PECOS

Predictive Engineering and Computational Sciences

Tools and Techniques for Software Verification using Manufactured Solutions

Nicholas Malaya

Center for Predictive Engineering and Computational Sciences (PECOS)
Institute for Computational Engineering and Sciences (ICES)
The University of Texas at Austin

June 30th, 2014

Outline

This talk is online:

`users.ices.utexas.edu/~nick/lanl`

Lecture

- Motivation for Verification
- Introduction to the Method of Manufactured Solutions
- Creating Manufactured Solutions
- The MASA Library

Why Verify?

Reinhart and Kenneth S. Rogoff: Growth in a Time of Debt

- “Our main result is that whereas the link between growth and debt seems relatively weak at normal debt levels, median growth rates for countries with public debt over roughly 90 percent of GDP are about one percent lower than otherwise”
- Dataset: “inflation and GDP growth across varying levels of debt for 20 advanced countries over the period 1946 through 2009”
- Rogoff testified in front of the Senate Budget Committee

Inform Decision Makers

- France: 30 Billion Euros
- England: 11.5 Billion Pounds
- “Spain’s national budget cuts of almost 14 percent and regional budget cuts of up to 10 percent in health and social services”

Verification Failure

Excel Error!

- “Instead of AVERAGE(L30:L49), AVERAGE(L30:L44) was used.”
- When corrected, GDP/DEBT ratios above 90% had growth of 2.2

JP Morgan needs more verification

Estimating risk with computer models

- JPMorgans Chief Investment Office needed a new value-at-risk (VaR) model for the synthetic credit portfolio
- ”London-based quantitative expert, mathematician and model developer“ was dispatched
- The model seriously underestimated the downside the synthetic credit portfolio
- Ultimately led to the bank to declare $6billioninlossesandanother$600 million in fines.

What went wrong?

- After subtracting the old rate from the new rate, the spreadsheet divided by their sum instead of their average
- The errors stemmed from a combination of copy-paste mistakes and a faulty equation created to crunch the numbers.

Scientific Computing In Practice

Numerical simulations have a broad range of applicability

- Computer Aided Design (e.g. Boeing 787)
- Human treatment and drug discovery
- Societal impact (global warming yes/no?)
- Disaster recovery and prediction (e.g. earthquakes, hurricanes, storm surges)
- Helping to further our understanding of the physical world (material science, supernovae)
- *Possibly getting you a degree...*

Computer Modeling and Simulation

Most types of programming:

- Some errors are tolerable, some physics are negotiable (e.g. shaders, rendering)
- Speed is negotiable
- Often has to be pretty or easy to use, or both

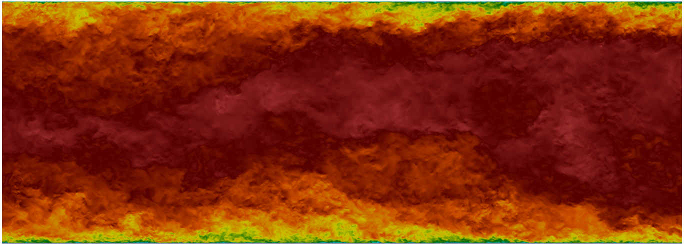
Scientific and Technical Computing:

- You have to be correct (or at least quantifiably wrong)
- We often don't know the "correct" answer
- Sometimes we cannot make new experiments (e.g. NNSA stockpile stewardship)
- Needs to be fast (*think hurricane weather prediction*)
- Software does not need to be easy to use

Direct Numerical Simulations

Physics Simulation

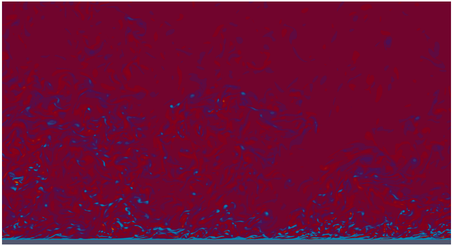
- 225 Billion Degrees of Freedom
- 524,288 Processing Cores
- 275 Million Compute Hours
- Indistinguishable from physical experiments
- **Are we confident in our predictions?**



Direct Numerical Simulations

It's not you, it's me.

- "We have released Mira for use. However, service is degraded. Several racks (1024 nodes each) remain offline, and the system will have a high risk of hardware errors."
- "A new efix was installed on Mira and Cetus which contains fixes and performance enhancements for the BG/Q MPI and PAMI packages. All users are strongly advised to recompile and relink their code."
- **Are we STILL confident in our predictions?**



What is verification?

Reality

$$\Downarrow$$

Mathematical Model

$$\frac{d^2x(t)}{dt^2} = \frac{F}{M}$$

$$\Downarrow$$

Numerical Representation

$$\frac{d^2x}{dt^2} = f''(t) = \frac{f(t+h) - 2f(t) + f(t-h)}{h^2}$$

Verification

Verification of Scientific Software

- Verification ensures that the outputs of a computation accurately reflect the solution of the mathematical models.

Code Verification

- Ensuring that the code used in the simulation correctly implements the intended numerical discretization of the model.
  - This concept is *not* unique to Scientific Software

Solution Verification

- Are the errors from the numerical discretization sufficiently small?
- Is the convergence rate consistent with the numerical scheme?

Solution Verification Methods

Method of Exact Solutions

- Numerically solve the governing equations for which the solution can be determined analytically.

MMS

- Often, analytical solutions either:
  - Do not exist (Navier-Stokes)
  - Do not fully exercise equations (e.g. a symmetric solution, nonlinearities)
- Alleviate this using Method of Manufactured Solutions (MMS)
  - Simply put, we “create” our own solutions

Manufactured solution to Laplace's Equation

Laplace's Equation:

$$\nabla^2 \phi = 0$$

In two dimensions:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

"Manufacture" a solution, with two constants:

$$\phi(x, y) = ({\color{red}Ly} - y)^2({\color{red}Ly} + y)^2 + ({\color{red}Lx} - x)^2({\color{red}Lx} + x)^2$$

Calculating the Source Term

We insert our manufactured solution back into the governing equations:

$$\frac{\partial^2((Lx-x)^2(Lx+x)^2)}{\partial x^2}+\frac{\partial^2((Ly-y)^2(Ly+y)^2)}{\partial y^2}=0$$

$$\begin{aligned}
&=2(Lx-x)^2-8(Lx-x)(Lx+x)+2(Lx+x)^2\\
&+2(Ly-y)^2-8(Ly-y)(Ly+y)+2(Ly+y)^2\\
&\color{red}{\neq 0}
\end{aligned}$$

This does not satisfy Laplace's Equation!

To balance the equation, add the residual to the RHS as a source term.

Example Verification Use Case

To solve Laplace's Equation numerically, we need a discretization scheme.

Let's use a 2nd order finite central difference:

$$\phi_i'' \approx \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{h^2} + O(h^2)$$

This requires solving the implicit system of equations:

$$A\vec{\phi} = f$$

You can use your favorite linear solver (e.g. PETSc) to solve the system.

Problem: Solve 2D Laplacian using Finite-Differencing

Outline

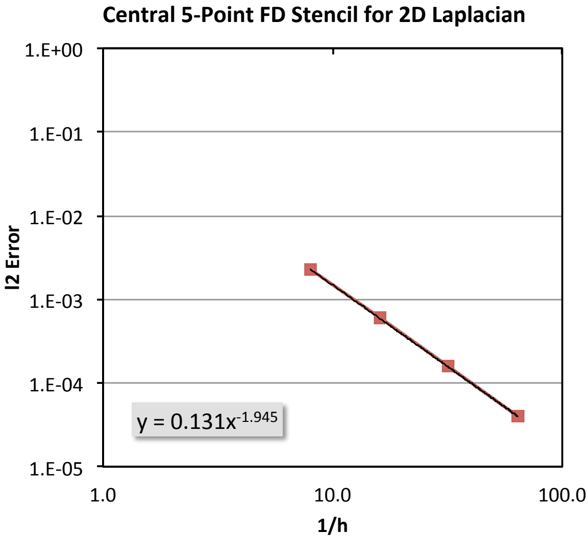
- *Goal:* Write a program in C/C++, F90
- *Inputs*:
  - # of points in one direction (*npts*)
  - the physical dimension of one side (L_x, L_y)
- *Output*: l_2 error between your numerical solution and an exact solution derived from a manufactured solution

$$l_2 = \sqrt{\frac{\sum_{i=1}^{\text{N}} (\phi_i - \phi_i^{\text{exact}})^2}{N}}$$

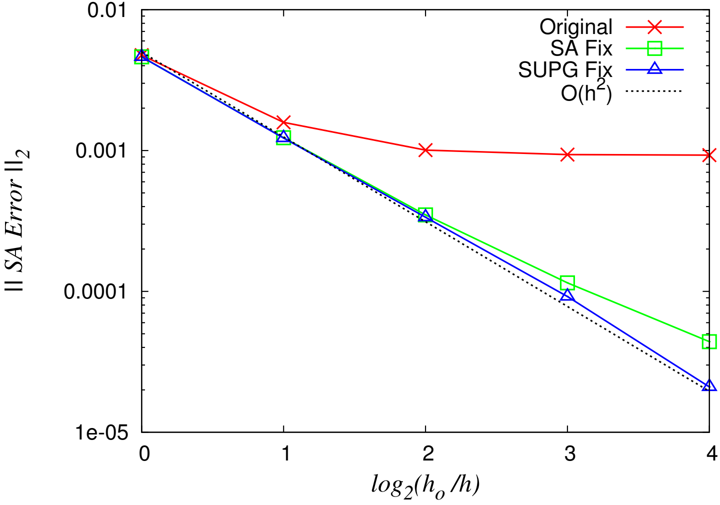
- *Runs*: Run your snazzy code for npts $= 5, 9, 17,$ and 33 and plot l_2 norm as a function of $1/h$ where $h = \text{length}/(\text{npts} - 1)$

Example Results: What we're hoping for

2nd Order Central Finite-difference Scheme



This Process Finds Bugs



Useful for Detecting Subtle Bugs

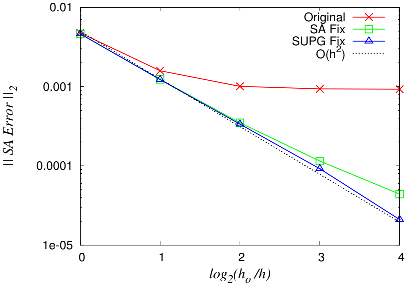
Verification of FIN-S

- FANS, Spalart-Allmaras
- Derivative:

$$\frac{d(sa)}{dx} = \frac{1}{\rho} * \left(\frac{d(\rho * sa)}{dx} - sa\frac{d\rho}{dx} \right)$$

- In code:

$$\frac{d(sa)}{dx} = \frac{1}{\rho} * \frac{d(\rho * sa)}{dx} - sa\frac{d\rho}{dx}$$



Summary

What we have learned:

- We can generate MS even for systems we do not have solutions for
- The MMS is a powerful method to verify rates of convergence

Why is this not more commonly done?

- Solution Generation is complex and time intensive
- Meaningful Solution generation can be subtle

A Real Example

MMS Creation Process

- Start by “manufacturing” a suitable closed-form exact solution
- For example, the 10 parameter trigonometric solution of the form: (Roy, 2002)

$$\hat{u}(x,y,z,t) = \hat{u}_0 + \hat{u}_x\, f_s\left(\frac{a_{\hat{u}x}\pi x}{L}\right) + \hat{u}_y\, f_s\left(\frac{a_{\hat{u}y}\pi y}{L}\right) + \\ + \hat{u}_z\, f_s\left(\frac{a_{\hat{u}z}\pi z}{L}\right) + \hat{u}_t\, f_s\left(\frac{a_{\hat{u}_t}\pi t}{L}\right)$$

- Apply this solution to equations of interest, solve for source terms (residual)

Accomplished using packages such as Maple, Mathematica, SymPy, Macsyma, etc.

