = user_function X = temperature name_xx = kxx name_yy = kyy name_zz = kzz
```

```
  Specific Heat = Constant cp=1
```

```
  heat conduction = Generalized
```

```
END ARIA MATERIAL Kryptonite
```

```
{if(useTpetra)}
```

```
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
```

```
  BEGIN SUPERLU SOLVER
```

```
  END
```

```
  END TPETRA EQUATION SOLVER
```

```
{else}
```

```
  Begin Trilinos Equation Solver Direct_Solver
```

```
  Solution Method = amesos-superlu
```

```
  End
```

```
{endif}
```

```
{if(useTpetra)}
```

```
  BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
```

```
  BEGIN GMRES SOLVER
```

```
  BEGIN JACOBI PRECONDITIONER
```

```
  END
```

```
  MAXIMUM ITERATIONS = 1000
```

```
  RESIDUAL SCALING = NONE
```

```
  CONVERGENCE TOLERANCE = 1.000000e-14
```

```
  END
```

```
  END TPETRA EQUATION SOLVER
```

```

{else}
  Begin Trilinos Equation Solver Iterative_Solver
    Solution Method = GMRES
    Preconditioning Method = Jacobi
    Maximum Iterations = 1000
    Residual Norm Scaling = None
    Residual Norm Tolerance = 1.0e-14
  End
{endif}

BEGIN FINITE ELEMENT MODEL cube
  database name = cube_contact_h{N}_hex27.g
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
    material Kryptonite
  END PARAMETERS FOR BLOCK block_1

  BEGIN PARAMETERS FOR BLOCK block_2
    material Kryptonite
  END PARAMETERS FOR BLOCK block_2

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out

```

```

Comment Character Is %
Write To File errors_thermal_steady_hex27_h{N}.dat
Floating Point Precision Is 3
Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
End

BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
Maximum Nonlinear Iterations = 10
Nonlinear Residual Tolerance = 1.0e-12
Nonlinear Correction Tolerance = 1.0e-12
Nonlinear Relaxation Factor = 1.0
use dof averaged nonlinear residual

use finite element model cube
Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q2 with DIFF SRC
EQ ENERGY for TEMPERATURE on block_2 using Q2 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
BC const dirichlet at surface_4 Temperature = 1.0
BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src

Source For ENERGY on block_2 = Constant value=1

```

```

Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
  contact surface surf_1 contains surface_7
  contact surface surf_2 contains surface_8

  begin interaction inter_1
    surfaces = surf_1 surf_2
  end
  begin enforcement enf_1
    Enforcement for Energy = Tied_Temperature
  end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_hex27_tied_contact_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.3.6. Steady Tet4 Contact

```

#N={N=5}
BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density = Constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat = Constant cp=1
    heat conduction = basic
  END ARIA MATERIAL Kryptonite

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
    BEGIN SUPERLU SOLVER
    END
    END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Direct_Solver
    Solution Method = amesos-superlu
    End
  {endif}

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
    BEGIN GMRES SOLVER
    BEGIN JACOBI PRECONDITIONER
    END
    MAXIMUM ITERATIONS = 1000
    RESIDUAL SCALING = NONE
    CONVERGENCE TOLERANCE = 1.000000e-15
  {endif}

```

```

        END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = GMRES
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-15
    End
{endif}

BEGIN FINITE ELEMENT MODEL cube
    database name = cube_contact_h{N}_tet4.e
    coordinate system is cartesian
    decomposition method = rcb

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

    BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
    END PARAMETERS FOR BLOCK block_2

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
    Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
    Load From File ./exact.so Using Function registerExactSolnDot
End

Begin User Function exact_src
    Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
    Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
    Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
    Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
    Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
    Store In l2_error
End

Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
    Store In h1_error
End

```

```

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_error
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential The_Time_Block
    Advance myRegion
    End
    Simulation Start Time = 0
    Simulation Termination Time = 1
  End
  End

  BEGIN ARIA REGION myRegion

  Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

  use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

  EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC
  EQ ENERGY for TEMPERATURE on block_2 using Q1 with DIFF SRC

  # surface_4: x=0
  # surface_6: x=1

  # surface_3: y=0
  # surface_5: y=1

  # surface_1: z=1
  # surface_2: z=0

  # const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

  # const flux BC (y)
  BC FLUX for Energy on surface_3 = constant flux = 3
  BC FLUX for Energy on surface_5 = constant flux = 5

  BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
  BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

  # convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

  BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1

```

```

BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
  contact surface surf_1 contains surface_7
  contact surface surf_2 contains surface_8

begin interaction inter_1
  surfaces = surf_1 surf_2
end
begin enforcement enf_1
  Enforcement for Energy = Tied_Temperature
end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_tet4_tied_contact_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
  element variables = l2_error h1_error linf_error
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.3.7. Steady Tet4Tet10 Contact

```

#N={N=5}
BEGIN SIERRA myJob

BEGIN ARIA MATERIAL Kryptonite
  Density = Constant rho=1
  Thermal Conductivity = constant k=1
  Specific Heat = Constant cp=1
  heat conduction = basic
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Direct_Solver
  Solution Method = amesos-superlu
  End
{endif}

{if(useTpetra)}

```

```

BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
  BEGIN GMRES SOLVER
    BEGIN JACOBI PRECONDITIONER
      END
      MAXIMUM ITERATIONS = 1000
      RESIDUAL SCALING = NONE
      CONVERGENCE TOLERANCE = 1.000000e-15
    END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Iterative_Solver
    Solution Method = GMRES
    Preconditioning Method = Jacobi
    Maximum Iterations = 1000
    Residual Norm Scaling = None
    Residual Norm Tolerance = 1.0e-15
  End
{endif}

BEGIN FINITE ELEMENT MODEL cube
  database name = cube_contact_h{N}_tet10.e
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
  END PARAMETERS FOR BLOCK block_1

  BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
  END PARAMETERS FOR BLOCK block_2

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./exact.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
  Store In l2_error
End

```

```

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
  Store In h1_error
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_error
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC
EQ ENERGY for TEMPERATURE on block_2 using Q1 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
  BC const dirichlet at surface_4 Temperature = 1.0
  BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3

```

```

BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
  BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
  BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
  contact surface surf_1 contains surface_7
  contact surface surf_2 contains surface_8

begin interaction inter_1
  surfaces = surf_1 surf_2
end
begin enforcement enf_1
  Enforcement for Energy = Tied_Temperature
end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_steady_tet4_tied_contact_h{N}.e
  at step 1, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
  element variables = l2_error h1_error linf_error
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.3.8. Steady Tet10 Contact

```

#N={N=5}
BEGIN SIERRA myJob

BEGIN ARIA MATERIAL Kryptonite
  Density = Constant rho=1
  Thermal Conductivity = constant k=1
  Specific Heat = Constant cp=1
  heat conduction = basic
END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
  END TPETRA EQUATION SOLVER

```

```

{else}
  Begin Trilinos Equation Solver Direct_Solver
  Solution Method = amesos-superlu
  End
{endif}

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
  BEGIN GMRES SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 1000
  RESIDUAL SCALING = NONE
  CONVERGENCE TOLERANCE = 1.000000e-15
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Iterative_Solver
  Solution Method = GMRES
  Preconditioning Method = Jacobi
  Maximum Iterations = 1000
  Residual Norm Scaling = None
  Residual Norm Tolerance = 1.0e-15
  End
{endif}

  BEGIN FINITE ELEMENT MODEL cube
  database name = cube_contact_h{N}_tet10.e
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
  END PARAMETERS FOR BLOCK block_1

  BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
  END PARAMETERS FOR BLOCK block_2

  END FINITE ELEMENT MODEL cube

  Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
  End

  Begin User Function exact_soln_dot
  Load From File ./exact.so Using Function registerExactSolnDot
  End

  Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
  End

  Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3
  End

  Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
  End

  Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
  End

  Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
  End

```

```

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
  Store In l2_error
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
  Store In h1_error
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_error
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential The_Time_Block
      Advance myRegion
    End
    Simulation Start Time = 0
    Simulation Termination Time = 1
  End
  End

  BEGIN ARIA REGION myRegion

  Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

  use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

  EQ ENERGY for TEMPERATURE on block_1 using Q2 with DIFF SRC
  EQ ENERGY for TEMPERATURE on block_2 using Q2 with DIFF SRC

  # surface_4: x=0
  # surface_6: x=1

  # surface_3: y=0
  # surface_5: y=1

  # surface_1: z=1
  # surface_2: z=0

  # const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0

```

```

        BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
    contact surface surf_1 contains surface_7
    contact surface surf_2 contains surface_8

begin interaction inter_1
    surfaces = surf_1 surf_2
end
begin enforcement enf_1
    Enforcement for Energy = Tied_Temperature
end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = thermal_steady_tet10_tied_contact_h{N}.e
    at step 1, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
    element variables = l2_error h1_error linf_error
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.3.9. Steady Tet10 Dash Contact

```

#N={N=4}
BEGIN SIERRA myJob

BEGIN ARIA MATERIAL Kryptonite
    Density = Constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat = Constant cp=1
    heat conduction = basic

```

```

END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Direct_Solver
  Solution Method = amesos-superlu
  End
{endif}

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
  BEGIN GMRES SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 1000
  RESIDUAL SCALING = NONE
  CONVERGENCE TOLERANCE = 1.000000e-15
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Iterative_Solver
  Solution Method = GMRES
  Preconditioning Method = Jacobi
  Maximum Iterations = 1000
  Residual Norm Scaling = None
  Residual Norm Tolerance = 1.0e-15
  End
{endif}

  BEGIN FINITE ELEMENT MODEL cube
  database name = cube_contact_h{N}_tet10.e
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
  END PARAMETERS FOR BLOCK block_1

  BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
  END PARAMETERS FOR BLOCK block_2

  END FINITE ELEMENT MODEL cube

  Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
  End

  Begin User Function exact_soln_dot
  Load From File ./exact.so Using Function registerExactSolnDot
  End

  Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
  End

  Begin User Function flux_surface_3
  Load From File ./exact.so Using Function registerFlux_Surface_3
  End

  Begin User Function flux_surface_5
  Load From File ./exact.so Using Function registerFlux_Surface_5
  End

```

```

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
  Store In l2_error
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
  Store In h1_error
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_error
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_h{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q2 with DIFF SRC
EQ ENERGY for TEMPERATURE on block_2 using Q2 with DIFF SRC

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0

```

```

# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1

    skin all blocks = on

    search = dash

begin interaction defaults
    general contact = on
end interaction defaults

begin dash options
    interaction definition scheme = explicit
    search length scaling = 0.75
end dash options

begin enforcement enf_1
    Enforcement for Energy = Dash_Tied
end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = thermal_steady_tet10_tied_dash_contact_h{N}.e
    at step 1, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
    element variables = l2_error h1_error linf_error
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

```

```
END SIERRA myJob
```

## 12.3.10. Transient Tet4Tet10 Contact

```
#N={N=5}
BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density          = Constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat     = Constant cp=1
    heat conduction   = basic
  END ARIA MATERIAL Kryptonite

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
    BEGIN SUPERLU SOLVER
    END
    END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Direct_Solver
    Solution Method = amesos-superlu
    End
  {endif}

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
    BEGIN GMRES SOLVER
    BEGIN JACOBI PRECONDITIONER
    END
    MAXIMUM ITERATIONS = 1000
    RESIDUAL SCALING = NONE
    CONVERGENCE TOLERANCE = 1.000000e-15
    END
    END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Iterative_Solver
    Solution Method = GMRES
    Preconditioning Method = Jacobi
    Maximum Iterations = 1000
    Residual Norm Scaling = None
    Residual Norm Tolerance = 1.0e-15
    End
  {endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = cube_contact_h{N}_tet10.e
    coordinate system is cartesian
    decomposition method = rcb

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

    BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
    END PARAMETERS FOR BLOCK block_2

  END FINITE ELEMENT MODEL cube

  Begin User Function exact_soln
    Load From File ./exact_transient.so Using Function registerExactSoln
  End

  Begin User Function exact_soln_dot
```

```

    Load From File ./exact_transient.so Using Function registerExactSolnDot
End

Begin User Function exact_src
    Load From File ./exact_transient.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
    Load From File ./exact_transient.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
    Load From File ./exact_transient.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
    Load From File ./exact_transient.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
    Load From File ./exact_transient.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
End

Begin Norm Postprocessor l2_dot
    Use Function exact_soln_dot
    Subtract Function time_derivative_at_time->TEMPERATURE
    Compute Norms L2
End

Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors_h{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Transient The_Time_Block
        Advance myRegion
    End
    Simulation Start Time = 0
    Simulation Termination Time = 3
End
Begin Parameters For Transient The_Time_Block
Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.5/2**N}

```

```

        Time Integration Method = Second_Order
        Time Step Variation = fixed
    End
End
    End

    BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC
EQ ENERGY for TEMPERATURE on block_2 using Q1 with MASS DIFF SRC

    IC const on block_1 temperature = 1.0
    IC const on block_2 temperature = 1.0

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
    contact surface surf_1 contains surface_7
    contact surface surf_2 contains surface_8

begin interaction inter_1
    surfaces = surf_1 surf_2
end
begin enforcement enf_1
    Enforcement for Energy = Tied_Temperature
end

```

```

        end

        Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = thermal_transient_tet10_tied_contact_h{N}.e
    at step 0, increment = {2*N}
    title Aria cube test
    nodal variables = solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

        END ARIA REGION myRegion

        END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.3.11. Transient Tet10 Contact

```

#N={N=5}
BEGIN SIERRA myJob

        BEGIN ARIA MATERIAL Kryptonite
            Density          = Constant rho=1
            Thermal Conductivity = constant k=1
            Specific Heat     = Constant cp=1
            heat conduction   = basic
        END ARIA MATERIAL Kryptonite

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN SUPERLU SOLVER
            END
        END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
    End
{endif}

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER ITERATIVE_SOLVER
        BEGIN GMRES SOLVER
            BEGIN JACOBI PRECONDITIONER
                END
            MAXIMUM ITERATIONS = 1000
            RESIDUAL SCALING = NONE
            CONVERGENCE TOLERANCE = 1.000000e-15
        END
    END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Iterative_Solver
        Solution Method = GMRES
        Preconditioning Method = Jacobi
        Maximum Iterations = 1000
        Residual Norm Scaling = None
        Residual Norm Tolerance = 1.0e-15
    End
{endif}

```

```

BEGIN FINITE ELEMENT MODEL cube
  database name = cube_contact_h{N}_tet10.e
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
  END PARAMETERS FOR BLOCK block_1

  BEGIN PARAMETERS FOR BLOCK block_2
material Kryptonite
  END PARAMETERS FOR BLOCK block_2

END FINITE ELEMENT MODEL cube

Begin User Function exact_soln
  Load From File ./exact_transient.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./exact_transient.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./exact_transient.so Using Function registerExactSrc
End

Begin User Function flux_surface_3
  Load From File ./exact_transient.so Using Function registerFlux_Surface_3
End

Begin User Function flux_surface_5
  Load From File ./exact_transient.so Using Function registerFlux_Surface_5
End

Begin User Function flux_surface_1
  Load From File ./exact_transient.so Using Function registerFlux_Surface_1
End

Begin User Function flux_surface_2
  Load From File ./exact_transient.so Using Function registerFlux_Surface_2
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out

