Error Estimation and Mesh Adaptation using Output Adjoints

Krzysztof J. Fidkowski, *University of Michigan*

38th Advanced CFD Lecture Series
von Karman Institute, Rhode-St-Genèse, Belgium

September 14-17, 2015

Outline

Introduction

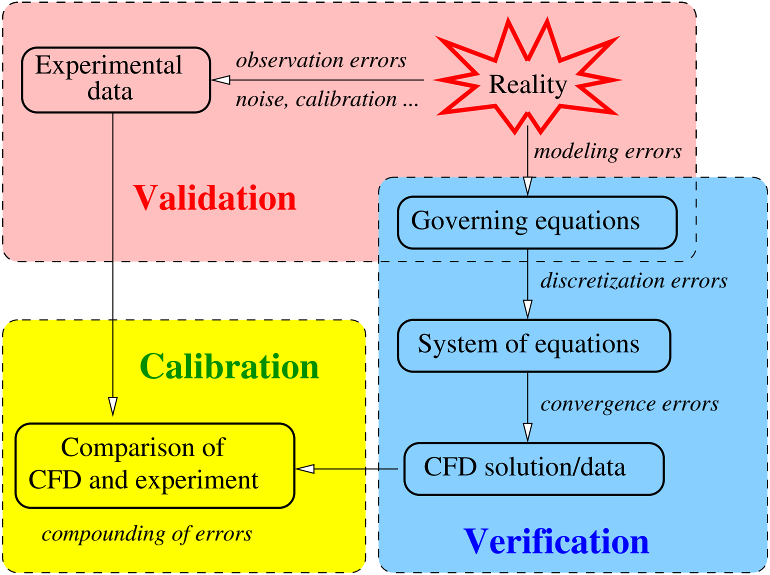
Complex CFD simulations are made possible by

- Increasing computational power
- Improvements in numerical algorithms

New liability: ensuring accuracy of computations

- Management by expert practitioners is not feasible for increasingly-complex flow fields
- Reliance on best-practice guidelines is an open-loop solution: numerical error is unchecked for novel configurations
- Output calculations are not yet sufficiently robust, even on relatively standard simulations

Errors in simulations come from various sources



Improving CFD robustness

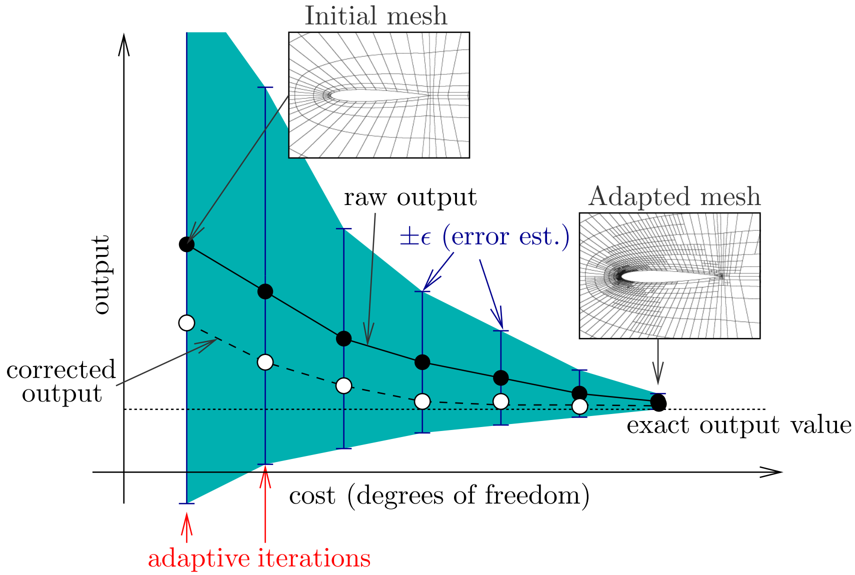
Error estimation

- Error estimates on outputs of interest are necessary for confidence in CFD results
- Mathematical theory exists for obtaining such estimates
- Recent works demonstrate the success of this theory for aerospace applications

