Aerothermodynamic Design Sensitivities for a Reacting Gas Flow Solver on an Unstructured Mesh Using a Discrete Adjoint Formulation

Kyle B. Thompson

Mechanical and Aerospace Engineering Department
North Carolina State University

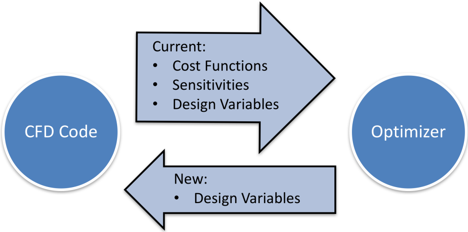
Aerothermodynamics Branch
NASA Langley Research Center

Month Day, 2017

Outline

Introduction - Design

- Gradient-based design optimization is based on the minimization of a target “cost” function by changing a set of design variables
- A CFD code can be coupled with a numerical optimization package to iteratively improve target aerothermodynamic quantities, by change inputs to the CFD code



CFD-Optimizer Relationship

Introduction - Design

- The top-level design process is simple, but CFD sensitivity analysis is expensive

Introduction - Design

- The top-level design process is simple, but CFD sensitivity analysis is expensive
- Need efficient way to compute cost function sensitivities for large number of design variables

Introduction - Design

- The top-level design process is simple, but CFD sensitivity analysis is expensive
- Need efficient way to compute cost function sensitivities for large number of design variables

Direct differentiation approach - Expensive

- Navier-Stokes equations can be directly differentiated to yield sensitivity derivatives necessary for gradient-based optimization
- Finite difference requires a minimum of **one flow solution for each design variable sensitivity**
- Prohibitively expensive for large number of design variables

Introduction - Design

- The top-level design process is simple, but CFD sensitivity analysis is expensive
- Need efficient way to compute cost function sensitivities for large number of design variables

Adjoint approach - More efficient

- Solve adjoint equations in addition to Navier Stokes flow equations to obtain sensitivity derivatives
- **One flow and adjoint solution needed for each cost function**, regardless of number of design variables
- Considerably more efficient than direct differentiation approach for large number of design variables

Introduction - Design

- Adjoint-based design optimization is widely adopted in compressible, perfect gas CFD solvers
- Reacting flow solvers have lagged in adopting adjoint-based approach, due to
  1. Complexity of linearizing the additional equations for multi-species chemical kinetics
  2. Resorting to Automatic Differentiation tools incurs performance overhead that is implementation-specific
  3. Serious memory and computational cost concerns when simulating a large number of species
- Points 1 and 2 can be overcome through stubbornness (or hiring a graduate student. . .)
- Point 3 is a serious concern, if reacting flow solver are to be made attractive for design optimization

Introduction - Improvement to State of the Art

- Current state of the art
  - Attempts made at both continuous[1] and discrete[2] adjoint formulations for a compressible reacting flow solver
  - These attempts suffer from quadratic scaling in memory and computational cost with number of species
  - Recent scheme at Barcelona Supercomputing Center[3] is promising, but only for incompressible reacting flows
- Improvement to the state of the art
  - New decoupled scheme for both hypersonic flow solver and adjoint solver that is robust for high-speed flows in chemical non-equilibrium
  - New schemes significantly improve scaling in computational cost and memory with number of species

[1] **Copeland**.

[2] **Lockwood**.

[3] **Esfahani:2016aa**.

Introduction - Decoupled Approach

- Reacting gas simulations require solving a large number of conservation equations
- Memory concerns
  - Size of Jacobians scales quadratically with number species in gas mixture
  - Solving system of equations in a tightly-coupled fashion can be limited by memory constraints
- Cost concerns
  - Cost of solving the linear system scales quadratically with number of species in gas mixture
- Efficiently solving adjoint problem is a primary motivator
  - Solving adjoint system particularly costly if linear solver is slow
  - Can be necessary to store jacobian twice $\rightarrow$ large memory overhead

Introduction - Decoupled Approach

- Loosely-coupled solvers have become popular in the combustion community[4]
  - Decouple species conservation equations from meanflow equations, and solve two smaller systems

$$\underset{(4+ns)\times(4+ns)}{\begin{pmatrix} \square & \square & \cdots & \square \\ \square & \square & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \square & \cdots & \cdots & \square \end{pmatrix}} \rightarrow \underset{5\times 5}{\begin{pmatrix} \square & \cdots & \square \\ \vdots & \ddots & \vdots \\ \square & \cdots & \square \end{pmatrix}} \text{ and } \underset{ns\times ns}{\begin{pmatrix} \square & \boxbslash & \cdots & \boxbslash \\ \boxbslash & \square & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \boxbslash & \cdots & \cdots & \square \end{pmatrix}}$$

- Candler, et al.[5] originally derived this for Steger-Warming scheme, this work extends to Roe FDS scheme

[4] **Sankaran**.

[5] **candler**.

Outline

Fully-Coupled Point Implicit Flow Solver

- All work presented is for inviscid flows in chemical non-equilibrium, using a one-temperature model, but is extendable to viscous flows.
- Beginning with the semi-discrete form

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{1}{\Omega} \sum_f (\mathbf{F} \cdot \mathbf{N})^f = \mathbf{W}$$

$$\mathbf{U} = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{N_s} \\ \rho \mathbf{u} \\ \rho E \end{pmatrix}, \quad \mathbf{F} \cdot \mathbf{N} = \begin{pmatrix} \rho_1 \overline{U} \\ \vdots \\ \rho_{N_s} \overline{U} \\ \rho \mathbf{u} \overline{U} + p\mathbf{n} \\ (\rho E + p) \overline{U} \end{pmatrix} N, \quad \mathbf{W} = \begin{pmatrix} \dot{\rho}_1 \\ \vdots \\ \dot{\rho}_{N_s} \\ 0 \\ 0 \end{pmatrix}$$

Fully-Coupled Point Implicit Flow Solver

- Using the Roe FDS scheme to compute the inviscid flux at the face, $\mathbf{F}^f$, and linearizing the system results in an implicit system

$$\left(\frac{\Omega}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{U}}}{\partial \mathbf{U}}\right)\Delta\mathbf{U} = \mathbf{R}_{\mathbf{U}}$$

Fully-Coupled Point Implicit Flow Solver

- Using the Roe FDS scheme to compute the inviscid flux at the face, $\mathbf{F}^f$, and linearizing the system results in an implicit system

$$\left(\frac{\Omega}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{U}}}{\partial \mathbf{U}}\right)\Delta\mathbf{U} = \mathbf{R}_{\mathbf{U}}$$