```

```

    Comment Character Is %
    Write To File errors_h{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Transient The_Time_Block
        Advance myRegion
    End
        Simulation Start Time = 0
        Simulation Termination Time = 3
End
Begin Parameters For Transient The_Time_Block
    Begin Parameters For Aria Region myRegion
        Initial Time Step Size = {0.5/2**N}
        Time Integration Method = Second_Order
        Time Step Variation = fixed
    End
End
    End

    BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

use finite element model cube
    Use Linear Solver Iterative_Solver #Direct_Solver

EQ ENERGY for TEMPERATURE on block_1 using Q2 with MASS DIFF SRC
EQ ENERGY for TEMPERATURE on block_2 using Q2 with MASS DIFF SRC

    IC const on block_1 temperature = 1.0
    IC const on block_2 temperature = 1.0

# surface_4: x=0
# surface_6: x=1

# surface_3: y=0
# surface_5: y=1

# surface_1: z=1
# surface_2: z=0

# const Temp BC (x)
    BC const dirichlet at surface_4 Temperature = 1.0
    BC const dirichlet at surface_6 Temperature = 1.0

# const flux BC (y)
BC FLUX for Energy on surface_3 = constant flux = 3
BC FLUX for Energy on surface_5 = constant flux = 5

BC FLUX for Energy on surface_3 = Encore_Function Name=flux_surface_3
BC FLUX for Energy on surface_5 = Encore_Function Name=flux_surface_5

# convective flux BC with const Temp and H (z)
    BC Flux for Energy on surface_1 = Nat_Conv T_Ref=1 H=1
    BC Flux for Energy on surface_2 = Nat_Conv T_Ref=2 H=2

```

```

BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1
BC FLUX for Energy on surface_2 = Encore_Function Name=flux_surface_2

# const source term
Source For ENERGY on block_1 = Constant value=1
Source For ENERGY on block_2 = Constant value=1

Source For ENERGY on block_1 = Encore_Function Name=exact_src
Source For ENERGY on block_2 = Encore_Function Name=exact_src

begin contact definition mpc1
  contact surface surf_1 contains surface_7
  contact surface surf_2 contains surface_8

begin interaction inter_1
  surfaces = surf_1 surf_2
end
begin enforcement enf_1
  Enforcement for Energy = Tied_Temperature
end
end

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = thermal_transient_tet10_tied_contact_h{N}.e
  at step 0, increment = {2**N}
  title Aria cube test
  nodal variables = solution->TEMPERATURE as T
  nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.4. ELEMENT DEATH

### 12.4.1. CDFEM Element Death (Heat Flux)

#### 12.4.1.1. *Tri3*

```
BEGIN SIERRA Aria

Title \$
1-d standard conduction problem, Carslaw and Jaeger P. 292\$

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
  BEGIN CG SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 10000
  RESIDUAL SCALING = RO
  CONVERGENCE TOLERANCE = 1.000000e-14
  END
  END TPETRA EQUATION SOLVER
{else}
  BEGIN Aztec EQUATION SOLVER solve_temperature
  Solution Method = cg
  Preconditioning Method = JACOBI
  Maximum Iterations = 10000
  Residual Norm Tolerance = 1e-14
  Residual norm scaling = RO
  END
{endif}

Begin Global Constants
  Stefan Boltzmann Constant = 5.67e-8 # W/m^2-K^4
  Ideal Gas Constant = 8.314 # J/mol-K
End

BEGIN ARIA MATERIAL solid
  DENSITY = constant rho = 1.
  Thermal Conductivity = constant k = 1.
  Specific Heat = Constant cp = 1.0
  Heat Conduction = basic
END ARIA MATERIAL solid

BEGIN FINITE ELEMENT MODEL VERIFY_DEATH
  DATABASE NAME = input{N}_tri3.e
  decomposition method = rcb
  COORDINATE SYSTEM = CARTESIAN
  DATABASE TYPE = EXODUSII
  USE MATERIAL solid FOR block_1
  USE MATERIAL solid FOR block_1_dead
END FINITE ELEMENT MODEL VERIFY_DEATH

Begin String Function exact_soln
  Value Is "ln(sqrt(x*x+y*y))*(1/ln(2-t))"
  Gradient Is "(x/(x*x+y*y))*(1/ln(2-t))" "(y/(x*x+y*y))*(1/ln(2-t))"
End

Begin String Function exact_src
  Value Is "ln(sqrt(x*x+y*y))*(-1/(ln(2-t)*ln(2-t)))*(1/(2-t))*(-1)"
End

# exact flux computed at interface (r=2-t)
Begin String Function exact_flux
  Value Is "-1/((2-t)*ln(2-t))"
End
```

```

# exact interface position (radial)
Begin String Function exact_interface
  Value Is "2-t"
End

# radius function - evaluate r(x,y)
Begin String Function radius
  Value Is "sqrt(x*x+y*y)"
End

Begin Norm Postprocessor l2
  Volumes block_1
  Use Function exact_soln
  Subtract Function solution->TEMPERATURE
  Compute Norms L2
  Store In l2_err
End

Begin Norm Postprocessor h1
  Volumes block_1
  Use Function exact_soln
  Subtract Function solution->TEMPERATURE
  Compute Norms H1
  Store In h1_err
End

Begin Norm Postprocessor linf
  Volumes block_1
  Use Function exact_soln
  Subtract Function solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_err
End

Begin Norm Postprocessor l2_interface
  Surfaces surface_1 surface_block_1_death_by_temp
  Use Function exact_interface
  Subtract Function radius
  Compute Norms L2
End

Begin Norm Postprocessor linf_interface
  Surfaces surface_1 surface_block_1_death_by_temp
  Use Function exact_interface
  Subtract Function radius
  Compute Norms Nodal LInfinity
  Store In linf_interface_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 6
  Floating Point Format Is Fixed
End

Begin Procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Simulation Start Time      = 0.0
      Simulation Termination Time = 0.9
    End
    Begin transient MySolveBlock
      Advance myRegion
    End
  End
  begin parameters for transient MySolveBlock

```

```

    start time = 0.0
    begin parameters for aria region myRegion
        initial time step size = {0.1*(0.5**(N-1))}
        Predictor-Corrector Tolerance = {0.05*(0.5**(N-1))}
        Maximum Time Step Size = {0.2*(0.5**(N-1))}
        Time Integration Method = bdf2
        time step variation = adaptive
    end
end
End

Begin Aria Region myRegion

    Use Finite Element Model VERIFY_DEATH

    Begin CDFEM Death death_by_temp
        add volume block_1
        Criterion is solution->Temperature > 1.0
    End

    Use Linear Solver solve_temperature

    nonlinear solution strategy = newton
    maximum nonlinear iterations = 2
    nonlinear correction tolerance = 1.0e-12
    nonlinear residual tolerance = 1.0e-12
    nonlinear relaxation factor = 1.0

    use dof averaged nonlinear residual

    eq energy for temperature on all_blocks using q1 with lumped_mass diff src

    IC Encore function on block_1 Temperature = exact_soln

    BC Dirichlet for Temperature on surface_2 = encore_function name=exact_soln

    SOURCE for Energy on block_1 = encore_function name=exact_src

    BC Flux for Energy on surface_1 = encore_function name=exact_flux
    BC Flux for Energy on surface_block_1_death_by_temp = encore_function name=exact_flux

    Output Number of Nodes

    Evaluate Postprocessor l2
    Evaluate Postprocessor h1
    Evaluate Postprocessor linf

    Evaluate Postprocessor l2_interface
    Evaluate Postprocessor linf_interface

    Begin Results Output Label diffusion output
        database name = output{N}.e
        at Step 0, Interval = {2**N}
        Nodal Variables = solution->temperature as TEMP
        Nodal Variables = linf_interface_err
        Element Variables = l2_err h1_err linf_err
    End Results Output Label diffusion output

End Aria Region myRegion

End Procedure myProcedure

END SIERRA ARIA

```

### 12.4.1.2. Tet4

BEGIN SIERRA Aria

Title \\$

1-d standard conduction problem, Carslaw and Jaeger P. 292\\$

{if(useTpetra)}

BEGIN TPETRA EQUATION SOLVER SOLVE\_TEMPERATURE

BEGIN CG SOLVER

BEGIN JACOBI PRECONDITIONER

END

MAXIMUM ITERATIONS = 10000

RESIDUAL SCALING = RO

CONVERGENCE TOLERANCE = 1.000000e-14

END

END TPETRA EQUATION SOLVER

{else}

BEGIN Aztec EQUATION SOLVER solve\_temperature

Solution Method = cg

Preconditioning Method = JACOBI

Maximum Iterations = 10000

Residual Norm Tolerance = 1e-14

Residual norm scaling = RO

END

{endif}

Begin Global Constants

Stefan Boltzmann Constant = 5.67e-8 # W/m<sup>2</sup>-K<sup>4</sup>

Ideal Gas Constant = 8.314 # J/mol-K

End

BEGIN ARIA MATERIAL solid

DENSITY = constant rho = 1.

Thermal Conductivity = constant k = 1.

Specific Heat = Constant cp = 1.0

Heat Conduction = basic

END ARIA MATERIAL solid

BEGIN FINITE ELEMENT MODEL VERIFY\_DEATH

DATABASE NAME = input{N}\_tet4.e

decomposition method = rcb

COORDINATE SYSTEM = CARTESIAN

DATABASE TYPE = EXODUSII

USE MATERIAL solid FOR block\_1

USE MATERIAL solid FOR block\_1\_dead

END FINITE ELEMENT MODEL VERIFY\_DEATH

Begin String Function exact\_soln

Value Is "(1+t)/sqrt(x\*x+y\*y+z\*z)"

Gradient Is "(1+t)\*(-1/(x\*x+y\*y+z\*z))\*(x/sqrt(x\*x+y\*y+z\*z))" "(1+t)\*(-1/(x\*x+y\*y+z\*z))\*(y/sqrt(x\*x+y\*y+z\*z))" "(1+t)\*(-1/(x\*x+y\*y+z\*z))\*(z/sqrt(x\*x+y\*y+z\*z))"

End

Begin String Function exact\_src

Value Is "1/sqrt(x\*x+y\*y+z\*z)"

End

# exact flux computed at interface (r=2-t)

Begin String Function exact\_flux

Value Is "-1/(1+t)"

###Value Is "-(1+t)" ### for verification of case with no element death

End

# exact interface position (radial)

Begin String Function exact\_interface

Value Is "1+t"

End

```

# radius function - evaluate r(x,y)
Begin String Function radius
  Value Is "sqrt(x*x+y*y+z*z)"
End

Begin Norm Postprocessor l2
  Volumes block_1
  Use Function exact_soln
  Subtract Function solution->TEMPERATURE
  Compute Norms L2
  Store In l2_err
End

Begin Norm Postprocessor h1
  Volumes block_1
  Use Function exact_soln
  Subtract Function solution->TEMPERATURE
  Compute Norms H1
  Store In h1_err
End

Begin Norm Postprocessor linf
  Volumes block_1
  Use Function exact_soln
  Subtract Function solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_err
End

Begin Norm Postprocessor l2_interface
  Surfaces surface_1 surface_block_1_death_by_temp
  Use Function exact_interface
  Subtract Function radius
  Compute Norms L2
End

Begin Norm Postprocessor linf_interface
  Surfaces surface_1 surface_block_1_death_by_temp
  Use Function exact_interface
  Subtract Function radius
  Compute Norms Nodal LInfinity
  Store In linf_interface_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 6
  Floating Point Format Is Fixed
End

Begin Procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Simulation Start Time          = 0.0
      Simulation Termination Time    = 0.75
    End
    Begin transient MySolveBlock
      Advance myRegion
    End
  End

  begin parameters for transient MySolveBlock
    start time = 0.0
    begin parameters for aria region myRegion
      initial time step size = {0.1*(0.5**(N-1))}
      Predictor-Corrector Tolerance = {0.05*(0.5**(N-1))}
    end
  end

```

```

        Maximum Time Step Size = {0.2*(0.5**(N-1))}
        Time Integration Method = bdf2
        time step variation = adaptive
    end
end
End

Begin Aria Region myRegion

    Use Finite Element Model VERIFY_DEATH

    Begin CDFEM Death death_by_temp
        add volume block_1
        Criterion is solution->Temperature > 1.0
    End

    Use Linear Solver solve_temperature

    nonlinear solution strategy = newton
    maximum nonlinear iterations = 2
    nonlinear correction tolerance = 1.0e-12
    nonlinear residual tolerance = 1.0e-12
    nonlinear relaxation factor = 1.0

    use dof averaged nonlinear residual

    eq energy for temperature on all_blocks using q1 with lumped_mass diff src

    IC Encore function on block_1 Temperature = exact_soln

    BC Dirichlet for Temperature on surface_2 = encore_function name=exact_soln

    SOURCE for Energy on block_1 = encore_function name=exact_src

    BC Flux for Energy on surface_1 = encore_function name=exact_flux
    BC Flux for Energy on surface_block_1_death_by_temp = encore_function name=exact_flux

    Output Number of Nodes

    Evaluate Postprocessor l2
    Evaluate Postprocessor h1
    Evaluate Postprocessor linf

    Evaluate Postprocessor l2_interface
    Evaluate Postprocessor linf_interface

    Begin Results Output Label diffusion output
        database name = output{N}.e
        at Step 0, Interval = {2**N}
        Nodal Variables = solution->temperature as TEMP
        Nodal Variables = linf_interface_err
        Element Variables = l2_err h1_err linf_err
    End Results Output Label diffusion output

End Aria Region myRegion

End Procedure myProcedure

END SIERRA ARIA

```

## 12.4.2. 3D Spherical Shell Enclosure

```

BEGIN SIERRA myJob

    Title Element Death Test Problem #1

```

```

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
  BEGIN CG SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  RESIDUAL SCALING = RO
  CONVERGENCE TOLERANCE = 1.000000e-09
  END
END TPETRA EQUATION SOLVER
{else}
  BEGIN TRILINOS EQUATION SOLVER solve_temperature
  Solution Method = CG
  Preconditioning Method = jacobi
  Residual Norm Scaling = RO
  Residual Norm Tolerance = 1.0e-09
  END
{endif}

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
END TPETRA EQUATION SOLVER
{else}
  BEGIN TRILINOS EQUATION SOLVER direct_solver
  Solution Method = amesos-superlu
  END
{endif}

BEGIN GLOBAL Constants
  Stefan Boltzmann Constant = 5.6704e-08 # W/m2-K4
End

BEGIN ARIA MATERIAL ss304
  Heat Conduction      = Basic
  Density              = Constant rho = 7862.0 $ kg/m**3
  Specific Heat        = Constant Cp = 10.0 $ J/gm/K
  Thermal Conductivity = Constant K = 1.0 $ W/m/K
END ARIA MATERIAL ss304

BEGIN ARIA MATERIAL fake
END ARIA MATERIAL fake

BEGIN FINITE ELEMENT MODEL myModel
  Database Name = two_sphere_shells_tet4_m{N}.g
  Coordinate System is cartesian
  decomposition method = rcb

  Use material ss304 for block_1 # Outer "case" block
  Use material ss304 for block_2 # Inner block that will have death