Maple MMS: 3D Navier-Stokes Energy Term

$$
\begin{aligned}
Qe = & -\frac{a_{px}\pi p_x}{L}\frac{\gamma}{\gamma-1}\sin\left(\frac{a_{px}\pi x}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]+\\
& +\frac{a_{py}\pi p_y}{L}\frac{\gamma}{\gamma-1}\cos\left(\frac{a_{py}\pi y}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]+\\
& -\frac{a_{pz}\pi p_z}{L}\frac{\gamma}{\gamma-1}\sin\left(\frac{a_{pz}\pi z}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]+\\
& +\frac{a_{\rho x}\pi\rho_x}{2L}\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]\left(\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]^2+\right.\\
& \quad\left.+\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]^2+\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]^2\right)+\\
& -\frac{a_{\rho y}\pi\rho_y}{2L}\sin\left(\frac{a_{\rho y}\pi y}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]\left(\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]^2+\right.\\
& \quad\left.+\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]^2+\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]^2\right)+\\
& +\frac{a_{\rho z}\pi\rho_z}{2L}\cos\left(\frac{a_{\rho z}\pi z}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]\left(\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]^2+\right.\\
& \quad\left.+\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]^2+\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]^2\right)+\\
& +\frac{a_{ux}\pi u_x}{2L}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left\{\left(\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]^2+\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]^2+\right.\right.\\
& \quad\left.+3\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]^2\right)\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]+\\
& \quad\left.+\left[p_0+p_x\cos\left(\frac{a_{px}\pi x}{L}\right)+p_y\sin\left(\frac{a_{py}\pi y}{L}\right)+p_z\cos\left(\frac{a_{pz}\pi z}{L}\right)\right]\frac{2\gamma}{(\gamma-1)}\right\}+\\
& -\frac{a_{uy}\pi u_y}{L}\sin\left(\frac{a_{uy}\pi y}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\cdot\\
& \quad\cdot\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]+\\
& -\frac{a_{uz}\pi u_z}{L}\sin\left(\frac{a_{uz}\pi z}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\cdot\\
& \quad\cdot\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]+
\end{aligned}
$$

But wait, there's more!

$$-\frac{a_{\rho x}^2\pi^2 k\rho_x\left[2\cos\left(\frac{a_{\rho x}\pi x}{L}\right)^2\rho_x+\sin\left(\frac{a_{\rho x}\pi x}{L}\right)\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\right]\left[p_0+p_x\cos\left(\frac{a_{px}\pi x}{L}\right)+p_y\sin\left(\frac{a_{py}\pi y}{L}\right)+p_z\cos\left(\frac{a_{pz}\pi z}{L}\right)\right]}{RL^2\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]^3}+$$

$$-\frac{a_{\rho y}^2\pi^2 k\rho_y\left[2\sin\left(\frac{a_{\rho y}\pi y}{L}\right)^2\rho_y+\cos\left(\frac{a_{\rho y}\pi y}{L}\right)\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\right]\left[p_0+p_x\cos\left(\frac{a_{px}\pi x}{L}\right)+p_y\sin\left(\frac{a_{py}\pi y}{L}\right)+p_z\cos\left(\frac{a_{pz}\pi z}{L}\right)\right]}{RL^2\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]^3}+$$

$$-\frac{a_{\rho z}^2\pi^2 k\rho_z\left[2\cos\left(\frac{a_{\rho z}\pi z}{L}\right)^2\rho_z+\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\right]\left[p_0+p_x\cos\left(\frac{a_{px}\pi x}{L}\right)+p_y\sin\left(\frac{a_{py}\pi y}{L}\right)+p_z\cos\left(\frac{a_{pz}\pi z}{L}\right)\right]}{RL^2\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]^3}+$$

$$-\frac{4}{3}\frac{a_{ux}^2\pi^2\mu u_x}{L^2}\left[\cos\left(\frac{a_{ux}\pi x}{L}\right)^2u_x-\sin\left(\frac{a_{ux}\pi x}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{uy}^2\pi^2\mu u_y}{L^2}\left[\sin\left(\frac{a_{uy}\pi y}{L}\right)^2u_y-\cos\left(\frac{a_{uy}\pi y}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{uz}^2\pi^2\mu u_z}{L^2}\left[\sin\left(\frac{a_{uz}\pi z}{L}\right)^2u_z-\cos\left(\frac{a_{uz}\pi z}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{vx}^2\pi^2\mu v_x}{L^2}\left[\sin\left(\frac{a_{vx}\pi x}{L}\right)^2v_x-\cos\left(\frac{a_{vx}\pi x}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{4}{3}\frac{a_{vy}^2\pi^2\mu v_y}{L^2}\left[\cos\left(\frac{a_{vy}\pi y}{L}\right)^2v_y-\sin\left(\frac{a_{vy}\pi y}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{vz}^2\pi^2\mu v_z}{L^2}\left[\cos\left(\frac{a_{vz}\pi z}{L}\right)^2v_z-\sin\left(\frac{a_{vz}\pi z}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{wx}^2\pi^2\mu w_x}{L^2}\left[\cos\left(\frac{a_{wx}\pi x}{L}\right)^2w_x-\sin\left(\frac{a_{wx}\pi x}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{wy}^2\pi^2\mu w_y}{L^2}\left[\cos\left(\frac{a_{wy}\pi y}{L}\right)^2w_y-\sin\left(\frac{a_{wy}\pi y}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{4}{3}\frac{a_{wz}^2\pi^2\mu w_z}{L^2}\left[\sin\left(\frac{a_{wz}\pi z}{L}\right)^2w_z-\cos\left(\frac{a_{wz}\pi z}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]\right]+$$

$$-\frac{a_{vx}\pi v_x}{L}\sin\left(\frac{a_{vx}\pi x}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\cdot$$

$$\cdot\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]+$$

$$+\frac{a_{vy}\pi v_y}{2L}\cos\left(\frac{a_{vy}\pi y}{L}\right)\left\{\left(\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]^2+\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]^2+\right.\right.$$

$$\left.+3\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]^2\right)\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]+$$

$$\left.+\left[p_0+p_x\cos\left(\frac{a_{px}\pi x}{L}\right)+p_y\sin\left(\frac{a_{py}\pi y}{L}\right)+p_z\cos\left(\frac{a_{pz}\pi z}{L}\right)\right]\frac{2\gamma}{\gamma-1}\right\}+$$

$$+\frac{a_{vz}\pi v_z}{L}\cos\left(\frac{a_{vz}\pi z}{L}\right)\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\cdot$$

$$\cdot\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]+$$

$$+\frac{a_{wx}\pi w_x}{L}\cos\left(\frac{a_{wx}\pi x}{L}\right)\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\cdot$$

$$\cdot\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]+$$

$$+\frac{a_{wy}\pi w_y}{L}\cos\left(\frac{a_{wy}\pi y}{L}\right)\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]\cdot$$

$$\cdot\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]+$$

$$-\frac{a_{wz}\pi w_z}{2L}\sin\left(\frac{a_{wz}\pi z}{L}\right)\left\{\left(3\left[w_0+w_x\sin\left(\frac{a_{wx}\pi x}{L}\right)+w_y\sin\left(\frac{a_{wy}\pi y}{L}\right)+w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right]^2+\left[v_0+v_x\cos\left(\frac{a_{vx}\pi x}{L}\right)+v_y\sin\left(\frac{a_{vy}\pi y}{L}\right)+v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right]^2+\right.\right.$$

$$\left.+\left[u_0+u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)+u_y\cos\left(\frac{a_{uy}\pi y}{L}\right)+u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right]^2\right)\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]+$$

$$\left.+\left[p_0+p_x\cos\left(\frac{a_{px}\pi x}{L}\right)+p_y\sin\left(\frac{a_{py}\pi y}{L}\right)+p_z\cos\left(\frac{a_{pz}\pi z}{L}\right)\right]\frac{2\gamma}{(\gamma-1)}\right\}+$$

$$+\frac{a_{px}^2\pi^2kp_x\cos\left(\frac{a_{px}\pi x}{L}\right)}{L^2R\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]}+$$

$$+\frac{a_{py}^2\pi^2kp_y\sin\left(\frac{a_{py}\pi y}{L}\right)}{L^2R\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]}+$$

$$+\frac{a_{pz}^2\pi^2kp_z\cos\left(\frac{a_{pz}\pi z}{L}\right)}{L^2R\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]}+$$

$$-\frac{2a_{px}a_{\rho x}\pi^2kp_x\rho_x\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\sin\left(\frac{a_{px}\pi x}{L}\right)}{L^2R\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]^2}+$$

$$-\frac{2a_{py}a_{\rho y}\pi^2kp_y\rho_y\sin\left(\frac{a_{\rho y}\pi y}{L}\right)\cos\left(\frac{a_{py}\pi y}{L}\right)}{L^2R\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]^2}+$$

$$-\frac{2a_{pz}a_{\rho z}\pi^2kp_z\rho_z\cos\left(\frac{a_{\rho z}\pi z}{L}\right)\sin\left(\frac{a_{pz}\pi z}{L}\right)}{L^2R\left[\rho_0+\rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)+\rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right)+\rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right]^2}+$$

$$+\frac{4}{3}\frac{a_{ux}a_{vy}\pi^2\mu u_xv_y}{L^2}\cos\left(\frac{a_{ux}\pi x}{L}\right)\cos\left(\frac{a_{vy}\pi y}{L}\right)+$$

$$-\frac{4}{3}\frac{a_{ux}a_{wz}\pi^2\mu u_xw_z}{L^2}\cos\left(\frac{a_{ux}\pi x}{L}\right)\sin\left(\frac{a_{wz}\pi z}{L}\right)+$$

$$-\frac{2a_{uy}a_{vx}\pi^2\mu u_yv_x}{L^2}\sin\left(\frac{a_{uy}\pi y}{L}\right)\sin\left(\frac{a_{vx}\pi x}{L}\right)+$$

$$+\frac{2a_{uz}a_{wx}\pi^2\mu u_zw_x}{L^2}\cos\left(\frac{a_{wx}\pi x}{L}\right)\sin\left(\frac{a_{uz}\pi z}{L}\right)+$$

$$-\frac{4}{3}\frac{a_{vy}a_{wz}\pi^2\mu v_yw_z}{L^2}\cos\left(\frac{a_{vy}\pi y}{L}\right)\sin\left(\frac{a_{wz}\pi z}{L}\right)+$$

$$-\frac{2a_{vz}a_{wy}\pi^2\mu v_zw_y}{L^2}\cos\left(\frac{a_{vz}\pi z}{L}\right)\cos\left(\frac{a_{wy}\pi y}{L}\right).$$