- With global system composed of block matrices

$$\underbrace{\left[\frac{\Omega}{\Delta t}\mathbf{I} + \begin{pmatrix} \frac{\partial \mathbf{R}_{\rho_1}}{\partial \rho_1} & \cdots & \frac{\partial \mathbf{R}_{\rho_1}}{\partial \rho_{N_s}} & \frac{\partial \mathbf{R}_{\rho_1}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho_1}}{\partial \rho E} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{\partial \mathbf{R}_{\rho_{N_s}}}{\partial \rho_1} & \cdots & \frac{\partial \mathbf{R}_{\rho_{N_s}}}{\partial \rho_{N_s}} & \frac{\partial \mathbf{R}_{\rho_{N_s}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho_{N_s}}}{\partial \rho E} \\ \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E} \\ \frac{\partial \mathbf{R}_{\rho E}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho E}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E} \end{pmatrix}\right]}_{(N_s+4)\times(N_s+4)} \underbrace{\begin{pmatrix} \Delta\mathbf{U}_{\rho_1} \\ \vdots \\ \Delta\mathbf{U}_{\rho_{N_s}} \\ \Delta\mathbf{U}_{\rho \mathbf{u}} \\ \Delta\mathbf{U}_{\rho E} \end{pmatrix}}_{(N_s+4)\times 1} = \underbrace{\begin{pmatrix} \mathbf{R}_{\rho_1} \\ \vdots \\ \mathbf{R}_{\rho_{N_s}} \\ \mathbf{R}_{\rho \mathbf{u}} \\ \mathbf{R}_{\rho E} \end{pmatrix}}_{(N_s+4)\times 1}$$

Fully-Coupled Point Implicit Flow Solver

- Constructing the Jacobian in a fully-coupled fashion results in large, dense block matricies
- Cost of multi-color Gauss-Seidel sweeps dominated by matrix-vector products

$$\text{Cost} \rightarrow O((N_s + 4)^2)$$

- Leads to onerous quadratic scaling with respect to number of species

Decoupled Point Implicit Flow Solver

- The main idea is to separate the meanflow and species composition equations, adding a new equation for the total mixture density
- Leads to two sets of conserved variables

$$\mathbf{U}' = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix} \qquad \hat{\mathbf{U}} = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{ns} \end{pmatrix}$$

Meanflow Species Composition

Decoupled Point Implicit Flow Solver

- The fluxes are solved in two sequential steps
  - The mixture fluxes are first solved as

$$\frac{\partial \mathbf{U}'}{\partial t} + \frac{1}{\Omega}\sum_f (\mathbf{F}' \cdot \mathbf{N})^f = 0$$

  - Followed by the species fluxes

$$\frac{\partial \hat{\mathbf{U}}}{\partial t} + \frac{1}{V}\sum_f (\hat{\mathbf{F}} \cdot \mathbf{N})^f = \hat{\mathbf{W}}$$

- Since the mixture density was determined in the first step, step two actually solves for the species mass fractions

$$\delta \hat{\mathbf{U}}^n = \rho^{n+1}\hat{\mathbf{V}}^{n+1} - \rho^n \hat{\mathbf{V}}^n = \rho^{n+1}\delta\hat{\mathbf{V}}^n + \hat{\mathbf{V}}^n \delta\rho^n$$
$$\hat{\mathbf{V}} = (c_1, \ldots, c_{ns})^T, c_s = \rho_s/\rho$$

Decoupled Point Implicit Flow Solver

- The Roe FDS scheme species mass fluxes can be rewritten as

$$\hat{\mathbf{F}}_{\rho_s} = c_s \mathbf{F}'_\rho + (c_s^L - \tilde{c}_s)\rho^L\lambda^+ + (c_s^R - \tilde{c}_s)\rho^R\lambda^-$$

$$\frac{\partial \hat{\mathbf{F}}_{\rho_s}}{\partial c_s^L} = \tilde{w}\mathbf{F}_\rho + (1-\tilde{w})\rho^L\lambda^+ - \tilde{w}\rho^R\lambda^-$$

$$\frac{\partial \hat{\mathbf{F}}_{\rho_s}}{\partial c_s^R} = (1-w)\mathbf{F}_\rho + (\tilde{w}-1)\rho^L\lambda^+ + \tilde{w}\rho^R\lambda^-$$

- Jacobian Approximations

$$\text{Step 1:} \quad \left.\frac{\partial \mathbf{F}}{\partial \mathbf{U}'}\right|_{\hat{\mathbf{V}}} = \underset{c_s = \text{Constant}}{5 \times 5 \,\text{Roe FDS Jacobian}}$$

$$\text{Step 2:} \quad \left.\frac{\partial \mathbf{F}}{\partial \hat{\mathbf{V}}}\right|_{\hat{\mathbf{U}}'} = \begin{pmatrix} \frac{\partial F_{\rho 1}}{\partial c_1} & & 0 \\ & \ddots & \\ 0 & & \frac{\partial F_{\rho ns}}{\partial c_{ns}} \end{pmatrix}$$

Decoupled Point Implicit Flow Solver

- Chemical source term linearized via

$$\hat{\mathbf{W}}^{n+1} = \hat{\mathbf{W}}^{n} + \left.\frac{\partial \hat{\mathbf{W}}}{\partial \mathbf{U}}\right|_{\mathbf{U}'} \frac{\partial \mathbf{U}}{\partial \hat{\mathbf{V}}}$$

$$\mathbf{C} = \left.\frac{\partial \hat{\mathbf{W}}}{\partial \mathbf{U}}\right|_{\mathbf{U}'} \frac{\partial \mathbf{U}}{\partial \hat{\mathbf{V}}}$$

- Full system to be solved in step two

$$\frac{\Omega}{\delta t}\rho^{n+1}\delta\hat{\mathbf{V}} + \sum_f \left(\frac{\partial \hat{\mathbf{F}}^f}{\partial \hat{\mathbf{V}}^L} \cdot \mathbf{N}^f \delta\hat{\mathbf{V}}^L + \frac{\partial \hat{\mathbf{F}}^f}{\partial \hat{\mathbf{V}}^R} \cdot \mathbf{N}^f \delta\hat{\mathbf{V}}^R \right)^{n,n+1} - \Omega C^{n,n+1}\delta\hat{\mathbf{V}}^n$$

$$= -\sum_f \left(\hat{\mathbf{F}}^{n,n+1} \cdot \mathbf{N} \right)^f + (\Omega)(\omega_r)(\mathbf{W}^{n,n+1}) + R_\rho \hat{\mathbf{V}}^n$$

$$R_\rho = \sum_f \sum_{s=1}^{N_s} (\hat{F}_{\rho_s}^{n,n+1} \cdot \mathbf{N})$$

- R_ρ is included to preserve $\sum_s c_s = 1$, $\sum_s \delta c_s = 0$.