Mesh adaptation

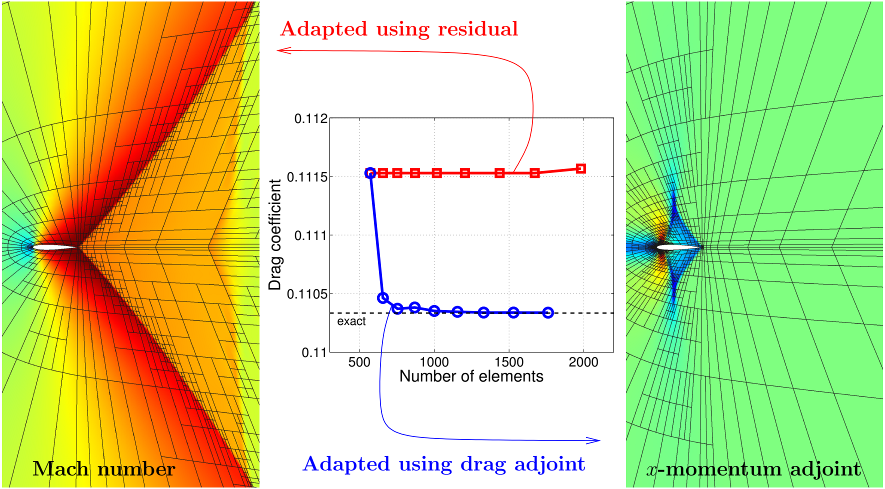
- Error estimation alone is not enough
- Engineering accuracy for complex aerospace simulations demands mesh adaptation to control numerical error
- Automated adaptation improves robustness by closing the loop in CFD analysis

A typical output-adaptive result



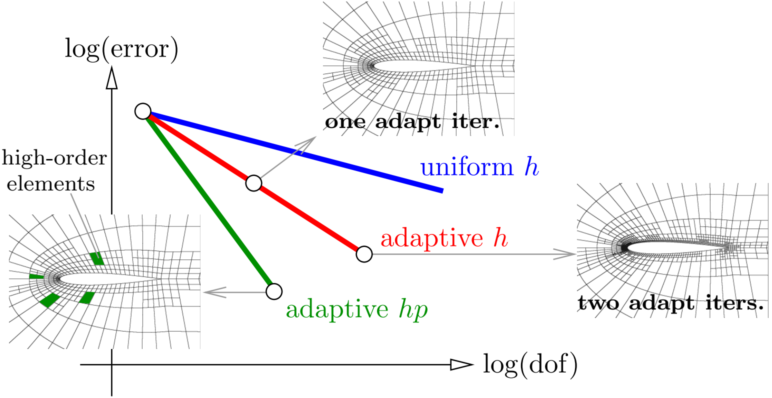
Why not just adapt “obvious” regions?

Fishtail shock in $M_\infty = 0.95$ inviscid flow over a NACA 0012 airfoil



hp Mesh adaptation

- Adaptation can isolate singularities with small elements
- In many high-order methods, local p-enrichment is possible
- Combination of both can yield a powerful method for efficiently improving accuracy



Outline

Conservation equations

$$\boxed{\mathbf{r}(\mathbf{u}) \equiv \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{H}}(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}}$$

- $\mathbf{u} \in \mathbb{R}^s$ is the state vector
- $\vec{\mathbf{H}} \in [\mathbb{R}^s]^d$ is the total flux with spatial components

$$\mathbf{H}_i \quad = \quad \underbrace{\mathbf{F}_i(\mathbf{u})}_{\text{inviscid flux}} + \underbrace{\mathbf{G}_i(\mathbf{u}, \nabla \mathbf{u})}_{\text{viscous flux}}$$

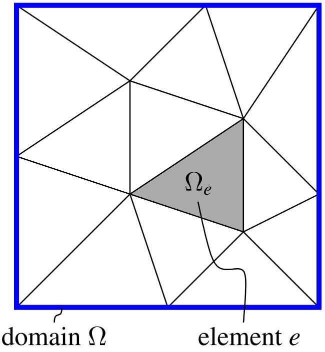
- $1 \le i \le d$ = spatial dimension
- The viscous flux is linear in the state gradient

$$\mathbf{G}_i(\mathbf{u}, \nabla \mathbf{u}) = -\mathbf{K}_{ij}(\mathbf{u})\, \partial_j \mathbf{u}$$

Finite-element solution approximation

Polynomials of order p_e on each element:

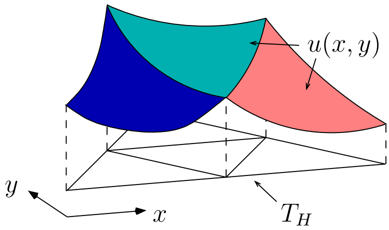
$$\mathbf{u}_h(\vec{x}) \approx \sum_{e=1}^{N_e} \sum_{n=1}^{N_{p_e}} \mathbf{U}_{en}\phi_{en}(\vec{x})$$



