Accuracy verification by means of exact and manufactured solutions

Edwin van der Weide
Department of Mechanical Engineering
University of Twente

UNIVERSITY OF TWENTE.

Thomas D. Economon
Multiphysics Modeling and Simulation
Robert Bosch LLC

BOSCH

Outline

- Motivation
- Manufactured solutions
- Implementation in SU2
- A sample test case, Navier-Stokes on a unit quad
- Results
- Conclusions

Motivation

Predictive simulations: do we solve the correct equations?

Validation => Set of validation cases (https://github.com/su2code/VandV)

Another important question: do we solve the equations correctly?

Verification => Set of verification cases (this work)

Verification should happen before validation!!!

Manufactured Solutions

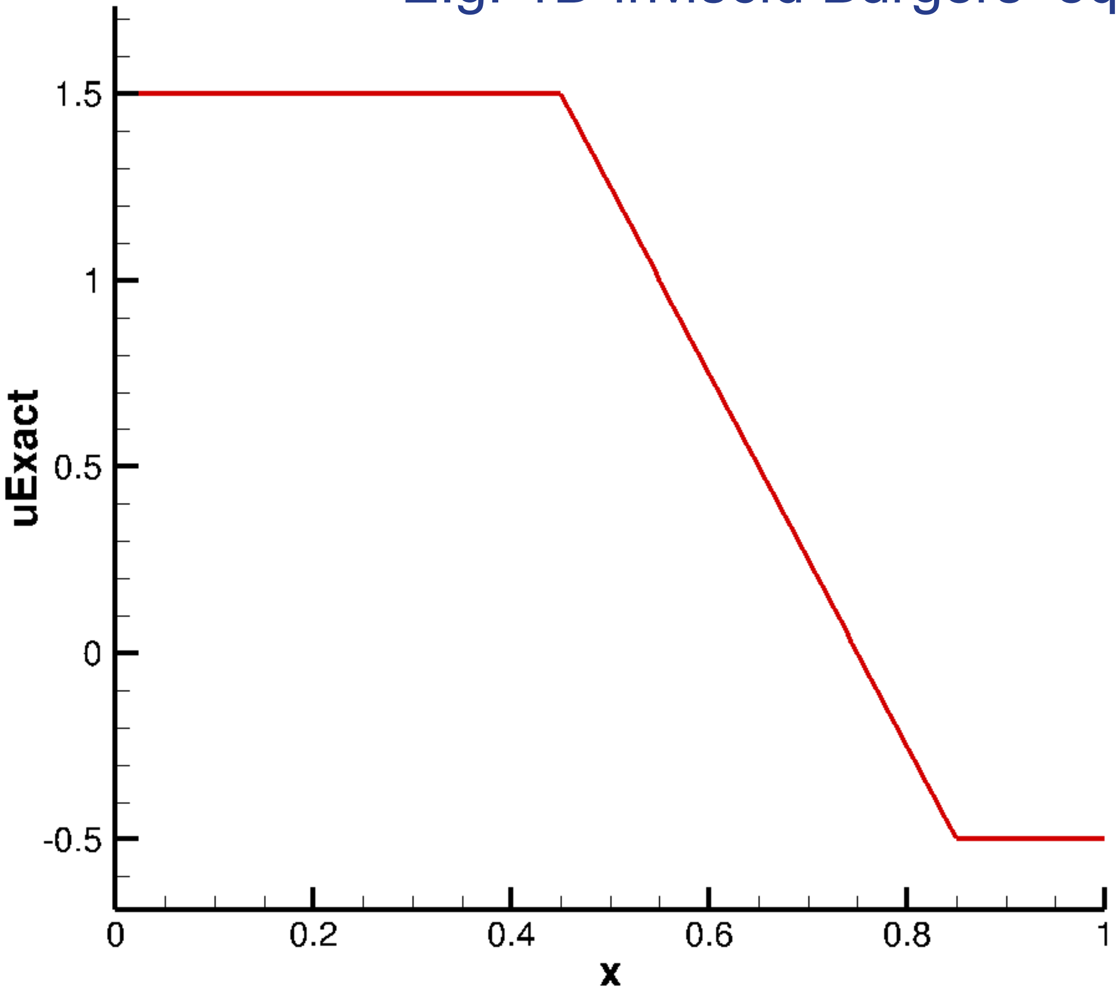
- Rigorous verification: Exact solution must be known. Integral quantities (e.g. forces) do not suffice.
- Limited number of exact solutions for Navier-Stokes (Couette, Hagen-Poiseuille). Usually too simple for a good assessment.
- Therefore: Manufactured Solutions
  - Idea: Manufacture a (sufficiently smooth) solution
  - Modify the governing equations (add a source term)
  - Solve the modified equations.

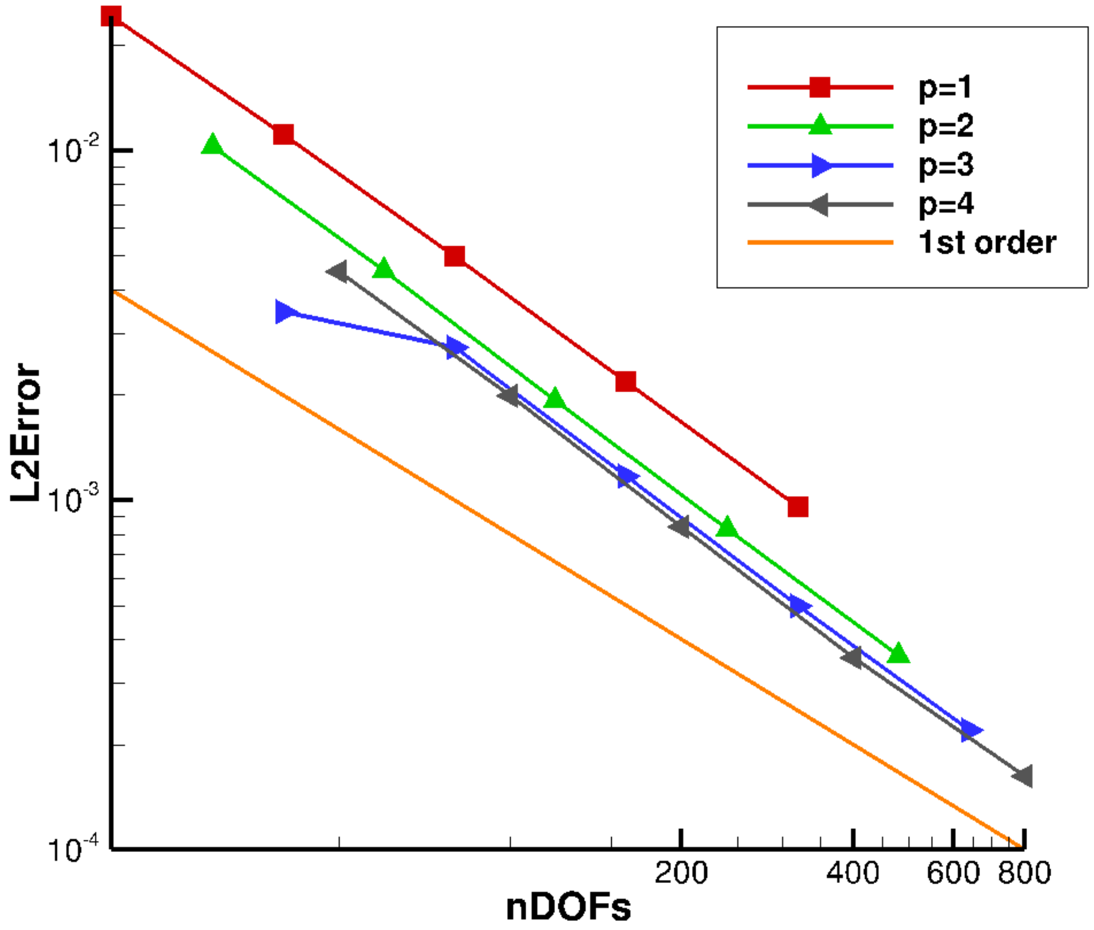
$$\frac{\partial U}{\partial t} + \frac{\partial F_i}{\partial x_i} = S$$

