PECOS

Predictive Engineering and Computational Sciences

Manufactured Solutions for the Favre-Averaged Navier-Stokes Equations with Eddy-Viscosity Turbulence Models

Todd A. Oliver, Kemelli C. Estacio-Hiroms, Nicholas Malaya,
Graham F. Carey

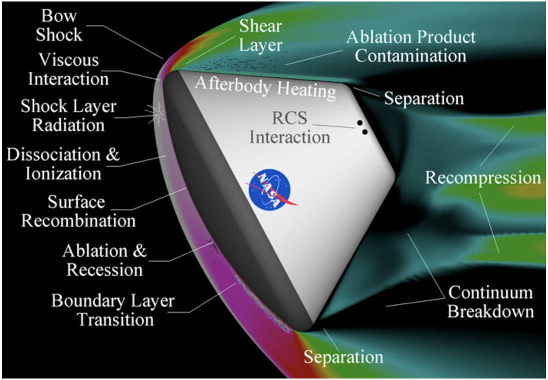
Institute for Computation Engineering and Sciences
The University of Texas at Austin

January 9th, 2012

Outline

1. Background: Verification and MMS
2. A new Spalart-Allmaras Manufactured Solution
3. MASA
4. Conclusions

Physics Problem



Atmospheric Entry

- Multiphysics submodels: Flow, Aerothermochemistry, Ablation, Surface chemistry, Radiation, Turbulence

Software Quality

Complex Codebase

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

General Problem in Scientific Software

- Computer models are being used to inform decision makers
  - Failures are costly ($, lives)
- How do you build confidence in the predictions from a codebase?
  - What constitutes a strong test?

Verification

Verification of Scientific Software

- Verification ensures that the outputs of a computation accurately reflect the solution of the mathematical models.
- Essentially, we are testing if we have correctly instantiated mathematical equations in our code.

Method of Exact Solutions

- Numerically solve a case where the solution is known.

Method of Manufactured Solutions

- Often, analytical solutions either:
    - Do not exist
    - Does not fully exercise equations (e.g. symmetric solution, nonlinearities)
- Alleviate this using MMS: “create” our own solutions

Generating MMS using Symbolic Packages

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, Numpy (output is machine-generated code)

“Manufactured” Code

```
 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;
```

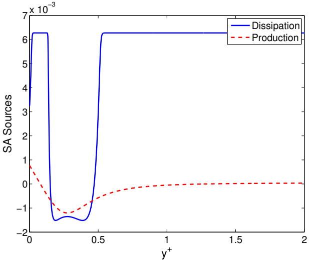
Desired Features of an SA MS

Physically-based MS

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

Shortcoming of other SA MS

- Bond solution: sinusoidal, only satisfies no-slip.
- Eça solutions are shown to have instabilities or near-wall features that disrupt the correct rate of convergence



Governing Equations

FANS + SA

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

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

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

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

Calorically perfect gas with *constant* viscosity:

$$\bar{p} = \bar{\rho}R\tilde{T}, \quad \tilde{e} = c_v\tilde{T}, \quad \tilde{h} = c_p\tilde{T},$$

Modifications

In the original formulation, S_{sa} is given by,

$$S_{\mathrm{sa}} = \Omega + \frac{\nu_{\mathrm{sa}}}{\kappa^2 d^2} f_{v2}$$

This definition is modified to:

$$S_{m0} = \frac{\nu_{\mathrm{sa}}}{\kappa^2 d^2} f_{v2}$$

S_{sa} is given by,

$$S_{\mathrm{sa}} = \Omega + S_m$$

where,

$$S_m = \begin{cases} S_{m0}, & S_{m0} \geq -c_{v2}\Omega \\ \dfrac{\Omega(c_{v2}^2\Omega + c_{v3}S_{m0})}{((c_{v3} - 2c_{v2})\Omega - S_{m0})}, & \text{otherwise.} \end{cases}$$

Our SA Manufactured Solution

- Parameters colored in red (values appear in paper appendix)
- Using 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,

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

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

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

Wall-normal velocity and SA State

- From an order of magnitude analysis of the continuity equation, the mean wall-normal velocity is set to,

$$\tilde{v} = -\eta_v \frac{du_\tau}{dx} y$$

- SA model designed such that $\nu_{\mathrm{sa}} = \kappa u_\tau y$ in the inner region of the boundary layer. Specifically, the SA state variable is given by,

$$\nu_{\mathrm{sa}} = \kappa u_\tau y - \alpha y^2,$$

Thermodynamic State

Specifying Temperature, Pressure and Density

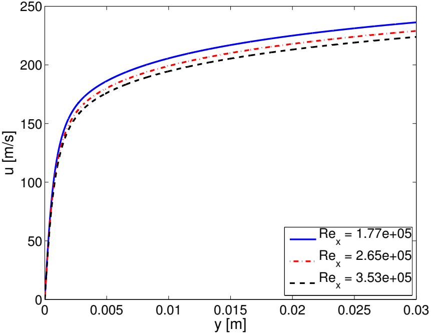
- Mean temperature uses White's(91) temperature-velocity relation:

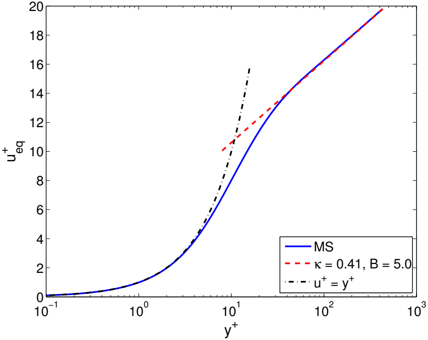
$$\tilde{T} = T_\infty \left[1 + r_T \frac{\gamma - 1}{2} {M_\infty}^2 \left(1 - \left(\frac{\tilde{u}}{u_\infty} \right)^2 \right) \right]$$

- Pressure is assumed to be a constant, p_0
- Density is computed from the ideal gas equation:

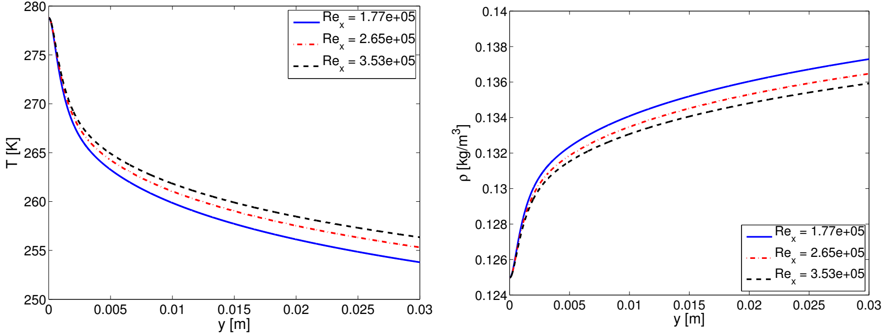
$$\bar{\rho} = \frac{p_0}{R\tilde{T}}$$

Manufactured Velocity Profiles



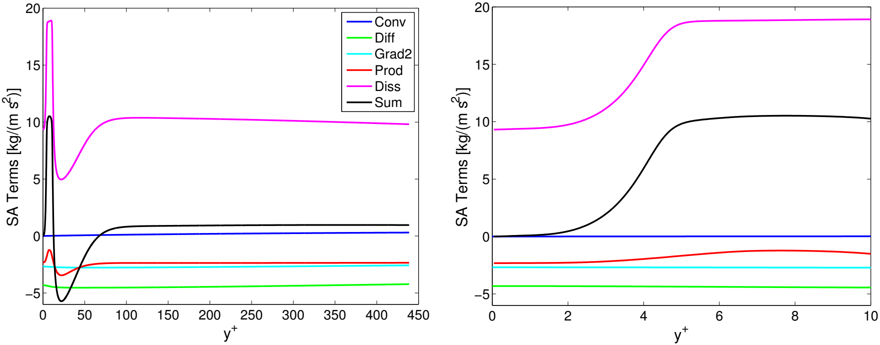


Manufactured Temperature and Density Profiles



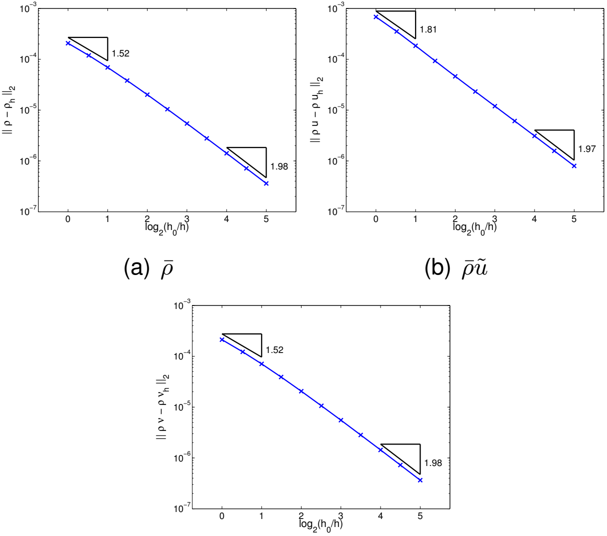
- T and ρ designed to provide moderate variation

SA Equation Budgets



- Inside viscous sublayer, source term is small relative to other terms
- Production and dissipation terms go to constants at the wall
- Source term is largest in buffer region