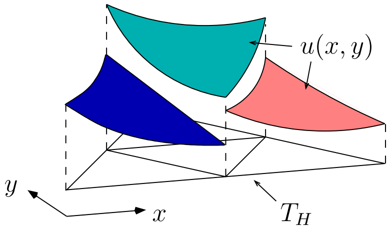
N_e	=	# of elements
N_{p_e}	=	# of basis fcns on element e
$\phi_{en}(\vec{x})$	=	n^{th} basis fcn of order p_e on e
p_e	=	approximation order on element e
$\mathbf{U}_{en}$	=	vector of s coefficients on n^{th} basis function on element e

Discontinuous basis functions

Continuous Galerkin (CG)



Discontinuous Galerkin (DG)



- DG approximation space: no inter-element continuity,

$$\mathbf{u} \in \boldsymbol{\mathcal{V}}_h = [\mathcal{V}_h]^s, \quad \mathcal{V}_h = \left\{ u \in L^2(\Omega) : u|_{\Omega_e} \in \mathcal{P}^{p_e}(\Omega_e)\ \forall\, \Omega_e \in T_h \right\}$$

- Equations: multiply by test functions and integrate by parts

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) \equiv \int_\Omega \mathbf{v}_h^T \mathbf{r}(\mathbf{u}_h) d\Omega = 0, \quad \forall \mathbf{v}_h \in \boldsymbol{\mathcal{V}}_h$$

- Elements coupled together through upwind flux functions

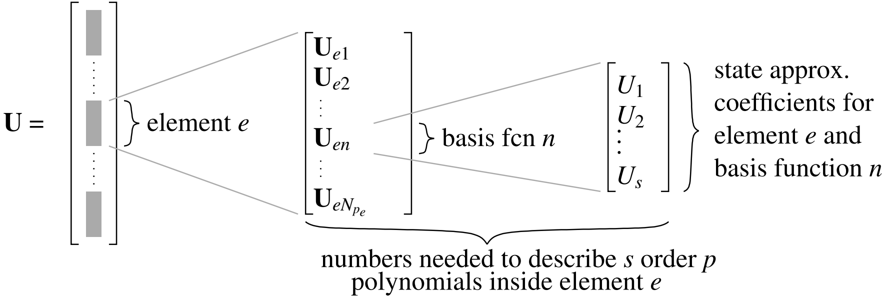
Discrete system

- Discrete residual on element e for n^{th} test function,

$$\mathbf{R}_{en} \equiv \{\mathcal{R}_h(\mathbf{u}_h, \phi_{en}\mathbf{e}_r)\}_{r=1\dots s} \in \mathbb{R}^s$$

- We lump all residuals and states into single vectors (size N),

$$\mathbf{R}(\mathbf{U}) = \mathbf{0}$$



Outline

Local sensitivities

- Suppose N_μ parameters affect our PDE, but we only have one scalar output, $J(\mathbf{U})$:

$$\underbrace{\boldsymbol{\mu}}_{\text{inputs} \in \mathbb{R}^{N_\mu}} \rightarrow \underbrace{\mathbf{R}(\mathbf{U},\boldsymbol{\mu}) = \mathbf{0}}_{N \text{ equations}} \rightarrow \underbrace{\mathbf{U}}_{\text{state} \in \mathbb{R}^N} \rightarrow \underbrace{J(\mathbf{U})}_{\text{output (scalar)}}$$

- We are interested in how J changes with $\boldsymbol{\mu}$,

$$\frac{dJ}{d\boldsymbol{\mu}} \in \mathbb{R}^{1\times N_\mu} = N_\mu \text{ sensitivities}$$

- Brute force approach: perturb each entry in $\boldsymbol{\mu}$ individually, re-solve the PDE, and measure the perturbation in the output

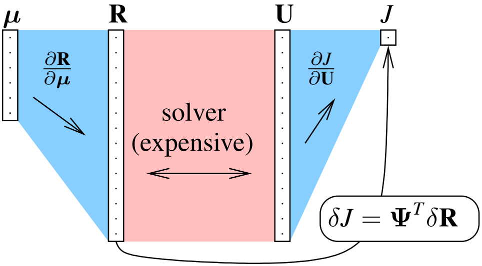
This is inefficient for large N_μ

The discrete adjoint

- We can efficiently compute sensitivities using a discrete adjoint vector, $\mathbf{\Psi} \in \mathbb{R}^N$,

$$\frac{dJ}{d\boldsymbol{\mu}} = \mathbf{\Psi}^T \frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}$$

- Each entry in $\mathbf{\Psi}$ is the sensitivity of J to residual source perturbations in the corresponding entry in $\mathbf{R}$



The discrete adjoint equation

- Consider a small perturbation $\delta\mathbf{R}$ to the residual
- The resulting (linearized) state perturbation, $\delta\mathbf{U}$ satisfies

$$\frac{\partial\mathbf{R}}{\partial\mathbf{U}}\delta\mathbf{U} + \delta\mathbf{R} = 0$$