Decoupled Point Implicit Flow Solver

Mixture Equations :

$$\left[\frac{\Omega}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{U}'}}{\partial \mathbf{U}'}\right] \Delta \mathbf{U}' = \mathbf{R}_{\mathbf{U}'}$$

$$\left[\frac{\Omega}{\Delta t}\mathbf{I} + \begin{pmatrix} \frac{\partial \mathbf{R}_\rho}{\partial \rho} & \frac{\partial \mathbf{R}_\rho}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_\rho}{\partial \rho E} \\ \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E} \\ \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E} \end{pmatrix}\right] \begin{pmatrix} \Delta \rho \\ \Delta \rho \mathbf{u} \\ \Delta \rho E \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N_s} (\mathbf{R}_{\rho_i}) \\ \mathbf{R}_{\rho \mathbf{u}} \\ \mathbf{R}_{\rho E} \end{pmatrix}$$

Species Continuity Equations :

$$\left[\frac{\Omega}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\hat{\mathbf{V}}}}{\partial \hat{\mathbf{V}}}\right] \Delta \hat{\mathbf{V}} = \mathbf{R}_{\hat{\mathbf{V}}}$$

$$\left[\frac{\rho V}{\Delta t}\mathbf{I} + \begin{pmatrix} \frac{\partial \mathbf{R}_{\rho_1}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho_1}}{\partial c_{ns}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{R}_{\rho_{ns}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho_{ns}}}{\partial c_{ns}} \end{pmatrix}\right] \begin{pmatrix} \Delta c_1 \\ \vdots \\ \Delta c_{ns} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{\rho_1} - c_1 \sum_{i=1}^{N_s} (\mathbf{R}_{\rho_i}) \\ \vdots \\ \mathbf{R}_{\rho_{N_s}} - c_{N_s} \sum_{i=1}^{N_s} (\mathbf{R}_{\rho_i}) \end{pmatrix}$$

Cost and Memory Savings of the Decoupled Flow Solver

- Most significant savings comes from the source term linearization being purely node-based
  - Convective contributions to block Jacobians are diagonal
  - Source term jacobian is dense block Jacobian
  - In the global system (w/chemistry), all off-diagonal block jacobians are diagonal

$$\begin{pmatrix} \square & & & & \\ & \ddots & & & \\ & & \square & & \\ & & & \ddots & \\ & & & & \square \end{pmatrix} \begin{pmatrix} \delta\hat{\mathbf{V}}_1 \\ \vdots \\ \delta\hat{\mathbf{V}}_i \\ \vdots \\ \delta\hat{\mathbf{V}}_{nodes} \end{pmatrix} = \begin{pmatrix} \hat{b}_1 \\ \vdots \\ \hat{b}_i \\ \vdots \\ \hat{b}_{nodes} \end{pmatrix} - \begin{pmatrix} (\sum_{j=1}^{N_{nb}} [\diagdown]\delta\hat{\mathbf{V}}_j)_1 \\ \vdots \\ (\sum_{j=1}^{N_{nb}} [\diagdown]\delta\hat{\mathbf{V}}_j)_i \\ \vdots \\ (\sum_{j=1}^{N_{nb}} [\diagdown]\delta\hat{\mathbf{V}}_j)_{nodes} \end{pmatrix}$$

- Matrix-vector products $\rightarrow$ inner products: $O(ns^2) \rightarrow O(ns)$

Cost and Memory Savings of the Decoupled Flow Solver

- Comparing size of Jacobian systems, using Compressed Row Storage

$$\mathbf{A}_d = \text{Decoupled system Jacobians}$$
$$\mathbf{A} = \text{Fully-coupled system Jacobians}$$

$$\begin{aligned} Relative\ Memory\ Cost &= \frac{size(\mathbf{A}_d)}{size(\mathbf{A})} \\ &= \lim_{ns\to\infty} \frac{(ns^2+5^2)(N_{nodes}) + (ns+5^2)(N_{nbrs})}{(ns+4)^2(N_{nodes}+N_{nbrs})} \\ &= \frac{N_{nodes}}{N_{nodes}+N_{nbrs}} \end{aligned}$$

Outline

Derivation of Discrete Adjoint Formulation

- The derivation of the adjoint approach to compute design sensitivities begins with forming the Lagrangian and differentiating with respect to the design variables

$$L(\mathbf{D}, \mathbf{Q}, \mathbf{X}, \boldsymbol{\Lambda}) = f(\mathbf{D}, \mathbf{Q}, \mathbf{X}) + \boldsymbol{\Lambda}^T \mathbf{R}(\mathbf{D}, \mathbf{Q}, \mathbf{X})$$

$$\begin{aligned} \mathbf{D} &= \text{design variables} & f &= \text{cost function} \\ \mathbf{Q} &= \text{flow variables} & \mathbf{R} &= \text{flow residual} \end{aligned}$$

$$\Lambda = \text{costate variables}$$

Derivation of Discrete Adjoint Formulation

- The derivation of the adjoint approach to compute design sensitivities begins with forming the Lagrangian and differentiating with respect to the design variables

$$L(\mathbf{D},\mathbf{Q},\mathbf{X},\boldsymbol{\Lambda}) = f(\mathbf{D},\mathbf{Q},\mathbf{X}) + \boldsymbol{\Lambda}^T\mathbf{R}(\mathbf{D},\mathbf{Q},\mathbf{X})$$

$$\frac{\partial L}{\partial \mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \left[\frac{\partial \mathbf{Q}}{\partial \mathbf{D}}\right]^T \left\{\frac{\partial f}{\partial \mathbf{Q}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^T \Lambda\right\} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{D}}\right]^T \Lambda$$

$\mathbf{D}$ = design variables $\quad$ f = cost function

$\mathbf{Q}$ = flow variables $\quad$ $\mathbf{R}$ = flow residual

Λ = costate variables

Derivation of Discrete Adjoint Formulation

- The derivation of the adjoint approach to compute design sensitivities begins with forming the Lagrangian and differentiating with respect to the design variables

$$L(\mathbf{D},\mathbf{Q},\mathbf{X},\boldsymbol{\Lambda}) = f(\mathbf{D},\mathbf{Q},\mathbf{X}) + \boldsymbol{\Lambda}^T \mathbf{R}(\mathbf{D},\mathbf{Q},\mathbf{X})$$

$$\frac{\partial L}{\partial \mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \boxed{\left[\frac{\partial \mathbf{Q}}{\partial \mathbf{D}}\right]^T \left\{ \frac{\partial f}{\partial \mathbf{Q}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^T \Lambda \right\}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{D}}\right]^T \Lambda$$

$\mathbf{D}$ = design variables f = cost function

$\mathbf{Q}$ = flow variables $\mathbf{R}$ = flow residual

Λ = costate variables

Derivation of Discrete Adjoint Formulation

- Need to eliminate flow variable dependence on design variables, $\frac{\partial \mathbf{Q}}{\partial \mathbf{D}}$
- Adjoint equation

$$\left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]^T \Lambda = -\frac{\partial f}{\partial \mathbf{Q}}$$