  #THIS BLOCK SHOULD BE REMOVED EVENTUALLY BUT IS CURRENTLY REQUIRED
  # Commenting it out leads to a segfault because of a null field data pointer for model_coordinates
  # This is very bizarre
  Use material fake for block_2_dead
  #BEGIN PARAMETERS FOR BLOCK block_2_dead
  # Material fake
  #END PARAMETERS FOR BLOCK block_2_dead

END FINITE ELEMENT MODEL myModel

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./exact.so Using Function registerExactSolnDot

```

```

End

Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSolnSrc
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
  Store In l2_err
  Volumes block_1 block_2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
  Store In l2_dot_err
  Volumes block_1 block_2
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms Nodal LInfinity
  Store In linf_err
  Volumes block_1 block_2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
  Store In h1_err
  Volumes block_1 block_2
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

Begin Average Value Postprocessor avg_T_surf1
  Use Function nonlinear_solution->Temperature
  Surfaces surface_1
End

Begin Average Value Postprocessor avg_T_surf2
  Use Function nonlinear_solution->Temperature
  Surfaces surface_2
End

Begin Average Value Postprocessor avg_T_surf2_death_temp1
  Use Function nonlinear_solution->Temperature
  Surfaces surface_block_2_death_temp1
End

Begin Average Value Postprocessor avg_T_surf3
  Use Function nonlinear_solution->Temperature
  Surfaces surface_3
End

Begin Average Value Postprocessor avg_T_surf4
  Use Function nonlinear_solution->Temperature
  Surfaces surface_4

```

```

End

BEGIN PROCEDURE myAriaProcedure

Begin Solution Control Description
Use System Main
Begin System Main
Simulation Start Time          = 0.0
Simulation Termination Time    = 0.5 #1.0
Begin Transient Stepper
Advance myRegion
End
End

Begin Parameters For Transient Stepper
Begin Parameters for Aria Region myRegion
Time Integration Method = BDF2
Time Step Variation     = fixed #Adaptive
Initial Time Step Size  = {0.05*0.5**(N)}
#Minimum Time Step Size = 1.0e-5
#Maximum Time Step Size = 1000.0
#predictor order = 0
#Predictor-Corrector Tolerance = 1.e-3
#Fail Time Step When Time Step Size Ratio Is Below 0.0
End
End
End

BEGIN ARIA REGION myRegion

Use Linear Solver solve_temperature #direct_solver
Use Finite Element Model myModel

Begin CDFEM Death death_temp1
add volume block_2
Criterion is solution->Temperature > 867.011674920813
End

Nonlinear Solution Strategy = newton
Maximum Nonlinear Iterations = 10
Nonlinear Residual Tolerance = 1.0e-06
Nonlinear Correction Tolerance = 1.0e-06
Nonlinear Relaxation Factor = 1.0
Accept Solution After Maximum Nonlinear Iterations = false

IC for Temperature for all_volumes = Encore_Function Name=exact_soln

BC Dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
BC Dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with lumped_mass Diff Src
EQ ENERGY for TEMPERATURE on block_2 using Q1 with lumped_mass Diff Src

Source For ENERGY on all_volumes = Encore_Function Name=exact_src

Begin Viewfactor Calculation vf_calc
Compute Rule          = Hemicube
Geometric Tolerance   = 1.0e-6
Hemicube Resolution   = 500
Hemicube Max Subdivides = 5
hemicube min separation = 5.0
Output Rule           = Verbose
End

Begin Viewfactor Smoothing smooth
Method                = least-squares
Convergence Tolerance = 1.0e-06
Maximum Iterations    = 500

```

```

weight power          = 2.0
Reciprocity Rule     = average
Output Rule          = verbose
End

Begin Viewfactor Smoothing no_smooth
Method               = none
Convergence Tolerance = 1.0E-08
Maximum Iterations   = 150
Weight Power         = 2.0
Reciprocity Rule     = average
Output Rule          = Summary
End

Begin Radiosity Solver Rad_Solv
Coupling             = mason
Solver               = chaparral GMRES
Convergence Tolerance = 1.0e-08
Maximum Iterations   = 300
Output Rule          = none
End

Begin Enclosure Definition enc1
add surface surface_2
add surface surface_3
add surface surface_block_2_death_temp1

meshed enclosure is block_2_dead
disable parallel redistribution

Emissivity = 0.6 On surface_block_2_death_temp1
Emissivity = 0.6 On surface_2
Emissivity = 0.7 On surface_3

Blocking Surfaces
Use Viewfactor Calculation vf_calc
Use Viewfactor Smoothing no_smooth
Use Radiosity Solver Rad_Solv
End

Postprocess Heat_Flux on All_Blocks

Evaluate Postprocessor avg_T_surf1
Evaluate Postprocessor avg_T_surf2
Evaluate Postprocessor avg_T_surf2_death_temp1
Evaluate Postprocessor avg_T_surf3
Evaluate Postprocessor avg_T_surf4

Begin Postprocessor Group zzz
Output Number Of Nodes
Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor linf
Evaluate Postprocessor h1
End

BEGIN RESULTS OUTPUT myLABEL diffusion output etc
Database Name = cdfem_rad_death_m{N}.e
at step 0, increment = 1

Title CDFEM Death Test Case #1
Nodal Variables = Solution->Temperature as T
Nodal Variables = pp->heat_flux as Heat_Flux
Nodal Variables = enc1_position linf_err
nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
Face Variables = face_area emissivity face_coverage face_temperature irradiance rad_flux radiosity
Element Variables = current_element_volume initial_element_volume volume_change_ratio l2_err l2_dot_err h1_err
END

```

```
Begin Heartbeat thermalrace
  stream Name = globals_tet4_m{N}.dat
  precision = 7
  timestamp format ''
  legend = off
  labels = off
  Variable = Global time as time
  Variable = Global l2_err
  Variable = Global l2_dot
  Variable = Global linf
  Variable = Global h1_err
  Variable = Global enc1_area
  Variable = Global enc1_volume
  Variable = Global avg_T_surf1
  Variable = Global avg_T_surf2 # this value is mostly NaNs
  Variable = Global avg_T_surf2_death_temp1
  Variable = Global avg_T_surf3
  Variable = Global avg_T_surf4
  at step 0, increment = 1
end

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob
```

## 12.5. TIME INTEGRATION

### 12.5.1. Adaptive Time Integration

#### 12.5.1.1. First Order Fixed

```
BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1
    Thermal Conductivity = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1
    Specific Heat = Constant cp=1
    heat conduction = basic
  END ARIA MATERIAL Kryptonite

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
  BEGIN SUPERLU SOLVER
  END
  END TPETRA EQUATION SOLVER
{else}
  Begin Trilinos Equation Solver Direct_Solver
  Solution Method = amesos-superlu
  End
{endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = square_h{N}_tri3.e
    coordinate system is cartesian

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

  END FINITE ELEMENT MODEL cube

  Begin User Function exact_soln
    Load From File ./somefunc.so Using Function registerExactSoln
  End

  Begin User Function exact_soln_dot
    Load From File ./somefunc.so Using Function registerExactSolnDot
  End

  Begin User Function exact_src
    Load From File ./somefunc.so Using Function registerExactSrc
  End

  Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
  End

  Begin Norm Postprocessor l2_dot
    Use Function exact_soln_dot
    Subtract Function time_derivative_at_time->TEMPERATURE
    Compute Norms L2
  End

  Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
  End
End
```

```

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_1st_ord_fixed_h{N}.dat
  Floating Point Precision Is 4
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Transient The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
  Initial Time Step Size = {0.1/2**N}
  Time Integration Method = First_Order
  Time Step Variation = fixed
  End
End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Direct_Solver

  IC Encore Function on block_1 temperature = exact_soln

  BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src

  Output Number of TimeSteps

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = aria_1st_ord_fixed_h{N}.e
  at step 0, increment = 1
  # time interval is 1.0
  title Aria cube test

```

```

    nodal variables = nonlinear_solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

```

```

    END ARIA REGION myRegion

```

```

END PROCEDURE myAriaProcedure

```

```

END SIERRA myJob

```

### 12.5.1.2. *First Order Adaptive*

```

BEGIN SIERRA myJob

```

```

    BEGIN ARIA MATERIAL Kryptonite
        Density          = Polynomial Variable=Temperature Order=1 CO=1 C1=0.1
        Thermal Conductivity = Polynomial Variable=Temperature Order=1 CO=1 C1=0.1
        Specific Heat      = Constant cp=1
        heat conduction    = basic
    END ARIA MATERIAL Kryptonite

```

```

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
        BEGIN SUPERLU SOLVER
            END
        END TPETRA EQUATION SOLVER
{else}
    Begin Trilinos Equation Solver Direct_Solver
        Solution Method = amesos-superlu
    End
{endif}

```

```

    BEGIN FINITE ELEMENT MODEL cube
        database name = square_h{N}_tri3.e
        coordinate system is cartesian

```

```

        BEGIN PARAMETERS FOR BLOCK block_1
        material Kryptonite
        END PARAMETERS FOR BLOCK block_1

```

```

    END FINITE ELEMENT MODEL cube

```

```

    Begin User Function exact_soln
        Load From File ./somefunc.so Using Function registerExactSoln
    End

```

```

    Begin User Function exact_soln_dot
        Load From File ./somefunc.so Using Function registerExactSolnDot
    End

```

```

    Begin User Function exact_src
        Load From File ./somefunc.so Using Function registerExactSrc
    End

```

```

    Begin Norm Postprocessor l2
        Use Function exact_soln
        Subtract Function nonlinear_solution->TEMPERATURE
        Compute Norms L2
    End

```

```

    Begin Norm Postprocessor l2_dot
        Use Function exact_soln_dot
        Subtract Function time_derivative_at_time->TEMPERATURE
        Compute Norms L2
    End

```

```

    Begin Norm Postprocessor h1

```

```

    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors_1st_ord_adapt_h{N}.dat
    Floating Point Precision Is 4
    Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Transient The_Time_Block
        Advance myRegion
    End
        Simulation Start Time = 0
        Simulation Termination Time = 1
End
Begin Parameters For Transient The_Time_Block
    Begin Parameters For Aria Region myRegion
        Initial Time Step Size = {0.1/2**N}
        Predictor-Corrector Tolerance = {1e-1/4**N}
        Maximum Time Step Size = {0.5/2**N}
        Time Integration Method = First_Order
        Time Step Variation = adaptive
    End
End
End

    BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

use finite element model cube
    Use Linear Solver Direct_Solver

    IC Encore Function on block_1 temperature = exact_soln

    BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
    BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
    BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
    BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src

    Output Number of TimeSteps

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1

```

```

Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = aria_1st_ord_adapt_h{N}.e
  at step 0, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
  nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

      END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

### 12.5.1.3. *Second Order Fixed*

```

BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density          = Polynomial Variable=Temperature Order=1 CO=1 C1=0.1
    Thermal Conductivity = Polynomial Variable=Temperature Order=1 CO=1 C1=0.1
    Specific Heat      = Constant cp=1
    heat conduction    = basic
  END ARIA MATERIAL Kryptonite

  load user plugin file ./somefunc.so

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
      BEGIN SUPERLU SOLVER
        END
      END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Direct_Solver
      Solution Method = amesos-superlu
    End
  {endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = square_h{N}_tri3.e
    coordinate system is cartesian

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

  END FINITE ELEMENT MODEL cube

  Begin User Function exact_soln
    Load From File ./somefunc.so Using Function registerExactSoln
  End

  Begin User Function exact_soln_dot
    Load From File ./somefunc.so Using Function registerExactSolnDot
  End

  Begin User Function exact_src
    Load From File ./somefunc.so Using Function registerExactSrc
  End

  Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2

```

```

End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_2nd_ord_fixed_h{N}.dat
  Floating Point Precision Is 4
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Transient The_Time_Block
  Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
  Initial Time Step Size = {0.1/2**N}
  Time Integration Method = Second_Order
  Time Step Variation = fixed
  End
End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Direct_Solver

  IC Encore Function on block_1 temperature = exact_soln

  BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

```

```
Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src
```

```
Output Number of TimeSteps
```

```
Evaluate Postprocessor l2  
Evaluate Postprocessor l2_dot  
Evaluate Postprocessor h1  
Evaluate Postprocessor linf
```

```
BEGIN RESULTS OUTPUT LABEL diffusion output  
database Name = aria_2nd_ord_fixed_h{N}.e  
at step 0, increment = 1  
# time interval is 1.0  
title Aria cube test  
nodal variables = nonlinear_solution->TEMPERATURE as T  
nodal variables = time_derivative_at_time->TEMPERATURE as TDOT  
END RESULTS OUTPUT LABEL diffusion output
```

```
END ARIA REGION myRegion
```

```
END PROCEDURE myAriaProcedure
```

```
END SIERRA myJob
```

#### 12.5.1.4. *Second Order Adaptive*

```
BEGIN SIERRA myJob
```

```
BEGIN ARIA MATERIAL Kryptonite  
Density = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1  
Thermal Conductivity = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1  
Specific Heat = Constant cp=1  
heat conduction = basic  
END ARIA MATERIAL Kryptonite
```

```
{if(useTpetra)}
```

```
BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER  
BEGIN SUPERLU SOLVER  
END  
END TPETRA EQUATION SOLVER
```

```
{else}
```

```
Begin Trilinos Equation Solver Direct_Solver  
Solution Method = amesos-superlu  
End
```

```
{endif}
```

```
BEGIN FINITE ELEMENT MODEL cube  
database name = square_h{N}_tri3.e  
coordinate system is cartesian
```

```
BEGIN PARAMETERS FOR BLOCK block_1  
material Kryptonite  
END PARAMETERS FOR BLOCK block_1
```

```
END FINITE ELEMENT MODEL cube
```

```
Begin User Function exact_soln  
Load From File ./somefunc.so Using Function registerExactSoln  
End
```

```
Begin User Function exact_soln_dot  
Load From File ./somefunc.so Using Function registerExactSolnDot  
End
```

```
Begin User Function exact_src  
Load From File ./somefunc.so Using Function registerExactSrc  
End
```

```

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_2nd_ord_adapt_h{N}.dat
  Floating Point Precision Is 4
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Transient The_Time_Block
    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.1/2**N}
    Predictor-Corrector Tolerance = {1e-1/4**N}
    Maximum Time Step Size = {0.5/2**N}
    Time Integration Method = Second_Order
    Time Step Variation = adaptive
  End
End
End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Direct_Solver

  IC Encore Function on block_1 temperature = exact_soln

```

```

        BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
        BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
        BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
        BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src

    Output Number of TimeSteps

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = aria_2nd_ord_adapt_h{N}.e
    at step 0, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
    nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

    END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

### 12.5.1.5. *BDF2 Fixed*

```

BEGIN SIERRA myJob

    BEGIN ARIA MATERIAL Kryptonite
        Density = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1
        Thermal Conductivity = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1
        Specific Heat = Constant cp=1
        heat conduction = basic
    END ARIA MATERIAL Kryptonite

    {if(useTpetra)}
        BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
            BEGIN SUPERLU SOLVER
                END
            END TPETRA EQUATION SOLVER
    {else}
        Begin Trilinos Equation Solver Direct_Solver
            Solution Method = amesos-superlu
        End
    {endif}

