alamo Documentation

Brandon Runnels

Dec 13, 2023

CONTENTS:

1 Getting Started **3**
- 1.1 Downloading Alamo . . . 3
- 1.2 Installing dependencies . . . 3
- 1.3 Configuring . . . 4
- 1.4 Compiling . . . 4
- 1.5 Testing . . . 5
- 1.6 Common Error Messages . . . 5
- 1.7 Generating this documentation . . . 5
- 1.8 Compiling on STAMPEDE2 . . . 6

2 SimBA **7**
- 2.1 Running SimBA . . . 7
- 2.2 Updating SimBA Records . . . 8
- 2.3 Sharing results on the network . . . 8

3 Autodoc and Autotest **9**
- 3.1 Autodoc system . . . 9
  - 3.1.1 Parser comments . . . 9
  - 3.1.2 Header comments . . . 10
  - 3.1.3 ReStructuredText . . . 11
  - 3.1.4 Documentation generation . . . 11
- 3.2 Autotest system . . . 11
  - 3.2.1 Test requirements . . . 11
  - 3.2.2 Creating a test . . . 11
  - 3.2.3 Input file directives . . . 12
  - 3.2.4 Test verification . . . 12

4 Tests **15**
- 4.1 Eshelby . . . 16
  - 4.1.1 [Eshelby] 2D-serial-5levels . . . 16
  - 4.1.2 [Eshelby] 3D-serial-4levels . . . 17
  - 4.1.3 [Eshelby] 3D-parallel-5levels . . . 17
- 4.2 HeatConduction . . . 18
  - 4.2.1 [HeatConduction] 2d-serial . . . 19
  - 4.2.2 [HeatConduction] 3d-parallel . . . 19
- 4.3 Inclusion . . . 20
  - 4.3.1 [Inclusion] 2d-serial . . . 20
- 4.4 PerturbedInterface . . . 22
  - 4.4.1 [PerturbedInterface] perturbed-interface . . . 23
- 4.5 PlateHole . . . 24

4.5.1 [PlateHole] 2D-serial . . . 24
4.5.2 [PlateHole] 3D-parallel . . . 24
4.5.3 [PlateHole] 3D-serial . . . 25
4.6 RubberPlateHole . . . 26
4.6.1 [RubberPlateHole] serial-2d . . . 27
4.7 RubberWithInclusion . . . 28
4.7.1 [RubberWithInclusion] serial-2d . . . 28
4.7.2 [RubberWithInclusion] parallel-2d . . . 28
4.8 SCPSandwich . . . 30
4.8.1 [SCPSandwich] serial . . . 30
4.8.2 [SCPSandwich] parallel . . . 30
4.9 Scratch . . . 32
4.9.1 [Scratch] 2d-serial . . . 32
4.9.2 [Scratch] 2d-parallel . . . 32
4.10 Solid . . . 34
4.10.1 [Solid] linear-isotropic . . . 34
4.10.2 [Solid] linear-cubic . . . 34
4.10.3 [Solid] affine-cubic . . . 34
4.10.4 [Solid] affine-hexagonal . . . 34
4.10.5 [Solid] j2 . . . 35
4.10.6 [Solid] affine-isotropic . . . 35
4.10.7 [Solid] neo-hookean . . . 35
4.10.8 [Solid] neo-hookean-2d . . . 35
4.10.9 [Solid] pseudolinear-cubic . . . 36
4.11 Suture . . . 38
4.11.1 [Suture] 2D-serial-4levels . . . 38
4.12 TopOp . . . 40
4.12.1 [TopOp] serial . . . 40
4.13 TrigTest . . . 42
4.13.1 [TrigTest] 2D-1AMR-Levels-serial . . . 42
4.13.2 [TrigTest] 2D-2AMRLevels-serial . . . 42
4.13.3 [TrigTest] 2D-3AMRLevels-serial . . . 43
4.13.4 [TrigTest] 2D-3AMRLevels-parallel . . . 43
4.13.5 [TrigTest] 3D-1AMRLevels-serial . . . 43
4.13.6 [TrigTest] 3D-3AMRLevels-parallel . . . 43
4.14 UniaxialTension . . . 45
4.14.1 [UniaxialTension] 2D-serial-1level . . . 45
4.14.2 [UniaxialTension] 2D-serial-2levels . . . 45
4.14.3 [UniaxialTension] 2D-serial-3levels . . . 45
4.14.4 [UniaxialTension] 2D-parallel-3levels . . . 46
4.14.5 [UniaxialTension] 3D-serial-3levels . . . 46
4.14.6 [UniaxialTension] 3D-parallel-3levels . . . 46
4.15 Voronoi . . . 49
4.15.1 [Voronoi] 2D-100grain-parallel . . . 49
4.15.2 [Voronoi] 2D-100grain-serial . . . 49
4.16 VoronoiElastic . . . 50
4.16.1 [VoronoiElastic] 2d-serial . . . 51
4.16.2 [VoronoiElastic] 2d-parallel . . . 51

5 Inputs **55**
5.1 BC . . . 55
5.1.1 BC::Constant . . . 55
5.1.2 BC::Step . . . 57
5.1.3 BC::Operator . . . 58

5.1.3.1 BC::Operator::Elastic . . . 58
5.2 IC . . . 62
5.2.1 IC::Affine . . . 62
5.2.2 IC::BMP . . . 62
5.2.3 IC::Constant . . . 62
5.2.4 IC::Cuboid . . . 63
5.2.5 IC::DoubleNotch . . . 63
5.2.6 IC::Expression . . . 63
5.2.7 IC::Ellipsoid . . . 64
5.2.8 IC::Ellipse . . . 64
5.2.9 IC::Laminate . . . 65
5.2.10 IC::Notch . . . 65
5.2.11 IC::PS . . . 65
5.2.12 IC::PSRead . . . 66
5.2.13 IC::PerturbedInterface . . . 66
5.2.14 IC::Sphere . . . 66
5.2.15 IC::TabulatedInterface . . . 67
5.2.16 IC::Trig . . . 67
5.2.17 IC::Voronoi . . . 67
5.3 IO . . . 67
5.3.1 IO::ParmParse . . . 67
5.3.2 IO::WriteMetaData . . . 67
5.4 Integrator . . . 68
5.4.1 Integrator::CahnHilliard . . . 69
5.4.2 Integrator::Fracture . . . 69
5.4.3 Integrator::Flame . . . 70
5.4.4 Integrator::HeatConduction . . . 70
5.4.5 Integrator::Mechanics . . . 71
5.4.6 Integrator::PolymerDegradation . . . 72
5.4.7 Integrator::PhaseFieldMicrostructure . . . 72
5.4.8 Integrator::SutureCrack . . . 74
5.4.9 Integrator::ThermoElastic . . . 74
5.4.10 Integrator::TopOp . . . 74
5.4.11 Integrator::Base . . . 75
5.4.11.1 Integrator::Base::Mechanics . . . 75
5.5 Model . . . 75
5.5.1 Model::Interface . . . 75
5.5.1.1 Model::Interface::Crack . . . 75
5.5.1.2 Model::Interface::GB . . . 76
5.5.2 Model::Solid . . . 77
5.5.2.1 Model::Solid::Affine . . . 77
5.5.2.2 Model::Solid::Elastic . . . 80
5.5.2.3 Model::Solid::Linear . . . 80
5.6 Numeric . . . 83
5.6.1 Numeric::Interpolator . . . 83
5.6.1.1 Numeric::Interpolator::Linear . . . 83
5.6.1.2 Numeric::Interpolator::NodeBilinear . . . 84
5.7 Operator . . . 84
5.7.1 Operator::Elastic . . . 84
5.8 Set . . . 84
5.9 Solver . . . 85
5.9.1 Solver::Nonlocal . . . 85
5.9.1.1 Solver::Nonlocal::Linear . . . 85
5.9.1.2 Solver::Nonlocal::Newton . . . 86

5.10 Util . . . 86

6 Inputs Search **87**

7 </> Developer Guide **97**
7.1 Step-by-step development guide . . . 97
7.2 Tutorial: A new Integrator . . . 99
7.3 Conventions . . . 99
7.3.1 Namespaces . . . 99
7.3.1.1 Regular and Pure Virtual classes . . . 100
7.3.2 Include Guards . . . 100
7.3.3 Member and Argument Variable Names . . . 100
7.4 Python (In development) . . . 100
7.4.1 Compiling Alamo Python interface . . . 101
7.4.2 Extending the Alamo Python interface . . . 101
7.5 Restart files . . . 102

⏩ Getting Started

Initial step-by-step instructions for downloading, building, and running Alamo. Basic instructions on troubleshooting.

Start here if you're brand new

♟ Running Tests

Explore the set of examples and regression tests that demonstrate Alamo functionality.

Next steps after compiling Alamo

🐾 SimBA

Use the Alamo **Sim**ulation **B**rowser **A**nalysis system to manage Alamo simulations for regression tests or your own projects.

Optional but useful for regular users

📦 API and Inputs

Browse the auto-generated documentation for Alamo classes and input file input specifications.

For those using existing code to run new siulations.

</> Developer Instructions

Basics of Alamo design philosophy and practice. How to create branches, implement new models, run tests, and avoid breaking the repository.

Starting point for all new alamo developers

Alamo's autodoc system

Alamo's custom auto-documentation and test system designed to enforce documentation and testing standards, while preventing stagnant/depricated content.

Required reading for all developers

Table of contents

CHAPTER

ONE

GETTING STARTED

Note: this README page is also the Doxygen main page, the Github readme page, and the Docs main page. You can view it by running `make docs` in the root directory, then opening `docs/doxygen/html/index.html` or `docs/build/html/index.html` in a web browser.

1.1 Downloading Alamo

Download alamo using git:

```
git clone git@github.com:solidsgroup/alamo.git
```

If you do not have a Github account and/or you have not uploaded your public SSH key, this will probably throw an error. You can download alamo using HTTPS instead,

```
https://github.com/solidsuccs/alamo.git
```

Note, however, that you will not be able to push anything using HTTPS authentication. The `master` branch is the most stable and is what is checked out by default. The `developent` branch is generally stable, and includes the latest functionality. To switch to `development`, in the alamo directory,

```
git checkout development
```

1.2 Installing dependencies

Alamo depends primarily on AMReX, but has some of its own dependencies, such as Eigen. The two officially supported Linux distributions are currently: the latest version of Ubuntu, and Ubuntu 20.04. (This is subject to change). If you are using Ubuntu (or a debian-based distribution) you can install all necessary dependencies by running the dependencies script:

```
sudo bash .github/workflows/dependencies.sh
```

If you are using a non-debian based system such as RedHat, you can install the corresponding packages in `dependencies.sh`. Windows and Mac OS are not supported.

1.3 Configuring

To compile alamo, you must first run the configure script. This is done simply by running the following in the alamo directory (note that AMReX download is triggered by this command, so it may take a couple minutes to complete depending on your internet connection)

```
./configure
```

By default, alamo will configure in 3D production mode. To compile in 2D debug mode,

```
./configure --dim=2 --debug
```

There are multiple compilation options available for Alamo, and they must all be specified at configure time. For a complete listing of the Alamo configuration options, type

```
./configure --help
```

Note: The configure script produces output designed to assist in determining compile issues with Alamo. Whenever you request help with alamo, please always include the complete output of the configure script.

1.4 Compiling

Once you have configured Alamo, compile it by

```
make
```

If you are on a platform with multiple cores, you can compile in parallel (for instance, with 4 cores) with

```
make -j4
```

The alamo exectuable will be stored in `./bin/` and name according to the options specified at configure time. For instance, if you are using GCC to make Alamo in 2D using debug mode, the alamo executable will be `./bin/alamo-2d-debug-g++`. You can work with multiple versions of Alamo at the same time without having to re-compile the entire code base. All you need to do is re-run the configure script, and previous versions of Alamo and AMReX will be saved automatically.

Warning: There is an issue with GNU Make that can cause I/O errors during parallel builds. You may get the following error:

```
make[1]: write error: stdout
```

To continue the build, just issue the `make` command again and it should continue normally. You can also add the `--output-sync=target` option which may help eliminate the issue.

1.5 Testing

Upon successful compilation, run tests by

```
make test
```

This will run the unit tests and regression tests for all compiled production versions of Alamo. If you have only run in 2D, only 2D tests will be generated. If you are a developer and you are preparing to merge your branch into `development`, you should perform a complete test via

```
./configure --dim=2
make
./configure --dim=3
make
make test
```

For a full description of the Alamo regression test system, please see

1.6 Common Error Messages

The following are some common error messages and problems encountered:

- `MLLinOp: grids not coarsenable between AMR levels` This is a conflict in the **multigrid solver** because the grid size is not a power of 2. Solve by changing the domain dimensions (*amr.n_cell*) so that they are powers of two.
- `static_cast<long>(i) < this->size() failed` One common reason this happens is if Dirichlet/Neumann boundaries are specified but no boundary values are provided.
- `error: lvalue required as left operand of assignment` This can happen when using the `()` operator with a `Isotropic Matrix4`-type object. Because this data structure only stores two constants, it is not possible to define any of the values using indices. (Similarly, you cannot set an `Isotropic` 4-matrix to a `Cubic` 4-matrix since the Cubic matrix has lower symmetry). If you get this error, you should use a lower-symmetry 4-matrix.
- `Inconsistent box arrays` This is known to happen when using an `Operator::Elastic` inside an `Integrator`, e.g. in `TimeStepBegin`. Typically this happens when the Elastic operator is not initialized within the routine in which it is used - i.e.e if it is declared as a member variable inside the `Integrator` - derived class. (The reason is that there are AMReX-specific functions that only get called by the constructor.) The fix is to initialize the operator object inside the routine in which it is used - either by making the member variable a pointer and using the `new` keyword, or by just creating the variable inside the function.

1.7 Generating this documentation

Generating documentation requires the following packages:

- Doxygen (on Ubuntu: `sudo apt install doxygen`)
- Sphinx (on Ubuntu: `sudo apt install python3-sphinx`)
- Breathe (on Ubuntu: `sudo apt install python3-breathe`)
- M2R (on Ubuntu: `python3 -m pip install m2r`)
- RTD theme (on Ubuntu: `python3 -m pip install sphinx_rtd_theme`)

- GraphViz (on Ubuntu: `sudo apt install graphviz`)

To generate the documentation, type

```
make docs
```

(You do not need to run `./configure` before generating documentation.) Documentation will be generated in *docs/build/html* and can be viewed using a browser.

1.8 Compiling on STAMPEDE2

To compile on STAMPEDE2 you must first load the following modules:

```
module load python3
```

This will load Python3. The following configure script is recommended:

```
./configure --build-amrex --get-eigen --comp=icc
```

where other arguments (e.g. `--dim=2`) can be added as necessary. Finally, make with

```
make
```

Warning: Remember to use good stewardship when compiling and running on a supercomputer. (For instance, do *not* use `make -j16` to build.) Be sure to consult the Stampede2 user guide: https://portal.tacc.utexas.edu/user-guides/stampede2; along with https://solids.uccs.edu/resources/xsede.php for general Stampede2/XSEDE instructions.

CHAPTER

TWO

SIMBA

Reproducibility is critical to ensuring the integrity of simulation results. SimBA is a set of python scripts designed to manage, organize, and view Alamo simulations through a web browser. It integrates with Alamo's metadata recording functionality and uses a database to manage simulation information.

2.1 Running SimBA

Suppose you have a directory containing a collection of simulation results.

```
output
output.old.1111111
output.old.2222222
output.old.3333333
```

Begin by running `./simba/database.py output*`. This should produce the following output:

```
ADDED TABLE: simulation_data
├Inserting: output
├Inserting: output.old.1111111
├Inserting: output.old.2222222
├Inserting: output.old.3333333
└Done
```

and should create a database called `results.db` in your directory. Note that the database name and table name are all customizable.

Now, you can start the web interface with `./simba/web.py`. Open a browser and go to `localhost:5000`. You should see each of your simulation entries listed. You can click on each entry to see greater detail, and you can edit the "Description" and "Tags" fields to record information about the simulation. You can also delete simulations (and the corresponding directory).

The SimBA web interface will automatically recognize images stored within the simulation directory, and will display them under the simulation entry.

2.2 Updating SimBA Records

Simulation directories are never overwritten - following the AMReX implementation, old directories are renamed if there is a naming conflict. SimBA recognizes this and accounts for it. In the above example, suppose that another simulation was run so that `output` was renamed to be `output.old.4444444`. Running SimBA again produces the following output

```
ADDED TABLE: simulation_data
├Inserting: output
├Updating:  output.old.1111111 ( record already exists )
├Updating:  output.old.2222222 ( record already exists )
├Updating:  output.old.3333333 ( record already exists )
├Moving:    output --> output.old.4444444
└Done
```

Alamo records a unique identifier for each simulation that is stored in the `metadata` file - this way simulations can be moved around and renamed without losing their designation in the SimBA database.

Note: SimBA does need to be updated using the `database.py` script - the web interface does not do this automatically (yet).

2.3 Sharing results on the network

An advantage of SimBA is that it allows the posting of results to be viewed over the internet. For instance, suppose you want someone else on your network to be able to view your SimBA page and the outputs that you have generated. To do this, start the web interface this way:

```
./simba/web -i 123.456.789.10
```

where `123.456.789.10` is your computer's IP address, which you can find (on Linux) using `ifconfig`.

Danger: Putting your results online allows other people to have interactive access with your database and your results! Unless you are on a private network, you should use the `-s` option to run in "Safe Mode" - this disables the deleting/editing capabilities.

Once SimBA is online, other people on your network can view the page by going to `123.456.789.10:5000` in their browser.

CHAPTER

THREE

AUTODOC AND AUTOTEST

Alamo includes a self-contained automatic documentation (autodoc) and automatic test (autotest) system. The purpose of the custom autodoc and autotest systems is to:

1. Make it easy for developers to add documentation and regression tests
2. Keep documentation and tests accessible and up-to-date
3. Prevent old or out-of-date documentation.

3.1 Autodoc system

The autodoc system parses the source files stored in `./src/` to generate the content of the Inputs portion of this documentation. There are two ways to create compliant documentation: via parser comments, and via header file comments.

3.1.1 Parser comments

The `IO::ParmParse` class (inherited from the AMReX `ParmParse` class) is used to read all input paramters from the input file, usually though lines like this:

```
pp.query("lame",lambda);
pp.query("shear",mu);
```

To generate the autodocumentation, simply add a standard C++ comment following the query line:

```
pp.query("lame",lambda); // Lame parameter
pp.query("shear",mu);    // Shear modulus
```

or immediately preceeding the query line:

```
// Lame parameter
pp.query("lame",lambda);
// Shear modulus
pp.query("shear",mu);
```

Note that the parser object must be called `pp` for the autodoc system to locate the query. As long as the convention is followed, the content of the comments will be automatically scraped and included in the Inputs section of the documentation. (For instance: *Model::Solid::Linear::Isotropic*)

3.1.2 Header comments

General comments to document a class, or collection of methods, goes at the top of the associated header file. Use standard C++ comments (//). These comments will get scraped by the autodoc system, and will be formatted along with the rest of the inputs.

For example, consider the documentation for the linear isotropic material model class. The following leading comment is formatted at *Model::Solid::Linear::Isotropic*

```
//
// This model implements an isotropic linear elastic material.
// See `this link <https://en.wikipedia.org/wiki/Linear_elasticity#(An)isotropic_
↪(in)homogeneous_media>`_
// for more information about the theory.
//
// Free energy for a linear material is defined as
//
// .. math::
//
//     W(\nabla\mathbf{u}) =
//     \frac{1}{2}\nabla\mathbf{u}\cdot\mathbb{C}\,\nabla\mathbf{u}
//
// For an isotropic material, stress and strain are related through
//
// .. math::
//
//     \mathbb{C}_{ijkl} = \lambda \delta_{ij}\varepsilon_{kk} + 2\mu\varepsilon_{ij}
//
// where :math:`\lambda` and :math:`\mu` are the Lame constant and shear modulus,␣
↪respectively.
// Users can specify these either through (:code:`lame` and :code:`shear`)
// OR (:code:`lambda` and :code:`mu`) OR (:code:`E` and :code:`nu`).
//
// Class methods:
//
// #. :code:`Isotropic()`:
//    Basic constructor. Does nothing, and leaves all values initiated as NAN.
// #. :code:`Isotropic(Solid<Set::Sym::Isotropic> base)`
//    Basic constructor. Does nothing gut allows for inheritance.
// #. :code:`Isotropic(Set::Scalar a_mu, Set::Scalar a_lambda)`
//    BAD old-fashioned constructor. Do not use!
// #. :code:`~Isotropic()`
//    Simple destructor. Don't need to change it.
// #. :code:`void Define(Set::Scalar a_mu, Set::Scalar a_lambda)`
//    BAD old-fashioned way of doing things. Use :code:`Parse` instead.
// #. :code:`Set::Scalar W(const Set::Matrix & gradu) const override`
//    Returns elastic free energy density
// #. :code:`Set::Matrix DW(const Set::Matrix & gradu) const override`
//    Returns first derivative of free energy, the stress tensor
// #. :code:`Set::Matrix4<...> DDW(const Set::Matrix & ) const override`
//    Returns second derivative of free energy, the modulus tensor
// #. :code:`virtual void Print(std::ostream &out) const override`
//    Prints the modulus tensor object to output stream (usually the terminal)
// #. :code:`static Isotropic Random()`
```

(continues on next page)

(continued from previous page)

```
//    Static method that generates a random yet acceptable model.
// #. :code:`static Isotropic Zero()`
//    Static method that generates a "zero" element (so that there is no effect under␣
↪addition)
// #. :code:`static void Parse(Isotropic & value, IO::ParmParse & pp)`
//    Parser where all the IO occurs
//
//
```

3.1.3 ReStructuredText

All comments are formatted using restructuredtext markup. You can make use of this is if you like, but it is not required for good documentation.