- Solve for Λ and compute sensitivity derivatives

$$\frac{\partial L}{\partial \mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \left[\frac{\partial \mathbf{R}}{\partial \mathbf{D}}\right]^T \Lambda$$

Fully Coupled Adjoint Solver

- Adjoint problem is a linear system

$$\begin{pmatrix} \frac{\partial \mathbf{R}_{\rho_i}}{\partial \rho_j}^T & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho_j}^T & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho_j}^T \\ \frac{\partial \mathbf{R}_{\rho_i}}{\partial \rho \mathbf{u}}^T & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}}^T & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}}^T \\ \frac{\partial \mathbf{R}_{\rho_i}}{\partial \rho E}^T & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E}^T & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E}^T \end{pmatrix} \begin{pmatrix} \Lambda_{\rho_i} \\ \Lambda_{\rho \mathbf{u}} \\ \Lambda_{\rho E} \end{pmatrix} = - \begin{pmatrix} \frac{\partial f}{\partial \rho_i} \\ \frac{\partial f}{\partial \rho \mathbf{u}} \\ \frac{\partial f}{\partial \rho E} \end{pmatrix}$$

- Can be solved with Krylov method (i.e. GMRES), but time marching similar to flow solver shown to be more robust

$$\left(\frac{V}{\Delta t} \mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{U}}}{\partial \mathbf{U}} \right) \Delta \Lambda = - \left(\frac{\partial \mathbf{R}_{\mathbf{U}}}{\partial \mathbf{U}}^T \Lambda_{\mathbf{U}}^n + \frac{\partial f}{\partial \mathbf{U}} \right)$$

- Straightforward to formulate, but cost and memory requirements scale quadratically with number of species

Decoupled Adjoint Scheme

- The decoupled flow solver has an analog in the adjoint
- First, recognize that the decoupled flow solver can be rewritten as a fully coupled system, with a change of variables and change of equations

$$\underbrace{\mathbf{U} = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{ns} \\ \rho\mathbf{u} \\ \rho E \end{pmatrix} \rightarrow \mathbf{V} = \begin{pmatrix} c_1 \\ \vdots \\ c_{ns} \\ \rho \\ \rho\mathbf{u} \\ \rho E \end{pmatrix}}_{\text{Change of Variables}}, \quad \underbrace{\mathbf{R}_{\mathbf{U}} = \begin{pmatrix} \mathbf{R}_{\rho_1} \\ \vdots \\ \mathbf{R}_{\rho_{N_s}} \\ \mathbf{R}_{\rho\mathbf{u}} \\ \mathbf{R}_{\rho E} \end{pmatrix} \rightarrow \mathbf{R}_{\mathbf{V}} = \begin{pmatrix} \mathbf{R}_{\rho_1} - c_1 \sum_{i=1}^{N_s} \left(\mathbf{R}_{\rho_i}\right) \\ \vdots \\ \mathbf{R}_{\rho_{N_s}} - c_{N_s} \sum_{i=1}^{N_s} \left(\mathbf{R}_{\rho_i}\right) \\ \sum_{i=1}^{N_s} \left(\mathbf{R}_{\rho_i}\right) \\ \mathbf{R}_{\rho\mathbf{u}} \\ \mathbf{R}_{\rho E} \end{pmatrix}}_{\text{Change of Equations}}$$

$$c_s = \frac{\rho_s}{\rho}, \quad \rho = \sum_{i=1}^{N_s} \left(\rho_i\right)$$

Decoupled Adjoint Scheme

- This change of variables/equations results in non-square transformation matricies

$$\frac{\partial \mathbf{U}}{\partial \mathbf{V}} = \begin{pmatrix} \rho & \dots & 0 & c_1 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \rho & c_{ns} & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}} = \begin{pmatrix} 1-c_1 & -c_1 & \dots & -c_1 & 0 & 0 \\ -c_2 & 1-c_2 & \dots & -c_2 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -c_{N_s} & -c_{N_s} & \dots & 1-c_{N_s} & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}$$

Decoupled Adjoint Scheme

- Using the transformation matricies, $\frac{\partial \mathbf{U}}{\partial \mathbf{V}}$ and $\frac{\partial \mathbf{R_U}}{\partial \mathbf{R_V}}$, it possible to treat the decoupled approach as a series of matrix operations

$$\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}} = \frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}} \frac{\partial \mathbf{R_U}}{\partial \mathbf{U}} \frac{\partial \mathbf{U}}{\partial \mathbf{V}}$$

Decoupled Adjoint Scheme

- Using the transformation matricies, $\frac{\partial \mathbf{U}}{\partial \mathbf{V}}$ and $\frac{\partial \mathbf{R_U}}{\partial \mathbf{R_V}}$, it possible to treat the decoupled approach as a series of matrix operations

$$\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}} = \frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}} \frac{\partial \mathbf{R_U}}{\partial \mathbf{U}} \frac{\partial \mathbf{U}}{\partial \mathbf{V}}$$

- These matrix operations can be thought of as left and right preconditioning

$$\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}} = \underbrace{\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}}\right)}_{\text{Left Preconditioner}} \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}\right) \underbrace{\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)}_{\text{Right Preconditioner}}$$

Decoupled Adjoint Scheme

- Using the transformation matricies, $\frac{\partial \mathbf{U}}{\partial \mathbf{V}}$ and $\frac{\partial \mathbf{R_U}}{\partial \mathbf{R_V}}$, it possible to treat the decoupled approach as a series of matrix operations

$$\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}} = \frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}} \frac{\partial \mathbf{R_U}}{\partial \mathbf{U}} \frac{\partial \mathbf{U}}{\partial \mathbf{V}}$$

- These matrix operations can be thought of as left and right preconditioning

$$\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}} = \underbrace{\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}}\right)}_{\text{Left Preconditioner}} \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}\right) \underbrace{\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)}_{\text{Right Preconditioner}}$$

- This transformation avoids having to explicitly form the jacobian $\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}}$, which are more complicated than $\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}$

Decoupled Adjoint Scheme

- Rewrite the adjoint system from $\mathbf{R_V}(\mathbf{V})$

$$\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}}\right)^T \Lambda_{\mathbf{V}} = -\frac{\partial f}{\partial \mathbf{V}}$$

Decoupled Adjoint Scheme

- Rewrite the adjoint system from $\mathbf{R_V}(\mathbf{V}) \rightarrow \mathbf{R_U}(\mathbf{U})$

$$\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}}\right)^T \Lambda_{\mathbf{V}} = -\frac{\partial f}{\partial \mathbf{V}}$$

$$\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^T \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}\right)^T \left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}}\right)^T \Lambda_{\mathbf{V}} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^T \left(\frac{\partial f}{\partial \mathbf{U}}\right)$$