- Also linearizing the output we have,

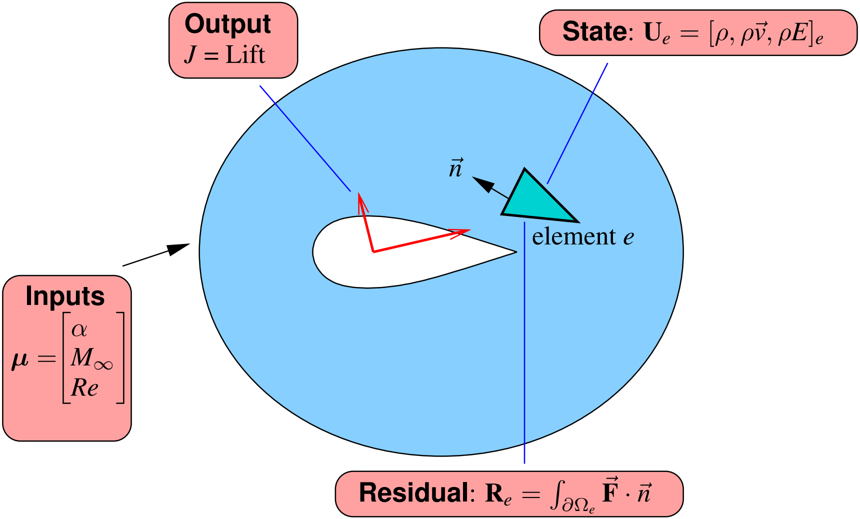
$$\delta J = \underbrace{\frac{\partial J}{\partial\mathbf{U}}\delta\mathbf{U} = \overbrace{\mathbf{\Psi}^T\delta\mathbf{R}}_{\text{adjoint definition}} = -\mathbf{\Psi}^T\frac{\partial\mathbf{R}}{\partial\mathbf{U}}\delta\mathbf{U}}^{\text{linearized equations}}$$

- Requiring the above to hold for arbitrary perturbations yields the linear *discrete adjoint equation*

$$\boxed{\left(\frac{\partial\mathbf{R}}{\partial\mathbf{U}}\right)^T\mathbf{\Psi} + \left(\frac{\partial J}{\partial\mathbf{U}}\right)^T = \mathbf{0}}$$

Adjoints in aerodynamics

Consider flow over an airfoil:

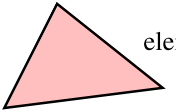


Output sensitivity to residuals: the adjoint

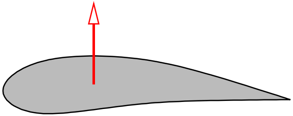
The lift adjoint $\mathbf{\Psi}$ is the sensitivity of lift to residual sources.

Lift$= J(\mathbf{U})$

We have a solution $\mathbf{U}$ when $\mathbf{R} = 0$



element e

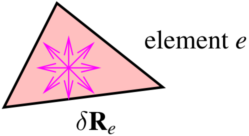


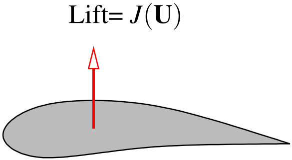
state $\mathbf{U}$

Output sensitivity to residuals: the adjoint

The lift adjoint $\mathbf{\Psi}$ is the sensitivity of lift to residual sources.

We have a solution $\mathbf{U}$ when $\mathbf{R} = 0$



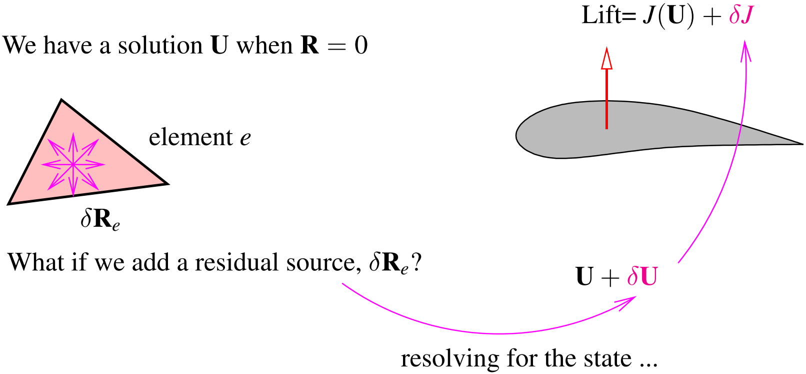


What if we add a residual source, $\delta\mathbf{R}_e$?