3.1.4 Documentation generation

To generate this documentation, complete with the scraped markup, run

```
make docs
```

in the alamo root directory. The file `alamo/docs/requirements.txt` contains a list of the necessary packages. You can install them using pip. Once the documenation generation is complete, you can view it by

```
google-chrome docs/build/html/index.html
```

3.2 Autotest system

Alamo contains a complete regression testing system to ensure that established capabilities do not get lost during subsequent development. Regression tests are run automatically by github, and must all pass prior to merge into the development or master branches. For information on how to run the regression tests, see *Testing*.

3.2.1 Test requirements

Remember that your test will be run automatically on GitHub and by other users, so avoid creating long, memory-intensive tests. Instead, design your tests so that they run long enough to identify errors, but short enough so that they can be performed tractable. Make sure that your tests are not redundant with existing tests as well.

3.2.2 Creating a test

1. Create a subdirectory in the `./tests` with your test name, e.g. "MyTest" [`./tests/MyTest`]
2. Place your test input file in this directory and name it *input* [`./tests/input`].
3. At the top of the test input file, add

   ```
   #@ [sectionname]
   ```

where `sectionname` can be any unique handle name. The `#@` directive is ignored by alamo, but all comments beginning with it will be parsed by the autotest system.

4. Test your test by running

```
./scripts/runtests ./tests/MyTest
```

If alamo has been compiled in 3D, this should cause your test to run. The output of your test will be stored in

```
./tests/MyTest/output_YYYY-MM-DD_HH.MM.SS_hostname_sectionname
```

Once you have done this, your test will be executed along with all other regression tests by GitHub. If your test fails to complete - for instance, it segfaults or aborts abnormally - this will trigger a "failed test"

3.2.3 Input file directives

There are several directives that you can use for your inputs. These are written using the #@ comments at the top of the input file.

```
#@  [2D-serial-5levels]        | each [...] specifies a run of the test
#@  dim    = 2                 | specify two dimensions
#@  nprocs = 1                 | specify running in serial (the default)
#@  check  = false             | for this test we do not want to run the verification
#@
#@  [3D-parallel-4levels]      | another test
#@  dim    = 3                 | specify running in 3D (the default)
#@  nprocs = 4                 | specify run in parallel with 4 processors
#@  args   = amr.max_level=4   | specify an additional argument to be added to input file
#@  benchmark-beaker = 16.10   | timing test: benchmark-id record current average time to
#@  benchmark-statler = 11.36  |     run on platform "id". The autotest system will let␣
↪you
#@  benchmark-github = 22.75   |     know if future changes slow the test down.
#@  ignore = myargument        | tell alamo to ignore certain arguments
```

For additional examples, see the Tests section.

3.2.4 Test verification

Successfully running a test does not tell you much unless you know that the correct answer was produced. To automatically verify your test, add an executable file called `test` to the test directory:

```
./tests/MyTest/test
```

The test script must be executable, and can be written in a language of your choice, as long as you have the appropriate shebang at the top. There are two requirements on the test script:

1. It must take a single argument, the test directory name.
2. It produces a zero error code if the test passes, and a nonzero error code if the test fails.

It is up to you to make sure that the test accurately assesses the output. You can store reference data inside the test directory; e.g.

```
./tests/MyTest/reference/stress_xx.dat
./tests/MyTest/reference/stress_xy.dat
....
```

as long as the data files are reasonably small in size and, of course, are text-based. For example tests, see the existing tests in the repository.

CHAPTER

FOUR

TESTS

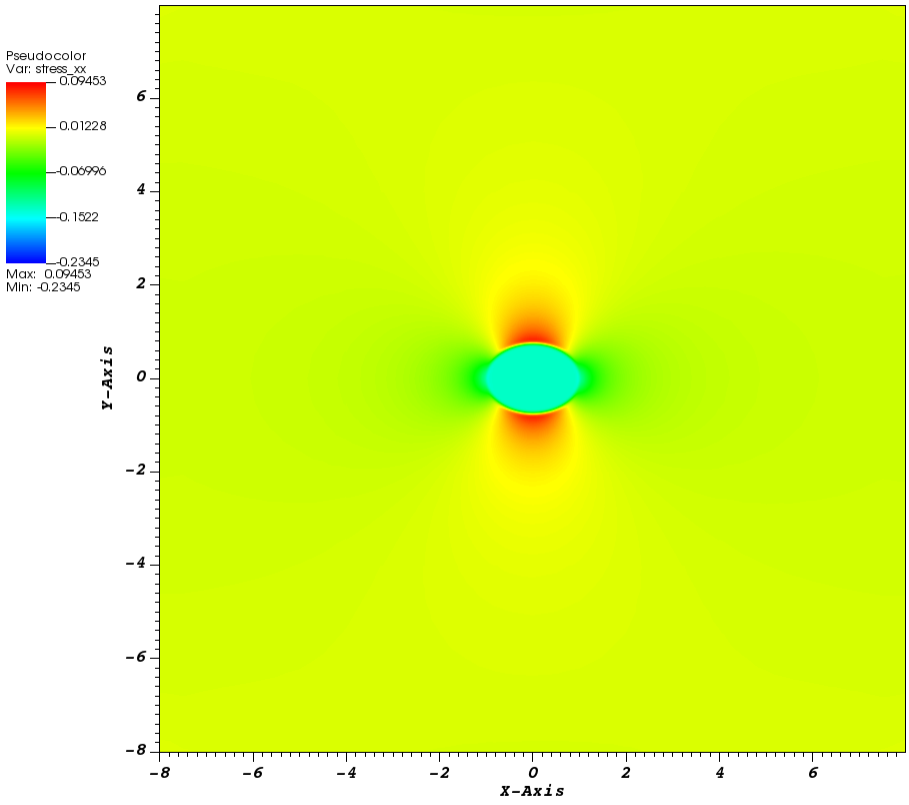
Status	Name	Sections	Dimension	Validation
warning	400GrainStampede2			
warning	Disconnection			
warning	Dynamics			
check_circle	*Eshelby*	3	2d 3d_rotation	verified
warning	EshelbyDynamics			
warning	Expression			
warning	Flame			
warning	Fracture2DDeflection			
check_circle	*HeatConduction*	2	2d 3d_rotation	verified
check_circle	*Inclusion*	1	2d	verified
check_circle	*PorousBedInterface*	1	2d	verified
check_circle	*PlateHole*	3	2d 3d_rotation	verified
check_circle	*RubberPlateHole*	1	2d	
check_circle	*RubberWithInclusion*	2	2d	verified
check_circle	*SCBSandwich*	2	2d	verified
check_circle	*Scratch*	2	2d	verified
check_circle	*Solid*	9	2d 3d_rotation	verified
check_circle	*Sphere*	1	2d	verified
warning	ThermoElastic			
check_circle	*TopOp*	1	2d	
check_circle	*TriTest*	6	2d 3d_rotation	verified
check_circle	*UniaxialTension*	6	2d 3d_rotation	verified
check_circle	*Voronoi*	2	2d	verified
check_circle	*VoronoiElastic*	2	2d	verified

4.1 Eshelby

This example demonstrates the Mechanics solver using the Eshelby inclusion problem. The example primarily tests the following Alamo capabilities

- *Integrator::Mechanics*
- *Model::Solid::Affine::Isotropic*

In 2D, the `sigma_xx` solution is:



In 3D, the solution is tested against the exact solution derived using Eshelby inclusion theory. For additional background see Runnels *et al*: https://doi.org/10.1016/j.jcp.2020.110065

4.1.1 [Eshelby] 2D-serial-5levels

2d	Two-dimensional
square	Serial
report_off	No testing script
play_circle	`./bin/alamo-2d-g++ tests/Eshelby/input`

4.1.2 [Eshelby] 3D-serial-4levels

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
timer	16.10s (beaker) 11.36s (statler) 22.75s (github)
play_circle	`./bin/alamo-3d-g++ tests/Eshelby/input amr.max_level="4"`

4.1.3 [Eshelby] 3D-parallel-5levels

3d_rotation	Three-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
timer	24.66s (beaker) 18.76s (statler)
play_circle	`mpiexec -np 4 ./bin/alamo-3d-g++ tests/Eshelby/input`

Listing 1: Input file (../../tests/Eshelby/input)

```
#@  [2D-serial-5levels]
#@  dim    = 2
#@  nprocs = 1
#@  check  = false
#@
#@  [3D-serial-4levels]
#@  dim    = 3
#@  nprocs = 1
#@  args   = amr.max_level=4
#@  benchmark-beaker = 16.10
#@  benchmark-statler = 11.36
#@  benchmark-github = 22.75
#@
#@  [3D-parallel-5levels]
#@  dim    = 3
#@  nprocs = 4
#@  benchmark-beaker = 24.66
#@  benchmark-statler = 18.76


alamo.program = mechanics
alamo.program.mechanics.model = affine.isotropic

plot_file                     = tests/Eshelby/output

# this is not a time integration, so do
# exactly one timestep and then quit
```

(continues on next page)

(continued from previous page)

```
timestep                    = 0.1
stop_time                    = 0.1

# amr parameters
amr.plot_int                     = 1
amr.max_level                     = 5
amr.n_cell                    = 32 32 32
amr.blocking_factor            = 8
amr.regrid_int                    = 1
amr.grid_eff                     = 1.0
amr.cell.all                = 1

# geometry
geometry.prob_lo            = -8 -8 -8
geometry.prob_hi            = 8 8 8
geometry.is_periodic            = 0 0 0

# ellipse configuration
ic.type        = ellipse
ic.ellipse.a   = 1.0 0.75 0.5  # ellipse radii
ic.ellipse.x0  = 0 0 0 # location of ellipse center
ic.ellipse.eps = 0.1 # diffuse boundary

# elastic moduli
nmodels = 2
model1.E = 210
model1.nu = 0.3
model1.F0  = 0.001 0 0 0 0.001 0 0 0 0.001 # eigenstrain
model2.E = 210
model2.nu = 0.3
model2.F0  = 0 0 0 0 0 0 0 0 0 # eigenstrain

solver.verbose = 3
solver.nriters = 1
solver.max_iter = 20
```

4.2 HeatConduction

This is the test for the *Integrator::HeatConduction* integrator. All BCs are Dirichlet. Low faces have constant nonzero temperature, high faces have constant zero temperature. Domain is initialized with zero temperature. Sample result below in 3D:

4.2.1 [HeatConduction] 2d-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/HeatConduction/input`

4.2.2 [HeatConduction] 3d-parallel

3d_rotation	Three-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
timer	7.66s (beaker)
play_circle	`mpiexec -np 4 ./bin/alamo-3d-g++ tests/HeatConduction/input`

Listing 2: Input file (../../tests/HeatConduction/input)

```
#@
#@ [2d-serial]
#@ dim=2
#@
#@ [3d-parallel]
#@ dim=3
#@ nprocs=4
#@ benchmark-beaker=7.66
#@

alamo.program = heat

plot_file     = tests/HeatConduction/output

# Simulation length
timestep = 0.0001
stop_time = 0.05

# AMR parameters
amr.plot_int = 10
amr.max_level = 3
amr.n_cell = 8 8 8
amr.blocking_factor = 1
amr.regrid_int = 10
amr.grid_eff = 1.0

# Specify geometry and unrefined mesh
geometry.prob_lo = 0 0 0
```

(continues on next page)

(continued from previous page)

```
geometry.prob_hi = 1 1 1
geometry.is_periodic= 0 0 0

# Criterion for mesh refinement
heat.alpha = 1.0
heat.refinement_threshold = 0.01

# Specify initial conditions
ic.type = sphere
ic.sphere.center = 0.5 0.5 0.5
ic.sphere.radius = 0.25
ic.sphere.inside = 0.0



# Boundary conditions
bc.temp.type.xhi = dirichlet
bc.temp.type.xlo = dirichlet
bc.temp.type.yhi = dirichlet
bc.temp.type.ylo = dirichlet
bc.temp.type.zhi = dirichlet
bc.temp.type.zlo = dirichlet
bc.temp.val.xhi  = 0.0
bc.temp.val.xlo  = 1.0
bc.temp.val.yhi  = 0.0
bc.temp.val.ylo  = 1.0
bc.temp.val.zhi  = 0.0
bc.temp.val.zlo  = 1.0
```

4.3 Inclusion

This demonstrates the evolution of an initially spherical inclusion, with anisotropic boundary energy, and with a volume constraint enforced using a Lagrange multiplier. See *Integrator::PhaseFieldMicrostructure* for additional information.

4.3.1 [Inclusion] 2d-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
timer	29.49s (beaker) 20.54s (statler)
play_circle	`./bin/alamo-2d-g++ tests/Inclusion/input`

Listing 3: Input file (../../tests/Inclusion/input)

```
#@
#@  [2d-serial]
#@  dim = 2
#@  check = true
#@  benchmark-beaker = 29.49
#@  benchmark-statler = 20.54
#@

alamo.program = microstructure

timestep = 0.01
stop_time = 2.0

plot_file = tests/Inclusion/output

amr.plot_dt = 0.1
amr.max_level = 2
amr.n_cell = 64 64 64
amr.blocking_factor = 8
amr.regrid_int = 10
amr.grid_eff = 1.0
amr.max_grid_size = 8

amr.thermo.plot_dt = 0.1

ic.type=sphere
ic.sphere.center = 4 4 4
ic.sphere.radius = 1

geometry.prob_lo = 0 0 0
geometry.prob_hi = 8 8 8
geometry.is_periodic= 0 0 0

bc.eta.type.xlo = neumann
bc.eta.type.xhi = neumann
bc.eta.type.ylo = neumann
bc.eta.type.yhi = neumann

pf.number_of_grains = 2
pf.M = 1.0
pf.mu = 10.0
pf.gamma = 1.0
pf.l_gb=0.1
pf.sigma0=0.075

anisotropy.on=1
anisotropy.timestep=0.0005
anisotropy.tstart= 1.
anisotropy.gb_type=sin
anisotropy.sin.sigma0=0.075
```

(continues on next page)

(continued from previous page)

```
anisotropy.sin.sigma1=0.07
anisotropy.sin.theta0=45
anisotropy.beta= 0.00001

elastic.on = 0

lagrange.on = 1
lagrange.lambda = 2.0
lagrange.tstart = 1.0
lagrange.vol0 = 4.0
```

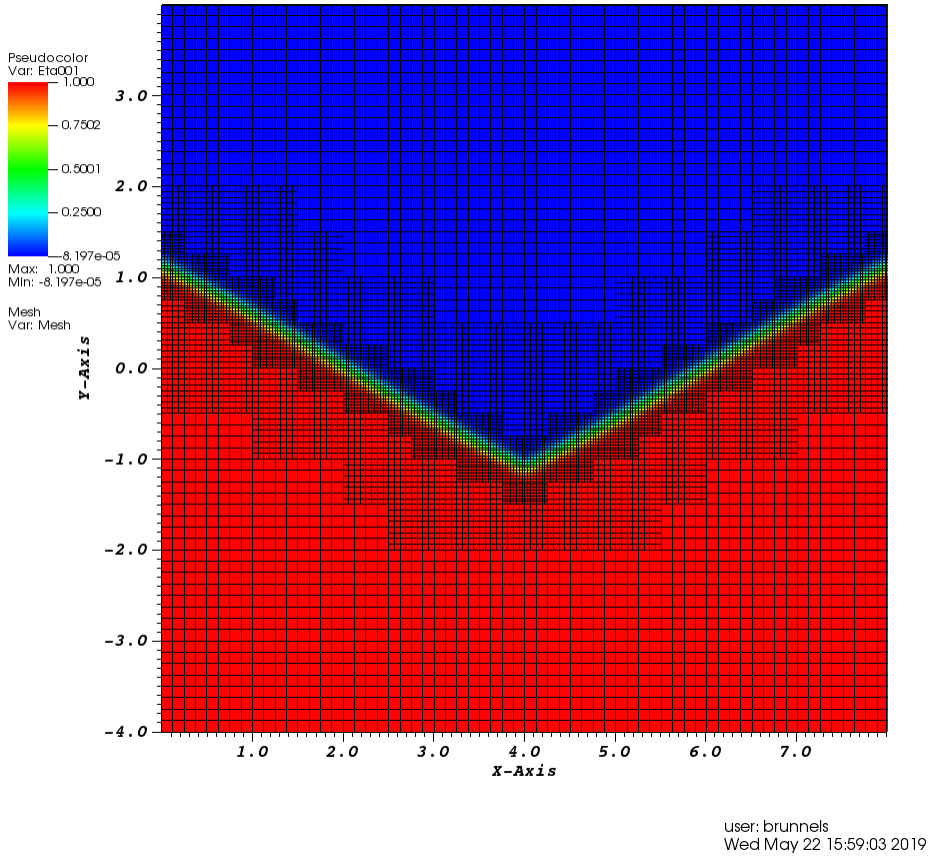
4.4 PerturbedInterface

This example simluates the evolution of a perturbed interface with strongly anisotropic grain boundary energy. For background see Ribot *et al*: https://doi.org/10.1088/1361-651X/ab47a0

This problem tests the following

- *Integrator::PhaseFieldMicrostructure*
- *Model::Interface::GB*

Example output:



Notice that the sharp corner in the equilibrium shape is the result of strong anisotropy.

4.4.1 [PerturbedInterface] perturbed-interface

2d	Two-dimensional
square	Serial
verified	Testing script present
timer	19.87s (beaker) 12.53s (statler) 18.75s (github)
play_circle	`./bin/alamo-2d-g++ tests/PerturbedInterface/input`

Listing 4: Input file (../../tests/PerturbedInterface/input)

```
#@
#@  [perturbed-interface]
#@  dim = 2
#@  check = true
#@  benchmark-beaker = 19.87
#@  benchmark-statler = 12.53
#@  benchmark-github = 18.75
#@

alamo.program = microstructure

timestep = 0.01
stop_time = 2.0

plot_file = tests/PerturbedInterface/output

amr.plot_dt = 0.1
amr.max_level = 2
amr.n_cell = 64 64 64
amr.blocking_factor = 8
amr.regrid_int = 10
amr.grid_eff = 1.0
amr.max_grid_size = 8

#amr.thermo.plot_dt = 0.1

ic.type=perturbed_interface
ic.wave_numbers=4
ic.wave_amplitudes=0.5
ic.normal = y
ic.mollifier = dirac

geometry.prob_lo = 0 -4 0
geometry.prob_hi = 8 4 8
geometry.is_periodic= 0 0 0

bc.eta.type.xlo = neumann
```

(continues on next page)

(continued from previous page)

```
bc.eta.type.xhi = neumann
bc.eta.type.ylo = dirichlet
bc.eta.type.yhi = dirichlet
bc.eta.val.ylo  = 1.0 0.0
bc.eta.val.yhi  = 0.0 1.0

pf.number_of_grains = 2
pf.M = 1.0
pf.mu = 10.0
pf.gamma = 1.0
pf.l_gb=0.1
pf.sigma0=0.075

anisotropy.on=1
anisotropy.timestep=0.001
anisotropy.tstart= 1.
anisotropy.gb_type=sin
anisotropy.sin.sigma0=0.075
anisotropy.sin.sigma1=0.07
anisotropy.sin.theta0=45
anisotropy.beta= 0.00001
#anisotropy.damp=1.0

elastic.on = 0
```

4.5 PlateHole

4.5.1 [PlateHole] 2D-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/PlateHole/input`

4.5.2 [PlateHole] 3D-parallel

3d_rotation	Three-dimensional
grid_view	Parallel (4 procs)
report_off	No testing script
timer	13.52s (beaker) 11.48s (statler)
play_circle	`mpiexec -np 4 ./bin/alamo-3d-g++ tests/PlateHole/input`

4.5.3 [PlateHole] 3D-serial

3d_rotation	Three-dimensional
square	Serial
report_off	No testing script
timer	42.00s (beaker)
play_circle	./bin/alamo-3d-g++ tests/PlateHole/input

Listing 5: Input file (../../tests/PlateHole/input)

```
#@  [2D-serial]
#@  dim    = 2
#@  nprocs = 1
#@  check  = true
#@
#@  [3D-parallel]
#@  dim    = 3
#@  nprocs = 4
#@  check  = false
#@  benchmark-beaker = 13.52
#@  benchmark-statler = 11.48
#@
#@  [3D-serial]
#@  dim    = 3
#@  nprocs = 1
#@  check  = false
#@  benchmark-beaker = 42.00




alamo.program = mechanics
alamo.program.mechanics.model = linear.isotropic

plot_file                     = tests/PlateHole/output

# this is not a time integration, so do
# exactly one timestep and then quit
timestep                    = 0.1
stop_time                    = 0.1
amr.plot_int                    = 1

amr.blocking_factor            = 8
amr.regrid_int                    = -1
amr.grid_eff                    = 1.0
amr.node.all                = 1
amr.cell.all                = 1

# grid
```

(continues on next page)

(continued from previous page)

```
amr.max_level                    = 5 #8
amr.n_cell                     = 32 32 32

# geometry
geometry.prob_lo            = -16 -16 -16
geometry.prob_hi            = 16 16 16
geometry.is_periodic            = 0 0 0

# ellipse configuration
psi.ic.type        = ellipse
psi.ic.ellipse.a   = 1.0 1.0 1.0  # ellipse radius
psi.ic.ellipse.x0  = 0 0 0 # location of ellipse center
psi.ic.ellipse.eps = 0.2
psi.ic.ellipse.invert = 1

# elastic moduli
nmodels = 1
model1.E = 1.0
model1.nu = 0.3
model1.planestress = 1

solver.verbose = 3
solver.nriters = 1
solver.max_iter = 20
solver.bottom_solver = smoother
solver.tol_abs = 1E-16 # This is very important for near-singular problems!!!

print_residual = 1

bc.type = tension_test
bc.tension_test.type = uniaxial_stress
bc.tension_test.disp=0.01

elasticop.small=1E-16
```

4.6 RubberPlateHole

This example demonstrates the Finite Kinematics newton solver in conjunction with the near-singular void field ψ to simulate a compliant material with a hole in the middle.