Decoupled Adjoint Scheme

- Rewrite the adjoint system from $\mathbf{R_V}(\mathbf{V}) \to \mathbf{R_U}(\mathbf{U})$

$$\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{V}}\right)^T \Lambda_\mathbf{V} = -\frac{\partial f}{\partial \mathbf{V}}$$

$$\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^T \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}\right)^T \left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}}\right)^T \Lambda_\mathbf{V} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^T \left(\frac{\partial f}{\partial \mathbf{U}}\right)$$

- Effectively another Left/Right Preconditioning

$$\underbrace{\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^T}_{\text{Left Preconditioning}} \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}\right)^T \underbrace{\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}}\right)^T \Lambda_\mathbf{V}}_{\text{Right Preconditioning}} = -\underbrace{\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^T}_{\text{Left Preconditioning}} \left(\frac{\partial f}{\partial \mathbf{U}}\right)$$

$$\left(\frac{\partial \mathbf{R_V}}{\partial \mathbf{R_U}}\right)^T \Lambda_\mathbf{V} = \Lambda_\mathbf{U}$$

Decoupled Adjoint Solver

- Time march adjoint solution with approximate jacobians

$$\left(\frac{V}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}^{T}\right)\Lambda_{\mathbf{V}} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^{T} \underbrace{\left(\frac{\partial \mathbf{R}_{\mathbf{U}}}{\partial \mathbf{U}}\Lambda_{\mathbf{U}} + \frac{\partial f}{\partial \mathbf{U}}\right)}_{\text{Fully Coupled Residual}}$$

- Split $\frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}$ in same fashion as flow solver

Decoupled Adjoint Solver

- Time march adjoint solution with approximate jacobians

$$\left(\frac{V}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}^{T}\right)\Lambda_{\mathbf{V}} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^{T}\underbrace{\left(\frac{\partial \mathbf{R}_{\mathbf{U}}}{\partial \mathbf{U}}\Lambda_{\mathbf{U}} + \frac{\partial f}{\partial \mathbf{U}}\right)}_{\text{Fully Coupled Residual}}$$

- Split $\frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}$ in same fashion as flow solver

$$\begin{pmatrix}
\frac{\partial \mathbf{R}'_{\rho 1}}{\partial c_1} & \dots & \frac{\partial \mathbf{R}'_{\rho 1}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}'_{\rho 1}}{\partial \rho} & \frac{\partial \mathbf{R}'_{\rho 1}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}'_{\rho 1}}{\partial \rho E} \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
\frac{\partial \mathbf{R}'_{\rho N_s}}{\partial c_1} & \dots & \frac{\partial \mathbf{R}'_{\rho N_s}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}'_{\rho N_s}}{\partial \rho} & \frac{\partial \mathbf{R}'_{\rho N_s}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}'_{\rho N_s}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho}}{\partial c_1} & \dots & \frac{\partial \mathbf{R}_{\rho}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_1} & \dots & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho E}}{\partial c_1} & \dots & \frac{\partial \mathbf{R}_{\rho E}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E}
\end{pmatrix}$$

Decoupled Adjoint Solver

- Time march adjoint solution with approximate jacobians

$$\left(\frac{V}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}^{T}\right)\Lambda_{\mathbf{V}} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^{T}\underbrace{\left(\frac{\partial \mathbf{R}_{\mathbf{U}}}{\partial \mathbf{U}}\Lambda_{\mathbf{U}} + \frac{\partial f}{\partial \mathbf{U}}\right)}_{\text{Fully Coupled Residual}}$$

- Split $\frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}$ in same fashion as flow solver

$$\begin{pmatrix}
\frac{\partial \mathbf{R}'_{\rho_1}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial \rho} & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial \rho E} \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
\frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial \rho} & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho E}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho E}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E}
\end{pmatrix}$$

Decoupled Adjoint Solver

- Time march adjoint solution with approximate jacobians

$$\left(\frac{V}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}\right)^{T} \Lambda_{\mathbf{V}} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{V}}\right)^{T} \underbrace{\left(\frac{\partial \mathbf{R}_{\mathbf{U}}}{\partial \mathbf{U}}\Lambda_{\mathbf{U}} + \frac{\partial f}{\partial \mathbf{U}}\right)}_{\text{Fully Coupled Residual}}$$

- Split $\frac{\partial \tilde{\mathbf{R}}_{\mathbf{V}}}{\partial \mathbf{V}}$ in same fashion as flow solver

$$\begin{pmatrix}
\frac{\partial \mathbf{R}'_{\rho_1}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial \rho} & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}'_{\rho_1}}{\partial \rho E} \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
\frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial \rho} & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}'_{\rho_{N_s}}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho E}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho E}}{\partial c_{N_s}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E}
\end{pmatrix}
\rightarrow
\overbrace{\begin{pmatrix}
\frac{\partial \mathbf{R}_{\rho_1}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho_1}}{\partial c_{N_s}} \\
\vdots & \ddots & \vdots \\
\frac{\partial \mathbf{R}_{\rho_{N_s}}}{\partial c_1} & \cdots & \frac{\partial \mathbf{R}_{\rho_{N_s}}}{\partial c_{N_s}}
\end{pmatrix}}^{\frac{\partial \tilde{\mathbf{R}}_{\hat{\mathbf{V}}}}{\partial \hat{\mathbf{V}}}},
\overbrace{\begin{pmatrix}
\frac{\partial \mathbf{R}_{\rho}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho \mathbf{u}}}{\partial \rho E} \\
\frac{\partial \mathbf{R}_{\rho E}}{\partial \rho} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho \mathbf{u}} & \frac{\partial \mathbf{R}_{\rho E}}{\partial \rho E}
\end{pmatrix}}^{\frac{\partial \tilde{\mathbf{R}}_{\mathbf{U}'}}{\partial \mathbf{U}'}}$$

Species Equations

Mixture Equations

Decoupled Adjoint Solver

- Separate into two systems and solve as block jacobi scheme

Species Continuity Equations :

$$\boxed{\left(\frac{\rho\Omega}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\hat{\mathbf{V}}}}{\partial \hat{\mathbf{V}}}\right)^T \Delta\Lambda_{\hat{\mathbf{V}}} = -\left(\frac{\partial \mathbf{U}}{\partial \hat{\mathbf{V}}}\right)^T \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}^T \Lambda^n_{\mathbf{U}} + \frac{\partial f}{\partial \mathbf{U}}\right)}$$

Mixture Equations :

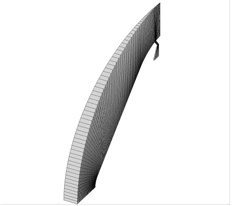
$$\boxed{\left(\frac{\Omega}{\Delta t}\mathbf{I} + \frac{\partial \tilde{\mathbf{R}}_{\mathbf{U}'}}{\partial \mathbf{U}'}\right)^T \Delta\Lambda_{\mathbf{U}'} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{U}'}\right)^T \left(\frac{\partial \mathbf{R_U}}{\partial \mathbf{U}}^T \Lambda^n_{\mathbf{U}} + \frac{\partial f}{\partial \mathbf{U}}\right)}$$