C-code output

```c
 RHO = rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L) + rho_t * sin(a_rhot * PI * t / L);
  U = u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L) + u_t * cos(a_ut * PI * t / L);
  V = v_0 + v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_t * sin(a_vt * PI * t / L);
  W = w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L) + w_t * cos(a_wt * PI * t / L);
  P = p_0 + p_x * cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L) + p_t * cos(a_pt * PI * t / L);
  M1 = A_mu * pow(P / R / RHO, 0.3e1 / 0.2e1);
  M2 = P / R / RHO + B_mu;
  MU = M1 / M2;
  Q_e = (0.3e1 * a_ux * u_x * cos(a_ux * PI * x / L) + a_vy * v_y * cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * RHO * U * U / L / 0.2e1 + (a_ux * u_x * cos(a_ux * PI
* x / L) + 0.3e1 * a_vy * v_y * cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * RHO * V * V / L / 0.2e1 + (a_ux * u_x * cos(a_ux * PI * x / L) + a_vy * v_y * cos(a_vy *
PI * y / L) - 0.3e1 * a_wz * w_z * sin(a_wz * PI * z / L)) * PI * RHO * W * W / L / 0.2e1 + (0.4e1 * a_ux * a_ux * u_x * sin(a_ux * PI * x / L) + 0.3e1 * a_uy * a_uy * u_y * cos(a_uy * PI * y
/ L) + 0.3e1 * a_uz * a_uz * u_z * cos(a_uz * PI * z / L)) * MU * PI * PI * U * pow(L, -0.2e1) / 0.3e1 + (0.3e1 * a_vx * a_vx * v_x * cos(a_vx * PI * x / L) + 0.4e1 * a_vy * a_vy * v_y *
sin(a_vy * PI * y / L) + 0.3e1 * a_vz * a_vz * v_z * sin(a_vz * PI * z / L)) * MU * PI * PI * V * pow(L, -0.2e1) / 0.3e1 + (0.3e1 * a_wx * a_wx * w_x * sin(a_wx * PI * x / L) + 0.3e1 * a_wy *
a_wy * w_y * sin(a_wy * PI * y / L) + 0.4e1 * a_wz * a_wz * w_z * cos(a_wz * PI * z / L)) * MU * PI * PI * W * pow(L, -0.2e1) / 0.3e1 + (a_ux * u_x * cos(a_ux * PI * x / L) + a_vy * v_y *
cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * P / (Gamma - 0.1e1) / L - (0.2e1 * a_rhox * a_px * rho_x * p_x * cos(a_rhox * PI * x / L) * sin(a_px * PI * x / L) + 0.2e1
* a_rhoy * a_py * rho_y * p_y * sin(a_rhoy * PI * y / L) * cos(a_py * PI * y / L) + 0.2e1 * a_rhoz * a_pz * rho_z * p_z * cos(a_rhoz * PI * z / L) * sin(a_pz * PI * z / L)) * PI * PI * k *
pow(L, -0.2e1) / R * pow(RHO, -0.2e1) + (U * U + V * V + W * W) * a_rhot * PI * rho_t * cos(a_rhot * PI * t / L) / L / 0.2e1 - a_pt * PI * p_t * sin(a_pt * PI * t / L) / (Gamma - 0.1e1) / L -
(a_uy * u_y * sin(a_uy * PI * y / L) + a_vx * v_x * sin(a_vx * PI * x / L)) * PI * RHO * U * V / L - (a_uz * u_z * sin(a_uz * PI * z / L) - a_wx * w_x * cos(a_wx * PI * x / L)) * PI * RHO * U
* W / L + (a_vz * v_z * cos(a_vz * PI * z / L) + a_wy * w_y * cos(a_wy * PI * y / L)) * PI * RHO * V * W / L + (a_px * a_px * p_x * cos(a_px * PI * x / L) + a_py * a_py * p_y * sin(a_py * PI
* y / L) + a_pz * a_pz * p_z * cos(a_pz * PI * z / L)) * PI * PI * k * pow(L, -0.2e1) / R / RHO - (0.4e1 * a_ux * a_ux * u_x * u_x * pow(cos(a_ux * PI * x / L), 0.2e1) - 0.4e1 * a_ux * a_vy *
u_x * v_y * cos(a_ux * PI * x / L) * cos(a_vy * PI * y / L) + 0.4e1 * a_ux * a_wz * u_x * w_z * cos(a_ux * PI * x / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_uy * a_uy * u_y * u_y *
pow(sin(a_uy * PI * y / L), 0.2e1) + 0.6e1 * a_uy * a_vx * u_y * v_x * sin(a_uy * PI * y / L) * sin(a_vx * PI * x / L) + 0.3e1 * a_uz * a_uz * u_z * u_z * pow(sin(a_uz * PI * z / L), 0.2e1) -
0.6e1 * a_uz * a_wx * u_z * w_x * sin(a_uz * PI * z / L) * cos(a_wx * PI * x / L) + 0.3e1 * a_vx * a_vx * v_x * v_x * pow(sin(a_vx * PI * x / L), 0.2e1) + 0.4e1 * a_vy * a_vy * v_y * v_y *
pow(cos(a_vy * PI * y / L), 0.2e1) + 0.4e1 * a_vy * a_wz * v_y * w_z * cos(a_vy * PI * y / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_vz * a_vz * v_z * v_z * pow(cos(a_vz * PI * z / L), 0.2e1) +
0.6e1 * a_vz * a_wy * v_z * w_y * cos(a_vz * PI * z / L) * cos(a_wy * PI * y / L) + 0.3e1 * a_wx * a_wx * w_x * w_x * pow(cos(a_wx * PI * x / L), 0.2e1) + 0.3e1 * a_wy * a_wy * w_y * w_y *
pow(cos(a_wy * PI * y / L), 0.2e1) + 0.4e1 * a_wz * a_wz * w_z * w_z * pow(sin(a_wz * PI * z / L), 0.2e1)) * MU * PI * PI * pow(L, -0.2e1) / 0.3e1 + (a_ux * u_x * cos(a_ux * PI * x / L) +
a_vy * v_y * cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * P / L - a_ut * PI * u_t * RHO * U * sin(a_ut * PI * t / L) / L + a_vt * PI * v_t * RHO * V * cos(a_vt * PI *
t / L) / L - a_wt * PI * w_t * RHO * W * sin(a_wt * PI * t / L) / L + (U * U + V * V + W * W) * a_rhox * PI * rho_x * U * cos(a_rhox * PI * x / L) / L / 0.2e1 - (U * U + V * V + W * W) *
a_rhoy * PI * rho_y * V * sin(a_rhoy * PI * y / L) / L / 0.2e1 + (U * U + V * V + W * W) * a_rhoz * PI * rho_z * W * cos(a_rhoz * PI * z / L) / L / 0.2e1 - (a_rhox * a_rhox * rho_x *
sin(a_rhox * PI * x / L) + a_rhoy * a_rhoy * rho_y * cos(a_rhoy * PI * y / L) + a_rhoz * a_rhoz * rho_z * sin(a_rhoz * PI * z / L)) * PI * PI * k * P * pow(L, -0.2e1) / R * pow(RHO, -0.2e1) -
(0.2e1 * a_rhox * a_rhox * rho_x * rho_x * pow(cos(a_rhox * PI * x / L), 0.2e1) + 0.2e1 * a_rhoy * a_rhoy * rho_y * rho_y * pow(sin(a_rhoy * PI * y / L), 0.2e1) + 0.2e1 * a_rhoz * a_rhoz *
rho_z * rho_z * pow(cos(a_rhoz * PI * z / L), 0.2e1)) * PI * PI * k * P * pow(L, -0.2e1) / R * pow(RHO, -0.3e1) - (0.3e1 * a_rhox * a_uz * rho_x * u_z * cos(a_rhox * PI * x / L) * sin(a_uz *
PI * z / L) - 0.3e1 * a_rhox * a_wx * rho_x * w_x * cos(a_rhox * PI * x / L) * cos(a_wx * PI * x / L) + 0.3e1 * a_rhoy * a_vz * rho_y * v_z * sin(a_rhoy * PI * y / L) * cos(a_vz * PI * z / L)
+ 0.3e1 * a_rhoy * a_wy * rho_y * w_y * sin(a_rhoy * PI * y / L) * cos(a_wy * PI * y / L) + 0.2e1 * a_rhoz * a_ux * rho_z * u_x * cos(a_rhoz * PI * z / L) * cos(a_ux * PI * x / L) + 0.2e1 *
a_rhoz * a_vy * rho_z * v_y * cos(a_rhoz * PI * z / L) * cos(a_vy * PI * y / L) + 0.4e1 * a_rhoz * a_wz * rho_z * w_z * cos(a_rhoz * PI * z / L) * sin(a_wz * PI * z / L)) * MU * (0.3e1 * B_mu
* R * RHO + P) * PI * PI * W / (B_mu * R * RHO + P) * pow(L, -0.2e1) / RHO / 0.6e1 - a_pz * PI * p_z * Gamma * W * sin(a_pz * PI * z / L) / (Gamma - 0.1e1) / L - (0.3e1 * a_px * a_uy * p_x *
u_y * sin(a_px * PI * x / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_px * a_vx * p_x * v_x * sin(a_px * PI * x / L) * sin(a_vx * PI * x / L) - 0.2e1 * a_py * a_ux * p_y * u_x * cos(a_py * PI * y
/ L) * cos(a_ux * PI * x / L) + 0.4e1 * a_py * a_vy * p_y * v_y * cos(a_py * PI * y / L) * cos(a_vy * PI * y / L) + 0.2e1 * a_py * a_wz * p_y * w_z * cos(a_py * PI * y / L) * sin(a_wz * PI *
z / L) - 0.3e1 * a_pz * a_vz * p_z * v_z * sin(a_pz * PI * z / L) * cos(a_vz * PI * z / L) - 0.3e1 * a_pz * a_wy * p_z * w_y * sin(a_pz * PI * z / L) * cos(a_wy * PI * y / L)) * (0.3e1 * B_mu
* R * RHO + P) * MU * PI * PI * V / (B_mu * R * RHO + P) * pow(L, -0.2e1) / P / 0.6e1 + a_py * PI * p_y * Gamma * V * cos(a_py * PI * y / L) / (Gamma - 0.1e1) / L - (0.3e1 * a_rhox * a_uy *
rho_x * u_y * cos(a_rhox * PI * x / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_rhox * a_vx * rho_x * v_x * cos(a_rhox * PI * x / L) * sin(a_vx * PI * x / L) - 0.2e1 * a_rhoy * a_ux * rho_y * u_x
* sin(a_rhoy * PI * y / L) * cos(a_ux * PI * x / L) + 0.4e1 * a_rhoy * a_vy * rho_y * v_y * sin(a_rhoy * PI * y / L) * cos(a_vy * PI * y / L) + 0.2e1 * a_rhoy * a_wz * rho_y * w_z *
sin(a_rhoy * PI * y / L) * sin(a_wz * PI * z / L) - 0.3e1 * a_rhoz * a_vz * rho_z * v_z * cos(a_rhoz * PI * z / L) * cos(a_vz * PI * z / L) - 0.3e1 * a_rhoz * a_wy * rho_z * w_y * cos(a_rhoz
* PI * z / L) * cos(a_wy * PI * y / L)) * MU * (0.3e1 * B_mu * R * RHO + P) * PI * PI * V / (B_mu * R * RHO + P) * pow(L, -0.2e1) / RHO / 0.6e1 + (0.4e1 * a_rhox * a_ux * rho_x * u_x *
cos(a_rhox * PI * x / L) * cos(a_ux * PI * x / L) - 0.2e1 * a_rhox * a_vy * rho_x * v_y * cos(a_rhox * PI * x / L) * cos(a_vy * PI * y / L) + 0.2e1 * a_rhox * a_wz * rho_x * w_z * cos(a_rhox
* PI * x / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_rhoy * a_uy * rho_y * u_y * sin(a_rhoy * PI * y / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_rhoy * a_vx * rho_y * v_x * sin(a_rhoy * PI * y /
L) * sin(a_vx * PI * x / L) - 0.3e1 * a_rhoz * a_uz * rho_z * u_z * cos(a_rhoz * PI * z / L) * sin(a_uz * PI * z / L) + 0.3e1 * a_rhoz * a_wx * rho_z * w_x * cos(a_rhoz * PI * z / L) *
cos(a_wx * PI * x / L)) * MU * (0.3e1 * B_mu * R * RHO + P) * PI * PI * U / (B_mu * R * RHO + P) * pow(L, -0.2e1) / RHO / 0.6e1 - a_px * PI * p_x * Gamma * U * sin(a_px * PI * x / L) / (Gamma
- 0.1e1) / L + (0.4e1 * a_px * a_ux * p_x * u_x * sin(a_px * PI * x / L) * cos(a_ux * PI * x / L) - 0.2e1 * a_px * a_vy * p_x * v_y * sin(a_px * PI * x / L) * cos(a_vy * PI * y / L) + 0.2e1 *
a_px * a_wz * p_x * w_z * sin(a_px * PI * x / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_py * a_uy * p_y * u_y * cos(a_py * PI * y / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_py * a_vx * p_y * v_x
* cos(a_py * PI * y / L) * sin(a_vx * PI * x / L) - 0.3e1 * a_pz * a_uz * p_z * u_z * sin(a_pz * PI * z / L) * sin(a_uz * PI * z / L) + 0.3e1 * a_pz * a_wx * p_z * w_x * sin(a_pz * PI * z /
L) * cos(a_wx * PI * x / L)) * (0.3e1 * B_mu * R * RHO + P) * MU * PI * PI * U / (B_mu * R * RHO + P) * pow(L, -0.2e1) / P / 0.6e1 - (0.3e1 * a_px * a_uz * p_x * u_z * sin(a_px * PI * x / L)
* sin(a_uz * PI * z / L) - 0.3e1 * a_px * a_wx * p_x * w_x * sin(a_px * PI * x / L) * cos(a_wx * PI * x / L) + 0.3e1 * a_py * a_vz * p_y * v_z * cos(a_py * PI * y / L) * cos(a_vz * PI * z /
L) + 0.3e1 * a_py * a_wy * p_y * w_y * cos(a_py * PI * y / L) * cos(a_wy * PI * y / L) + 0.2e1 * a_pz * a_ux * p_z * u_x * sin(a_pz * PI * z / L) * cos(a_ux * PI * x / L) + 0.2e1 * a_pz *
a_vy * p_z * v_y * sin(a_pz * PI * z / L) * cos(a_vy * PI * y / L) + 0.4e1 * a_pz * a_wz * p_z * w_z * sin(a_pz * PI * z / L) * sin(a_wz * PI * z / L)) * (0.3e1 * B_mu * R * RHO + P) * MU *
PI * PI * W / (B_mu * R * RHO + P) * pow(L, -0.2e1) / P / 0.6e1;
```