It also demonstrates the use of the *IC::Expression* class with the `variable` functionality.

4.6.1 [RubberPlateHole] serial-2d

2d	Two-dimensional
square	Serial
report_off	No testing script
play_circle	`./bin/alamo-2d-g++ tests/RubberPlateHole/input`

Listing 6: Input file (../../tests/RubberPlateHole/input)

```
#@
#@ [serial-2d]
#@ dim=2
#@ check=false
#@

alamo.program = mechanics
alamo.program.mechanics.model = elastic.neohookean
plot_file = output
type = static
timestep = 0.1
stop_time = 1.0

amr.plot_int = 1
amr.max_level = 2
amr.n_cell = 16 16 16
amr.blocking_factor = 2
amr.thermo.int = 1
amr.thermo.plot_int = 1
amr.node.all = 1
amr.cell.all = 1

geometry.prob_lo = -1 -1 -1
geometry.prob_hi =  1  1  1

ic.type = ellipse
ic.ellipse.a = 0.25 0.25 0.25
ic.ellipse.x0 = 0.5 0.5 0.5
ic.ellipse.eps = 0.05

nmodels = 1
model1.mu = 30
model1.kappa = 60

psi.ic.type=expression
psi.ic.expression.constant.eps = 0.05
psi.ic.expression.constant.R   = 0.25
psi.ic.expression.region0 = "0.5 + 0.5*tanh((x^2 + y^2 - R)/eps)"

solver.verbose = 2
```

(continues on next page)

(continued from previous page)

```
solver.max_iter = 150
solver.nriters = 1000
solver.nrtolerance = 1E-5

ref_threshold = 100

bc.type = tension_test
bc.tension_test.type = uniaxial_stress
bc.tension_test.disp = (0,1:0,0.5)

solver.dump_on_fail = 1
amrex.signal_handling = 0
amrex.throw_exception = 1
```

4.7 RubberWithInclusion

This example demonstrates the Finite Kinematics newton solver for a composite consisting of a compliant rubber-like material and a (near) rigid metal or ceramic-like material. The difference in elastic moduli is 10x. The low faces (xlo, ylo, zlo) are homogeneous dirichlet in the normal direction and naumann in the lateral directions. The three-dimensional test yields the following:

For regression testing purposes, only the two-dimensional case is considered for efficiency. In two dimensions, the distortion of the mesh is more severe, which usually means that additional Newton iterations are required.

4.7.1 [RubberWithInclusion] serial-2d

2d	Two-dimensional
square	Serial
verified	Testing script present
timer	13.8s (beaker)
play_circle	`./bin/alamo-2d-g++ tests/RubberWithInclusion/input`

4.7.2 [RubberWithInclusion] parallel-2d

2d	Two-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
timer	6.6s (beaker)
play_circle	`mpiexec -np 4 ./bin/alamo-2d-g++ tests/RubberWithInclusion/input`

Listing 7: Input file (../../tests/RubberWithInclusion/input)

```
#@
#@ [serial-2d]
#@ dim=2
#@ benchmark-beaker=13.8
#@ check-file=reference/thermo.dat
#@
#@ [parallel-2d]
#@ dim=2
#@ nprocs=4
#@ benchmark-beaker=6.6
#@ check-file=reference/thermo.dat
#@

alamo.program = mechanics
alamo.program.mechanics.model = elastic.neohookean
plot_file = tests/RubberWithInclusion/output
type = static
timestep = 0.1
stop_time = 1.0

amr.plot_int = 1
amr.max_level = 2
amr.n_cell = 16 16 16
amr.blocking_factor = 2
amr.thermo.int = 1
amr.thermo.plot_int = 1

geometry.prob_lo = 0 0 0
geometry.prob_hi = 1 1 1

ic.type = ellipse
ic.ellipse.a = 0.25 0.25 0.25
ic.ellipse.x0 = 0.5 0.5 0.5
ic.ellipse.eps = 0.05

nmodels = 2
model1.mu = 30
model1.kappa = 60
model2.mu = 3.0
model2.kappa = 6.0

solver.verbose = 3
solver.max_iter = 150
solver.nriters = 1000
solver.nrtolerance = 1E-5


ref_threshold = 100

bc.type = tension_test
```

(continues on next page)

(continued from previous page)

```
bc.tension_test.type = uniaxial_stress
bc.tension_test.disp = (0,1:0,0.5)

solver.dump_on_fail = 1
amrex.signal_handling = 0
amrex.throw_exception = 1
```

4.8 SCPSandwich

4.8.1 [SCPSandwich] serial

2d	Two-dimensional
square	Serial
report_off	No testing script
play_circle	`./bin/alamo-2d-g++ tests/SCPSandwich/input stop_time="0.01"`

4.8.2 [SCPSandwich] parallel

2d	Two-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
play_circle	`mpiexec -np 4 ./bin/alamo-2d-g++ tests/SCPSandwich/input`

Listing 8: Input file (../../tests/SCPSandwich/input)

```
#@
#@  [serial]
#@  dim = 2
#@  check = false
#@  args = stop_time=0.01
#@
#@  [parallel]
#@  dim = 2
#@  nprocs = 4
#@  check = true
#@

alamo.program = flame

# AMR parameters
plot_file = tests/SCPSandwich/output
```

(continues on next page)

(continued from previous page)

```
amr.plot_dt = 0.01
amr.max_level = 7
amr.n_cell = 16 2 2
amr.blocking_factor = 2
amr.regrid_int = 100
amr.base_regrid_int = 20
amr.grid_eff = 0.7

# Geometry
geometry.prob_lo = 0.0 -0.25 -0.1 # [mm]
geometry.prob_hi = 4.0 0.25 0.1 # [mm]
geometry.is_periodic = 0 1 1

# Timestep and duration
timestep = 0.00005 # [s]
stop_time = 0.2 # [s]

# Phase field params
pf.eps = 0.0005 # [mm]
pf.lambda = 0.001
pf.kappa = 1.0
pf.gamma = 0.02726
pf.w1 = 1.0
pf.w12 = 2.0
pf.w0 = 0.0

# AP / HTPB regression rate params
pf.r_ap = 1.222 # [ mm / s ]
pf.n_ap = 1.042 # [ 1 ]
pf.r_htpb = 0.1 # [ mm / s]
pf.r_comb = 10.0 # [ mm / s]
pf.n_htpb = 0.0 # [ 1 ]
pf.n_comb = 0.0 # [ 1 ]
# Pressure used in regression rate calc
pf.P = 0.2

# Phase field IC
eta.ic.type = constant
eta.ic.constant.value = 1

# Species field params
phi.ic.type = laminate
phi.ic.laminate.number_of_inclusions = 1
phi.ic.laminate.center  = 0.0 0.0 0.0
phi.ic.laminate.thickness = 0.1
phi.ic.laminate.orientation = 0 1
phi.ic.laminate.eps = 0.015
phi.ic.laminate.singlefab = 1
phi.ic.laminate.invert = 1

# Boundary conditions on eta
pf.eta.bc.type.xlo = dirichlet
```

(continues on next page)

(continued from previous page)

```
pf.eta.bc.type.xhi = dirichlet
pf.eta.bc.type.ylo = neumann
pf.eta.bc.type.yhi = neumann
pf.eta.bc.val.xlo = 0.0
pf.eta.bc.val.xhi = 1.0
pf.eta.bc.val.ylo = 0.0
pf.eta.bc.val.yhi = 0.0
```

4.9 Scratch

This input file simulates the effect of a moving point load applied over the surface of the domain. The J2 plastic material accumulates residual stress left by the application of the point load. This test exercises the following capabilities:

- *Integrator::Mechanics*
- *BC::Operator::Elastic::Expression*
- *Model::Solid::Affine::J2*

To generate the movie here, run in 3D. A 3D test is not included in the regression test suite due to the extensive time required.

4.9.1 [Scratch] 2d-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
timer	16.9s (beaker)
play_circle	`./bin/alamo-2d-g++ tests/Scratch/input`

4.9.2 [Scratch] 2d-parallel

2d	Two-dimensional
grid_view	Parallel (2 procs)
verified	Testing script present
timer	12.49s (beaker)
play_circle	`mpiexec -np 2 ./bin/alamo-2d-g++ tests/Scratch/input`

Listing 9: Input file (../../tests/Scratch/input)

```
#@
#@ [2d-serial]
#@ dim=2
#@ benchmark-beaker=16.9
#@
#@ [2d-parallel]
#@ dim=2
#@ nprocs=2
#@ benchmark-beaker=12.49
#@

alamo.program                = mechanics
alamo.program.mechanics.model = affine.j2
plot_file = tests/Scratch/output

timestep                     = 0.05
stop_time                     = 6.0

# amr parameters
amr.plot_int                      = 10
amr.max_level                      = 3
amr.n_cell                       = 16 16 16
amr.blocking_factor               = 4
amr.regrid_int                       = 1
amr.grid_eff                      = 1.0
amr.node.any = 0
amr.cell.all = 1

# geometry
geometry.prob_lo             = -8 -8 -8
geometry.prob_hi             = 8 8 8
geometry.is_periodic             = 0 0 0

# elastic moduli
nmodels = 1
model1.E = 210
model1.nu = 0.3
model1.sigma0=0.2

# solver parameters
solver.verbose = 3
solver.nriters = 1
solver.max_iter = 40

# Elastic refinement criterion
ref_threshold = 0.0005

# Expression-based boundary condition
bc.type = expression
bc.expression.type.yhi = trac trac trac
bc.expression.val.yhi = "0.0" "-exp(-((x-t)*(x-t)+z*z)/0.4)" "0.0"
```

4.10 Solid

4.10.1 [Solid] linear-isotropic

3d_rotation	Three-dimensional
square	Serial
report_off	No testing script
play_circle	`./bin/alamo-3d-g++ tests/Solid/input`

4.10.2 [Solid] linear-cubic

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-3d-g++ tests/Solid/input alamo.program.mechanics.model="linear.cubic" model1.C11="168.3" model1.C12="1.221" model1.C44="0.757" ignore="model1.E model1.nu"`

4.10.3 [Solid] affine-cubic

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-3d-g++ tests/Solid/input alamo.program.mechanics.model="affine.cubic" model1.C11="168.3" model1.C12="1.221" model1.C44="0.757" model1.F0="0.001 0 0 0 0.001 0 0 0 0.001" ignore="model1.E model1.nu"`

4.10.4 [Solid] affine-hexagonal

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-3d-g++ tests/Solid/input alamo.program.mechanics.model="affine.hexagonal" model1.C11="0.597" model1.C12="0.262" model1.C13="0.217" model1.C33="0.617" model1.C44="0.164" model1.F0="0.001 0 0 0 0.001 0 0 0 0.001" ignore="model1.E model1.nu"`

4.10.5 [Solid] j2

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
timer	10.42s (beaker) 7.02s (statler) 11.09s (github)
play_circle	./bin/alamo-3d-g++ tests/Solid/input timestep="0.001" alamo.program.mechanics.model="affine.j2" model1.E="210" model1.nu="0.3" model1.sigma0="0.2" bc.tension_test.disp="(0,0.25,0.75,1.0:0,0.002,-0.002,0)"

4.10.6 [Solid] affine-isotropic

3d_rotation	Three-dimensional
square	Serial
report_off	No testing script
play_circle	./bin/alamo-3d-g++ tests/Solid/input alamo.program.mechanics.model="affine.isotropic" model1.F0="0.001 0 0 0 0.001 0 0 0 0.001" model1.E="210" model1.nu="0.3"

4.10.7 [Solid] neo-hookean

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
timer	10.73s (beaker) 8.17s (statler) 11.35s (github)
play_circle	./bin/alamo-3d-g++ tests/Solid/input timestep="0.01" alamo.program.mechanics.model="elastic.neohookean" model1.mu="3.0" model1.kappa="6.5" bc.tension_test.disp="(0,1:0,1)" solver.nriters="10" ignore="model1.E model1.nu"

4.10.8 [Solid] neo-hookean-2d

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	./bin/alamo-2d-g++ tests/Solid/input timestep="0.01" alamo.program.mechanics.model="elastic.neohookean" model1.mu="3.0" model1.kappa="6.5" bc.tension_test.disp="(0,1:0,1)" solver.nriters="10" ignore="model1.E model1.nu"

4.10.9 [Solid] pseudolinear-cubic

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
play_circle	./bin/alamo-3d-g++ tests/Solid/input alamo.program.mechanics.model="elastic.pseudolinearcubic" model1.C11="168.3" model1.C12="1.221" model1.C44="0.757" ignore="model1.E model1.nu"

Listing 10: Input file (../../tests/Solid/input)

```
#@
#@  [linear-isotropic]
#@  dim = 3
#@  check = false
#@
#@  [linear-cubic]
#@  dim = 3
#@  check-file = reference/linear-cubic.dat
#@  args = alamo.program.mechanics.model=linear.cubic
#@  args = model1.C11 = 168.3
#@  args = model1.C12 = 1.221
#@  args = model1.C44 = 0.757
#@  ignore = model1.E model1.nu
#@
#@  [affine-cubic]
#@  dim = 3
#@  check-file = reference/affine-cubic.dat
#@  args = alamo.program.mechanics.model=affine.cubic
#@  args = model1.C11 = 168.3
#@  args = model1.C12 = 1.221
#@  args = model1.C44 = 0.757
#@  args = model1.F0=0.001 0 0 0 0.001 0 0 0 0.001
#@  ignore = model1.E model1.nu
#@
#@
#@  [affine-hexagonal]
#@  dim = 3
#@  check-file = reference/affine-hexagonal.dat
#@  args = alamo.program.mechanics.model=affine.hexagonal
#@  args = model1.C11 = 0.597
#@  args = model1.C12 = 0.262
#@  args = model1.C13 = 0.217
#@  args = model1.C33 = 0.617
#@  args = model1.C44 = 0.164
#@  args = model1.F0=0.001 0 0 0 0.001 0 0 0 0.001
#@  ignore = model1.E model1.nu
#@
#@  [j2]
#@  dim = 3
```

(continues on next page)

(continued from previous page)

```
#@  check-file = reference/j2.dat
#@  args = timestep = 0.001
#@  args = alamo.program.mechanics.model=affine.j2
#@  args = model1.E=210
#@  args = model1.nu=0.3
#@  args = model1.sigma0=0.2
#@  args = bc.tension_test.disp = (0,0.25,0.75,1.0:0,0.002,-0.002,0)
#@  benchmark-beaker = 10.42
#@  benchmark-statler = 7.02
#@  benchmark-github = 11.09
#@
#@  [affine-isotropic]
#@  dim = 3
#@  check = false
#@  args = alamo.program.mechanics.model=affine.isotropic
#@  args = model1.F0=0.001 0 0 0 0.001 0 0 0 0.001
#@  args = model1.E=210
#@  args = model1.nu=0.3
#@
#@  [neo-hookean]
#@  dim=3
#@  check-file = reference/neo-hookean.dat
#@  args = timestep=0.01
#@  args = alamo.program.mechanics.model=elastic.neohookean
#@  args = model1.mu=3.0
#@  args = model1.kappa=6.5
#@  args = bc.tension_test.disp=(0,1:0,1)
#@  args = solver.nriters=10
#@  ignore = model1.E model1.nu
#@  benchmark-beaker = 10.73
#@  benchmark-statler = 8.17
#@  benchmark-github = 11.35
#@
#@  [neo-hookean-2d]
#@  dim=2
#@  check-file = reference/neo-hookean-2d.dat
#@  args = timestep=0.01
#@  args = alamo.program.mechanics.model=elastic.neohookean
#@  args = model1.mu=3.0
#@  args = model1.kappa=6.5
#@  args = bc.tension_test.disp=(0,1:0,1)
#@  args = solver.nriters=10
#@  ignore = model1.E model1.nu
#@
#@  [pseudolinear-cubic]
#@  dim = 3
#@  check-file = reference/pseudolinear-cubic.dat
#@  args = alamo.program.mechanics.model=elastic.pseudolinearcubic
#@  args = model1.C11=168.3
#@  args = model1.C12=1.221
#@  args = model1.C44=0.757
#@  ignore = model1.E model1.nu
```

(continues on next page)

(continued from previous page)

```
#@

alamo.program = mechanics


plot_file                     = tests/Solid/output

type=static

timestep                     = 0.01
stop_time                     = 1.0

# amr parameters
amr.plot_dt                      = 0.1
amr.max_level                      = 0
amr.n_cell                     = 4 4 4
amr.blocking_factor          = 2

amr.thermo.int = 1
amr.thermo.plot_int = 1

# geometry
geometry.prob_lo             = 0 0 0
geometry.prob_hi             = 1 1 1

nmodels = 1
model1.E=210
model1.nu=0.3

solver.verbose = 3
solver.nriters = 1
solver.max_iter = 30 #30

bc.type = tension_test
bc.tension_test.type = uniaxial_stress
bc.tension_test.disp=(0,1:0,0.01)
```

4.11 Suture

4.11.1 [Suture] 2D-serial-4levels

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/Suture/input`

Listing 11: Input file (../../tests/Suture/input)

```
#@  [2D-serial-4levels]
#@  dim    = 2
#@  nprocs = 1

alamo.program               = mechanics
alamo.program.mechanics.model = affine.isotropic
plot_file                    = tests/Suture/output

# this is not a time integration, so do
# exactly one timestep and then quit
timestep                    = 0.1
stop_time                    = 0.1

# amr parameters
amr.plot_int                     = 1
amr.max_level                     = 4
amr.n_cell                     = 32 32 32
amr.blocking_factor             = 4
amr.regrid_int                     = 1
amr.grid_eff                     = 1.0
amr.cell.all = 1

# geometry
geometry.prob_lo            = -8 -8 -8
geometry.prob_hi            = 8 8 8
geometry.is_periodic            = 0 0 0


ic.type         = bmp
ic.bmp.filename = tests/Suture/interface-blur.bmp
ic.bmp.fit      = fitwidth
ic.bmp.channel  = g

#Aluminum Oxide
nmodels = 2
model1.E = 370 #GPa
model1.nu = 0.22
model1.F0 = -0.001 0 0 0 -0.001 0 0 0 -0.001
#Nickel
model2.E = 207 #GPa
model2.nu = 0.31
model2.F0 = 0.001 0 0 0 0.001 0 0 0 0.001

solver.verbose = 3
solver.nriters = 1
solver.fixed_iter=2000
## Sliders on all walls
bc.type = constant
bc.constant.type.ylo    = trac disp
bc.constant.type.xloylo = disp disp
```

(continues on next page)

(continued from previous page)

```
bc.constant.type.xhiylo = disp disp
bc.constant.type.xlo    = disp trac
bc.constant.type.xhi    = disp trac
bc.constant.type.xloyhi = disp disp
bc.constant.type.xhiyhi = disp disp
bc.constant.type.yhi    = trac disp
```

4.12 TopOp

4.12.1 [TopOp] serial

2d	Two-dimensional
square	Serial
report_off	No testing script
play_circle	./bin/alamo-2d-g++ tests/TopOp/input stop_time="1.0"

Listing 12: Input file (../../tests/TopOp/input)

```
#@
#@ [serial]
#@ dim = 2
#@ args=stop_time=1.0
#@ check=false
#@

alamo.program               = topop
plot_file                    = tests/TopOp/output

timestep                    = 0.1
stop_time                    = 20.0

# amr parameters
amr.plot_int                    = 1 #10000
amr.max_level                    = 2
amr.n_cell                    = 64 32 32
amr.blocking_factor             = 4
amr.regrid_int                     = 1
amr.grid_eff                    = 1.0
amr.node.all = 1
amr.thermo.plot_dt = 0.1

# geometry
geometry.prob_lo            = 0 0 0
geometry.prob_hi            = 1.8 1 1
geometry.is_periodic             = 0 0 0
```

(continues on next page)

(continued from previous page)

```
#
# Isotropic elastic model
#
model.E = 3.70
model.nu = 0.22

#
# Topology optimization parameters
#
alpha = 200.
beta = 0.05
gamma = 50000.0
L.str = 0.005
volume0 = 0.5
lambda.str = 400.0

#
# BCs for psi variable
#
psi.ic.type = constant
psi.ic.constant.value = 1.0
psi.bc.type.xhi = NEUMANN
psi.bc.type.xlo = NEUMANN
psi.bc.type.yhi = NEUMANN
psi.bc.type.ylo = NEUMANN


#### Canonical test case: cantilever with load at tip
#bc.type = expression
#bc.expression.type.xloylo = disp disp
#bc.expression.type.xlo    = disp disp
#bc.expression.type.xloyhi = disp disp
#bc.expression.type.ylo    = trac trac
#bc.expression.type.yhi    = trac trac
#bc.expression.type.xhi    = trac trac
#bc.expression.type.xhiylo = trac trac
#bc.expression.type.xhiyhi = trac trac
#bc.expression.val.yhi = "0.0" "0.0"
#bc.expression.val.xhiylo = "0.0" "0.0"
#bc.expression.val.xhiyhi = "0.0" "0.0"
#bc.expression.val.ylo = "0.0" "0"
#bc.expression.val.xhi = "0.0" "-0.1 * (y > 0.45) * (y < 0.55)"
#L.str = 0.01

### Canonical test case: cantilever with full load at tip
bc.type = constant
bc.constant.type.xloylo = disp disp
bc.constant.type.xlo    = disp disp
bc.constant.type.xloyhi = disp disp
bc.constant.type.ylo    = trac trac
bc.constant.type.yhi    = trac trac
```

(continues on next page)

