Encore: A Toolkit for Verification, Postprocessing, Error Estimation and Adaptivity

SAND2008-0143P

Derek Gaston, Brian Carnes, Kevin Copps

drgasto@sandia.gov
bcarnes@sandia.gov
kdcopps@sandia.gov

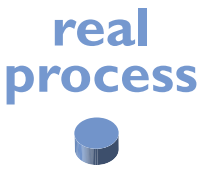
January 10, 2008

Sandia National Laboratories

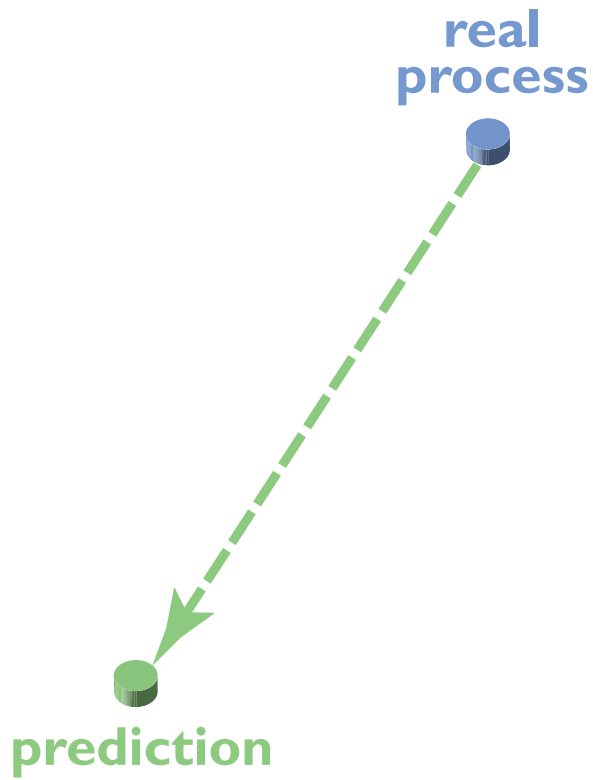
ADVANCED SIMULATION & COMPUTING™

Outline

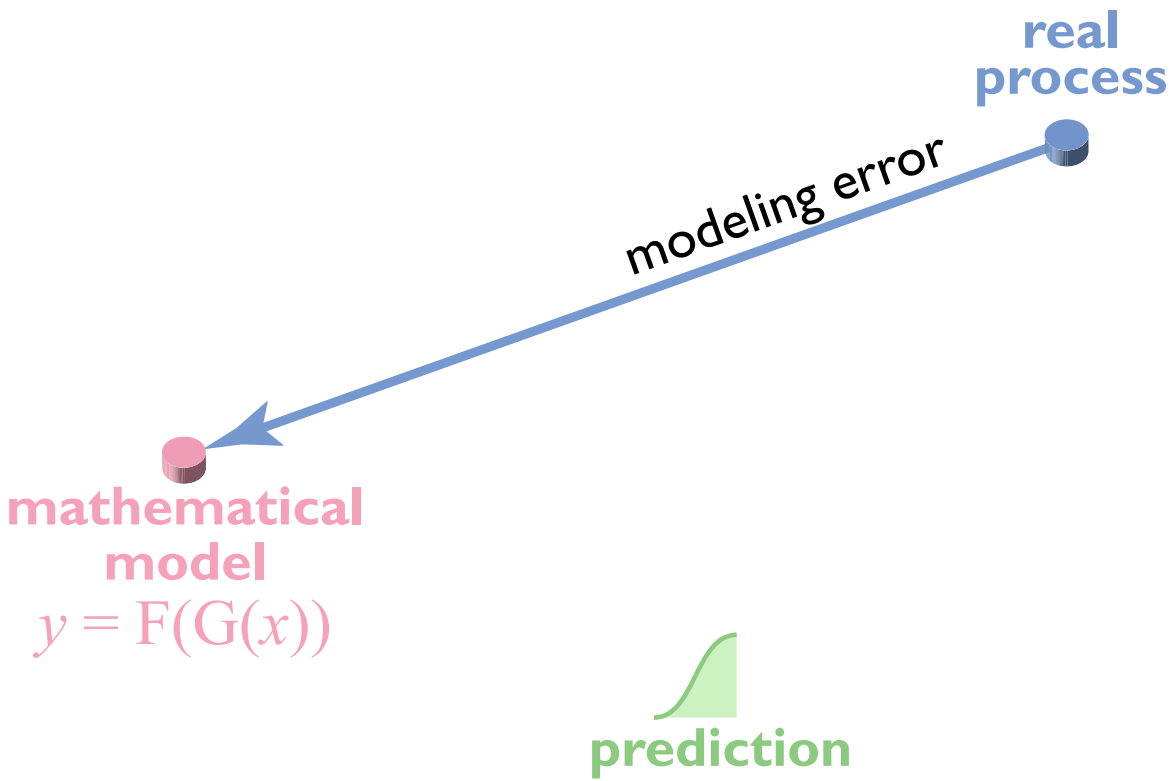
Errors and Uncertainties



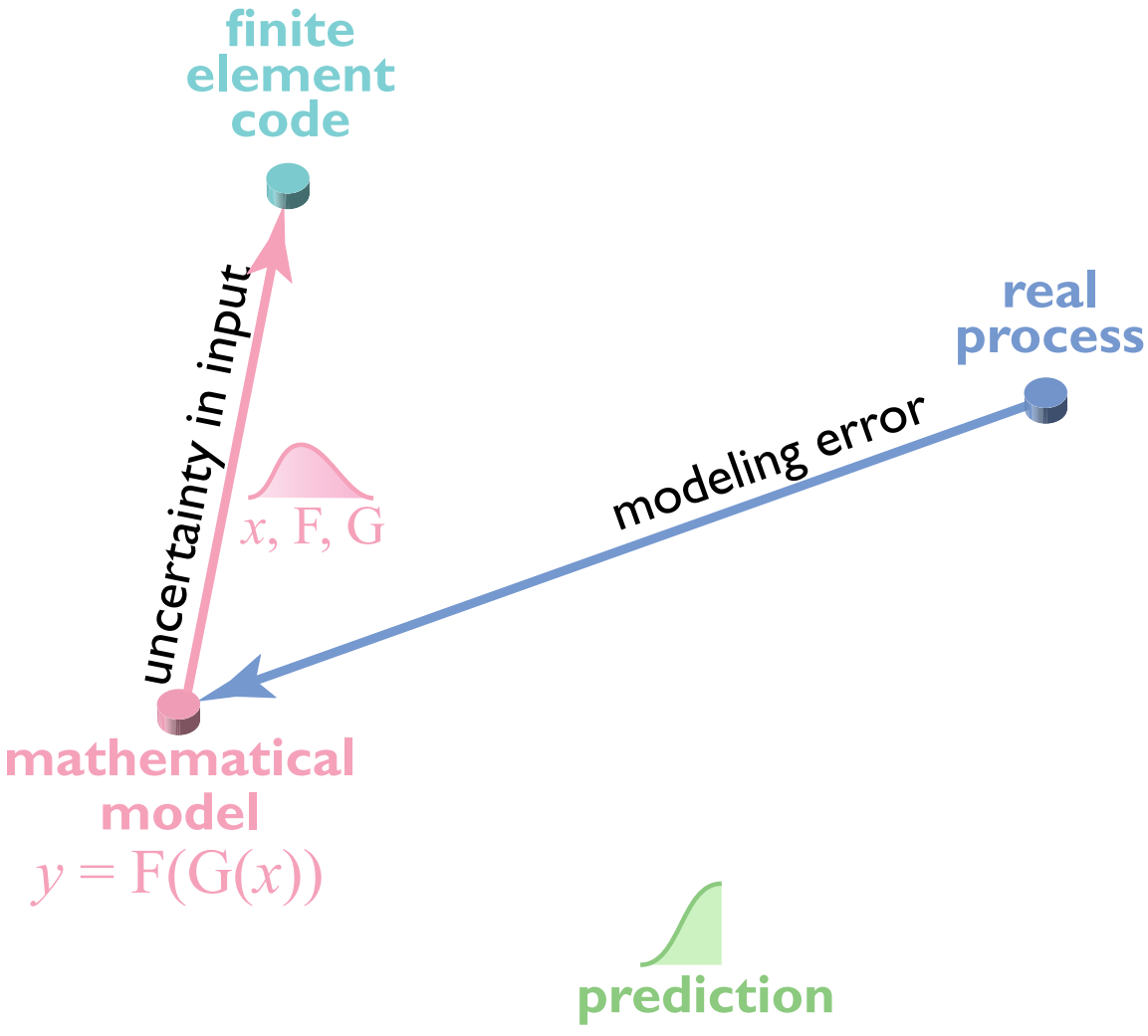
Errors and Uncertainties



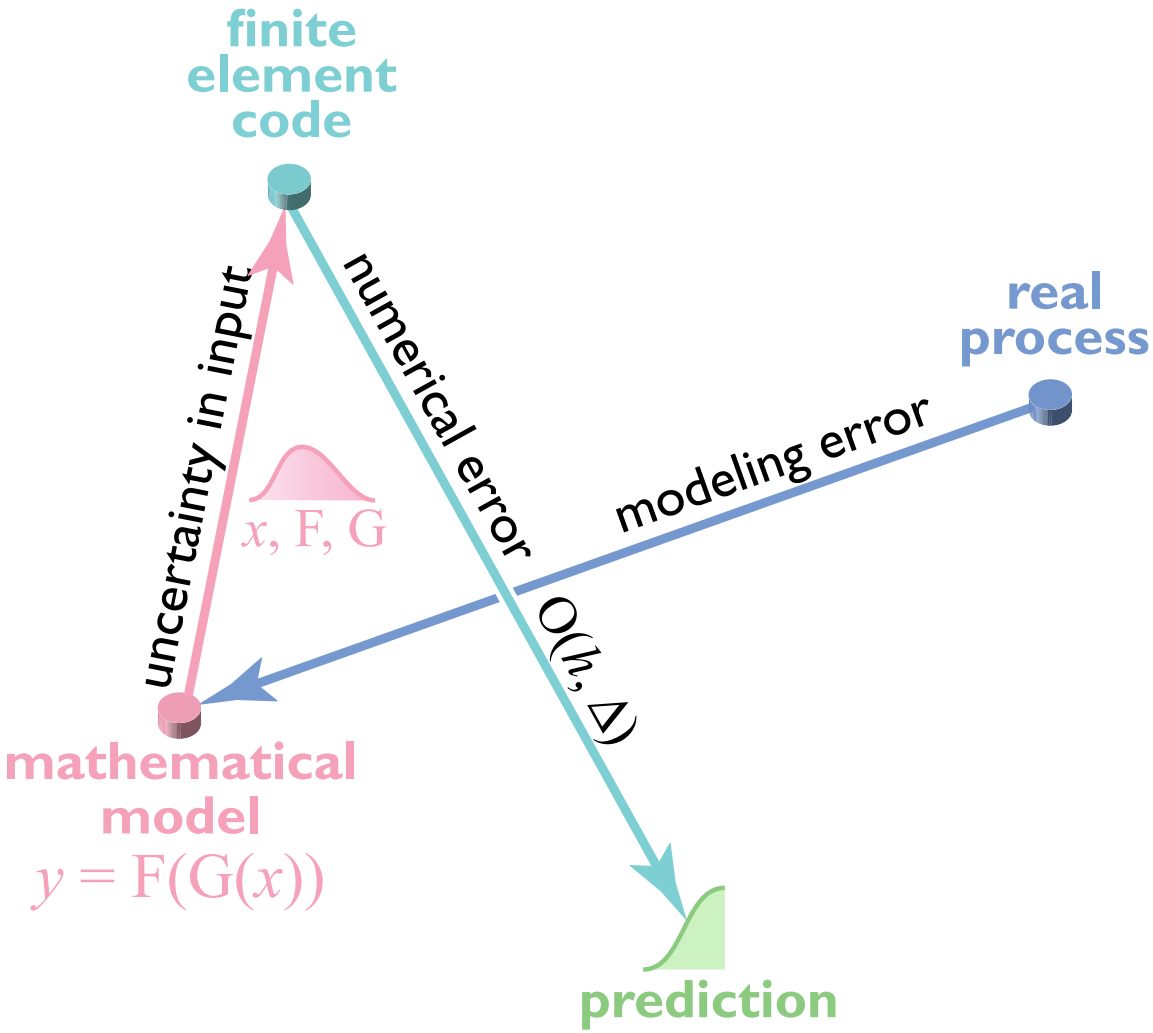
Errors and Uncertainties



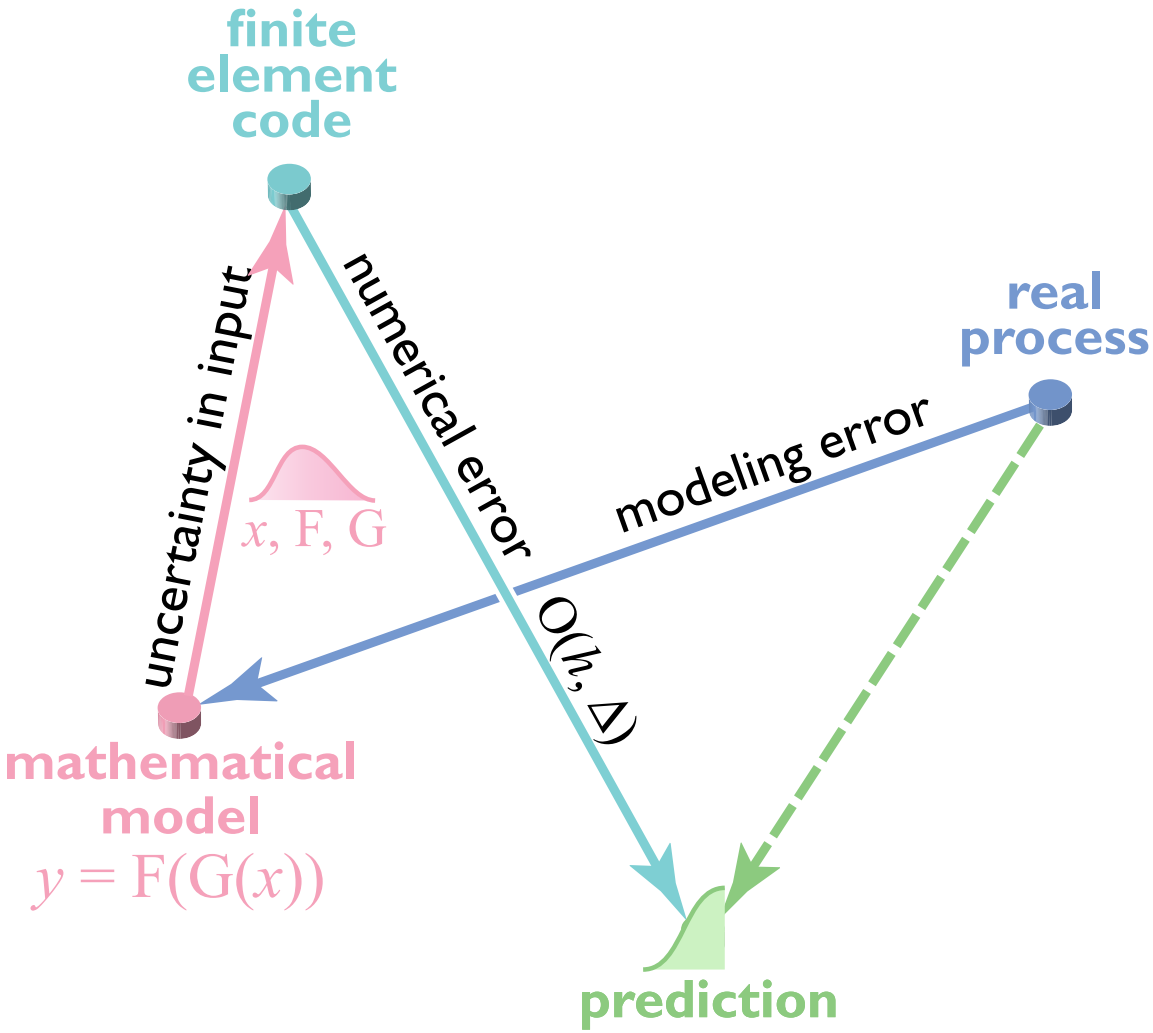
Errors and Uncertainties



Errors and Uncertainties



Errors and Uncertainties



Strategic Goals of Encore

Enable **predictive** simulation

- Unified, modular services for code and solution verification.
- Manage tradeoff of resources versus accuracy.
- Make it easy for users and developers to provide extensions.
- Bridge between ASC codes and UQ tools (Dakota, Trilinos).

Strategic Goals of Encore

Enable **predictive** simulation

- Unified, modular services for code and solution verification.
- Manage tradeoff of resources versus accuracy.
- Make it easy for users and developers to provide extensions.
- Bridge between ASC codes and UQ tools (Dakota, Trilinos).

Strategic Goals of Encore

Enable **predictive** simulation

- Unified, modular services for code and solution verification.
- Manage tradeoff of resources versus accuracy.
- Make it easy for users and developers to provide extensions.
- Bridge between ASC codes and UQ tools (Dakota, Trilinos).

Strategic Goals of Encore

Enable **predictive** simulation

- Unified, modular services for code and solution verification.
- Manage tradeoff of resources versus accuracy.
- Make it easy for users and developers to provide extensions.
- Bridge between ASC codes and UQ tools (Dakota, Trilinos).

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Features

- Code/Solution Verification
  - Facilities for generating and driving manufactured solutions.
  - Comparison between two simulations or simulation and analytic.
  - Tools for order of accuracy verification and extrapolation.
- Postprocessing
  - Calculating derived (response, observable) quantities.
  - Extendable through simple user subroutine capability.
- Error Estimation
  - Suite of physics independent error indicators.
  - Support for physics dependent error estimation. (Adjoint, etc.)
- Adaptivity
  - New flexible, user driven adaptive system.
  - Markers based on features or error.
  - Consolidation of Sierra adaptivity infrastructure.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Encore Vitals

- Version 1.0 released Oct. 2007
  Presentation Nov. 5th, 2007
  Hands-On training Dec. 6th, 2007
- Funded in part by ASC Algorithms and V&V Program.
- Software Quality Data:
  - Trackers Submitted: 233 Closed: 152
  - Lines of Code: 30,000 (about 1/3 of **Calore**)
  - Regression Tests: Over 120
    75% Line Coverage
- Online library linked with any Sierra Mechanics application.
  Currently **Adagio**, **Aria** and **Calore**.
- Off-line stand-alone executable.
  Physics independent, read and write Exodus files.

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

Leveraging SIERRA Mechanics Technology

- Can be run from any Sierra Mechanics supported platform.
- **Encore** output is either Exodus file(s) or tabular data.
  No graphical output.
- Ability to process application's runtime data.
- SIERRA Framework Features used by **Encore**:
  - Parallel load balancing
  - Distributed I/O database
  - Element library (2D and 3D, unstructured grids)
  - Data transfers across different grids
  - Dynamic mesh refinement/coarsening and rebalancing
  - Curved geometry definition

part 1

Verification and Postprocessing

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

Overview of Verification and Postprocessing

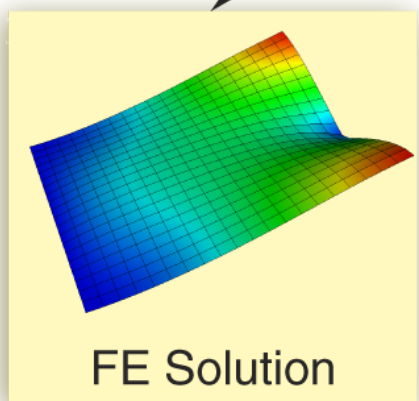
- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

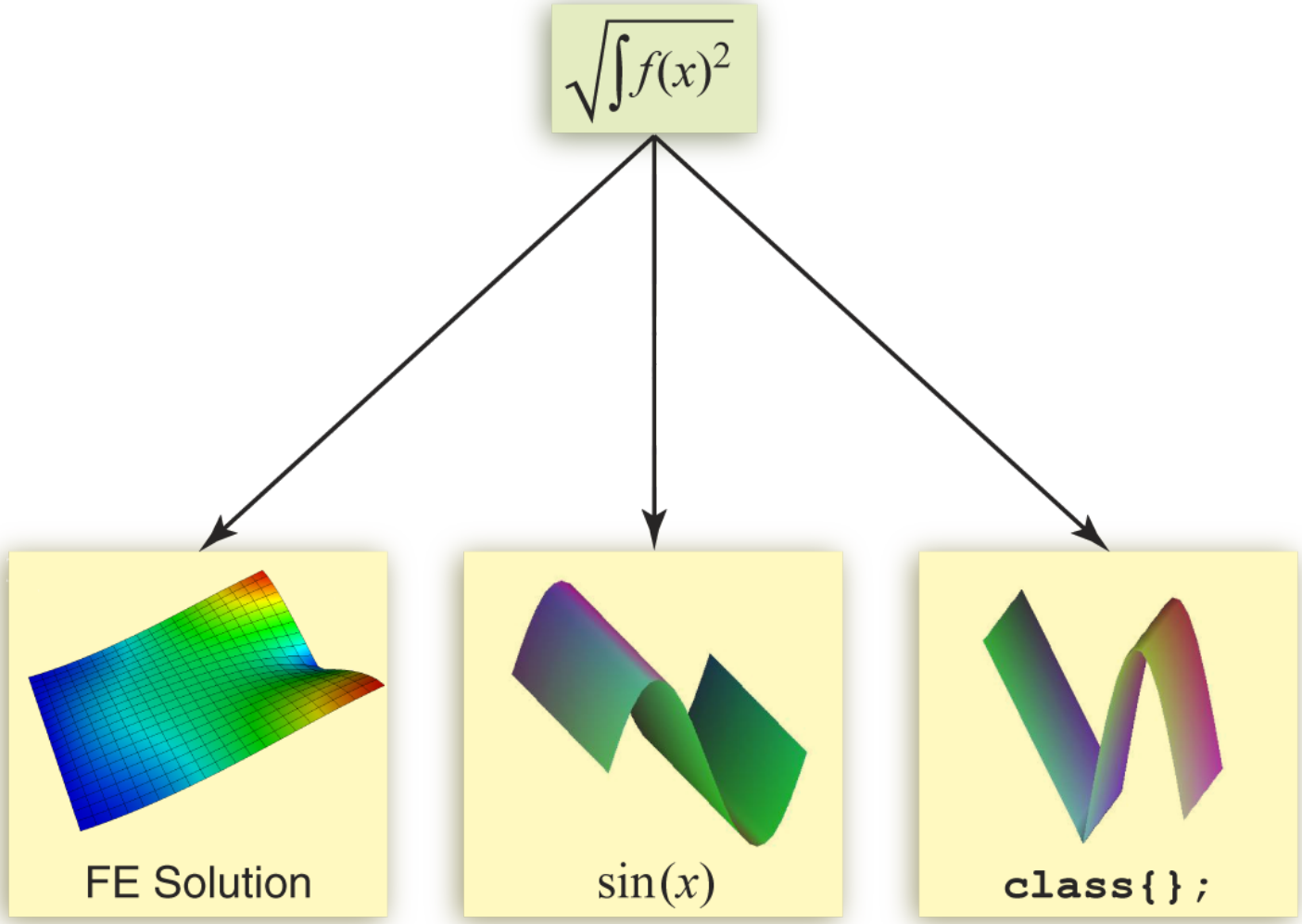
Overview of Verification and Postprocessing

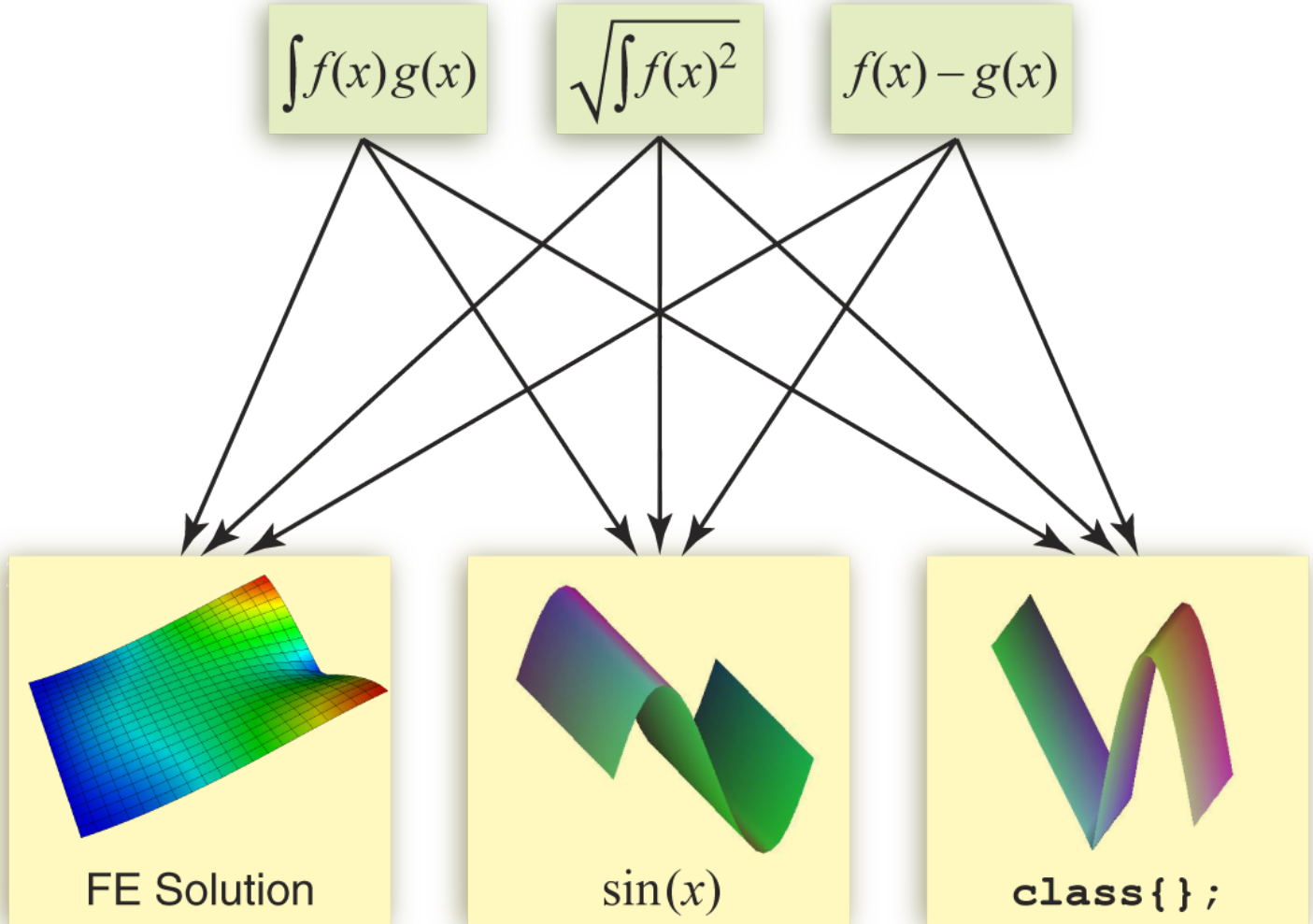
- Function Interface
- Postprocessors
  - Evaluation
  - Norms
  - Integrals
  - Import/Create
  - Interpolate
  - Patch Recovery
  - User Subroutines
- Grid to grid transfers