Meaningful Verification is difficult

What Constitutes a Robust Test?

- How “strong” is a verification test for a particular codebase?
  - Can you characterize your confidence in a codebase?
  - What constitutes a strong test?

Verification Metrics

- In general, you cannot “verify” a codebase
  - You can, however, detect **verification failures**

Physically-based MS

- Exercise each term in the PDE in a similar way to that of a real solution

Example Problem

Complex Codebase

- Finite Element hypersonic code, fully implicit Navier-Stokes (FIN-S)
- Favre-Averaged Navier Stokes (FANS) + Spalart-Allmaras (SA) turbulence model

FANS + SA

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial}{\partial x_i}(\bar{\rho}\tilde{u}_i) = 0$$

$$\frac{\partial}{\partial t}(\bar{\rho}\tilde{u}_i) + \frac{\partial}{\partial x_j}(\bar{\rho}\tilde{u}_j\tilde{u}_i) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(2(\mu + \mu_t)\tilde{S}_{ji}\right)$$

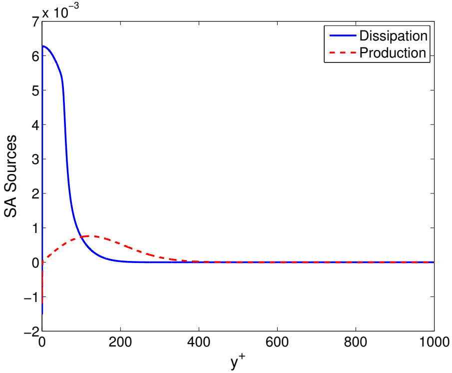
$$\frac{\partial}{\partial t}\left[\bar{\rho}\left(\tilde{e} + \frac{1}{2}\tilde{u}_i\tilde{u}_i\right)\right] + \frac{\partial}{\partial x_j}\left[\bar{\rho}\tilde{u}_j\left(\tilde{h} + \frac{1}{2}\tilde{u}_i\tilde{u}_i\right)\right] = \frac{\partial}{\partial x_j}\left(2(\mu + \mu_t)\tilde{S}_{ji}\tilde{u}_i\right) + \frac{\partial}{\partial x_j}\left[\left(\frac{\mu}{\mathrm{Pr}} + \frac{\mu_t}{\mathrm{Pr}_t}\right)\frac{\partial \tilde{h}}{\partial x_j}\right]$$

$$\frac{\partial}{\partial t}(\bar{\rho}\nu_{\mathrm{sa}}) + \frac{\partial}{\partial x_j}(\bar{\rho}\tilde{u}_j\nu_{\mathrm{sa}}) = c_{b1}S_{\mathrm{sa}}\bar{\rho}\nu_{\mathrm{sa}} - c_{w1}f_w\bar{\rho}\left(\frac{\nu_{\mathrm{sa}}}{d}\right)^2 + \frac{1}{\sigma}\frac{\partial}{\partial x_k}\left[(\mu + \bar{\rho}\nu_{\mathrm{sa}})\frac{\partial \nu_{\mathrm{sa}}}{\partial x_k}\right] + \frac{c_{b2}}{\sigma}\bar{\rho}\frac{\partial \nu_{\mathrm{sa}}}{\partial x_k}\frac{\partial \nu_{\mathrm{sa}}}{\partial x_k}$$

Insufficient Verification

Shortcoming of other SA MS

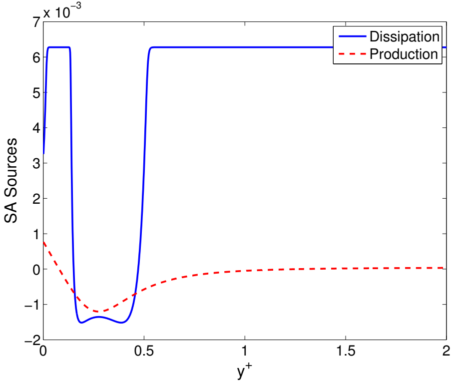
- Bond solution: sinusoidal, only satisfies no-slip.
- Eça solutions were noted to have a suboptimal rate of convergence



A Closer Look

Shortcoming of other SA MS

- Eça solutions are shown to have instabilities or near-wall features that disrupt the correct rate of convergence
  - Offending term: $\nu_{sa} = \tilde{\nu}_{\text{max}} \eta_\nu^2 e^{1-\eta_\nu^2}$



Our SA Manufactured Solution

- Use our understanding of incompressible flow physics to inform our modeling assumptions for this MS

Streamwise Velocity

- The mean streamwise velocity is given by,

$$\tilde{u} = \frac{u_\infty}{A} \sin\left(\frac{A}{u_\infty} u_{eq}\right)$$

The van Driest equivalent velocity can be written as,

$$u_{eq} = u_\tau u_{eq}^+,$$

- Must specify both u_τ and u_{eq}^+

Completing Streamwise Velocity Specification

Correlations

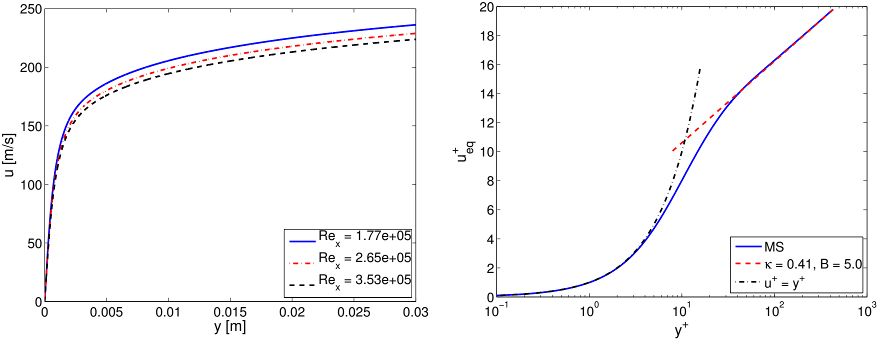
- Friction velocity can be determined from the skin friction coefficient
- The incompressible $1/7$th power law is used for the skin friction coefficient. Thus,

$$c_{f,\mathtt{inc}}(Re_x) = C_{cf} Re_x^{-1/7}$$

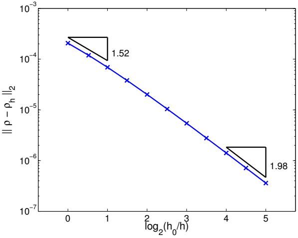
- To complete the manufactured solution, u^+_{eq} is set using the velocity profile model of Cebeci and Bradshaw (1980):

$$u^+_{eq} = \frac{1}{\kappa}\log\left(1+\kappa y^+\right) + C_1\left[1 - e^{-y^+/\eta_1} - \frac{y^+}{\eta_1}e^{-y^+ b}\right]$$

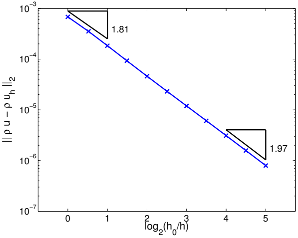
Manufactured Velocity Profiles



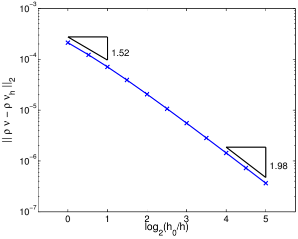
Convergence Rates: Re_x



(a) $\bar{\rho}$



(b) $\bar{\rho}\tilde{u}$



(c) $\bar{\rho}\nu_{\mathrm{sa}}$

Summary

What we have learned:

- Be skeptical of an MS like you would be skeptical of a model
  - Manufactured Solutions must be verified!
- Exercise each term in the PDE in a similar way to that of a real solution
- *More details:* "Manufactured Solutions for the Favre-Averaged Navier-Stokes Equations with Eddy-Viscosity Turbulence Models"
  - AIAA 50th Aerospace Sciences Meeting

Why is this not more commonly done?

- Manufactured Solution generation is a time consuming process
  - Rule of 10%
- Domain expert input needed to generate manufactured solutions!

Manufactured Analytical Solutions Abstractions Library

Goal: Provide a repository and standardized interface for MMS usage

High Priority:

- Extreme fidelity to generated MMS
- Portability
- Traceability
- Extensible

Low Priority:

- Speed/Performance

Verifying the “Verifier”

Precision is not negotiable.