$\mathbf{U}$

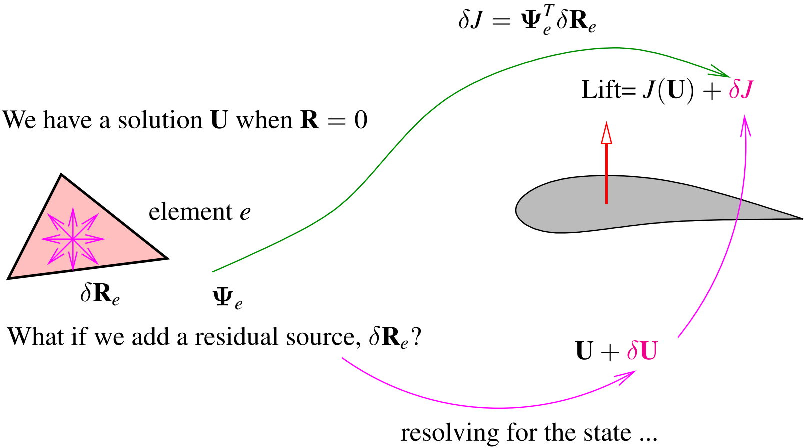
Output sensitivity to residuals: the adjoint

The lift adjoint $\boldsymbol{\Psi}$ is the sensitivity of lift to residual sources.

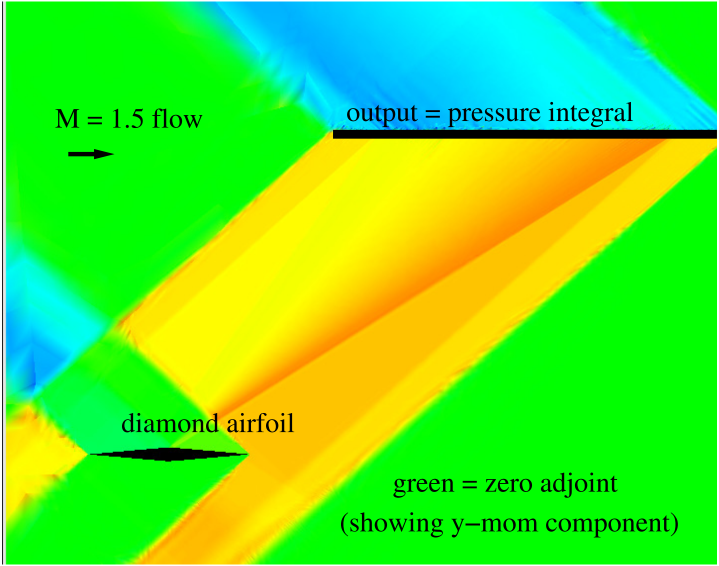


Output sensitivity to residuals: the adjoint

The lift adjoint $\mathbf{\Psi}$ is the sensitivity of lift to residual sources.

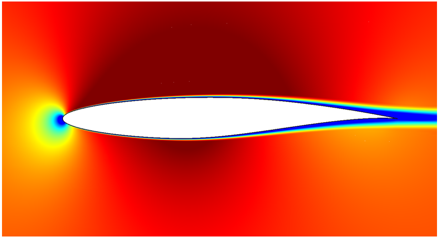


Sample steady adjoint solution

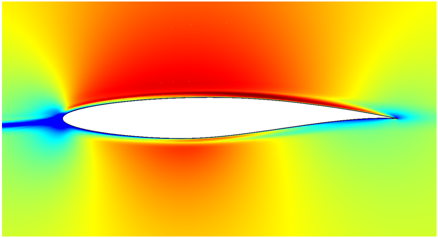


Another steady adjoint solution

RAE 2822, $M_\infty = 0.5$, $Re = 10^5$, $\alpha = 1^\circ$



x-momentum primal state



cons. of x-mom drag adjoint

- Adjoint shares similar qualitative features with primal
- Note wake “reversal” in adjoint solution
- The discrete adjoint solution approximates the continuous adjoint when the discretization is *adjoint consistent*

Adjoint implementation

- The discrete adjoint, $\mathbf{\Psi}$, is obtained by solving a linear system
- This system involves linearizations about the primal solution, $\mathbf{U}$, which is generally obtained first
- When the full Jacobian matrix, $\frac{\partial \mathbf{R}}{\partial \mathbf{U}}$, and an associated linear solver are available, the transpose linear solve is straightforward
- When the Jacobian matrix is not stored, the discrete adjoint solve is more involved: all operations in the primal solve must be linearized, transposed, and applied in reverse order
- In unsteady discretizations, the adjoint must be marched backward in time from the final to the initial state