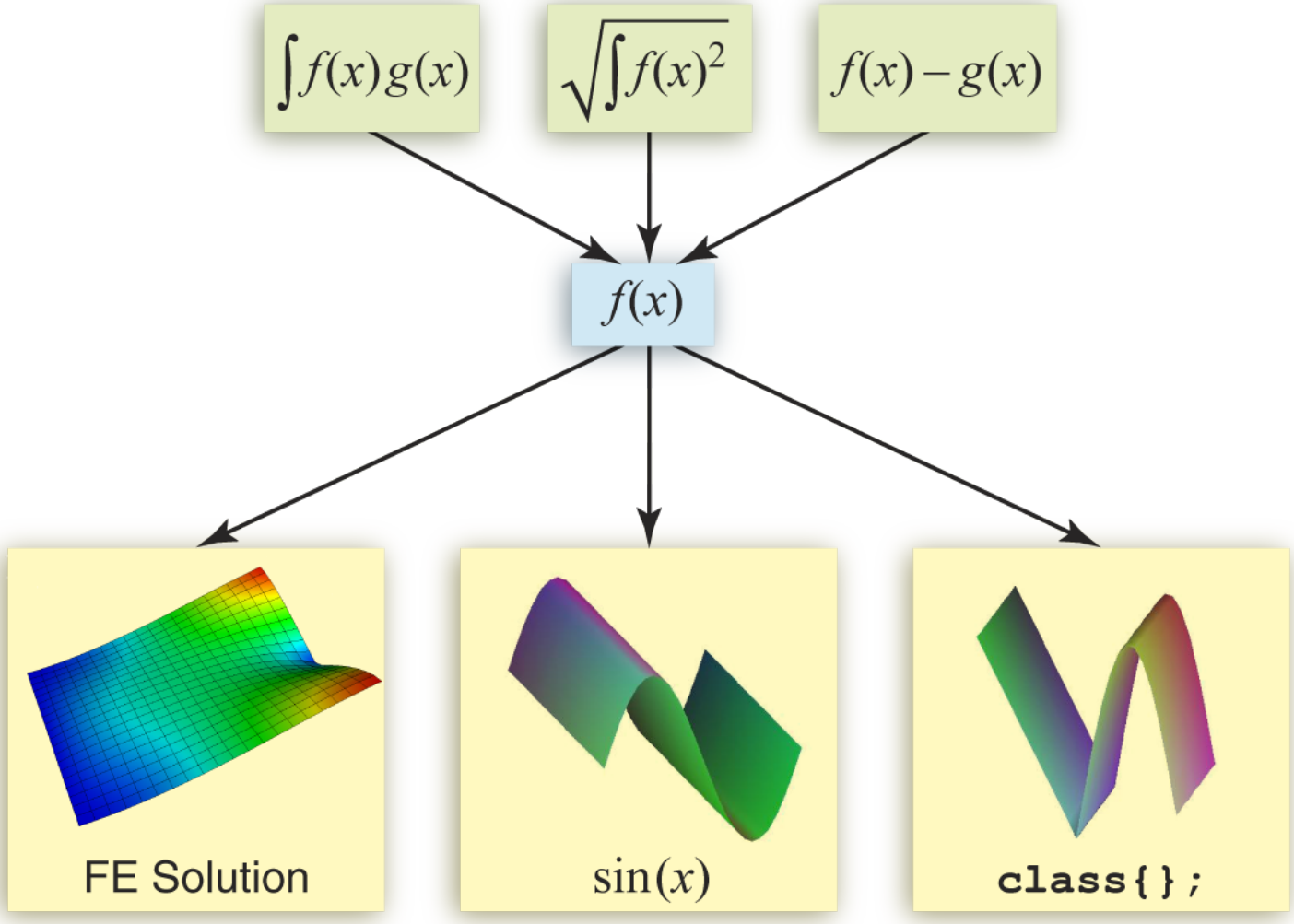
$$\sqrt{\int f(x)^2}$$

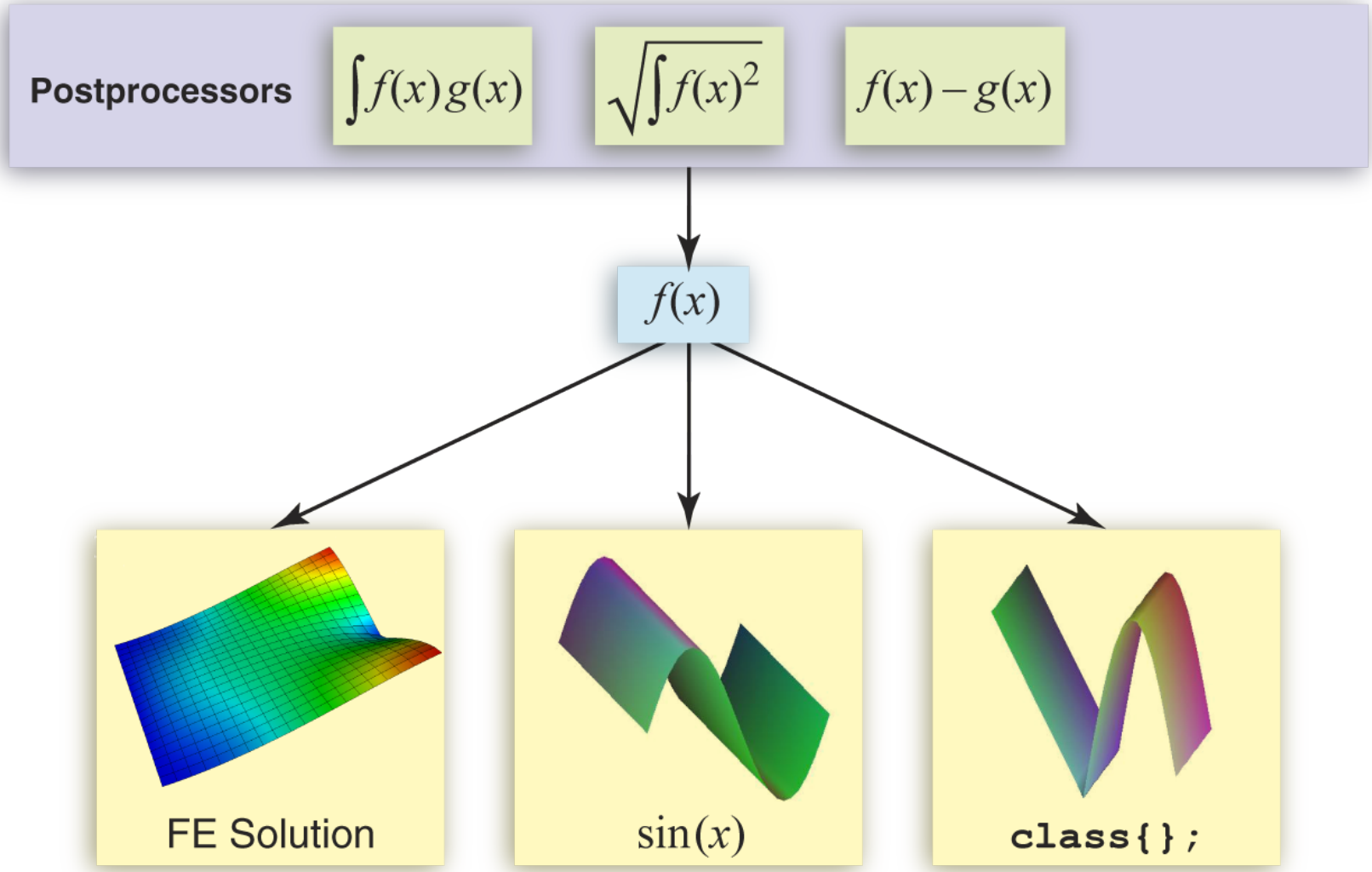
$$\sqrt{\int f(x)^2}$$

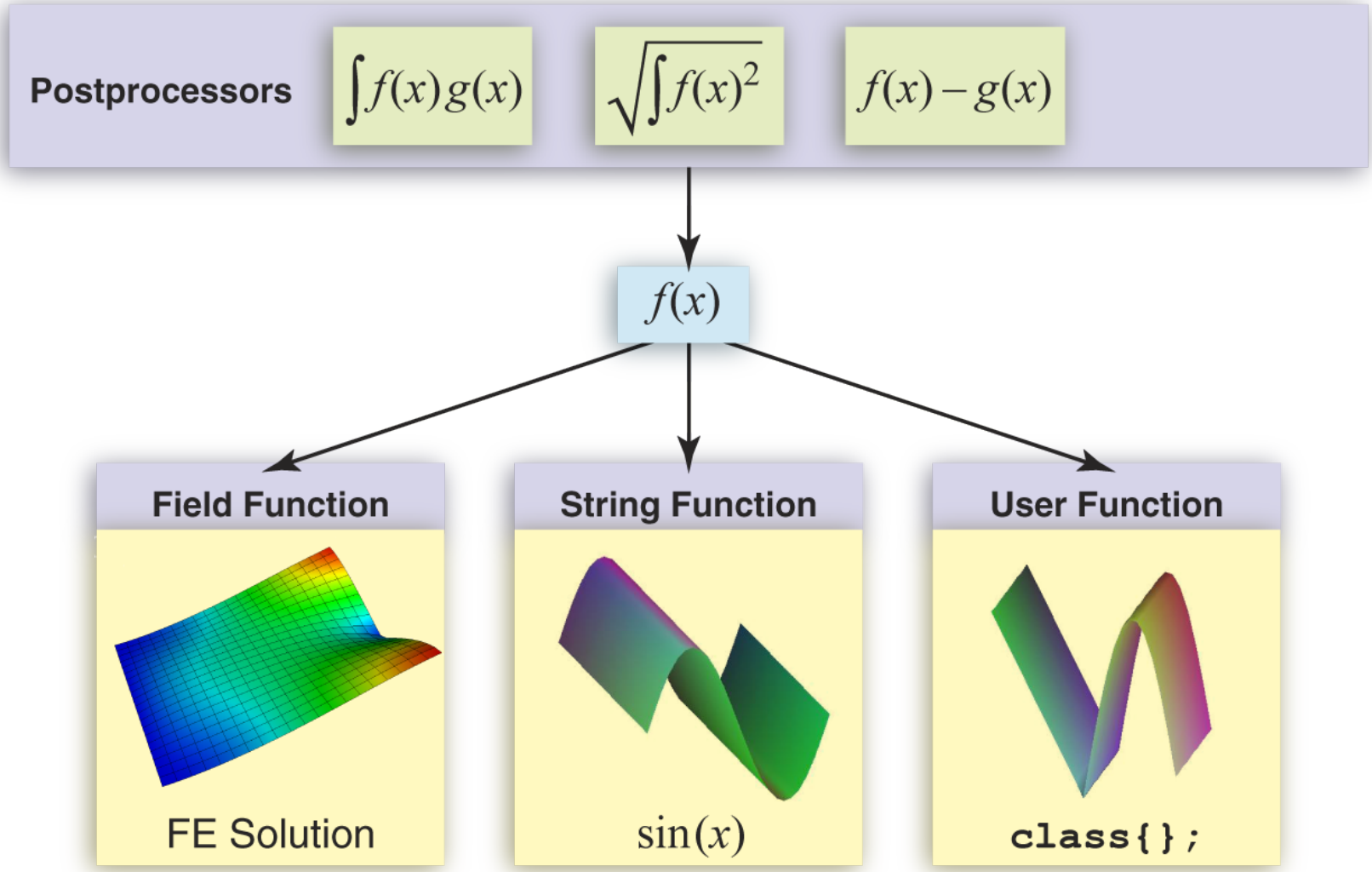


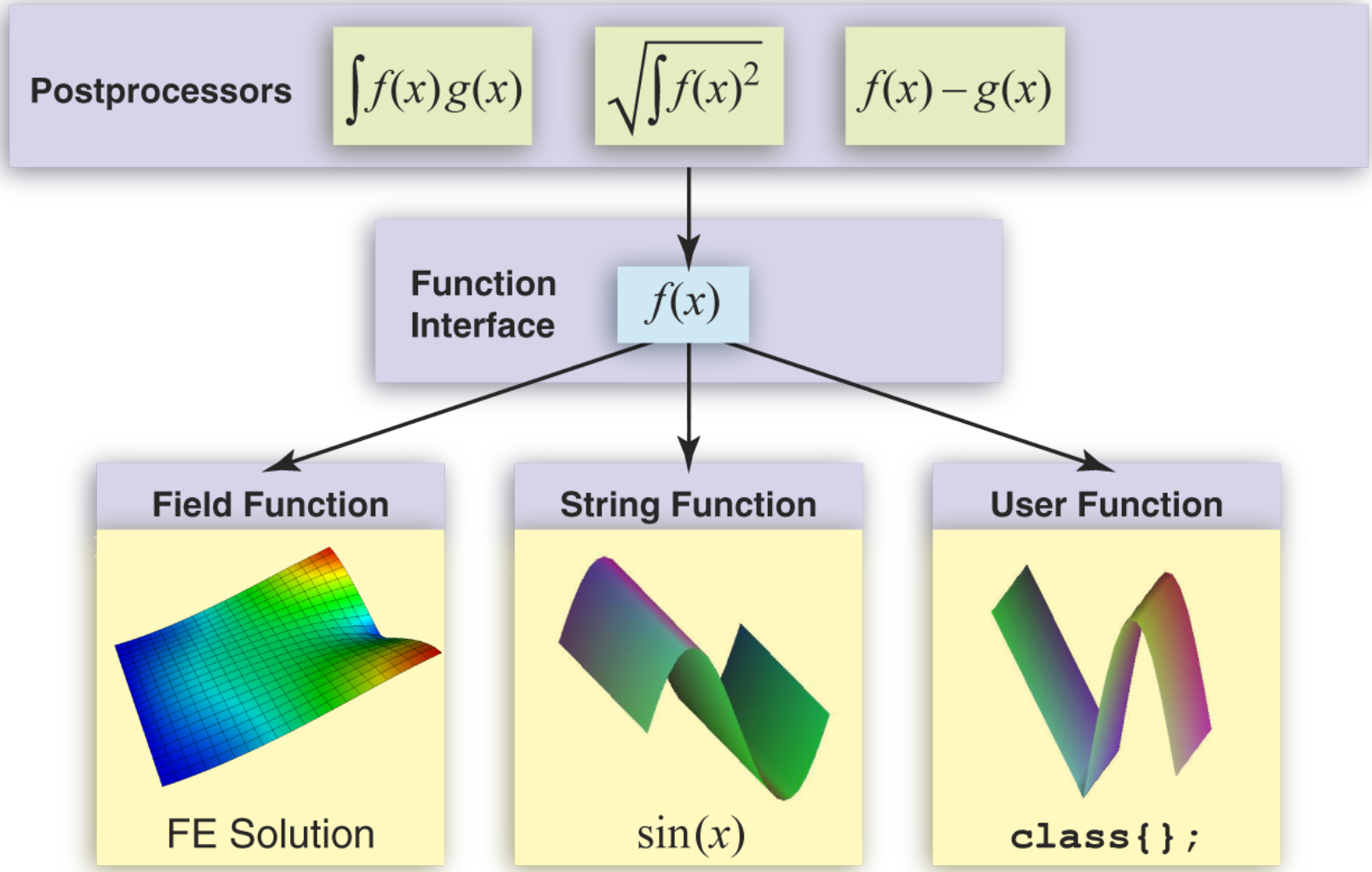

$\sqrt{\int f(x)^2}$
FE Solution
$\sin(x)$
class{};


$\int f(x)g(x)$
$\sqrt{\int f(x)^2}$
$f(x)-g(x)$
FE Solution
$\sin(x)$
class{};


$\int f(x)g(x)$
$\sqrt{\int f(x)^2}$
$f(x)-g(x)$
$f(x)$
FE Solution
$\sin(x)$
class{};


Postprocessors
$\int f(x)g(x)$
$\sqrt{\int f(x)^2}$
$f(x)-g(x)$
$f(x)$
FE Solution
$\sin(x)$
class{};


Postprocessors
$\int f(x)g(x)$
$\sqrt{\int f(x)^2}$
$f(x)-g(x)$
$f(x)$
Field Function
FE Solution
String Function
$\sin(x)$
User Function
class{};


Postprocessors
$\int f(x)g(x)$
$\sqrt{\int f(x)^2}$
$f(x)-g(x)$
Function Interface
$f(x)$
Field Function
FE Solution
String Function
$\sin(x)$
User Function
class{};

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$$f(x)$$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Function—abstract interface

$f(x)$

- Users do not need to know implementation details.
- Functions can represent analytic expressions or simulation results.
- Users can mix and match types of functions.
- Functions can be scalar/vector/tensor valued.
- All types of functions provide:
  - Value
  - Gradient
  - Time derivative
  - Flux
  - Stress

Types of Functions

- A **Field Function** represents simulation fields on a mesh.
- **String Functions** are analytic expressions that users type in an input file.
  - For example: "`x + a * sin(y)`".
- **User Functions** are C++ code that users write, allowing piecewise defined values and series solutions. They are dynamically linked into a SIERRA Mechanics applications—fast and easy.
- Function combinations: **Difference Function**, **Product Function**.
  - For example: the difference between a String Function and Field Function

$$error = u(\mathbf{x}, t) - u_h(\mathbf{x}, t)$$

Types of Functions

- A **Field Function** represents simulation fields on a mesh.
- **String Functions** are analytic expressions that users type in an input file.
  - For example: "`x + a * sin(y)`".
- **User Functions** are C++ code that users write, allowing piecewise defined values and series solutions. They are dynamically linked into a SIERRA Mechanics applications—fast and easy.
- Function combinations: **Difference Function**, **Product Function**.
  - For example: the difference between a String Function and Field Function

$$error = u(\mathbf{x}, t) - u_h(\mathbf{x}, t)$$

Types of Functions

- A **Field Function** represents simulation fields on a mesh.
- **String Functions** are analytic expressions that users type in an input file.
  - For example: "`x + a * sin(y)`".
- **User Functions** are C++ code that users write, allowing piecewise defined values and series solutions. They are dynamically linked into a SIERRA Mechanics applications—fast and easy.
- Function combinations: **Difference Function**, **Product Function**.
  - For example: the difference between a String Function and Field Function

$$error = u(\mathbf{x}, t) - u_h(\mathbf{x}, t)$$

Types of Functions

- A **Field Function** represents simulation fields on a mesh.
- **String Functions** are analytic expressions that users type in an input file.
  - For example: "`x + a * sin(y)`".
- **User Functions** are C++ code that users write, allowing piecewise defined values and series solutions. They are dynamically linked into a SIERRA Mechanics applications—fast and easy.
- Function combinations: **Difference Function**, **Product Function**.
  - For example: the difference between a String Function and Field Function

$$error = u(\mathbf{x}, t) - u_h(\mathbf{x}, t)$$

Types of Functions

- A **Field Function** represents simulation fields on a mesh.
- **String Functions** are analytic expressions that users type in an input file.
  - For example: "`x + a * sin(y)`".
- **User Functions** are C++ code that users write, allowing piecewise defined values and series solutions. They are dynamically linked into a SIERRA Mechanics applications—fast and easy.
- Function combinations: **Difference Function**, **Product Function**.
  - For example: the difference between a String Function and Field Function

$$error = u(\mathbf{x}, t) - u_h(\mathbf{x}, t)$$

Types of Functions

- A **Field Function** represents simulation fields on a mesh.
- **String Functions** are analytic expressions that users type in an input file.
  - For example: "`x + a * sin(y)`".
- **User Functions** are C++ code that users write, allowing piecewise defined values and series solutions. They are dynamically linked into a SIERRA Mechanics applications—fast and easy.
- Function combinations: **Difference Function**, **Product Function**.
  - For example: the difference between a String Function and Field Function

$$error = u(\mathbf{x}, t) - u_h(\mathbf{x}, t)$$

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

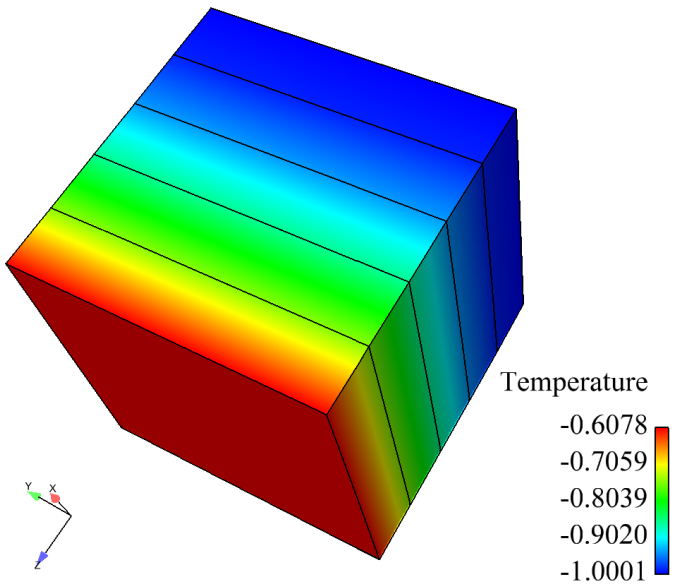
Postprocessors

- Postprocessors (PPs) are calculations that run:
  - After initial conditions,
  - After a non-linear step,
  - After every timestep,
  - At a certain simulation time.
- Input to most PPs are Functions.
- Output of PPs are
  - Simulation fields (on nodes, elements, etc.)
  - Formatted tables in a text file—for Excel/Matlab/etc.
- PPs are fully parallel.
- Can be easily extended by users.

Evaluate Postprocessor

- In order to compare to experimental values, analysts need to probe simulation values.
- The **Evaluation PP** takes a Function and evaluates it at any point in space/time.
- Any type of evaluation (Value, Gradient, Time Derivative, Stress, Flux) can be performed on a function.
- In order to perform code verification, a user could evaluate the difference between the simulation and an analytic solution while the simulation is running.

Example: Evaluate the simulation field temperature at a point, $T(0, 0, 0)$

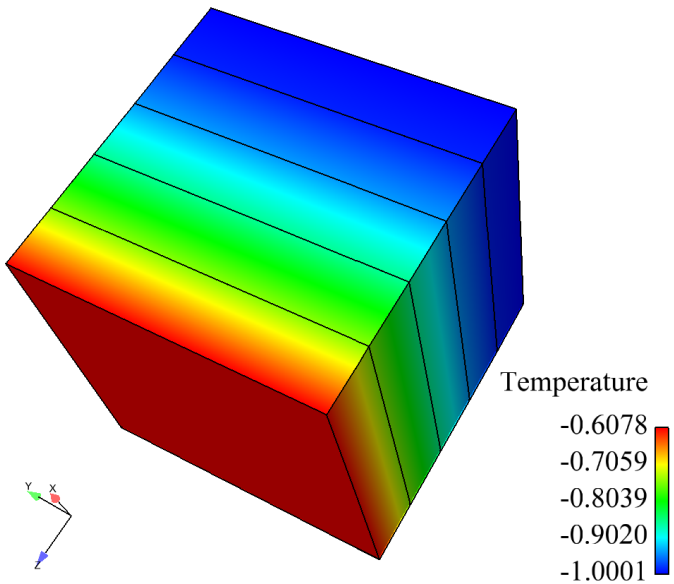



```
encorereg Postprocessor Output for Group agroup
Time  Evaluate: ffunc
----- ---------------
    0       -0.901515
0.005       -0.900399
 0.01       -0.899257
0.015       -0.898088
```

Evaluate Postprocessor

- In order to compare to experimental values, analysts need to probe simulation values.
- The **Evaluation PP** takes a Function and evaluates it at any point in space/time.
- Any type of evaluation (Value, Gradient, Time Derivative, Stress, Flux) can be performed on a function.
- In order to perform code verification, a user could evaluate the difference between the simulation and an analytic solution while the simulation is running.

Example: Evaluate the simulation field temperature at a point, $T(0, 0, 0)$

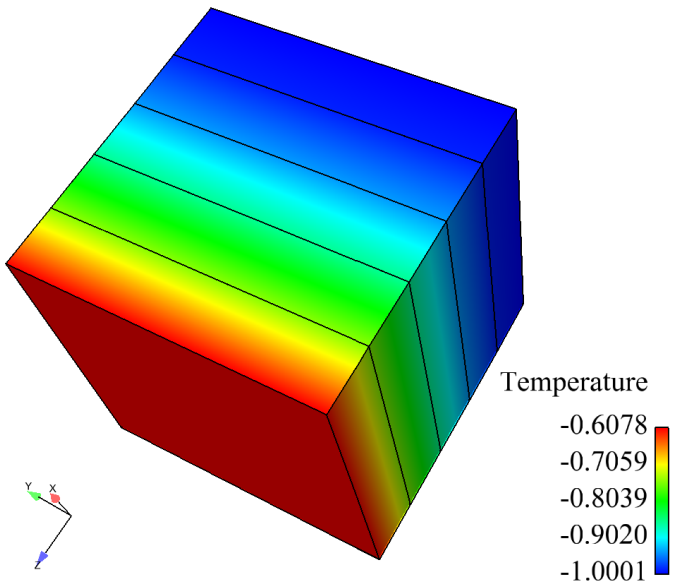



```
encorereg Postprocessor Output for Group agroup
Time  Evaluate: ffunc
----- ---------------
    0       -0.901515
0.005       -0.900399
 0.01       -0.899257
0.015       -0.898088
```

Evaluate Postprocessor

- In order to compare to experimental values, analysts need to probe simulation values.
- The **Evaluation PP** takes a Function and evaluates it at any point in space/time.
- Any type of evaluation (Value, Gradient, Time Derivative, Stress, Flux) can be performed on a function.
- In order to perform code verification, a user could evaluate the difference between the simulation and an analytic solution while the simulation is running.

Example: Evaluate the simulation field temperature at a point, $T(0, 0, 0)$

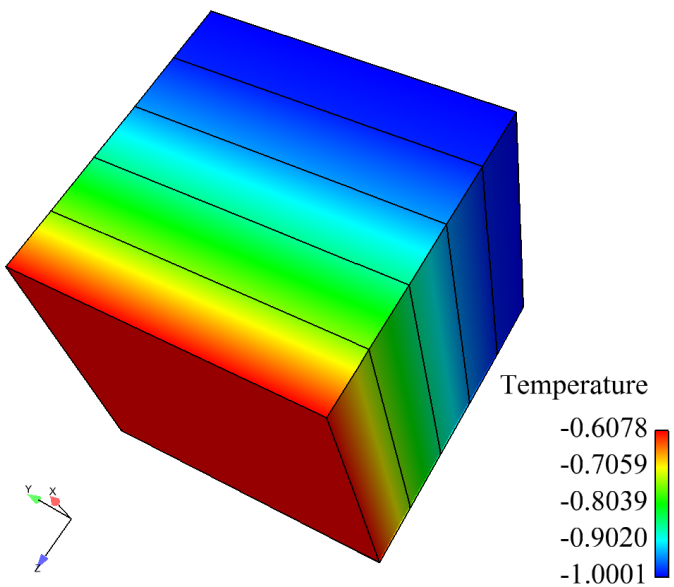



```
encorereg Postprocessor Output for Group agroup
Time  Evaluate: ffunc
----- ---------------
    0       -0.901515
0.005       -0.900399
 0.01       -0.899257
0.015       -0.898088
```

Evaluate Postprocessor

- In order to compare to experimental values, analysts need to probe simulation values.
- The **Evaluation PP** takes a Function and evaluates it at any point in space/time.
- Any type of evaluation (Value, Gradient, Time Derivative, Stress, Flux) can be performed on a function.
- In order to perform code verification, a user could evaluate the difference between the simulation and an analytic solution while the simulation is running.