    BEGIN FINITE ELEMENT MODEL cube
        database name = square_h{N}_tri3.e
        coordinate system is cartesian

        BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
        END PARAMETERS FOR BLOCK block_1

    END FINITE ELEMENT MODEL cube

    Begin User Function exact_soln
        Load From File ./somefunc.so Using Function registerExactSoln
    End

```

```

Begin User Function exact_soln_dot
  Load From File ./somefunc.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./somefunc.so Using Function registerExactSrc
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_bdf2_fixed_h{N}.dat
  Floating Point Precision Is 4
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Transient The_Time_Block
      Advance myRegion
    End
    Simulation Start Time = 0
    Simulation Termination Time = 1
  End
  Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.1/2**N}
    Time Integration Method = BDF2
    Time Step Variation = fixed
  End
End

  End

  BEGIN ARIA REGION myRegion

  Nonlinear Solution Strategy = Newton
    Maximum Nonlinear Iterations = 10
    Nonlinear Residual Tolerance = 1.0e-12
    Nonlinear Correction Tolerance = 1.0e-12
    Nonlinear Relaxation Factor = 1.0
    use dof averaged nonlinear residual

```

```

use finite element model cube
  Use Linear Solver Direct_Solver

  IC Encore Function on block_1 temperature = exact_soln

  BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src

  Output Number of TimeSteps

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = aria_bdf2_fixed_h{N}.e
  at step 0, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
  nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

  END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

### 12.5.1.6. *BDF2 Adaptive*

```

BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1
    Thermal Conductivity = Polynomial Variable=Temperature Order=1 C0=1 C1=0.1
    Specific Heat = Constant cp=1
    heat conduction = basic
  END ARIA MATERIAL Kryptonite

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
    BEGIN SUPERLU SOLVER
    END
  END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Direct_Solver
    Solution Method = amesos-superlu
    End
  {endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = square_h{N}_tri3.e
    coordinate system is cartesian

    BEGIN PARAMETERS FOR BLOCK block_1
  material Kryptonite
    END PARAMETERS FOR BLOCK block_1

  END FINITE ELEMENT MODEL cube

```

```

Begin User Function exact_soln
  Load From File ./somefunc.so Using Function registerExactSoln
End

Begin User Function exact_soln_dot
  Load From File ./somefunc.so Using Function registerExactSolnDot
End

Begin User Function exact_src
  Load From File ./somefunc.so Using Function registerExactSrc
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor l2_dot
  Use Function exact_soln_dot
  Subtract Function time_derivative_at_time->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_bdf2_adapt_h{N}.dat
  Floating Point Precision Is 4
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Transient The_Time_Block
    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
Begin Parameters For Transient The_Time_Block
  Begin Parameters For Aria Region myRegion
    Initial Time Step Size = {0.1/2**N}
    Predictor-Corrector Tolerance = {1e-1/4**N}
    Maximum Time Step Size = {0.5/2**N}
    Time Integration Method = BDF2
    Time Step Variation = adaptive
  End
End

  BEGIN ARIA REGION myRegion

```

```

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Direct_Solver

  IC Encore Function on block_1 temperature = exact_soln

  BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with MASS DIFF SRC

Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src

  Output Number of TimeSteps

Evaluate Postprocessor l2
Evaluate Postprocessor l2_dot
Evaluate Postprocessor h1
Evaluate Postprocessor linf

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = aria_bdf2_adapt_h{N}.e
  at step 0, increment = 1
  # time interval is 1.0
  title Aria cube test
  nodal variables = nonlinear_solution->TEMPERATURE as T
  nodal variables = time_derivative_at_time->TEMPERATURE as TDOT
END RESULTS OUTPUT LABEL diffusion output

  END ARIA REGION myRegion

  END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.6. ENCLOSURE RADIATION

### 12.6.1. 2D Cylindrical Shell Enclosure

```
BEGIN SIERRA Aria

Title Verification for two concentric spheres with radiation gap between them

Begin Global Constants
  Stefan Boltzmann constant = 5.6704E-8
End Global Constants

Begin Aria Material inner
  Heat conduction = Basic
  Density = constant rho = 1.0
  Specific heat = constant cp = 1.0
  Thermal conductivity = constant k = 2.0
End Aria Material inner

Begin Aria Material outer
  Heat conduction = Basic
  Density = constant rho = 1.0
  Specific heat = constant cp = 1.0
  Thermal conductivity = constant k = 0.35
End Aria Material outer

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

Begin Finite Element Model VERIFY_RAD_GAP
  Database name = input{N}.g
  Coordinate System = Cartesian
  Database Type = EXODUSII
  Begin Parameters for Block block_1
    Material inner
  End
  Begin Parameters for Block block_2
    Material outer
  End
End Finite Element Model VERIFY_RAD_GAP
```

```

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
    BEGIN CG SOLVER
      BEGIN JACOBI PRECONDITIONER
        END
      MAXIMUM ITERATIONS = 1000
      RESIDUAL SCALING = RHS
      CONVERGENCE TOLERANCE = 1.000000e-14
    END
  END TPETRA EQUATION SOLVER
{else}
  BEGIN Aztec Equation Solver solve_temperature
    Solution Method = cg
    Preconditioning Method = jacobi
    Maximum Iterations = 1000
    Residual Norm Tolerance = 1.0e-14
    Residual Norm Scaling = RHS
  END
{endif}

Begin Procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential Steady
        Advance myRegion
      End
    End
  End

  Begin Aria Region myRegion

    Use Finite Element Model VERIFY_RAD_GAP
    Use Linear Solver solve_temperature

    Nonlinear Solution Strategy = Newton
    maximum nonlinear iterations = 1000
    nonlinear residual tolerance = 1.0e-10
    nonlinear relaxation factor = 1.0

    EQ energy for Temperature for all_volumes using Q1 with Diff
    IC const for all_volumes Temperature = 300.0
    BC const Dirichlet at surface_1 Temperature = 300.0
    BC const Dirichlet at surface_4 Temperature = 1300.0

    Output Number of Nodes
    Evaluate Postprocessor l2
    Evaluate Postprocessor linf
    Evaluate Postprocessor h1

    begin enclosure definition sph_shell
      add surface surface_2
      add surface surface_3
      blocking surfaces
      use viewfactor calculation vf_calc
      use viewfactor smoothing vf_smooth
      use radiosity solver rad_solver
    #   enclosure id           = 1
      emissivity = 0.50 on surface_2
      emissivity = 0.80 on surface_3
    end enclosure definition sph_shell

    begin viewfactor calculation vf_calc
      bsp tree max depth = 0 and min list length = 25
      compute rule           = hemicube
      geometric tolerance    = 1.0E-6

```

```

    hemicube max subdivides    = 5
    hemicube min separation    = 5.0
    hemicube resolution        = 500
#   check rowsum with tolerance = .001
    output rule                = verbose
end viewfactor calculation vf_calc

begin viewfactor smoothing vf_smooth
  convergence tolerance = 1.0E-10
  method                = least-squares
  weight power          = 2
  maximum iterations    = 150
  reciprocity rule      = average
  output rule           = verbose
end viewfactor smoothing vf_smooth

begin radiosity solver rad_solver
  coupling              = mason
  solver                = chaparral gmres
  convergence tolerance = 1.0E-8
  maximum iterations    = 800
  output rule           = verbose
end radiosity solver rad_solver

Begin Results Output Label diffusion output
  database name = output{N}.e
  at Step 0, increment = 1
  Nodal Variables = solution->temperature as TEMP
End Results Output Label diffusion output

End Aria Region myRegion

End Procedure myProcedure

END SIERRA Aria

```

## 12.6.2. 2D Annular Enclosure

```

BEGIN SIERRA Aria

Title Verification for two concentric spheres with radiation gap between them

Begin Global Constants
  Stefan Boltzmann constant = 5.6704E-8
End Global Constants

Begin Aria Material inner
  Heat conduction = Basic
  Thermal conductivity = constant k = 1.0
End Aria Material inner

Begin User Function exact_soln
  Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_src
  Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_1
  Load From File ./exact.so Using Function registerExactFlux
End

Begin User Function exact_radiosity
  Load From File ./exact.so Using Function registerExactRadiosity
End

```

```

Begin User Function exact_irradiance
  Load From File ./exact.so Using Function registerExactIrradiance
End

Begin Field Function radiosity
  Use Edge Field radiosity As Value
End

Begin Field Function irradiance
  Use Edge Field irradiance As Value
End

Begin Field Function rad_flux
  Use Edge Field rad_flux As Value
End

Begin Norm Postprocessor l2
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms L2
  Store In l2_err
End

Begin Norm Postprocessor h1
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms H1
  Store In h1_err
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_err
End

Begin Norm Postprocessor l2_radiosity
  Surfaces surface_1
  Use Function exact_radiosity
  Subtract Function radiosity
  Compute Norms L2
  Store In l2_radiosity_err
End

Begin Norm Postprocessor l2_irradiance
  Surfaces surface_1
  Use Function exact_irradiance
  Subtract Function irradiance
  Compute Norms L2
  Store In l2_irradiance_err
End

Begin Norm Postprocessor l2_heatflux
  Surfaces surface_1
  Use Function flux_surface_1
  Subtract Function rad_flux
  Compute Norms L2
  Store In l2_heatflux_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

```

```

Begin Finite Element Model VERIFY_RAD_GAP
  Database name = annulus_crack_h{N}_tri3.e
  Coordinate System = Cartesian
  decomposition method = rcb
  Database Type = EXODUSII
  Begin Parameters for Block block_1
    Material inner
  End
End Finite Element Model VERIFY_RAD_GAP

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
    BEGIN BICGSTAB SOLVER
      BEGIN JACOBI PRECONDITIONER
        END
      MAXIMUM ITERATIONS = 1000
      RESIDUAL SCALING = RHS
      CONVERGENCE TOLERANCE = 1.000000e-14
    END
  END TPETRA EQUATION SOLVER
{else}
  BEGIN Aztec Equation Solver solve_temperature
    Solution Method = bicgstab
    Preconditioning Method = jacobi
    Maximum Iterations = 1000
    Residual Norm Tolerance = 1.0e-14
    Residual Norm Scaling = RHS
  END
{endif}

Begin Procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential Steady
        Advance myRegion
      End
    End
  End
End

Begin Aria Region myRegion

  Use Finite Element Model VERIFY_RAD_GAP
  Use Linear Solver solve_temperature

  Nonlinear Solution Strategy = Newton
  maximum nonlinear iterations = 1000
  nonlinear residual tolerance = 1.0e-10
  nonlinear correction tolerance = 1.0e-10
  nonlinear relaxation factor = 1.0

  EQ energy for Temperature for all_volumes using Q1 with Diff Src
  IC const for all_volumes Temperature = 600.0

  Source For ENERGY on block_1 = Encore_Function Name=exact_src

  BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

  Output Number of Nodes

  Evaluate Postprocessor l2
  Evaluate Postprocessor h1
  Evaluate Postprocessor linf

```

```

Evaluate Postprocessor l2_radiosity
Evaluate Postprocessor l2_irradiance
#Evaluate Postprocessor l2_heatflux

Interpolate Function Value of exact_soln Into Nodal Field Tex
Interpolate Function Value of exact_src Into Nodal Field Src

Begin Results Output Label diffusion output
  database name = output{N}.e
  at Step 1, increment = 1
  Nodal Variables = solution->temperature as TEMP
  Nodal Variables = Tex linf_err Src
  Element Variables = l2_err h1_err TGrad linf_err
  edge variables = radiosity as J
  edge variables = rad_flux as q
  edge variables = irradiance as I
  edge variables = l2_radiosity_err
  edge variables = l2_irradiance_err
  edge variables = l2_heatflux_err
End Results Output Label diffusion output

# TODO: remove this boundary condition
#   # DEBUG
#   BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln

# TODO: remove this boundary condition
#   # DEBUG
#   BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1

# TODO: un-comment the enclosure BC
begin enclosure definition sph_shell
  add surface surface_1
  nonblocking surfaces
  use viewfactor calculation vf_calc_pairwise #vf_calc_hemicube #
  use viewfactor smoothing vf_smooth
  use radiosity solver rad_solver
  emissivity = 0.9 on surface_1
  Database Name is enc{N}.vf in pnetcdf format
  disable parallel redistribution
end

begin viewfactor calculation vf_calc_hemicube
  compute rule          = hemicube
  geometric tolerance  = {0.5*0.5**N}
  hemicube max subdivides = {2*(2**N)}
  hemicube min separation = 5.0
  hemicube resolution   = {100*(2**N)}
  output rule          = verbose
end

begin viewfactor calculation vf_calc_pairwise
  compute rule          = pairwise
  geometric tolerance  = {0.5*0.5**N}
  output rule          = verbose
  Pairwise Monte Carlo Sample Rule = Halton
  Pairwise Monte Carlo Tol1 = 1e-5
  Pairwise Monte Carlo Tol2 = 1e-5
  Pairwise Number Of Visibility Samples = 1
  Pairwise Visibility Sample Rule = Uniform
end

begin viewfactor smoothing vf_smooth
  convergence tolerance = 1.0E-10
  method                = least-squares
  weight power          = 2
  maximum iterations    = {150*(2**N)}
  reciprocity rule      = average

```

```

        output rule          = verbose
    end viewfactor smoothing vf_smooth

    begin radiosity solver rad_solver
        coupling              = mason
        solver                 = chaparral gmres
        convergence tolerance = 1.0E-11
        maximum iterations    = {150*(2**N)}
        output rule           = none
    end radiosity solver rad_solver

    End Aria Region myRegion

End Procedure myProcedure

END SIERRA Aria

```

### 12.6.3. 3D Spherical Shell Enclosure

### 12.6.4. 3D Spherical Shell Partial Enclosure

```

BEGIN SIERRA Aria

Title Verification for two concentric spheres with radiation gap between them

load user plugin file ./exact.so

Begin Global Constants
    Stefan Boltzmann constant = 5.6704E-8
End Global Constants

Begin Aria Material inner
    Heat conduction = Basic
    Density = constant rho = 1.0
    Specific heat = constant cp = 1.0
    Thermal conductivity = constant k = 2.0
    emissivity = constant e = 0.50
End Aria Material inner

Begin Aria Material outer
    Heat conduction = Basic
    Density = constant rho = 1.0
    Specific heat = constant cp = 1.0
    Thermal conductivity = constant k = 0.35
    emissivity = constant e = 0.80
End Aria Material outer

    Begin User Function exact_soln
        Load From File ./exact.so Using Function registerExactSoln
    End

    Begin Norm Postprocessor l2
        Use Function exact_soln
        Volumes block_1 block_2
        Subtract Function nonlinear_solution->TEMPERATURE
        Compute Norms L2
        Store In l2_err
    End

    Begin Norm Postprocessor h1
        Use Function exact_soln
        Volumes block_1 block_2
        Subtract Function nonlinear_solution->TEMPERATURE
        Compute Norms H1
    End

```

```

    Store In h1_err
End

Begin Norm Postprocessor linf
  Use Function exact_soln
  Volumes block_1 block_2
  Subtract Function nonlinear_solution->TEMPERATURE
  Compute Norms LInfinity
  Store In linf_err
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 3
  Floating Point Format Is Scientific
End