(continued from previous page)

```
bc.constant.type.xhi    = trac trac
bc.constant.type.xhiylo = trac trac
bc.constant.type.xhiyhi = trac trac
bc.constant.val.xhi = 0.0 -0.1


#
# Solver inputs
#
elasticop.small=0.01
solver.bottom_solver = smoother
solver.tol_abs = 1E-16 # This is very important for near-singular problems!!!
solver.verbose = 3
solver.nriters = 1
solver.fixed_iter=2000

#
# Mechanics inputs
#
interval = 1
print_residual=1
eta_ref_threshold = 0.05
elastic_ref_threshold = 100000
zero_out_displacement = 1
```

4.13 TrigTest

4.13.1 [TrigTest] 2D-1AMR-Levels-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/TrigTest/input amr.max_level="0"`

4.13.2 [TrigTest] 2D-2AMRLevels-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/TrigTest/input amr.max_level="1"`

4.13.3 [TrigTest] 2D-3AMRLevels-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/TrigTest/input`

4.13.4 [TrigTest] 2D-3AMRLevels-parallel

2d	Two-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
play_circle	`mpiexec -np 4 ./bin/alamo-2d-g++ tests/TrigTest/input`

4.13.5 [TrigTest] 3D-1AMRLevels-serial

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-3d-g++ tests/TrigTest/input`

4.13.6 [TrigTest] 3D-3AMRLevels-parallel

3d_rotation	Three-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
play_circle	`mpiexec -np 4 ./bin/alamo-3d-g++ tests/TrigTest/input`

Listing 13: Input file (../../tests/TrigTest/input)

```
#@  [2D-1AMR-Levels-serial]
#@  dim = 2
#@  nprocs = 1
#@  args = amr.max_level=0
#@
#@  [2D-2AMRLevels-serial]
#@  dim = 2
```

(continues on next page)

(continued from previous page)

```
#@  nprocs = 1
#@  args = amr.max_level=1
#@
#@  [2D-3AMRLevels-serial]
#@  dim = 2
#@  nprocs = 1
#@
#@  [2D-3AMRLevels-parallel]
#@  dim = 2
#@  nprocs = 4
#@
#@  [3D-1AMRLevels-serial]
#@  dim = 3
#@  nprocs = 1
#@
#@  [3D-3AMRLevels-parallel]
#@  dim = 3
#@  nprocs = 4

alamo.program                 = mechanics
alamo.program.mechanics.model = linear.laplacian
plot_file                      = tests/TrigTest/output

# this is not a time integration, so do
# exactly one timestep and then quit
timestep                      = 0.1
stop_time                      = 0.1

# amr parameters
amr.plot_int                     = 1
amr.max_level                     = 2
amr.n_cell                      = 32 32 32
amr.blocking_factor               = 4
amr.regrid_int                      = 1
amr.grid_eff                     = 1.0
amr.cell.all                  = 1

# use an explicit mesh (i.e. no adaptive meshing)
explicitmesh.on               = 1
explicitmesh.lo1              = 16 16 16
explicitmesh.hi1              = 47 47 47
explicitmesh.lo2              = 48 48 48
explicitmesh.hi2              = 79 79 79

# geometry
geometry.prob_lo              = 0 0 0
geometry.prob_hi              = 1 1 1
geometry.is_periodic             = 0 0 0

rhs.type = trig
rhs.trig.nr = 0 0 0
rhs.trig.ni = 1 1 1
```

(continues on next page)

(continued from previous page)

```
rhs.dim = 1
rhs.alpha = 1

solver.verbose = 3
solver.nriters = 1
solver.max_iter = 20

### UNIAXIAL TENSION ###
elastic.bc.type = constant
```

4.14 UniaxialTension

4.14.1 [UniaxialTension] 2D-serial-1level

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/UniaxialTension/input amr.max_level="0"`

4.14.2 [UniaxialTension] 2D-serial-2levels

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/UniaxialTension/input amr.max_level="1"`

4.14.3 [UniaxialTension] 2D-serial-3levels

2d	Two-dimensional
square	Serial
verified	Testing script present
play_circle	`./bin/alamo-2d-g++ tests/UniaxialTension/input`

4.14.4 [UniaxialTension] 2D-parallel-3levels

2d	Two-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
play_circle	`mpiexec -np 4 ./bin/alamo-2d-g++ tests/UniaxialTension/input`

4.14.5 [UniaxialTension] 3D-serial-3levels

3d_rotation	Three-dimensional
square	Serial
verified	Testing script present
timer	8.29s (github)
play_circle	`./bin/alamo-3d-g++ tests/UniaxialTension/input`

4.14.6 [UniaxialTension] 3D-parallel-3levels

3d_rotation	Three-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
play_circle	`mpiexec -np 4 ./bin/alamo-3d-g++ tests/UniaxialTension/input`

Listing 14: Input file (../../tests/UniaxialTension/input)

```
#@  [2D-serial-1level]
#@  dim=2
#@  nprocs=1
#@  args=amr.max_level=0
#@
#@  [2D-serial-2levels]
#@  dim=2
#@  nprocs=1
#@  args=amr.max_level=1
#@
#@  [2D-serial-3levels]
#@  dim=2
#@  nprocs=1
#@
#@  [2D-parallel-3levels]
#@  dim=2
#@  nprocs=4
```

(continues on next page)

(continued from previous page)

```
#@
#@  [3D-serial-3levels]
#@  dim=3
#@  nprocs=1
#@  benchmark-github = 8.29
#@
#@  [3D-parallel-3levels]
#@  dim=3
#@  nprocs=4
#@

alamo.program                 = mechanics
alamo.program.mechanics.model = linear.isotropic
plot_file                      = tests/UniaxialTension/output

# this is not a time integration, so do
# exactly one timestep and then quit
timestep                      = 0.1
stop_time                      = 0.1

# amr parameters
amr.plot_int                       = 1
amr.max_level                       = 2
amr.n_cell                        = 32 32 32
amr.blocking_factor                = 4
amr.regrid_int                        = 1
amr.grid_eff                       = 1.0
amr.cell.all                  = 1

# use an explicit mesh (i.e. no adaptive meshing)
explicitmesh.on               = 1
explicitmesh.lo1              = 16 16 16
explicitmesh.hi1              = 47 47 47
explicitmesh.lo2              = 48 48 48
explicitmesh.hi2              = 79 79 79

# geometry
geometry.prob_lo              = -8 -8 -8
geometry.prob_hi              = 8 8 8
geometry.is_periodic              = 0 0 0

# elastic moduli
model1.planestress = true
model1.E = 210
model1.nu = 0.3

solver.verbose = 3
solver.nriters = 1
solver.max_iter = 50

### UNIAXIAL TENSION ###
bc.type = constant
```

(continues on next page)

(continued from previous page)

```
### --- Faces
bc.constant.type.xlo = disp trac trac
bc.constant.type.xhi = disp trac trac
bc.constant.val.xhi = 0.1 0.0 0.0
bc.constant.type.ylo = trac disp trac
bc.constant.type.yhi = trac trac trac
bc.constant.type.zlo = trac trac disp
bc.constant.type.zhi = trac trac trac

### --- Edges
###     XY edges
bc.constant.type.xloylo = disp trac trac
bc.constant.type.xloyhi = disp trac trac
bc.constant.type.xhiylo = disp trac trac
bc.constant.type.xhiyhi = disp trac trac
bc.constant.val.xhiylo = 0.1 0.0 0.0
bc.constant.val.xhiyhi = 0.1 0.0 0.0
###     ZX edges
bc.constant.type.zloxlo = disp trac trac
bc.constant.type.zhixlo = disp trac trac
bc.constant.type.zloxhi = disp trac trac
bc.constant.type.zhixhi = disp trac trac
bc.constant.val.zloxhi = 0.1 0.0 0.0
bc.constant.val.zhixhi = 0.1 0.0 0.0
###     YZ edges
bc.constant.type.ylozlo = trac disp disp
bc.constant.type.ylozhi = trac disp trac
bc.constant.type.yhizlo = trac trac disp
bc.constant.type.yhizhi = trac trac trac


### --- Corners
###     on xmin face
bc.constant.type.xloylozlo = disp disp disp
bc.constant.type.xloylozhi = disp disp trac
bc.constant.type.xloyhizlo = disp trac disp
bc.constant.type.xloyhizhi = disp trac trac
###     on xmax face
bc.constant.type.xhiylozlo = disp disp disp
bc.constant.val.xhiylozlo = 0.1 0.0 0.0
bc.constant.type.xhiylozhi = disp disp trac
bc.constant.val.xhiylozhi = 0.1 0.0 0.0
bc.constant.type.xhiyhizlo = disp trac disp
bc.constant.val.xhiyhizlo = 0.1 0.0 0.0
bc.constant.type.xhiyhizhi = disp trac trac
bc.constant.val.xhiyhizhi = 0.1 0.0 0.0
```

4.15 Voronoi

This test problem illustrates the use of the phase field method when initialized with a Voronoi tesselation. This example tests the following capabilities

- *Integrator::PhaseFieldMicrostructure*
- *IC::Voronoi*

The following is the result in 2D; the problem can be run in 3D as well.

Notice that there are 100 initial grains in the tesselation, but only 10 actual grains in the simulation. Therefore, some grains will eventually merge together when the boundary between them disappears.

4.15.1 [Voronoi] 2D-100grain-parallel

2d	Two-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
timer	16.04s (beaker) 11.58s (statler)
play_circle	`mpiexec -np 4 ./bin/alamo-2d-g++ tests/Voronoi/input`

4.15.2 [Voronoi] 2D-100grain-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
timer	50.84s (beaker) 37.94s (statler) 60.03s (github)
play_circle	`./bin/alamo-2d-g++ tests/Voronoi/input`

Listing 15: Input file (../../tests/Voronoi/input)

```
#@
#@ [2D-100grain-parallel]
#@ nprocs = 4
#@ dim = 2
#@ benchmark-beaker = 16.04
#@ benchmark-statler = 11.58
#@
#@ [2D-100grain-serial]
#@ nprocs = 1
#@ dim = 2
#@ benchmark-beaker = 50.84
#@ benchmark-statler = 37.94
```

(continues on next page)

(continued from previous page)

```
#@ benchmark-github = 60.03
#@

alamo.program               = microstructure
plot_file                    = tests/Voronoi/output

timestep                    = 0.005
stop_time                    = 1

amr.plot_dt                    = 0.1

amr.max_level                    = 3
amr.n_cell                     = 32 32 32
amr.blocking_factor             = 4
amr.regrid_int                    = 10
amr.grid_eff                    = 1.0

ic.type                             = voronoi
ic.voronoi.number_of_grains = 100
#ic.voronoi.number_of_grains = 100
#ic.type                             = constant
#ic.value = 1 0 0 0 0 0 0 0 0 0
geometry.prob_lo            = 0 0 0
geometry.prob_hi            = 5 5 5
geometry.is_periodic            = 1 1 1

bc.eta.type.xhi                         = periodic
bc.eta.type.xlo                         = periodic
bc.eta.type.yhi                         = periodic
bc.eta.type.ylo                         = periodic
bc.eta.type.zhi                         = periodic
bc.eta.type.zlo                         = periodic

pf.number_of_grains             = 10
pf.M                             = 1.0
pf.mu                             = 10.0
pf.gamma                    = 1.0
pf.l_gb                             = 0.05
pf.sigma0                    = 0.075
```

4.16 VoronoiElastic

This problem is identical to *Voronoi*, but solves mechanical equilibrium in the microstructure with grain-dependent moduli. This problem tests the following capabilities

- *Integrator::PhaseFieldMicrostructure*
- *Integrator::Mechanics*
- *IC::Voronoi*
- *Model::Solid::Affine::Cubic*

The following is the result in 2D:

4.16.1 [VoronoiElastic] 2d-serial

2d	Two-dimensional
square	Serial
verified	Testing script present
timer	65.12s (beaker) 45.09s (statler) 70.39s (github)
play_circle	`./bin/alamo-2d-g++ tests/VoronoiElastic/input`

4.16.2 [VoronoiElastic] 2d-parallel

2d	Two-dimensional
grid_view	Parallel (4 procs)
verified	Testing script present
timer	19.99s (beaker) 16.68s (statler)
play_circle	`mpiexec -np 4 ./bin/alamo-2d-g++ tests/VoronoiElastic/input`

Listing 16: Input file (../../tests/VoronoiElastic/input)

```
#@
#@ [2d-serial]
#@ dim=2
#@ benchmark-beaker = 65.12
#@ benchmark-statler = 45.09
#@ benchmark-github = 70.39
#@
#@ [2d-parallel]
#@ dim=2
#@ nprocs=4
#@ benchmark-beaker=19.99
#@ benchmark-statler=16.68
#@

alamo.program                       = microstructure
plot_file                         = tests/VoronoiElastic/output

timestep                         = 0.0005
stop_time                         = 0.1

amr.plot_dt                         = 0.01
amr.max_level                         = 2
amr.n_cell                        = 64 64 64
```

(continues on next page)

(continued from previous page)

```
amr.max_grid_size = 8

amr.blocking_factor                = 8
amr.base_regrid_int                = 100
amr.grid_eff                        = 1.0
amr.ref_threshold                 = 0.1
amr.cell.all = 1
amr.max_grid_size = 8

ic.type                                 = voronoi
ic.voronoi.number_of_grains        = 40

geometry.prob_lo                = 0 0 0
geometry.prob_hi                = 5 5 5
geometry.is_periodic                = 0 0 0

bc.eta.type.xhi                         = neumann
bc.eta.type.xlo                         = neumann
bc.eta.type.yhi                         = neumann
bc.eta.type.ylo                         = neumann
bc.eta.type.zhi                         = neumann
bc.eta.type.zlo                         = neumann

pf.number_of_grains                = 10
pf.M                                = 1.0
pf.mu                               = 10.0
pf.gamma                        = 1.0
pf.l_gb                                 = 0.05
pf.sigma0                        = 0.075

mechanics.interval                = 10
mechanics.type                        = static
mechanics.bc.type               = constant
mechanics.bc.constant.type.xlo     = disp disp disp
mechanics.bc.constant.type.xhi          = disp disp disp
mechanics.bc.constant.type.xloyhi  = disp disp disp
mechanics.bc.constant.type.xhiyhi  = disp disp disp
mechanics.bc.constant.type.ylo                = neumann neumann neumann
mechanics.bc.constant.type.yhi                = neumann neumann neumann
mechanics.bc.constant.val.xhi           = 0.0 0.01 0.0
mechanics.bc.constant.val.xlo           = 0.0 0.0 0.0
mechanics.bc.constant.val.xhiyhi        = 0.0 0.01 0.0
mechanics.bc.constant.val.xloyhi        = 0.0 0.0 0.0
mechanics.bc.constant.val.xhiylo        = 0.0 0.01 0.0
mechanics.bc.constant.val.xloylo        = 0.0 0.0 0.0
mechanics.bc.constant.val.yhi               = 0.0 0.0 0.0
mechanics.bc.constant.val.ylo               = 0.0 0.0 0.0
mechanics.model.C11             = 1.68
mechanics.model.C12             = 1.21
mechanics.model.C44             = 0.75
mechanics.model.random          = 1
#mechanics.solver.fixed_iter     = 100
```

(continues on next page)

(continued from previous page)

```
mechanics.solver.verbose        = 3
mechanics.print_model = 1
mechanics.print_residual = 1
mechanics.solver.tol_rel = 1E-8
mechanics.solver.tol_abs = 1E-8
mechanics.solver.average_down_coeffs = 0
mechanics.time_evolving = 0
```

CHAPTER

FIVE

INPUTS

5.1 BC

This is the mechanism for impositing boundary conditions on `Set::Field` objects of scalar type. Typical convention is for `[prefix]` to be given by the field name. For instance,

```
bc.temp.type.xhi = dirichlet dirichlet dirichlet
bc.temp.val.xhi  = 0.0 1.0 0.0
```

corresponds to the boundary condition for temperature. See the specific integrator for details.

5.1.1 BC::Constant

This is the most commonly used standard boundary condition implementation. The name "Constant" refers to the invariance of the BC value or character along each face of the simulation domain; however, you may cause a change in the value with time by using a *Numeric::Interpolator::Linear* string.

The BC on each face is specified with a `type` string that specifies the nature of the BC, and a `val` that specifies the value. By default, all types are Dirichlet and all values are 0.0. The complete list of BC types are below:

BC Type	Description
`Dirichlet`	Standard Dirichlet boundary condition (https://en.wikipedia.org/wiki/Dirichlet_boundary_condition). Fixes the value of the **function** at the boundary.
`dirichlet`	
`EXT_DIR`	
`Neumann`	Standard Neumann boundary condition (https://en.wikipedia.org/wiki/Neumann_boundary_condition)
`neumann`	
`Periodic`	Standard periodic boundary condition. (https://en.wikipedia.org/wiki/Periodic_boundary_conditions). **Important**: Ensure that your geometry is specified to be periodic using `geometry.is_periodic`. For instance, if periodic in x, set to `1 0 0`
`periodic`	
`INT_DIR`	
`REFLECT_ODD`	Effectively the same as dirichlet but with value fixed to 0
`reflect_odd`	
`REFLECT_EVEN`	Effectively the same as neumann but with value fixed to 0
`BOGUS_BC`	Used for internal operations - not for general use
`FOEXTRAP`	These are BC types inherited from AMReX and not yet implemented in Alamo. If you need BCs of this type, you will need to implement them yourself.
`HOEXTRAP`	
`Interior`	
`Inflow`	
`Outflow`	
`Symmetry`	
`SlipWall`	
`NoSlipWall`	
`interior`	
`Marshak`	
`SanchezPomraning`	
`inflow`	

The BC values can be specified either as a number (e.g. `1.0`) or using a linear interpolator string (e.g. `(0,1:2.3,-0.1)`).

The number of values and types must be either 0 (for defaults), 1 (to set the same for all field components) or N (where N=number of field components).

Parameter	Type	Description
[prefix].type.xlo	queryarr	BC type on the lower x edge (2d) face (3d)
[prefix].type.xhi	queryarr	BC type on the upper x edge (2d) face (3d)
[prefix].type.ylo	queryarr	BC type on the lower y edge (2d) face (3d)
[prefix].type.yhi	queryarr	BC type on the upper y edge (2d) face (3d)
[prefix].type.zlo	queryarr	BC type on the lower z face (processed but ignored in 2d to prevent unused input errors)
[prefix].type.zhi	queryarr	BC type on the upper z face (processed but ignored in 2d to prevent unused input errors)
[prefix].val.xlo	queryarr	BC value on the lower x edge (2d) face (3d)
[prefix].val.xhi	queryarr	BC value on the upper x edge (2d) face (3d)
[prefix].val.ylo	queryarr	BC value on the lower y edge (2d) face (3d)
[prefix].val.yhi	queryarr	BC value on the upper y edge (2d) face (3d)
[prefix].val.zlo	queryarr	BC value on the lower z face (processed but ignored in 2d to prevent unused input errors)
[prefix].val.zhi	queryarr	BC value on the upper z face (processed but ignored in 2d to prevent unused input errors)

5.1.2 BC::Step

Warning: This will be replaced by the more general `BC::Expression`

Parameter	Type	Description
[prefix].h1	query	Location of the step on the xlo edge/face
[prefix].h2	query	Location of the step on the xhi edge/face

5.1.3 BC::Operator

The `BC::Operator` family of BC classes are designed to work with implicit operators. **They are not the same thing as other BC types** and cannot be used in that same way. (This is not an ideal taxonomy for BC structures - we may transition other BC types to be subspaces of BC::Integrator.)

5.1.3.1 BC::Operator::Elastic

Class of BC operators that work with *Operator::Elastic*.

BC::Operator::Elastic::Constant

This BC defines boundary conditions that are constant with respect to space. However they may change in time using the *Numeric::Interpolator::Linear* method.

In 2D, BCs are prescribed for each edge (4) and corner (4) on the boundary, for a total of 8 possible regions. In 3D, BCs are prescribed for each face (6), each edge (12), and each corner (8), for a total of 26 possible regions. Care must be taken to define BCs for edges/corners consistently with the faces/edges, or you will get poor convergence and inaccurate behavior. (See *BC::Operator::Elastic::TensionTest* for a reduced BC with streamlined load options.)

To define a boundary condition, you must define both a "type" and a "value" in each direction. The type can be "displacement", "disp", "neumann", "traction", "trac". The value is the corresponding value. All BCs are, by default, dirichlet (displacement) with a value of zero.