Outline

Output error estimation

We want: $\delta J = J_H(\mathbf{U}_H) - J(\mathbf{U})$

This is the difference between J computed with the discrete system solution, $\mathbf{U}_H$, and J computed with the *exact* solution, $\mathbf{U}$

We'll settle for: $\delta J = J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)$

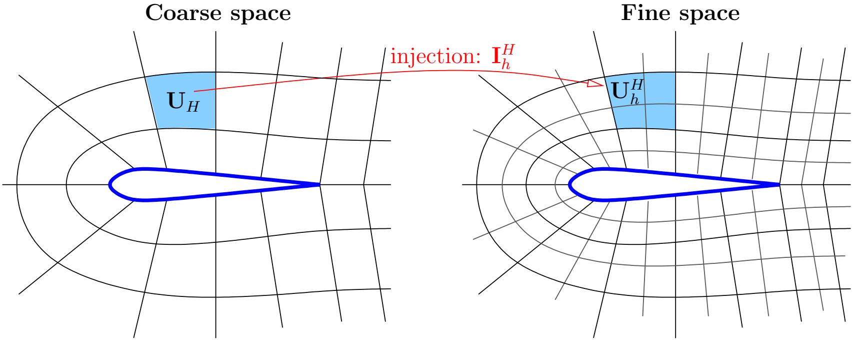
This is the difference in J relative to a finer discretization (h)

$$\text{coarse space:} \quad \rightarrow \underbrace{\mathbf{R}_H(\mathbf{U}_H) = 0}_{N_H \text{ equations}} \rightarrow \underbrace{\mathbf{U}_H}_{\text{state} \in \mathbb{R}^{N_H}} \rightarrow \underbrace{J_H(\mathbf{U}_H)}_{\text{output (scalar)}}$$

$$\text{fine space:} \quad \rightarrow \underbrace{\mathbf{R}_h(\mathbf{U}_h) = 0}_{N_h \text{ equations}} \rightarrow \underbrace{\mathbf{U}_h}_{\text{state} \in \mathbb{R}^{N_h}} \rightarrow \underbrace{J_h(\mathbf{U}_h)}_{\text{output (scalar)}}$$

Fine-space injection

- The fine space can arise from h or p refinement
- Define an injection of the coarse state into the fine space



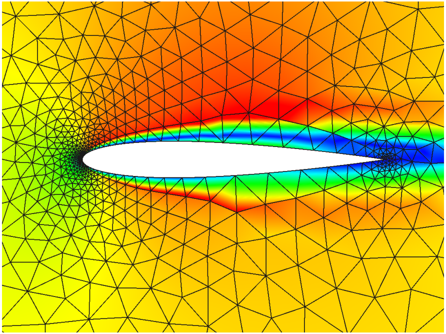
- $\mathbf{U}_h^H$ will generally not satisfy the fine-space equations,

$$\mathbf{R}_h(\mathbf{U}_h^H) \neq \mathbf{0}$$

Fine-space residuals

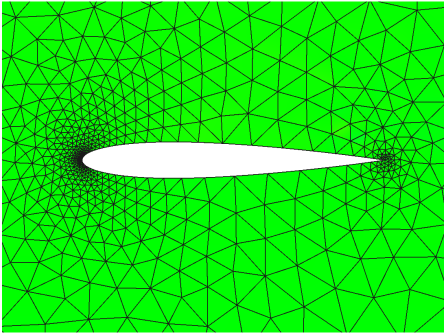
- A finer space (e.g. order enrichment) can uncover residuals in a converged solution
- Example: NACA 0012 at $\alpha = 2^\circ$ in $Re = 5000$, $M_\infty = 0.5$ flow

Coarse space state, $\mathbf{U}_H$



$p_H = 1$

Coarse space residual, $\mathbf{R}_H(\mathbf{U}_H)$

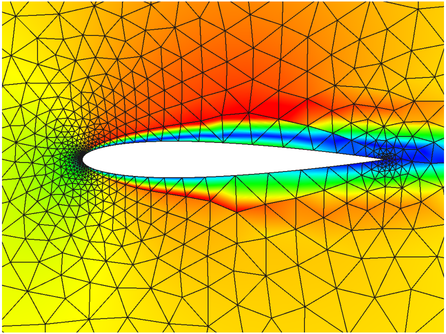


Zero as expected

Fine-space residuals

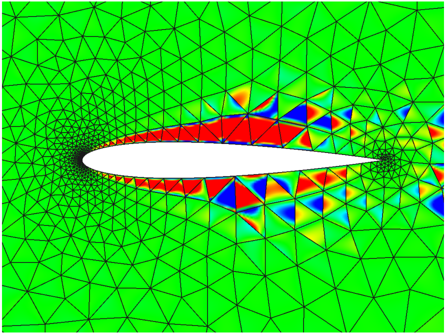
- A finer space (e.g. order enrichment) can uncover residuals in a converged solution
- Example: NACA 0012 at $\alpha = 2^\circ$ in $Re = 5000$, $M_\infty = 0.5$ flow