BEGIN Field Function numerical_solution
  USE NODAL FIELD solution->TEMPERATURE
END Field Function numerical_solution

BEGIN Difference Function temp_error
  Difference is exact_soln - numerical_solution
END Difference Function temp_error

Begin Finite Element Model VERIFY_RAD_GAP
  Database name = sphere_cutout_h{N}.g
  Coordinate System = Cartesian
  Database Type = EXODUSII
  Begin Parameters for Block block_1
    Material inner
  End
  Begin Parameters for Block block_2
    Material outer
  End
End Finite Element Model VERIFY_RAD_GAP

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
    BEGIN CG SOLVER
      BEGIN JACOBI PRECONDITIONER
      END
      MAXIMUM ITERATIONS = 1000
      RESIDUAL SCALING = RHS
      CONVERGENCE TOLERANCE = 1.000000e-10
    END
  END TPETRA EQUATION SOLVER
{else}
  BEGIN Aztec Equation Solver solve_temperature
    Solution Method = cg
    Preconditioning Method = jacobi
    Maximum Iterations = 1000
    Residual Norm Tolerance = 1.0e-10
    Residual Norm Scaling = RHS
  END
{endif}

Begin Procedure myProcedure

  Begin Solution Control Description
    Use System Main
    Begin System Main
      Begin Sequential Steady
        Advance myRegion
      End
    End
  End
End

```

```

Begin Aria Region myRegion

Use Finite Element Model VERIFY_RAD_GAP
Use Linear Solver solve_temperature

Nonlinear Solution Strategy = Newton
maximum nonlinear iterations = 1000
nonlinear residual tolerance = 1.0e-8
nonlinear correction tolerance = 1.0e-8
nonlinear relaxation factor = 1.0

EQ energy for Temperature for all_volumes using Q1 with Diff
IC const for all_volumes Temperature = 300.0
BC const Dirichlet at surface_1 Temperature = 300.0
BC const Dirichlet at surface_4 Temperature = 1300.0
BC Dirichlet for Temperature on surface_5 = encore_function name=exact_soln

Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor linf
Evaluate Postprocessor h1

interpolate function value of exact_soln into nodal field analytic_temp on volumes block_1 block_2
Interpolate Function Value of temp_error into nodal field temp_error on volumes block_1 block_2

begin enclosure definition sph_shell
  add surface surface_2
  add surface surface_3
  blocking surfaces
  use viewfactor calculation vf_calc
  use viewfactor smoothing vf_smooth
  use radiosity solver rad_solver
  emissivity = 0.50 on surface_2
  emissivity = 0.80 on surface_3
  Partial Enclosure Emissivity = 0.8
  Partial Enclosure Area = {2.0*PI*0.03*(0.03-0.025)}
  Partial Enclosure Temperature = 1035.02
end enclosure definition sph_shell

begin viewfactor calculation vf_calc
  bsp tree max depth = 0 and min list length = 25
  compute rule          = hemicube
  geometric tolerance   = 1.0E-10
  hemicube max subdivides = 5
  hemicube min separation = 5.0
  hemicube resolution    = 500
#   check rowsum with tolerance = .001
  output rule          = verbose
end viewfactor calculation vf_calc

begin viewfactor smoothing vf_smooth
  method          = none
end viewfactor smoothing vf_smooth

begin radiosity solver rad_solver
  coupling          = mason
  solver            = chaparral gmres
  convergence tolerance = 1.0E-9
  maximum iterations = 80
  output rule       = summary
end radiosity solver rad_solver

Begin Results Output Label diffusion output
  database name = output{N}.e
  at Step 0, increment = 1
  Nodal Variables = solution->temperature as T

```

```

        nodal variables = analytic_temp
        nodal variables = temp_error
        element variables = h1_err l2_err linf_err
    End Results Output Label diffusion output

End Aria Region myRegion

End Procedure myProcedure

END SIERRA Aria

```

## 12.6.5. Fully 2D Enclosure Radiation

```

BEGIN SIERRA Aria

Title Verification for two concentric spheres with radiation gap between them

Begin Global Constants
    Stefan Boltzmann constant = 5.6704E-8
End Global Constants

Begin Aria Material inner
    Heat conduction = Basic
    Thermal conductivity = constant k = 1.0
End Aria Material inner

Begin User Function exact_soln
    Load From File ./exact.so Using Function registerExactSoln
End

Begin User Function exact_src
    Load From File ./exact.so Using Function registerExactSrc
End

Begin User Function flux_surface_1
    Load From File ./exact.so Using Function registerExactFlux
End

Begin User Function exact_radiosity
    Load From File ./exact.so Using Function registerExactRadiosity
End

Begin User Function exact_irradiance
    Load From File ./exact.so Using Function registerExactIrradiance
End

Begin Field Function radiosity
    Use Edge Field radiosity As Value
End

Begin Field Function irradiance
    Use Edge Field irradiance As Value
End

Begin Field Function rad_flux
    Use Edge Field rad_flux As Value
End

Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms L2
    Store In l2_err
End

Begin Norm Postprocessor h1

```

```

    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms H1
    Store In h1_err
End

Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function nonlinear_solution->TEMPERATURE
    Compute Norms LInfinity
    Store In linf_err
End

Begin Norm Postprocessor l2_radiosity
    Surfaces surface_1
    Use Function exact_radiosity
    Subtract Function radiosity
    Compute Norms L2
    Store In l2_radiosity_err
End

Begin Norm Postprocessor l2_irradiance
    Surfaces surface_1
    Use Function exact_irradiance
    Subtract Function irradiance
    Compute Norms L2
    Store In l2_irradiance_err
End

Begin Norm Postprocessor l2_heatflux
    Surfaces surface_1
    Use Function flux_surface_1
    Subtract Function rad_flux
    Compute Norms L2
    Store In l2_heatflux_err
End

Begin Postprocessor Output Control pp_out
    Comment Character Is %
    Write To File errors{N}.dat
    Floating Point Precision Is 3
    Floating Point Format Is Scientific
End

Begin Finite Element Model VERIFY_RAD_GAP
    Database name = annulus_crack_h{N}_tri3.e
    Coordinate System = Cartesian
    decomposition method = rcb
    Database Type = EXODUSII
    Begin Parameters for Block block_1
        Material inner
    End
End Finite Element Model VERIFY_RAD_GAP

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
        BEGIN BICGSTAB SOLVER
            BEGIN JACOBI PRECONDITIONER
                END
                MAXIMUM ITERATIONS = 1000
                RESIDUAL SCALING = RHS
                CONVERGENCE TOLERANCE = 1.000000e-14
            END
        END TPETRA EQUATION SOLVER
{else}
    BEGIN Aztec Equation Solver solve_temperature
        Solution Method = bicgstab
        Preconditioning Method = jacobi

```

```

        Maximum Iterations = 1000
        Residual Norm Tolerance = 1.0e-14
        Residual Norm Scaling = RHS
    END
{endif}

Begin Procedure myProcedure

Begin Solution Control Description
    Use System Main
    Begin System Main
        Begin Sequential Steady
            Advance myRegion
        End
    End
End

Begin Aria Region myRegion

    Use Finite Element Model VERIFY_RAD_GAP
    Use Linear Solver solve_temperature

    Nonlinear Solution Strategy = Newton
    maximum nonlinear iterations = 1000
    nonlinear residual tolerance = 1.0e-10
    nonlinear correction tolerance = 1.0e-10
    nonlinear relaxation factor = 1.0

    EQ energy for Temperature for all_volumes using Q1 with Diff Src
    IC const for all_volumes Temperature = 600.0

    Source For ENERGY on block_1 = Encore_Function Name=exact_src

    BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
    BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
    BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

    Output Number of Nodes

    Evaluate Postprocessor l2
    Evaluate Postprocessor h1
    Evaluate Postprocessor linf

    Evaluate Postprocessor l2_radiosity
    Evaluate Postprocessor l2_irradiance
    #Evaluate Postprocessor l2_heatflux

    Interpolate Function Value of exact_soln Into Nodal Field Tex
    Interpolate Function Value of exact_src Into Nodal Field Src

    Begin Results Output Label diffusion output
        database name = output{N}.e
        at Step 1, increment = 1
        Nodal Variables = solution->temperature as TEMP
        Nodal Variables = Tex linf_err Src
        Element Variables = l2_err h1_err TGrad linf_err
        edge variables = radiosity as J
        edge variables = rad_flux as q
        edge variables = irradiance as I
        edge variables = l2_radiosity_err
        edge variables = l2_irradiance_err
        edge variables = l2_heatflux_err
    End Results Output Label diffusion output

# TODO: remove this boundary condition
# # DEBUG
# BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln

```

```

# TODO: remove this boundary condition
#   # DEBUG
#   BC FLUX for Energy on surface_1 = Encore_Function Name=flux_surface_1

# TODO: un-comment the enclosure BC
begin enclosure definition sph_shell
  add surface surface_1
  nonblocking surfaces
  use viewfactor calculation vf_calc_pairwise #vf_calc_hemicube #
  use viewfactor smoothing vf_smooth
  use radiosity solver rad_solver
  emissivity = 0.9 on surface_1
  Database Name is enc{N}.vf in pnetcdf format
  disable parallel redistribution
end

begin viewfactor calculation vf_calc_hemicube
  compute rule          = hemicube
  geometric tolerance   = {0.5*0.5**N}
  hemicube max subdivides = {2*(2**N)}
  hemicube min separation = 5.0
  hemicube resolution   = {100*(2**N)}
  output rule          = verbose
end

begin viewfactor calculation vf_calc_pairwise
  compute rule          = pairwise
  geometric tolerance   = {0.5*0.5**N}
  output rule          = verbose
  Pairwise Monte Carlo Sample Rule = Halton
  Pairwise Monte Carlo Tol1 = 1e-5
  Pairwise Monte Carlo Tol2 = 1e-5
  Pairwise Number Of Visibility Samples = 1
  Pairwise Visibility Sample Rule = Uniform
end

begin viewfactor smoothing vf_smooth
  convergence tolerance = 1.0E-10
  method                = least-squares
  weight power          = 2
  maximum iterations    = {150*(2**N)}
  reciprocity rule      = average
  output rule          = verbose
end viewfactor smoothing vf_smooth

begin radiosity solver rad_solver
  coupling              = mason
  solver                = chaparral gmres
  convergence tolerance = 1.0E-11
  maximum iterations    = {150*(2**N)}
  output rule          = none
end radiosity solver rad_solver

End Aria Region myRegion

End Procedure myProcedure

END SIERRA Aria

```

## 12.7. CHEMISTRY

### 12.7.1. First Order Reaction (Uniform Temperature)

### 12.7.2. First Order Reaction (Spatially Varying Temperature)

```
BEGIN SIERRA Aria

Title Verification Problem for Coupled Chemistry Diffusion

load user plugin file ./Exact_solution.so

Begin Field Function species
  Use Quadrature Field species
  Integration Order Is 2
  Dimension Is 2
End

Begin User Function ufuncAB
  Load From File ./Exact_solution.so Using Function registerExactSolnAB
End

Begin User Function ufuncT
  Load From File ./Exact_solution.so Using Function registerExactSolnT
End

Begin User Function exact_src
  Load From File ./Exact_solution.so Using Function registerExactSrc
End

Begin Postprocessor Output Control pp_out
  Comment Character is %
  Write To File errors{N}.dat
End

BEGIN Aria MATERIAL hmx
  density = constant rho = 1
  specific heat = constant cp = 1
  heat conduction = basic
  thermal conductivity = constant k = 1
  begin parameters for chemeq model hmx

    number of reactions = 1

    species names are A B
    species phases are Condensed GAS

    Condensed Fraction = 0.0
    Steric Coefficients are 0.0
    Log Preexponential Factors are 5
    Activation Energies are 1000.0
    Energy Releases are 0 # insure temperature stays const

    Concentration Exponents for A are 1.0
    Concentration Exponents for B are 0.0

    Stoichiometric coefficients for A are -1.0
    Stoichiometric coefficients for B are 1.0

  end parameters for chemeq model hmx

end Aria MATERIAL hmx

BEGIN GLOBAL CONSTANTS
```

```

    Ideal Gas Constant = 1.9872 #CGS_cal
END GLOBAL CONSTANTS

BEGIN FINITE ELEMENT MODEL block
    Database Name = input{N}.g
    decomposition method = rcb
    Use material hmx for block_1
END FINITE ELEMENT MODEL block

{if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
        BEGIN CG SOLVER
            BEGIN JACOBI PRECONDITIONER
            END
            MAXIMUM ITERATIONS = 1000
            RESIDUAL SCALING = NONE
            CONVERGENCE TOLERANCE = 1.000000e-12
        END
    END TPETRA EQUATION SOLVER
{else}
    BEGIN AZTEC EQUATION SOLVER solve_temperature
        solution method = cg
        preconditioning method = jacobi
        maximum iterations = 1000
        residual norm scaling = NONE
        residual norm tolerance = 1.0E-12
    END AZTEC EQUATION SOLVER solve_temperature
{endif}

BEGIN procedure myProcedure

begin solution control description
    Use System Main
    Begin System Main
        Simulation Start Time      = 0.0
        Simulation Termination Time = 0.04
        Begin Transient time_block
            advance myregion
        End Transient time_block
    End System Main

    BEGIN parameters for transient time_block
        start time = 0.0
        termination time = 0.04

        BEGIN PARAMETERS FOR Aria REGION myRegion
            time step variation = fixed
            time integration method = second_order
            initial time step size = {0.01*0.5**(N-1)}
        END PARAMETERS FOR Aria REGION myRegion

    END parameters for transient time_block

end solution control description

begin aria region myRegion

    Output Number Of Nodes
    Compute Difference L2      Of ufuncAB species
    Compute Difference L2      Of ufuncT solution->temperature

    Interpolate Function Value of ufuncAB Into Element Field AEX

    EQ energy for temperature on all_blocks using Q1 with diff mass src

    Source for Energy on block_1 = chemeq_heating MODEL = hmx
    Source For ENERGY on block_1 = Encore_Function Name=exact_src

```

```

BC const dirichlet at surface_1 Temperature = 400.0

IC Encore Function on block_1 temperature = ufuncT

maximum nonlinear iterations = 10
nonlinear residual tolerance = 1.0E-10
nonlinear correction tolerance = 1.0E-10
Nonlinear Relaxation Factor = 1.