Parameter	Type	Description
`[prefix].type.xloylozlo`	queryarr	3D Corner
`[prefix].type.xloylozhi`	queryarr	3D Corner
`[prefix].type.xloyhizlo`	queryarr	3D Corner
`[prefix].type.xloyhizhi`	queryarr	3D Corner
`[prefix].type.xhiylozlo`	queryarr	3D Corner
`[prefix].type.xhiylozhi`	queryarr	3D Corner
`[prefix].type.xhiyhizlo`	queryarr	3D Corner
`[prefix].type.xhiyhizhi`	queryarr	3D Corner
`[prefix].type.ylozlo`	queryarr	3D Edge
`[prefix].type.ylozhi`	queryarr	3D Edge
`[prefix].type.yhizlo`	queryarr	3D Edge
`[prefix].type.yhizhi`	queryarr	3D Edge
`[prefix].type.zloxlo`	queryarr	3D Edge

continues on next page

Table 1 – continued from previous page

Parameter	Type	Description
[prefix].type.zloxhi	queryarr	3D Edge
[prefix].type.zhixlo	queryarr	3D Edge
[prefix].type.zhixhi	queryarr	3D Edge
[prefix].type.xloylo	queryarr	3D Edge / 2D Corner
[prefix].type.xloyhi	queryarr	3D Edge / 2D Corner
[prefix].type.xhiylo	queryarr	3D Edge / 2D Corner
[prefix].type.xhiyhi	queryarr	3D Edge / 2D Corner
[prefix].type.xlo	queryarr	3D Face 2D Edge
[prefix].type.xhi	queryarr	3D Face 2D Edge
[prefix].type.ylo	queryarr	3D Face 2D Edge
[prefix].type.yhi	queryarr	3D Face 2D Edge
[prefix].type.zlo	queryarr	3D Face
[prefix].type.zhi	queryarr	3D Face
[prefix].val.xloylozlo	queryarr	3D Corner
[prefix].val.xloylozhi	queryarr	3D Corner
[prefix].val.xloyhizlo	queryarr	3D Corner
[prefix].val.xloyhizhi	queryarr	3D Corner
[prefix].val.xhiylozlo	queryarr	3D Corner
[prefix].val.xhiylozhi	queryarr	3D Corner
[prefix].val.xhiyhizlo	queryarr	3D Corner
[prefix].val.xhiyhizhi	queryarr	3D Corner
[prefix].val.ylozlo	queryarr	3D Edge
[prefix].val.ylozhi	queryarr	3D Edge
[prefix].val.yhizlo	queryarr	3D Edge
[prefix].val.yhizhi	queryarr	3D Edge

continues on next page

Table 1 – continued from previous page

Parameter	Type	Description
[prefix].val.zloxlo	queryarr	3D Edge
[prefix].val.zloxhi	queryarr	3D Edge
[prefix].val.zhixlo	queryarr	3D Edge
[prefix].val.zhixhi	queryarr	3D Edge
[prefix].val.xloylo	queryarr	3D Edge / 2D Corner
[prefix].val.xloyhi	queryarr	3D Edge / 2D Corner
[prefix].val.xhiylo	queryarr	3D Edge / 2D Corner
[prefix].val.xhiyhi	queryarr	3D Edge / 2D Corner
[prefix].val.xlo	queryarr	3D Face / 2D Edge
[prefix].val.xhi	queryarr	3D Face / 2D Edge
[prefix].val.ylo	queryarr	3D Face / 2D Edge
[prefix].val.yhi	queryarr	3D Face / 2D Edge
[prefix].val.zlo	queryarr	3D Face
[prefix].val.zhi	queryarr	3D Face

BC::Operator::Elastic::Expression

Parameter	Type	Description
This is basically the same as *BC::Operator::Elastic::Constant* except that you can use dynamically compiled expressions in space and time to define values. Usage is exactly the same except that the "val" inputs can depend on x, y, z, and t.		
`[prefix].type.xloylozlo`	queryarr	3D Corner
`[prefix].type.xloylozhi`	queryarr	3D Corner
`[prefix].type.xloyhizlo`	queryarr	3D Corner
`[prefix].type.xloyhizhi`	queryarr	3D Corner
`[prefix].type.xhiylozlo`	queryarr	3D Corner
`[prefix].type.xhiylozhi`	queryarr	3D Corner
`[prefix].type.xhiyhizlo`	queryarr	3D Corner
`[prefix].type.xhiyhizhi`	queryarr	3D Corner
`[prefix].type.ylozlo`	queryarr	3D Edge
`[prefix].type.ylozhi`	queryarr	3D Edge
`[prefix].type.yhizlo`	queryarr	3D Edge
`[prefix].type.yhizhi`	queryarr	3D Edge
`[prefix].type.zloxlo`	queryarr	3D Edge
`[prefix].type.zloxhi`	queryarr	3D Edge
`[prefix].type.zhixlo`	queryarr	3D Edge
`[prefix].type.zhixhi`	queryarr	3D Edge
`[prefix].type.xloylo`	queryarr	3D Edge / 2D Corner
`[prefix].type.xloyhi`	queryarr	3D Edge / 2D Corner
`[prefix].type.xhiylo`	queryarr	3D Edge / 2D Corner
`[prefix].type.xhiyhi`	queryarr	3D Edge / 2D Corner
`[prefix].type.xlo`	queryarr	3D Face 2D Edge
`[prefix].type.xhi`	queryarr	3D Face 2D Edge
`[prefix].type.ylo`	queryarr	3D Face 2D Edge
`[prefix].type.yhi`	queryarr	3D Face 2D Edge
`[prefix].type.zlo`	queryarr	3D Face
`[prefix].type.zhi`	queryarr	3D Face

BC::Operator::Elastic::TensionTest

Parameter	Type	Description
`[prefix].type`	query	Tension test type. (Only option right now is `uniaxial_stress_clamp`)
`[prefix].disp`	query	Applied displacement (can be interpolator)
`[prefix].trac`	query	Applied traction (can be interpolator)

5.2 IC

5.2.1 IC::Affine

Initialize a field to a value (alpha) on the positive side of a hyperplane

> **Warning:** This is an old-fasioned IC that will soon be deleted.

Parameter	Type	Description
`ic.n`	queryarr	Normal vector for hyperplane
`ic.alpha`	query	Value of the field on the positive side of the hyperplane

5.2.2 IC::BMP

Initialize a field using a bitmap image. (2D only)

Note that in GIMP, you must select "do not write color space information" and "24 bit R8 G8 B8" when exporting the BMP file.

Parameter	Type	Description
`[prefix].filename`	query	BMP filename.
`[prefix].fit`	query	How to fit. (options: stretch, fitheight, fitwidth)
`[prefix].channel`	query	Color channel to use (options: r, R, g, G, b, B)
`[prefix].min`	query	Scaling value - minimum (default: 0.0)
`[prefix].max`	query	Scaling value - maximum (default: 255.0)

5.2.3 IC::Constant

Basic IC that just sets the entire field to a constant value. Works with a single or multiple-component field.

Parameter	Type	Description
`[prefix].value`	queryarr	Array of constant values. The number of values should equal either 1 or N where N is the number of fab components

5.2.4 IC::Cuboid

Parameter	Type	Description
`ic.center`	queryarr	Coordinates (X Y Z) of the center of the square/cube.
`ic.length`	queryarr	Lenth of the square/cube edges

5.2.5 IC::DoubleNotch

Implement a double-notch. Primarily used in fracture.

Warning: This function is depricated. Use *IC::Expression* instead.

Parameter	Type	Description
`[prefix].thickness`	query	Thickness of the notches
`[prefix].width`	query	Width of the notches
`[prefix].x0`	queryarr	Center of the notches
`[prefix].L`	query	Length of the notches

5.2.6 IC::Expression

Initialize a field using a mathematical expression. Expressions are imported as strings and are compiled real-time using the AMReX Parser.

Works for single or multiple-component fields. Use the `regionN` (N=0,1,2, etc. up to number of components) to pass expression. For example:

```
ic.region0 = "sin(x*y*z)"
ic.region1 = "3.0*(x > 0.5 and y > 0.5)"
```

for a two-component field. It is up to you to make sure your expressions are parsed correctly; otherwise you will get undefined behavior.

Constants You can add constants to your expressions using the `constant` directive. For instance, in the following code

```
psi.ic.type=expression
psi.ic.expression.constant.eps = 0.05
psi.ic.expression.constant.R   = 0.25
psi.ic.expression.region0 = "0.5 + 0.5*tanh((x^2 + y^2 - R)/eps)"
```

the constants `eps` and `R` are defined by the user and then used in the subsequent expression. The variables can have any name made up of characters that is not reserved. However, if multiple ICs are used, they must be defined each time for each IC.

Parameter	Type	Description
`[prefix].coord`	query	coordinate system to use: “cartesian” (for x,y,z,t) and “polar” (for r, theta, z, t)

5.2.7 IC::Ellipsoid

This IC initializes a single-component fab using the formula

$$\phi = \begin{cases} \alpha_{in} & (\mathbf{x} - \mathbf{x}_0)^T \mathbf{A} (\mathbf{x} - \mathbf{x}_0) \leq r \\ \alpha_{out} & \text{else} \end{cases}$$

The boundary can be mollified using an error function with parameter ε

Warning: This IC is redundant with *IC::Ellipse* and *IC::Expression* and will probably be consolidated

Parameter	Type	Description
`[prefix].center`	queryarr	Center of the ellipse $\mathbf{x}_0$
`[prefix].A`	queryarr	Matrix defining elipse radii and orientation
`[prefix].radius`	queryarr	"Vector of radii (use instead of A)"
`[prefix].eps`	queryarr	Mollifying value for erf
`[prefix].in_value`	query	Value of field inside ellipse (default: 0)
`[prefix].out_value`	query	Value of field outside ellipse (default: 1)
`[prefix].mollifier`	query	Type of mollifier to use (options: dirac, [gaussian])

5.2.8 IC::Ellipse

If `number_of_inclusions` is specified, then multiple ellipses are specified. In this case, each parameter must have number_of_inclusion*M values, where M is the number of values specified for the single ellipse case.

Warning: This class is redundant with *IC::Ellipsoid* and *IC::Expression* and will probably be consolidated.

Parameter	Type	Description
`[prefix].x0`	queryarr	Coorinates of ellipse center
`[prefix].eps`	query	Diffuse boundary thickness
`[prefix].A`	queryarr	DxD square matrix defining an ellipse.
`[prefix].a`	queryarr	If `A` is not defined, then assume a sphere with radius `a`
`[prefix].number_of_inclusions`	query	Number of ellipses
`[prefix].center`	queryarr	center of the ellipse
`[prefix].A`	queryarr	either a vector containing ellipse radii, or a matrix defining the ellipse
`[prefix].A`	queryarr	Same
`[prefix].radius`	queryarr	Array of radii [depricated]
`[prefix].eps`	queryarr	Regularization for smooth boundary
`[prefix].invert`	query	Flip the inside and the outside

5.2.9 IC::Laminate

Create a single laminate with specified orientation, thickness, and offset.

Parameter	Type	Description
[prefix].number_of_inclusions	query	How many laminates (MUST be greater than or equal to 1). Default = 1
[prefix].center	queryarr	(x,y,[z]) values for the center point of the laminate
[prefix].thickness	queryarr	thickness of the laminate
[prefix].orientation	queryarr	Vector normal to the interface of the laminate
[prefix].eps	queryarr	Diffuse thickness
[prefix].mollifier	query	Type of mollifer to use (options: dirac, [gaussian])
[prefix].singlefab	query	Switch to mode where only one component is used.
[prefix].invert	query	Take the complement of the laminate

5.2.10 IC::Notch

Create a simple notch in an otherwise uniformly filled region. (This was created for, and mostly used for, Mode II fracture tests.)

Parameter	Type	Description
[prefix].center	queryarr	Center of notch
[prefix].orientation	queryarr	Vector describing notch orientation
[prefix].thickness	queryarr	Thickness of notch
[prefix].length	queryarr	Length of notch
[prefix].radius	queryarr	Radius of notch ends
[prefix].eps	query	Magnitude of mollifier
[prefix].mollifier	query	

5.2.11 IC::PS

Fill a domain with randomly packed and sized spheres.

Warning: This is not used for anything useful as far as we can see. It will likely be removed shortly.

Parameter	Type	Description
[prefix].nspheres	query	
[prefix].matrix	query	
[prefix].inclusion	query	

5.2.12 IC::PSRead

Fill a domain (region where field=0) with packed spheres (regions where field=1). Sphere locations and radii are determined from an xyzr file.

Parameter	Type	Description
`[prefix].eps`	query	Diffuseness of the sphere boundary
`[prefix].filename`	query	Location of .xyzr file

5.2.13 IC::PerturbedInterface

Initialize a perturbed interface using Fourier Modes

Notes: 1. todo Extend this class to allow for 3D perturbations, currently only 2D are allowed 2. todo Allow for cosine (or complex exponential) expansions rather than just sin. 3. note This is a **two grain only** initial condition. 4. note This replaces the depricated "perturbed_bar" initial condition from previous versions

The interface is defined as the $x = 0$ plane (2D), or the $x = 0, z = 0$ plane (3D). The equation for the interface is given by $y(x,z) = \sum_{n\in\{n_1,\ldots,n_N\}} A_n \sin(n\pi x/L_x)$ where A_n are the amplitudes (stored in #wave_amplitudes), $n_1,\ldots,n_N \subset \mathbb{Z}_+$ are wave numbers (stored in #wave_numbers), and L_x is the length in the x direction (obtained using the #geom object).

Grain 1 is defined as being above $y(x,z)$, Grain 2 is defined as being below.

Parameter	Type	Description
`[prefix].wave_numbers`	queryarr	Wave numbers
`[prefix].wave_amplitudes`	queryarr	Wave amplitudes
`[prefix].normal`	query	Which axis is normal to the interface (x,y,z)
`[prefix].offset`	query	Interface offset from origin
`[prefix].reverse`	query	If true, flip the interface (default:false)
`[prefix].mollifier`	query	Mollifier (options: dirac, [gaussian])
`[prefix].eps`	query	Magnitude of mollifier

5.2.14 IC::Sphere

Parameter	Type	Description
This is a somewhat antiquated IC that will eventually be replaced with the Expression IC.		
`[prefix].radius`	query	Radius of the sphere
`[prefix].center`	queryarr	Vector location of the sphere center
`[prefix].inside`	query	Value of the field inside the sphere
`[prefix].outside`	query	Value of the field outside teh sphere
`[prefix].type`	query	Type - can be cylinder oriented along the x, y, z directions or full sphere.

5.2.15 IC::TabulatedInterface

Parameter	Type	Description
[prefix].xs	queryarr	
[prefix].ys	queryarr	

5.2.16 IC::Trig

Parameter	Type	Description
[prefix].nr	queryarr	
[prefix].ni	queryarr	
[prefix].dim	query	
[prefix].alpha	query	

5.2.17 IC::Voronoi

Parameter	Type	Description
[prefix]. number_of_grains	query	
[prefix].alpha	queryarr	
[prefix].seed	query	

5.3 IO

5.3.1 IO::ParmParse

This is a thin wrapper to the amrex::ParmParse class This class exists to add some additional parsing capability, e.g. parsing Set::Matrix and Set::Vector data types.

IO::ParmParse uses static `Parse()` functions to perform cascading class-based input parsing. See the *Autodoc and Autotest* section for instructions on adding documentation.

This is standard infrastructure code; make sure you know what you ard doing before you change it.

5.3.2 IO::WriteMetaData

This provide an IO routine for generating run-specific output. Every Alamo run produces a metadata file and (if applicable) a diff.patch file that reflects the exact state when the simulation was run. Note that the file `WriteMetaData.cpp` file is _always_ recompiled upon `make`, in order to pull the latest information about the local state at build.

This is standard infrastructure code; make sure you know what you ard doing before you change it.

5.4 ⚙ Integrator

Pure abstract class for managing data structures, time integration (with substepping), mesh refinement, and I/O.

Native input file parameters:

```
max_step  = [maximum number of timesteps]
stop_time = [maximum simulation time]
timestep  = [time step for coarsest level]

amr.regrid_int = [number of timesteps between regridding]
amr.plot_int   = [number of timesteps between dumping output]
amr.plot_file  = [base name of output directory]

amr.nsubsteps  = [number of temporal substeps at each level. This can be
                  either a single int (which is then applied to every refinement
                  level) or an array of ints (equal to amr.max_level)
                  corresponding to the refinement for each level.]
```

Inherited input file parameters (from amrex AmrMesh class):

```
amr.v                 = [verbosity level]
amr.max_level         = [maximum level of refinement]
amr.n_proper          =
amr.grid_eff          =
amr.n_error_buff      =
amr.ref_ratio_vect    = [refinement ratios in each direction]
amr.ref_ratio         = [refinement ratio in all directions (cannot be used with ref_
↪ratio_vect)]
amr.max_grid_x        =
amr.blocking_factor   =
amr.n_cell            = [number of cells on coarsest level]
amr.refine_grid_layout =
amr.check_input       =
```

Parameter	Type	Description
These are basic parameters that are, in general, common to all Alamo simulations.		
max_step	query	Number of iterations before ending
stop_time	query	Simulation time before ending
timestep	query	Nominal timestep on amrlev = 0
restart	query	Name of restart file to READ from
restart_cell	query	Name of cell-fab restart file to read from
restart_node	query	Name of node-fab restart file to read from
ignore	queryarr	Space-separated list of entries to ignore
These are parameters that are specific to the AMR/regridding part of the code.		
amr.regrid_int	query	Regridding interval in step numbers
amr.base_regrid_int	query	Regridding interval based on coarse level only
amr.plot_int	query	Interval (in timesteps) between plotfiles
amr.plot_dt	query	Interval (in simulation time) between plotfiles
amr.plot_file	query	Output file
amr.cell.all	query	Turn on to write all output in cell fabs (default: off)
amr.cell.any	query	Turn off to prevent any cell based output (default: on)
amr.node.all	query	Turn on to write all output in node fabs (default: off)
amr.node.any	query	Turn off to prevent any node based output (default: on)
amr.max_plot_level	query	Specify a maximum level of refinement for output files
amr.nsubsteps	queryarr	Number of substeps to take on each level (default: 2)
amr.nsubsteps	query	Number of substeps to take on each level (set all levels to this value)
Information on how to generate thermodynamic data (to show up in thermo.dat)		
amr.thermo.int	query	Default: integrate every time. Integration interval (1)
amr.thermo.plot_int	query	Interval (in timesteps) between writing
amr.thermo.plot_dt	query	Interval (in simulation time) between writing
Instead of using AMR, prescribe an explicit, user-defined set of grids to work on. This is pretty much always used for testing purposes only.		
explicitmesh.on	query	Use explicit mesh instead of AMR

5.4.1 Integrator::CahnHilliard

CahnHilliard

5.4.2 Integrator::Fracture

Parameter	Type	Description
IC for crack field		
crack.ic.type	queryarr	
crack.ic.notch	queryclass	
crack.ic.ellipsoid	queryclass	
elastic.solver	queryclass	Solver::Nonlocal::Linear<brittle_fracture_model_type_test> solver(op_b);

5.4.3 Integrator::Flame

Parameter	Type	Description
pf.eps	query	Burn width thickness
pf.kappa	query	Interface energy param
pf.gamma	query	Scaling factor for mobility
pf.lambda	query	Chemical potential multiplier
pf.w1	query	Unburned rest energy
pf.w12	query	Barrier energy
pf.w0	query	Burned rest energy
pf.P	query	Pressure [UNITS?]
pf.r_ap	query	AP Power law multiplier
pf.n_ap	query	AP Power law exponent
pf.r_htpb	query	HTPB Power law multiplier
pf.n_htpb	query	HTPB Power law exponent
pf.r_comb	query	Combination power law multiplier
pf.n_comb	query	Combination power law exponent
pf.eta.bc	queryclass	See *BC::Constant*
eta.ic.type	query	IC type - [packedspheres,laminate] - see classes for more information
thermal.on	query	These parameters are for the **Thermal transport model** Whether to use the thermal model
thermal.rho1	query	Density (before)
thermal.rho0	query	Density (after)
thermal.ka	query	Thermal conductivity (before and after)
thermal.kh	query	Thermal conductivity (before and after)
thermal.k0	query	Thermal conductivity (before and after)
thermal.cp1	query	Specific heat (before and after)
thermal.cp0	query	Specific heat (before and after)
thermal.delA	query	Thermal flux of each material
thermal.delH	query	Thermal flux of each material
amr. refinement_criterion	query	Refinement criterion for eta field
amr. refinement_criterion_temp	query	Refinement criterion for temperature field
amr. refinament_restriction	query	Eta value to restrict the refinament for the temperature field
phi.ic.type	query	IC type (psread, laminate, constant)
model_ap	queryclass	
model_htpb	queryclass	

5.4.4 Integrator::HeatConduction

This implements a basic heat conduction method in Alamo. The partial differential equation to be solved is

$$\frac{\partial T}{\partial t} = \alpha \, \Delta T$$

where T is temperature, t is time, and $alpha$ is the thermal diffusivity. Integration is performed explicitly in time using forward Euler, and differentiation is performed using the finite difference method.

Parameter	Type	Description
The Parse function initializes the HeatConduction object using a parser, pp. Note that this is a static function, which means it does not have direct access to member variables. Instead, it initializes the variables inside the argument, "value", and so all references to member items are prefixed by "value."		
`[prefix].heat.alpha`	query	Diffusion coefficient [1.0]
`[prefix].heat.refinement_threshold`	query	Criterion for mesh refinement [0.01]
`[prefix].ic.type`	query	Initial condition type ([sphere], constant, expression)
`[prefix].bc.temp`	queryclass	*BC::Constant* parameters with prefix bc.temp

5.4.5 Integrator::Mechanics

This is a general purpose integrator that focuses on solving elasticity/mechanics equations in the absence of other multiphysics simulations. It is enabled by `alamo.program=mechanics` if used on its own, in which case there is no prefix. If it is being used by another integrator, see that integrator to determine the value of `[prefix]` (often equal to `elastic`).

This integrator inherits from *Integrator::Base::Mechanics*; see documentation for that integrator for additional parameters.

Model setup There are two basic tools for setting up a mechanics problem in this integrator.

1. The `eta` field: this is used to mix models of different types. Use `nmodels` to specify how many material models to use, and then specify each model as `model1`, `model2`, etc. The type of moel is specified using the `alamo.program.mechanics.model` input.

   Once you have done this, you must determine the spatial distribution of each model. This is done by choosing an IC for the eta field with `ic.type`. The `IC::Expression` is the most general and recommended. The models are then mixed linearly, i.e.

   $$W_{\text{eff}} = \sum_{i=1}^{N} W_i \, \eta_i(\mathbf{x})$$

   See the *Eshelby* test for an example of model mixing.

2. The `psi` field: this is used specifically for cases where a "void" region is desired. Its usage is similar to the `eta` case, and is conceptually similar in that it scales the model field to near-zero in order to mimic the (lack of) mechanical behavior in an empty region. It is important to use `psi` here, for reasons that are discussed in detail in this paper. The initialization of `psi` is similar to that for `eta`.

   See the *PlateHole* and *RubberPlateHole* for canonical exmaples. The *Integrator::Fracture* and *Integrator::TopOp* integrators are examples of integrators that leverage the psi property.

Body forces currently have limited support due to the relatively low number of times they are needed. See the *Integrator::Base::Mechanics* documentation for detail. See the *TrigTest* test for examples of current body force implementation.