Smoothness requirement

Solutions must be sufficiently smooth to obtain design accuracy of the discretization schemes

E.g. 1D inviscid Burgers' equation with piecewise linear solution





Implementation in SU2

Requirements (see also PR 672)

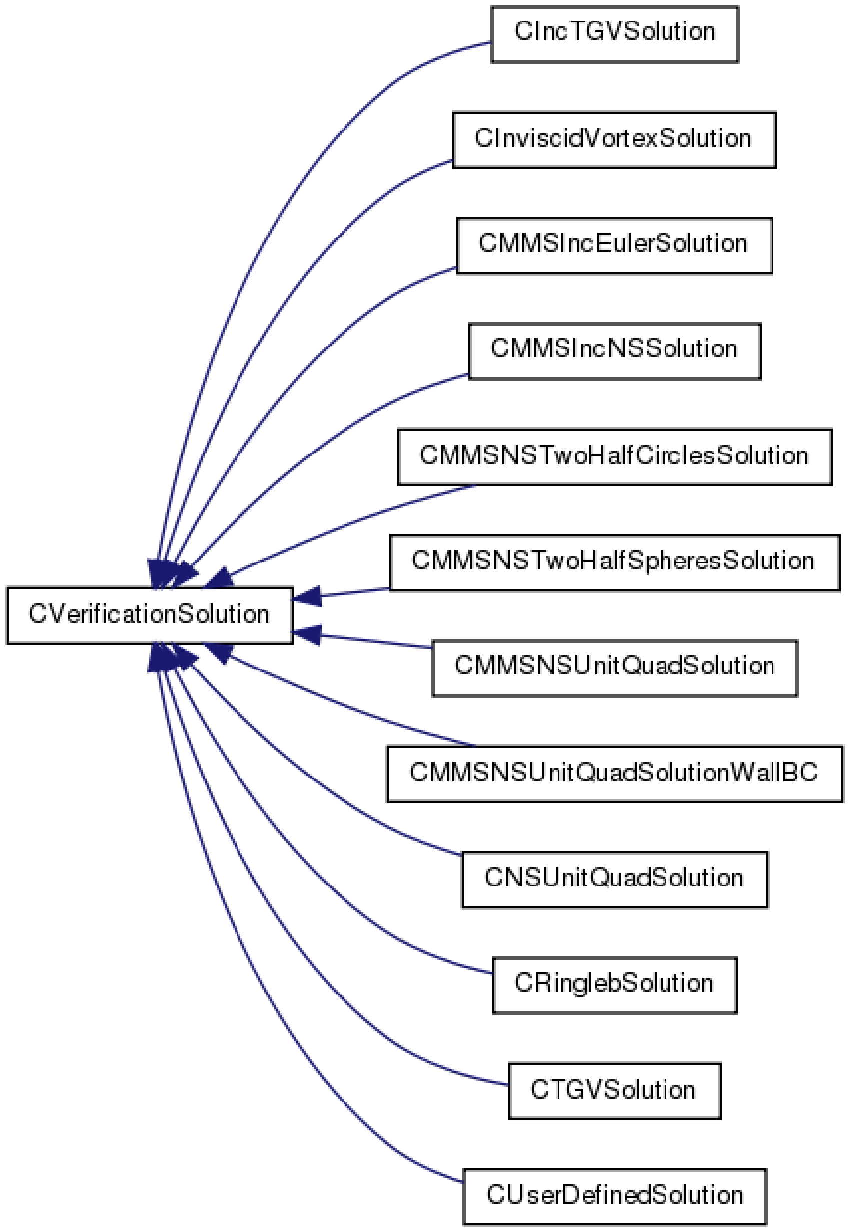
- Lightweight and small memory footprint
- Reusable by different solvers (so far FV and DG)
- Require almost no changes to the solver classes
- Easy to add new cases
- Primary use: formal verification by analytical or manufactured solutions
- Secondary uses:
  - Imposing initial conditions
  - Imposing time dependent boundary conditions

