

Verification of PDE Discretization Library and Flow Application Codes

Roy H. Stogner

The University of Texas at Austin

Jul 31, 2009

3D Hypersonic Flow Approximations

Math, Software Components

- Discretized Formulation
- Spatial Discretization
- Time Discretization
- System Assembly
- Nonlinear Solver
- Linear Solver
- I/O
- Postprocessing

Approximate LoC

FIN-S	6,000
libMesh	54,000
PETSc	170,000
Other	56,000

3D Hypersonic Flow Approximations

Components To Validate

- Discretized Formulation
- Spatial Discretization
- Time Discretization
- System Assembly
- Nonlinear Solver
- Linear Solver
- I/O
- Postprocessing

Potentially Buggy LoC

FIN-S	6,000
libMesh	54,000
PETSc	170,000
Other	56,000

Modular Software Development

Separate Components

- Linear, nonlinear algebraic solvers are independent of discretization
- System assembly, some solution I/O & postprocessing can be independent of discretization
- Time, space discretizations are independent of PDE
- Some error analysis, sensitivity methods can be independent of PDE

Reusable components get re-tested with each re-use.

Errors too subtle to notice in solutions to complex physics are easy to spot in benchmark problem results.

Regression Tests

Rebuild Libraries, Rerun Tests

- Example application codes double as test cases
- Catches changes' "unintended consequences"
- Continuous Build System automation
 - ▶ Tests previously run "by hand" by `libMesh` developers
 - ▶ `libMesh`, FIN-S basic tests now in BuildBot at UT
 - ▶ `libMesh`, application tests in Trac/Bitten at INL

Regression Tests

Rebuild Libraries, Rerun Tests

- Example application codes double as test cases
- Catches changes' "unintended consequences"
- Continuous Build System automation
 - ▶ Tests previously run "by hand" by libMesh developers
 - ▶ libMesh, FIN-S basic tests now in BuildBot at UT
 - ▶ libMesh, application tests in Trac/Bitten at INL

Has Found

- Regressions in non-standard compile-time options
 - ▶ Complex-valued solutions, Infinite Elements
 - ▶ AMR, MPI, tracefiles, GMV, etc. disabled
- Internal use of deprecated APIs

High-level Assertions

`libmesh_assert`, PETSc debug mode

- Active only in “debug” runs
- Function preconditions
 - ▶ Are function arguments all valid?
- Function postconditions
 - ▶ Does function result satisfy requirements?
- Approx. 4000 assertions in `libMesh`, FIN-S

High-level Assertions

`libmesh_assert`, PETSc debug mode

- Active only in “debug” runs
- Function preconditions
 - ▶ Are function arguments all valid?
- Function postconditions
 - ▶ Does function result satisfy requirements?
- Approx. 4000 assertions in `libMesh`, FIN-S

Has Found **MANY** Library Bugs

- Uninitialized data
- Unpartitioned elements
- “Tearing” in neighbor map reconstruction
- Parallel vector operation miscommunication
- Parallel mesh adaptivity synchronization failures
- Out-of-order API calls
- $N_{elem} < N_{proc}$ bugs
- Bad I/O node numbering
- Unsupported I/O format features
- Failure to “deep copy” a Mesh

High-level Assertions Examples

```
libmesh_assert(c < _variables.size());
libmesh_assert(s < elem->n_sides());
libmesh_assert((ig >= Ug.first_local_index()) &&
               (ig < Ug.last_local_index()));
libmesh_assert(requested_ids[p].size() == ghost_objects_from_proc[p]);
libmesh_assert(obj_procid != DofObject::invalid_processor_id);
MeshTools::libmesh_assert_valid_node_procids(mesh);
libmesh_assert(neigh->has_children());
libmesh_assert(this->closed());
libmesh_assert(this->initialized());
libmesh_assert(mesh.is_prepared());
libmesh_assert(error_estimator.error_norm.type(var) == H1_SEMINORM ||
               error_estimator.error_norm.type(var) == W1_INF_SEMINORM);
libmesh_assert(number_h_refinements > 0 || number_p_refinements > 0);
```

High-level Assertions

Standard `assert` macro is sufficient

Prints failed assertion, prints file/line numbers, exits

`libmesh_assert` macro now adds:

- Per-processor stack trace files
- C++ exception thrown

High-level Assertions

Standard `assert` macro is sufficient

Prints failed assertion, prints file/line numbers, exits

`libmesh_assert` macro now adds:

- Per-processor stack trace files
- C++ exception thrown

Has Found **COUNTLESS** Application Bugs

- API mistakes
 - ▶ Misordered function args
 - ▶ Sharing non-shareable objects
 - ▶ Querying data before calculating it
 - ▶ Finite element types on incompatible geometric elements
- Runtime problems
 - ▶ Invalid input files
 - ▶ Unsupported p-refinement levels
 - ▶ Attempting incompatible mesh modification

Low-level Assertions

Defining `_GLIBCXX_DEBUG`

- Runtime bounds-checking of standard vector, iterators
- Similar to ifort “-check bounds”
- Out Of Bounds errors otherwise lead to corrupt data, not just segfaults!

Low-level Assertions

Defining `_GLIBCXX_DEBUG`

- Runtime bounds-checking of standard vector, iterators
- Similar to ifort “-check bounds”
- Out Of Bounds errors otherwise lead to corrupt data, not just segfaults!

Has Found

- Many off-by-one indexing, transposed indexing errors
- Array allocation errors
- MPI communication mismatches

Unit Tests

Testing One Object At A Time

- Reusable modules interact with all other code through a limited API
- That API can be tested directly outside of application code
- Test one method at a time, isolate problems locally
- 108 unit tests currently in `libMesh`

Unit Tests

Testing One Object At A Time

- Reusable modules interact with all other code through a limited API
- That API can be tested directly outside of application code
- Test one method at a time, isolate problems locally
- 108 unit tests currently in `libMesh`

Has Found

- Bad 5th order tetrahedral quadrature rule
- ... not much else

Unit Tests

Testing One Object At A Time

- Reusable modules interact with all other code through a limited API
- That API can be tested directly outside of application code
- Test one method at a time, isolate problems locally
- 108 unit tests currently in `libMesh`

Has Found

- Bad 5th order tetrahedral quadrature rule
- ... not much else

Tests are useful **only when run!**

- Regression tests run first \Rightarrow catch bugs first
- Adding `libmesh` unit tests to Buildbot

Unit Tests Example

```
#include <quadrature.h>

class QuadratureTest : public CppUnit::TestCase {
public:
    CPPUNIT_TEST_SUITE( QuadratureTest );
    CPPUNIT_TEST( test3DWeights<FIFTH> ); // etc.

    template <Order order>
    void test3DWeights ()
    {
        AutoPtr<QBase> qrule = QBase::build(QGAUSS, 3, order);
        qrule->init (TET10);
        sum = 0;
        for (unsigned int qp=0; qp<qrule->n_points(); qp++)
            sum += qrule->w(qp);
        CPPUNIT_ASSERT_DOUBLES_EQUAL( 1./6., sum , TOLERANCE*TOLERANCE );
    }
};
```

Parametric Testing

One Test Code, Many Tests

- Keep test codes generic
- Execute with many different parameter choices
- `libMesh` compile time examples:
 - ▶ Algebraic solver interface
 - ▶ Real/complex arithmetic
 - ▶ Mesh data structure
- `libMesh` run time examples:
 - ▶ Geometric element type
 - ▶ Finite element type
 - ▶ Polynomial degree
 - ▶ Error indicator type
 - ▶ Processor count
 - ▶ Partitioner
 - ▶ Adaptive refinement strategy
 - ▶ I/O format

Validation Benchmark Problems

Choosing Test Problems

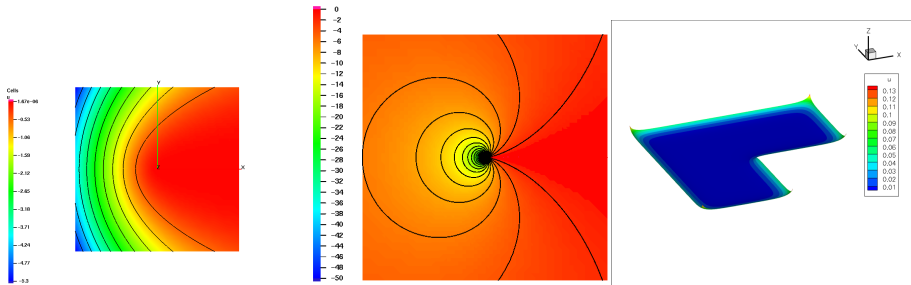
Capitalize on anything you know a priori:

- Known solutions
 - ▶ Exact solution to discretized problem
 - ▶ Limit solution of continuous problem
 - ▶ Known quantities of interest
- Known asymptotic convergence rates
- **Known residuals**

Known Solutions

Examples

- Incompressible flow around a cusp
- Wetting angle in Laplace-Young surface tension



Manufactured Solutions

Any Solution for Any Equation

- Real Physics: $\mathbf{R}(\mathbf{u}) = 0$ for $\mathbf{u} = \mathbf{u}^r$
- Choose manufactured solution \mathbf{u}^m
- Desired Physics: $\mathbf{R}^m(\mathbf{u}) = 0$ for $\mathbf{u} = \mathbf{u}^m$
- Construct $\mathbf{R}^m(\mathbf{u}) \equiv \mathbf{R}(\mathbf{u}) - \mathbf{R}(\mathbf{u}^m)$

Manufactured Solutions

Any Solution for Any Equation

- Real Physics: $\mathbf{R}(\mathbf{u}) = 0$ for $\mathbf{u} = \mathbf{u}^r$
- Choose manufactured solution \mathbf{u}^m
- Desired Physics: $\mathbf{R}^m(\mathbf{u}) = 0$ for $\mathbf{u} = \mathbf{u}^m$
- Construct $\mathbf{R}^m(\mathbf{u}) \equiv \mathbf{R}(\mathbf{u}) - \mathbf{R}(\mathbf{u}^m)$

Has Found

- Assorted application bugs
- Adjoint Sensitivity Bugs!

Manufactured Solution Example

Convection-Diffusion Problem with Adjoints

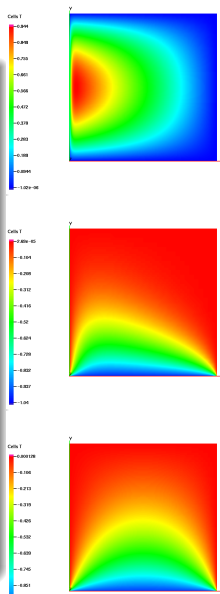
Residual equation:

$$\mathbf{R}(\mathbf{u}) = \nabla \cdot \alpha \nabla \mathbf{u} + \beta \vec{e}_x \cdot \nabla \mathbf{u} + \mathbf{f} = 0$$

Manufactured solution:

$$\mathbf{u} \equiv 4(1 - e^{-\alpha x} - (1 - e^{-\alpha})x)y(1 - y)$$

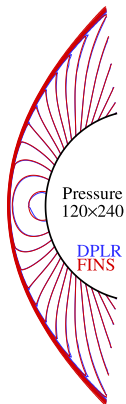
- Homogeneous Dirichlet boundary
- α controls flux strength, layer
- Choose any convection strength β , solve for \mathbf{f}
- $\beta = 0$ gives simple series adjoint solutions



Code-to-Code Comparisons

“Lumped” Verification

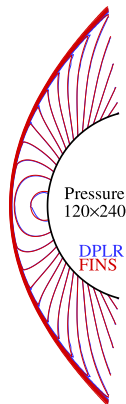
- Differing results from:
 - ▶ Different models
 - ▶ Different formulations
 - ▶ Different discretizations



Code-to-Code Comparisons

“Lumped” Verification

- Differing results from:
 - ▶ Different models
 - ▶ Different formulations
 - ▶ Different discretizations



Has Found

- Differing FIN-S shock capturing operator behaviors
- Differing DPLR/LAURA high temperature models

Manufactured Iterates

The Last Resort

- Your manufactured solution fails validation
- The bug location isn't obvious
- You can't invert the problem by hand

Test $R(U)$

- Small (one, two element!?) meshes
- Cartesian grids

Manufactured Iterates

The Last Resort

- Your manufactured solution fails validation
- The bug location isn't obvious
- You can't invert the problem by hand

Test $R(U)$

- Small (one, two element!?) meshes
- Cartesian grids

Has Found

- Invalid mixed formulations
- Bad basis values on new FE type
- Bad adaptive constraint application