MASA Testing

- Error target $< 1e\text{-}15$
  - Absolute error on local machines
  - Relative error (other)
  - On all supported compiler sets
- -O0 not sufficient
  - -fp-model precise (Intel)
  - -fno-unsafe-math-optimizations (GNU)
  - -Kieee -Mnofpapprox (PGI)
- “make check”
  - Run by Buildbot every two hours

```
[nick@magus trunk]$ make check
-----------------------------------------
Initializing MASA Tests
-----------------------------------------
PASS: init.sh
PASS: misc
PASS: fail_cond
PASS: catch_exception
PASS: register
PASS: poly
PASS: uninit
PASS: vec
PASS: purge
PASS: heat_const_steady
PASS: euler1d

.           .
.           .
.           .

-----------------------------------------
Finalizing MASA Tests
-----------------------------------------
===================
All 65 tests passed
===================
```

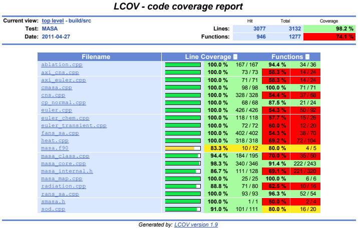
Portability

Testing

- GIT: version control
- Buildbot: automated testing
- GCOV: line coverage
  - 15,826 lines of code
  - 13,195 lines of testing
  - 98%+ line coverage

Software Environment

- Built with: Autotools, C++
- Supports Intel, GNU, Portland Group compilers
- C/C++ interfaces
- Fortran interfaces provided through iso_c_bindings
  - Fortran 2003 Standard
- Python interfaces generated with SWIG



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - build/src				
Test:	MASA	Lines:	3077	3132	98.2 %
Date:	2011-04-27	Functions:	946	1277	74.1 %

Filename	Line Coverage		Functions	
ablation.cpp	100.0 %	167 / 167	94.4 %	34 / 36
axi_cns.cpp	100.0 %	73 / 73	58.3 %	14 / 24
axi_euler.cpp	100.0 %	71 / 71	58.3 %	14 / 24
cmasa.cpp	100.0 %	98 / 98	100.0 %	71 / 71
cns.cpp	100.0 %	328 / 328	54.4 %	37 / 68
cp_normal.cpp	100.0 %	68 / 68	87.5 %	21 / 24
euler.cpp	100.0 %	426 / 426	54.3 %	50 / 92
euler_chem.cpp	100.0 %	118 / 118	57.7 %	15 / 26
euler_transient.cpp	100.0 %	72 / 72	60.0 %	12 / 20
fans_sa.cpp	100.0 %	402 / 402	54.3 %	38 / 70
heat.cpp	100.0 %	318 / 318	69.2 %	72 / 104
masa.f90	83.3 %	10 / 12	80.0 %	4 / 5
masa_class.cpp	94.4 %	184 / 195	70.0 %	35 / 50
masa_core.cpp	98.3 %	340 / 346	91.4 %	222 / 243
masa_internal.h	86.7 %	111 / 128	69.1 %	221 / 320
masa_map.cpp	100.0 %	25 / 25	100.0 %	6 / 6
radiation.cpp	88.8 %	71 / 80	62.5 %	10 / 16
rans_sa.cpp	100.0 %	93 / 93	96.3 %	52 / 54
smasa.h	100.0 %	1 / 1	50.0 %	2 / 4
sod.cpp	91.0 %	101 / 111	80.0 %	16 / 20

Generated by: LCOV version 1.9

Traceability

Doxygen provides code and model documentation

where $\phi = \rho, u, v, w$ or p, and $f_s(\cdot)$ functions denote either sine or cosine function. Note that in this case, ϕ_x, ϕ_y and ϕ_z are constants and the subscripts do not denote differentiation.

Although **?** provide the constants used in the manufactured solutions for the 2D supersonic and sub-sonic cases for Euler and Navier-Stokes equations, only the source term for the 2D mass conservation equation (3.20) is presented.

Source terms for mass conservation (Q_ρ), momentum (Q_u, Q_v and Q_w) and total energy (Q_{e_t}) equations are obtained by symbolic manipulations of compressible steady Euler equations above using Maple 13 (**?**) and are presented in the following sections for the one, two and three-dimensional cases.

3.2.2.1 1D Steady Euler

The manufactured analytical solutions (3.52) for each one of the variables in one-dimensional case of Euler equations are:

$$\begin{aligned} \rho(x) &= \rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) \\ u(x) &= u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) \\ p(x) &= p_0 + p_x \cos\left(\frac{a_{px}\pi x}{L}\right) \end{aligned} \tag{3.26}$$

The MMS applied to Euler equations consists in modifying the 1D Euler equations (3.20) – (3.22) by adding a source term to the right-hand side of each equation:

$$\begin{aligned} &\frac{\partial(\rho u)}{\partial x} = Q_\rho \\ &\frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(p)}{\partial x} = Q_u \\ &\frac{\partial(\rho u e_t)}{\partial x} + \frac{\partial(p u)}{\partial x} = Q_{e_t} \end{aligned} \tag{3.27}$$

so the modified set of equations (3.27) conveniently has the analytical solution given in Equation (3.53).

Source terms Q_ρ, Q_u and Q_{e_t} are obtained by symbolic manipulations of equations above using Maple and are presented in the following sections. The following auxiliary variables have been included in order to improve readability and computational efficiency:

$$\begin{aligned} \mathtt{Rho}_1 &= \rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) \\ U_1 &= u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) \\ \mathtt{P}_1 &= p_0 + p_x \cos\left(\frac{a_{px}\pi x}{L}\right) \end{aligned}$$

where the subscripts refer to the 1D case.

The mass conservation equation written as an operator is:

$$\mathcal{L} = \frac{\partial(\rho u)}{\partial x}$$

k	u_0	u_x	u_y	u_z	v_0	L
v_0	v_x	v_y	v_z	w_0	w_x	w_y
rho_0	rho_x	rho_y	rho_z	p_0	p_y	p_z
a_px	a_py	a_pz	a_rhox	a_rhoy	a_rhoz	a_ux
a_uy	a_uz	a_vx	a_vy	a_vz	a_wx	a_wy
a_wz	mu	Gamma				

Table 3.6: Parameters used by the 3D Steady Euler

- masa_eval_2d_exact_u()
- masa_eval_2d_exact_v()
- masa_eval_2d_exact_p()
- masa_eval_2d_exact_rho()
- masa_eval_2d_grad_u()
- masa_eval_2d_grad_v()
- masa_eval_2d_grad_p()
- masa_eval_2d_grad_rho()

3.2.3.3 3D Steady Euler

Initialization:

- euler_3d

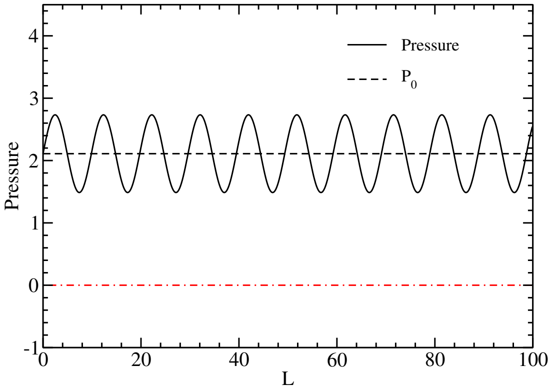
Functions:

- masa_init()
- masa_eval_3d_source_rho_u()
- masa_eval_3d_source_rho_v()
- masa_eval_3d_source_rho_w()
- masa_eval_3d_source_rho_e()
- masa_eval_3d_source_rho()
- masa_eval_3d_exact_u()
- masa_eval_3d_exact_v()
- masa_eval_3d_exact_w()
- masa_eval_3d_exact_p()

Providing Reasonable Defaults

Requirements:

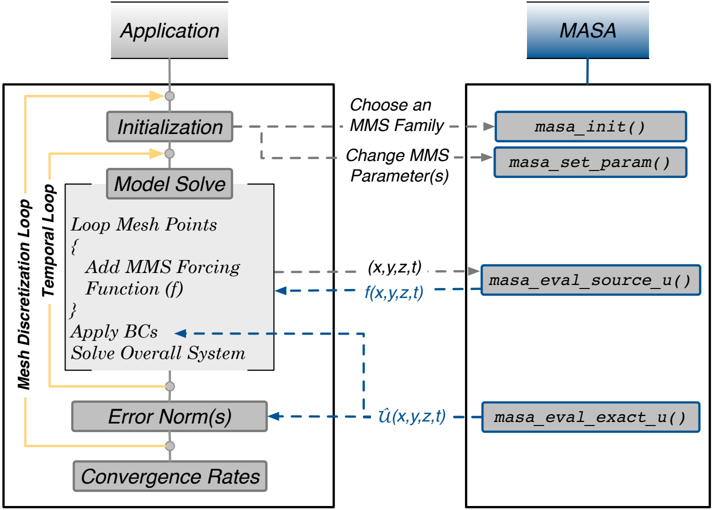
- Sufficient algebraic complexity to exercise all terms
- Physically consistent
  - e.g. solutions should not return negative densities, temperatures, etc.



Available Solutions in MASA 0.41.1

Equations	Dimensions	Time
Euler	1,2,3, axi	Transient, Steady
Non linear heat conduction	1,2,3	Transient, Steady
Navier-Stokes	1,2,3, axi	Transient, Steady
N-S + Sutherland	3	Transient, Steady
N-S + ablation	1	Transient, Steady
Burgers	2	Transient, Steady
Sod Shock Tube	1	Transient
Euler + chemistry	1	Steady
RANS: Spalart-Allmaras	1	Steady
FANS: SA	2	Steady
FANS: SA + wall	2	Steady
Radiation	1	Steady
SMASA: Gaussian	1	Steady

General Verification Approach Using MMS and MASA



Fortran 90: What you need from MASA

```
program main
  use masa
  implicit none

  dx = real(lx)/real(nx)
  dy = real(ly)/real(ny);

  ! initialize the problem
  call masa_init("laplace example","laplace_2d")

  ! evaluate source terms (2D)
  do i=0, nx
     do j=0, ny

        y = j*dy
        x = i*dx

        ! evalulate source term
        field = masa_eval_2d_source_f   (x,y)

        ! evaluate analytical term
        exact_phi = masa_eval_2d_exact_phi (x,y)

     enddo
  enddo

end program main
```

C: What you need from MASA

```
#include <masa.h>

int main()
{
  err += masa_init("laplace example","laplace_2d");

  // grab / set parameter values
  Lx = masa_get_param("Lx");
  masa_set_param("Ly",42.0);

  for(int i=0;i<nx;i++)
    for(int j=0;j<nx;j++)
      {
        x=i*dx;
        y=j*dy;

        // source term
        ffield    = masa_eval_2d_source_f (x,y);

        // manufactured solution
        phi_field = masa_eval_2d_exact_phi(x,y);

       } // finished iterating over space
} //end program
```

Future Solution Development

Single Physics

- Additional RANS models (v^2-f, k-ϵ, etc.)
- Shocks

Stochastic PDEs

- Conjugate Priors
- Fokker-Plank (time evolution of pdfs)
- How do you verify something that is random?

Different physical systems

- Einstein's field equations (General Relativity)
- Schrodinger equation (Quantum Mechanics)
- Black-Scholes (Finance)
- etc.

Importing New Solutions

Requirements

- Latex documents can be loaded directly into MASA documentation
  - Model document detailing analytical solution and source terms
  - Interface documentation detailing parameters and functions
- Source and analytical terms in C/C++/Fortran90
  - Can be integrated into your local MASA copy automagically (perl!)
  - Submit a patch
    - (unit tests would be nice)
- Willingness to share
- Publish these solutions!

- Success of MASA depends on use as a community tool

Shortcomings

Symbolic Shortcomings

- Even with factorization, source terms still massive
- Generating manufactured solutions was a **full time job** at PECOS
- Everything we have discussed has been generated outside of MASA
  - Not thrilled with Maple, Mathematica

Enter Automatic Differentiation

- AD numerically evaluates the derivative of a function
  - applies chain rule repeatedly
- Superior error characteristics (round-off)
- Slow (but we barely care)
- Several libraries: NAG, Sacado, etc.