Polymorphism

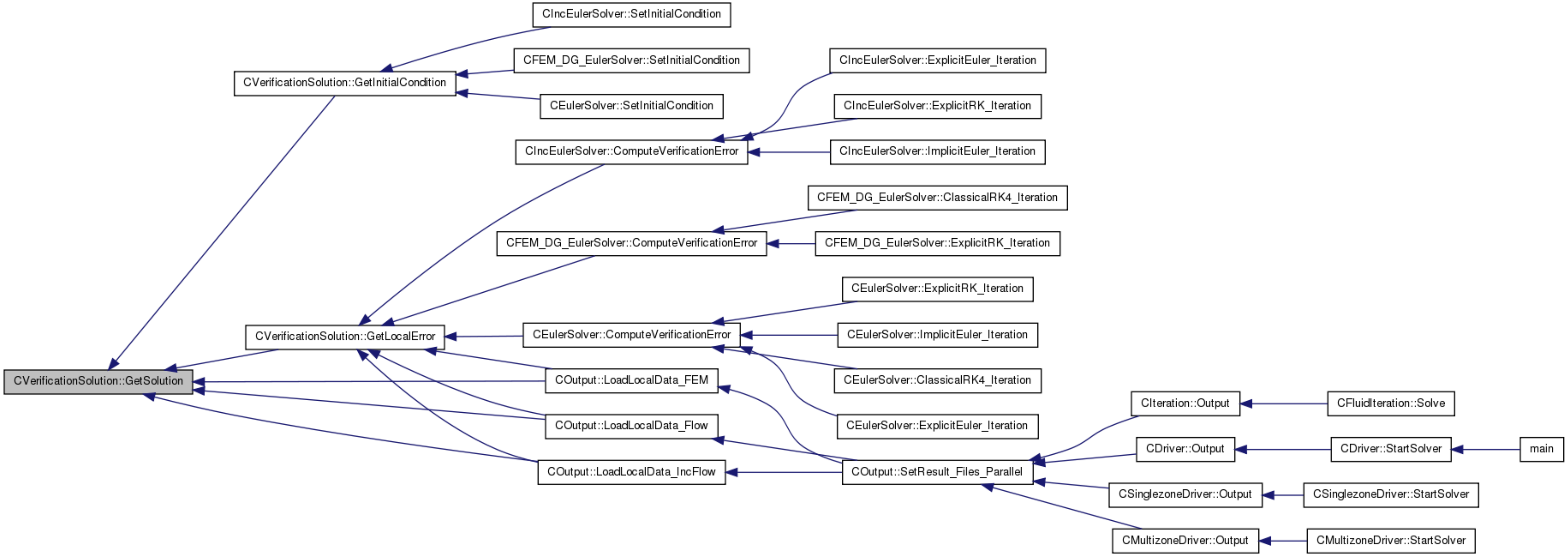
Base class CVerificationSolution handles everything
(except the instantiation in CSolver::SetVerification_Solution)

- Set initial condition
- Get the exact solution for error analysis (possibly time dependent)
- Get the boundary state (possibly time dependent) for BC handling via BC_Custom
- Get the MMS source terms (possibly time dependent)
- Indicate whether or not a solution is manufactured
- Indicate whether or not the exact solution is known