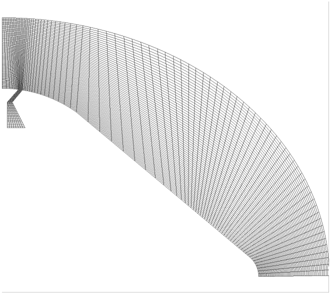
Outline

Geometry and Test Conditions

- Requirements of demonstration problem:
  - Steady flow
  - Hypersonic flow condition with challenging physics
  - Chemical non-equilibrium
- Extending hypersonic retro-propulsion work by Gnoffo et al. from perfect gas simulation to hydrogen-air mixture meets these requirements
- Focus on finite-rate chemical kinetics to affect aerothermodynamic quantities.
- Results can be non-physical (inviscid flow), but must be relatively free of numerical scheme inconsistencies

Geometry and Test Conditions





Annular jet geometry.

Parameter	Description	Value
r_{throat}	nozzle throat radius, m	0.02
$r_{plenum,inner}$	inside nozzle radius at plenum face, m	0.02
$r_{plenum,outer}$	outside nozzle radius at plenum face, m	0.07
$r_{exit,inner}$	inside nozzle radius at exit, m	0.064
$r_{exit,outer}$	outside nozzle radius at exit, m	0.08
l_{conv}	distance from plenum to throat, m	0.05
θ_c	cone half angle, deg	50.0

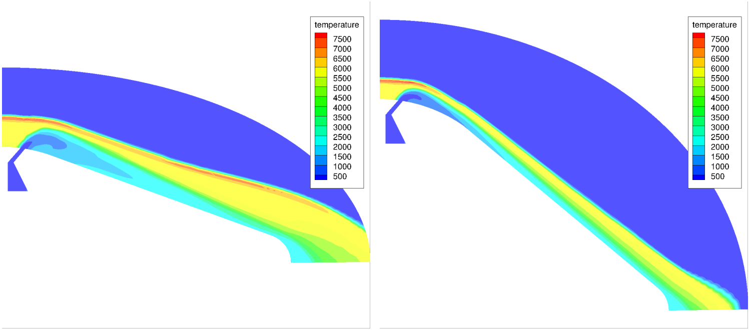
Annular nozzle geometry inputs.

Flow Condition	Description	Value
V_∞	freestream velocity, m/s	5686.24
ρ_∞	freestream density, kg/m^3	0.001
T_∞	freestream temperature, K	200.0
M_∞	freestream Mach number (derived)	20.0

Flow conditions.

Flow Features and Cone Angle Effects

- Flow separates downstream of nozzle exit, creating a buffer gas zone of relatively low temperature.

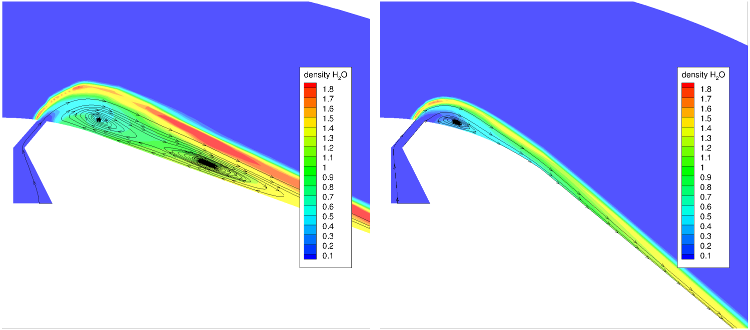


(a) 70^o Cone angle (b) 50^o Cone angle

Annular jet temperature contours, blowing pure H_2.

Flow Features and Cone Angle Effects

- Strong dependence between cone angle and size of recirculation zone



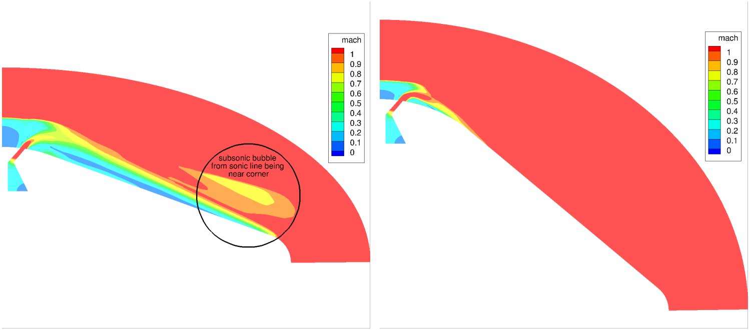
(a) 70^o Cone angle

(b) 50^o Cone angle

Annular jet H_2O density contours, blowing pure H_2.

Flow Features and Cone Angle Effects

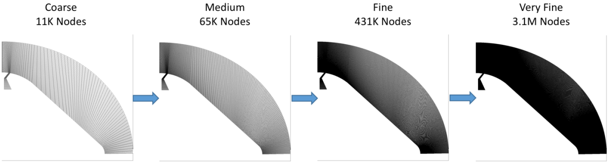
- 70^o Cone angle leads to sonic-corner body with low-amplitude oscillations in the separation bubble



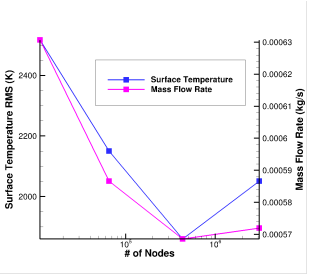
(a) 70^o Cone angle (b) 50^o Cone angle

Annular jet sonic line comparison, blowing pure H_2.

Mesh Refinement Study



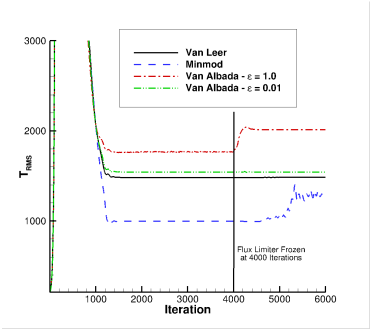
- Flow becomes unsteady as mesh is refined
- Very unsteady by finest grid level (3.1M Nodes)
- Coarse mesh used in optimization still retains all of the challenging physics of the problem

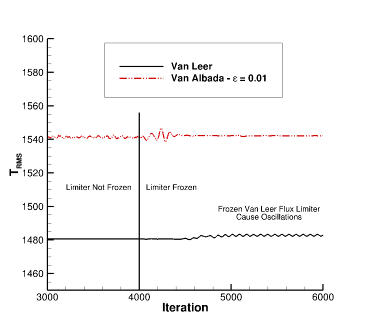


Surface temperature and plenum mass flow rate.