MASA PDE Examples

Manufactured Solution

```
// Arbitrary manufactured solutions
U.template get<0>() = u_0 + u_x * sin(a_ux * PI * x / L) +
                            u_y * cos(a_uy * PI * y / L);

U.template get<1>() = v_0 + v_x * cos(a_vx * PI * x / L) +
                            v_y * sin(a_vy * PI * y / L);

ADScalar RHO = rho_0 + rho_x * sin(a_rhox * PI * x / L) +
                       rho_y * cos(a_rhoy * PI * y / L);

ADScalar P = p_0 + p_x * cos(a_px * PI * x / L) +
                   p_y * sin(a_py * PI * y / L);
```

MASA PDE Examples

Source Terms: Euler

$$\nabla \cdot (\rho u) = 0$$

$$\nabla \cdot (eu) + p\nabla \cdot u = 0$$

```
// Gas state
ADScalar T  = P / RHO / R;
ADScalar E  = 1. / (Gamma-1.) * P / RHO;
ADScalar ET = E + .5 * U.dot(U);

// Mass, momentum and energy
Scalar   Q_rho   = raw_value(divergence(RHO*U));
RawArray Q_rho_u = raw_value(divergence(RHO*U.outerproduct(U)) +
                   P.derivatives());
 Scalar  Q_rho_e = raw_value(divergence((RHO*ET+P)*U));
```

Future AD work

Future Work

- Automatic Latex Generation
- Latex Parser – MMS generator
- Will this work with complex multiphysics?
- Inverse Problems

Snapshot

Release

- MASA 0.43.2 current release
- `https://github.com/manufactured-solutions/MASA`
- Open source, LGPL V2.1, free

Publications

- “MASA: a library for verification using manufactured analytical solutions”
  - DOI: 10.1007/s00366-012-0267-9
- A TRANSIENT MANUFACTURED SOLUTION FOR THE COMPRESSIBLE NAVIER-STOKES EQUATIONS WITH A POWER LAW VISCOSITY
- Manufactured Solutions for the Favre-Averaged Navier-Stokes Equations with Eddy-Viscosity Turbulence Models
- ”A linear regression model for verification of linear problems using Bayesian calibration“ (in prep)

Conclusions

Summary

- MMS is not a difficult concept, but can be tricky and time consuming
- Must have a high degree of confidence in your verification suite
- MASA is an open source library designed to:
  - Increase use of existing MMS in the community
  - Provide a standardized interface and toolset to the community
  - Serve as an example of high quality verification software
  - Available at: https://red.ices.utexas.edu/projects/software

Conclusions

Thank you!

Have a well verified day.

nick@ices.utexas.edu

Combinatorial Explosion

Explosion in source term size

- Complexity increases with more sophisticated mathematical models
  - Sutherland viscosity model has over 1,612,000 characters
- Large memory requirements (128 GB not sufficient for Sutherland)
- Computational intensity
- segfaults

Hierarchic MMS

- Decompose each equation into sub-terms
- Each term is operated on individually to plug in a manufactured solution
- Resulting expressions are re-combined to regain source term

The Hierarchic MMS

Consider the full 3D Navier-Stokes energy equation:

$$\mathcal{L} = \frac{\partial(\rho e_t)}{\partial t} + \nabla \cdot (\rho \mathbf{u} e_t) + \nabla \cdot \mathbf{q} + \nabla \cdot (p\mathbf{u}) - \nabla \cdot (\boldsymbol{\tau} \cdot \mathbf{u})$$

Decompose:

$$\mathcal{L}_1 = \frac{\partial(\rho e_t)}{\partial t}$$
$$\mathcal{L}_2 = \nabla \cdot (\rho \mathbf{u} e_t)$$
$$\mathcal{L}_3 = \nabla \cdot \mathbf{q}$$
$$\mathcal{L}_4 = \nabla \cdot (p\mathbf{u})$$
$$\mathcal{L}_5 = -\nabla \cdot (\boldsymbol{\tau} \cdot \mathbf{u})$$

Hierarchic MMS extensions:

- Expand each component of divergence
- Transient and steady cases ($\mathcal{L}_1$ = 0)
- Sutherland model only requires altering $\mathcal{L}_5$
- Navier-Stokes $\rightarrow$ Euler and additional terms

Hierarchic MMS Results

Reductions

- Symbolic factorization tricks permit further simplification of these operators:

Term	Before	After	Reduction
$\mathcal{L}_1$	70.1×10^3	1.1×10^3	98.4%
$\mathcal{L}_2$	292.8×10^3	4.0×10^3	98.6%
$\mathcal{L}_3$	11.3×10^3	1.2×10^3	89.3%
$\mathcal{L}_4$	1.5×10^3	5.8×10^2	61.3%
$\mathcal{L}_5$	3.1×10^3	1.3×10^3	58.0%
$\mathcal{L}$	378.8×10^3	8.2×10^3	97.8%