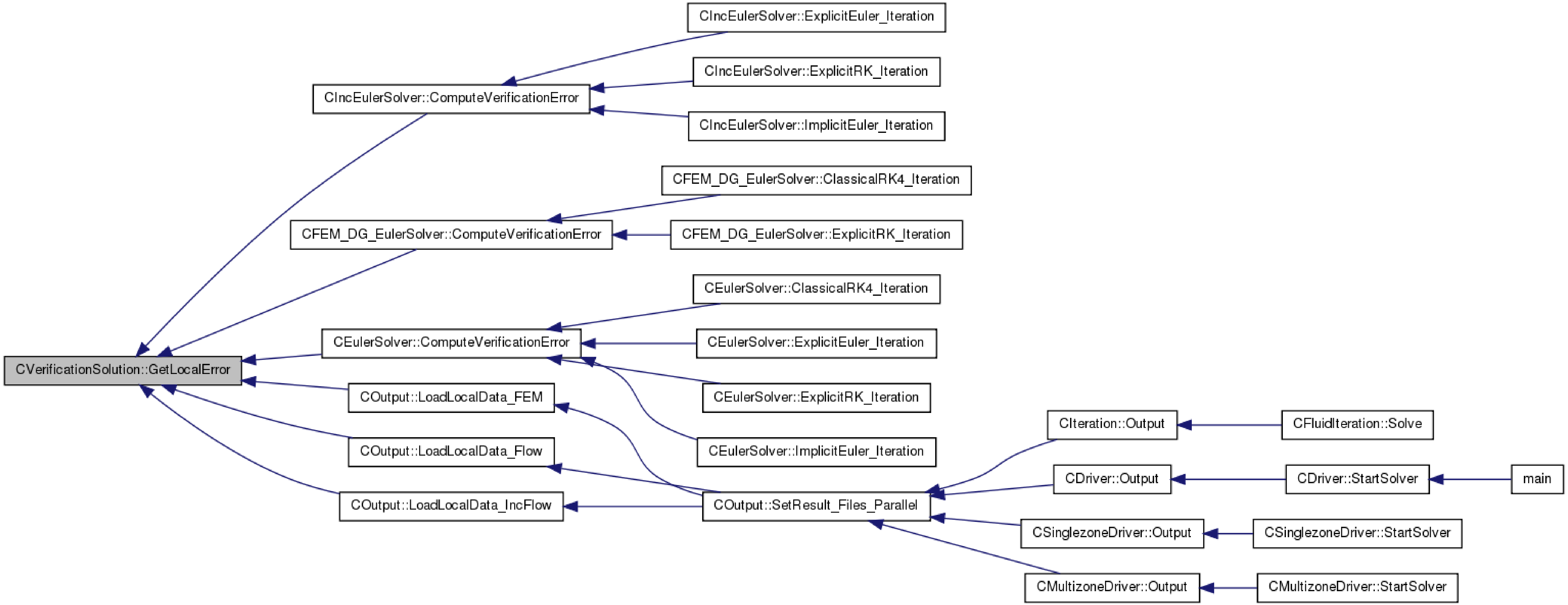
CVerificationSolution Inheritance



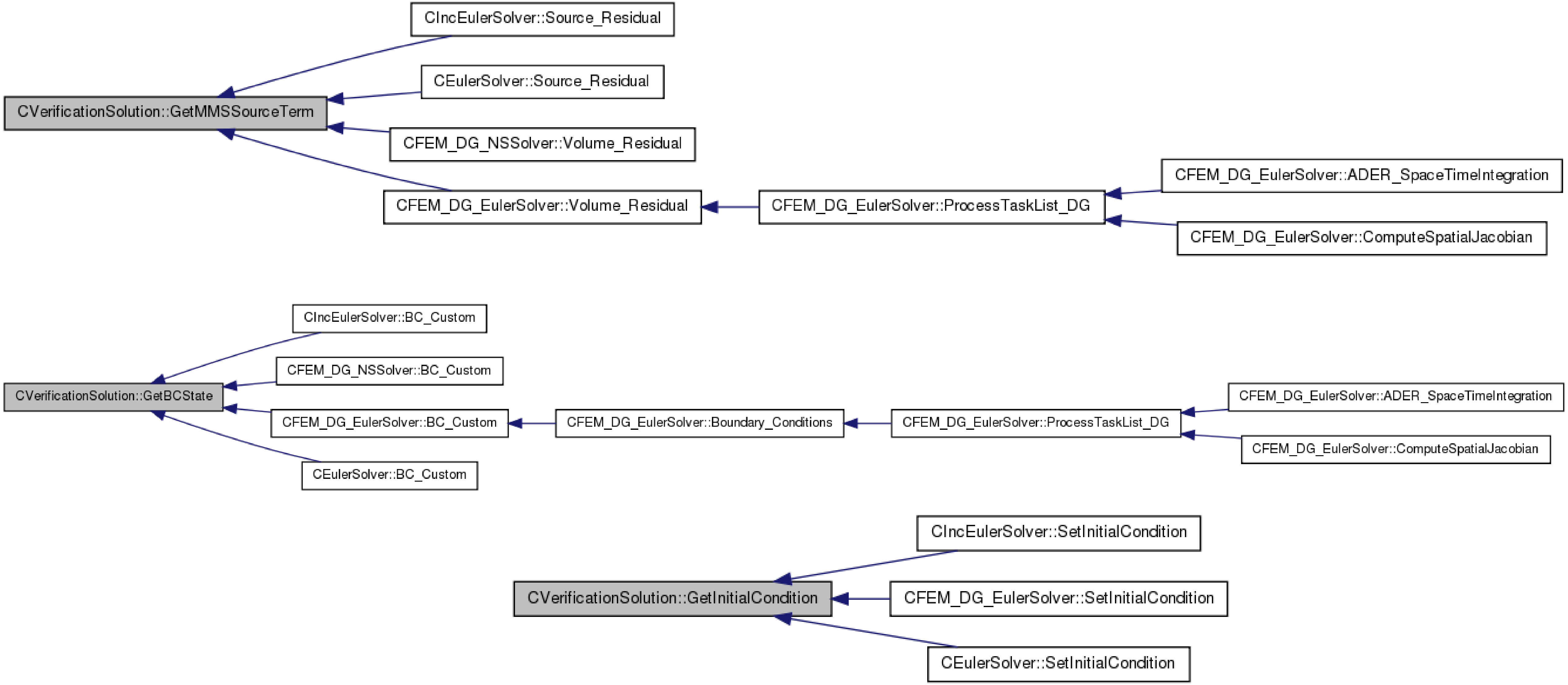
Some call graphs of interest



Some call graphs of interest



Some call graphs of interest



A sample test case, compressible Navier-Stokes on a unit quad

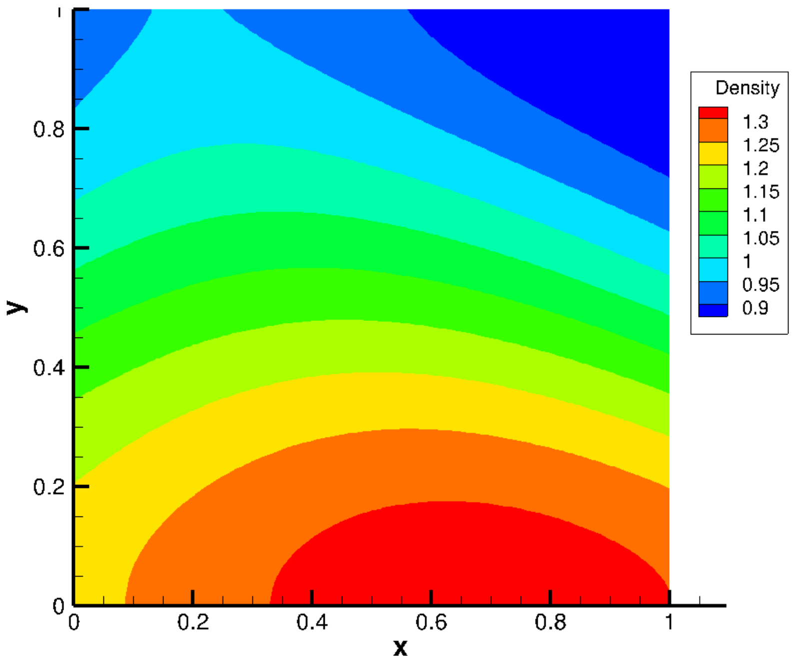
(C.J. Roy et. al., International Journal for Numerical Methods in Fluids, vol. 44, issue 6, pp. 599-620, 2004)

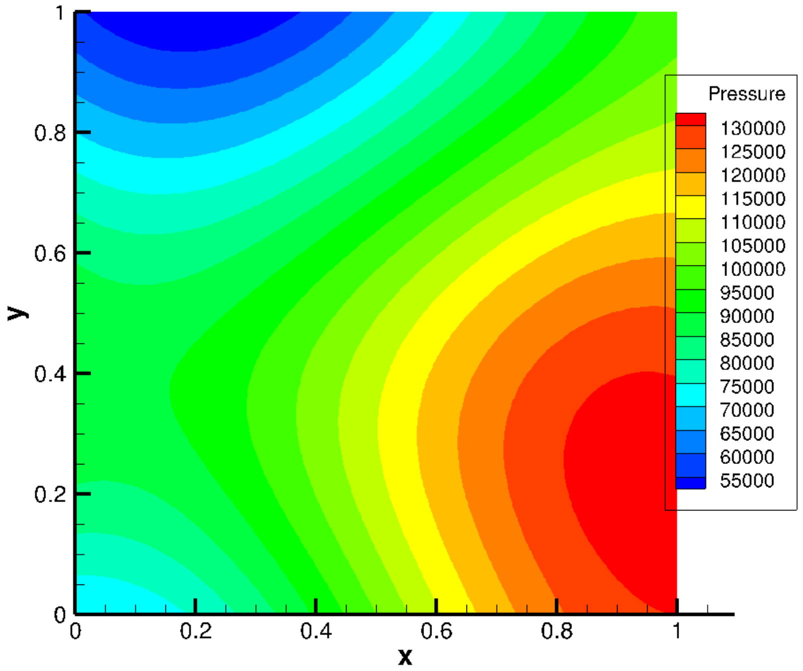
$$\rho = \rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y \cos\left(\frac{a_{\rho y}\pi y}{L}\right) + \rho_{xy} \cos\left(\frac{a_{\rho xy}\pi xy}{L^2}\right)$$