Injected state, $\mathbf{U}_h^H$



$p_h = 2$

Fine space residual, $\mathbf{R}_h(\mathbf{U}_h^H)$



Nonzero: new info

The adjoint-weighted residual

- $\mathbf{U}_h^H$ solves a *perturbed* fine-space problem

$$\text{find } \mathbf{U}_h' \text{ such that:} \quad \mathbf{R}_h(\mathbf{U}_h') \underbrace{-\mathbf{R}_h(\mathbf{U}_h^H)}_{\delta \mathbf{R}_h} = 0 \quad \Rightarrow \text{answer: } \mathbf{U}_h' = \mathbf{U}_h^H$$

- The fine-space adjoint, $\mathbf{\Psi}_h$, then tells us to expect an output perturbation of

$$\underbrace{J_h(\mathbf{U}_h^H) - J_h(\mathbf{U}_h)}_{\approx \delta J} = \mathbf{\Psi}_h^T \delta \mathbf{R}_h = -\mathbf{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H)$$

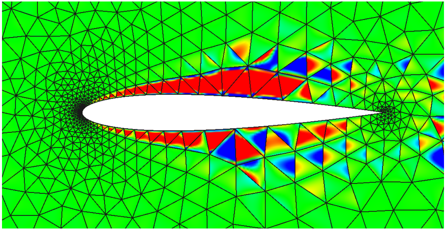
[4: Becker and Rannacher, 2007] [9: Giles and Pierce, 1997]

- This equation assumes small perturbations (e.g. if nonlinear)
- In summary, we have an *adjoint-weighted residual*:

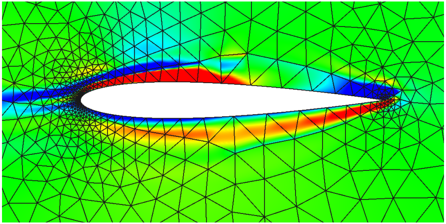
$$\boxed{\delta J \approx -\mathbf{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H)}$$

Adjoint-weighted residual example

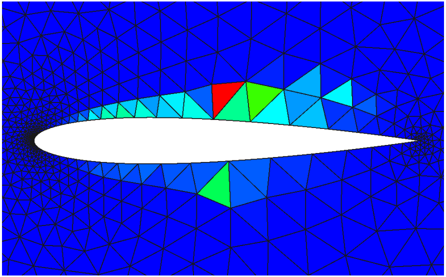
Fine space residual, $\mathbf{R}_h(\mathbf{U}_h^H)$



Fine space adjoint, $\mathbf{\Psi}_h$



Error indicator, $\epsilon_e = |\mathbf{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H)|$



Output error: $\delta J \approx -\mathbf{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H)$

Idea: adapt where ϵ_e is high, to reduce the residual there

Two more definitions

Corrected output

$$J_H^{\text{corrected}} = J_H - \delta J$$

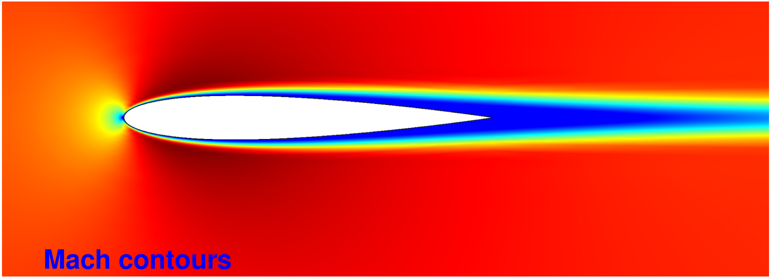
- Should converge faster than J_H
- Remaining error = error left in corrected output

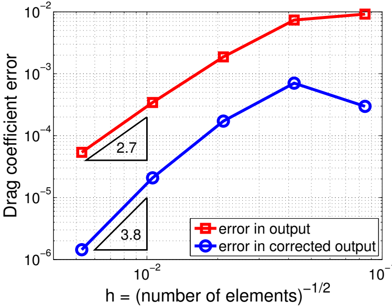
Error effectivity

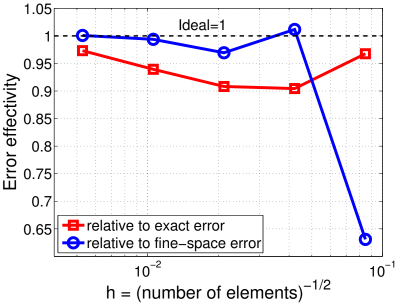
$$\eta_H = \frac{J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h)}{J_H(\mathbf{U}_H) - J}$$

- J = exact output
- We want η_H close to 1
- Effectivity is affected by choice of fine space

Drag error in viscous flow over an airfoil







Approximations

How do we calculate $\mathbf{\Psi}_h$ = the adjoint on the fine space?