BEGIN CHEMEQ SOLVER FOR hmx
  ODE SOLVER = CVODE ADAMS 12 FUNCTIONAL
  Absolute Tolerance = 1e-14
  Relative Tolerance = 1e-10
  Chemistry step multiplier = 100 # default
  Epsilon Min = 0.0001 # default
  Epsilon Max = 10.0 # default
  Minimum Chemistry Timestep = 1.0E-15 # default
  Percentage Asymptotics = 0.0 # default
  Asymptotic tolerance = 100.0 # default
  Minimum Concentration for A = 1.0E-12 # default
  Activation Temperature = 100.0
  Deactivation Temperature = 500.0 Continue
  species A = 1.0
  species B = 0.0
END CHEMEQ SOLVER FOR hmx

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = output{N}.e
  At Step 0, Increment = {2**(N)}
  Timestep Adjustment Interval = 1
  Title Aria Chem/Diffusion Verification
  Nodal Variables = solution->temperature as T
  Element Variables = A B AEX species
END RESULTS OUTPUT LABEL diffusion output

USE FINITE ELEMENT MODEL block
use LINEAR SOLVER solve_temperature

END Aria REGION myRegion

END procedure myProcedure

END SIERRA Aria

```

### 12.7.3. First Order Reaction

```

BEGIN SIERRA Aria

  Title Verification Problem for Coupled Chemistry Diffusion

  load user plugin file ./Exact_solution.so

  Begin Field Function species
    Use Quadrature Field species
    Integration Order Is 2
    Dimension Is 2
  End

  Begin User Function ufuncAB
    Load From File ./Exact_solution.so Using Function registerExactSolnAB
  End

  Begin User Function ufuncT
    Load From File ./Exact_solution.so Using Function registerExactSolnT
  End

```

```

Begin User Function energySrc
  Load From File ./Exact_solution.so Using Function registerEnergySrc
End

Begin Norm Postprocessor L2_AB
  Use Function ufuncAB
  Subtract Function species
  Compute Norms L2
End

Begin Norm Postprocessor L2_T
  Use Function ufuncT
  Subtract Function solution->temperature
  Compute Norms L2
End

Begin Norm Postprocessor LInf_T
  Use Function ufuncT
  Subtract Function solution->temperature
  Compute Norms LInfinity
End

Begin Norm Postprocessor H1_T
  Use Function ufuncT
  Subtract Function solution->temperature
  Compute Norms H1
End

Begin Postprocessor Output Control pp_out
  Comment Character is %
  Write To File error{N}.txt
End

BEGIN Aria MATERIAL hmx
  density = constant rho = 1
  specific heat = constant cp = 1
  heat conduction = basic
  thermal conductivity = constant k = 1
  begin parameters for chemeq model hmx

    number of reactions = 1

    species names are A B
    species phases are Condensed GAS

    Condensed Fraction = 0.0
    Steric Coefficients are 0.0
    Log Preexponential Factors are 5
    Activation Energies are 1000.0
    Energy Releases are -20.0

    Concentration Exponents for A are 1.0
    Concentration Exponents for B are 0.0

    Stoichiometric coefficients for A are -1.0
    Stoichiometric coefficients for B are 1.0

  end parameters for chemeq model hmx
end Aria MATERIAL hmx

BEGIN GLOBAL CONSTANTS
  Ideal Gas Constant = 1.9872 #CGS_cal
END GLOBAL CONSTANTS

BEGIN FINITE ELEMENT MODEL block

```

```

Database Name = grid{N}x.exo $ exodusii
decomposition method = rib

Use material hmx for block_1
END FINITE ELEMENT MODEL block

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
  BEGIN CG SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 1000
  RESIDUAL SCALING = NONE
  CONVERGENCE TOLERANCE = 1.000000e-06
  END
END TPETRA EQUATION SOLVER
{else}
  BEGIN AZTEC EQUATION SOLVER solve_temperature
  solution method = cg
  preconditioning method = jacobi
  maximum iterations = 1000
  residual norm scaling = NONE
  residual norm tolerance = 1.0E-6
  END AZTEC EQUATION SOLVER solve_temperature
{endif}

BEGIN procedure myProcedure

begin solution control description
  Use System Main
  Begin System Main
  Simulation Start Time      = 0.0
  Simulation Termination Time = 0.04
  Begin Transient time_block
  advance myregion
  End Transient time_block
  End System Main

  BEGIN parameters for transient time_block
  start time = 0.0
  termination time = 0.04

  BEGIN PARAMETERS FOR Aria REGION myRegion
  time step variation = fixed
  time integration method = second_order
  initial time step size = {0.001*0.5**(N-1)}
  END PARAMETERS FOR Aria REGION myRegion

  END parameters for transient time_block

end solution control description

begin aria region myRegion

Output Number Of Elements

Evaluate Postprocessor L2_AB
Evaluate Postprocessor L2_T
Evaluate Postprocessor LInf_T
Evaluate Postprocessor H1_T

EQ energy for temperature on all_blocks using Q1 with diff mass src

Source for Energy on all_blocks = chemeq_heating MODEL = hmx

use data block region_data

maximum nonlinear iterations = 10

```

```

nonlinear residual tolerance = 1.0E-10
nonlinear correction tolerance = 1.0E-10
Nonlinear Relaxation Factor = 1.

BEGIN CHEMEQ SOLVER FOR hmx
#   Chemistry step multiplier = 10.0
#   Epsilon Min = 0.0001
#   Epsilon Max = 10.0
#   Minimum Chemistry Timestep = 1.0E-15
#   Percentage Asymptotics = 0.0
#   Asymptotic tolerance = 100.0
ODE SOLVER = CVODE ADAMS 12 FUNCTIONAL
Absolute Tolerance = 1e-12
Relative Tolerance = 1e-9
Minimum Concentration for A = 1.0E-08
Activation Temperature = 0.0
  species A = 1.0
  species B = 0.0
END CHEMEQ SOLVER FOR hmx

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = output{N}.e
  At Step 0, Increment = {2**(N-1)}
  Timestep Adjustment Interval = 1
  Title Aria Chem/Diffusion Verification
  Nodal Variables = solution->temperature as T
  Element Variables = A B species
END RESULTS OUTPUT LABEL diffusion output

IC Encore Function on block_1 temperature = ufuncT

BC dirichlet for Temperature at surface_1 = Encore_Function Name=ufuncT

Source For ENERGY on block_1 = Encore_Function Name=energySrc

USE FINITE ELEMENT MODEL block
use LINEAR SOLVER solve_temperature

END Aria REGION myRegion

END procedure myProcedure

END SIERRA Aria

```

## 12.7.4. DAE and Pressure Test

```

BEGIN SIERRA Aria

  Title Verification Problem for Coupled Chemistry Diffusion

  load user plugin file ./Exact_solution.so

  Begin Field Function species
    Use Quadrature Field species
    Integration Order Is 2
    Dimension Is 1
  End

  Begin User Function ufuncA
    Load From File ./Exact_solution.so Using Function registerExactSolnA
  End

  Begin Norm Postprocessor L2_A
    Use Function ufuncA
    Subtract Function species
    Compute Norms L2

```

```

End

Begin Postprocessor Output Control pp_out
  Comment Character is %
  Write To File error.txt
End

BEGIN Aria MATERIAL hmx
  density = constant rho = 1
  specific heat = constant cp = 1
  heat conduction = basic
  thermal conductivity = constant k = 1
  pressure = constant value=3

  begin parameters for chemeq model hmx
    number of reactions = 1

    species names are A
    species phases are Condensed

    Condensed Fraction = 0.0
    Steric Coefficients are 0.0
    Log Preexponential Factors are {log(5)}
    Activation Energies are 10.0
    Energy Releases are 0.0

    Concentration Exponents for A are 0.0

    Stoichiometric coefficients for A are -1.0

    # Pressure dependence
    Reference pressure = 2.
    Pressure exponents are 2.
    Pressure = From_Material_Definition

    #Distributed activation energy
    Activation energy st devs are 1.
    extent of reaction based on A

  end parameters for chemeq model hmx

end Aria MATERIAL hmx

BEGIN GLOBAL CONSTANTS
  Ideal Gas Constant = 1.
END GLOBAL CONSTANTS

BEGIN FINITE ELEMENT MODEL block
  Database Name = iblock.g
  decomposition method = rib

  Use material hmx for block_1
END FINITE ELEMENT MODEL block

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
    BEGIN CG SOLVER
      BEGIN JACOBI PRECONDITIONER
        END
        MAXIMUM ITERATIONS = 1000
        RESIDUAL SCALING = NONE
        CONVERGENCE TOLERANCE = 1.000000e-12
      END
    END TPETRA EQUATION SOLVER
{else}
  BEGIN AZTEC EQUATION SOLVER solve_temperature
    solution method = cg
    preconditioning method = jacobi

```

```

        maximum iterations = 1000
        residual norm scaling = NONE
        residual norm tolerance = 1.0E-12
    END AZTEC EQUATION SOLVER solve_temperature
{endif}

BEGIN procedure myProcedure

begin solution control description
  Use System Main
  Begin System Main
    Simulation Start Time      = 0.0
    Simulation Termination Time = 2.0
    Begin Transient time_block
      advance myregion
    End Transient time_block
  End System Main

  BEGIN parameters for transient time_block
    start time = 0.0
    termination time = 2.0

    BEGIN PARAMETERS FOR Aria REGION myRegion
      time step variation = fixed
      time integration method = second_order
      initial time step size = {0.01*0.5}
    END PARAMETERS FOR Aria REGION myRegion

  END parameters for transient time_block
end solution control description

begin aria region myRegion

Evaluate Postprocessor L2_A

EQ energy for temperature on all_blocks using Q1 with diff mass src
Source for Energy on all_blocks = chemeq_heating MODEL = hmx
IC for temperature on all_blocks = constant value=3

maximum nonlinear iterations = 10
nonlinear residual tolerance = 1.0E-10
nonlinear correction tolerance = 1.0E-10
Nonlinear Relaxation Factor = 1.

BEGIN CHEMEQ SOLVER FOR hmx
  ODE SOLVER = CVODE ADAMS 12 FUNCTIONAL
  Absolute Tolerance = 1e-12
  Relative Tolerance = 1e-10
  Activation Temperature = 0.0
  species A = 1.0
END CHEMEQ SOLVER FOR hmx

Begin Postprocessor Group exact_soln
  Interpolate function value of ufuncA into nodal field exact_A
End

BEGIN RESULTS OUTPUT LABEL diffusion output
  database Name = output.e
  At Step 0, Increment = 1
  Timestep Adjustment Interval = 1
  Title Aria Chem/Diffusion Verification
  Nodal Variables = solution->temperature as T
  Nodal Variables = exact_A
  Element Variables = A
END RESULTS OUTPUT LABEL diffusion output

USE FINITE ELEMENT MODEL block

```

```

usE LINEAR SOLVER solve_temperature

END Aria REGION myRegion

END procedure myProcedure

END SIERRA Aria

```

## 12.7.5. PMDI Plugin Test

```

{ECHO(OFF)}
{include("params_nom")}
{ECHO(OFF)}
{include("params")}

BEGIN SIERRA aria
Title PMDI_Plugin_Verification
load user plugin file pmdi_multispecies.so

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
  BEGIN CG SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 10000
  RESIDUAL SCALING = NONE
  CONVERGENCE TOLERANCE = 1.000000e-08
  END
END TPETRA EQUATION SOLVER
{else}
  BEGIN TRILINOS EQUATION SOLVER solve_temperature
  solution method = CG
  preconditioning method = jacobi
  maximum iterations = 10000
  residual norm tolerance = 1e-08
  residual norm scaling = NONE
  END TRILINOS EQUATION SOLVER solve_temperature
{endif}

BEGIN GLOBAL CONSTANTS
  Stefan Boltzmann Constant = 5.67e-8 # W/m^2-K^4
  ideal gas constant = 8314. # J/kgmol-K
END GLOBAL CONSTANTS

begin data block pmdi_data
real data_real = (\# Variable definition, units
  {to} \# to Initial gas temperature, K- necessary to calculate pressure for first timestep
  {vex0Tvex} \# vex/Tvex Excess volume/temp excess volume, m^3/K
  {po} \# po Initial pressure, PA
  {rbo*uden_pmdi} \# rbo Initial bulk density, kg/m^3
  {rco} \# rco Initial condensed density, kg/m^3
  {kb_1} \# kb, effective cond. for Keff, W/mK (for201b: 0.0486 0.706) radiation coefficient- 16/3/(a +sig s)
  {kb_2} \# kb, W/mK
  {t_1} \# t, K
  {t_2} \# t, K
  {rad_coef} \#
  {ukb} \#
  {ukrad} \#
  {ukeff_pmdi} \#
  {upress} )
end data block pmdi_data

BEGIN ARIA MATERIAL pmdifoam
use data block pmdi_data
Emissivity = constant e = {0.8*uemis_pmdi}
density = constant rho = {rbo*uden_pmdi} #

```

```

specific heat = constant cp = 1
tensor thermal conductivity = calore_user_sub name = ktdirpu type = element_tensor # W/m-K

Heat Conduction = generalized

BEGIN PARAMETERS FOR CHEMEQ MODEL reaction_model
  number of reactions is 3
  species names are FOAMA FOAMB FOAMC CHAR CO2 LMWO HMWO
  species phases are Condensed Condensed Condensed Condensed Gas Condensed Condensed
  condensed fraction is 0. # Not used
  steric coefficients are 0. 0. 0. # Not used
  log preexponential factors are 0. 0. 0. # Set these to 0 to prevent any reactions for the purpose of verification
  activation energies are {179441062.*uE1_pmdi} {179441062.*uE1_pmdi} {179441062.*uE1_pmdi} # J/kmol (e/R=21583 KENDATA)
  energy release units are per unit mass
  energy releases are 0 0 0 # J/Kg no energy for first cut

  # Rxn-->1 2 # Mechanism
  concentration exponents for FOAMA ARE 1. 0. 0. # A -> CO2 --> 0.45 PMDIRPU -> 0.252 CO2 + 0.198 LMWO
  concentration exponents for FOAMB ARE 0. 1. 0. # B-> HMWO --> 0.15 PMDIRPU -> 0.15 HMWO
  concentration exponents for FOAMC ARE 0. 0. 1. # C-> HMWO --> 0.4 PMDIRPU -> 0.2 HMWO+0.2 char
  concentration exponents for CHAR ARE 0. 0. 0. # 20% CHAR FORMATION
  concentration exponents for CO2 ARE 0. 0. 0. #
  concentration exponents for LMWO ARE 0. 0. 0. #
  concentration exponents for HMWO ARE 0. 0. 0.

  stoichiometric coefficients for FOAMA ARE -1.0 0.0 0.0 # dA/dt = r1
  stoichiometric coefficients for FOAMB ARE 0.0 -1.0 0.0
  stoichiometric coefficients for FOAMC ARE 0.0 0.0 -1.0
  stoichiometric coefficients for CHAR ARE 0.0 0.0 0.5
  stoichiometric coefficients for CO2 ARE +0.56 0.0 0.0 # dB/dt = 0.252/0.45 r1
  stoichiometric coefficients for LMWO ARE +0.44 0.0 0.0 # dC/dt = 0.198/0.45 r1
  stoichiometric coefficients for HMWO ARE +0.0 +1.0 +0.5 # dD/dt = r2 + r3

  aux variable names are sf, phi, keff, frxn, krad #, p, krad, kbulk
  aux variable subroutine is calcauxvar
END PARAMETERS FOR CHEMEQ MODEL reaction_model
END ARIA MATERIAL pmdifoam

BEGIN FINITE ELEMENT MODEL FoamInCan
  database name is 1block.g
  Use Material pmdifoam for block_1
END FINITE ELEMENT MODEL FoamInCan

BEGIN PROCEDURE myProcedure

begin solution control description
  use system main
  begin system main
    simulation start time = 0.0
    simulation termination time = 1.0

    begin transient solution_block_1
      advance myRegion
    end transient solution_block_1
  end system main

  begin parameters for transient solution_block_1
    start time = 0.0
    begin parameters for aria region myRegion
      time step variation = fixed
      initial time step size = 0.1
    end parameters for aria region myRegion
  end
end
end solution control description