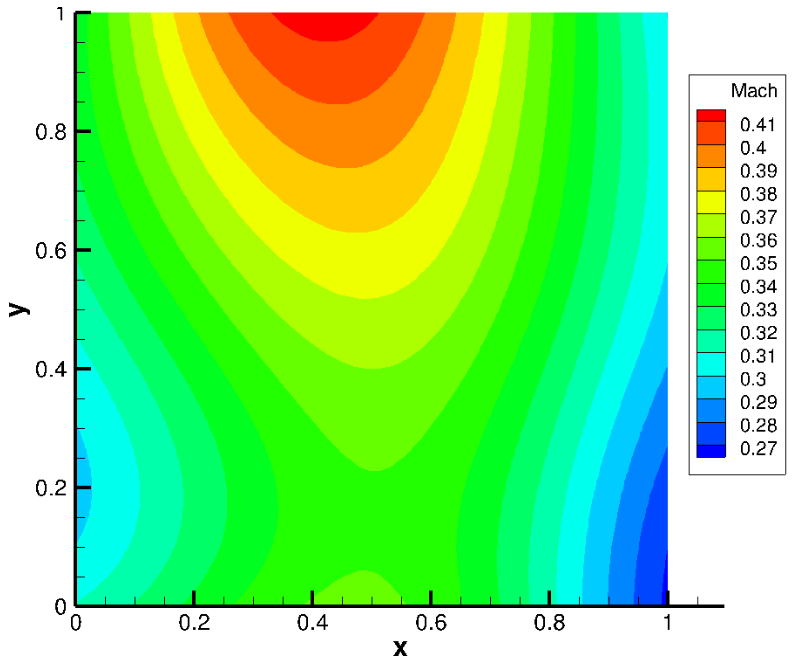
$$u = u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y \cos\left(\frac{a_{uy}\pi y}{L}\right) + u_{xy} \cos\left(\frac{a_{uxy}\pi xy}{L^2}\right)$$

$$v = v_0 + v_x \cos\left(\frac{a_{vx}\pi x}{L}\right) + v_y \sin\left(\frac{a_{vy}\pi y}{L}\right) + v_{xy} \cos\left(\frac{a_{vxy}\pi xy}{L^2}\right)$$

$$p = p_0 + p_x \cos\left(\frac{a_{px}\pi x}{L}\right) + p_y \sin\left(\frac{a_{py}\pi y}{L}\right) + p_{xy} \sin\left(\frac{a_{pxy}\pi xy}{L^2}\right)$$







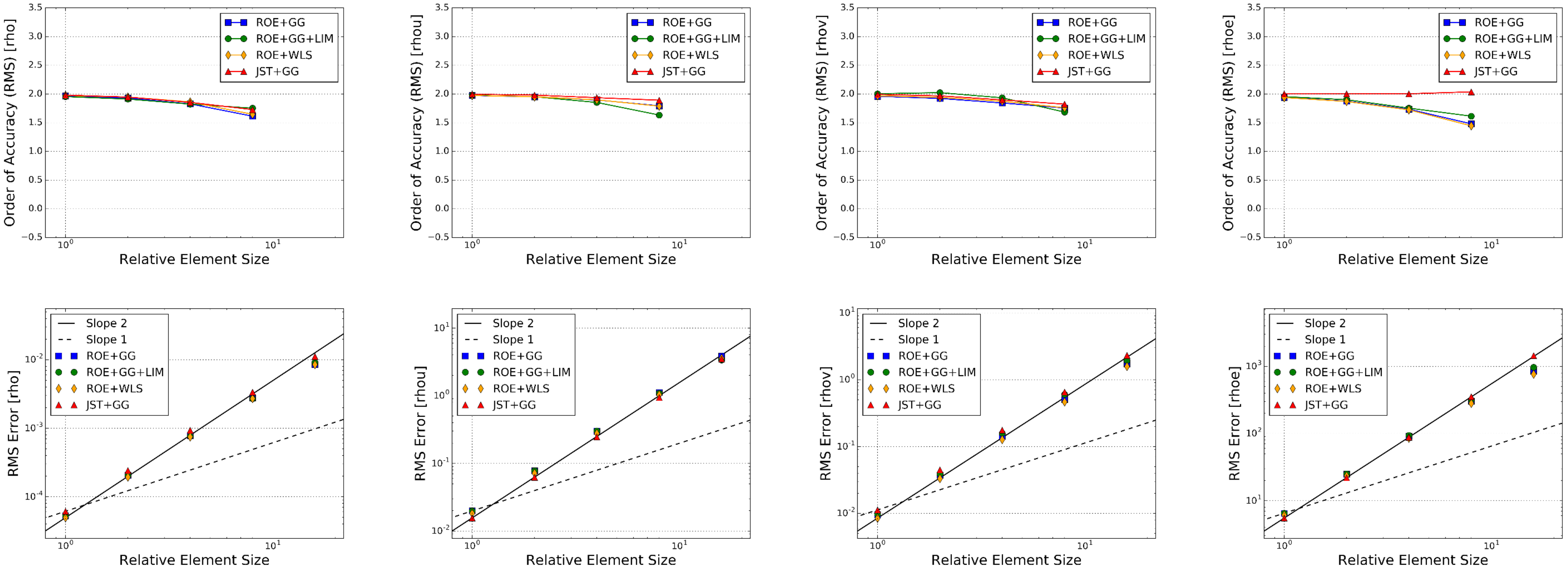
MMS Source term, automatically generated by SymPy or Maple

```
const su2double t321 = t285 * rho_xy;
const su2double t324 = t42 * t42;
const su2double t325 = rho_y * rho_y;
const su2double t327 = t311 * rho_y;
const su2double t330 = t7 * t7;
const su2double t331 = rho_x * rho_x;
const su2double t336 = rho_0 * rho_0;
const su2double t337 = 0.2e1 * rho_x * rho_0 * t38 + t319 * t318 + 0.2e1 * t44 * t321 + t325 * t324 + 0.2e1 * t42 * t327 - t331 * t330 + t331 + t336;
const su2double t338 = 0.1e1 / t337;
const su2double t340 = 0.1e1 / RGas;
const su2double t341 = t340 * t13;
const su2double t349 = t106 * t285 * P_xy;
const su2double t365 = t337 * t337;
const su2double t366 = 0.1e1 / t365;
const su2double t369 = a_rhoxy * t319 * t44;
const su2double t376 = a_rhoxy * t321;
const su2double t398 = u_xy * y * t177;
const su2double t399 = t120 * t124;
const su2double t402 = a_ux * u_x * t48;
const su2double t404 = a_vy * v_y * t80;
const su2double t428 = (-t13 * (t137 * t138 * L + t174 * t177) * Pi - t13 * (t165 * t166 * L + v_xy * y * t124) * Pi) * Viscosity;
const su2double t430 = t203 + t205;
const su2double t442 = a_Py * a_Py;
const su2double t446 = P_xy * t150 * t248;
const su2double t452 = P_y * t201;
const su2double t454 = a_Pxy * x;
const su2double t457 = (t452 * L * a_Py + t261 * t454) * rho_xy;
const su2double t460 = t19 * x * t251;
const su2double t474 = a_rhoy * a_rhoy;
const su2double t475 = t474 * rho_y;
const su2double t480 = P_xy * t59;
const su2double t486 = t235 * t2;
const su2double t564 = t36 * ( t221 * t219 + t223 * t22 / 0.2e1 + (t171 * t77 + t56 * t36) * t46 - t101 + t108) + t56 * t243
                     + Conductivity * t341 * t338 * (t44 * (t234 * t246 + t252 * t250) * rho_xy - t267 * a_rhoxy * t264
                     - t19 * t219 * y * t10 - t277 * t112 * t273 - t106 * t7 * t2 * t4 * P_xy * rho_x * t260 + t288 * t250
                     + (t42 * P_x * t233 * t2 * Pi * rho_y * t245 - t238 * t38 * t2 * Pi * t291)