Example: Evaluate the simulation field temperature at a point, $T(0, 0, 0)$




```
encorereg Postprocessor Output for Group agroup
Time  Evaluate: ffunc
----- ---------------
    0       -0.901515
0.005       -0.900399
 0.01       -0.899257
0.015       -0.898088
```

Verification Using Norms

- For code verification using order of convergence, accurate calculation of global norms is essential.
- Output of **Encore** Norms:
  - element-wise contributions for visualization or adaptivity,
  - global values in text file.
- Common global norms in **Encore**:

$$||u||_{L^2(\Omega)} \equiv \left(\int_\Omega |u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{H^1(\Omega)} \equiv \left(\int_\Omega |\nabla u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |\nabla u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{L^\infty(\Omega)} \equiv \max_{x \in \Omega} |u(x)| \approx \max_{elem} \underbrace{\max_q |u(x_q)|}_{\text{element contribution}}$$

Verification Using Norms

- For code verification using order of convergence, accurate calculation of global norms is essential.
- Output of **Encore** Norms:
  - element-wise contributions for visualization or adaptivity,
  - global values in text file.
- Common global norms in **Encore**:

$$||u||_{L^2(\Omega)} \equiv \left(\int_\Omega |u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{H^1(\Omega)} \equiv \left(\int_\Omega |\nabla u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |\nabla u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{L^\infty(\Omega)} \equiv \max_{x \in \Omega} |u(x)| \approx \max_{elem} \underbrace{\max_q |u(x_q)|}_{\text{element contribution}}$$

Verification Using Norms

- For code verification using order of convergence, accurate calculation of global norms is essential.
- Output of **Encore** Norms:
  - element-wise contributions for visualization or adaptivity,
  - global values in text file.
- Common global norms in **Encore**:

$$||u||_{L^2(\Omega)} \equiv \left(\int_\Omega |u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{H^1(\Omega)} \equiv \left(\int_\Omega |\nabla u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |\nabla u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{L^\infty(\Omega)} \equiv \max_{x \in \Omega} |u(x)| \approx \max_{elem} \underbrace{\max_q |u(x_q)|}_{\text{element contribution}}$$

Verification Using Norms

- For code verification using order of convergence, accurate calculation of global norms is essential.
- Output of **Encore** Norms:
  - element-wise contributions for visualization or adaptivity,
  - global values in text file.
- Common global norms in **Encore**:

$$||u||_{L^2(\Omega)} \equiv \left(\int_\Omega |u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{H^1(\Omega)} \equiv \left(\int_\Omega |\nabla u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |\nabla u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{L^\infty(\Omega)} \equiv \max_{x \in \Omega} |u(x)| \approx \max_{elem} \underbrace{\max_q |u(x_q)|}_{\text{element contribution}}$$

Verification Using Norms

- For code verification using order of convergence, accurate calculation of global norms is essential.
- Output of **Encore** Norms:
  - element-wise contributions for visualization or adaptivity,
  - global values in text file.
- Common global norms in **Encore**:

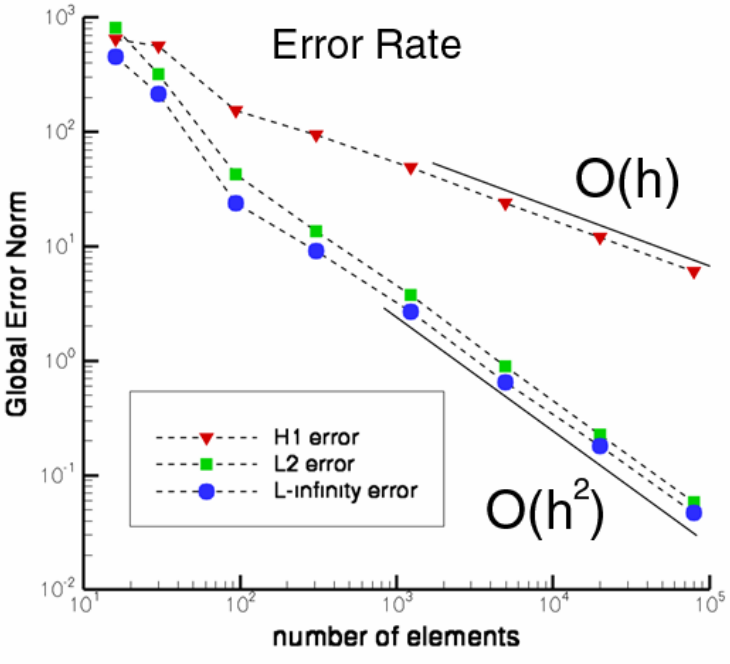
$$||u||_{L^2(\Omega)} \equiv \left(\int_\Omega |u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

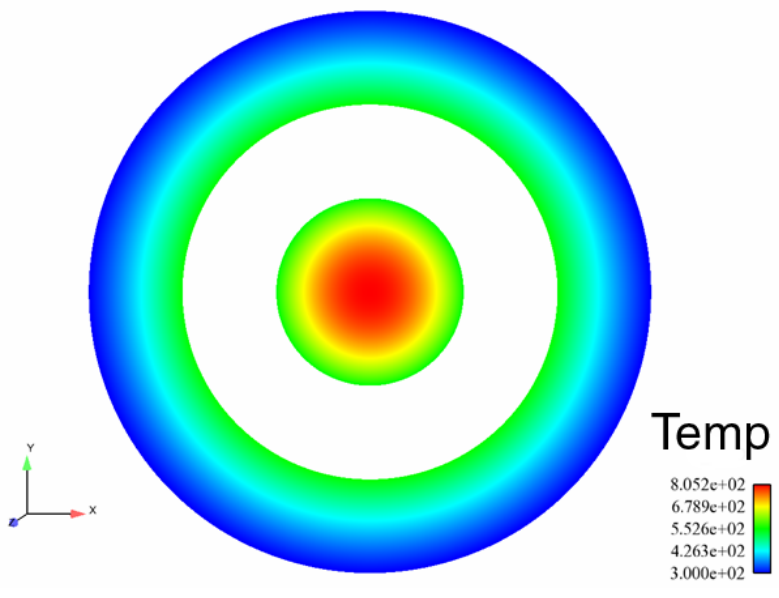
$$||u||_{H^1(\Omega)} \equiv \left(\int_\Omega |\nabla u|^2 \, dx \right)^{\frac{1}{2}} \approx \left(\sum_{elem} \underbrace{\sum_q |\nabla u(x_q)|^2 \, |JxW|_q}_{\text{element contribution}} \right)^{\frac{1}{2}}$$

$$||u||_{L^\infty(\Omega)} \equiv \max_{x \in \Omega} |u(x)| \approx \max_{elem} \underbrace{\max_q |u(x_q)|}_{\text{element contribution}}$$

Example: Calore Verification

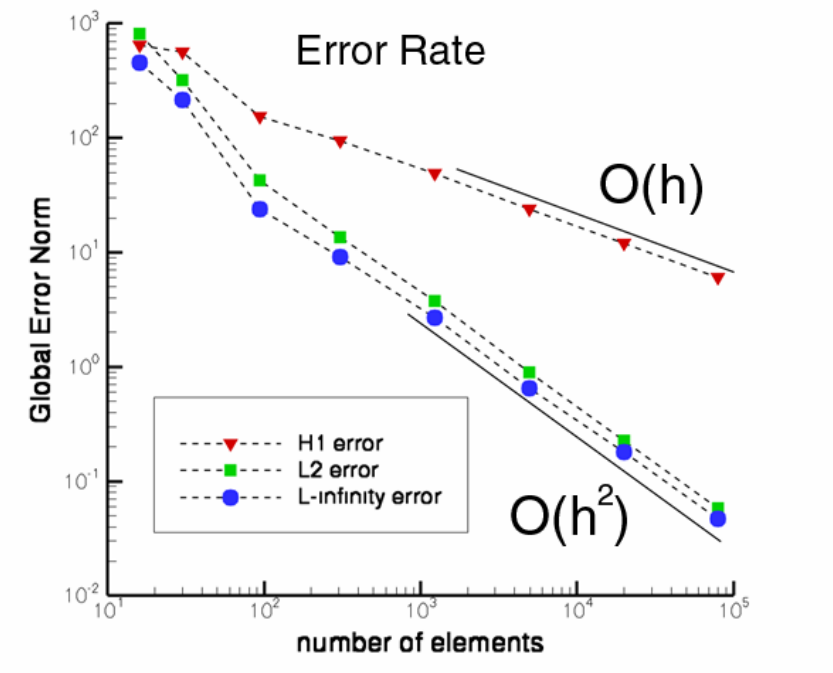
- These norms have been used in verification studies of **Calore** (FY07 Level 2 Tri-Lab Verification Milestone).
- Example: **Calore** coupled conduction/enclosure radiation problem.
- Using an exact analytic solution, the error in each global norm can be calculated, and the order of convergence verified.

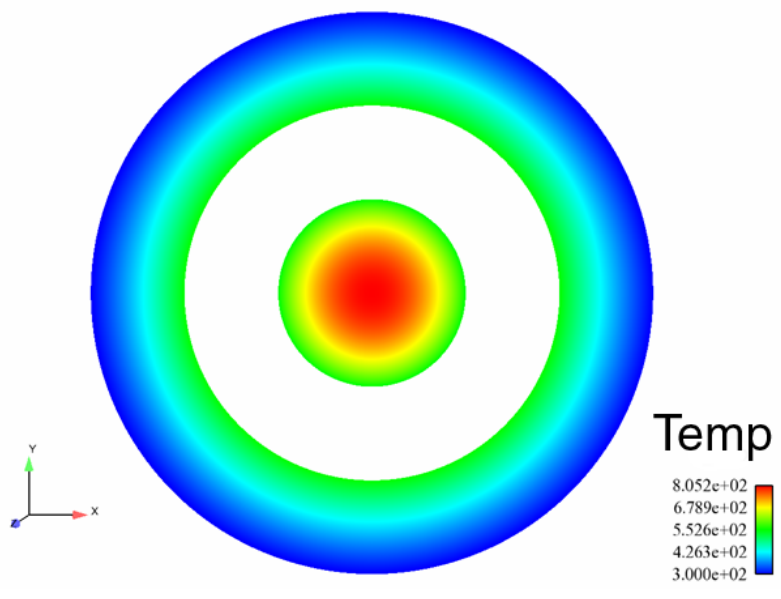




Example: Calore Verification

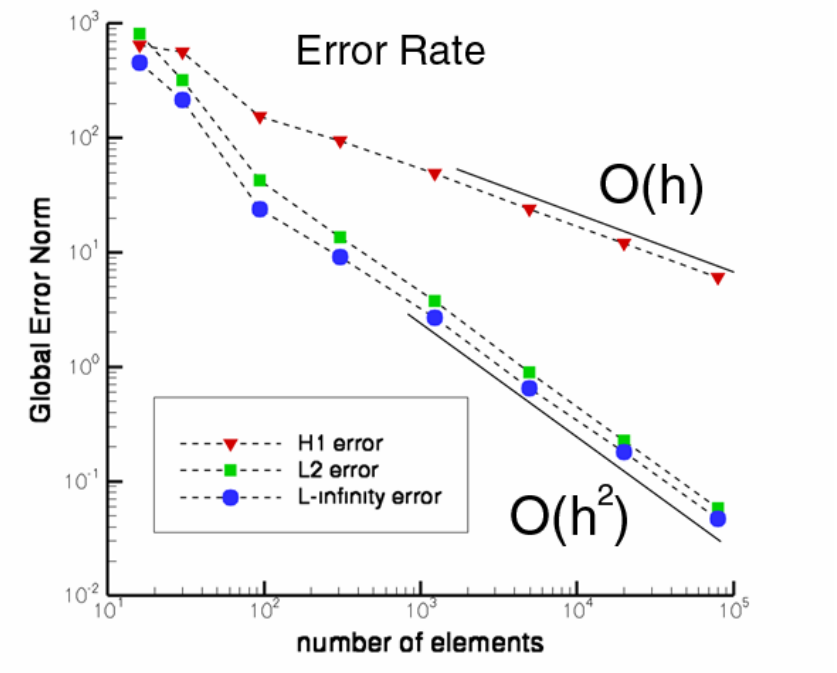
- These norms have been used in verification studies of **Calore** (FY07 Level 2 Tri-Lab Verification Milestone).
- Example: **Calore** coupled conduction/enclosure radiation problem.
- Using an exact analytic solution, the error in each global norm can be calculated, and the order of convergence verified.

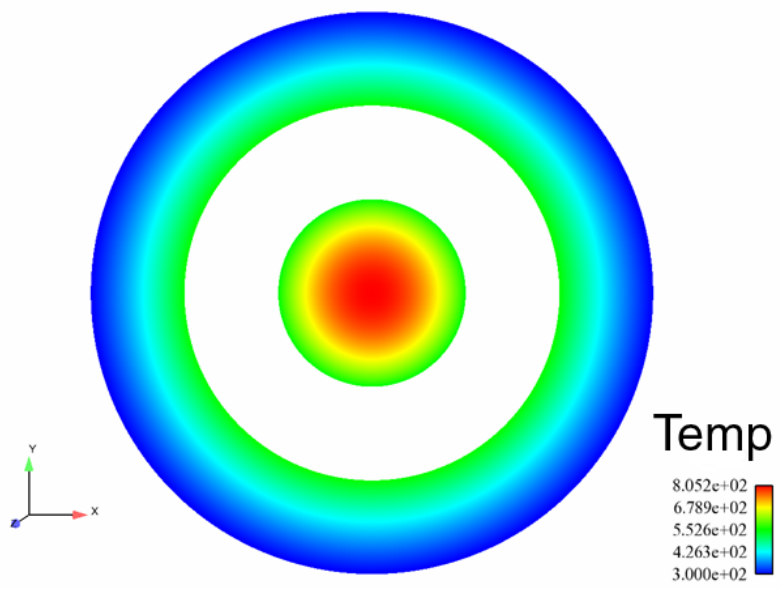




Example: Calore Verification

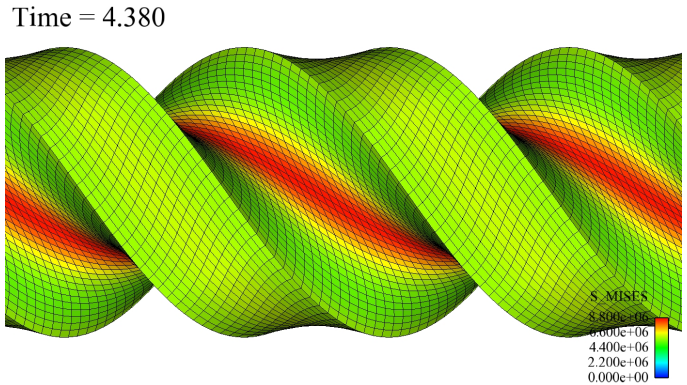
- These norms have been used in verification studies of **Calore** (FY07 Level 2 Tri-Lab Verification Milestone).
- Example: **Calore** coupled conduction/enclosure radiation problem.
- Using an exact analytic solution, the error in each global norm can be calculated, and the order of convergence verified.





Verification of Mechanical Response

- A major activity in FY08 is verification for the Sierra codes **Adagio** and **Presto** (with Joe Bishop and Pat Knupp).
- This will involve Encore tools for norms, transfers, manufactured solutions, and error estimators.



Example: **Presto** verification

Example: **Adagio** stress/displacement verification for linear elastic beam.

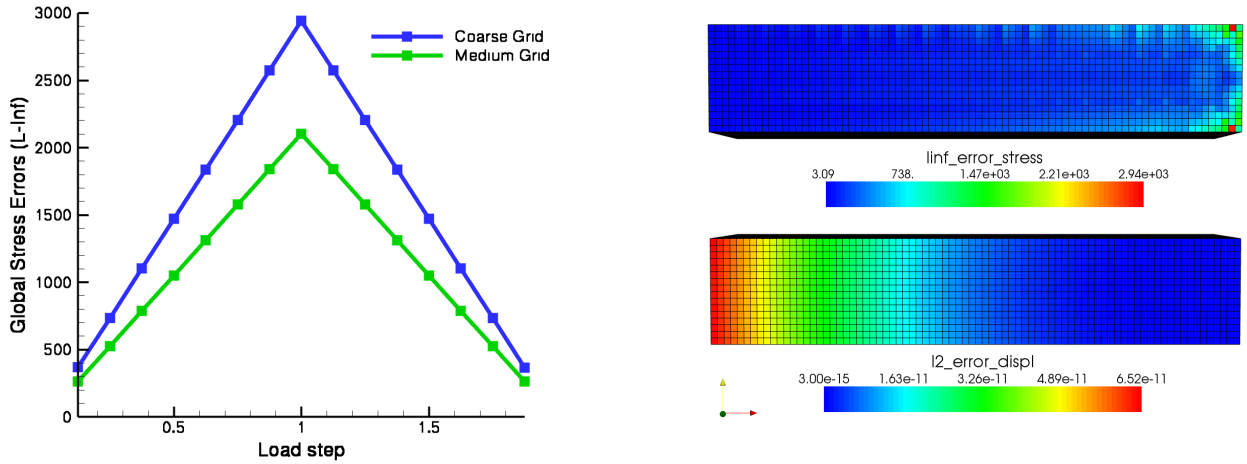
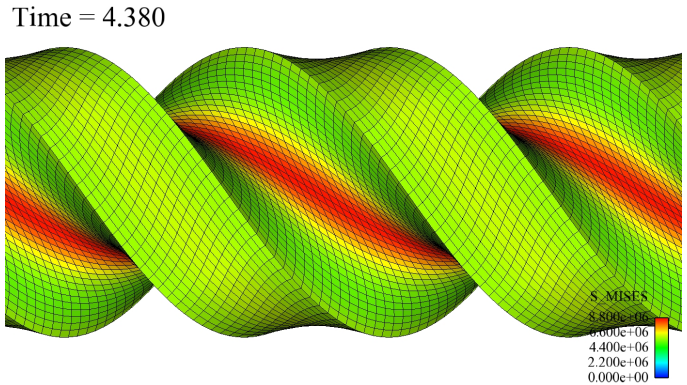


Figure: (left) Stress error rates (right) Stress/displacement error distribution

Verification of Mechanical Response

- A major activity in FY08 is verification for the Sierra codes **Adagio** and **Presto** (with Joe Bishop and Pat Knupp).
- This will involve Encore tools for norms, transfers, manufactured solutions, and error estimators.



Example: **Presto** verification

Example: **Adagio** stress/displacement verification for linear elastic beam.

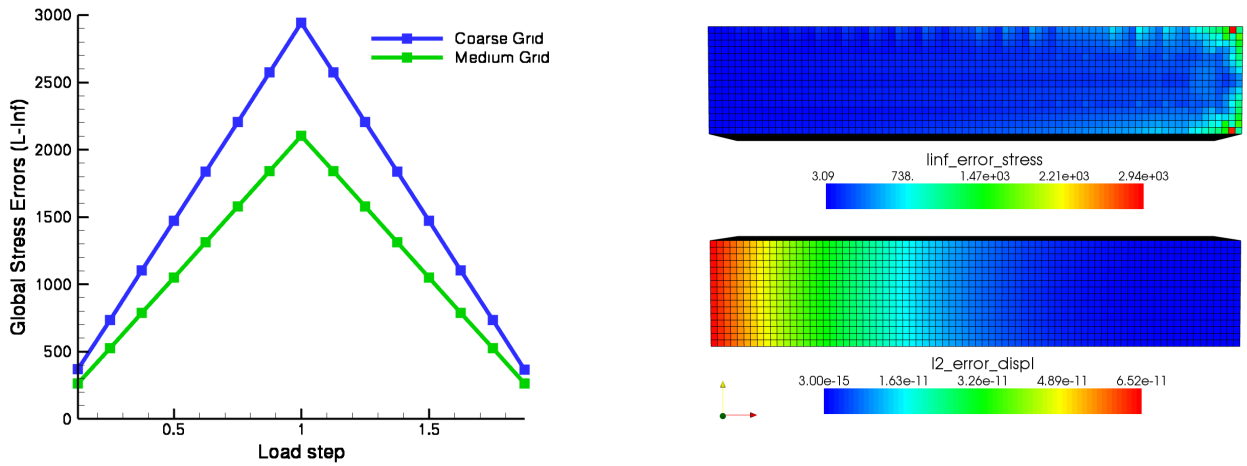


Figure: (left) Stress error rates (right) Stress/displacement error distribution

Integral Based PPs

- Many quantities that analysts calculate are spatial integrals of simulation fields.
- Accurate integrals can be computed with higher order quadrature.
- **Integral**: computes integral over a volume or sideset.
- **Average Value**: computes integral then divides by the volume/area.
- **Surface Normal**: computes integral of the Function dotted with the unit normal over a sideset.

Integral Based PPs

- Many quantities that analysts calculate are spatial integrals of simulation fields.
- Accurate integrals can be computed with higher order quadrature.
- **Integral**: computes integral over a volume or sideset.
- **Average Value**: computes integral then divides by the volume/area.
- **Surface Normal**: computes integral of the Function dotted with the unit normal over a sideset.

Integral Based PPs

- Many quantities that analysts calculate are spatial integrals of simulation fields.
- Accurate integrals can be computed with higher order quadrature.
- **Integral**: computes integral over a volume or sideset.
- **Average Value**: computes integral then divides by the volume/area.
- **Surface Normal**: computes integral of the Function dotted with the unit normal over a sideset.

Integral Based PPs

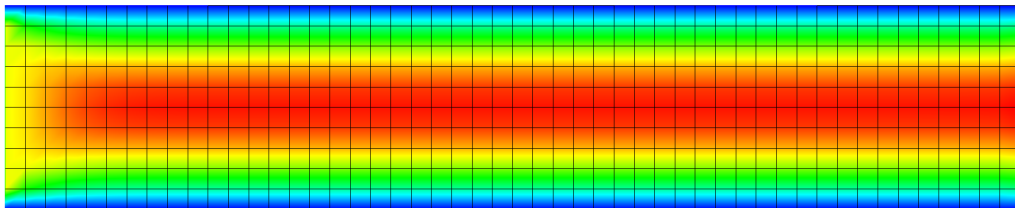
- Many quantities that analysts calculate are spatial integrals of simulation fields.
- Accurate integrals can be computed with higher order quadrature.
- **Integral**: computes integral over a volume or sideset.
- **Average Value**: computes integral then divides by the volume/area.
- **Surface Normal**: computes integral of the Function dotted with the unit normal over a sideset.

Integral Based PPs

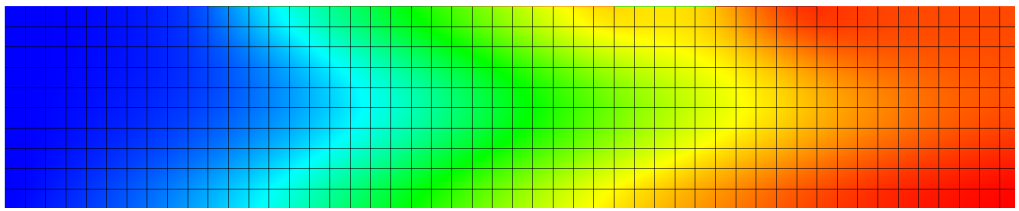
- Many quantities that analysts calculate are spatial integrals of simulation fields.
- Accurate integrals can be computed with higher order quadrature.
- **Integral**: computes integral over a volume or sideset.
- **Average Value**: computes integral then divides by the volume/area.
- **Surface Normal**: computes integral of the Function dotted with the unit normal over a sideset.

Impact: Integral PP

- **Charon** is an electrical and reacting flow simulation code.
- **Dakota** is an optimization toolkit.
- An analyst using Charon wanted to compute a mass flux exiting the domain and use Dakota to optimize that quantity.
- **Encore** calculates the mass flux correctly.
- This is an improvement over existing processing tools which give inaccurate values on curved geometries.
- The procedure: Charon $\rightarrow$ Exodus $\rightarrow$ Encore $\rightarrow$ Text File $\rightarrow$ Dakota.



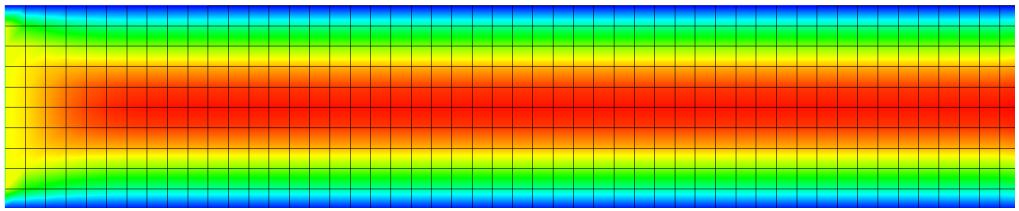
X Velocity Component.



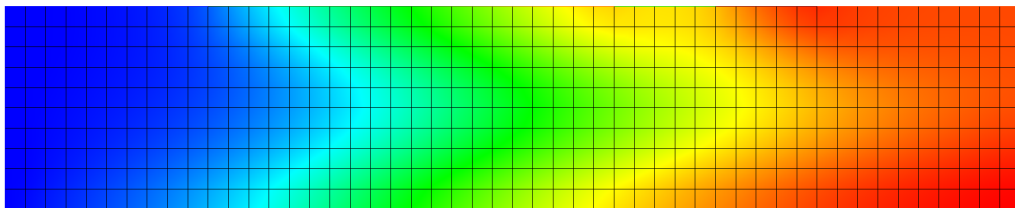
Oxygen Density.

Impact: Integral PP

- **Charon** is an electrical and reacting flow simulation code.
- **Dakota** is an optimization toolkit.
- An analyst using Charon wanted to compute a mass flux exiting the domain and use Dakota to optimize that quantity.
- **Encore** calculates the mass flux correctly.
- This is an improvement over existing processing tools which give inaccurate values on curved geometries.
- The procedure: Charon $\rightarrow$ Exodus $\rightarrow$ Encore $\rightarrow$ Text File $\rightarrow$ Dakota.



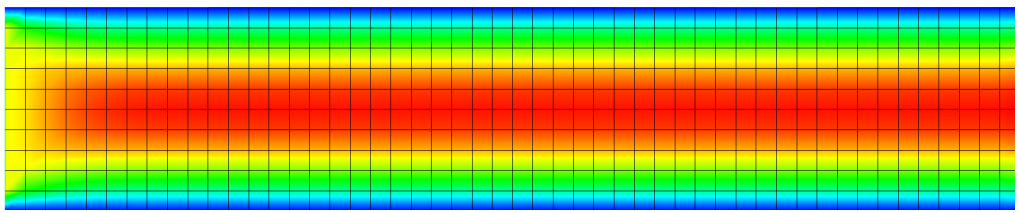
X Velocity Component.



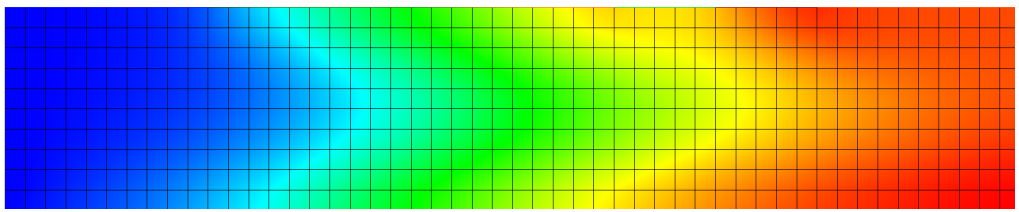
Oxygen Density.

Impact: Integral PP

- **Charon** is an electrical and reacting flow simulation code.
- **Dakota** is an optimization toolkit.
- An analyst using Charon wanted to compute a mass flux exiting the domain and use Dakota to optimize that quantity.
- **Encore** calculates the mass flux correctly.
- This is an improvement over existing processing tools which give inaccurate values on curved geometries.
- The procedure: Charon $\rightarrow$ Exodus $\rightarrow$ Encore $\rightarrow$ Text File $\rightarrow$ Dakota.



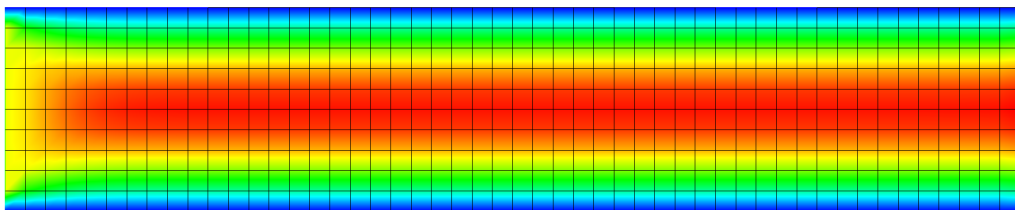
X Velocity Component.



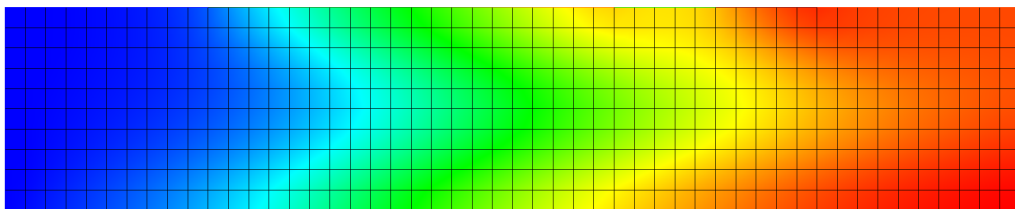
Oxygen Density.

Impact: Integral PP

- **Charon** is an electrical and reacting flow simulation code.
- **Dakota** is an optimization toolkit.
- An analyst using Charon wanted to compute a mass flux exiting the domain and use Dakota to optimize that quantity.
- **Encore** calculates the mass flux correctly.
- This is an improvement over existing processing tools which give inaccurate values on curved geometries.
- The procedure: Charon → Exodus → Encore → Text File → Dakota.



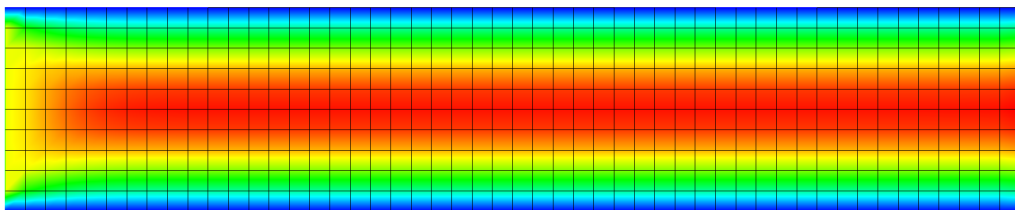
X Velocity Component.



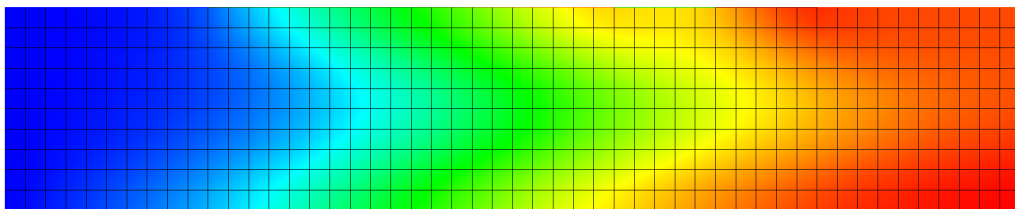
Oxygen Density.