Reduced source term C-output

```c
 RHO = rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L) + rho_t * sin(a_rhot * PI * t / L);
  U = u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L) + u_t * cos(a_ut * PI * t / L);
  V = v_0 + v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_t * sin(a_vt * PI * t / L);
  W = w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L) + w_t * cos(a_wt * PI * t / L);
  P = p_0 + p_x * cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L) + p_t * cos(a_pt * PI * t / L);
  M1 = A_mu * pow(P / R / RHO, 0.3e1 / 0.2e1);
  M2 = P / R / RHO + B_mu;
  MU = M1 / M2;
  Q_e = (0.3e1 * a_ux * u_x * cos(a_ux * PI * x / L) + a_vy * v_y * cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * RHO * U * U / L / 0.2e1 + (a_ux * u_x * cos(a_ux * PI
* x / L) + 0.3e1 * a_vy * v_y * cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * RHO * V * V / L / 0.2e1 + (a_ux * u_x * cos(a_ux * PI * x / L) + a_vy * v_y * cos(a_vy *
PI * y / L) - 0.3e1 * a_wz * w_z * sin(a_wz * PI * z / L)) * PI * RHO * W * W / L / 0.2e1 + (0.4e1 * a_ux * a_ux * u_x * sin(a_ux * PI * x / L) + 0.3e1 * a_uy * a_uy * u_y * cos(a_uy * PI * y
/ L) + 0.3e1 * a_uz * a_uz * u_z * cos(a_uz * PI * z / L)) * MU * PI * PI * U * pow(L, -0.2e1) / 0.3e1 + (0.3e1 * a_vx * a_vx * v_x * cos(a_vx * PI * x / L) + 0.4e1 * a_vy * a_vy * v_y *
sin(a_vy * PI * y / L) + 0.3e1 * a_vz * a_vz * v_z * sin(a_vz * PI * z / L)) * MU * PI * PI * V * pow(L, -0.2e1) / 0.3e1 + (0.3e1 * a_wx * a_wx * w_x * sin(a_wx * PI * x / L) + 0.3e1 * a_wy *
a_wy * w_y * sin(a_wy * PI * y / L) + 0.4e1 * a_wz * a_wz * w_z * cos(a_wz * PI * z / L)) * MU * PI * PI * W * pow(L, -0.2e1) / 0.3e1 + (a_ux * u_x * cos(a_ux * PI * x / L) + a_vy * v_y *
cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * P / (Gamma - 0.1e1) / L - (0.2e1 * a_rhox * a_px * rho_x * p_x * cos(a_rhox * PI * x / L) * sin(a_px * PI * x / L) + 0.2e1
* a_rhoy * a_py * rho_y * p_y * sin(a_rhoy * PI * y / L) * cos(a_py * PI * y / L) + 0.2e1 * a_rhoz * a_pz * rho_z * p_z * cos(a_rhoz * PI * z / L) * sin(a_pz * PI * z / L)) * PI * PI * k *
pow(L, -0.2e1) / R * pow(RHO, -0.2e1) + (U * U + V * V + W * W) * a_rhot * PI * rho_t * cos(a_rhot * PI * t / L) / L / 0.2e1 - a_pt * PI * p_t * sin(a_pt * PI * t / L) / (Gamma - 0.1e1) / L -
(a_uy * u_y * sin(a_uy * PI * y / L) + a_vx * v_x * sin(a_vx * PI * x / L)) * PI * RHO * U * V / L - (a_uz * u_z * sin(a_uz * PI * z / L) - a_wx * w_x * cos(a_wx * PI * x / L)) * PI * RHO * U
* W / L + (a_vz * v_z * cos(a_vz * PI * z / L) + a_wy * w_y * cos(a_wy * PI * y / L)) * PI * RHO * V * W / L + (a_px * a_px * p_x * cos(a_px * PI * x / L) + a_py * a_py * p_y * sin(a_py * PI
* y / L) + a_pz * a_pz * p_z * cos(a_pz * PI * z / L)) * PI * PI * k * pow(L, -0.2e1) / R / RHO - (0.4e1 * a_ux * a_ux * u_x * u_x * pow(cos(a_ux * PI * x / L), 0.2e1) - 0.4e1 * a_ux * a_vy *
u_x * v_y * cos(a_ux * PI * x / L) * cos(a_vy * PI * y / L) + 0.4e1 * a_ux * a_wz * u_x * w_z * cos(a_ux * PI * x / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_uy * a_uy * u_y * u_y *
pow(sin(a_uy * PI * y / L), 0.2e1) + 0.6e1 * a_uy * a_vx * u_y * v_x * sin(a_uy * PI * y / L) * sin(a_vx * PI * x / L) + 0.3e1 * a_uz * a_uz * u_z * u_z * pow(sin(a_uz * PI * z / L), 0.2e1) -
0.6e1 * a_uz * a_wx * u_z * w_x * sin(a_uz * PI * z / L) * cos(a_wx * PI * x / L) + 0.3e1 * a_vx * a_vx * v_x * v_x * pow(sin(a_vx * PI * x / L), 0.2e1) + 0.4e1 * a_vy * a_vy * v_y * v_y *
pow(cos(a_vy * PI * y / L), 0.2e1) + 0.4e1 * a_vy * a_wz * v_y * w_z * cos(a_vy * PI * y / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_vz * a_vz * v_z * v_z * pow(cos(a_vz * PI * z / L), 0.2e1) +
0.6e1 * a_vz * a_wy * v_z * w_y * cos(a_vz * PI * z / L) * cos(a_wy * PI * y / L) + 0.3e1 * a_wx * a_wx * w_x * w_x * pow(cos(a_wx * PI * x / L), 0.2e1) + 0.3e1 * a_wy * a_wy * w_y * w_y *
pow(cos(a_wy * PI * y / L), 0.2e1) + 0.4e1 * a_wz * a_wz * w_z * w_z * pow(sin(a_wz * PI * z / L), 0.2e1)) * MU * PI * PI * pow(L, -0.2e1) / 0.3e1 + (a_ux * u_x * cos(a_ux * PI * x / L) +
a_vy * v_y * cos(a_vy * PI * y / L) - a_wz * w_z * sin(a_wz * PI * z / L)) * PI * P / L - a_ut * PI * u_t * RHO * U * sin(a_ut * PI * t / L) / L + a_vt * PI * v_t * RHO * V * cos(a_vt * PI *
t / L) / L - a_wt * PI * w_t * RHO * W * sin(a_wt * PI * t / L) / L + (U * U + V * V + W * W) * a_rhox * PI * rho_x * U * cos(a_rhox * PI * x / L) / L / 0.2e1 - (U * U + V * V + W * W) *
a_rhoy * PI * rho_y * V * sin(a_rhoy * PI * y / L) / L / 0.2e1 + (U * U + V * V + W * W) * a_rhoz * PI * rho_z * W * cos(a_rhoz * PI * z / L) / L / 0.2e1 - (a_rhox * a_rhox * rho_x *
sin(a_rhox * PI * x / L) + a_rhoy * a_rhoy * rho_y * cos(a_rhoy * PI * y / L) + a_rhoz * a_rhoz * rho_z * sin(a_rhoz * PI * z / L)) * PI * PI * k * P * pow(L, -0.2e1) / R * pow(RHO, -0.2e1) -
(0.2e1 * a_rhox * a_rhox * rho_x * rho_x * pow(cos(a_rhox * PI * x / L), 0.2e1) + 0.2e1 * a_rhoy * a_rhoy * rho_y * rho_y * pow(sin(a_rhoy * PI * y / L), 0.2e1) + 0.2e1 * a_rhoz * a_rhoz *
rho_z * rho_z * pow(cos(a_rhoz * PI * z / L), 0.2e1)) * PI * PI * k * P * pow(L, -0.2e1) / R * pow(RHO, -0.3e1) - (0.3e1 * a_rhox * a_uz * rho_x * u_z * cos(a_rhox * PI * x / L) * sin(a_uz *
PI * z / L) - 0.3e1 * a_rhox * a_wx * rho_x * w_x * cos(a_rhox * PI * x / L) * cos(a_wx * PI * x / L) + 0.3e1 * a_rhoy * a_vz * rho_y * v_z * sin(a_rhoy * PI * y / L) * cos(a_vz * PI * z / L)
+ 0.3e1 * a_rhoy * a_wy * rho_y * w_y * sin(a_rhoy * PI * y / L) * cos(a_wy * PI * y / L) + 0.2e1 * a_rhoz * a_ux * rho_z * u_x * cos(a_rhoz * PI * z / L) * cos(a_ux * PI * x / L) + 0.2e1 *
a_rhoz * a_vy * rho_z * v_y * cos(a_rhoz * PI * z / L) * cos(a_vy * PI * y / L) + 0.4e1 * a_rhoz * a_wz * rho_z * w_z * cos(a_rhoz * PI * z / L) * sin(a_wz * PI * z / L)) * MU * (0.3e1 * B_mu
* R * RHO + P) * PI * PI * W / (B_mu * R * RHO + P) * pow(L, -0.2e1) / RHO / 0.6e1 - a_pz * PI * p_z * Gamma * W * sin(a_pz * PI * z / L) / (Gamma - 0.1e1) / L - (0.3e1 * a_px * a_uy * p_x *
u_y * sin(a_px * PI * x / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_px * a_vx * p_x * v_x * sin(a_px * PI * x / L) * sin(a_vx * PI * x / L) - 0.2e1 * a_py * a_ux * p_y * u_x * cos(a_py * PI * y
/ L) * cos(a_ux * PI * x / L) + 0.4e1 * a_py * a_vy * p_y * v_y * cos(a_py * PI * y / L) * cos(a_vy * PI * y / L) + 0.2e1 * a_py * a_wz * p_y * w_z * cos(a_py * PI * y / L) * sin(a_wz * PI *
z / L) - 0.3e1 * a_pz * a_vz * p_z * v_z * sin(a_pz * PI * z / L) * cos(a_vz * PI * z / L) - 0.3e1 * a_pz * a_wy * p_z * w_y * sin(a_pz * PI * z / L) * cos(a_wy * PI * y / L)) * (0.3e1 * B_mu
* R * RHO + P) * MU * PI * PI * V / (B_mu * R * RHO + P) * pow(L, -0.2e1) / P / 0.6e1 + a_py * PI * p_y * Gamma * V * cos(a_py * PI * y / L) / (Gamma - 0.1e1) / L - (0.3e1 * a_rhox * a_uy *
rho_x * u_y * cos(a_rhox * PI * x / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_rhox * a_vx * rho_x * v_x * cos(a_rhox * PI * x / L) * sin(a_vx * PI * x / L) - 0.2e1 * a_rhoy * a_ux * rho_y * u_x
* sin(a_rhoy * PI * y / L) * cos(a_ux * PI * x / L) + 0.4e1 * a_rhoy * a_vy * rho_y * v_y * sin(a_rhoy * PI * y / L) * cos(a_vy * PI * y / L) + 0.2e1 * a_rhoy * a_wz * rho_y * w_z *
sin(a_rhoy * PI * y / L) * sin(a_wz * PI * z / L) - 0.3e1 * a_rhoz * a_vz * rho_z * v_z * cos(a_rhoz * PI * z / L) * cos(a_vz * PI * z / L) - 0.3e1 * a_rhoz * a_wy * rho_z * w_y * cos(a_rhoz
* PI * z / L) * cos(a_wy * PI * y / L)) * MU * (0.3e1 * B_mu * R * RHO + P) * PI * PI * V / (B_mu * R * RHO + P) * pow(L, -0.2e1) / RHO / 0.6e1 + (0.4e1 * a_rhox * a_ux * rho_x * u_x *
cos(a_rhox * PI * x / L) * cos(a_ux * PI * x / L) - 0.2e1 * a_rhox * a_vy * rho_x * v_y * cos(a_rhox * PI * x / L) * cos(a_vy * PI * y / L) + 0.2e1 * a_rhox * a_wz * rho_x * w_z * cos(a_rhox
* PI * x / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_rhoy * a_uy * rho_y * u_y * sin(a_rhoy * PI * y / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_rhoy * a_vx * rho_y * v_x * sin(a_rhoy * PI * y /
L) * sin(a_vx * PI * x / L) - 0.3e1 * a_rhoz * a_uz * rho_z * u_z * cos(a_rhoz * PI * z / L) * sin(a_uz * PI * z / L) + 0.3e1 * a_rhoz * a_wx * rho_z * w_x * cos(a_rhoz * PI * z / L) *
cos(a_wx * PI * x / L)) * MU * (0.3e1 * B_mu * R * RHO + P) * PI * PI * U / (B_mu * R * RHO + P) * pow(L, -0.2e1) / RHO / 0.6e1 - a_px * PI * p_x * Gamma * U * sin(a_px * PI * x / L) / (Gamma
- 0.1e1) / L + (0.4e1 * a_px * a_ux * p_x * u_x * sin(a_px * PI * x / L) * cos(a_ux * PI * x / L) - 0.2e1 * a_px * a_vy * p_x * v_y * sin(a_px * PI * x / L) * cos(a_vy * PI * y / L) + 0.2e1 *
a_px * a_wz * p_x * w_z * sin(a_px * PI * x / L) * sin(a_wz * PI * z / L) + 0.3e1 * a_py * a_uy * p_y * u_y * cos(a_py * PI * y / L) * sin(a_uy * PI * y / L) + 0.3e1 * a_py * a_vx * p_y * v_x
* cos(a_py * PI * y / L) * sin(a_vx * PI * x / L) - 0.3e1 * a_pz * a_uz * p_z * u_z * sin(a_pz * PI * z / L) * sin(a_uz * PI * z / L) + 0.3e1 * a_pz * a_wx * p_z * w_x * sin(a_pz * PI * z /
L) * cos(a_wx * PI * x / L)) * (0.3e1 * B_mu * R * RHO + P) * MU * PI * PI * U / (B_mu * R * RHO + P) * pow(L, -0.2e1) / P / 0.6e1 - (0.3e1 * a_px * a_uz * p_x * u_z * sin(a_px * PI * x / L)
* sin(a_uz * PI * z / L) - 0.3e1 * a_px * a_wx * p_x * w_x * sin(a_px * PI * x / L) * cos(a_wx * PI * x / L) + 0.3e1 * a_py * a_vz * p_y * v_z * cos(a_py * PI * y / L) * cos(a_vz * PI * z /
L) + 0.3e1 * a_py * a_wy * p_y * w_y * cos(a_py * PI * y / L) * cos(a_wy * PI * y / L) + 0.2e1 * a_pz * a_ux * p_z * u_x * sin(a_pz * PI * z / L) * cos(a_ux * PI * x / L) + 0.2e1 * a_pz *
a_vy * p_z * v_y * sin(a_pz * PI * z / L) * cos(a_vy * PI * y / L) + 0.4e1 * a_pz * a_wz * p_z * w_z * sin(a_pz * PI * z / L) * sin(a_wz * PI * z / L)) * (0.3e1 * B_mu * R * RHO + P) * MU *
PI * PI * W / (B_mu * R * RHO + P) * pow(L, -0.2e1) / P / 0.6e1;
```