Boundary conditions are implemented using the *BC::Operator::Elastic* classes. See the documentation on these classes for more detail. See any of the mechanics-based tests for examples of boundary condition application.

Parameter	Type	Description
Mechanics inputs. See also *Integrator::Base::Mechanics*		
[prefix].nmodels	query	Number of elastic model varieties
[prefix].eta_ref_threshold	query	Refinement threshold for eta field
[prefix].ref_threshold	query	Refinement threshold for strain gradient
[prefix].ic.type	query	Read IC type for the eta field
[prefix].eta.reset_on_regrid	query	Whether to re-initialize eta when re-gridding occurs. Default is false unless eta ic is set, then default is. true.
[prefix].psi.ic.type	query	Read IC type for the eta field
[prefix].psi.reset_on_regrid	query	Whether to re-initialize psi when re-gridding occurs. Default is false unless a psi ic is set, then default is true.

5.4.6 Integrator::PolymerDegradation

file PolymerDegradation.H

Parameter	Type	Description
Water diffusion		
water.on	query	
water.diffusivity	query	Diffusivity
water.refinement_threshold	query	AMR refinement criterion
water.ic_type	query	
water.ic.value	queryarr	
water.ic.bc	queryclass	

5.4.7 Integrator::PhaseFieldMicrostructure

file PhaseFieldMicrostructure.H

Parameter	Type	Description
[prefix].pf.number_of_grains	query	Number of grain fields (may be more if using different IC)
[prefix].pf.M	query	Mobility
[prefix].pf.mu	query	Phase field μ
[prefix].pf.gamma	query	Phase field γ
[prefix].pf.sigma0	query	Initial GB energy if not using GB anisotropy
[prefix].pf.l_gb	query	Mobility
[prefix].pf.elastic_df	query	Determine whether to use elastic driving force

continues on next page

Table 3 – continued from previous page

Parameter	Type	Description
[prefix].pf.elastic_mult	query	Multiplier of elastic energy
[prefix].pf.elastic_threshold	query	Elastic threshold (ϕ_0)
[prefix].amr.max_level	query	Maximum AMR level
[prefix].amr.ref_threshold	query	Phase field refinement threshold
[prefix].mechanics.tstart	query	Elasticity
[prefix].mechanics.model	queryclass	By default, read in the model specified by "mechanics.model"
[prefix].lagrange.on	query	Lagrange multiplier method for enforcing volumes Turn on
[prefix].lagrange.lambda	query	Lagrange multiplier value
[prefix].lagrange.vol0	query	Prescribed volume
[prefix].lagrange.tstart	query	Time to start enforcing Lagrange multipler
[prefix].anisotropy.on	query	Anisotropic grain boundary energy parameters Turn on
[prefix].anisotropy.beta	query	Regularization param
[prefix].anisotropy.tstart	query	Time to turn on anisotropy
[prefix].anisotropy.timestep	query	Modify timestep when turned on
[prefix].anisotropy.plot_int	query	Modify plot_int when turned on
[prefix].anisotropy.plot_dt	query	Modify plot_dt when turned on
[prefix].anisotropy.thermo_int	query	Modify thermo int when turned on
[prefix].anisotropy.thermo_plot_int	query	Modify thermo plot int when turned on
[prefix].anisotropy.elastic_int	query	Frequency of elastic calculation
[prefix].anisotropy.regularization	query	Type of regularization to use

continues on next page

Table 3 – continued from previous page

Parameter	Type	Description
[prefix].anisotropy.gb_type	query	Set the anisotropic GB model Type of GB energy to use
[prefix].bc.eta.type	query	Type (constnat)
[prefix].ic.type	query	IC Type

5.4.8 Integrator::SutureCrack

Parameter	Type	Description
IC for crack field		
crack.ic.type	query	
crack.ic.notch	queryclass	

5.4.9 Integrator::ThermoElastic

Parameter	Type	Description
[prefix].hc	queryclass	
[prefix].el	queryclass	
[prefix].alpha	queryarr	

5.4.10 Integrator::TopOp

Parameter	Type	Description
The mechanics integrator manages the solution of an elastic solve using the MLMG solver.		
[prefix].model	queryclass	
[prefix].psi.ic.type	query	Read IC type for the eta field
[prefix].eta_ref_threshold	query	
[prefix].alpha	query	
[prefix].beta	query	
[prefix].gamma	query	
[prefix].L	queryclass	
[prefix].volume0frac	query	
[prefix].volume0	query	
[prefix].lambda	queryclass	

5.4.11 Integrator::Base

5.4.11.1 Integrator::Base::Mechanics

Parameter	Type	Description
The mechanics integrator manages the solution of an elastic solve using the MLMG solver.		
[prefix].type	query	
[prefix].time_evolving	query	
[prefix].solver	queryclass	Read parameters for *Solver::Nonlocal::Newton* solver
[prefix].viscous.mu	query	
[prefix].bc.type	query	Determine the boundary condition type (contant, tension_test, expression)
[prefix].print_model	query	
[prefix].rhs.type	query	Initializer for RHS
[prefix].interval	query	Timestep interval for elastic solves (default - solve every time)
[prefix].max_coarsening_level	query	Maximum multigrid coarsening level (default - none, maximum coarsening)
[prefix].print_residual	query	
[prefix].elastic_ref_threshold	query	Whether to refine based on elastic solution
[prefix].zero_out_displacement	query	Set this to true to zero out the displacement before each solve. (This is a temporary fix - we need to figure out why this is needed.)

5.5 Model

In Alamo, any type of constitutive behavior is encapsulated in a "Model". There is a broad range of model types, and so there is no abstract Model class. Rather, the base Model classes are stored in subsets of Model.

5.5.1 Model::Interface

5.5.1.1 Model::Interface::Crack

Model::Interface::Crack::Constant

Parameter	Type	Description
[prefix].G_c	query	
[prefix].zeta	query	
[prefix]. mobility	query	
[prefix]. threshold	query	
[prefix].gtype	query	
[prefix].wtype	query	
[prefix]. exponent	query	

Model::Interface::Crack::Sin

Parameter	Type	Description
[prefix].Gc0	query	
[prefix].Gc1	query	
[prefix].theta0	query	
[prefix].zeta	query	
[prefix]. mobility	query	
[prefix]. threshold	query	
[prefix].gtype	query	
[prefix].wtype	query	
[prefix]. exponent	query	

5.5.1.2 Model::Interface::GB

Model::Interface::GB::AbsSin

Parameter	Type	Description
[prefix].theta0	query	
[prefix].sigma0	query	convert degrees into radians
[prefix].sigma1	query	

Model::Interface::GB::Read

Parameter	Type	Description
[prefix].filename	query	

Model::Interface::GB::SH

Parameter	Type	Description
[prefix].theta0	query	
[prefix].phi0	query	convert degrees into radians
[prefix].sigma0	query	convert degrees into radians
[prefix].sigma1	query	
[prefix].regularization	query	

Model::Interface::GB::Sin

Parameter	Type	Description
[prefix].theta0	query	
[prefix].sigma0	query	convert degrees into radians
[prefix].sigma1	query	
[prefix].n	query	

5.5.2 Model::Solid

Solid models are used with the *Integrator::Mechanics* integrator, which implements the *Solver::Nonlocal::Newton* elasticity solver. All solid models inherit from the `Model::Solid` base class, which requires all of the necessary components to be used in a mechanics solve. Model classes have basically two functions:

1. Provide energy (W), stress (DW), and modulus (DDW) based on a kinematic variable
2. Evolve internal variables in time.

5.5.2.1 Model::Solid::Affine

"Affine" generally means "linear with an offset". Here we use "affine" to refer to models that are elastic with an eigenstrain, i.e.

$$\sigma = \mathbb{C}(\varepsilon - \varepsilon_0)$$

The quantity ε_0 is any kind of eigenstrain - examples include thermal strain, plastic strain, or viscous strain. This class can be used directly, where the eigenstrain is read in or set by the integrator. There are also classes (particularly visco/plastic models) that inherit from this type of model.

Model::Solid::Affine::CubicDegradable

Parameter	Type	Description
This class inherits from *Model::Solid::Affine::Cubic*. It provides the ability to "degrade" while retaining information about its original, pristine state		
`[prefix].C11`	query	Original, undegraded $\mathbb{C}_{11}$
`[prefix].C12`	query	Original, undegraded $\mathbb{C}_{12}$
`[prefix].C44`	query	Original, undegraded $\mathbb{C}_{44}$
`[prefix].phi1`	query	Bunge Euler angles ϕ_1
`[prefix].Phi`	query	Bunge Euler angles Φ
`[prefix].phi2`	query	Bunge Euler angles ϕ_2

Model::Solid::Affine::Cubic

Parameter	Type	Description
This class extends *Model::Solid::Linear::Cubic* by adding an eigenstrain. (See the Linear::Cubic class for more inputs for this model)		
`[prefix].F0`	queryarr	Eigenstrain

Model::Solid::Affine::Hexagonal

Parameter	Type	Description
`[prefix].F0`	queryarr	Eigenstrain

Model::Solid::Affine::Isotropic

Parameter	Type	Description
`[prefix].lame`	query	Lame modulus
`[prefix].shear`	query	Shear modulus
`[prefix].E`	query	Elastic modulus
`[prefix].nu`	query	Poisson's ratio
`[prefix].E`	query	
`[prefix].shear`	query	
`[prefix].E`	query	
`[prefix].lame`	query	
`[prefix].nu`	query	
`[prefix].shear`	query	
`[prefix].lambda`	query	
`[prefix].nu`	query	
`[prefix].bulk`	query	
`[prefix].shear`	query	
`[prefix].F0`	queryarr	Eigendeformation gradient

Model::Solid::Affine::IsotropicDegradable

Parameter	Type	Description
`[prefix].lame`	query	
`[prefix].shear`	query	
`[prefix].E`	query	
`[prefix].nu`	query	

Model::Solid::Affine::J2Plastic

Parameter	Type	Description
`[prefix].lambda`	query	
`[prefix].mu`	query	
`[prefix].E`	query	
`[prefix].nu`	query	
`[prefix].yield`	query	
`[prefix].hardening`	query	
`[prefix].theta`	query	

Model::Solid::Affine::J2

This models an isotropic elastic-perfectly-plastic, non-time-dependent solid model.

The energy and derivatives are:

$$W = \frac{1}{2}(\varepsilon - \varepsilon_p) : \mathbb{C}(\varepsilon - \varepsilon_p) \tag{5.1}$$

$$DW = \mathbb{C}(\varepsilon - \varepsilon_p) \tag{5.2}$$

$$DDW = \mathbb{C} \tag{5.3}$$

where $\mathbb{C}$ is an isotropic `Set::Matrix4` and ε_p is is stored in the `F0` eigenstrain.

The plastic strain is evolved according to the following:

1. Calculate the deviatoric stress $\sigma_v = \sigma - \frac{1}{3}tr(\sigma)\mathbf{I}$
2. Calculate $J_2 = \sqrt{\frac{3}{2}\sigma_v : \sigma_v}$
3. If $J_2 < \sigma_0$ then quit - no plasticity occurs
4. Calculate $\Delta\sigma = (1 - \frac{\sigma_0}{J_2})$, which projects the stress back on the yield surface.
5. Convert to change in plastic strain, $\Delta\varepsilon = \mathbb{C}^{-1}\Delta\sigma$
6. Update plastic strain: $\varepsilon_p += \Delta\varepsilon$

Notes:

- This does not implement any kind of hardening model. Rate hardening, isotropic hardening, and kinematic hardening have yet to be implemneted.

Parameter	Type	Description
See also inputs to *Model::Solid::Affine::Isotropic*		
`[prefix].sigma0`	query	J2 Yield criterion

Model::Solid::Affine::J2PlasticDegradable

Parameter	Type	Description
`[prefix].lambda`	query	
`[prefix].mu`	query	
`[prefix].E`	query	
`[prefix].nu`	query	
`[prefix].yield`	query	
`[prefix].hardening`	query	
`[prefix].theta`	query	

5.5.2.2 Model::Solid::Elastic

Model::Solid::Elastic::NeoHookean

Parameter	Type	Description
`[prefix].mu`	query	
`[prefix].kappa`	query	

Model::Solid::Elastic::PseudoLinearCubic

Parameter	Type	Description
`[prefix].C11`	query	Elastic constant (default: 1.68)
`[prefix].C12`	query	Elastic constant (default: 1.21)
`[prefix].C44`	query	Elastic constant (default: 0.75)
`[prefix].phi1`	query	Bunge Euler angle ϕ_1
`[prefix].Phi`	query	Bunge Euler angle Φ
`[prefix].phi2`	query	Bunge Euler angle ϕ_2

5.5.2.3 Model::Solid::Linear

Model::Solid::Linear::Cubic

This class implements basic cubic elasticity. For a discussion on cubic elasticity, please see this link.

Parameter	Type	Description
[prefix].C11	query	Elastic constant (default: 1.68)
[prefix].C12	query	Elastic constant (default: 1.21)
[prefix].C44	query	Elastic constant (default: 0.75)
[prefix].phi1	query	Bunge Euler angle ϕ_1
[prefix].Phi	query	Bunge Euler angle Φ
[prefix].phi2	query	Bunge Euler angle ϕ_2

Model::Solid::Linear::Hexagonal

This class implements basic hexagonal elasticity.

Parameter	Type	Description
[prefix].C11	query	Elastic constant
[prefix].C12	query	Elastic constant
[prefix].C13	query	Elastic constant
[prefix].C33	query	Elastic constant
[prefix].C44	query	Elastic constant
[prefix].phi1	query	Bunge Euler angle ϕ_1
[prefix].Phi	query	Bunge Euler angle Φ
[prefix].phi2	query	Bunge Euler angle ϕ_2

Model::Solid::Linear::IsotropicDegradableTanh

Parameter	Type	Description
[prefix].E1	query	
[prefix].E2	query	
[prefix].Tg	query	
[prefix].Ts	query	
[prefix].nu	query	
[prefix].temp	query	

Model::Solid::Linear::Isotropic

This model implements an isotropic linear elastic material. See this link for more information about the theory.

Free energy for a linear material is defined as

$$W(\nabla \mathbf{u}) = \frac{1}{2} \nabla \mathbf{u} \cdot \mathbb{C} \nabla \mathbf{u}$$

For an isotropic material, stress and strain are related through

$$\mathbb{C}_{ijkl} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}$$

where λ and μ are the Lame constant and shear modulus, respectively. Users can specify these either through (`lame` and `shear`) OR (`lambda` and `mu`) OR (`E` and `nu`).

Class methods:

1. `Isotropic()`: Basic constructor. Does nothing, and leaves all values initiated as NAN.
2. `Isotropic(Solid<Set::Sym::Isotropic> base)` Basic constructor. Does nothing gut allows for inheritance.
3. `Isotropic(Set::Scalar a_mu, Set::Scalar a_lambda)` BAD old-fashioned constructor. Do not use!
4. `~Isotropic()` Simple destructor. Don't need to change it.
5. `void Define(Set::Scalar a_mu, Set::Scalar a_lambda)` BAD old-fashioned way of doing things. Use `Parse` instead.
6. `Set::Scalar W(const Set::Matrix & gradu) const override` Returns elastic free energy density
7. `Set::Matrix DW(const Set::Matrix & gradu) const override` Returns first derivative of free energy, the stress tensor
8. `Set::Matrix4<...> DDW(const Set::Matrix & ) const override` Returns second derivative of free energy, the modulus tensor
9. `virtual void Print(std::ostream &out) const override` Prints the modulus tensor object to output stream (usually the terminal)
10. `static Isotropic Random()` Static method that generates a random yet acceptable model.
11. `static Isotropic Zero()` Static method that generates a "zero" element (so that there is no effect under addition)
12. `static void Parse(Isotropic & value, IO::ParmParse & pp)` Parser where all the IO occurs

Parameter	Type	Description
`[prefix].planestress`	query	Whether or not to use the plane stress approximation.
`[prefix].lame`	query	Lame parameter
`[prefix].shear`	query	Shear modulus (redundant with "mu")
`[prefix].lambda`	query	Lame parameter
`[prefix].mu`	query	Shear modulus (redundant with "shear")
`[prefix].E`	query	Elastic modulus
`[prefix].nu`	query	Poisson's ratio

Model::Solid::Linear::IsotropicDegradable

Parameter	Type	Description
`[prefix].lambda`	query	
`[prefix].mu`	query	
`[prefix].E`	query	
`[prefix].nu`	query	

Model::Solid::Linear::Laplacian

Parameter	Type	Description

5.6 Numeric

5.6.1 Numeric::Interpolator

This namespace has evolved to contain some general Interpolator routines that are actually unrelated. The *Numeric::Interpolator::Linear* is an I/O utility. The *Numeric::Interpolator::NodeBilinear* is a low-level AMReX override that the average user (or even coder) will probably not need to touch.

5.6.1.1 Numeric::Interpolator::Linear

This is a general-purpose routine for specifying time-interpolated quantities in an input file. Interpolators obey the following syntax:

```
(t1,t2,..,tn;v1,v2,...,vn)
```

where `tn` are times and `vn` are values at those times.

Note: do not include whitespace in an interpolator, this will mess with the parser.

Interpolators can usually be used in place of regular numbers, but only when supported by the calling parse function.

For instance, the *BC::Operator::Elastic::TensionTest* allows the user to specify fixed displacement; for instance, to specify a value of 0.1:

```
bc.tenstion_test.disp = 0.1
```

However, it may be desirable to change the value over time. In this case, one may specify an interpolator string instead:

```
bc.tenstion_test.disp = (1.0,1.5,2.0;0.0,0.1,0.0)
```

This will cause the displacement value to be zero for `t<1.0`; ramp to `0.1` as `1.0<t<2.0`; ramp back to `0.0` as `2.0<t<3.0`; and then remain at the constant value of `3.0` for `t>3.0`.

Parameter	Type	Description
`[prefix].str`	query	Interpolator string used when Parsed from queryclass.

5.6.1.2 Numeric::Interpolator::NodeBilinear

Provide an interpolator function that can work with templated fields.

Warning: This is a low-level class used to add templating functionality to unerlying AMReX classes. Edit at your own risk!

5.7 Operator

5.7.1 Operator::Elastic

Parameter	Type	Description
`[prefix].small`	query	

5.8 Set

The Set namespace defines the Alamo datatypes. This includes scalars (`Set::Scalar`), vectors (`Set::Vector`), and nD tensors (`Set::Tensor`). Scalars are an alias for AMReX `Real`. Vectors and tensors are an alias for the Tuxfamily Eigen vectors and matrices, and so you can use Eigen operations on them (like `A.determinant()`, etc.) See the Eigen documentation for more.

The Set namespace defines the `Matrix4` datatype for fourth order tensors. These can have isotropic, major, minor, majorminor, etc., symmetries.

The Set namespace also defines the `Field` datatype, which is where most data is stored. It is templated so that `Set::Field<Set::Scalar>` is a field of scalars, `Set::Field<Set::Vector>` is a vector field, etc. One can also have, for instance, a field of models, like `Set::Field<Model::Solid::Linear::Isotropic>`.

5.9 Solver

5.9.1 Solver::Nonlocal

5.9.1.1 Solver::Nonlocal::Linear

Parameter	Type	Description
These are the parameters that are read in for a standard multigrid linear solve.		
[prefix].max_iter	query	Max number of iterations to perform before erroring out
[prefix].bottom_max_iter	query	Max number of iterations on the bottom solver
[prefix].max_fmg_iter	query	Max number of F-cycle iterations to perform
[prefix].fixed_iter	query	Number of fixed iterations to perform before exiting gracefully
[prefix].verbose	query	Verbosity of the solver (1-5)
[prefix].pre_smooth	query	Number of smoothing operations before bottom solve (2)
[prefix].post_smooth	query	Number of smoothing operations after bottom solve (2)
[prefix].final_smooth	query	Number of final smoothing operations when smoother is used as bottom solver (8)
[prefix].bottom_smooth	query	Additional smoothing after bottom CG solver (0)
[prefix].bottom_solver	query	The method that is used for the multigrid bottom solve (cg, bicgstab, smoother)
[prefix].bottom_tol_rel	query	Relative tolerance on bottom solver
[prefix].bottom_tol_abs	query	Absolute tolerance on bottom solver
[prefix].tol_rel	query	Relative tolerance
[prefix].tol_abs	query	Absolute tolerance
[prefix].omega	query	Omega (used in gauss-seidel solver)
[prefix].average_down_coeffs	query	Whether to average down coefficients or use the ones given. (Setting this to true is important for fracture.)
[prefix].normalize_ddw	query	Whether to normalize DDW when calculating the diagonal. This is primarily used when DDW is near-singular - like when there is a "void" region or when doing phase field fracture.
[prefix].dump_on_fail	query	[false] If set to true, output diagnostic multifab information whenever the MLMG solver fails to converge. (Note: you must also set amrex.signalhandling=0 and amrex.throw_exception=1 for this to work.)
[prefix].abort_on_fail	query	[true] If set to false, MLMG will not die if convergence criterion is not reached. (Note: you must also set amrex.signalhandling=0 and amrex.throw_exception=1 for this to work.)

5.9.1.2 Solver::Nonlocal::Newton

Parameter	Type	Description
These paramters control a standard Newton-Raphson solve. **Note**: This class inherits all of the linear solve paramters from its parent class, *Solver::Nonlocal::Linear*		
`[prefix].nriters`	query	Number of newton-raphson iterations.
`[prefix].nrtolerance`	query	

5.10 Util

CHAPTER

SIX

INPUTS SEARCH

Use the following box to search over all alamo inputs and descriptions. This is the same list as in the *Inputs Search* section as generated by the autodoc system.