Impact: Integral PP

- **Charon** is an electrical and reacting flow simulation code.
- **Dakota** is an optimization toolkit.
- An analyst using Charon wanted to compute a mass flux exiting the domain and use Dakota to optimize that quantity.
- **Encore** calculates the mass flux correctly.
- This is an improvement over existing processing tools which give inaccurate values on curved geometries.
- The procedure: Charon → Exodus → Encore → Text File → Dakota.



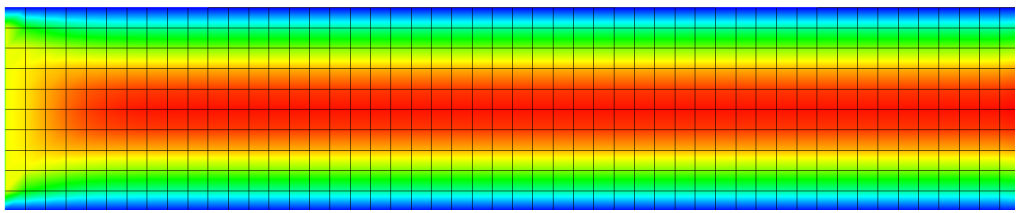
X Velocity Component.



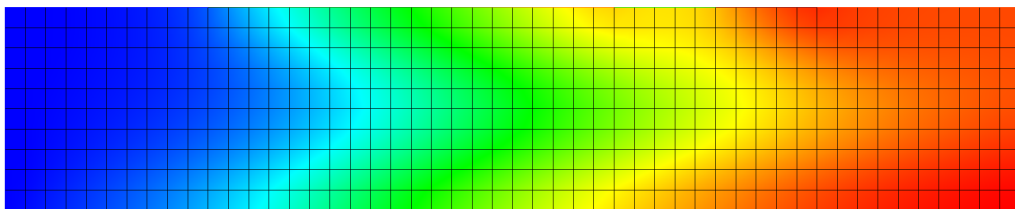
Oxygen Density.

Impact: Integral PP

- **Charon** is an electrical and reacting flow simulation code.
- **Dakota** is an optimization toolkit.
- An analyst using Charon wanted to compute a mass flux exiting the domain and use Dakota to optimize that quantity.
- **Encore** calculates the mass flux correctly.
- This is an improvement over existing processing tools which give inaccurate values on curved geometries.
- The procedure: Charon $\rightarrow$ Exodus $\rightarrow$ Encore $\rightarrow$ Text File $\rightarrow$ Dakota.



X Velocity Component.



Oxygen Density.

Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Postprocessors That Create Fields

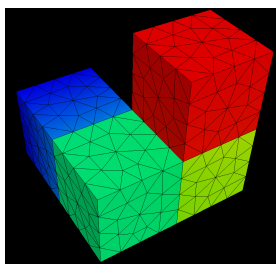
- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

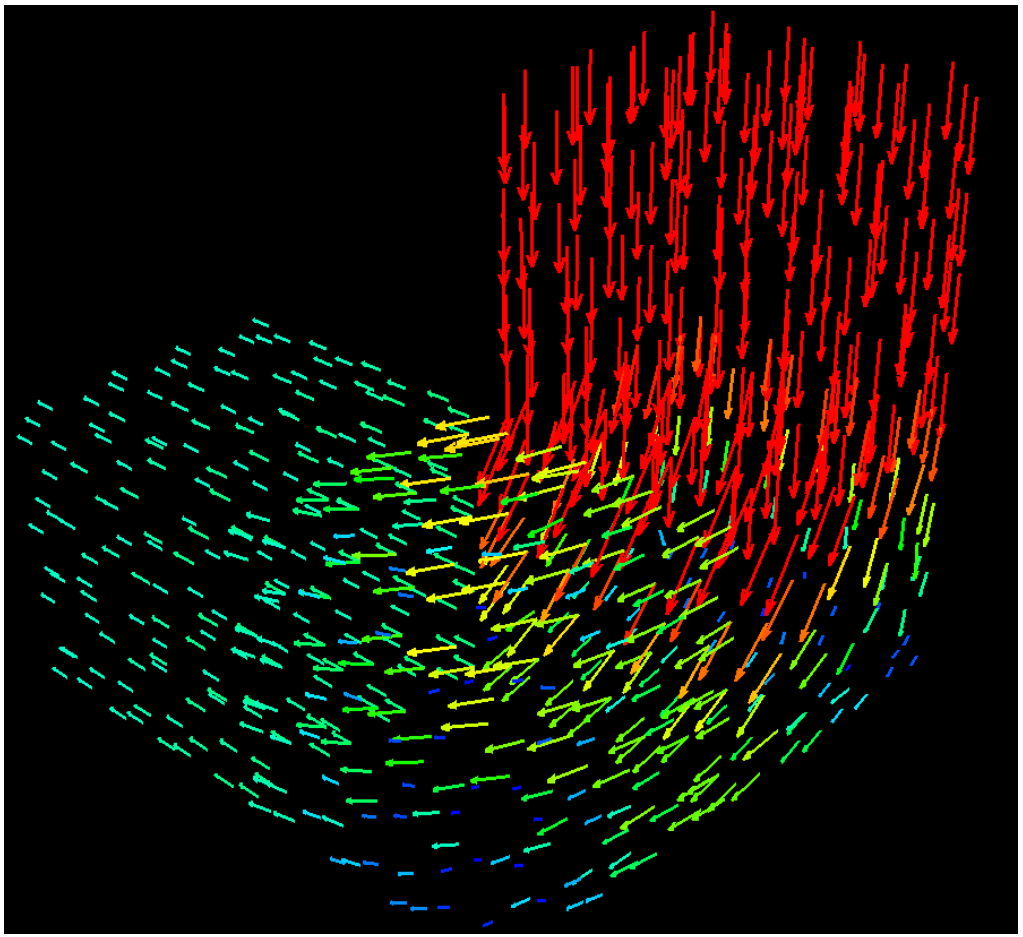
Postprocessors That Create Fields

- Often users need to calculate auxiliary simulation fields for analysis or to feedback into the simulation.
- **Import Field** reads a field from an Exodus file.
- **Create Field** allows for creation of any size and type of field.
- **Interpolate Function** takes a Function, interpolating it into a nodal or element field.
  - Allows visualization of an analytic Function.
  - Useful for viewing the difference between two Functions.
  - If the values of a Function are derived from a material property, then you can visualize an interpolation of that property into an element field.
- **Recover Function** performs nodal patch recovery on a field. Useful for higher order recovery or nodal recovery of an element field (such as stress).

Example: Interpolate Function PP

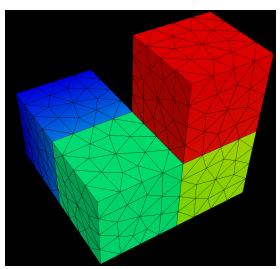
- **Calore** users wanted to compute the heat flux field.
- In online mode, **Encore** can access material properties for calculating heat flux.
- The Interpolate PP puts Function values into an element vector field.
- Example: transient **Calore** simulation with thermal contact between blocks.

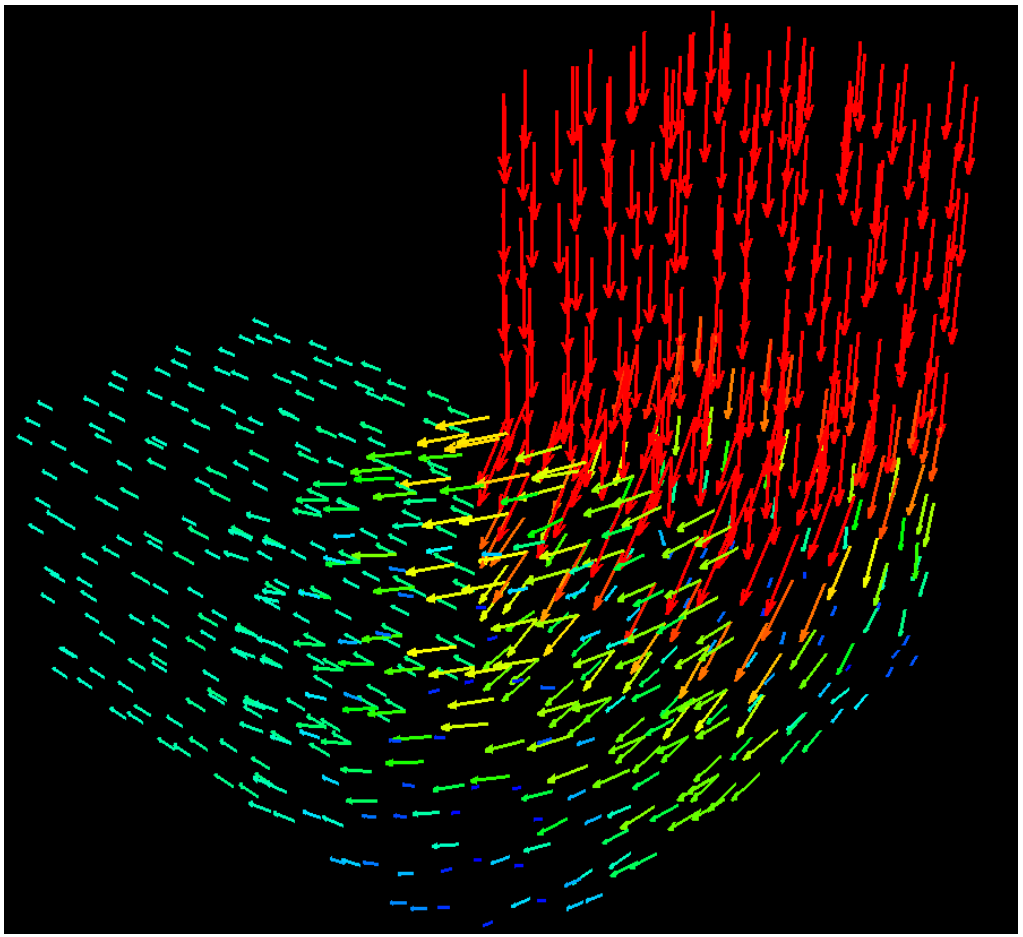




Example: Interpolate Function PP

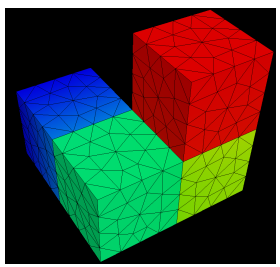
- **Calore** users wanted to compute the heat flux field.
- In online mode, **Encore** can access material properties for calculating heat flux.
- The Interpolate PP puts Function values into an element vector field.
- Example: transient **Calore** simulation with thermal contact between blocks.

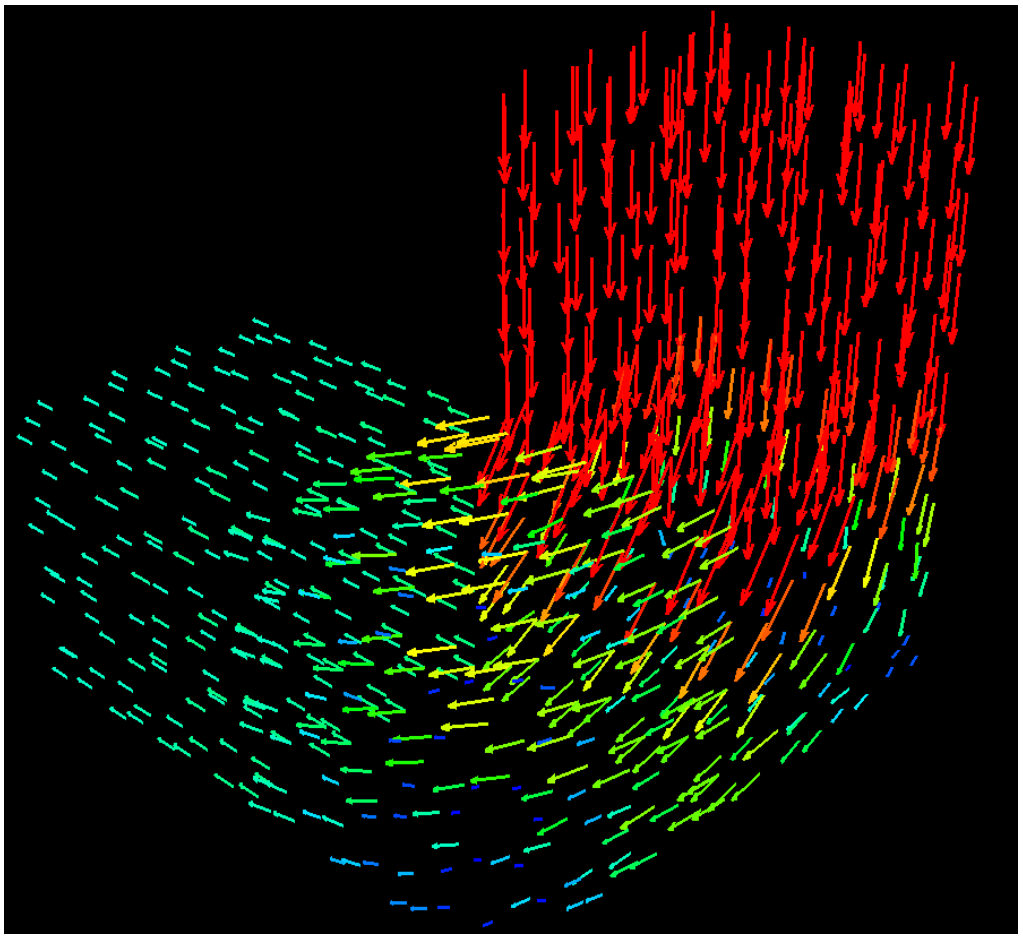




Example: Interpolate Function PP

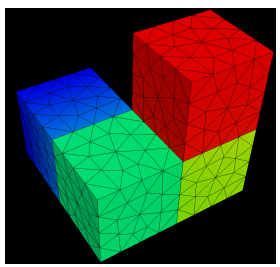
- **Calore** users wanted to compute the heat flux field.
- In online mode, **Encore** can access material properties for calculating heat flux.
- The Interpolate PP puts Function values into an element vector field.
- Example: transient **Calore** simulation with thermal contact between blocks.

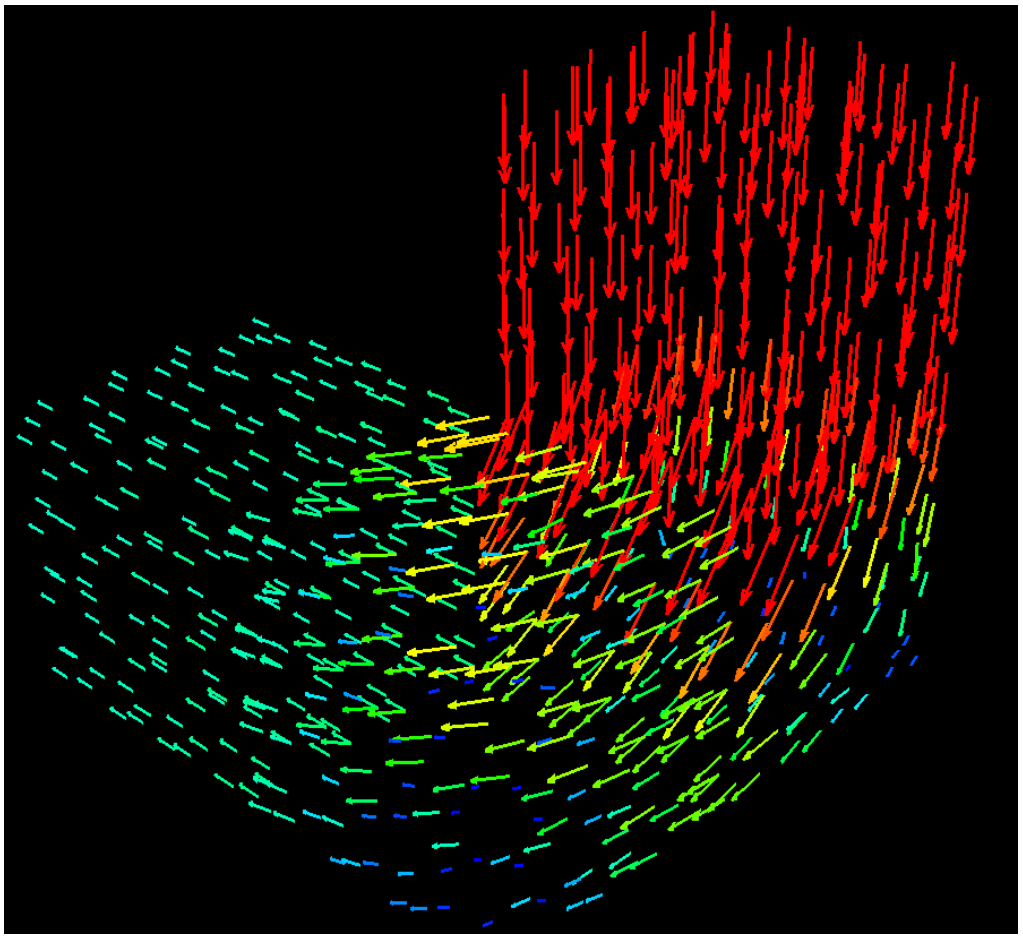




Example: Interpolate Function PP

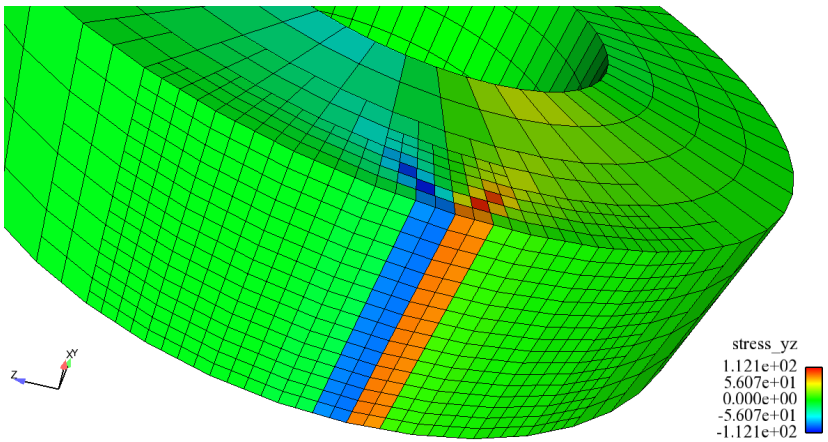
- **Calore** users wanted to compute the heat flux field.
- In online mode, **Encore** can access material properties for calculating heat flux.
- The Interpolate PP puts Function values into an element vector field.
- Example: transient **Calore** simulation with thermal contact between blocks.



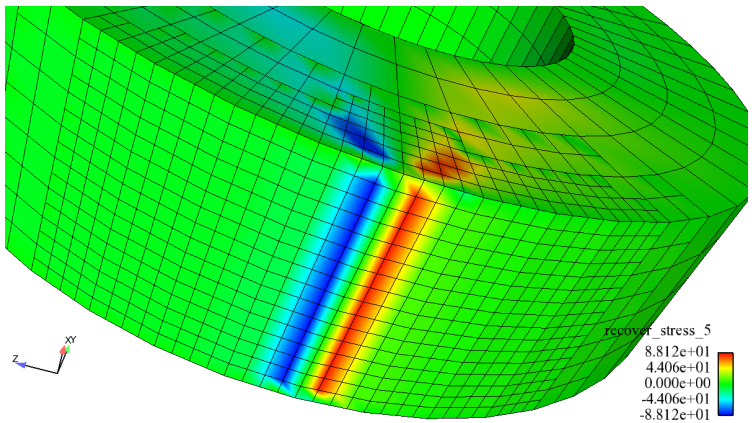


Example: Recover Function PP

- Stress recovery (averaging) is a common technique in solid mechanics.
- This can be used for postprocessing or to create an error estimator.
- The quasistatics code **Adagio** computes one stress per element, which is discontinuous and low order.
- Encore can recover this stress to create a continuous stress field that is also more accurate.



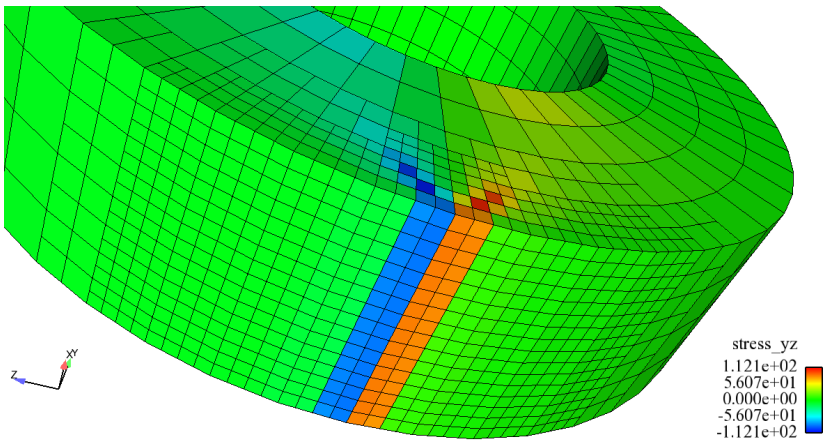
Element σ_{yz}



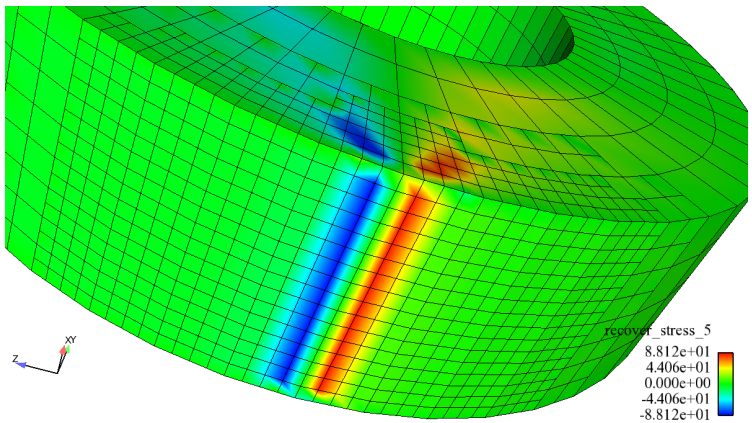
Recovered Nodal σ_{yz}

Example: Recover Function PP

- Stress recovery (averaging) is a common technique in solid mechanics.
- This can be used for postprocessing or to create an error estimator.
- The quasistatics code **Adagio** computes one stress per element, which is discontinuous and low order.
- Encore can recover this stress to create a continuous stress field that is also more accurate.



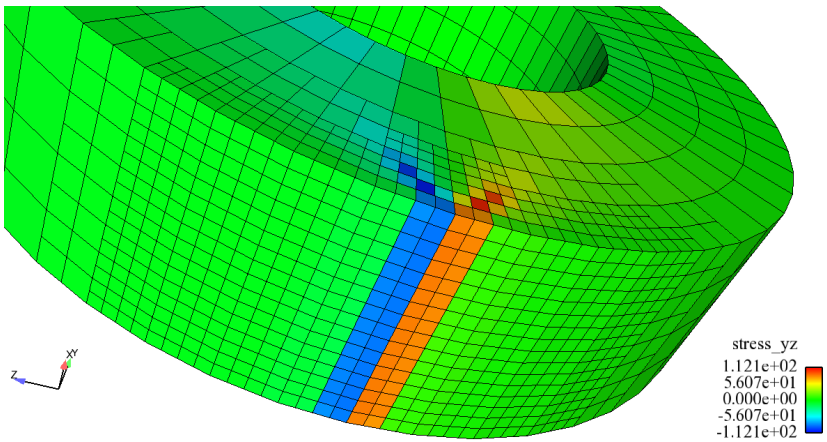
Element σ_{yz}



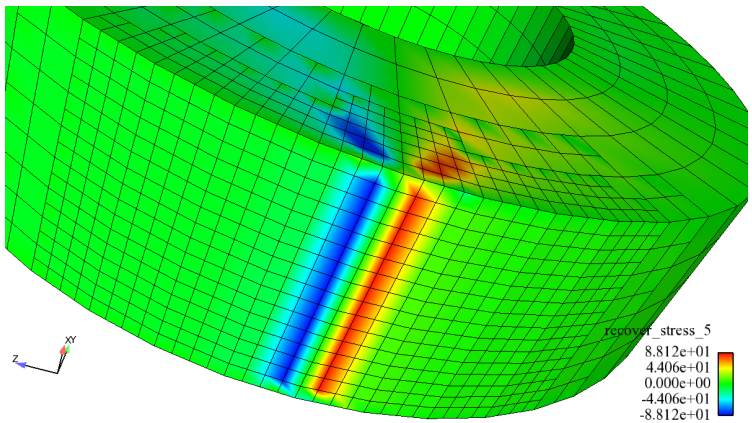
Recovered Nodal σ_{yz}

Example: Recover Function PP

- Stress recovery (averaging) is a common technique in solid mechanics.
- This can be used for postprocessing or to create an error estimator.
- The quasistatics code **Adagio** computes one stress per element, which is discontinuous and low order.
- Encore can recover this stress to create a continuous stress field that is also more accurate.



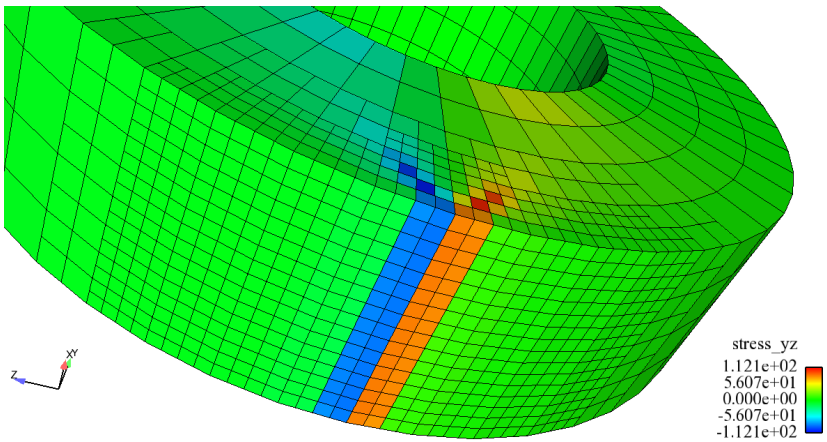
Element σ_{yz}



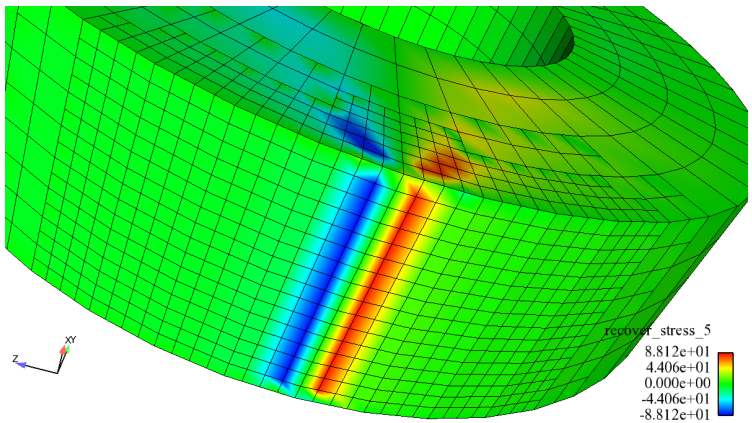
Recovered Nodal σ_{yz}

Example: Recover Function PP

- Stress recovery (averaging) is a common technique in solid mechanics.
- This can be used for postprocessing or to create an error estimator.
- The quasistatics code **Adagio** computes one stress per element, which is discontinuous and low order.
- Encore can recover this stress to create a continuous stress field that is also more accurate.



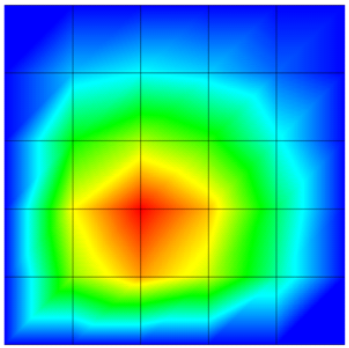
Element σ_{yz}



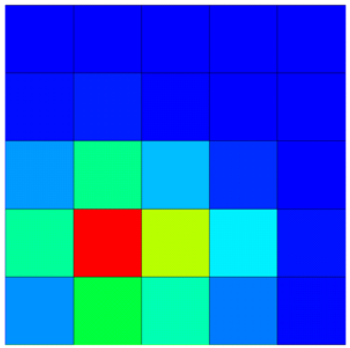
Recovered Nodal σ_{yz}

Transfers

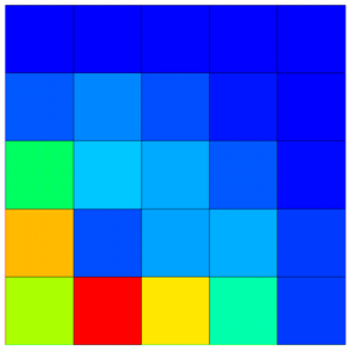
- Transfers project solutions from one mesh to another.
- This allows for solution comparison on different grids.
- 3 types of transfers: Nodal, Element, Quadrature.