Flux Limiter Sensitivity

- First-order and second-order solutions very different, so choice of flux limiter has strong impact on steady state solution
- Van Albada with tuneable parameter, $\varepsilon = 0.01$, least sensitive to "freezing" the limiter





Flux limiter impact on surface temperature.

Outline

Cost Function Definition

- The optimization goal is to minimize a cost function generically defined as

$$f = \sum_{j=1}^{N_{func}} w_j \left(C_j - C_{j^*} \right)^{p_j}$$

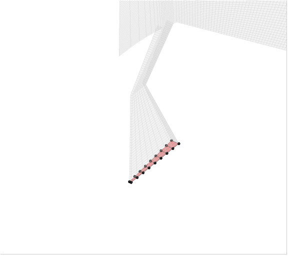
$$C_j = \text{Component value} \quad C_{j^*} = \text{Component target}$$

$$w_j = \text{Component weight} \quad p_j = \text{Component power}$$

$$N_{func} = \text{Number of cost function components}$$

Cost Function Integrated Areas

(a) T_{RMS} integrated area



(b) $\dot{m}_p$ integrated area

- Define cost function to minimize surface temperature RMS, T_{RMS} and mass flow rate $\dot{m}_p$ through the annular plenum

$$f = w_1 \left(\dot{m}_p - \dot{m}_p^*\right)^2 + w_2 \left(T_{RMS} - T_{RMS}^*\right)^2$$

Design Variables

- These plenum conditions are used as design variables:
  - Plenum total pressure, $P_{p,o}$
  - Plenum fuel-air ratio, ϕ_p
- The mass fractions of species H_2 and N_2 at the plenum are defined by the fuel-air ratio

$$c_{H_2} = \phi_p$$
$$c_{N_2} = 1 - \phi_p$$

- $\phi_p = 1$ corresponds to blowing pure H_2 from annular nozzle
- $\phi_p = 0$ corresponds to blowing pure N_2 from annular nozzle

First-Order, Inverse Design Optimization

- Sensitivity to frozen flux limiter mandated first-order for inverse design optimization
- Cost function with semi-arbitary targets specified

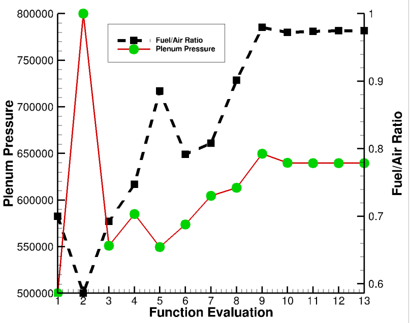
$$f = w_1 \left(\dot{m}_p - 0.0024\right)^2 + w_2 \left(T_{RMS} - 2000\right)^2$$

- Weights chosen to normalize components for starting condition

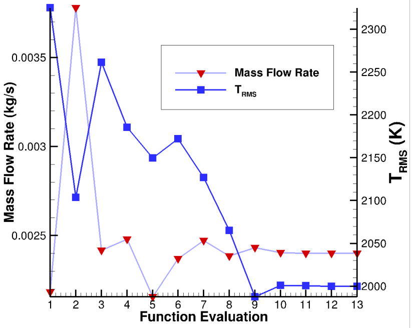
$$\frac{w_1}{w_2} = \frac{\left(T_{RMS} - 2000\right)^2}{\left(\dot{m}_p - 0.0024\right)^2}$$

First-Order, Inverse Design Optimization

- Target design is mostly met within 10 function evaluations



(a) Design variables

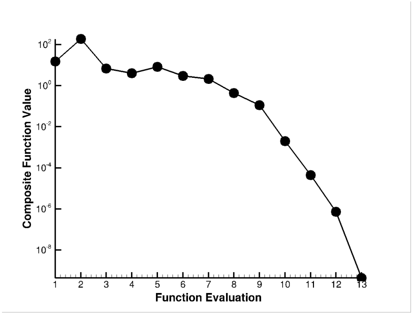


(b) Surface temperature RMS and mass flow rate

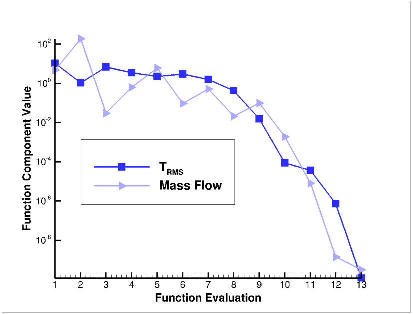
Inverse design history

First-Order, Inverse Design Optimization

- Optimization terminated when cost function $< 10^{-8}$
- Cost function components are well normalized and approach zero as optimizer drives design to specified target



(a) Cost function value



(b) Component values

Inverse design history

Second-Order, Direct Design Optimization

- For direct optimization, the cost function targets for T_{RMS} and $\dot{m}_p$ are set to zero

$$f = w_1 \left(\dot{m}_p\right)^2 + w_2 \left(T_{RMS}\right)^2$$

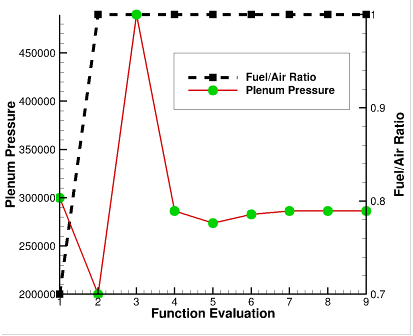
- Weights again chosen to normalize components based on starting design condition

$$\frac{w_1}{w_2} = \frac{\left(T_{RMS}\right)^2}{\left(\dot{m}_p\right)^2}$$

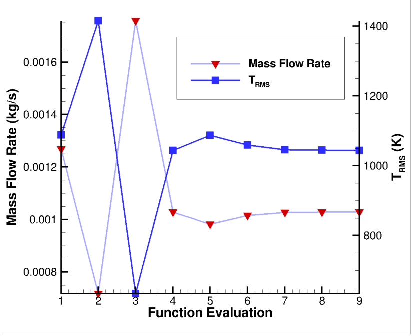
- This direct design optimization is much more sensitive to the choice of weights than the previous inverse design optimization
- Choice of weight can also be based on design requirements (i.e. preference to less mass over lower temperature)

Second-Order, Direct Design Optimization

- Design mostly driven by chemistry and H_2 lower molecular weight, relative to N_2
- The optimization is terminated after 9 function evaluations, when change in all design variables $< 10^{-8}$