                     + t107 * t104 * t296 * t1 + t311 * t234 * t2 * t246 - t308 * t306 * t291) * L) * Pi;
const su2double t565 = -0.2e1 * (t44 * t7 * t3 * rho_xy * rho_x * a_rhox + t42 * t7 * t3 * rho_y * rho_x * a_rhox
                     + rho_x * rho_0 * t7 * t2 * t4 + t308 * a_rhox * t331 * t7 - t267 * t369 - t267 * t376) * Conductivity * t341 * t366
                     * (t44 * t264 - t19 * t239 * y * t10 - t349 * t260 + (t311 * P_x * t99 * a_Px + t42 * t258 * a_Px * rho_y
                     + t237 * t296 * t1 + t7 * t306 * t1) * L) * Pi - fourThird * t36 * t132 * t131
                     - fourThird * t56 * t132 * (-t398 + t399 / 0.2e1 + (t402 - t404 / 0.2e1) * L) * Pi;
const su2double t566 = -t77 * t192 - t171 * t428 + t77 * ( t221 * t430 + t223 * t64 / 0.2e1 + (t143 * t36 + t88 * t77) * t46 + t203 + t205)
                     + t88 * t243 - Conductivity * t340 * t338 * t13 * (t44 * (-t236 * Pi * t442 - t252 * t446) * rho_xy
                     - t460 * a_rhoxy * t457 + t19 * t430 * x * t10 + t277 * t150 * t273 - t106 * t59 * t2 * t40 * P_xy * rho_y * t454
                     - t288 * t446 + (-P_y * t38 * t486 * Pi * rho_x * t442 - t42 * P_y * t486 * Pi * rho_y * t442
                     - P_y * t486 * Pi * rho_0 * t442 + t238 * t42 * t2 * Pi * t475 + t204 * t104 * t480 * t58
                     + t42 * t3 * t306 * t475) * L) * Pi;
const su2double t567 = 0.2e1 * (-t44 * t59 * t3 * rho_xy * rho_y * a_rhoy
                     - t60 * a_rhoy * t325 * t42 - t60 * a_rhoy * t327 - t460 * t369 - t460 * t376) * Conductivity * t340 * t366 * t13
                     * (t44 * t457 + t19 * t239 * x * t10 + t349 * t454 + (P_y * t38 * t201 * a_Py * rho_x + t42 * t452 * a_Py * rho_y
                     + t452 * a_Py * rho_0 + t237 * t480 * t58 + t59 * t306 * t58) * L) * Pi - t36 * t161
                     - t143 * t428 + 0.2e1 / 0.3e1 * t77 * t132 * t215 + 0.2e1 / 0.3e1 * t88 * t132 * (-t398 + 0.2e1 * t399
                     + (t402 - 0.2e1 * t404) * L) * Pi;

val_source[0]      = t88 * t46 + t77 * t64 + t37 + t57;
val_source[1]      = t91 * t22 + 0.2e1 * t56 * t93 - t101 + t108 - fourThird * t132 * t131 + t77 * t36 * t64 + t77 * t143 * t46 + t88 * t93 - t161;
val_source[2]      = t77 * t37 + t77 * t57 + t171 * t93 - t192 + t193 * t64 + 0.2e1 * t88 * t77 * t46 + t203 + t205 + 0.2e1 / 0.3e1 * t132 * t215;
val_source[3]      = 0.0;
val_source[nDim+1] = t564 + t565 + t566 + t567;
```