Note If searching for documentation on an alamo command that you found in an input file, remember that the prefix may not be included in this table. For instance, if you are looking for documentation on the following inputs

```
hc.heat.alpha = 1.0
el.nmodels = 2
```

the prefixes `hc` and `el` are not necessarily included, and you will not find them if you do an exact search. Instead, do a reverse search, starting with `alpha` and `nmodels`, then work out the prefixes out.

Parameter	Namespace / Class	Description
[prefix].type.xlo	*BC::Constant*	BC type on the lower x edge (2d)
[prefix].type.xhi	*BC::Constant*	BC type on the upper x edge (2d)
[prefix].type.ylo	*BC::Constant*	BC type on the lower y edge (2d)
[prefix].type.yhi	*BC::Constant*	BC type on the upper y edge (2d)
[prefix].type.zlo	*BC::Constant*	BC type on the lower z face (proc
[prefix].type.zhi	*BC::Constant*	BC type on the upper z face (proc
[prefix].val.xlo	*BC::Constant*	BC value on the lower x edge (2c
[prefix].val.xhi	*BC::Constant*	BC value on the upper x edge (2c
[prefix].val.ylo	*BC::Constant*	BC value on the lower y edge (2c
[prefix].val.yhi	*BC::Constant*	BC value on the upper y edge (2c
[prefix].val.zlo	*BC::Constant*	BC value on the lower z face (pro
[prefix].val.zhi	*BC::Constant*	BC value on the upper z face (pro
[prefix].h1	*BC::Step*	Location of the step on the xlo ec
[prefix].h2	*BC::Step*	Location of the step on the xhi ec
[prefix].type.xloylozlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xloylozhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xloyhizlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xloyhizhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xhiylozlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xhiylozhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xhiyhizlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.xhiyhizhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].type.ylozlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.ylozhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.yhizlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.yhizhi	*BC::Operator::Elastic::Constant*	3D Edge

Parameter	Namespace / Class	Description
[prefix].type.zloxlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.zloxhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.zhixlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.zhixhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].type.xloylo	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].type.xloyhi	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].type.xhiylo	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].type.xhiyhi	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].type.xlo	*BC::Operator::Elastic::Constant*	3D Face 2D Edge
[prefix].type.xhi	*BC::Operator::Elastic::Constant*	3D Face 2D Edge
[prefix].type.ylo	*BC::Operator::Elastic::Constant*	3D Face 2D Edge
[prefix].type.yhi	*BC::Operator::Elastic::Constant*	3D Face 2D Edge
[prefix].type.zlo	*BC::Operator::Elastic::Constant*	3D Face
[prefix].type.zhi	*BC::Operator::Elastic::Constant*	3D Face
[prefix].val.xloylozlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xloylozhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xloyhizlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xloyhizhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xhiylozlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xhiylozhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xhiyhizlo	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.xhiyhizhi	*BC::Operator::Elastic::Constant*	3D Corner
[prefix].val.ylozlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.ylozhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.yhizlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.yhizhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.zloxlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.zloxhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.zhixlo	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.zhixhi	*BC::Operator::Elastic::Constant*	3D Edge
[prefix].val.xloylo	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].val.xloyhi	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].val.xhiylo	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].val.xhiyhi	*BC::Operator::Elastic::Constant*	3D Edge / 2D Corner
[prefix].val.xlo	*BC::Operator::Elastic::Constant*	3D Face / 2D Edge
[prefix].val.xhi	*BC::Operator::Elastic::Constant*	3D Face / 2D Edge
[prefix].val.ylo	*BC::Operator::Elastic::Constant*	3D Face / 2D Edge
[prefix].val.yhi	*BC::Operator::Elastic::Constant*	3D Face / 2D Edge
[prefix].val.zlo	*BC::Operator::Elastic::Constant*	3D Face
[prefix].val.zhi	*BC::Operator::Elastic::Constant*	3D Face
[prefix].type.xloylozlo	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xloylozhi	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xloyhizlo	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xloyhizhi	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xhiylozlo	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xhiylozhi	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xhiyhizlo	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.xhiyhizhi	*BC::Operator::Elastic::Expression*	3D Corner
[prefix].type.ylozlo	*BC::Operator::Elastic::Expression*	3D Edge

Parameter	Namespace / Class	Description
[prefix].type.ylozhi	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.yhizlo	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.yhizhi	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.zloxlo	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.zloxhi	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.zhixlo	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.zhixhi	*BC::Operator::Elastic::Expression*	3D Edge
[prefix].type.xloylo	*BC::Operator::Elastic::Expression*	3D Edge / 2D Corner
[prefix].type.xloyhi	*BC::Operator::Elastic::Expression*	3D Edge / 2D Corner
[prefix].type.xhiylo	*BC::Operator::Elastic::Expression*	3D Edge / 2D Corner
[prefix].type.xhiyhi	*BC::Operator::Elastic::Expression*	3D Edge / 2D Corner
[prefix].type.xlo	*BC::Operator::Elastic::Expression*	3D Face 2D Edge
[prefix].type.xhi	*BC::Operator::Elastic::Expression*	3D Face 2D Edge
[prefix].type.ylo	*BC::Operator::Elastic::Expression*	3D Face 2D Edge
[prefix].type.yhi	*BC::Operator::Elastic::Expression*	3D Face 2D Edge
[prefix].type.zlo	*BC::Operator::Elastic::Expression*	3D Face
[prefix].type.zhi	*BC::Operator::Elastic::Expression*	3D Face
[prefix].type	*BC::Operator::Elastic::TensionTest*	Tension test type. (Only option ri
[prefix].disp	*BC::Operator::Elastic::TensionTest*	Applied displacement (can be int
[prefix].trac	*BC::Operator::Elastic::TensionTest*	Applied traction (can be interpol
ic.n	*IC::Affine*	Normal vector for hyperplane
ic.alpha	*IC::Affine*	Value of the field on the positive
[prefix].filename	*IC::BMP*	BMP filename.
[prefix].fit	*IC::BMP*	How to fit. (options: stretch, fithe
[prefix].channel	*IC::BMP*	Color channel to use (options: r,
[prefix].min	*IC::BMP*	Scaling value - minimum (defaul
[prefix].max	*IC::BMP*	Scaling value - maximum (defaul
[prefix].value	*IC::Constant*	Array of constant values. The nu
ic.center	*IC::Cuboid*	Coordinates (X Y Z) of the cente
ic.length	*IC::Cuboid*	Lenth of the square/cube edges
[prefix].thickness	*IC::DoubleNotch*	Thickness of the notches
[prefix].width	*IC::DoubleNotch*	Width of the notches
[prefix].x0	*IC::DoubleNotch*	Center of the notches
[prefix].L	*IC::DoubleNotch*	Length of the notches
[prefix].coord	*IC::Expression*	coordinate system to use: “cartes
[prefix].center	*IC::Ellipsoid*	Center of the ellipse $\mathbf{x}_0$
[prefix].A	*IC::Ellipsoid*	Matrix defining elipse radii and c
[prefix].radius	*IC::Ellipsoid*	“Vector of radii (use instead of A
[prefix].eps	*IC::Ellipsoid*	Mollifying value for erf
[prefix].in_value	*IC::Ellipsoid*	Value of field inside ellipse (defa
[prefix].out_value	*IC::Ellipsoid*	Value of field outside ellipse (def
[prefix].mollifier	*IC::Ellipsoid*	Type of mollifier to use (options:
[prefix].x0	*IC::Ellipse*	Coorinates of ellipse center
[prefix].eps	*IC::Ellipse*	Diffuse boundary thickness
[prefix].A	*IC::Ellipse*	DxD square matrix defining an el
[prefix].a	*IC::Ellipse*	If A is not defined, then assume a
[prefix].number_of_inclusions	*IC::Ellipse*	Number of ellipses
[prefix].center	*IC::Ellipse*	center of the ellipse
[prefix].A	*IC::Ellipse*	either a vector containing ellipse

Parameter	Namespace / Class	Description
[prefix].A	*IC::Ellipse*	Same
[prefix].radius	*IC::Ellipse*	Array of radii [depricated]
[prefix].eps	*IC::Ellipse*	Regularization for smooth bound
[prefix].invert	*IC::Ellipse*	Flip the inside and the outside
[prefix].number_of_inclusions	*IC::Laminate*	How many laminates (MUST be
[prefix].center	*IC::Laminate*	(x,y,[z]) values for the center poi
[prefix].thickness	*IC::Laminate*	thickness of the laminate
[prefix].orientation	*IC::Laminate*	Vector normal to the interface of
[prefix].eps	*IC::Laminate*	Diffuse thickness
[prefix].mollifier	*IC::Laminate*	Type of mollifer to use (options:
[prefix].singlefab	*IC::Laminate*	Switch to mode where only one c
[prefix].invert	*IC::Laminate*	Take the complement of the lami
[prefix].center	*IC::Notch*	Center of notch
[prefix].orientation	*IC::Notch*	Vector describing notch orientati
[prefix].thickness	*IC::Notch*	Thickness of notch
[prefix].length	*IC::Notch*	Length of notch
[prefix].radius	*IC::Notch*	Radius of notch ends
[prefix].eps	*IC::Notch*	Magnitude of mollifier
[prefix].mollifier	*IC::Notch*	
[prefix].nspheres	*IC::PS*	
[prefix].matrix	*IC::PS*	
[prefix].inclusion	*IC::PS*	
[prefix].eps	*IC::PSRead*	Diffuseness of the sphere bounda
[prefix].filename	*IC::PSRead*	Location of .xyzr file
[prefix].wave_numbers	*IC::PerturbedInterface*	Wave numbers
[prefix].wave_amplitudes	*IC::PerturbedInterface*	Wave amplitudes
[prefix].normal	*IC::PerturbedInterface*	Which axis is normal to the inter
[prefix].offset	*IC::PerturbedInterface*	Interface offset from origin
[prefix].reverse	*IC::PerturbedInterface*	If true, flip the interface (default:
[prefix].mollifier	*IC::PerturbedInterface*	Mollifier (options: dirac, [gaussia
[prefix].eps	*IC::PerturbedInterface*	Magnitude of mollifier
[prefix].radius	*IC::Sphere*	Radius of the sphere
[prefix].center	*IC::Sphere*	Vector location of the sphere cen
[prefix].inside	*IC::Sphere*	Value of the field inside the spher
[prefix].outside	*IC::Sphere*	Value of the field outside teh sph
[prefix].type	*IC::Sphere*	Type - can be cylinder oriented a
[prefix].xs	*IC::TabulatedInterface*	
[prefix].ys	*IC::TabulatedInterface*	
[prefix].nr	*IC::Trig*	
[prefix].ni	*IC::Trig*	
[prefix].dim	*IC::Trig*	
[prefix].alpha	*IC::Trig*	
[prefix].number_of_grains	*IC::Voronoi*	
[prefix].alpha	*IC::Voronoi*	
[prefix].seed	*IC::Voronoi*	
max_step	*Integrator*	Number of iterations before endi
stop_time	*Integrator*	Simulation time before ending
timestep	*Integrator*	Nominal timestep on amrlev = 0
restart	*Integrator*	Name of restart file to READ fro

Parameter	Namespace / Class	Description
restart_cell	*Integrator*	Name of cell-fab restart file to re
restart_node	*Integrator*	Name of node-fab restart file to r
ignore	*Integrator*	Space-separated list of entries to
amr.regrid_int	*Integrator*	Regridding interval in step numb
amr.base_regrid_int	*Integrator*	Regridding interval based on coa
amr.plot_int	*Integrator*	Interval (in timesteps) between p
amr.plot_dt	*Integrator*	Interval (in simulation time) betw
amr.plot_file	*Integrator*	Output file
amr.cell.all	*Integrator*	Turn on to write all output in cell
amr.cell.any	*Integrator*	Turn off to prevent any cell based
amr.node.all	*Integrator*	Turn on to write all output in nod
amr.node.any	*Integrator*	Turn off to prevent any node base
amr.max_plot_level	*Integrator*	Specify a maximum level of refin
amr.nsubsteps	*Integrator*	Number of substeps to take on ea
amr.nsubsteps	*Integrator*	Number of substeps to take on ea
amr.thermo.int	*Integrator*	Default: integrate every time. Int
amr.thermo.plot_int	*Integrator*	Interval (in timesteps) between w
amr.thermo.plot_dt	*Integrator*	Interval (in simulation time) betw
explicitmesh.on	*Integrator*	Use explicit mesh instead of AM
crack.ic.type	*Integrator::Fracture*	
crack.ic.notch	*Integrator::Fracture*	
crack.ic.ellipsoid	*Integrator::Fracture*	
elastic.solver	*Integrator::Fracture*	Solver::Nonlocal::Linear<brittle_
pf.eps	*Integrator::Flame*	Burn width thickness
pf.kappa	*Integrator::Flame*	Interface energy param
pf.gamma	*Integrator::Flame*	Scaling factor for mobility
pf.lambda	*Integrator::Flame*	Chemical potential multiplier
pf.w1	*Integrator::Flame*	Unburned rest energy
pf.w12	*Integrator::Flame*	Barrier energy
pf.w0	*Integrator::Flame*	Burned rest energy
pf.P	*Integrator::Flame*	Pressure [UNITS?]
pf.r_ap	*Integrator::Flame*	AP Power law multiplier
pf.n_ap	*Integrator::Flame*	AP Power law exponent
pf.r_htpb	*Integrator::Flame*	HTPB Power law multiplier
pf.n_htpb	*Integrator::Flame*	HTPB Power law exponent
pf.r_comb	*Integrator::Flame*	Combination power law multiplie
pf.n_comb	*Integrator::Flame*	Combination power law exponen
pf.eta.bc	*Integrator::Flame*	See *BC::Constant*
eta.ic.type	*Integrator::Flame*	IC type - [packedspheres,laminat
thermal.on	*Integrator::Flame*	These parameters are for the **The**
thermal.rho1	*Integrator::Flame*	Density (before)
thermal.rho0	*Integrator::Flame*	Density (after)
thermal.ka	*Integrator::Flame*	Thermal conductivity (before and
thermal.kh	*Integrator::Flame*	Thermal conductivity (before and
thermal.k0	*Integrator::Flame*	Thermal conductivity (before and
thermal.cp1	*Integrator::Flame*	Specific heat (before and after)
thermal.cp0	*Integrator::Flame*	Specific heat (before and after)
thermal.delA	*Integrator::Flame*	Thermal flux of each material
thermal.delH	*Integrator::Flame*	Thermal flux of each material

Parameter	Namespace / Class	Description
amr.refinement_criterion	*Integrator::Flame*	Refinement criterion for eta field
amr.refinement_criterion_temp	*Integrator::Flame*	Refinement criterion for tempera
amr.refinament_restriction	*Integrator::Flame*	Eta value to restrict the refinamer
phi.ic.type	*Integrator::Flame*	IC type (psread, laminate, consta
model_ap	*Integrator::Flame*	
model_htpb	*Integrator::Flame*	
[prefix].heat.alpha	*Integrator::HeatConduction*	Diffusion coefficient [1.0]
[prefix].heat.refinement_threshold	*Integrator::HeatConduction*	Criterion for mesh refinement [0.
[prefix].ic.type	*Integrator::HeatConduction*	Initial condition type ([sphere], c
[prefix].bc.temp	*Integrator::HeatConduction*	*BC::Constant* parameters with pr
[prefix].nmodels	*Integrator::Mechanics*	Number of elastic model varietie
[prefix].eta_ref_threshold	*Integrator::Mechanics*	Refinement threshold for eta field
[prefix].ref_threshold	*Integrator::Mechanics*	Refinement threshold for strain g
[prefix].ic.type	*Integrator::Mechanics*	Read IC type for the eta field
[prefix].eta.reset_on_regrid	*Integrator::Mechanics*	Whether to re-initialize eta when
[prefix].psi.ic.type	*Integrator::Mechanics*	Read IC type for the eta field
[prefix].psi.reset_on_regrid	*Integrator::Mechanics*	Whether to re-initialize psi when
water.on	*Integrator::PolymerDegradation*	
water.diffusivity	*Integrator::PolymerDegradation*	Diffusivity
water.refinement_threshold	*Integrator::PolymerDegradation*	AMR refinement criterion
water.ic_type	*Integrator::PolymerDegradation*	
water.ic.value	*Integrator::PolymerDegradation*	
water.ic.bc	*Integrator::PolymerDegradation*	
[prefix].pf.number_of_grains	*Integrator::PhaseFieldMicrostructure*	Number of grain fields (may be n
[prefix].pf.M	*Integrator::PhaseFieldMicrostructure*	Mobility
[prefix].pf.mu	*Integrator::PhaseFieldMicrostructure*	Phase field μ
[prefix].pf.gamma	*Integrator::PhaseFieldMicrostructure*	Phase field γ
[prefix].pf.sigma0	*Integrator::PhaseFieldMicrostructure*	Initial GB energy if not using GB
[prefix].pf.l_gb	*Integrator::PhaseFieldMicrostructure*	Mobility
[prefix].pf.elastic_df	*Integrator::PhaseFieldMicrostructure*	Determine whether to use elastic
[prefix].pf.elastic_mult	*Integrator::PhaseFieldMicrostructure*	Multiplier of elastic energy
[prefix].pf.elastic_threshold	*Integrator::PhaseFieldMicrostructure*	Elastic threshold (ϕ_0)
[prefix].amr.max_level	*Integrator::PhaseFieldMicrostructure*	Maximum AMR level
[prefix].amr.ref_threshold	*Integrator::PhaseFieldMicrostructure*	Phase field refinement threshold
[prefix].mechanics.tstart	*Integrator::PhaseFieldMicrostructure*	Elasticity
[prefix].mechanics.model	*Integrator::PhaseFieldMicrostructure*	By default, read in the model spe
[prefix].lagrange.on	*Integrator::PhaseFieldMicrostructure*	Lagrange multiplier method for e
[prefix].lagrange.lambda	*Integrator::PhaseFieldMicrostructure*	Lagrange multiplier value
[prefix].lagrange.vol0	*Integrator::PhaseFieldMicrostructure*	Prescribed volume
[prefix].lagrange.tstart	*Integrator::PhaseFieldMicrostructure*	Time to start enforcing Lagrange
[prefix].anisotropy.on	*Integrator::PhaseFieldMicrostructure*	Anisotropic grain boundary energ
[prefix].anisotropy.beta	*Integrator::PhaseFieldMicrostructure*	Regularization param
[prefix].anisotropy.tstart	*Integrator::PhaseFieldMicrostructure*	Time to turn on anisotropy
[prefix].anisotropy.timestep	*Integrator::PhaseFieldMicrostructure*	Modify timestep when turned on
[prefix].anisotropy.plot_int	*Integrator::PhaseFieldMicrostructure*	Modify plot_int when turned on
[prefix].anisotropy.plot_dt	*Integrator::PhaseFieldMicrostructure*	Modify plot_dt when turned on
[prefix].anisotropy.thermo_int	*Integrator::PhaseFieldMicrostructure*	Modify thermo int when turned o
[prefix].anisotropy.thermo_plot_int	*Integrator::PhaseFieldMicrostructure*	Modify thermo plot int when turn
[prefix].anisotropy.elastic_int	*Integrator::PhaseFieldMicrostructure*	Frequency of elastic calculation

Parameter	Namespace / Class	Description
[prefix].anisotropy.regularization	*Integrator::PhaseFieldMicrostructure*	Type of regularization to use
[prefix].anisotropy.gb_type	*Integrator::PhaseFieldMicrostructure*	Set the anisotropic GB model Ty
[prefix].bc.eta.type	*Integrator::PhaseFieldMicrostructure*	Type (constnat)
[prefix].ic.type	*Integrator::PhaseFieldMicrostructure*	IC Type
crack.ic.type	*Integrator::SutureCrack*	
crack.ic.notch	*Integrator::SutureCrack*	
[prefix].hc	*Integrator::ThermoElastic*	
[prefix].el	*Integrator::ThermoElastic*	
[prefix].alpha	*Integrator::ThermoElastic*	
[prefix].model	*Integrator::TopOp*	
[prefix].psi.ic.type	*Integrator::TopOp*	Read IC type for the eta field
[prefix].eta_ref_threshold	*Integrator::TopOp*	
[prefix].alpha	*Integrator::TopOp*	
[prefix].beta	*Integrator::TopOp*	
[prefix].gamma	*Integrator::TopOp*	
[prefix].L	*Integrator::TopOp*	
[prefix].volume0frac	*Integrator::TopOp*	
[prefix].volume0	*Integrator::TopOp*	
[prefix].lambda	*Integrator::TopOp*	
[prefix].type	*Integrator::Base::Mechanics*	
[prefix].time_evolving	*Integrator::Base::Mechanics*	
[prefix].solver	*Integrator::Base::Mechanics*	Read parameters for *Solver::Non*
[prefix].viscous.mu	*Integrator::Base::Mechanics*	
[prefix].bc.type	*Integrator::Base::Mechanics*	Determine the boundary conditio
[prefix].print_model	*Integrator::Base::Mechanics*	
[prefix].rhs.type	*Integrator::Base::Mechanics*	Initializer for RHS
[prefix].interval	*Integrator::Base::Mechanics*	Timestep interval for elastic solve
[prefix].max_coarsening_level	*Integrator::Base::Mechanics*	Maximum multigrid coarsening l
[prefix].print_residual	*Integrator::Base::Mechanics*	
[prefix].elastic_ref_threshold	*Integrator::Base::Mechanics*	Whether to refine based on elasti
[prefix].zero_out_displacement	*Integrator::Base::Mechanics*	Set this to true to zero out the dis
[prefix].G_c	*Model::Interface::Crack::Constant*	
[prefix].zeta	*Model::Interface::Crack::Constant*	
[prefix].mobility	*Model::Interface::Crack::Constant*	
[prefix].threshold	*Model::Interface::Crack::Constant*	
[prefix].gtype	*Model::Interface::Crack::Constant*	
[prefix].wtype	*Model::Interface::Crack::Constant*	
[prefix].exponent	*Model::Interface::Crack::Constant*	
[prefix].Gc0	*Model::Interface::Crack::Sin*	
[prefix].Gc1	*Model::Interface::Crack::Sin*	
[prefix].theta0	*Model::Interface::Crack::Sin*	
[prefix].zeta	*Model::Interface::Crack::Sin*	
[prefix].mobility	*Model::Interface::Crack::Sin*	
[prefix].threshold	*Model::Interface::Crack::Sin*	
[prefix].gtype	*Model::Interface::Crack::Sin*	
[prefix].wtype	*Model::Interface::Crack::Sin*	
[prefix].exponent	*Model::Interface::Crack::Sin*	
[prefix].theta0	*Model::Interface::GB::AbsSin*	
[prefix].sigma0	*Model::Interface::GB::AbsSin*	convert degrees into radians