Hierarchical Models

Per-Operator Tesing

- Coefficients selectively “turn off” parts of equations
- Allows easier construction of analytic solutions
- Assists with “narrowing down” other bugs

Model Simplification

- Model linearity can be tested in solver
- Reduce complex physics to simple physics case
- Code-to-code testing

Hierarchical Models

Per-Operator Tesing

- Coefficients selectively “turn off” parts of equations
- Allows easier construction of analytic solutions
- Assists with “narrowing down” other bugs

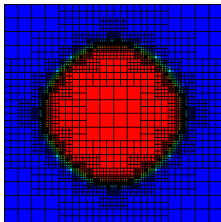
Model Simplification

- Model linearity can be tested in solver
- Reduce complex physics to simple physics case
- Code-to-code testing

Has Found

- Bugs in non-Newtonian viscous flow code
- Bugs in new FIN-S multi-species capabilities

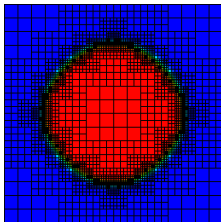
Symmetry Tests



Symmetry In, Symmetry Out

- Mirror, radial symmetries
- Beware unstable solution modes!

Symmetry Tests



Symmetry In, Symmetry Out

- Mirror, radial symmetries
- Beware unstable solution modes!

Has Found

- Indexing errors in DPLR coupling
- Artificial “pinning” in Cahn-Hilliard discretizations

Jacobian Verification

Inexact Newton Step

$$\mathbf{J}(\mathbf{u}^{n-1}) (\mathbf{u}^n - \mathbf{u}^{n-1}) \equiv -\mathbf{R}(\mathbf{u}^{n-1})$$
$$\mathbf{J} \equiv \frac{\partial \mathbf{R}}{\partial \mathbf{u}}$$

- Library code handles inexact solve tolerances, line search, etc.
- \mathbf{R} , \mathbf{J} are application-dependent

Library Jacobian Construction

Finite Differencing

$$\mathbf{J}_{ij} \approx \frac{\mathbf{R}_i(\mathbf{u} + \varepsilon \mathbf{e}_j) - \mathbf{R}_i(\mathbf{u} - \varepsilon \mathbf{e}_j)}{2\varepsilon}$$

Greedy or element-wise algorithms handle sparsity

Complex-Step Perturbations

$$\mathbf{J}_{ij} \approx \frac{\text{Im}[\mathbf{R}_i(\mathbf{u} + \varepsilon \mathbf{e}_j \sqrt{-1})]}{\varepsilon}$$

Avoids floating point subtractive cancellation error

Automatic Differentiation

- Variable constructors seed derivatives
- Variable operations evaluate derivatives

Jacobian Verification

Test Analytic vs. Numeric Jacobians

- Relative error in matrix norm
- If match isn't within tolerance, either:
 - ▶ The discretization or floating point error has overwhelmed the finite differenced Jacobian
 - Unlikely for good choices of finite difference perturbations
 - Can be investigated
 - ▶ The residual is non-differentiable at that iterate
 - Can be checked analytically
 - ▶ The Jacobian calculation is wrong
 - ▶ The residual calculation is wrong

Has Found

- Application codes with bad Jacobians (often!)
- Application codes with bad residuals

A Priori Asymptotic Convergence Rates

Pros

- Applicable to wide ranges of problems
- Does not require exact, or even manufactured, solution
- Part of solution verification, not just code verification

Cons

- Does not verify physics
- Requires asymptotic assumption
- Part of solution verification, not just code verification

A Priori Asymptotic Convergence Rates

Pros

- Applicable to wide ranges of problems
- Does not require exact, or even manufactured, solution
- Part of solution verification, not just code verification

Cons

- Does not verify physics
- Requires asymptotic assumption
- Part of solution verification, not just code verification

Has Found

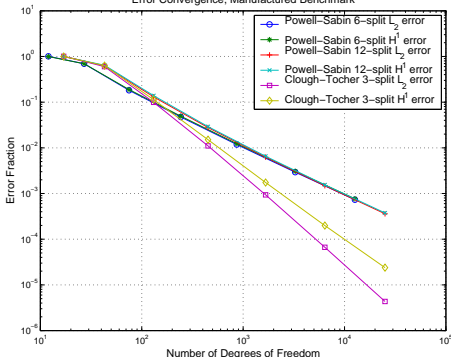
- Errors in new FE options
- Errors in adaptivity application
- Errors in application codes