Case publicly available for reproduction: https://github.com/su2code/VandV/tree/master/mms/fvm_navierstokes

FVM Compressible Navier-Stokes MMS Results

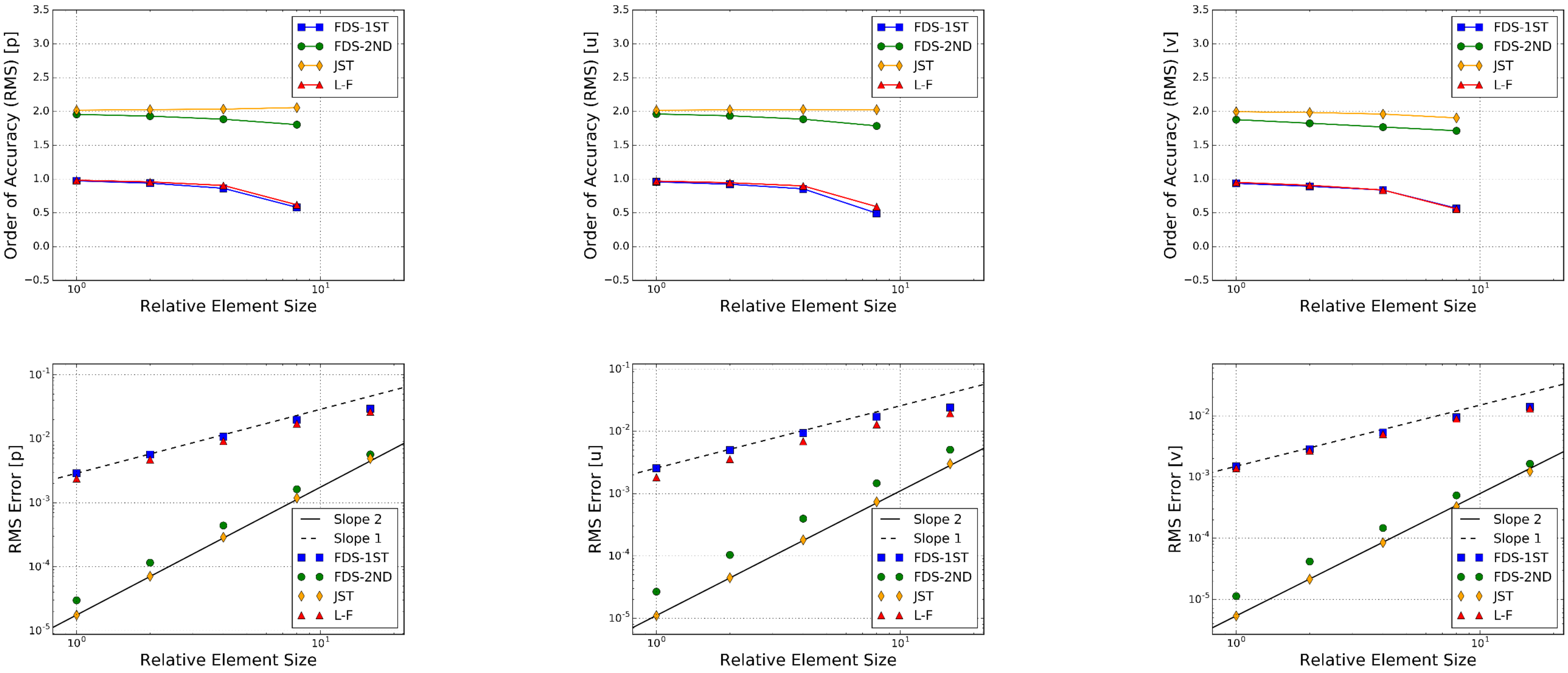


Triangular NxN grids (9,17,33,65,129) with 2nd order schemes -> Observed accuracy asymptotes to expected value of 2 for all cases!

Roe = Roe MUSCL, JST = Jameson-Schmidt-Turkel, GG = Green-Gauss, LIM = Venkatakrishnan-Wang limiter, WLS = Weighted Least-Squares

Case publicly available for reproduction: https://github.com/su2code/VandV/tree/master/mms/fvm_incomp_euler

FVM Incompressible Euler MMS Results



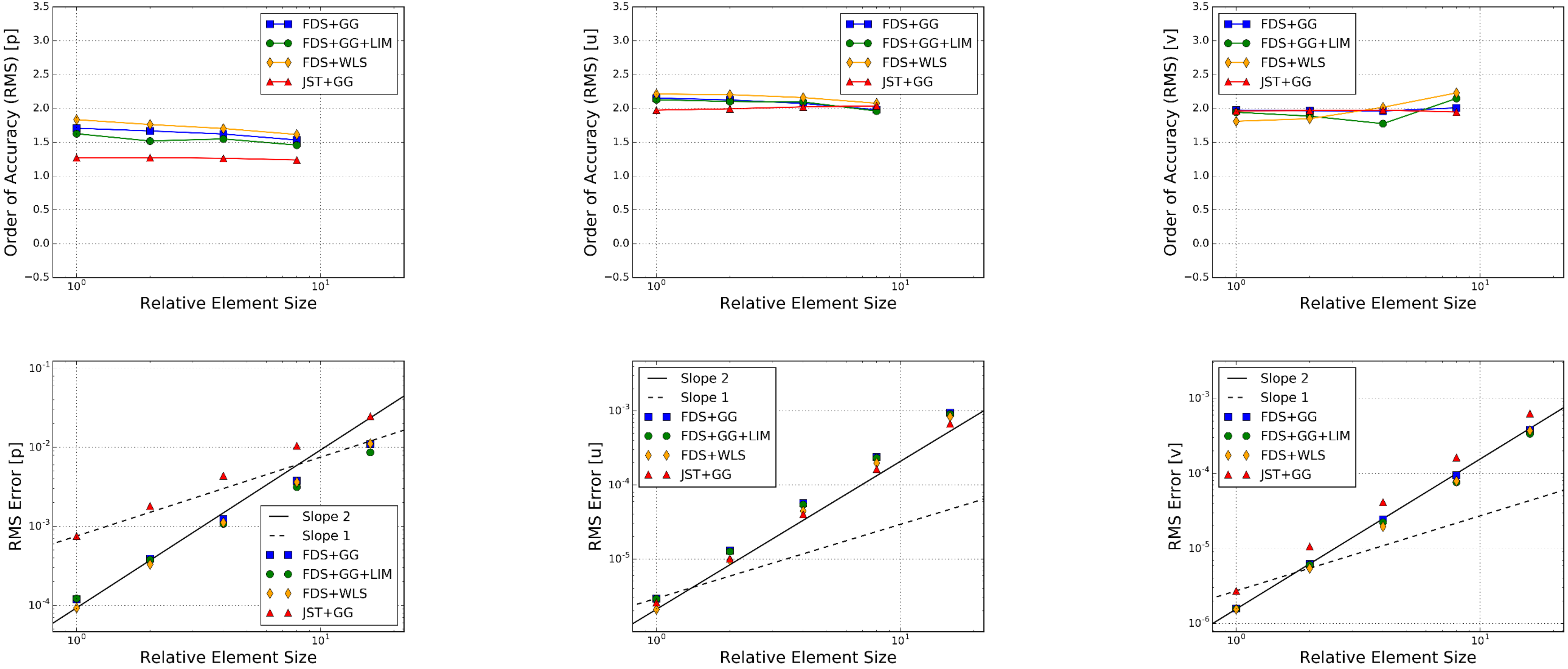
Triangular NxN grids (9,17,33,65,129) with 1st and 2nd order schemes -> Observed accuracy asymptotes to expected value for all cases!

FDS = Flux difference splitting (GG grad for MUSCL), JST = Jameson-Schmidt-Turkel-like scheme, L-F = Lax-Friedrichs 1st order

Solution from: Salari K, and Knupp P, "Code verification by the method of manufactured solutions," SAND 2000-1444, Sandia National Laboratories, Albuquerque, NM, 2000

Case publicly available for reproduction: https://github.com/su2code/VandV/tree/master/mms/fvm_incomp_navierstokes