BEGIN ARIA REGION myRegion
  use finite element model FoamInCan Model Coordinates are model_coordinates
  use linear solver solve_temperature

```

```

nonlinear solution strategy = newton
maximum nonlinear iterations = 10
nonlinear residual tolerance = 1.0e-8
nonlinear relaxation factor = 1.0
use dof averaged nonlinear residual

BEGIN CHEMEQ SOLVER FOR reaction_model
  ODE SOLVER = CVODE ADAMS 12 FUNCTIONAL
  absolute tolerance = 1e-12
  relative tolerance = 1e-9
#   aux variable names are sf, phi, rbulk, keff
  aux variable sf = 1.0 # initial solid fraction value
  aux variable phi = {phi} # initial gas volume fraction
  aux variable keff = 0. # initial effective thermal conductivity
  aux variable frxn = 1.0 # initial bulk density
  aux variable krad = 0.0 # radcond
  species FOAMA = {1./7.}
  species FOAMB = {1./7.}
  species FOAMC = {1./7.}
  species CHAR = {1./7.}
  species CO2 = {1./7.}
  species LMWO = {1./7.}
  species HMWO = {1./7.}
  minimum concentration for FOAMA = 1e-12
  minimum concentration for FOAMB = 1e-12
  minimum concentration for FOAMC = 1e-12
  chemistry step multiplier = 1E5
END CHEMEQ SOLVER FOR reaction_model

EQ ENERGY for TEMPERATURE on block_1 using Q1 with mass src
Source for energy on block_1 = chemeq_heating model=reaction_model
IC for temperature on block_1 = constant value={2*to}

#-----
# User variable definition
#-----
  Define Global Scalar gmasco2 as real operation sum initial value 0.0
  Define Global Scalar gmasn2 as real operation sum initial value 0.0
  Define Global Scalar gmaslowmw as real operation sum initial value 0.0
  Define Global Scalar gmashighmw as real operation sum initial value 0.0

  Define Global Scalar itv as real operation sum initial value 0.0
  Define Global Scalar gvtot as real operation sum initial value 0
  Define Global Scalar p as real operation min initial value 101325.0
  Define Global Scalar psig as real operation min initial value 0.0
  Define Global Scalar padmix as real operation min initial value 101325.0
  Define Global Scalar psigadmix as real operation min initial value 0.0
  Define Global Scalar mcvT as real operation sum initial value 0.0
  Define Global Scalar mcv as real operation min initial value 0.0
  Define Global Scalar gvool as real operation sum initial value 0.0

  Define Global Scalar psig1 as real operation min initial value 0.0
  Define Global Scalar psigadmix1 as real operation min initial value 0.0
  Define Global Scalar poc as real operation max initial value 0.0
  Define Global Scalar count as int operation min initial value 0

  Define Global Scalar psigxuncert as real operation min initial value 0.0
  Define Global Scalar pxuncertsig as real operation min initial value 0.0

  Define Global Scalar molesn2 as real operation min initial value 0.0
  Define Global Scalar molesco2 as real operation min initial value 0.0

  Define Global Scalar moleslowmw as real operation min initial value 0.0
  Define Global Scalar moleshighmw as real operation min initial value 0.0
  Define Global Scalar molesofv as real operation min initial value 0.0
  Define Global Scalar molestotal as real operation min initial value 0.0

```

```
BEGIN RESULTS OUTPUT output_1
  Database Name is %B.e
  Database Type is EXODUSII
  at step 0, increment is 1
  nodal variables = solution->temperature as temp
  nodal variables = solution->temperatureDot as TDOT
  element variables = Density as RHO
  element variables = FOAMA FOAMB FOAM C CO2 CHAR LMWO HMWO
  element variables = sf, phi, keff, frxn, krad
  global variables = p
  global variables = psig
END RESULTS OUTPUT output_1

END ARIA REGION myRegion
END PROCEDURE myProcedure
END SIERRA aria
```

## 12.8. MISCELLANEOUS

### 12.8.1. Thermal Postprocessing

```
BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density      = CONSTANT rho = 1
    Thermal Conductivity = Constant k  = 1
    Specific Heat  = Constant cp  = 1
    heat conduction    = basic
  END

  BEGIN ARIA MATERIAL Mathite
    Density      = CONSTANT rho = 1
    Thermal Conductivity = Constant k  = 1
    Specific Heat  = Constant cp  = 1
    heat conduction    = basic
  END

  Begin Global Constants
    Stefan Boltzmann Constant = 5.67e-8
  End

  Begin Aria Material surf_2_models
    BC Reference Temperature = encore_function name = cf_Tref
    Heat Transfer Coefficient = constant h=10.0
  End

  Begin Aria Material surf_3_models
    BC Rad Reference Temperature = encore_function name = rf_Tref
    Emissivity                    = Constant E=0.6
    Radiation form factor          = Constant F=1.0
  End

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER SOLVE_TEMPERATURE
      BEGIN CG SOLVER
        BEGIN JACOBI PRECONDITIONER
          END
          MAXIMUM ITERATIONS = 1000
          RESIDUAL SCALING = RO
          CONVERGENCE TOLERANCE = 1.000000e-14
        END
      END TPETRA EQUATION SOLVER
  {else}
    BEGIN AZTEC EQUATION SOLVER solve_temperature
      solution method = cg
      preconditioning method = jacobi
      maximum iterations = 1000
      residual norm scaling = r0
      residual norm tolerance = 1.0e-14
    END AZTEC EQUATION SOLVER solve_temperature
  {endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = cube_two_blocks_hex8_h{N}.g
    coordinate system is cartesian

    # [0,1] x [-0.5,0.5] x [-0.5,0.5]
    BEGIN PARAMETERS FOR BLOCK block_1
  material Kryptonite
    END

    # [-1,0] x [-0.5,0.5] x [-0.5,0.5]
    BEGIN PARAMETERS FOR BLOCK block_2
```

```

material Mathite
  END

  Use Material surf_2_models for surface_2
  Use Material surf_3_models for surface_3

END FINITE ELEMENT MODEL cube

# specify text output for Encore PPs
Begin Postprocessor Output Control pp_out
  Write to File encoreinfo{N}.txt
  Enable Small Output Rounding To Zero
  Floating Point Precision Is 10
End

Begin Global Function Parameters gfp
  Parameter T0 = 400 # [K]
  Parameter C0 = 2.0
  Parameter C1 = 3.0
  Parameter C2 = 4.0
  Parameter C3 = 0.4
  Parameter h = 10.0
  Parameter eps = 0.6
  Parameter sigma = 5.67e-8
End

Begin Field Function ffunc
  Use Nodal Field nonlinear_solution->TEMPERATURE
End

# exact solution
Begin User Function ufunc
  Load From File ./somefunc.so Using Function registerExactSoln
End

Begin Difference Function dfunc
  Difference Is ufunc - ffunc
End

Begin User Function src
  Integration Order Is 4
  Load From File ./somefunc.so Using Function registerSrc
End

Begin User Function cf_Tref
  Integration Order Is 4
  Load From File ./somefunc.so Using Function registerConvHeatFlux_Tref
End

# exact convective flux
Begin User Function cf_bc_exact
  Integration Order Is 4
  Load From File ./somefunc.so Using Function registerExactConvHeatFlux
End

Begin User Function rf_Tref
  Load From File ./somefunc.so Using Function registerRadFlux_Tref
End

# exact radiative flux
Begin User Function rf_bc_exact
  Integration Order Is 4
  Load From File ./somefunc.so Using Function registerExactRadFlux
End

Begin Norm Postprocessor l2_error
  Use Function ufunc
  Subtract Function ffunc

```

```

    Compute Norm L2
End

Begin Integrate Function Postprocessor cf_bc_ipo_ex
    Use Function cf_bc_exact
    Surfaces surface_2
    Disable Output
End

Begin Integrate Function Postprocessor rf_bc_ipo_ex
    Use Function rf_bc_exact
    Surfaces surface_3
    Disable Output
End

Begin Average Value Postprocessor cf_bc_ifo_ex
    Use Function cf_bc_exact
    Surfaces surface_2
    Disable Output
End

Begin Average Value Postprocessor rf_bc_ifo_ex
    Use Function rf_bc_exact
    Surfaces surface_3
    Disable Output
End

Begin Integrate Function Postprocessor src_ipo_ex
    Use Function src
    Volumes block_1 block_2
    Disable Output
End

Begin Evaluate Function Postprocessor eval_b1
    Use Function ffunc
    Evaluate Value
    # random interior point in block_2
    Location -0.151720462393008 0.146935733548329 -0.393641401879319
    parametric search tolerance 1.0e-10
End

Begin Evaluate Function Postprocessor eval_b1_ex
    Use Function ufunc
    Evaluate Value
    # random interior point in block_2
    Location -0.151720462393008 0.146935733548329 -0.393641401879319
    Disable Output
End

Begin Evaluate Function Postprocessor eval_b1b2
    Use Function ffunc
    Evaluate Value
    # random interior point on block interface (x=0)
    Location 0 0.162595269728099 -0.377464159584852
    parametric search tolerance 1.0e-10
End

Begin Evaluate Function Postprocessor eval_b1b2_ex
    Use Function ufunc
    Evaluate Value
    # random interior point on block interface (x=0)
    Location 0 0.162595269728099 -0.377464159584852
    Disable Output
End

Begin Evaluate Function Postprocessor eval_s2
    Use Function ffunc
    Evaluate Value

```

```

# random interior point on sideset 2 (z=0.5)
Location -0.855758209426849 0.159603369582751 0.5
End

Begin Evaluate Function Postprocessor eval_s2_ex
Use Function ufunc
Evaluate Value
# random interior point on sideset 2 (z=0.5)
Location -0.855758209426849 0.159603369582751 0.5
Disable Output
End

Begin Difference Postprocessor cf_bc_ipo_err
Difference is cf_bc_ipo_ex - cf_bc_ipo
End

Begin Difference Postprocessor rf_bc_ipo_err
Difference is rf_bc_ipo_ex - rf_bc_ipo
End

Begin Difference Postprocessor cf_bc_ifo_err
Difference is cf_bc_ifo_ex - cf_bc_ifo
End

Begin Difference Postprocessor rf_bc_ifo_err
Difference is rf_bc_ifo_ex - rf_bc_ifo
End

Begin Difference Postprocessor src_ipo_err
Difference is src_ipo_ex - src_ipo
End

Begin Difference Postprocessor eval_b1_err
Difference is eval_b1_ex - eval_b1
End

Begin Difference Postprocessor eval_b1b2_err
Difference is eval_b1b2_ex - eval_b1b2
End

Begin Difference Postprocessor eval_s2_err
Difference is eval_s2_ex - eval_s2
End

Begin Tabular Function Output Postprocessor tfo_sset2
Use Functions model_coordinates ffunc ufunc dfunc
Surfaces surface_2
Write To File values_sset2_{N}.dat
End

BEGIN PROCEDURE myAriaProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Transient MySolveBlock
    Advance myRegion
    End
Simulation Max Global Iterations = 1
Simulation Start Time = 0
Simulation Termination Time = 1
End
        Begin Parameters for Transient MySolveBlock
        End
    End

    BEGIN ARIA REGION myRegion

```

```

use finite element model cube
use linear solver solve_temperature

nonlinear solution strategy = newton

maximum nonlinear iterations = 10
nonlinear residual tolerance = 1.0e-16
nonlinear correction tolerance = 1.0e-12
nonlinear relaxation factor = 1.0
    use dof averaged nonlinear residual

EQ ENERGY for TEMPERATURE on all_blocks using Q1 with DIFF SRC

    MESH GROUP Dirichlet_Surface = surface_4 surface_5 surface_6 surface_7
BC Dirichlet for Temperature on Dirichlet_Surface = encore_function name=ufunc

    BC Flux for Energy on surface_2 = Generalized_Nat_Conv Power_Output=cf_bc_ipo Flux_Output=cf_bc_ifo

    BC Flux for Energy on surface_3 = Generalized_Rad Power_Output=rf_bc_ipo Flux_Output=rf_bc_ifo

SOURCE for ENERGY on all_blocks = encore_function name=src Power_Output=src_ipo

Evaluate Postprocessor eval_b1
Evaluate Postprocessor eval_b1b2
Evaluate Postprocessor eval_s2

Evaluate Postprocessor tfo_sset2

Begin Postprocessor Group zzz
    Output Number of Nodes

    Evaluate Postprocessor l2_error

    Evaluate Postprocessor cf_bc_ipo_ex
    Evaluate Postprocessor cf_bc_ipo_err

    Evaluate Postprocessor rf_bc_ipo_ex
    Evaluate Postprocessor rf_bc_ipo_err

    Evaluate Postprocessor cf_bc_ifo_ex
    Evaluate Postprocessor cf_bc_ifo_err

    Evaluate Postprocessor rf_bc_ifo_ex
    Evaluate Postprocessor rf_bc_ifo_err

    Evaluate Postprocessor src_ipo_ex
    Evaluate Postprocessor src_ipo_err

    Evaluate Postprocessor eval_b1_ex
    Evaluate Postprocessor eval_b1_err

    Evaluate Postprocessor eval_b1b2_ex
    Evaluate Postprocessor eval_b1b2_err

    Evaluate Postprocessor eval_s2_ex
    Evaluate Postprocessor eval_s2_err
End

END ARIA REGION myRegion

END PROCEDURE myAriaProcedure

END SIERRA myJob

```

## 12.8.2. Postprocess Min/Max

```
BEGIN SIERRA myJob

  BEGIN ARIA MATERIAL Kryptonite
    Density          = constant rho=1
    Thermal Conductivity = constant k=1
    Specific Heat     = Constant cp=1
    heat conduction   = basic
  END ARIA MATERIAL Kryptonite

  {if(useTpetra)}
    BEGIN TPETRA EQUATION SOLVER DIRECT_SOLVER
      BEGIN SUPERLU SOLVER
        END
      END TPETRA EQUATION SOLVER
  {else}
    Begin Trilinos Equation Solver Direct_Solver
      Solution Method = amesos-superlu
    End
  {endif}

  BEGIN FINITE ELEMENT MODEL cube
    database name = square_h{N}.e
    coordinate system is cartesian
    decomposition method = rib

    BEGIN PARAMETERS FOR BLOCK block_1
material Kryptonite
    END PARAMETERS FOR BLOCK block_1

  END FINITE ELEMENT MODEL cube

  Begin String Function exact_soln
    Value Is "sin(7*x) * sin(8*y)"
    Gradient Is "7 * cos(7*x) * sin(8*y)" "8 * sin(7*x) * cos(8*y)"
  End

  Begin String Function exact_src
    Value Is "(49 + 64) * sin(7*x) * sin(8*y)"
  End

  Begin Field Function ffunc
    Use Nodal Field nonlinear_solution->TEMPERATURE
  End

  Begin Norm Postprocessor l2
    Use Function exact_soln
    Subtract Function ffunc
    Compute Norms L2
  End

  Begin Norm Postprocessor h1
    Use Function exact_soln
    Subtract Function ffunc
    Compute Norms H1
  End

  Begin Norm Postprocessor linf
    Use Function exact_soln
    Subtract Function ffunc
    Compute Norms LInfinity
  End