Original source term C-output

```
Q_e = -0.2e1 * cos(a_rhox * PI * x / L) * rho_x * sin(a_px * PI * x / L) * k * p_x * a_px * a_rhox * PI * PI * pow(L, -0.2e1) / R * pow(rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y *
cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L), -0.2e1) - 0.2e1 * sin(a_rhoy * PI * y / L) * rho_y * cos(a_py * PI * y / L) * k * p_y * a_py * a_rhoy * PI * PI * pow(L, -0.2e1) /
R * pow(rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L), -0.2e1) - 0.2e1 * cos(a_rhoz * PI * z / L) * rho_z * sin(a_pz * PI * z
/ L) * k * p_z * a_pz * a_rhoz * PI * PI * pow(L, -0.2e1) / R * pow(rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L), -0.2e1) -
(v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z *
sin(a_rhoz * PI * z / L)) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L)) * u_y * sin(a_uy * PI * y / L) * a_uy * PI / L + (w_0 + w_x *
sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L)) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI
* z / L)) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L)) * w_x * cos(a_wx * PI * x / L) * a_wx * PI / L - (w_0 + w_x * sin(a_wx * PI * x
/ L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L)) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) * (u_0 +
u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L)) * u_z * sin(a_uz * PI * z / L) * a_uz * PI / L + sin(a_py * PI * y / L) * k * p_y * a_py * a_py *
PI * PI * pow(L, -0.2e1) / R / (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) + cos(a_px * PI * x / L) * k * p_x * a_px *
a_px * PI * PI * pow(L, -0.2e1) / R / (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) + (w_0 + w_x * sin(a_wx * PI * x / L) +
w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L)) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) * (v_x *
cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0) * v_z * cos(a_vz * PI * z / L) * a_vz * PI / L + cos(a_pz * PI * z / L) * k * p_z * a_pz * a_pz *
PI * PI * pow(L, -0.2e1) / R / (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) - (0.2e1 * pow(cos(a_rhox * PI * x / L), 0.2e1)
* rho_x + sin(a_rhox * PI * x / L) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L))) * (p_0 + p_x * cos(a_px * PI * x / L) +
p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L)) * k * rho_x * a_rhox * a_rhox * PI * PI * pow(L, -0.2e1) / R * pow(rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy *
PI * y / L) + rho_z * sin(a_rhoz * PI * z / L), -0.3e1) - (0.2e1 * pow(sin(a_rhoy * PI * y / L), 0.2e1) * rho_y + cos(a_rhoy * PI * y / L) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y
* cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L))) * (p_0 + p_x * cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L)) * k * rho_y * a_rhoy *
a_rhoy * PI * PI * pow(L, -0.2e1) / R * pow(rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L), -0.3e1) - (0.2e1 * pow(cos(a_rhoz *
PI * z / L), 0.2e1) * rho_z + sin(a_rhoz * PI * z / L) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L))) * (p_0 + p_x *
cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L)) * k * rho_z * a_rhoz * a_rhoz * PI * PI * pow(L, -0.2e1) / R * pow(rho_0 + rho_x * sin(a_rhox * PI * x /
L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L), -0.3e1) + 0.4e1 / 0.3e1 * mu * u_x * v_y * cos(a_ux * PI * x / L) * cos(a_vy * PI * y / L) * a_ux * a_vy * PI * PI *
pow(L, -0.2e1) - 0.4e1 / 0.3e1 * mu * u_x * w_z * cos(a_ux * PI * x / L) * sin(a_wz * PI * z / L) * a_ux * a_wz * PI * PI * pow(L, -0.2e1) - 0.2e1 * mu * u_y * v_x * sin(a_uy * PI * y / L) *
sin(a_vx * PI * x / L) * a_uy * a_vx * PI * PI * pow(L, -0.2e1) + 0.2e1 * mu * u_z * w_x * cos(a_wx * PI * x / L) * sin(a_uz * PI * z / L) * a_uz * a_wx * PI * PI * pow(L, -0.2e1) - 0.4e1 /
0.3e1 * mu * v_y * w_z * cos(a_vy * PI * y / L) * sin(a_wz * PI * z / L) * a_vy * a_wz * PI * PI * pow(L, -0.2e1) - 0.2e1 * mu * v_z * w_y * cos(a_vz * PI * z / L) * cos(a_wy * PI * y / L) *
a_vz * a_wy * PI * PI * pow(L, -0.2e1) + (w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L)) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y
* cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) * (v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0) * w_y * cos(a_wy * PI * y /
L) * a_wy * PI * (-(0.3e1 * pow(w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L), 0.2e1) + pow(v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy *
PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0, 0.2e1) + pow(u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L), 0.2e1)) * (rho_0 + rho_x *
sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) / L / 0.2e1 - Gamma * (p_0 + p_x * cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z
* cos(a_pz * PI * z / L)) / L / (Gamma - 0.1e1)) * w_z * sin(a_wz * PI * z / L) * a_wz * PI / L + (-0.1e1) * (v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI
* z / L) + v_0) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI
* y / L) + u_z * cos(a_uz * PI * z / L)) * v_x * sin(a_vx * PI * x / L) * a_vx * PI * ((pow(u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L),
0.2e1) + pow(w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L), 0.2e1) + 0.3e1 * pow(v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L)
+ v_z * sin(a_vz * PI * z / L) + v_0, 0.2e1)) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) / L / 0.2e1 + Gamma * (p_0 +
p_x * cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L)) / L / (Gamma - 0.1e1)) * v_y * cos(a_vy * PI * y / L) * a_vy * PI / L - Gamma * (w_0 + w_x *
sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L)) * sin(a_pz * PI * z / L) * p_z * a_pz * PI / L / (Gamma - 0.1e1) + 0.1e1 / 0.2e1 * cos(a_rhoz * PI * z /
L) * (w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L)) * (pow(u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z *
cos(a_uz * PI * z / L), 0.2e1) + pow(w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L), 0.2e1) + pow(v_x * cos(a_vx * PI * x / L) + v_y *
sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0, 0.2e1)) * rho_z * a_rhoz * PI * ((pow(w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z
/ L), 0.2e1) + pow(v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0, 0.2e1) + 0.3e1 * pow(u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy *
PI * y / L) + u_z * cos(a_uz * PI * z / L), 0.2e1)) * (rho_0 + rho_x * sin(a_rhox * PI * x / L) + rho_y * cos(a_rhoy * PI * y / L) + rho_z * sin(a_rhoz * PI * z / L)) / L / 0.2e1 + Gamma *
(p_0 + p_x * cos(a_px * PI * x / L) + p_y * sin(a_py * PI * y / L) + p_z * cos(a_pz * PI * z / L)) / L / (Gamma - 0.1e1)) * u_x * cos(a_ux * PI * x / L) * a_ux * PI / L - Gamma * (u_0 + u_x *
sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L)) * sin(a_px * PI * x / L) * p_x * a_px * PI / L / (Gamma - 0.1e1) + Gamma * (v_x * cos(a_vx * PI * x / L)
+ v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0) * cos(a_py * PI * y / L) * p_y * a_py * PI / L / (Gamma - 0.1e1) + 0.4e1 / 0.3e1 * (-pow(cos(a_ux * PI * x / L), 0.2e1) *
u_x + sin(a_ux * PI * x / L) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L))) * mu * u_x * a_ux * a_ux * PI * PI * pow(L, -0.2e1) +
(-pow(sin(a_uy * PI * y / L), 0.2e1) * u_y + cos(a_uy * PI * y / L) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L))) * mu * u_y * a_uy *
a_uy * PI * PI * pow(L, -0.2e1) + (-pow(sin(a_uz * PI * z / L), 0.2e1) * u_z + cos(a_uz * PI * z / L) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz *
PI * z / L))) * mu * u_z * a_uz * a_uz * PI * PI * pow(L, -0.2e1) - (pow(sin(a_vx * PI * x / L), 0.2e1) * v_x - cos(a_vx * PI * x / L) * (v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI *
y / L) + v_z * sin(a_vz * PI * z / L) + v_0)) * mu * v_x * a_vx * a_vx * PI * PI * pow(L, -0.2e1) - 0.4e1 / 0.3e1 * (pow(cos(a_vy * PI * y / L), 0.2e1) * v_y - sin(a_vy * PI * y / L) * (v_x *
cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0)) * mu * v_y * a_vy * a_vy * PI * PI * pow(L, -0.2e1) - (pow(cos(a_vz * PI * z / L), 0.2e1) * v_z -
sin(a_vz * PI * z / L) * (v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0)) * mu * v_z * a_vz * a_vz * PI * PI * pow(L, -0.2e1) +
(-pow(cos(a_wx * PI * x / L), 0.2e1) * w_x + sin(a_wx * PI * x / L) * (w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L))) * mu * w_x * a_wx *
a_wx * PI * PI * pow(L, -0.2e1) + (-pow(cos(a_wy * PI * y / L), 0.2e1) * w_y + sin(a_wy * PI * y / L) * (w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz *
PI * z / L))) * mu * w_y * a_wy * a_wy * PI * PI * pow(L, -0.2e1) + 0.4e1 / 0.3e1 * (-pow(sin(a_wz * PI * z / L), 0.2e1) * w_z + cos(a_wz * PI * z / L) * (w_0 + w_x * sin(a_wx * PI * x / L) +
w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L))) * mu * w_z * a_wz * a_wz * PI * PI * pow(L, -0.2e1) + cos(a_rhox * PI * x / L) * (u_0 + u_x * sin(a_ux * PI * x / L) + u_y *
cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L)) * (pow(u_0 + u_x * sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L), 0.2e1) + pow(w_0 + w_x *
sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI * z / L), 0.2e1) + pow(v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) +
v_0, 0.2e1)) * rho_x * a_rhox * PI / L / 0.2e1 - sin(a_rhoy * PI * y / L) * (v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0) * (pow(u_0 + u_x
* sin(a_ux * PI * x / L) + u_y * cos(a_uy * PI * y / L) + u_z * cos(a_uz * PI * z / L), 0.2e1) + pow(w_0 + w_x * sin(a_wx * PI * x / L) + w_y * sin(a_wy * PI * y / L) + w_z * cos(a_wz * PI *
z / L), 0.2e1) + pow(v_x * cos(a_vx * PI * x / L) + v_y * sin(a_vy * PI * y / L) + v_z * sin(a_vz * PI * z / L) + v_0, 0.2e1)) * rho_y * a_rhoy * PI / L / 0.2e1;
```

Shortcomings

Symbolic Shortcomings

- Hierarchic decomposition is an improvement, but is it enough?
  - Even with factorization, source terms still massive
  - Generating manufactured solutions was a **full time job** at PECOS
- Everything we have discussed has been generated outside of MASA
  - Not thrilled with Maple, Mathematica

Enter Automatic Differentiation

- AD numerically evaluates the derivative of a function
  - applies chain rule repeatedly
- Superior error characteristics (round-off)
- Slow (but we barely care)
- Several libraries: NAG, Sacado, etc.

Review: Complex Numbers

- A new element i
- Take the quotient with $i^2 \equiv -1$

$$\mathbb{C}[\mathbb{R}] \equiv \{a + bi : a, b \in \mathbb{R}\}$$

- Arithmetic:

$$\begin{aligned}
(a + bi) + (c + di) &= ((a + c) + (b + d)i) \\
(a + bi) - (c + di) &= ((a + c) - (b + d)i) \\
(a + bi) \times (c + di) &= (ac) + adi + bci + bdi^2 \\
&= ((ac) + (ad + bc)i - (bd))
\end{aligned}$$

“Dual Numbers” - [Clifford 1873],[Study 1891]

- A new element ϵ
- Closed under addition and multiplication:

$$\left\{ \sum_{i=0}^{m} a_i \epsilon^i : a_i \in \mathbb{R}, m < \infty \right\}$$

- Take the quotient with $\epsilon^2 \equiv 0$

$$\mathbb{D}[\mathbb{R}] \equiv \{a + b\epsilon : a, b \in \mathbb{R}\}$$

- Used with quaternions to represent rotations and translations
- Arithmetic:

$$\begin{aligned}
(a + b\epsilon) + (c + d\epsilon) &= ((a + c) + (b + d)\epsilon) \\
(a + b\epsilon) - (c + d\epsilon) &= ((a + c) - (b + d)\epsilon) \\
(a + b\epsilon) \times (c + d\epsilon) &= (ac) + ad\epsilon + bc\epsilon + bd\epsilon^2 \\
&= ((ac) + (ad + bc)\epsilon)
\end{aligned}$$

"Hyper-dual Numbers" - [Fike 2009]

- Add two new elements ϵ_1, ϵ_2 to $\mathbb{R}$
- Take the quotient with $\epsilon_1^2 \equiv \epsilon_2^2 \equiv 0$

$$\mathbb{H}[\mathbb{R}] \equiv \{a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2 : a, b, c, d \in \mathbb{R}\}$$

- Arithmetic:

$$(a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2) + (e + f\epsilon_1 + g\epsilon_2 + h\epsilon_1\epsilon_2) =$$
$$((a + e) + (b + f)\epsilon_1 + (c + g)\epsilon_2 + (d + h)\epsilon_1\epsilon_2)$$
$$(a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2) - (e + f\epsilon_1 + g\epsilon_2 + h\epsilon_1\epsilon_2) =$$
$$((a - e) + (b - f)\epsilon_1 + (c - g)\epsilon_2 + (d - h)\epsilon_1\epsilon_2)$$
$$(a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2) \times (e + f\epsilon_1 + g\epsilon_2 + h\epsilon_1\epsilon_2) =$$
$$(ae) + (af + be)\epsilon_1 + (ag + ce)\epsilon_2 + (ah + de + bg + cf)\epsilon_1\epsilon_2$$

“Hyper-Dual Numbers”

- With $\epsilon_1^2 \equiv \epsilon_2^2 \equiv 0 \equiv (\epsilon_1\epsilon_2)^2 = 0$
- Where $X \equiv x + \epsilon_1 + \epsilon_2$, we find:

$$f(X) = f(x) + f'(x)\epsilon_1 + f'(x)\epsilon_2 + f''(x)\epsilon_1\epsilon_2$$

- The Taylor series truncates exactly at the second-derivative term
- Using hyper-dual numbers results in first- and second-derivative calculations that are exact, regardless of step size
- Methods for computing exact higher derivatives can be created by using more non-real parts (ϵ_3, for instance)
  - Accomplished using Templates in C++