Asymptotic Convergence Rate Examples

Biharmonic Problem, Manufactured Solution

- $\Delta^2 \mathbf{u} = \mathbf{f}$
- C^1 Macroelement bases

Error Convergence, Manufactured Benchmark

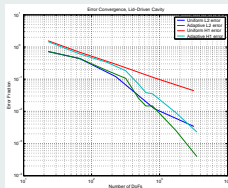
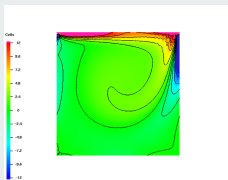


Verification Results

- Code verification failures - bugs in basis transformations
- Solution verification “failure” - higher order Nitsche lift fails for L_2 error with quadratic elements for fourth order problems

Asymptotic Convergence Rate Examples

Lid-Driven Cavity Flow

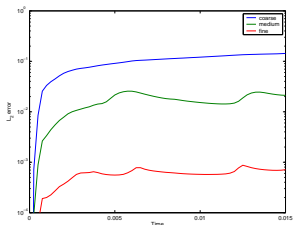
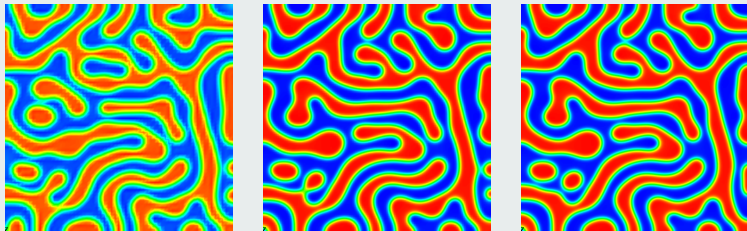


Has Found

- Solution Verification:
 - ▶ Good/Bad adaptive refinement strategies
 - ▶ Non-conforming boundary condition enforcement
- Code Verification:
 - ▶ Library failure to handle $h \approx \mathcal{O}(10^{-6})$
 - ▶ Application failure to enforce hanging node constraints

Asymptotic Convergence Rate Examples

Cahn-Hilliard Phase Evolution

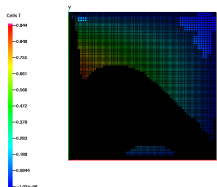
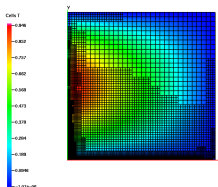
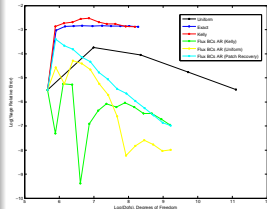


Gives some confidence in even highly nonlinear, stochastic problems

Asymptotic Convergence Rate Examples

Goal-Oriented Refinement

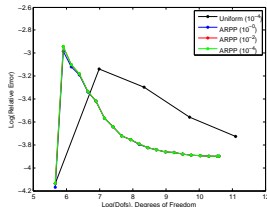
- Superconvergence on some grids
- Convergence “plateaus” found in multiple refinement strategies
- `UniformRefinementEstimator` required new code to solve for adjoint solution errors
- `PatchRecoveryErrorEstimator` required new seminorm integration (H^1 vs. $W^{1,\text{inf}}$) to give compatible error subestimates



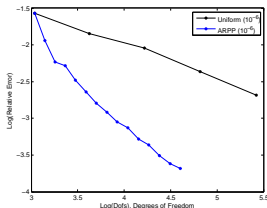
Asymptotic Convergence Rate Examples

Adjoint-based Parameter Sensitivity

- Convergence to analytic sensitivity plateaus at 2% relative error in **every** refinement strategy
- Finite differenced partial derivatives not responsible
- Manufactured solution allowed sensitivity subcomponent comparison to analytic solutions
- Sign errors in `libMesh` parameter sensitivity method



Asymptotic Convergence Rate Examples



Adjoint-based Parameter Sensitivity

- “Off by 100%” error remaining in one term of solution
- Switch to $u'' = f$, 1D quadratic solutions, manufactured residual test
- Identified bug in repeated `adjoint_solve rhs assembly`
- Returned to manufactured solution benchmark: now converges to true solution

Thank you