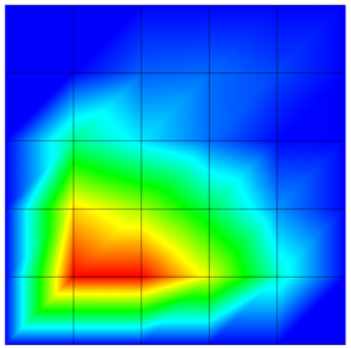
Coarse grid solution



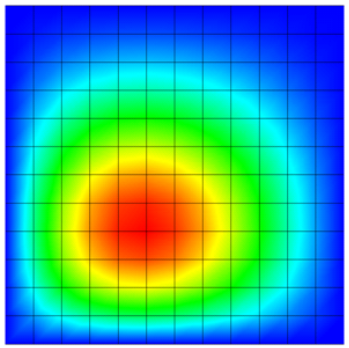
L2 error



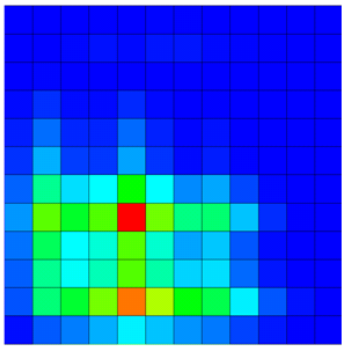
H1 error



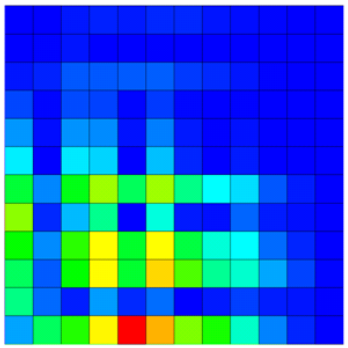
Nodal error



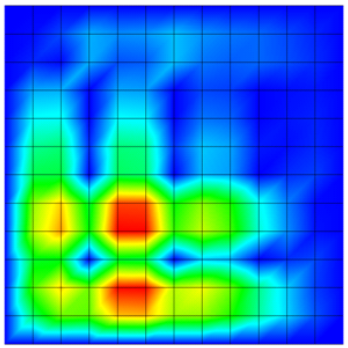
Fine grid solution



L2 error



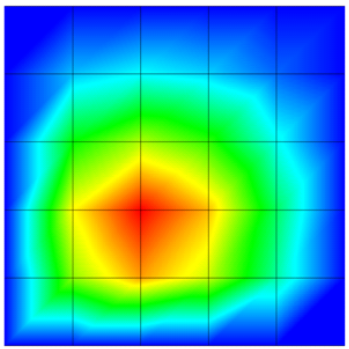
H1 error



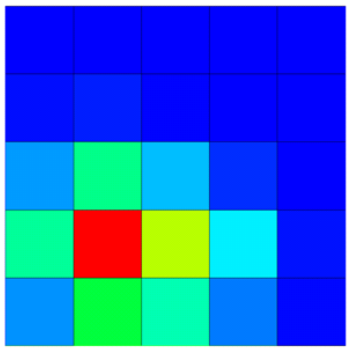
Nodal error

Transfers

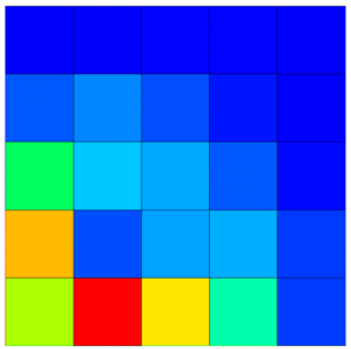
- Transfers project solutions from one mesh to another.
- This allows for solution comparison on different grids.
- 3 types of transfers: Nodal, Element, Quadrature.



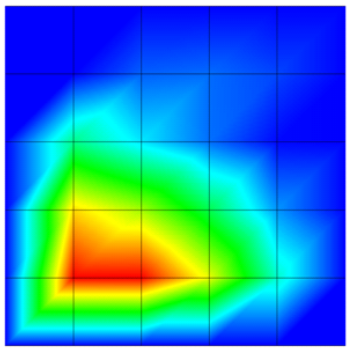
Coarse grid solution



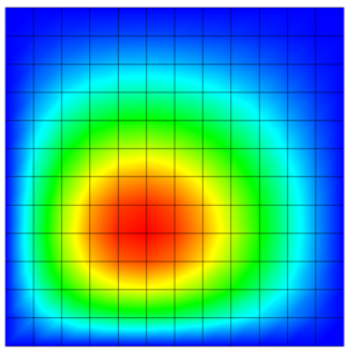
L2 error



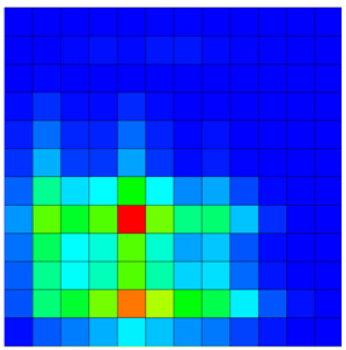
H1 error



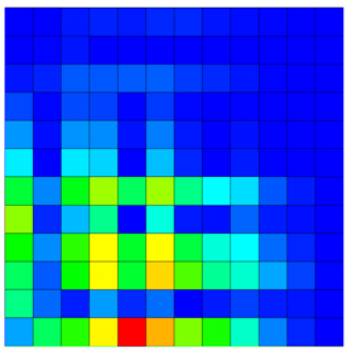
Nodal error



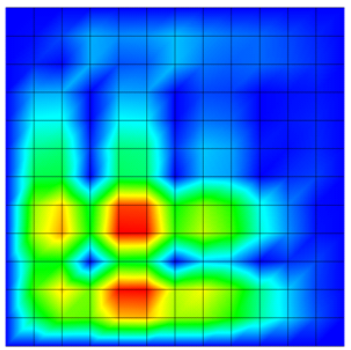
Fine grid solution



L2 error



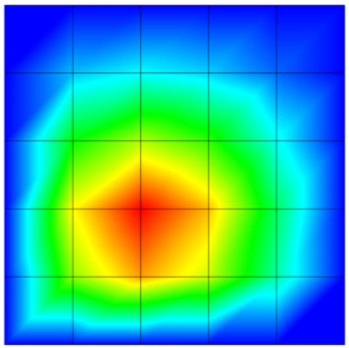
H1 error



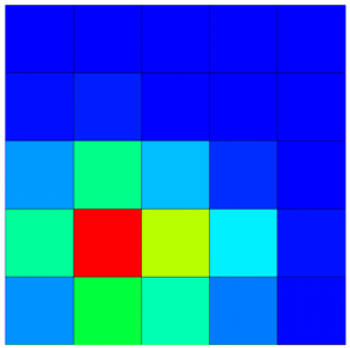
Nodal error

Transfers

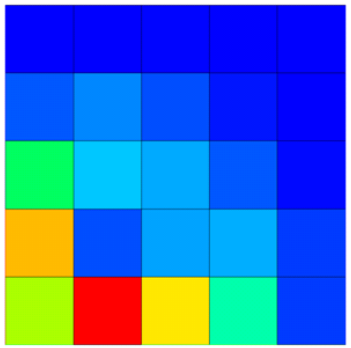
- Transfers project solutions from one mesh to another.
- This allows for solution comparison on different grids.
- 3 types of transfers: Nodal, Element, Quadrature.



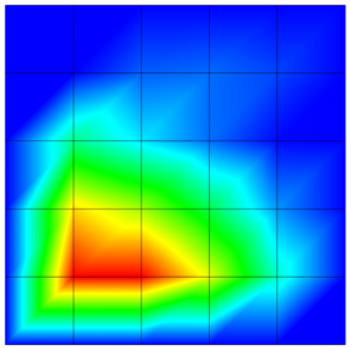
Coarse grid solution



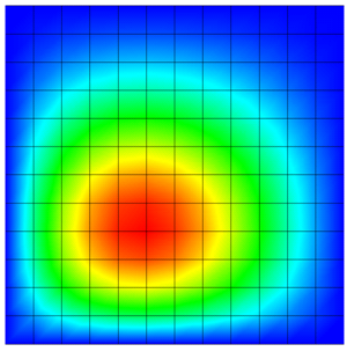
L2 error



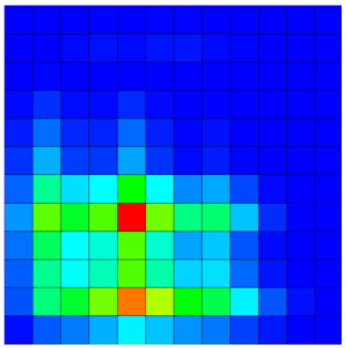
H1 error



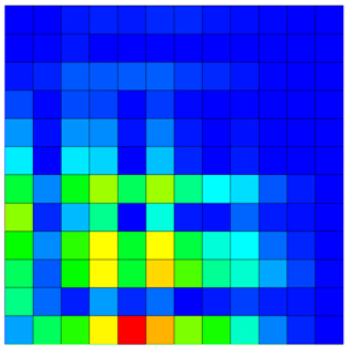
Nodal error



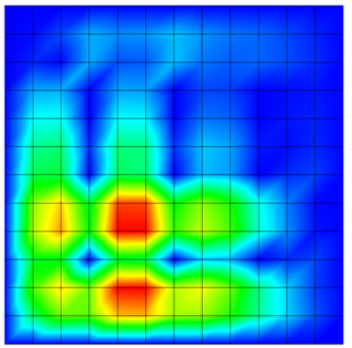
Fine grid solution



L2 error



H1 error



Nodal error

Impact: Transfers for Simulation Comparison

- A **Calore** customer noticed a discrepancy between runs done in **Coyote** and **Calore**.
- To investigate, a developer used **Encore** to compare the solutions.
- Encore can handle comparing solutions with differing timesteps.
- Being able to visualize the differences led to finding a bug in the way triangles were being handled in the Calore input file.

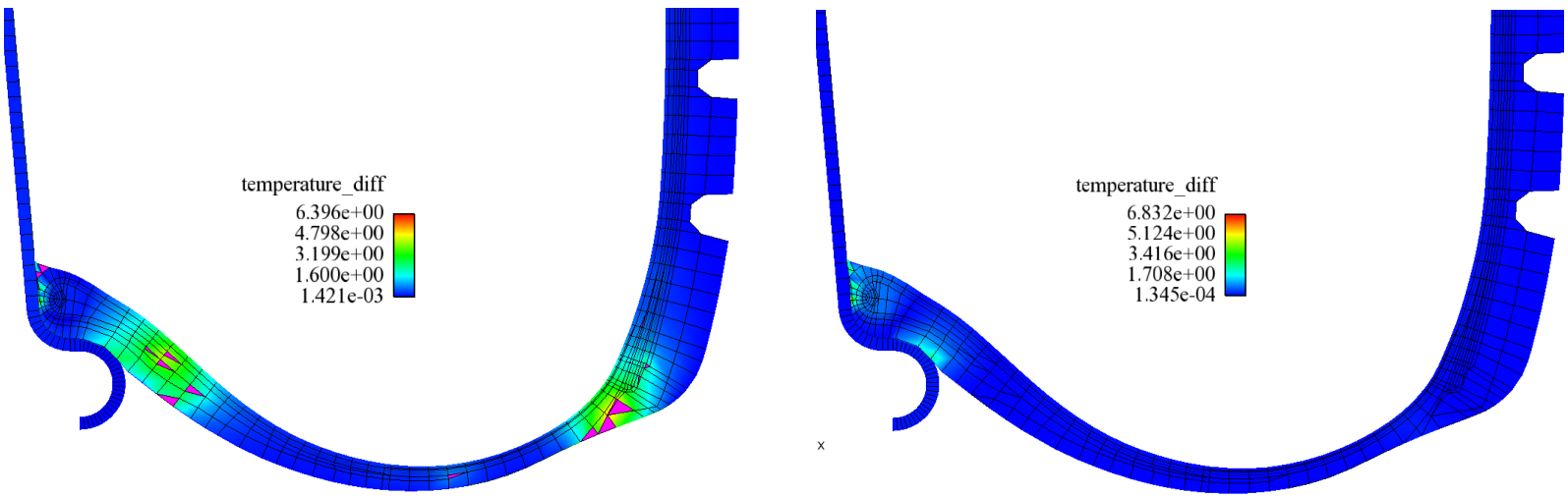


Figure: Temperature difference before and after bug fix.

Impact: Transfers for Simulation Comparison

- A **Calore** customer noticed a discrepancy between runs done in **Coyote** and **Calore**.
- To investigate, a developer used **Encore** to compare the solutions.
- Encore can handle comparing solutions with differing timesteps.
- Being able to visualize the differences led to finding a bug in the way triangles were being handled in the Calore input file.

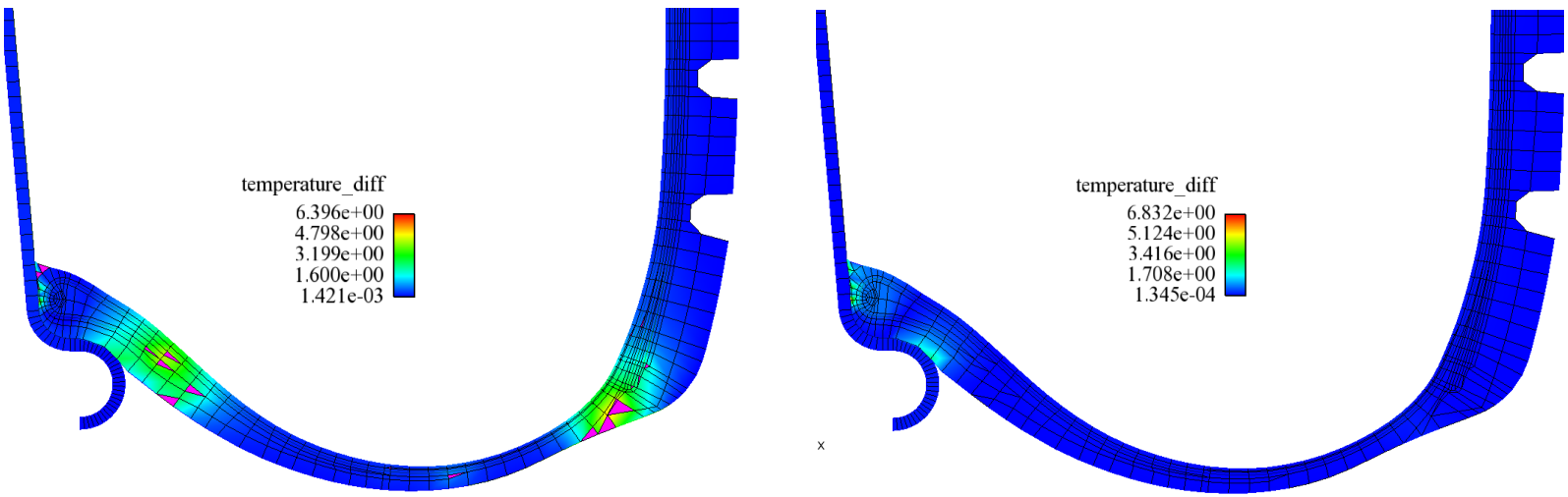


Figure: Temperature difference before and after bug fix.

Impact: Transfers for Simulation Comparison

- A **Calore** customer noticed a discrepancy between runs done in **Coyote** and **Calore**.
- To investigate, a developer used **Encore** to compare the solutions.
- Encore can handle comparing solutions with differing timesteps.
- Being able to visualize the differences led to finding a bug in the way triangles were being handled in the Calore input file.

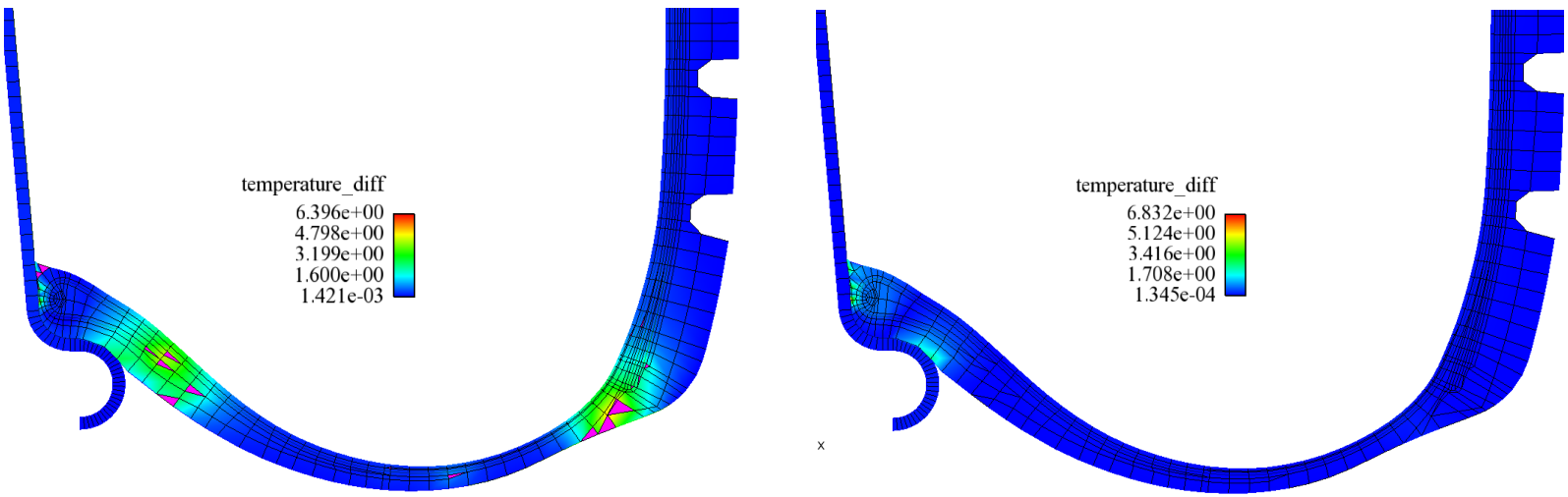


Figure: Temperature difference before and after bug fix.

Impact: Transfers for Simulation Comparison

- A **Calore** customer noticed a discrepancy between runs done in **Coyote** and **Calore**.
- To investigate, a developer used **Encore** to compare the solutions.
- Encore can handle comparing solutions with differing timesteps.
- Being able to visualize the differences led to finding a bug in the way triangles were being handled in the Calore input file.

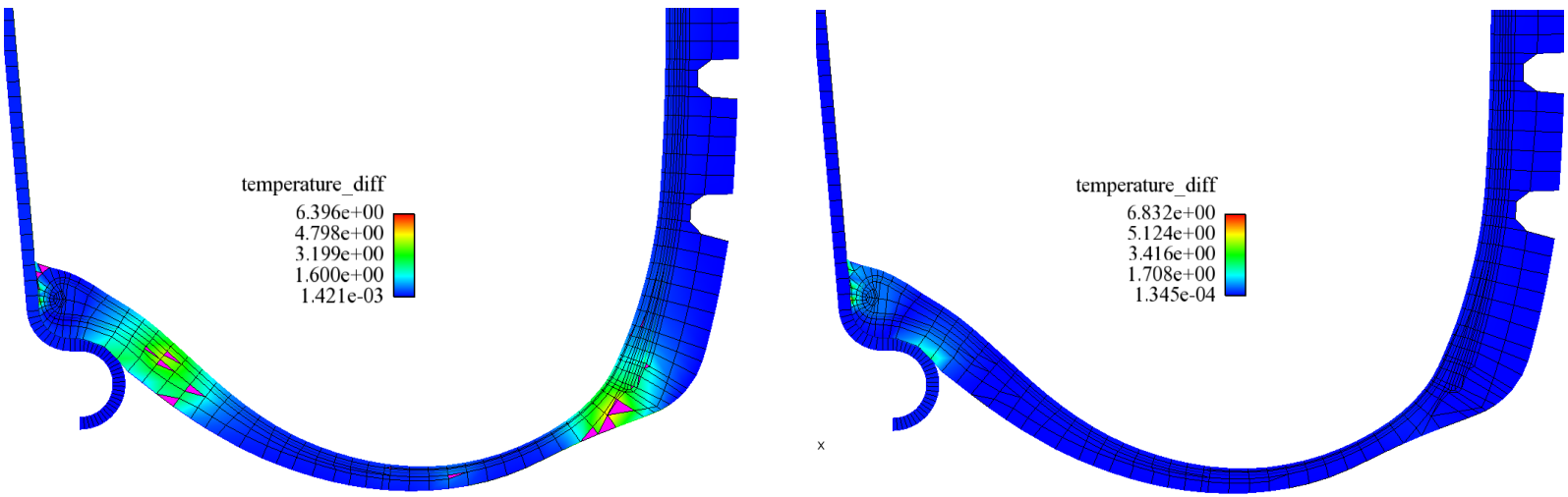
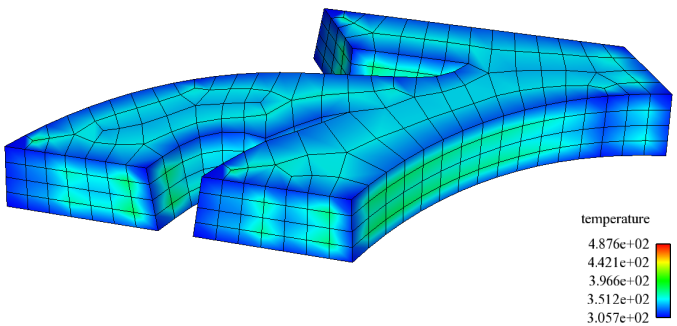


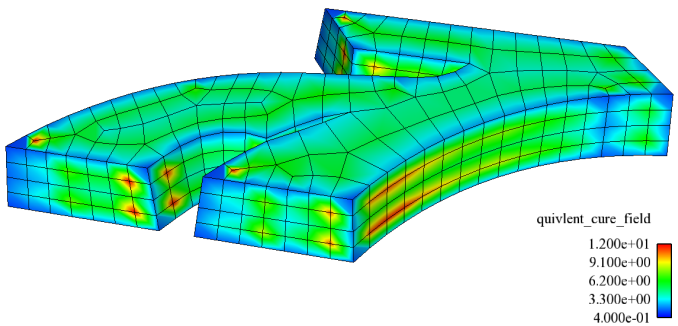
Figure: Temperature difference before and after bug fix.

Impact: User Postprocessor

- A User PP can be created allowing for complete flexibility to calculate whatever is desired.
- Writing a User PP is similar to a User Function.
- An external customer has used this capability to calculate a time integrated nodal quantity in both *online* and *offline* modes.



(a) Temperature.



(b) Time Integrated PP Value.

part 2

Error Estimation and Adaptivity

Overview of Error Estimation and Adaptivity

- The Adaptivity Cycle
- Physics Independent Error Indicators
- Adjoint Based Error Estimators
- Markers
- User-Driven Control of Adaptivity

Overview of Error Estimation and Adaptivity

- The Adaptivity Cycle
- Physics Independent Error Indicators
- Adjoint Based Error Estimators
- Markers
- User-Driven Control of Adaptivity

Overview of Error Estimation and Adaptivity

- The Adaptivity Cycle
- Physics Independent Error Indicators
- Adjoint Based Error Estimators
- Markers
- User-Driven Control of Adaptivity

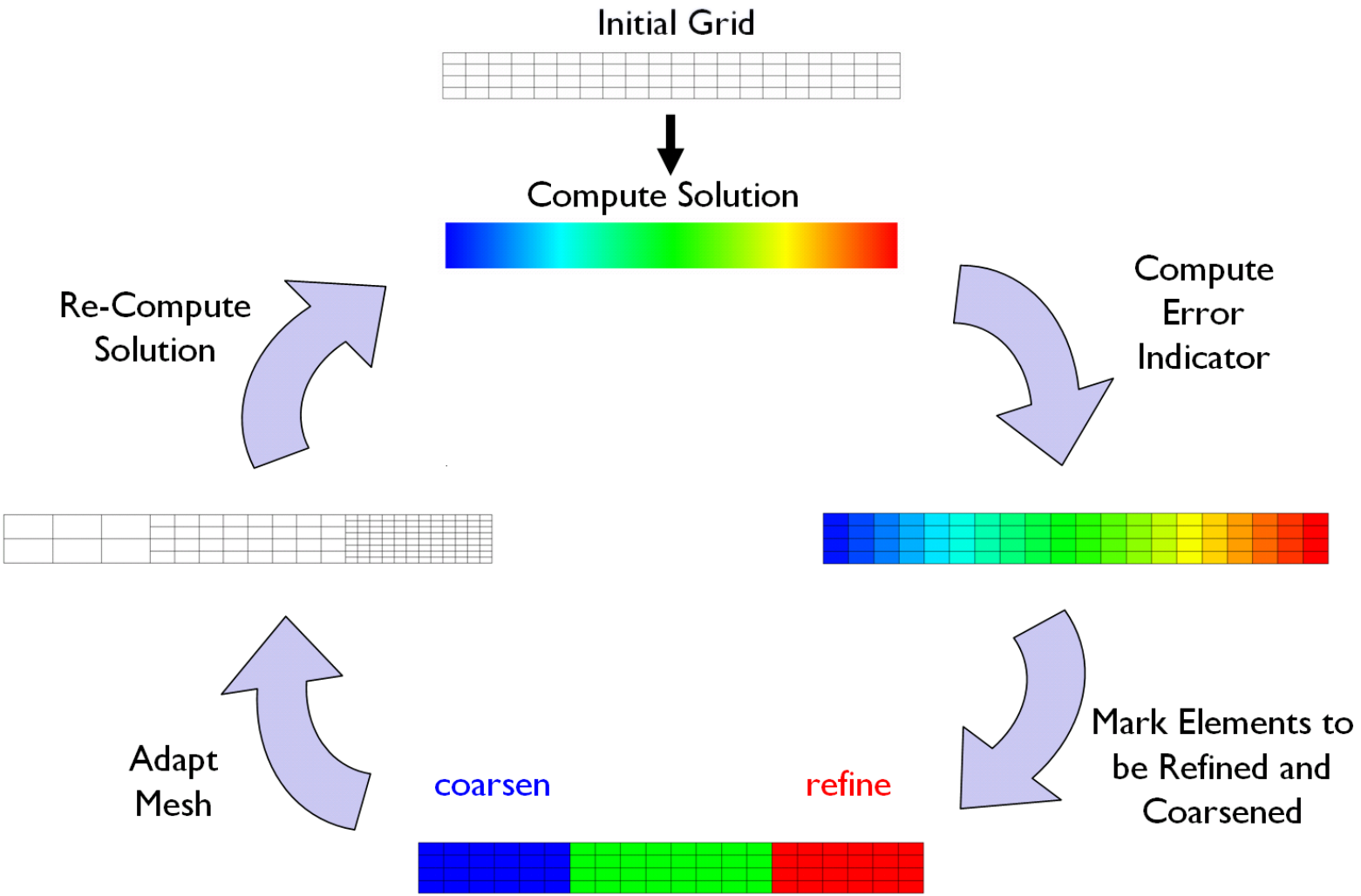
Overview of Error Estimation and Adaptivity

- The Adaptivity Cycle
- Physics Independent Error Indicators
- Adjoint Based Error Estimators
- Markers
- User-Driven Control of Adaptivity

Overview of Error Estimation and Adaptivity

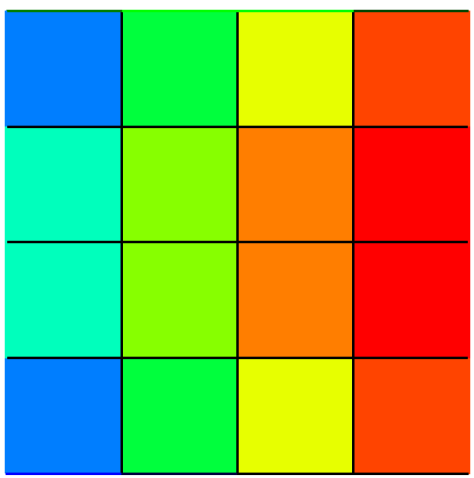
- The Adaptivity Cycle
- Physics Independent Error Indicators
- Adjoint Based Error Estimators
- Markers
- User-Driven Control of Adaptivity

The Adaptivity Cycle

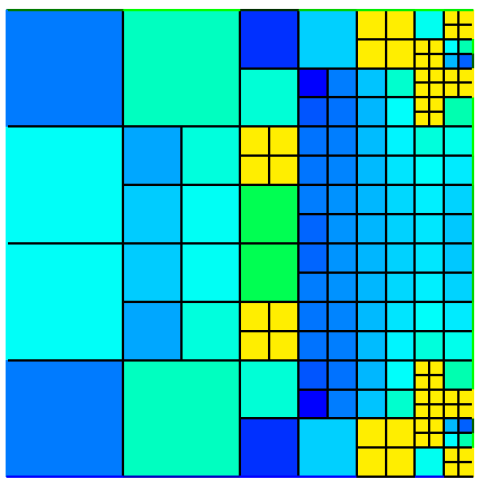


Physics Independent Error Indicators

- Error Indicators have two purposes:
  - Estimate the numerical error in the simulation on a given grid.
  - Provide input to mesh adaptivity, in the form of a spatial error distribution
- Encore implements several physics independent error indicators, allowing re-use in multiple application codes.
- Physics independent indicators operate solely on computed fields without knowledge of engineering/physics equations.