(a) Design variables

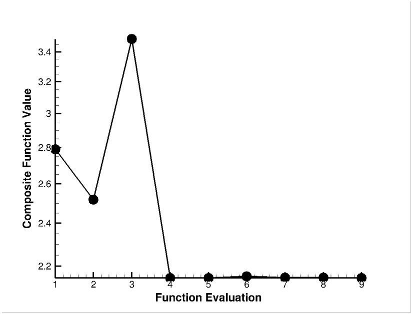


(b) Surface temperature RMS and mass flow rate

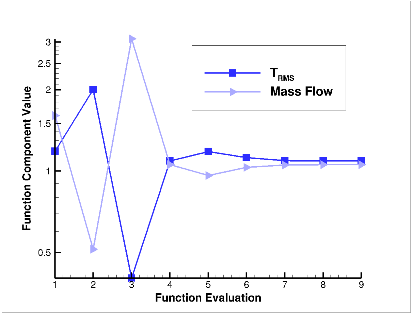
Inverse design history

Second-Order, Direct Design Optimization

- Cost function components are well normalized
- Noticable net improvements stop after 4 function evaluations, due to weak dependence on plenum total pressure



(a) Cost function value



(b) Component values

Inverse design history

Outline

Accuracy of the Adjoint Sensitivity Derivatives

- To verify adjoint, derivatives computed by complex step eliminates subtractive cancellation error

$$\frac{\partial f}{\partial x} = \frac{Im\left[f\left(x+ih\right)\right]}{h} - O(h^2)$$

- FUN3D is "complexified" by ruby scripting, and sensitivities computed by perturbing design variables individually
- The "complexified" solver is restarted with the real-value flow solver solution to avoid recomputing flux limiter
    - Ensures that frozen flux limiter values used by adjoint solver and "complexified" flow solver are identical

Accuracy of the Adjoint Sensitivity Derivatives

- Excellent agreement between derivatives computed by complex step and those computed by adjoint for frozen flow

Design Variable	Adjoint	Complex	Relative Difference
$P_{p,o}$	-0.18451007644622E-06	-0.184510076442032E-06	2.27e-11
$T_{p,o}$	0.62086963151678E-03	0.620869631517086E-03	4.93e-13
ϕ_p	-0.34045335117520E-01	-0.340453351177196E-01	5.86e-12

Sensitivity derivative comparison - H_2-N_2 frozen.

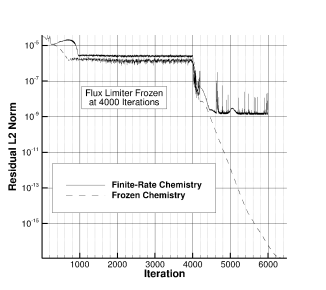
- Less agreement for chemically reacting flow

Design Variable	Adjoint	Complex	Relative Difference
$P_{p,o}$	-0.11081315601976E-06	-0.110529774659536E-06	2.56e-03
$T_{p,o}$	0.19089941237390E-03	0.190892847933810E-03	3.44e-05
ϕ_p	-0.28035409045530E-01	-0.280251731728184E-01	3.65e-04

Sensitivity derivative comparison - H_2-N_2 reacting.

Accuracy of the Adjoint Sensitivity Derivatives

- Difference is explained by relative convergence of frozen flow vs. flow with finite-rate chemistry



- Convergence of reacting flow stalls ~ 8 orders of magnitude before the convergence of frozen flow.
- This degrades the discrete comparison of adjoint and complex step derivatives, since many digits are still changing

Consistency of Decoupled Flow Solver

- Decoupled and fully coupled flow solvers produce the same surface temperature RMS and almost the same plenum mass flow rate for frozen flow

Quantity	Decoupled	Fully Coupled	Relative Difference
$\dot{m}_p$	0.8450858226893225E-03	0.8450858226893034E-03	2.26E-14
T_{RMS}	0.1508600871984388E+04	0.1508600871984388E+04	0

Difference with frozen chemistry.

- Comparison degrades for chemically reacting flows, but is consistent with the aforementioned relative level of convergence

Quantity	Decoupled	Fully Coupled	Relative Difference
$\dot{m}_p$	0.8448720721551258E-03	0.8448720721551088E-03	2.01E-14
T_{RMS}	0.1545729169703027E+04	0.1545720949811712E+04	5.32E-06

Difference with finite-rate chemistry.

Consistency of Decoupled Adjoint Solver

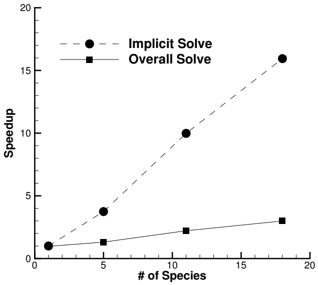
- Very good agreement between the sensitivity derivatives computed by the decoupled and fully coupled adjoint solvers

Design Variable	Decoupled Sensitivity	Fully Coupled Sensitivity	Rel. Diff
$P_{p,o}$	-0.11081315601976E-06	-0.11081315649260E-06	4E-9
$T_{p,o}$	0.19089941243495E-03	0.19089941237390E-03	3E-10
ϕ_p	-0.28035409093945E-01	-0.28035409045530E-01	1E-9

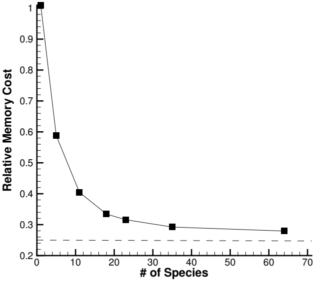
Annular jet plenum sensitivity relative difference.

- All differences less than those in comparison to complex step derivatives
- More than sufficient for optimization procedure to converge to the same optimal result

Relative Speedup of Decoupled Flow Solver



(a) Relative cost



(b) Relative memory required

- 5 km/s Cylinder flow confirms that speedup in linear solver and overall solver increases linearly with number of species
- Relative memory required by the flow solver approaches $\sim 1/4$, which is correct when off-diagonal jacobian elements are single precision

Relative Speedup of Decoupled Flow Solver

- For annular jet demonstration problem, speedup is much better for frozen flow, where convergence is clean

Scheme	Time (s)	Speedup
Fully Coupled (Approximate)	348.6	1.0 (baseline)
Fully Coupled (Exact)	265.6	1.31
Decoupled	138.7	2.51

Relative speedup for frozen chemistry.

- Less impressive for chemically reacting flow

Scheme	Time (s)	Speedup
Fully Coupled (Exact)	280.6	1.0 (baseline)
Fully Coupled (Approximate)	270.4	1.04
Decoupled	168.6	1.66

Relative speedup for finite-rate chemistry.

Relative Speedup of Decoupled Flow Solver

- Reasons for decrease in decoupled flow solver speedup with finite-rate chemistry
  - Exact linearizations signicantly improvement iterative convergence, but slightly more expensive to compute jacobian
    - Exact linearizations do not help convergence significantly during start-up transients, where flow is highly non-linear
  - Computing the Jacobian is a dominant cost in the decoupled flow solver
    - The Jacobian is recomputed more frequently when solution is not converging, which decreases the speedup of the decoupled scheme
- Future work will focus on improving iterative convergence with finite-rate chemistry and optimizing the Jacobian computation