  Begin Min Max Postprocessor max_node_b1
    Use Function ffunc
    Compute Max
```

```

    Volumes block_1
End

Begin Min Max Postprocessor min_node_b1
    Use Function ffunc
    Compute Min
    Volumes block_1
End

Begin Min Max Postprocessor max_node_s2
    Use Function ffunc
    Compute Max
    Surfaces surface_2 # x=1
End

Begin Min Max Postprocessor min_node_s2
    Use Function ffunc
    Compute Min
    Surfaces surface_2 # x=1
End

# code used to compute exact errors in Min Max PP
Begin String Function sfunc_max_node_b1_ex
    Value Is "1.0"
End

Begin String Function sfunc_min_node_b1_ex
    Value Is "-1.0"
End

Begin String Function sfunc_max_node_s2_ex
    Value Is "sin(7)"
End

Begin String Function sfunc_min_node_s2_ex
    Value Is "-sin(7)"
End

Begin Evaluate Function Postprocessor max_node_b1_ex
    Use Function sfunc_max_node_b1_ex
    Location 0 0 0
End

Begin Evaluate Function Postprocessor min_node_b1_ex
    Use Function sfunc_min_node_b1_ex
    Location 0 0 0
End

Begin Evaluate Function Postprocessor max_node_s2_ex
    Use Function sfunc_max_node_s2_ex
    Location 0 0 0
End

Begin Evaluate Function Postprocessor min_node_s2_ex
    Use Function sfunc_min_node_s2_ex
    Location 0 0 0
End

Begin Difference Postprocessor max_node_b1_err
    Difference is max_node_b1_ex - max_node_b1
End

Begin Difference Postprocessor min_node_b1_err
    Difference is min_node_b1 - min_node_b1_ex
End

Begin Difference Postprocessor max_node_s2_err
    Difference is max_node_s2_ex - max_node_s2

```

```

End

Begin Difference Postprocessor min_node_s2_err
  Difference is min_node_s2 - min_node_s2_ex
End
# end code used to compute exact errors in Min Max PP

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors_h{N}.dat
  Floating Point Precision Is 8
  Floating Point Format Is Scientific
End

BEGIN PROCEDURE myAriaProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential The_Time_Block
    Advance myRegion
  End
  Simulation Start Time = 0
  Simulation Termination Time = 1
End
Begin Parameters For Transient The_Time_Block
End

  End

  BEGIN ARIA REGION myRegion

Nonlinear Solution Strategy = Newton
  Maximum Nonlinear Iterations = 10
  Nonlinear Residual Tolerance = 1.0e-12
  Nonlinear Correction Tolerance = 1.0e-12
  Nonlinear Relaxation Factor = 1.0
  use dof averaged nonlinear residual

use finite element model cube
  Use Linear Solver Direct_Solver

  BC dirichlet for Temperature at surface_1 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_2 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_3 = Encore_Function Name=exact_soln
  BC dirichlet for Temperature at surface_4 = Encore_Function Name=exact_soln

EQ ENERGY for TEMPERATURE on block_1 using Q1 with DIFF SRC

Source For ENERGY on block_1 = Encore_Scalar_Function Name=exact_src

  Evaluate Postprocessor max_node_b1
  Evaluate Postprocessor min_node_b1

  Evaluate Postprocessor max_node_s2
  Evaluate Postprocessor min_node_s2

  Begin Postprocessor Group zzz
    Output Number of Nodes

Evaluate Postprocessor l2
Evaluate Postprocessor h1
  Evaluate Postprocessor linf

  Evaluate Postprocessor max_node_b1_err
  Evaluate Postprocessor min_node_b1_err
  Evaluate Postprocessor max_node_s2_err
  Evaluate Postprocessor min_node_s2_err
End

```

```

        Begin Solution Options
    post process normalized temperature on surface_2 as t_s2
    post process normalized temperature on block_1 as t_b1
        End

BEGIN RESULTS OUTPUT LABEL diffusion output
    database Name = aria_h{N}.e
    at step 0, increment = 1
    # time interval is 1.0
    title Aria cube test
    nodal variables = nonlinear_solution->TEMPERATURE as T
    global variables = t_s2 t_b1
END RESULTS OUTPUT LABEL diffusion output

        Begin History Output blah
            database Name = aria_h{N}.hist
    at time 1 interval is 1
            Variable = global t_s2
            Variable = global t_b1
        End

        END ARIA REGION myRegion

    END PROCEDURE myAriaProcedure

END SIERRA myJob

```

### 12.8.3. Local Coordinates: Cartesian

```

# Aria input file heat conduction in local
# coordinate system

BEGIN SIERRA MyProblem

    Begin User Function ufunc
        Load From File ./cartesian.so Using Function registerExactSolution
    End

    Begin Postprocessor Output Control pp_out
        Comment Character Is %
        Write To File errors{N}.dat
        Floating Point Precision Is 3
        Floating Point Format Is Scientific
    End

    load user plugin file ./cartesian.so
    load user plugin file ./cartesian.so
    load user plugin file ./cartesian.so

    begin data block region_data
#           T0   T1   Kxx Kyy Kzz Lx Ly Lz theta1 theta2
    Real r_data = 400.0 100.0 10.0 1.0 1.0 1.0 1.0 1.0 45.0 22.5
    end data block region_data

    BEGIN LOCAL COORDINATE SYSTEM CS_Block
        TYPE = Cartesian
        ORIGIN = 0.000000          0.000000          0.000000
        POINT = 0.707106781186548 0.653281482438188 0.270598050073098
        VECTOR = 0                -0.382683432365090 0.923879532511287
    END

    BEGIN ARIA MATERIAL M_Block
        DENSITY = CONSTANT rho = 0.1
        TENSOR THERMAL CONDUCTIVITY = CONSTANT XX=10.0 YY=1.0 ZZ=1.0
        SPECIFIC HEAT = CONSTANT CP = 0.5

```

```

Heat Conduction          = Generalized
END ARIA MATERIAL M_Block

{if(useTpetra)}
BEGIN TPETRA EQUATION SOLVER LINEARSOLVER
BEGIN GMRES SOLVER
  BEGIN JACOBI PRECONDITIONER
  END
  MAXIMUM ITERATIONS = 1000
  RESIDUAL SCALING = R0
  CONVERGENCE TOLERANCE = 1.000000e-12
END
END TPETRA EQUATION SOLVER
{else}
BEGIN AZTEC EQUATION SOLVER LinearSolver
SOLUTION METHOD = gmres
PRECONDITIONING METHOD = jacobi
MAXIMUM ITERATIONS = 1000
RESIDUAL NORM TOLERANCE = 1.0e-12
RESIDUAL NORM SCALING = r0
END AZTEC EQUATION SOLVER LinearSolver
{endif}

BEGIN FINITE ELEMENT MODEL FE_Block
DATABASE NAME = cartesian{N}.g
coordinate system is cartesian
decomposition method = rcb

BEGIN PARAMETERS FOR BLOCK block_1
MATERIAL M_Block
LOCAL COORDINATE SYSTEM = CS_Block
END PARAMETERS FOR BLOCK block_1
END

BEGIN PROCEDURE MyProcedure

  Begin Solution Control Description
Use System Main
Begin System Main
  Begin Sequential MyBlock
  Advance Region_Block
  End
End

  End

BEGIN ARIA REGION Region_Block
  nonlinear solution strategy = newton
  use data block region_data

  Output Number of Nodes
  Compute Difference L2      Of ufunc solution->temperature Store In l2_error_norm2
  Compute Difference LInfinity Of ufunc solution->temperature Store In linf_error_norm

  NONLINEAR RESIDUAL TOLERANCE = 1.0e-10
  NONLINEAR CORRECTION TOLERANCE = 1.0e-10
  MAXIMUM NONLINEAR ITERATIONS = 10
  NONLINEAR RELAXATION FACTOR = 1.0

  IC for temperature on block_1 = calore_user_sub name = localCoord_ic type=node

  BC dirichlet for temperature on surface_1 = calore_user_sub name = localCoord_bc type=node
  BC dirichlet for temperature on surface_2 = calore_user_sub name = localCoord_bc type=node
  BC dirichlet for temperature on surface_3 = calore_user_sub name = localCoord_bc type=node
  BC dirichlet for temperature on surface_4 = calore_user_sub name = localCoord_bc type=node
  BC dirichlet for temperature on surface_5 = calore_user_sub name = localCoord_bc type=node
  BC dirichlet for temperature on surface_6 = calore_user_sub name = localCoord_bc type=node

#  IC CONST ON block_1 Temperature = 0.0

```

```

# # along z
# BC Const Dirichlet on surface_1 Temperature = 100.0
# BC Const Dirichlet on surface_2 Temperature = 0.0
# # along y
# BC Linear Dirichlet on surface_3 Temperature Coeff = 50. 0. -38.37 92.39
# BC Linear Dirichlet on surface_5 Temperature Coeff = 50. 0. -38.37 92.39
# # along x
# BC Linear Dirichlet on surface_4 Temperature Coeff = 50. 0. -38.37 92.39
# BC Linear Dirichlet on surface_6 Temperature Coeff = 50. 0. -38.37 92.39

EQ ENERGY FOR TEMPERATURE ON block_1 USING Q1 WITH DIFF #SRC
#SOURCE for Temperature on block_1 =

Begin Volume Heating juan
  add volume block_1
  element subroutine = localCoord_vhs
End

USE FINITE ELEMENT MODEL FE_Block

BEGIN RESULTS OUTPUT TemperatureOutput
  DATABASE NAME = output{N}.e
  AT STEP 1, INCREMENT = 1
  TITLE Aria Temperature in Local Coordinate System Verification Problem
  NODAL VARIABLES = solution->TEMPERATURE AS T
  Element Variables = l2_error_norm2 as l2error
  Element Variables = linf_error_norm as linf
END RESULTS OUTPUT TemperatureOutput

USE LINEAR SOLVER LinearSolver
END

END
END SIERRA MyProblem

```

## 12.8.4. Local Coordinates: Cylindrical

```

# Aria input file heat conduction in local
# coordinate system

BEGIN SIERRA MyProblem

  Begin Field Function ffunc
    Use Nodal Field solution->temperature
  End

  Begin User Function ufunc
    Load From File ./cylindrical.so Using Function registerExactSolution
  End

  Begin Definition for Function krr
    Type is piecewise linear
    Begin Values
      0 1.0
      400 1.0
    End Values
    Scale by 10.0
  End

  Begin Definition for Function ktt
    Type is piecewise linear
    Begin Values
      0 1.0
      400 1.0
    End Values
    Scale by 1.0
  End

```

```

End

Begin Definition for Function kzz
  Type is piecewise linear
  Begin Values
    0 1.0
    400 1.0
  End Values
End

Begin Postprocessor Output Control pp_out
  Comment Character Is %
  Write To File errors{N}.dat
  Floating Point Precision Is 6
  Floating Point Format Is Scientific
End

load user plugin file ./cylindrical.so

begin data block region_data
#      TO    T1   Krr  Ktt  Kzz  Lx  Lz  theta1  theta2
  Real r_data = 400.0 100.0 10.0 1.0 1.0 1.0 1.0 45.0  22.5
end data block region_data

BEGIN LOCAL COORDINATE SYSTEM CS_Block
  TYPE = Cylindrical
  ORIGIN = 0.000000      0.000000      0.000000
  POINT = 0.707106781186548 0.653281482438188 0.270598050073098
  VECTOR = 0      -0.382683432365090 0.923879532511287
END

BEGIN ARIA MATERIAL M_Block
  DENSITY = CONSTANT rho = 0.1
  #tensor thermal conductivity = user_function X=Temperature Name_XX=krr Name_YY=ktt Name_ZZ=kzz
  TENSOR THERMAL CONDUCTIVITY = CONSTANT XX=10.0 YY=1.0 ZZ=1.0
  SPECIFIC HEAT = CONSTANT CP = 0.5
  Heat Conduction = Generalized
END ARIA MATERIAL M_Block

{if(useTpetra)}
  BEGIN TPETRA EQUATION SOLVER LINEARSOLVER
    BEGIN GMRES SOLVER
      BEGIN JACOBI PRECONDITIONER
        END
      MAXIMUM ITERATIONS = 1000
      RESIDUAL SCALING = R0
      CONVERGENCE TOLERANCE = 1.000000e-12
    END
  END TPETRA EQUATION SOLVER
{else}
  BEGIN AZTEC EQUATION SOLVER LinearSolver
    SOLUTION METHOD = gmres
    PRECONDITIONING METHOD = jacobi
    MAXIMUM ITERATIONS = 1000
    RESIDUAL NORM TOLERANCE = 1.0e-12
    RESIDUAL NORM SCALING = r0
  END AZTEC EQUATION SOLVER LinearSolver
{endif}

BEGIN FINITE ELEMENT MODEL FE_Block
  DATABASE NAME = cylindrical{N}.g
  coordinate system is cartesian
  decomposition method = rcb

  BEGIN PARAMETERS FOR BLOCK block_1
    MATERIAL M_Block
    LOCAL COORDINATE SYSTEM = CS_Block
  END PARAMETERS FOR BLOCK block_1

```

```

END

BEGIN PROCEDURE MyProcedure

    Begin Solution Control Description
Use System Main
Begin System Main
    Begin Sequential MyBlock
        Advance Region_Block
    End
End
    End

BEGIN ARIA REGION Region_Block
    nonlinear solution strategy = newton
    use data block region_data

    Output Number of Nodes
    Compute Difference L2      Of ufunc ffunc Store In l2_error_norm2
    Compute Difference LInfinity Of ufunc ffunc Store In linf_error_norm

    NONLINEAR RESIDUAL TOLERANCE = 1.0e-10
    NONLINEAR CORRECTION TOLERANCE = 1.0e-10
    MAXIMUM NONLINEAR ITERATIONS = 10
    NONLINEAR RELAXATION FACTOR = 1.0

    BC dirichlet for temperature on surface_1 = calore_user_sub name = localCoord_bc type=node
    BC dirichlet for temperature on surface_2 = calore_user_sub name = localCoord_bc type=node
    BC dirichlet for temperature on surface_3 = calore_user_sub name = localCoord_bc type=node

    EQ ENERGY FOR TEMPERATURE ON block_1 USING Q1 WITH DIFF SRC

    Begin Volume Heating juan
        add volume block_1
        element subroutine = localCoord_vhs
    End

    Begin Initial Condition BlockName
        All Volumes
        Temperature = 400.0
    End

    USE FINITE ELEMENT MODEL FE_Block

    Interpolate Function Value of ufunc Into Nodal Field Tex

    BEGIN RESULTS OUTPUT TemperatureOutput
        DATABASE NAME = output{N}.e
        AT STEP 1, INCREMENT = 1
        TITLE Aria Temperature in Local Coordinate System Verification Problem
        NODAL VARIABLES = solution->TEMPERATURE AS T
        NODAL VARIABLES = Tex
        Element Variables = l2_error_norm2 as l2error
        Element Variables = linf_error_norm as linf
    END RESULTS OUTPUT TemperatureOutput

    USE LINEAR SOLVER LinearSolver
END

END
END SIERRA MyProblem

```



# Appendices



# DISTRIBUTION

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Technical Library	01177	libref@sandia.gov







Sandia  
National  
Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.