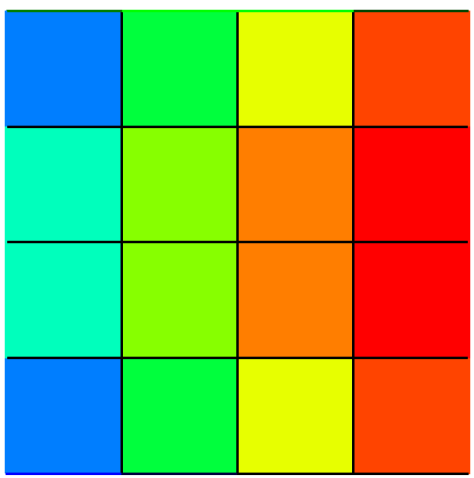
Error = 133%



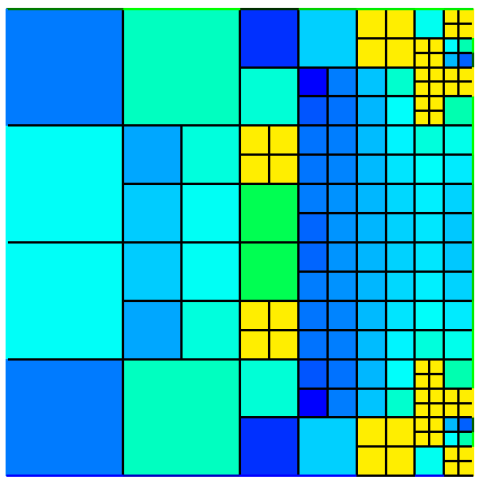
Error = 46%

Physics Independent Error Indicators

- Error Indicators have two purposes:
  - Estimate the numerical error in the simulation on a given grid.
  - Provide input to mesh adaptivity, in the form of a spatial error distribution
- Encore implements several physics independent error indicators, allowing re-use in multiple application codes.
- Physics independent indicators operate solely on computed fields without knowledge of engineering/physics equations.



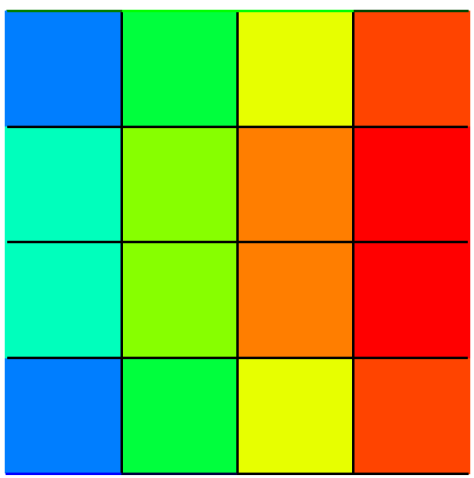
Error = 133%



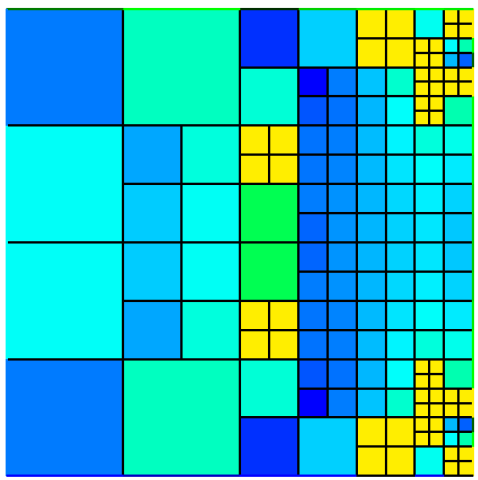
Error = 46%

Physics Independent Error Indicators

- Error Indicators have two purposes:
  - Estimate the numerical error in the simulation on a given grid.
  - Provide input to mesh adaptivity, in the form of a spatial error distribution
- Encore implements several physics independent error indicators, allowing re-use in multiple application codes.
- Physics independent indicators operate solely on computed fields without knowledge of engineering/physics equations.



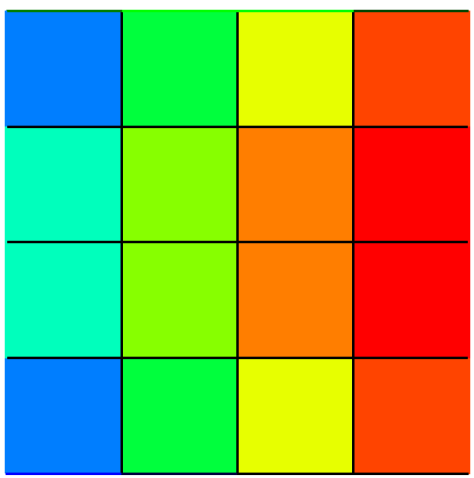
Error = 133%



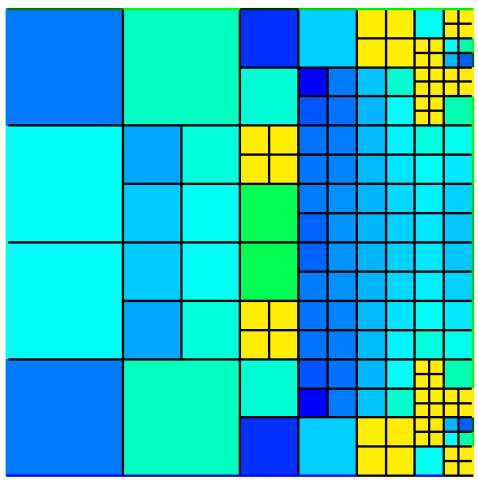
Error = 46%

Physics Independent Error Indicators

- Error Indicators have two purposes:
  - Estimate the numerical error in the simulation on a given grid.
  - Provide input to mesh adaptivity, in the form of a spatial error distribution
- Encore implements several physics independent error indicators, allowing re-use in multiple application codes.
- Physics independent indicators operate solely on computed fields without knowledge of engineering/physics equations.



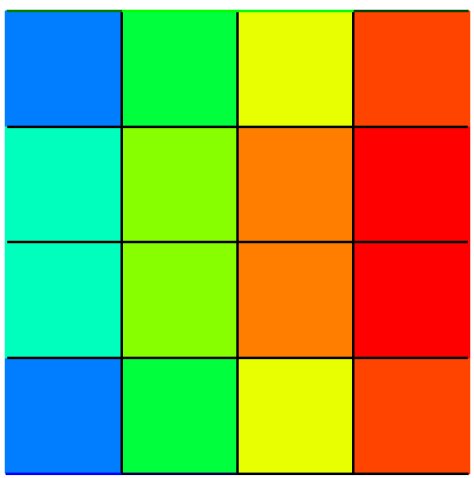
Error = 133%



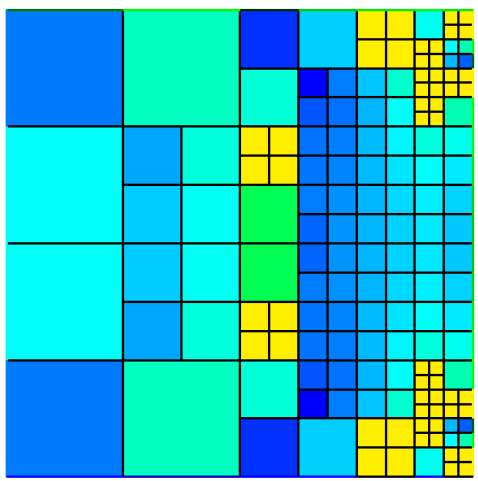
Error = 46%

Physics Independent Error Indicators

- Error Indicators have two purposes:
  - Estimate the numerical error in the simulation on a given grid.
  - Provide input to mesh adaptivity, in the form of a spatial error distribution
- Encore implements several physics independent error indicators, allowing re-use in multiple application codes.
- Physics independent indicators operate solely on computed fields without knowledge of engineering/physics equations.



Error = 133%



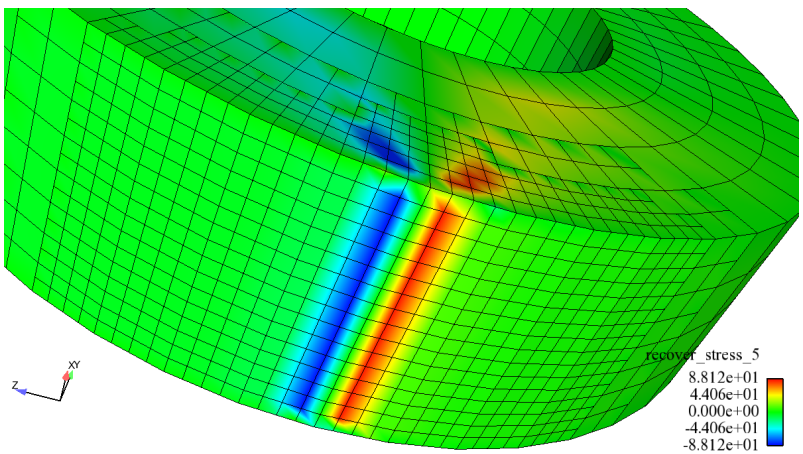
Error = 46%

Recovery Indicator

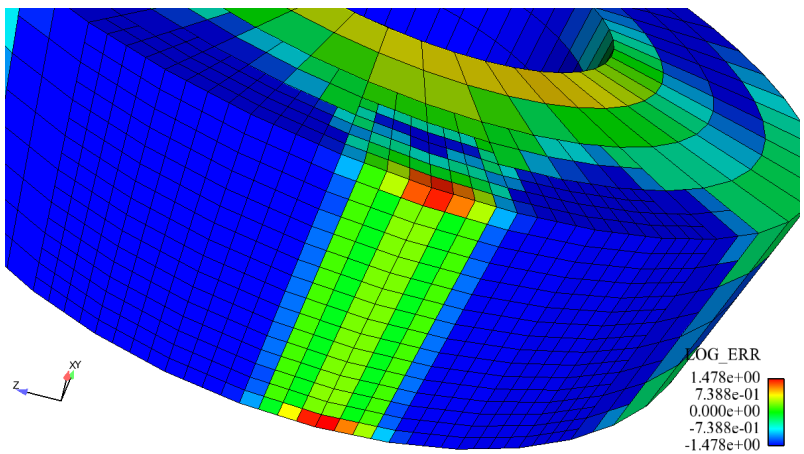
- Recovery Indicator computes difference between flux/stress and recovered (averaged) flux/stress (ZZ estimator).

$$\|\sigma - \sigma_h\| \approx \|\sigma^*(\sigma_h) - \sigma_h\|$$

- Example: simplistic rolling tire result from **Adagio** using feature based refinement in a box.
- Figures are of tire surface that is in contact with the road.



Recovered σ_{yz}



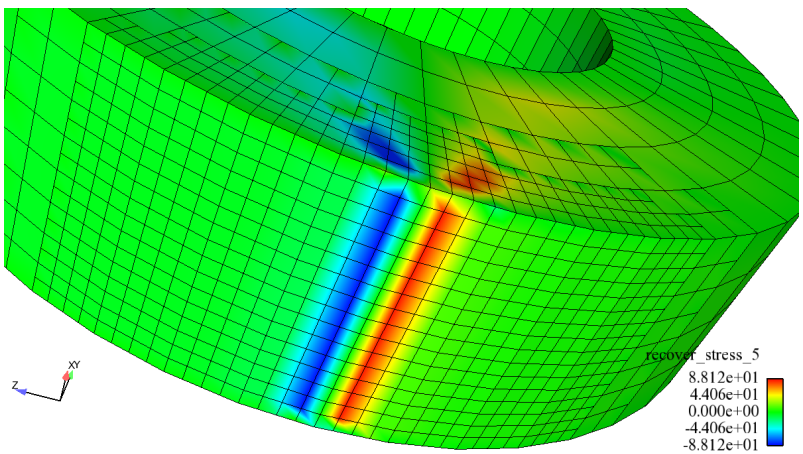
Log of Indicator

Recovery Indicator

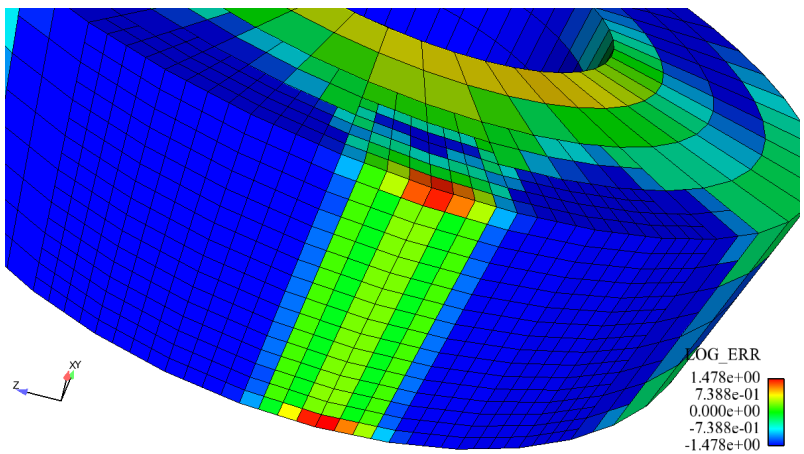
- Recovery Indicator computes difference between flux/stress and recovered (averaged) flux/stress (ZZ estimator).

$$\|\sigma - \sigma_h\| \approx \|\sigma^*(\sigma_h) - \sigma_h\|$$

- Example: simplistic rolling tire result from **Adagio** using feature based refinement in a box.
- Figures are of tire surface that is in contact with the road.



Recovered σ_{yz}



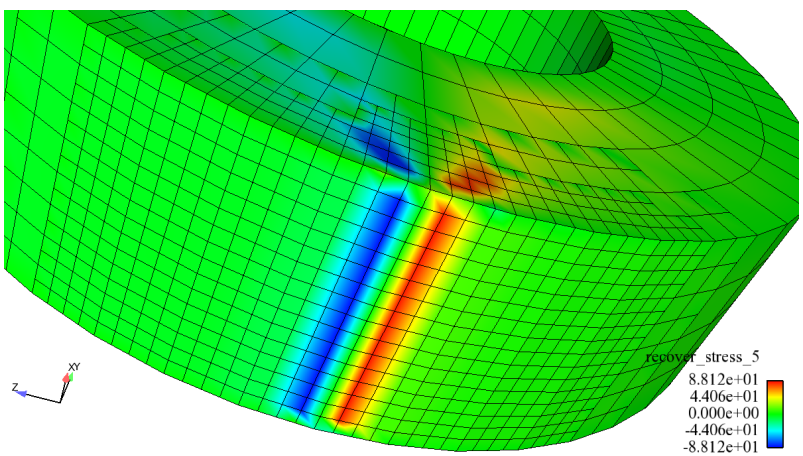
Log of Indicator

Recovery Indicator

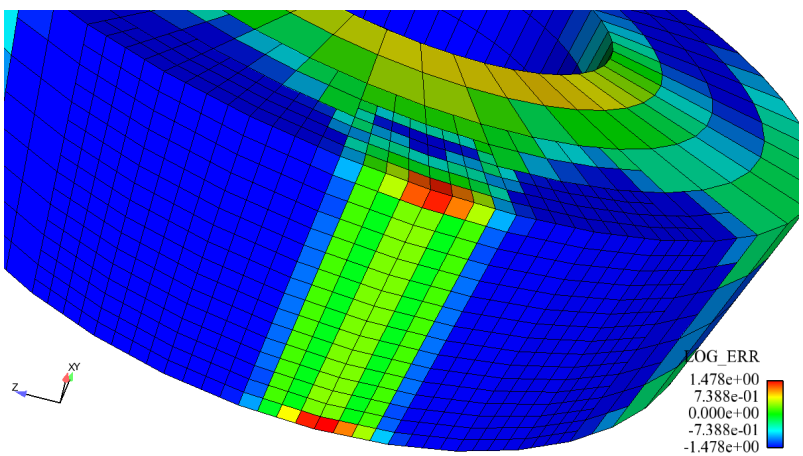
- Recovery Indicator computes difference between flux/stress and recovered (averaged) flux/stress (ZZ estimator).

$$\|\sigma - \sigma_h\| \approx \|\sigma^*(\sigma_h) - \sigma_h\|$$

- Example: simplistic rolling tire result from **Adagio** using feature based refinement in a box.
- Figures are of tire surface that is in contact with the road.



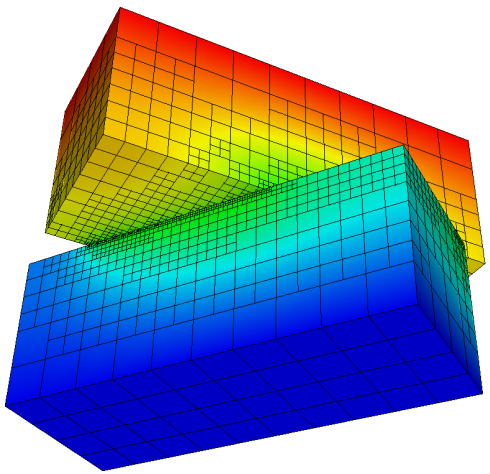
Recovered σ_{yz}



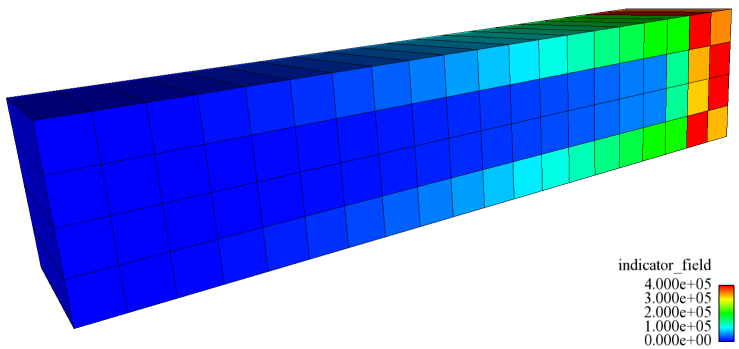
Log of Indicator

Jump Indicator

- Jump Indicator computes jumps (residuals) across inter-element interfaces.
- Gradients/fluxes produce normal jumps.
- Stresses produce jumps in tractions.



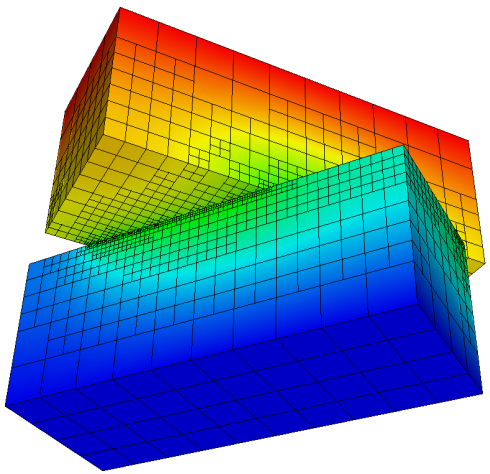
Calore thermal contact



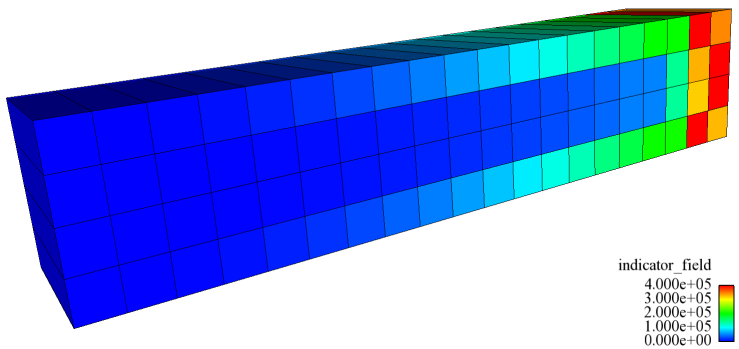
Adagio stress jump

Jump Indicator

- Jump Indicator computes jumps (residuals) across inter-element interfaces.
- Gradients/fluxes produce normal jumps.
- Stresses produce jumps in tractions.



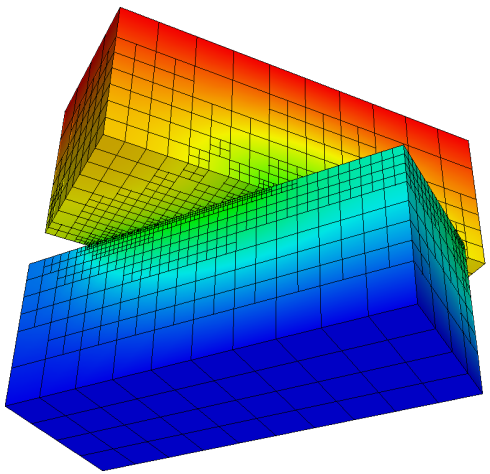
Calore thermal contact



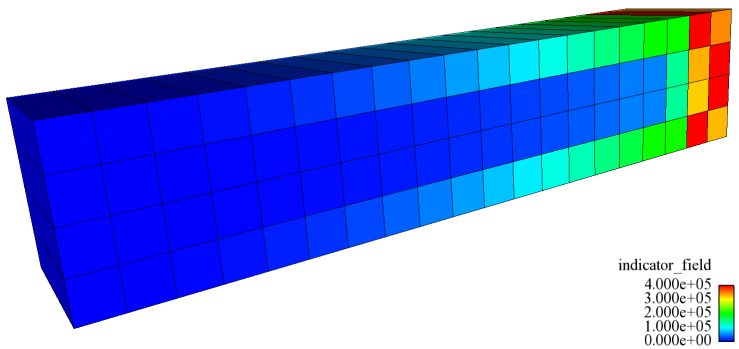
Adagio stress jump

Jump Indicator

- Jump Indicator computes jumps (residuals) across inter-element interfaces.
- Gradients/fluxes produce normal jumps.
- Stresses produce jumps in tractions.



Calore thermal contact



Adagio stress jump

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Adjoint Based Error Estimators

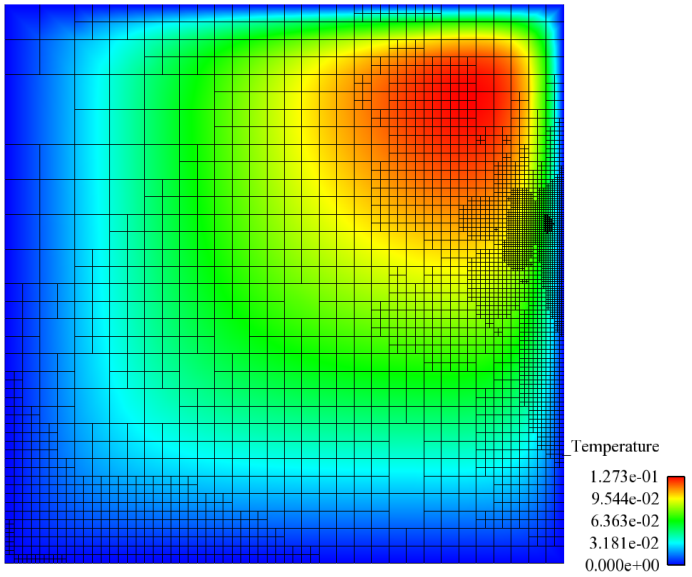
- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

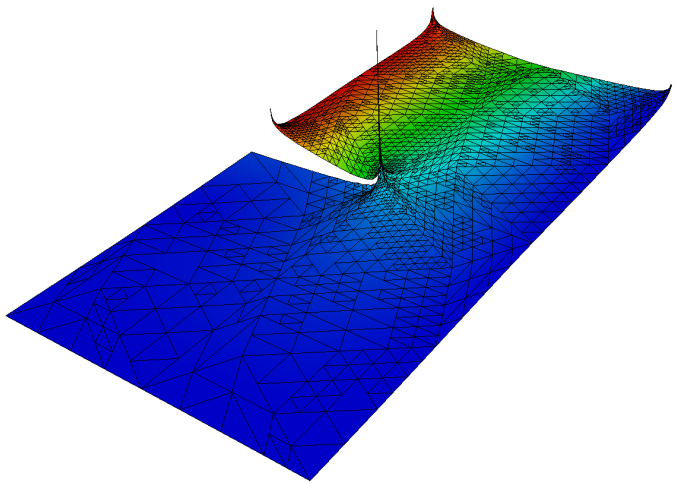
Adjoint Based Error Estimators

- Often analysts are interested in specific outputs of a simulation, rather than simulation fields:
  - Point values
  - Integral/average values
  - Surface fluxes
- These outputs correspond exactly to Encore PPs.
- Adjoint based error estimators and adaptivity are targeted to produce accurate PPs with efficient use of resources.
- These error estimators require an auxiliary linear adjoint solve.
- An FY08 ASC funded project (Algorithm Integration) is working on this technology for Sierra Mechanics codes.
- This work will directly impact the Sierra multiphysics code **Aria**.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.

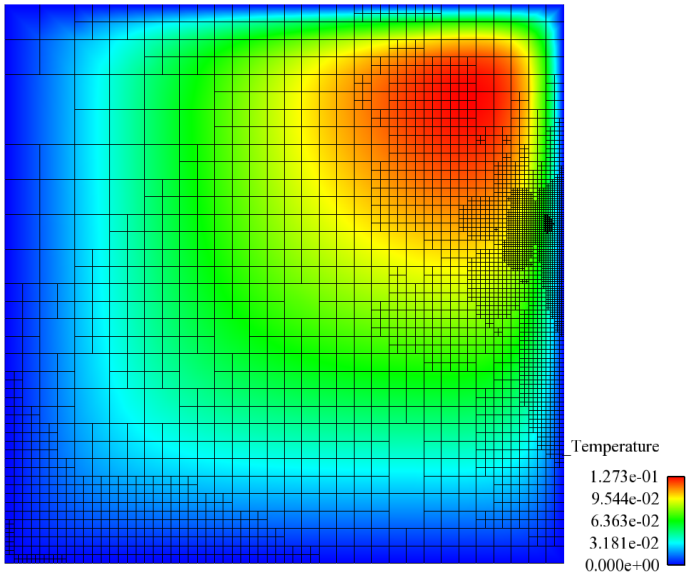


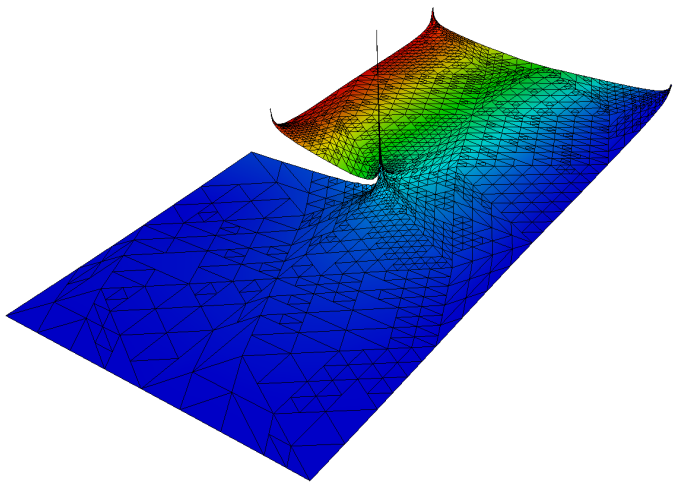


- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.