Parameter	Namespace / Class	Description
[prefix].sigma1	*Model::Interface::GB::AbsSin*	
[prefix].filename	*Model::Interface::GB::Read*	
[prefix].theta0	*Model::Interface::GB::SH*	
[prefix].phi0	*Model::Interface::GB::SH*	convert degrees into radians
[prefix].sigma0	*Model::Interface::GB::SH*	convert degrees into radians
[prefix].sigma1	*Model::Interface::GB::SH*	
[prefix].regularization	*Model::Interface::GB::SH*	
[prefix].theta0	*Model::Interface::GB::Sin*	
[prefix].sigma0	*Model::Interface::GB::Sin*	convert degrees into radians
[prefix].sigma1	*Model::Interface::GB::Sin*	
[prefix].n	*Model::Interface::GB::Sin*	
[prefix].C11	*Model::Solid::Affine::CubicDegradable*	Original, undegraded $\mathbb{C}_{11}$
[prefix].C12	*Model::Solid::Affine::CubicDegradable*	Original, undegraded $\mathbb{C}_{12}$
[prefix].C44	*Model::Solid::Affine::CubicDegradable*	Original, undegraded $\mathbb{C}_{44}$
[prefix].phi1	*Model::Solid::Affine::CubicDegradable*	Bunge Euler angles ϕ_1
[prefix].Phi	*Model::Solid::Affine::CubicDegradable*	Bunge Euler angles Φ
[prefix].phi2	*Model::Solid::Affine::CubicDegradable*	Bunge Euler angles ϕ_2
[prefix].F0	*Model::Solid::Affine::Cubic*	Eigenstrain
[prefix].F0	*Model::Solid::Affine::Hexagonal*	Eigenstrain
[prefix].lame	*Model::Solid::Affine::Isotropic*	Lame modulus
[prefix].shear	*Model::Solid::Affine::Isotropic*	Shear modulus
[prefix].E	*Model::Solid::Affine::Isotropic*	Elastic modulus
[prefix].nu	*Model::Solid::Affine::Isotropic*	Poisson's ratio
[prefix].E	*Model::Solid::Affine::Isotropic*	
[prefix].shear	*Model::Solid::Affine::Isotropic*	
[prefix].E	*Model::Solid::Affine::Isotropic*	
[prefix].lame	*Model::Solid::Affine::Isotropic*	
[prefix].nu	*Model::Solid::Affine::Isotropic*	
[prefix].shear	*Model::Solid::Affine::Isotropic*	
[prefix].lambda	*Model::Solid::Affine::Isotropic*	
[prefix].nu	*Model::Solid::Affine::Isotropic*	
[prefix].bulk	*Model::Solid::Affine::Isotropic*	
[prefix].shear	*Model::Solid::Affine::Isotropic*	
[prefix].F0	*Model::Solid::Affine::Isotropic*	Eigendeformation gradient
[prefix].lame	*Model::Solid::Affine::IsotropicDegradable*	
[prefix].shear	*Model::Solid::Affine::IsotropicDegradable*	
[prefix].E	*Model::Solid::Affine::IsotropicDegradable*	
[prefix].nu	*Model::Solid::Affine::IsotropicDegradable*	
[prefix].lambda	*Model::Solid::Affine::J2Plastic*	
[prefix].mu	*Model::Solid::Affine::J2Plastic*	
[prefix].E	*Model::Solid::Affine::J2Plastic*	
[prefix].nu	*Model::Solid::Affine::J2Plastic*	
[prefix].yield	*Model::Solid::Affine::J2Plastic*	
[prefix].hardening	*Model::Solid::Affine::J2Plastic*	
[prefix].theta	*Model::Solid::Affine::J2Plastic*	
[prefix].sigma0	*Model::Solid::Affine::J2*	J2 Yield criterion
[prefix].lambda	*Model::Solid::Affine::J2PlasticDegradable*	
[prefix].mu	*Model::Solid::Affine::J2PlasticDegradable*	
[prefix].E	*Model::Solid::Affine::J2PlasticDegradable*	

Parameter	Namespace / Class	Description
[prefix].nu	*Model::Solid::Affine::J2PlasticDegradable*	
[prefix].yield	*Model::Solid::Affine::J2PlasticDegradable*	
[prefix].hardening	*Model::Solid::Affine::J2PlasticDegradable*	
[prefix].theta	*Model::Solid::Affine::J2PlasticDegradable*	
[prefix].mu	*Model::Solid::Elastic::NeoHookean*	
[prefix].kappa	*Model::Solid::Elastic::NeoHookean*	
[prefix].C11	*Model::Solid::Elastic::PseudoLinearCubic*	Elastic constant (default: 1.68)
[prefix].C12	*Model::Solid::Elastic::PseudoLinearCubic*	Elastic constant (default: 1.21)
[prefix].C44	*Model::Solid::Elastic::PseudoLinearCubic*	Elastic constant (default: 0.75)
[prefix].phi1	*Model::Solid::Elastic::PseudoLinearCubic*	Bunge Euler angle ϕ_1
[prefix].Phi	*Model::Solid::Elastic::PseudoLinearCubic*	Bunge Euler angle Φ
[prefix].phi2	*Model::Solid::Elastic::PseudoLinearCubic*	Bunge Euler angle ϕ_2
[prefix].C11	*Model::Solid::Linear::Cubic*	Elastic constant (default: 1.68)
[prefix].C12	*Model::Solid::Linear::Cubic*	Elastic constant (default: 1.21)
[prefix].C44	*Model::Solid::Linear::Cubic*	Elastic constant (default: 0.75)
[prefix].phi1	*Model::Solid::Linear::Cubic*	Bunge Euler angle ϕ_1
[prefix].Phi	*Model::Solid::Linear::Cubic*	Bunge Euler angle Φ
[prefix].phi2	*Model::Solid::Linear::Cubic*	Bunge Euler angle ϕ_2
[prefix].C11	*Model::Solid::Linear::Hexagonal*	Elastic constant
[prefix].C12	*Model::Solid::Linear::Hexagonal*	Elastic constant
[prefix].C13	*Model::Solid::Linear::Hexagonal*	Elastic constant
[prefix].C33	*Model::Solid::Linear::Hexagonal*	Elastic constant
[prefix].C44	*Model::Solid::Linear::Hexagonal*	Elastic constant
[prefix].phi1	*Model::Solid::Linear::Hexagonal*	Bunge Euler angle ϕ_1
[prefix].Phi	*Model::Solid::Linear::Hexagonal*	Bunge Euler angle Φ
[prefix].phi2	*Model::Solid::Linear::Hexagonal*	Bunge Euler angle ϕ_2
[prefix].E1	*Model::Solid::Linear::IsotropicDegradableTanh*	
[prefix].E2	*Model::Solid::Linear::IsotropicDegradableTanh*	
[prefix].Tg	*Model::Solid::Linear::IsotropicDegradableTanh*	
[prefix].Ts	*Model::Solid::Linear::IsotropicDegradableTanh*	
[prefix].nu	*Model::Solid::Linear::IsotropicDegradableTanh*	
[prefix].temp	*Model::Solid::Linear::IsotropicDegradableTanh*	
[prefix].planestress	*Model::Solid::Linear::Isotropic*	Whether or not to use the plane s
[prefix].lame	*Model::Solid::Linear::Isotropic*	Lame parameter
[prefix].shear	*Model::Solid::Linear::Isotropic*	Shear modulus (redundant with "
[prefix].lambda	*Model::Solid::Linear::Isotropic*	Lame parameter
[prefix].mu	*Model::Solid::Linear::Isotropic*	Shear modulus (redundant with "
[prefix].E	*Model::Solid::Linear::Isotropic*	Elastic modulus
[prefix].nu	*Model::Solid::Linear::Isotropic*	Poisson's ratio
[prefix].lambda	*Model::Solid::Linear::IsotropicDegradable*	
[prefix].mu	*Model::Solid::Linear::IsotropicDegradable*	
[prefix].E	*Model::Solid::Linear::IsotropicDegradable*	
[prefix].nu	*Model::Solid::Linear::IsotropicDegradable*	
[prefix].str	*Numeric::Interpolator::Linear*	Interpolator string used when Par
[prefix].small	*Operator::Elastic*	
[prefix].max_iter	*Solver::Nonlocal::Linear*	Max number of iterations to perf
[prefix].bottom_max_iter	*Solver::Nonlocal::Linear*	Max number of iterations on the
[prefix].max_fmg_iter	*Solver::Nonlocal::Linear*	Max number of F-cycle iterations
[prefix].fixed_iter	*Solver::Nonlocal::Linear*	Number of fixed iterations to per

Parameter	Namespace / Class	Description
[prefix].verbose	*Solver::Nonlocal::Linear*	Verbosity of the solver (1-5)
[prefix].pre_smooth	*Solver::Nonlocal::Linear*	Number of smoothing operations
[prefix].post_smooth	*Solver::Nonlocal::Linear*	Number of smoothing operations
[prefix].final_smooth	*Solver::Nonlocal::Linear*	Number of final smoothing opera
[prefix].bottom_smooth	*Solver::Nonlocal::Linear*	Additional smoothing after botto
[prefix].bottom_solver	*Solver::Nonlocal::Linear*	The method that is used for the m
[prefix].bottom_tol_rel	*Solver::Nonlocal::Linear*	Relative tolerance on bottom solv
[prefix].bottom_tol_abs	*Solver::Nonlocal::Linear*	Absolute tolerance on bottom sol
[prefix].tol_rel	*Solver::Nonlocal::Linear*	Relative tolerance
[prefix].tol_abs	*Solver::Nonlocal::Linear*	Absolute tolerance
[prefix].omega	*Solver::Nonlocal::Linear*	Omega (used in gauss-seidel solv
[prefix].average_down_coeffs	*Solver::Nonlocal::Linear*	Whether to average down coeffici
[prefix].normalize_ddw	*Solver::Nonlocal::Linear*	Whether to normalize DDW whe
[prefix].dump_on_fail	*Solver::Nonlocal::Linear*	[false] If set to true, output diagn
[prefix].abort_on_fail	*Solver::Nonlocal::Linear*	[true] If set to false, MLMG will
[prefix].nriters	*Solver::Nonlocal::Newton*	Number of newton-raphson iterat
[prefix].nrtolerance	*Solver::Nonlocal::Newton*	

CHAPTER

SEVEN

</> DEVELOPER GUIDE

7.1 Step-by-step development guide

This is a quick tutorial intended for new Alamo developers to get up to speed with contributing to the Alamo code base.

1. **Create Github credentials.**
    1. If you do not have one already, create a new GitHub account, and email your username to brunels@iastate.edu to request push access
    2. Follow these Github instructions to add an SSH key to your account.
    3. important Make sure your repository points to the SSH version of Alamo. You can check this with the command

    ```
    git remote show origin
    ```

    If the output begins with

    ```
    Fetch URL: git@github.com:solidsgroup/alamo.git
    Push  URL: git@github.com:solidsgroup/alamo.git
    ```

    then you have configured correctly check. However, if the output begins with

    ```
    Fetch URL: https://github.com/solidsuccs/alamo.git
    Push  URL: https://github.com/solidsuccs/alamo.git
    ```

    this means that you are using HTTPS authentication, which will not allow you to push. To fix, run

    ```
    git remote remove origin
    git remote add origin git@github.com:solidsgroup/alamo.git
    ```

    Run a quick `git pull` to make sure that you still have access. If you get an authentication error, this likely means that your SSH key is not configured correctly.
2. **Create a new branch.** You can create a branch from the terminal, or you can use the Github online interface. Follow these guidlines when naming your branch
    - Use combinations of lowercase letters and hyphens, avoid mixed case, numbers, and underscores (`gas-combustion` good, `GasCombustion2` bad).
    - Be as descriptive as possible and avoid overly general names (`gb-damage-model` good, `model1` / `mymodel` / `foo` bad). Long is ok.
    - Keep it professional. No profanity or excessive whimsy in branch names.

If you have created a branch (e.g. `mytestbranch`) locally, make sure to run

```
git push --set-upstream origin mytestbranch
```

to make pushing easier.

3. **Create a capability input file**. This is an input file that is designed to demonstrate the functionality of your new code. The capability input file should always be located in the alamo root directory. The `plot_file` should always be `output`. Once you are ready to merge your code into development, convert your input file to a regression test and remove it from the root. (See the Autodoc and Autotest section).

4. **Use an EditorConfig-compliant text editor.** Alamo uses EditorConfig to maintain code formatting standards. VS Code is recommended, as it supports multiple keybinding schemes and automatically supports editorconfig.

5. **Write your code.** As you are writing, keep the following in mind:

always	Commit your code - at least once per day you are coding, even if the code is not working. There are multiple great tutorials if you are not sure how to commit your code.
always	Write meaningful commit messages. This will help you more than you may realize when you are trying to debug your code later.
always	Follow the development guide below and stick to the specified convention.
regularly	Merge the latest development into your code (see below for more details). The more frequently you do this, the less painful your life will be later.
regularly	Run the regression test suite with `make test`. This helps to ensure that you are not breaking your (or someone else's) code.
regularly	Ensure that your commits pass the commit tests. Sometimes this is not possible, especially if your code is in active development, but the more you keep your code up to standard, the easier your life will be later.
regularly	Document your code. Use the autodoc system to ensure a base level of documentation.
rarely	Break backwards compatibility. If you are implementing an improved version of a model that is better than the legacy model, leave the option to run using the existing model. This is important because publications may have used the legacy model, and may need the legacy model to reproduce important results. Backwards compatibility should only be broken when implementing fundamental new features. Even then, changes required to reproduce legacy results should be kept minimal.
never	Modify the GitHub actions to get your branch to pass the tests. Email failure notifications may be annoying, but they are there for a reason.
never	Commit large or unnecessary files to the repository. Use `git status` liberally, before and after `git commit`, before you push. Edit your gitignore file as needed.
never	Commit sensitive content to the repository. This is usually an issue only if your project is export controlled, which is rare. But when in doubt, ask.

6. **Merge your branch into development** To include your changes into the development branch, submit a pull request from your branch into development. This will trigger a review before the merge can be approved. (Note that all tests must pass before the merge will be reviewed.) The more frequently you merge in the latest from development, the easier your merge will be.

   How do I know my code is ready to merge? You should always merge your code once you know that it works, but sometimes this is a gray area. You should always merge your code if you are submitting a publication, or if you are a PhD student about to graduate.

7.2 Tutorial: A new Integrator

Integrators are the basic workhorse in Alamo. They are called Integrators because they usually (although not always) are intended to integrate a PDE in time. In this tutorial, we will create a new integrator based on the existing HeatConduction integrator.

1. Create a copy of `./src/Integrator/HeatConduction.H` inside the same directory, called *MyPDE.H* (We will use the working name "MyPDE" here - you can replace with your own name.)
2. Rename all instances of HeatConduction with MyPDE inside `./src/Integrator/MyPDE.H`. This includes the names throughout the code as well as the include guard in the first two lines.
3. Include the file in `./src/alamo.cc`

   ```
   #include "Integrator/MyPDE.H"
   ```

   and add a caller inside the main function:

   ```
   //existing
   else if (program == "thermoelastic")integrator = new Integrator::ThermoElastic(pp);
   //new
   else if (program == "mypde")        integrator = new Integrator::MyPDE(pp);
   ```

4. Finally copy the Heat Conduction example file to the root directory

   ```
   cp ./tests/HeatConduction/input ./input
   ```

   In the input file change the `plot_file` to *output* and `alamo.program` to `mypde`.
5. You should now be able to compile and run the code. The result will be the same as for the heat conduction example, but will be based on the newly copied integrator.
6. You can now begin alternating the code to achieve different outputs. The HeatConduction integrator has extensive, line-by-line documentation.

7.3 Conventions

7.3.1 Namespaces

Classes and structures are organized using namespaces. Namespaces should be brief but descriptive, and two nested namespaces should correspond to two nested subsets. For instance `Shape::Polygon::Triangle` is acceptable whereas `Shape::Circle::Round` is not (since "round" is not a subset of "Circle").

The directory structure must reflect the namespace structure. For instance `Shape::Polygon::Triangle` must be declared in `src/Shape/Polygon/Triangle.H`.

7.3.1.1 Regular and Pure Virtual classes

Pure virtual classes must have the same name as the namespace in which they are contained. For instance `Model::Elastic::Elastic` is pure virtual, whereas `Model::Elastic::Isotropic` must inherit from `Model::Elastic::Elastic` and is not pure virtual.

7.3.2 Include Guards

Include guards should be all uppercase, and should be the same as the path (relative to *src/*) with / replaced by underscores. For instance: a file called *src/Integrator/Integrator.H* should have the include guard

```
#ifndef INTEGRATOR_INTEGRATOR_H
#define INTEGRATOR_INTEGRATOR_H
...
#endif
```

7.3.3 Member and Argument Variable Names

Members of a class should have names beginning with `m_`; argument variables should begin with `a_`. For instance:

```
class MyClass
{
public:
   MyClass(a_myvariable) : m_myvariable(a_myvariable)
   {}
private:
   m_myvariable;
}
```

7.4 Python (In development)

Alamo supports a (currently limited) Python interface. It can be run as simply as

```
python alamo.py
```

where `alamo.py` is a python script that you write. An example (that currently works) is

```
import alamo

alamo.Util.Initialize()

mytest = alamo.Test.Operator.Elastic()

mytest.setBounds([1.0,1.0,1.0])

for lev in [1,2,3]:
    print("Levels of refinement: " + str(lev))
mytest.Define([32,32,32],lev,1,mytest.Grid.XYZ)
failed = mytest.TrigTest(0,0,1,"")
```

(continues on next page)

(continued from previous page)

```
if failed: print("Test failed")
else: print ("Test passed")

alamo.Util.Finalize()
```

Note that the C++ namespace structure is mirrored in the Python bindings.

Warning: Note that object constructors require `()`, e.g. `test = alamo.Test.Operator.Elastic()`. If you forget the `()` the python interpreter **will not complain!** Instead, it will give you an extremely cryptic message about member methods not having the right number of arguments. A typical error message might look like this:

```
Traceback (most recent call last):
File "alamo.py", line 7, in <module>
mytest.setBounds([1.0,1.0,1.0])
TypeError: unbound method Boost.Python.function object must be called with Elastic␣
↪instance as first argument (got list instance instead)
```

7.4.1 Compiling Alamo Python interface

The python interface requires all code to be compiled using the `-fPIC` flag. To compile AMReX,

```
./compile [whatever flags you would normally use
make USE_COMPILE_PIC=TRUE
make install
```

To compile Alamo, you need the Boost python library. On ubuntu, install with

```
sudo apt install libboost-python-dev
```

Then to compile Alamo,

```
./configure [whatever flags you would normally use] --python
make python
```

This will create a file called `alamo.so` in the Alamo root directory. (Note that this file must be in your python path (or your current directory) to use.)

7.4.2 Extending the Alamo Python interface

Python bindings are currently limited and you will likely need to add them if you want them for a specific component. Alamo uses the Boost Python library to create the Python bindings. All of this code is located in the `py/` directory, and mimics the directory structure of `src/`. These are all C++ files, but use the `.cpy` extension. To add bindings (e.g. to an object) you must

1. Create an appropriately named file (e.g. `py/Test/Operator/Elastic.cpy` to bind `Test::Operator::Elastic`).
2. Define a function that encapsulates the bindings (e.g. `void exportTestOperatorElastic`).
3. Write the Boost Python bindings in the function. Keep in mind that you need to specify namespace explicitly.
4. Include the file in `py/alamo.cpy`

5. Call the function in `py/alamo.cpy`

7.5 Restart files

You can restart a simulation from any point at which you have dumped an output file. For instance, suppose you have run alamo with this command

```
./bin/alamo-2d-g++ input
```

and it produced the following output.

```
./output
    00000cell 00000node
    00010cell 00010node
    00020cell 00020node
    00030cell 00030node
    00040cell 00040node
    00050cell 00050node
    00060cell 00060node
    00070cell 00070node
    celloutput.visit
    nodeoutput.visit
    metadata
```

You may wish to *restart* the simulation without starting from the beginning. Perhaps the simulation was fine at `00060` but became unstable at `00070`, and you want to continue from that point with a different parameter value. To do that, you can simply run:

```
./bin/alamo-2d-g++ input restart=./output/00060cell
```

for the simulation to pick up where it left off, but with updated values (or updated code) that may have changed since you first ran it.

It is important to distinguish what the restart function does and does not do.

The restart function *DOES*:

- Import all of the data from fields named "XXX" "YYY" etc into corresponding fields with the same names in the new simulation.
- Set the clock equal to the time corresponding to the output file.
- Set the current timestep equal to that corresponding to the output file.

The restart function DOES NOT:

- Set any other simulation parameters. The input file is responsible for setting all simulation parameters. It is up to you to use them (or change them) as needed.
- Initialize non-matching data. If you try to restart using data containing fields "AAA" and "BBB" but your simulation expects fields "BBB" and "CCC", only "BBB" will be initialized - the rest is undefined.
- Continue to output to the same directory. A different directory will be created. Only the metadata file will record where the restard data came from.