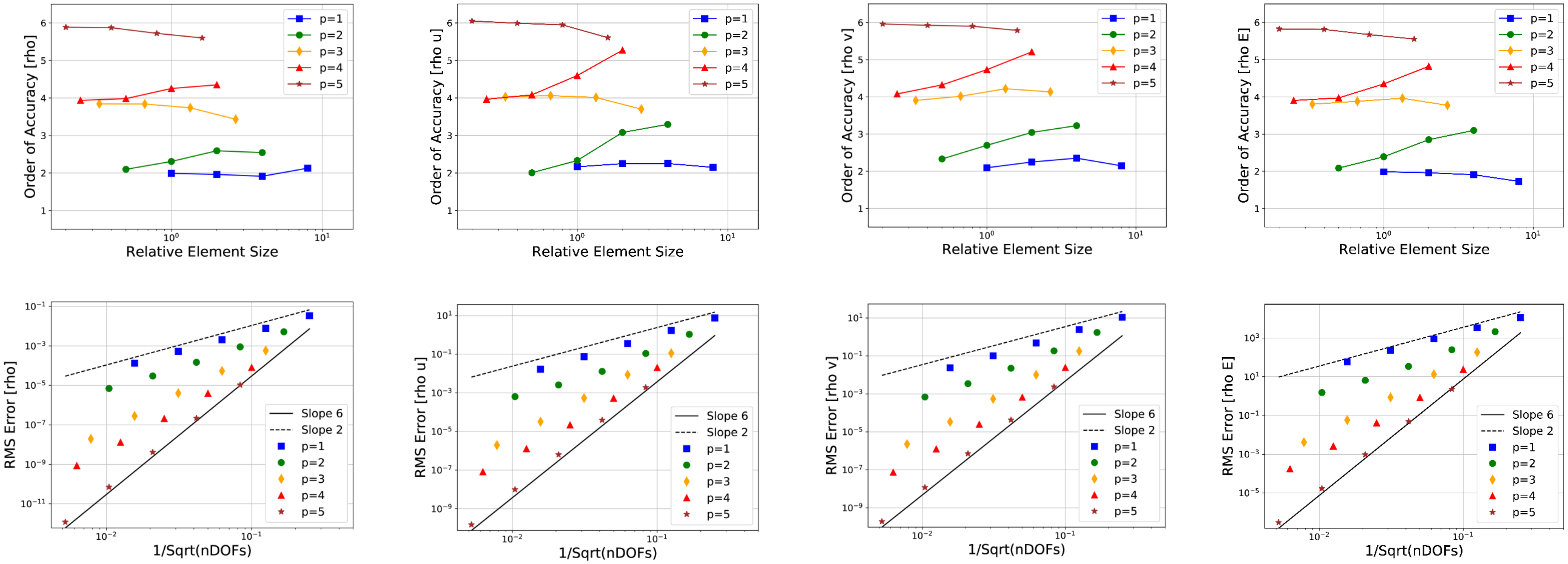
FVM Incompressible Navier-Stokes MMS Results



Triangular NxN grids (9,17,33,65,129) with 2nd order schemes -> Observed accuracy asymptotes to expected value of 2 for all cases (except JST pressure equation).
FDS = Flux difference splitting MUSCL, JST = Jameson-Schmidt-Turkel-like scheme, GG = Green-Gauss, LIM = Venkatakrishnan limiter, WLS = Weighted Least-Squares
Solution from: Salari K, and Knupp P, "Code verification by the method of manufactured solutions," SAND 2000-1444, Sandia National Laboratories, Albuquerque, NM, 2000

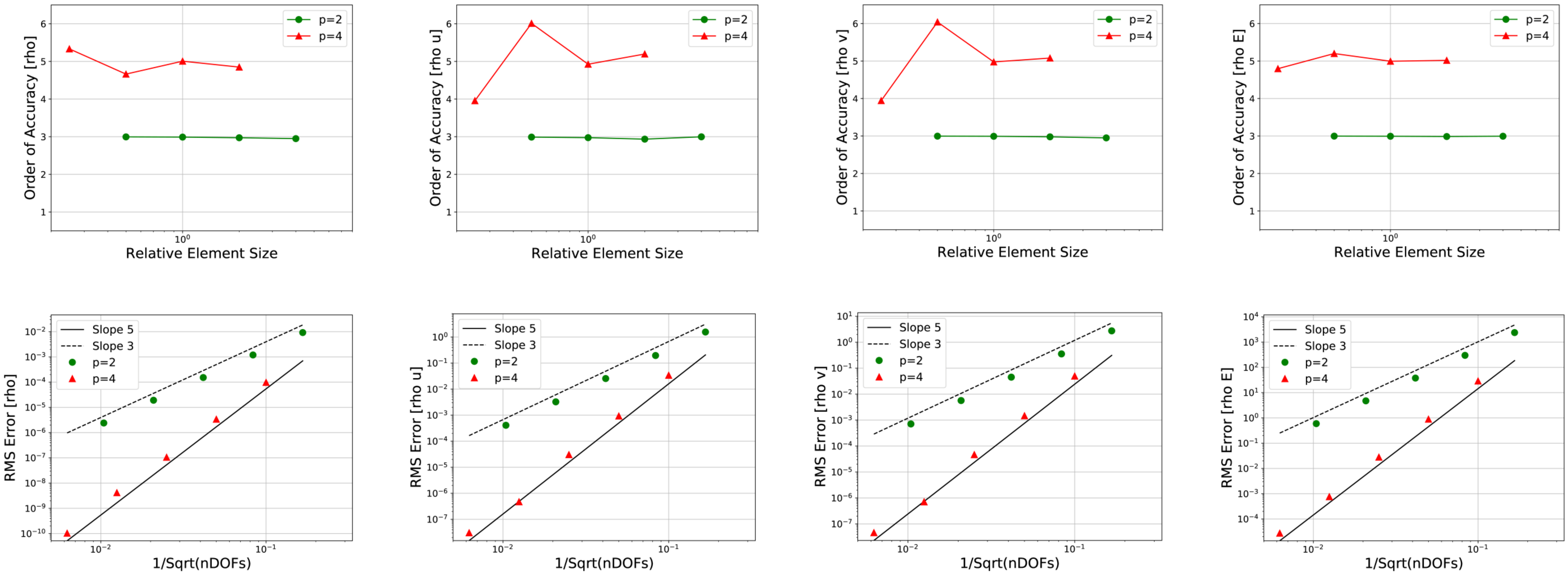
Case publicly available for reproduction: https://github.com/su2code/VandV/tree/master/mms/dg_navierstokes/unit_quad

DG Compressible Navier-Stokes MMS Results



Quadrilateral NxN grids (2,4,8,16,32 elements) with polynomial degree p = 1 …5 -> Observed accuracy asymptotes to expected values for odd degree polynomials. Even degree polynomials show accuracies one order less than expected.

DG Compressible Euler MMS Results



Quadrilateral NxN grids (2,4,8,16,32 elements) with polynomial degree p = 2 and 4 -> p = 2: design accuracy; p = 4: erratic around design accuracy.

Conclusions

- Rigorous approach to verification of the discretization schemes in SU2
- Main tool: Method of Manufactured Solutions
- Polymorphism is used to minimize the modifications in the solver classes
- 11 cases + one user defined class are currently implemented
- Most schemes show design accuracy, but there are some questionable ones (even degree polynomials for the DG solver for Navier-Stokes)
- RANS and (relative) motion cases must (and will) be added
- Code and cases are being made available to the public to demonstrate accuracy of SU2 and support open science/reproducibility. Build on it!