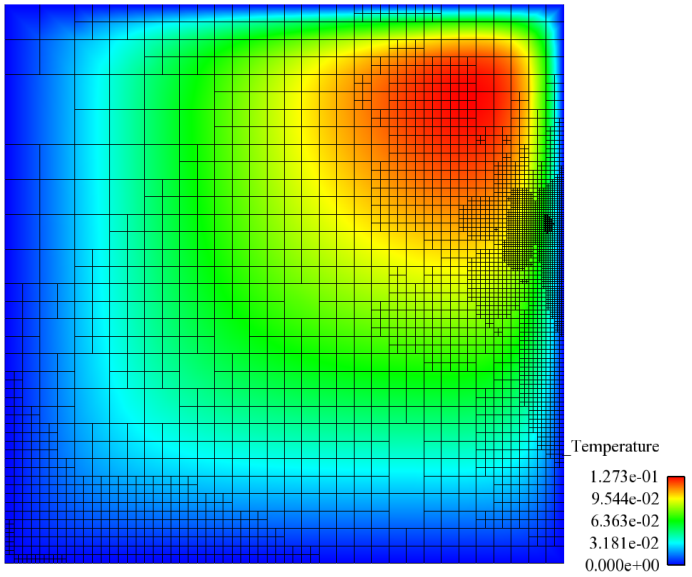


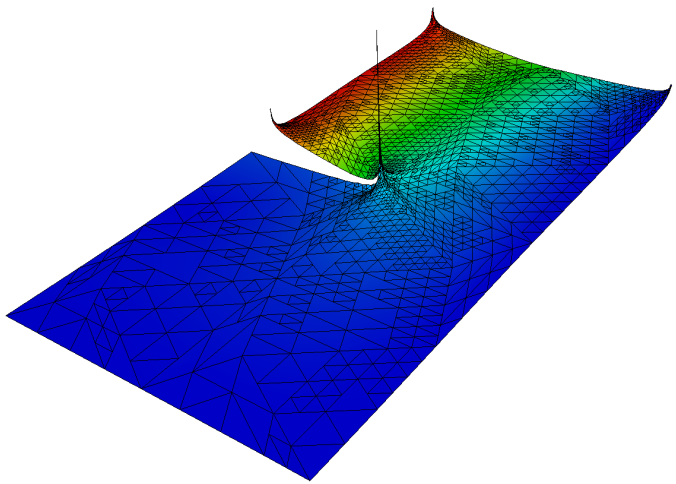


- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.

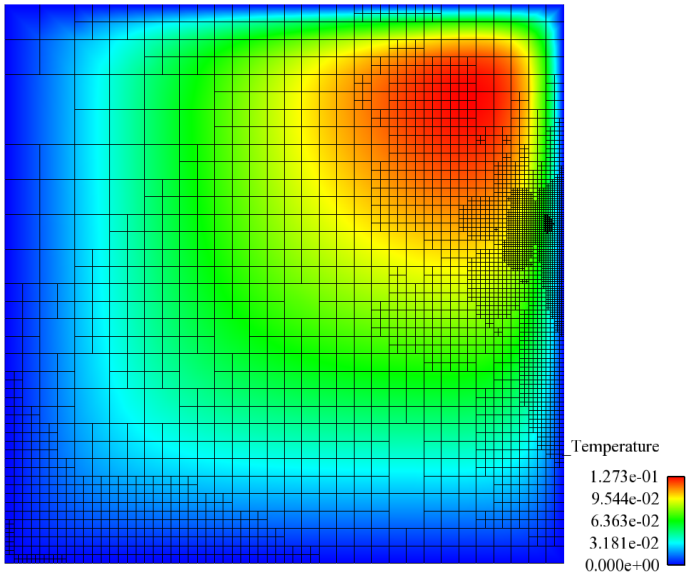


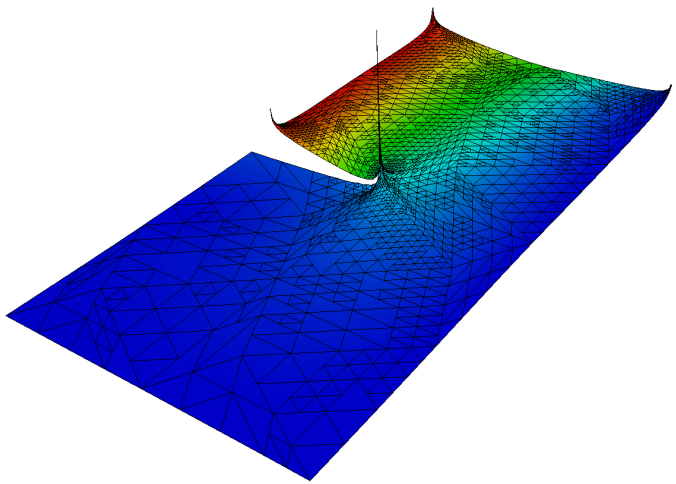


- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.

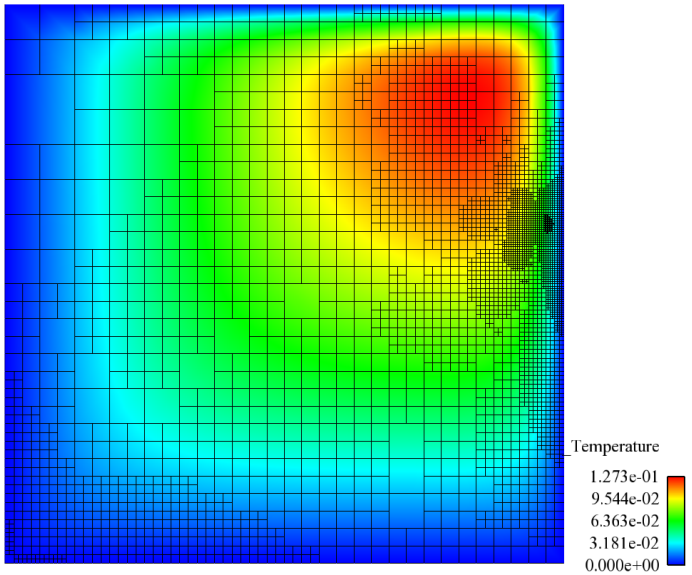


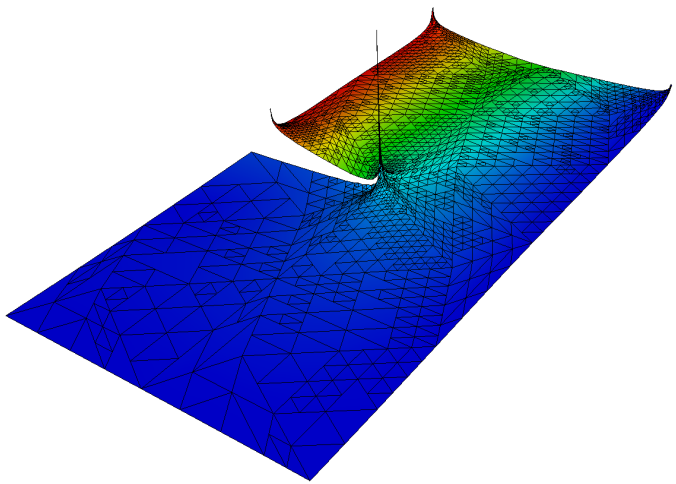


- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.

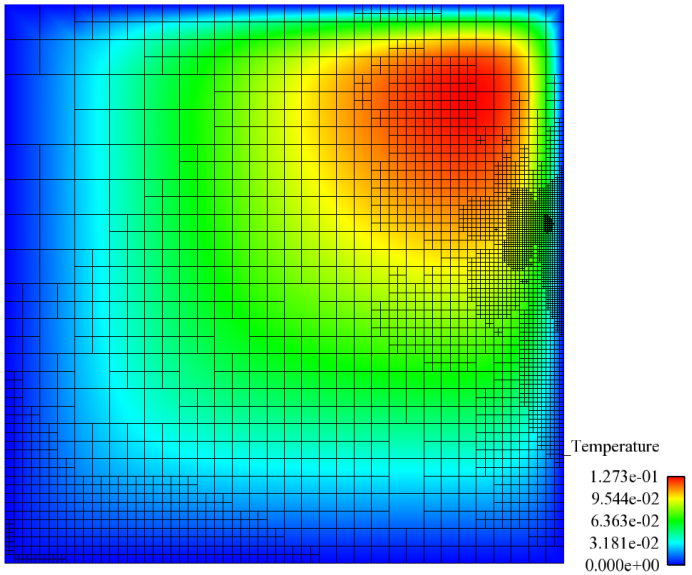


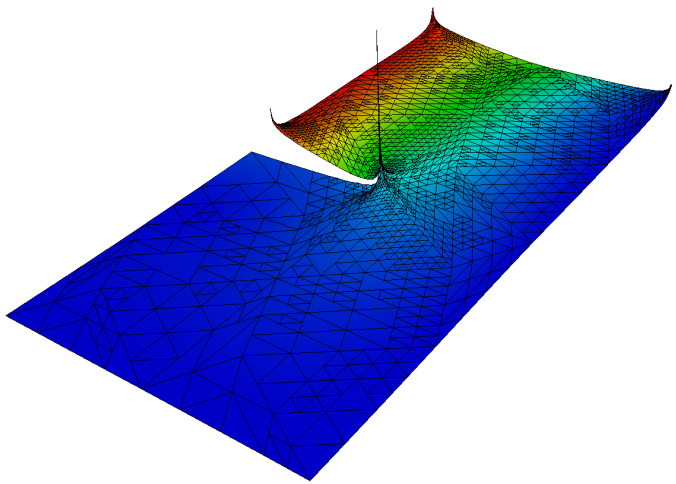


- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.

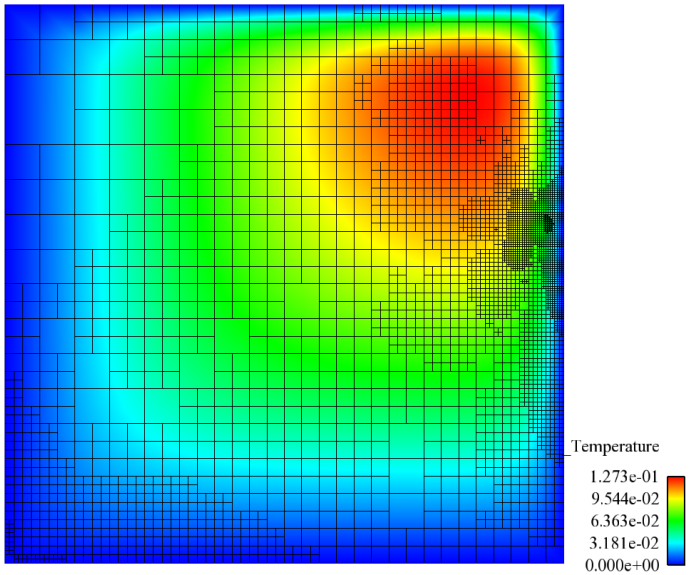


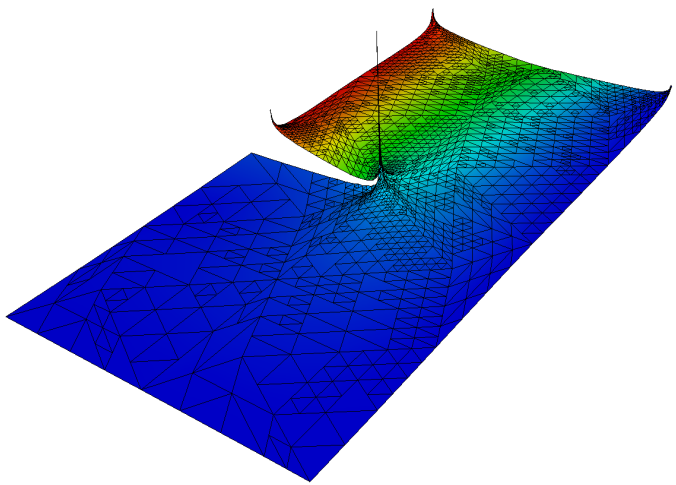


- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Examples: Adjoint Based Error Estimators

- Temperature field from **Calore** thermal advection-diffusion example.
- The PP is point evaluation near right boundary.
- The adjoint error estimator produces adaptivity that is optimal for this output.





- **Aria** nonlinear quasistatics example.
- Contours of Von Mises stress field colored by magnitude of adjoint displacement field.
- The PP is the integral surface traction on the upper left surface.
- The adaptivity resolves stress singularities critical to calculation of an accurate force-displacement curve.

Markers

- Mesh adaptivity requires elements to be marked to be refined, to be coarsened, or not at all.
- Markers can be
  - Error Based: element-wise error distribution in the simulation
  - Feature Based: geometric or solution features to resolve

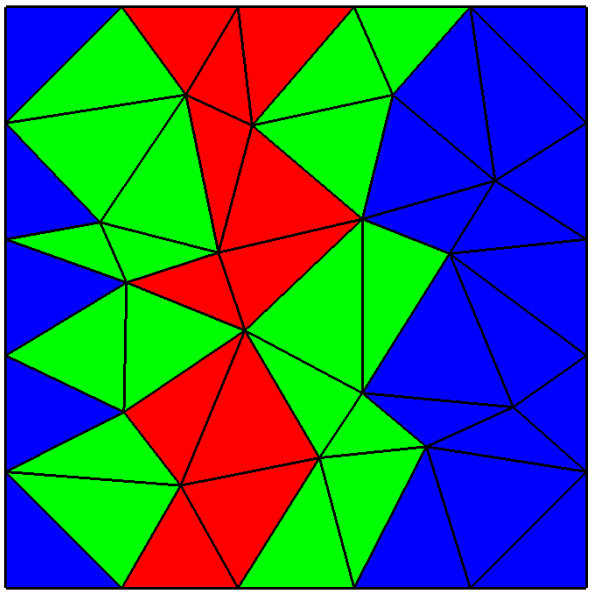


Figure: REFINE, DO NOTHING, COARSEN

Markers

- Mesh adaptivity requires elements to be marked to be refined, to be coarsened, or not at all.
- Markers can be
  - Error Based: element-wise error distribution in the simulation
  - Feature Based: geometric or solution features to resolve

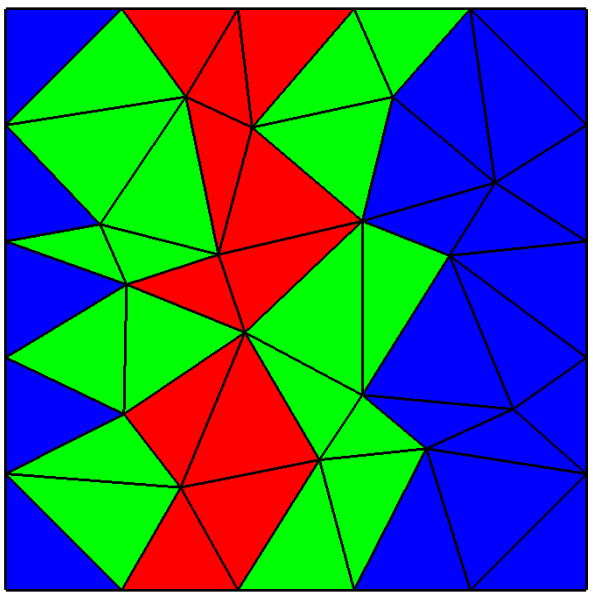


Figure: REFINE, DO NOTHING, COARSEN

Markers

- Mesh adaptivity requires elements to be marked to be refined, to be coarsened, or not at all.
- Markers can be
  - Error Based: element-wise error distribution in the simulation
  - Feature Based: geometric or solution features to resolve

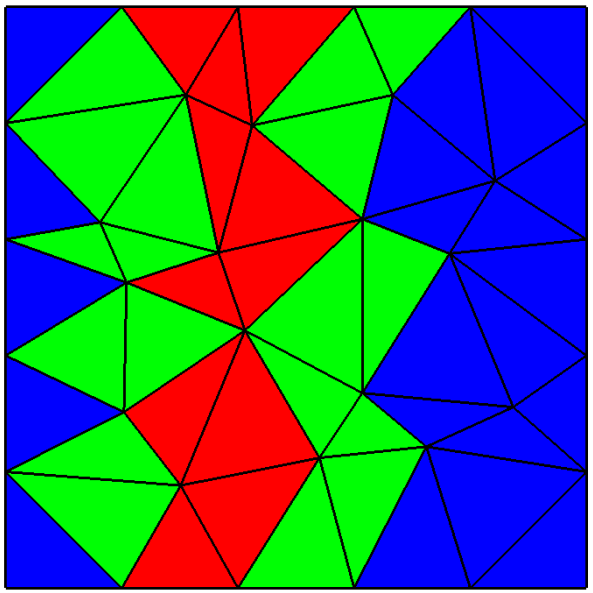


Figure: REFINE, DO NOTHING, COARSEN

Markers

- Mesh adaptivity requires elements to be marked to be refined, to be coarsened, or not at all.
- Markers can be
  - Error Based: element-wise error distribution in the simulation
  - Feature Based: geometric or solution features to resolve

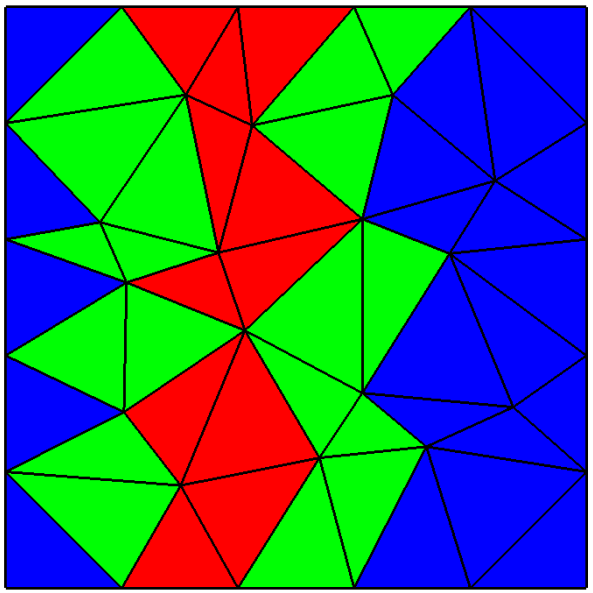
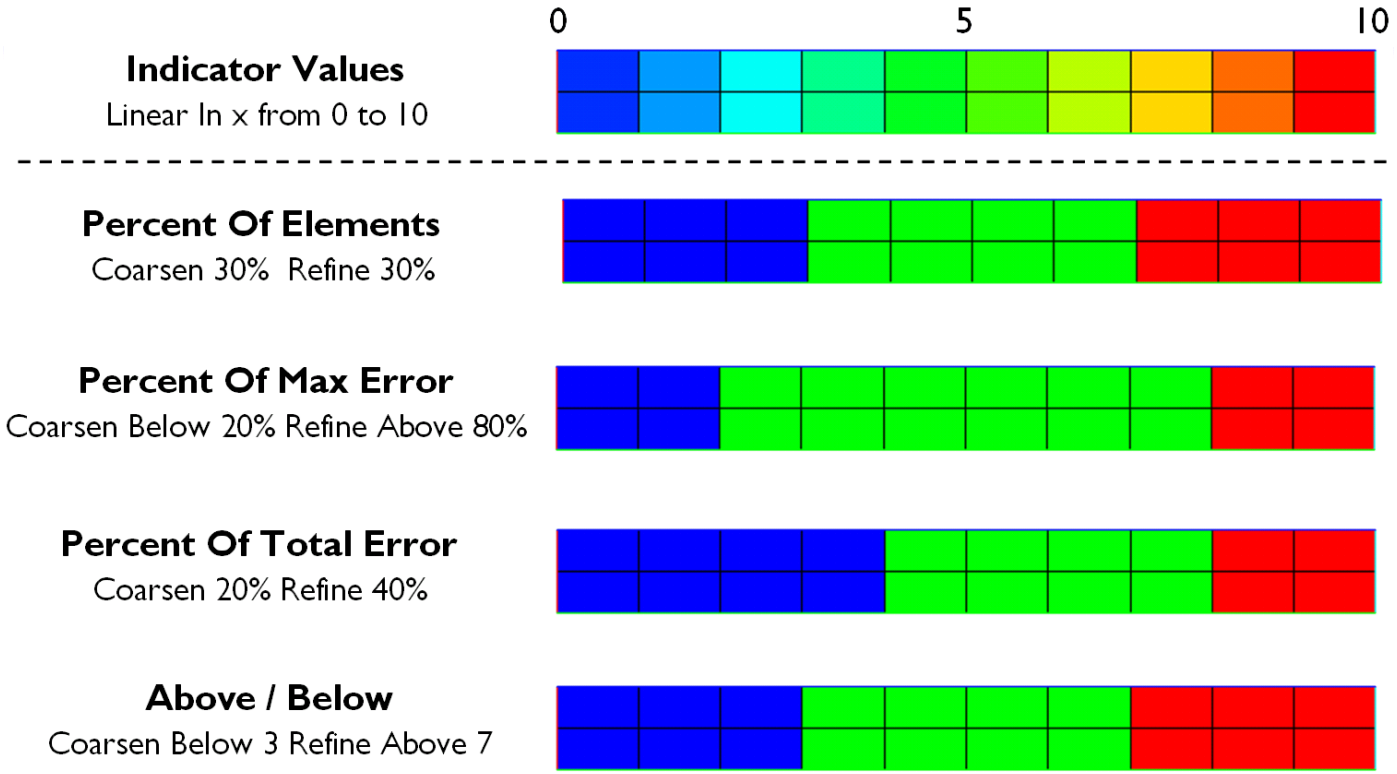


Figure: REFINE, DO NOTHING, COARSEN

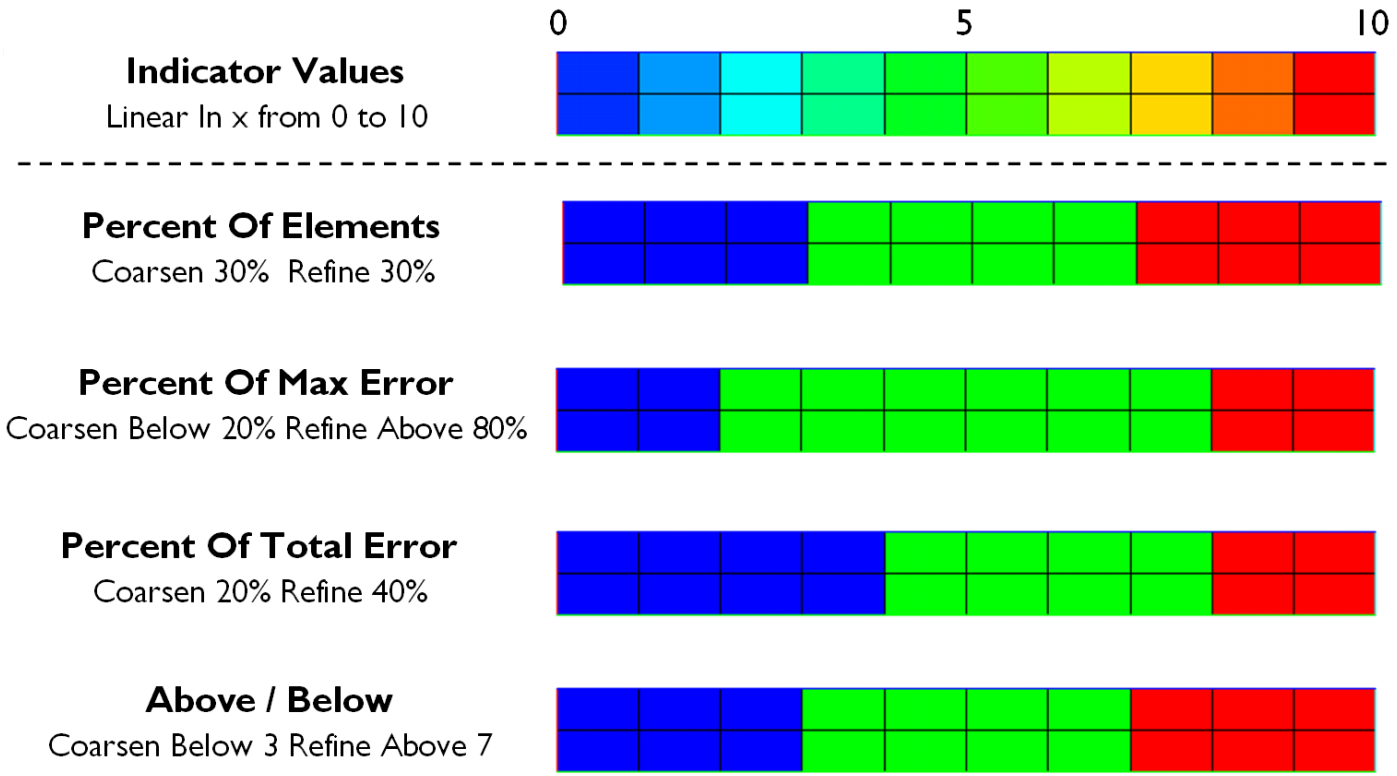
Error Based Markers

- Error based markers always take an element indicator field as input.
- The values are the element contributions to total error.
- Example: REFINE, DO NOTHING, COARSEN



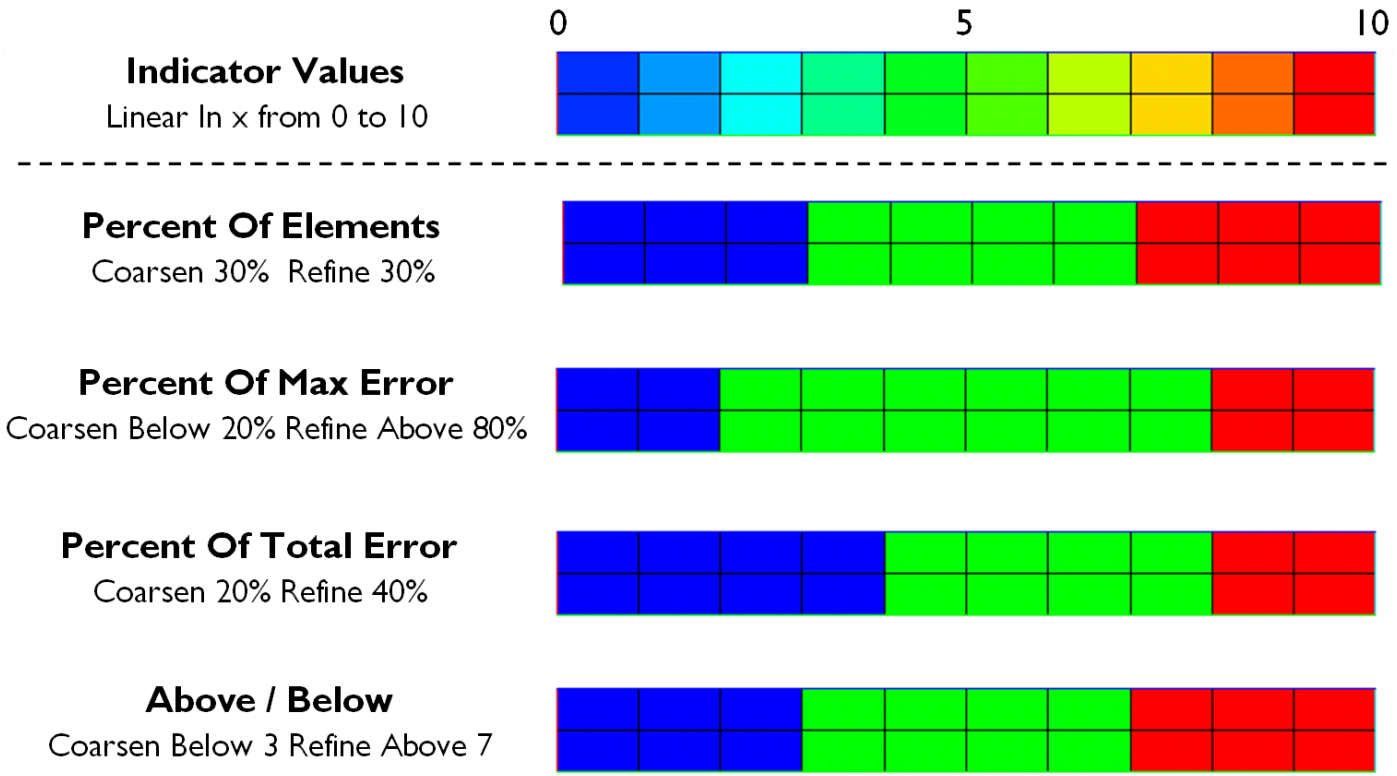
Error Based Markers

- Error based markers always take an element indicator field as input.
- The values are the element contributions to total error.
- Example: REFINE, DO NOTHING, COARSEN



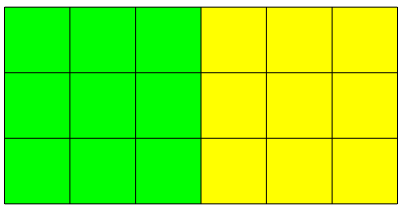
Error Based Markers

- Error based markers always take an element indicator field as input.
- The values are the element contributions to total error.
- Example: REFINE, DO NOTHING, COARSEN

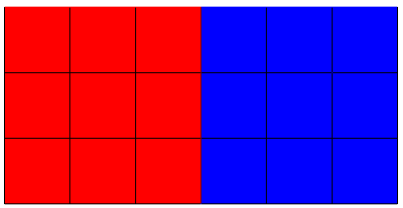


Feature Based Markers

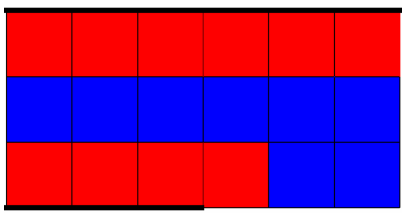
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