Convergence Rates: Low Re_x

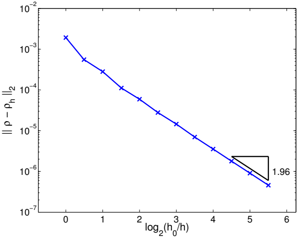


(a) $\bar{\rho}$

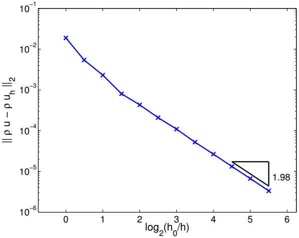
(b) $\bar{\rho}\tilde{u}$

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

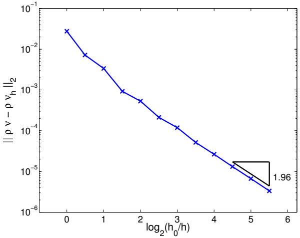
Convergence Rates: $Re_x = 3.5 * 10^5$



(d) $\bar{\rho}$



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



(f) $\bar{\rho}\nu_{\rm sa}$

Manufactured Analytical Solutions Abstractions Library

Goal: Provide a repository and standardized interface for MMS usage

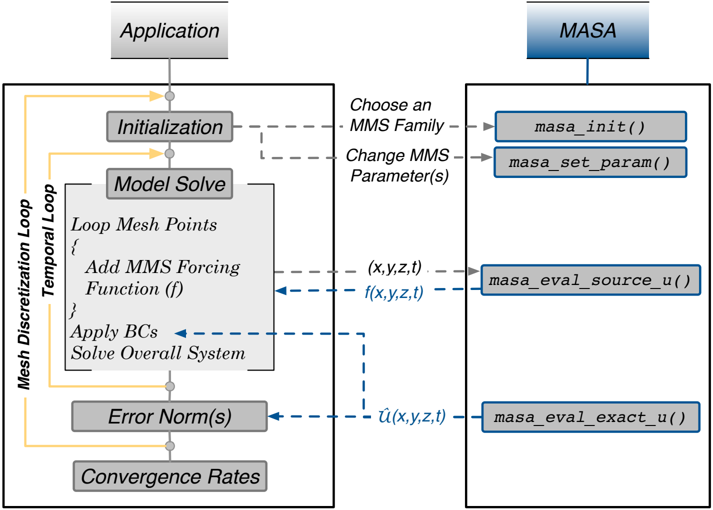
High Priority:

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

Low Priority:

- Speed/Performance

General Verification Approach Using MMS and MASA



Verifying the “Verifier”

Precision is not negotiable: users must trust MASA output!

MASA Testing

- Error target $<$ 1e-15
  - Relative error
  - 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 62 tests passed
===================
```

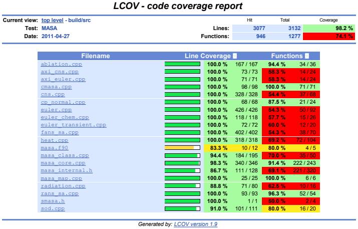
Portability

Software Environment

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

Testing

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



LCOV - code coverage report

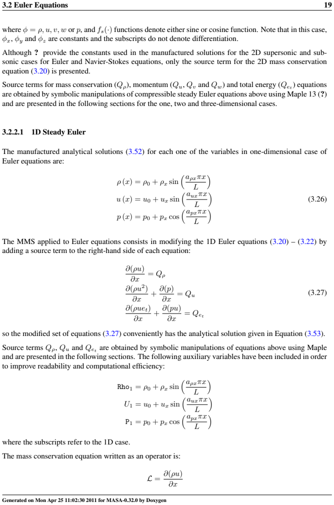
			Hit	Total	Coverage
Current view:	top level - build/src				
Test:	MASA	Lines:	3077	3132	98.2 %
Date:	2011-04-27	Functions:	946	1277	74.1 %

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



3.2 Euler Equations **19**

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 subsonic 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:

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

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:

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

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:

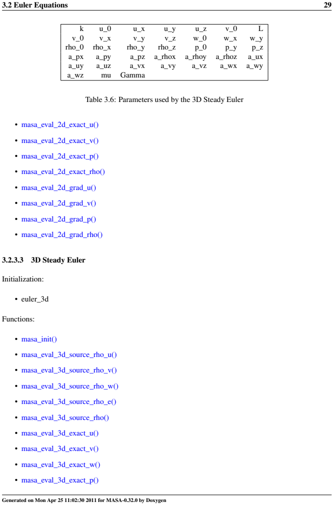
$$\begin{aligned} \mathtt{Rho}_1 &= \rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right) \\ U_1 &= u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) \\ \mathtt{P}_1 &= p_0 + p_x \cos\left(\frac{a_{px}\pi x}{L}\right) \end{aligned}$$

where the subscripts refer to the 1D case.

The mass conservation equation written as an operator is:

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

Generated on Mon Apr 25 11:02:30 2011 for MASA-0.32.0 by Doxygen



3.2 Euler Equations **29**

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()

Generated on Mon Apr 25 11:02:30 2011 for MASA-0.32.0 by Doxygen

Available Solutions in MASA 0.40

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

Future Solution Development

Single Physics

- Additional RANS models (k-ω, k-ϵ, etc.)
- Shocks

Multiphysics

- Turbulence with chemistry
- Flow with improved transport

Importing New Solutions in MASA 0.40

- Model document detailing analytical solution and source terms
- Latex documents can be loaded directly into MASA documentation
- Source and analytical terms in C/C++/Fortran90
- Willingness to share

Conclusions

New SA Model

- Generated a new, physically-based, MS
- Great care goes into constructing MS

MASA

- Open-source, extensible repository for MS and software verification

Getting the word out

- AIAA journal paper (in preparation)
- Engineering with Computers (submitted)
- Download MASA 0.40 at:
  `https://red.ices.utexas.edu/projects/`

Thank you!

Questions?

nick@ices.utexas.edu