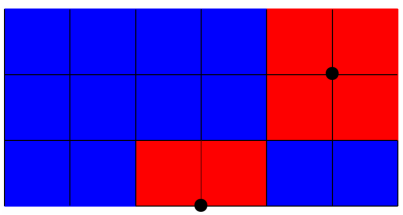
two element blocks



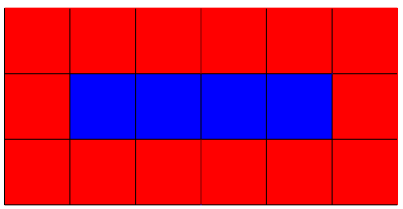
mark left block



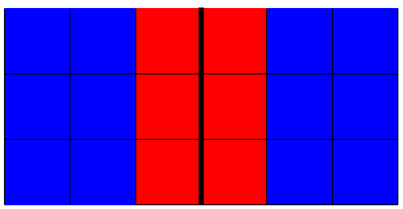
mark a sideset



mark a nodeset



exterior boundary

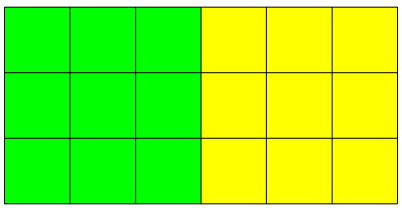


block interfaces

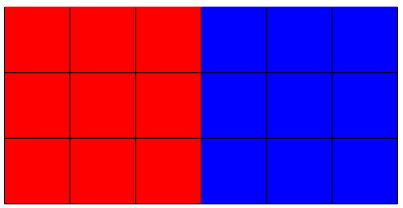
- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

Feature Based Markers

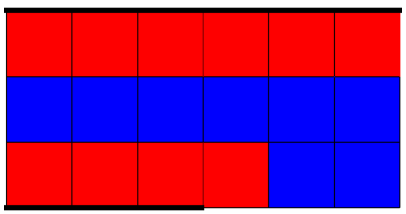
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



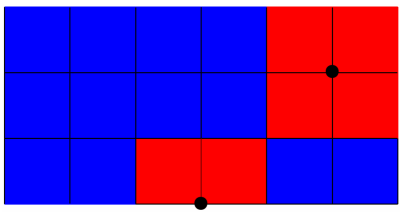
two element blocks



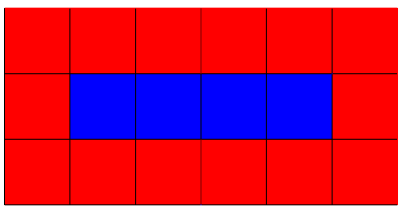
mark left block



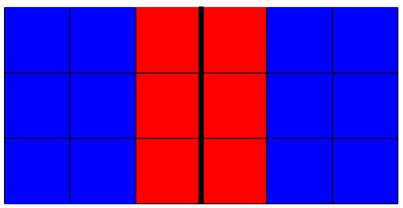
mark a sideset



mark a nodeset



exterior boundary

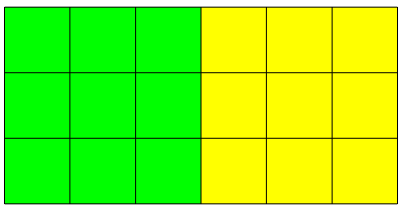


block interfaces

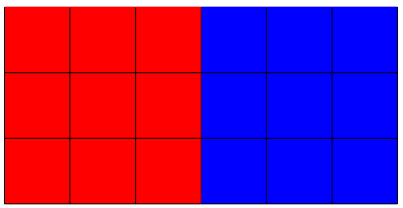
- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

Feature Based Markers

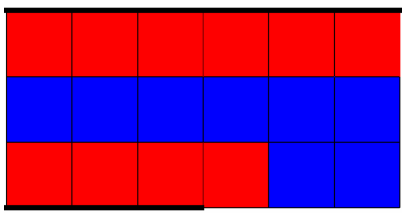
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



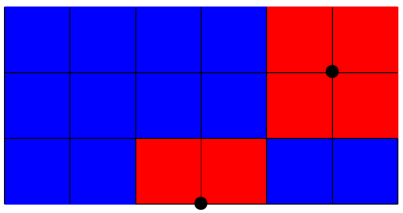
two element blocks



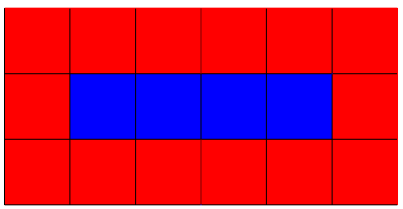
mark left block



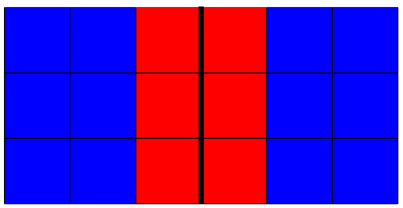
mark a sideset



mark a nodeset



exterior boundary

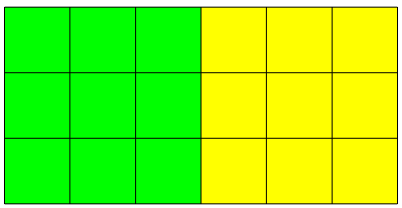


block interfaces

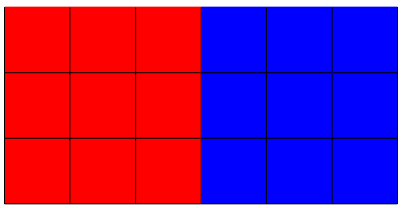
- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

Feature Based Markers

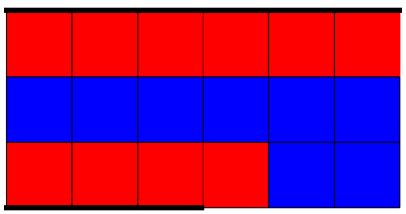
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



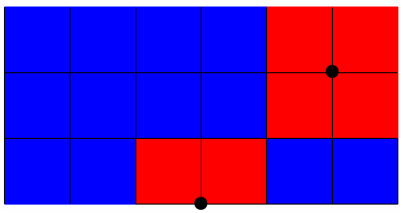
two element blocks



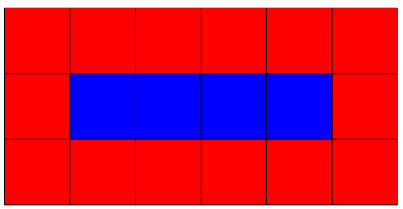
mark left block



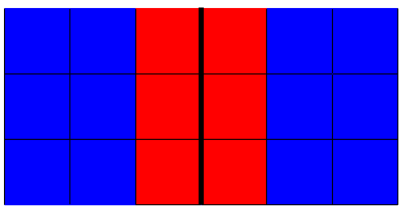
mark a sideset



mark a nodeset



exterior boundary

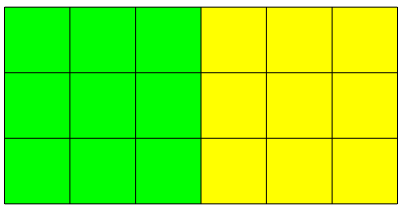


block interfaces

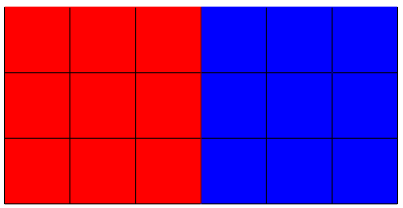
- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

Feature Based Markers

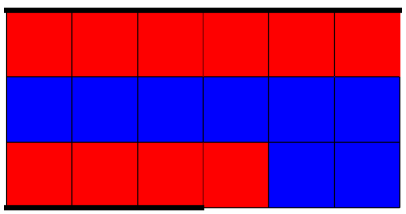
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



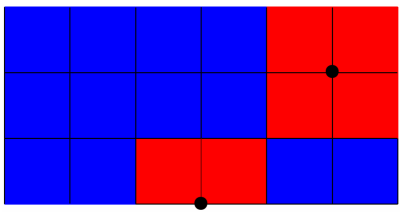
two element blocks



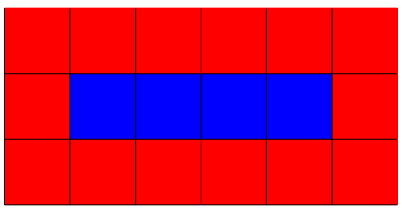
mark left block



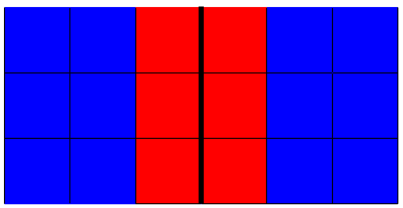
mark a sideset



mark a nodeset



exterior boundary

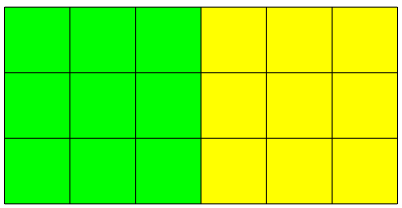


block interfaces

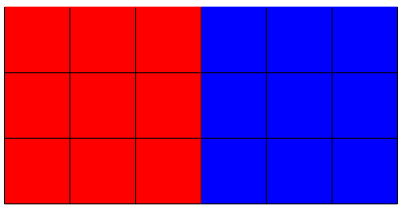
- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

Feature Based Markers

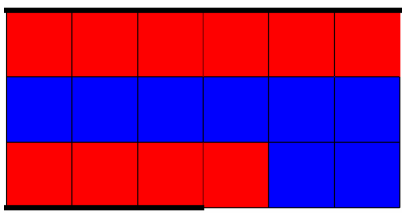
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



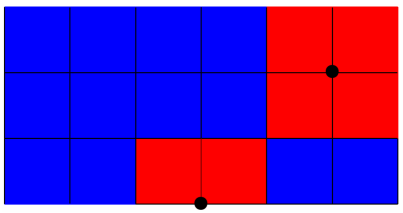
two element blocks



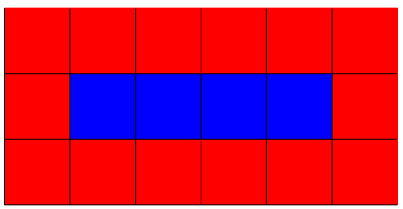
mark left block



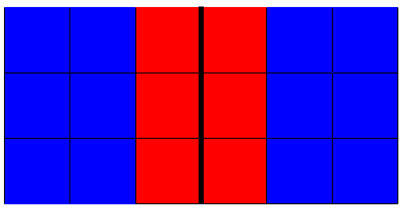
mark a sideset



mark a nodeset



exterior boundary

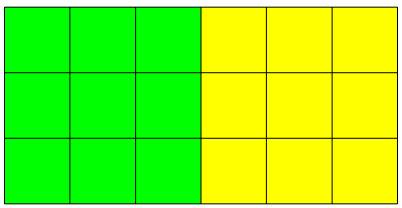


block interfaces

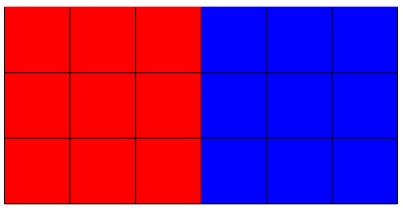
- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

Feature Based Markers

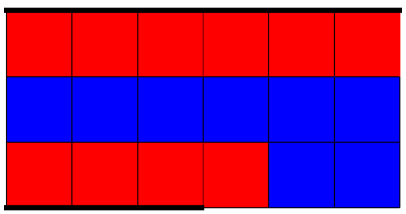
- Sometimes users need to adapt the mesh based on geometric/solution features.
- Example: some basic geometric markers



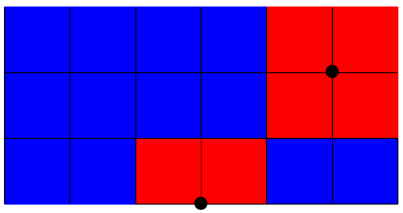
two element blocks



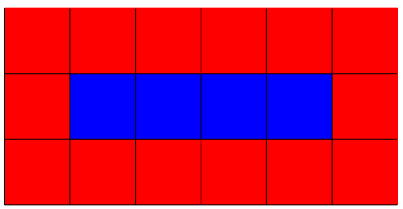
mark left block



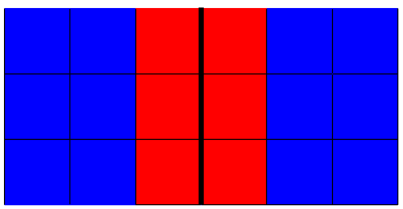
mark a sideset



mark a nodeset



exterior boundary

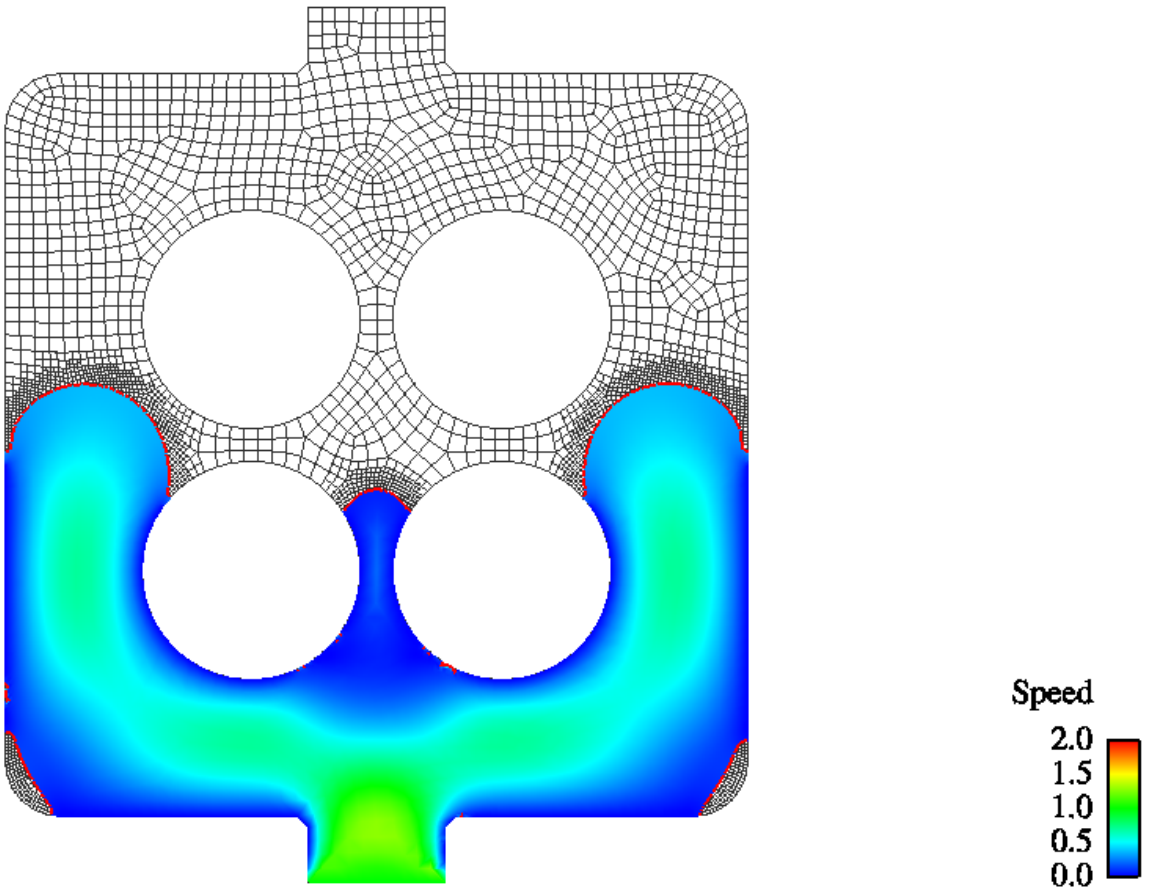


block interfaces

- Some additional markers
  - when solution values are in an interval
  - adjacent to re-entrant corners
  - within a specified distance to a level-set
  - within a bounding box

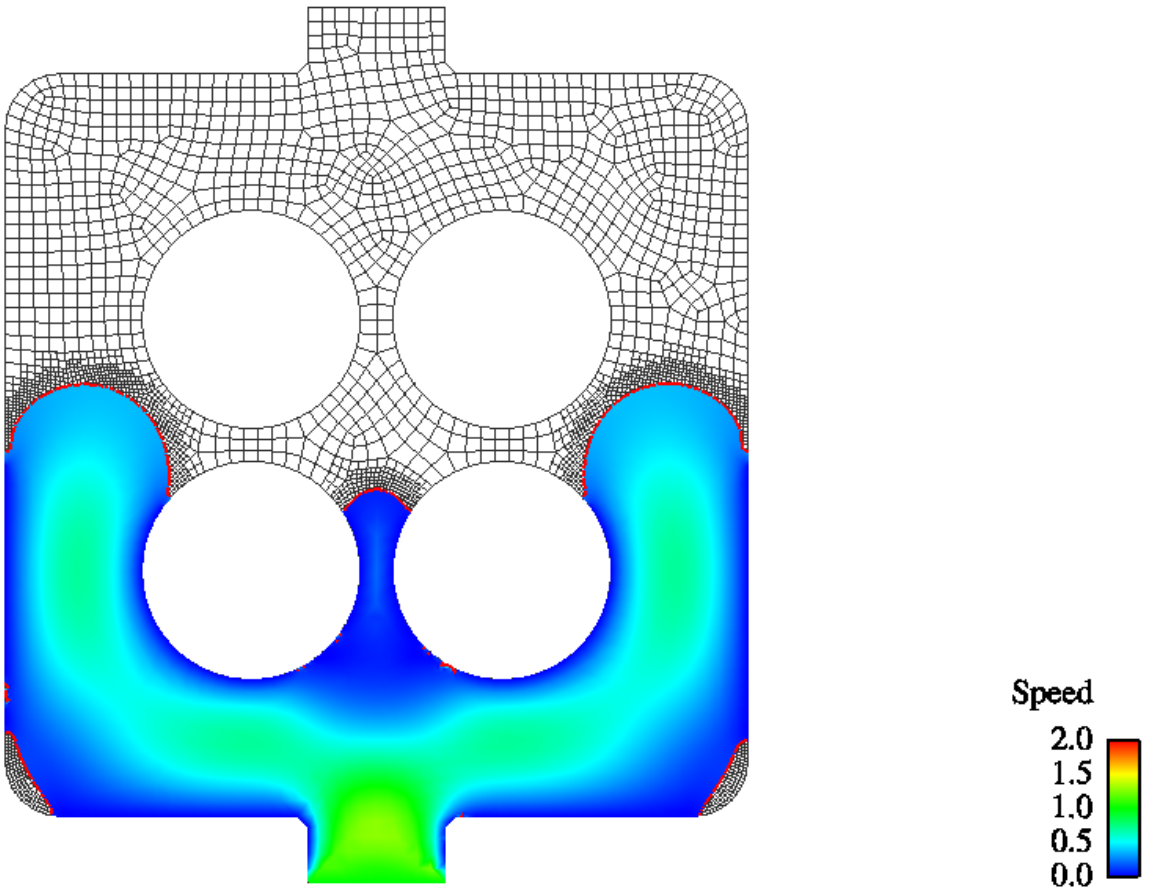
Impact: Feature Based Markers

- **Aria** is a Sierra Mechanics multiphysics code.
- A level set equation is used to model multiphase flow for mold filling.
- The **Level Set Marker** marks elements to resolve the level set.
- This can be combined with error based markers.



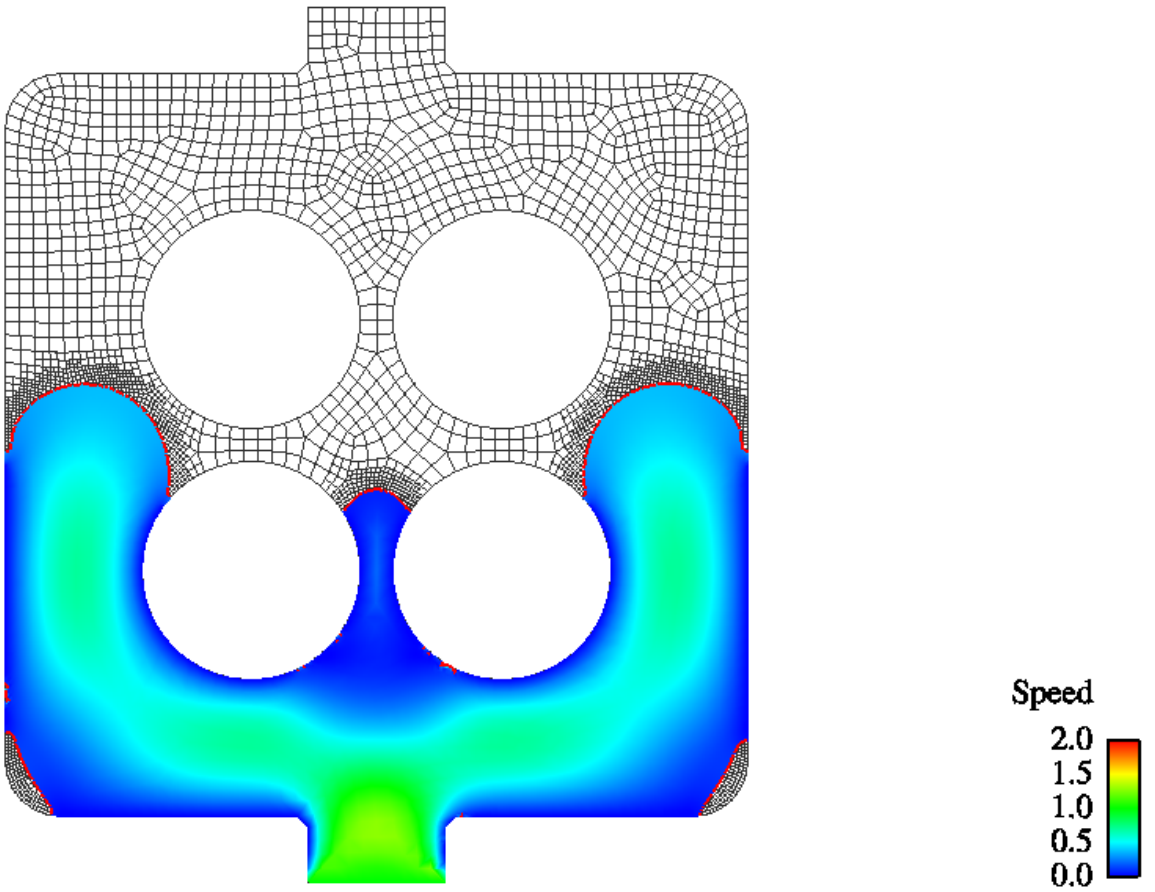
Impact: Feature Based Markers

- **Aria** is a Sierra Mechanics multiphysics code.
- A level set equation is used to model multiphase flow for mold filling.
- The **Level Set Marker** marks elements to resolve the level set.
- This can be combined with error based markers.



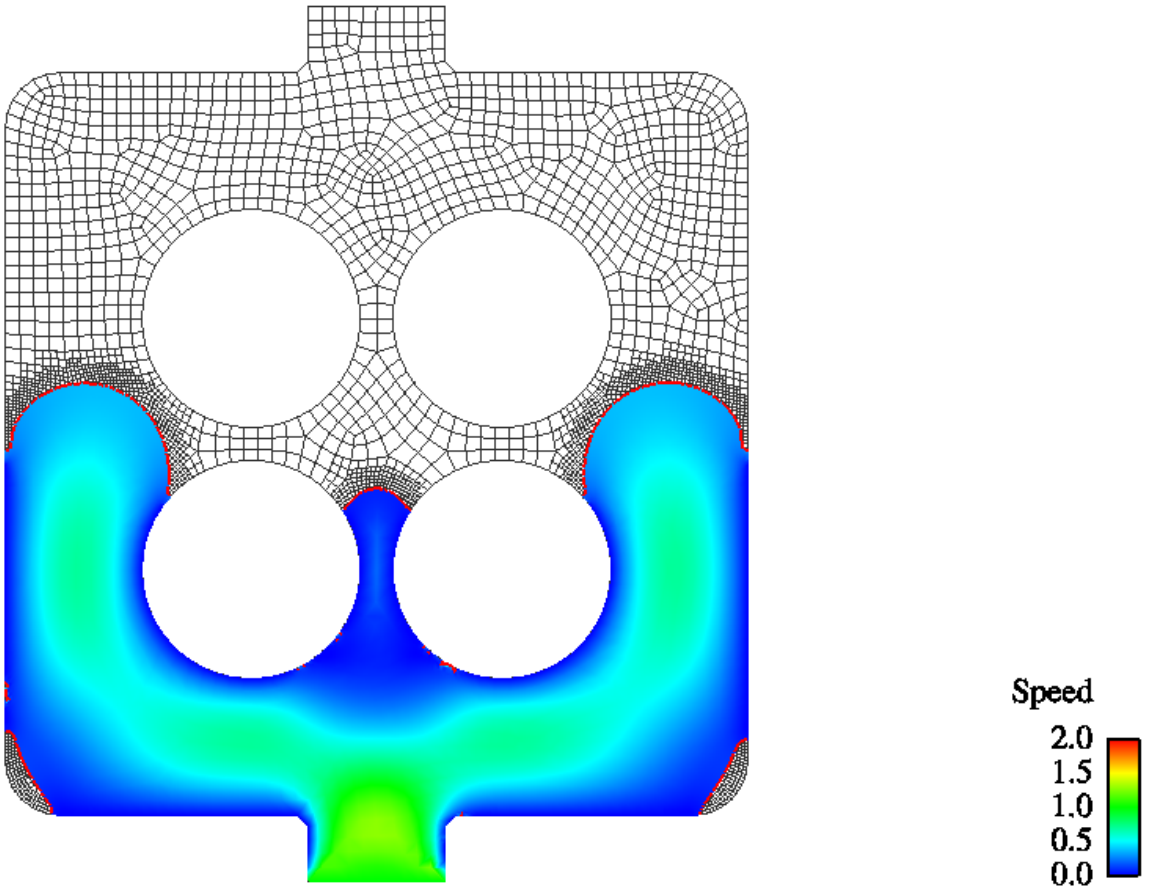
Impact: Feature Based Markers

- **Aria** is a Sierra Mechanics multiphysics code.
- A level set equation is used to model multiphase flow for mold filling.
- The **Level Set Marker** marks elements to resolve the level set.
- This can be combined with error based markers.



Impact: Feature Based Markers

- **Aria** is a Sierra Mechanics multiphysics code.
- A level set equation is used to model multiphase flow for mold filling.
- The **Level Set Marker** marks elements to resolve the level set.
- This can be combined with error based markers.



User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

User-Driven Control of Adaptivity

- Users can gain from using error estimation and adaptivity to verify the accuracy of calculations.
- This solution verification must be reliable and easy to use.
- **Encore** has implemented an adaptivity system that is transparent, flexible, and user-driven.
- This is combined with the physics independent and adjoint-based error estimators.
- Uses of adaptivity:
  - Initial adaptive refinement before solve
  - Uniform refinement of a mesh (no solve)
  - Adaptive refinement loop for stationary/transient problems
  - Adaptivity based on combining error and feature based markers

Outlook for Encore

Current and Future Activities

- Error estimation and adaptivity for transient non-linear simulations
- Verification of mechanical response
- Adjoint based sensitivity analysis
- Adjoint based error estimation for quantities of interest
- User requirements
- Linking directly to codes outside Sierra Mechanics

Outlook for Encore

Current and Future Activities

- Error estimation and adaptivity for transient non-linear simulations
- Verification of mechanical response
- Adjoint based sensitivity analysis
- Adjoint based error estimation for quantities of interest
- User requirements
- Linking directly to codes outside Sierra Mechanics

Outlook for Encore

Current and Future Activities

- Error estimation and adaptivity for transient non-linear simulations
- Verification of mechanical response
- Adjoint based sensitivity analysis
- Adjoint based error estimation for quantities of interest
- User requirements
- Linking directly to codes outside Sierra Mechanics

Outlook for Encore

Current and Future Activities

- Error estimation and adaptivity for transient non-linear simulations
- Verification of mechanical response
- Adjoint based sensitivity analysis
- Adjoint based error estimation for quantities of interest
- User requirements
- Linking directly to codes outside Sierra Mechanics

Outlook for Encore

Current and Future Activities

- Error estimation and adaptivity for transient non-linear simulations
- Verification of mechanical response
- Adjoint based sensitivity analysis
- Adjoint based error estimation for quantities of interest
- User requirements
- Linking directly to codes outside Sierra Mechanics

Outlook for Encore

Current and Future Activities

- Error estimation and adaptivity for transient non-linear simulations
- Verification of mechanical response
- Adjoint based sensitivity analysis
- Adjoint based error estimation for quantities of interest
- User requirements
- Linking directly to codes outside Sierra Mechanics