Options:

1. Solve for $\mathbf{U}_h$ and then $\mathbf{\Psi}_h$ – expensive! Potentially still useful to drive adaptation. [14: Solín and Demkowicz, 2004]
2. Solve for the coarse space adjoint, $\mathbf{\Psi}_H$, and:
    1. Reconstruct $\mathbf{\Psi}_H$ on the fine space using a higher-accuracy stencil. Smoothness assumption on adjoint.
    [13: Rannacher, 2001] [3: Barth and Larson, 2002]
    [15: Venditti and Darmofal, 2002] [10: Lu, 2005] [8: Fidkowski and Darmofal, 2007]
    2. Initialize $\mathbf{\Psi}_h$ with $\mathbf{\Psi}_H$ and take a few iterative solution (smoothing) steps on the fine space.
    [2: Barter and Darmofal, 2008] [11: Oliver and Darmofal, 2008]

Corrections and remainders

- Define $\mathbf{\Psi}_h^H \equiv \mathbf{I}_h^H \mathbf{\Psi}_H$ = injection of $\mathbf{\Psi}_H$ into h.
- Define the adjoint perturbation, $\delta\mathbf{\Psi}_h \equiv \mathbf{\Psi}_h^H - \mathbf{\Psi}_h$
- Rewrite the adjoint-weighted residual as:

$$\delta J = \underbrace{-\left(\mathbf{\Psi}_h^H\right)^T \mathbf{R}_h(\mathbf{U}_h^H)}_{\text{computable correction}} + \underbrace{\left(\delta\mathbf{\Psi}_h\right)^T \mathbf{R}_h(\mathbf{U}_h^H)}_{\text{remaining error}} + \underbrace{\mathcal{O}(\delta\mathbf{U}_h, \delta\mathbf{\Psi}_h)^2}_{\text{error in estimate}}$$

- The computable correction is tempting to use directly, but:
  - It does not incorporate fine-space information $\Rightarrow$ it performs poorly as an adaptive indicator
  - It is zero for FEM with Galerkin orthogonality
- For nonlinear problems the “error in the estimate” can be reduced to third-order via [13: Rannacher, 2001]:

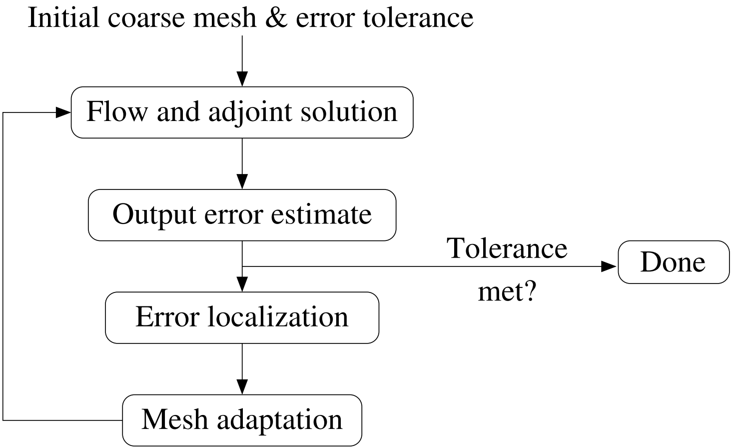
$$\delta J \approx -\left(\mathbf{\Psi}_h^H\right)^T \mathbf{R}_h(\mathbf{U}_h^H) + \frac{1}{2}\left(\delta\mathbf{\Psi}_h\right)^T \mathbf{R}_h(\mathbf{U}_h^H) + \frac{1}{2}\left(\delta\mathbf{U}_h\right)^T \mathbf{R}_h^{\psi}(\mathbf{\Psi}_h^H)$$

Error estimation summary

1. Solve the coarse-discretization forward and adjoint problems: $\mathbf{U}_H$ and $\mathbf{\Psi}_H$
2. Pick a fine discretization "h" (mesh refinement or order enrichment)
3. Calculate or approximate $\mathbf{\Psi}_h$ = adjoint on the fine space
4. Project $\mathbf{U}_H$ onto the fine discretization and calculate the residual $\mathbf{R}_h(\mathbf{U}_h^H)$
5. Weight the fine-space residual with the fine-space adjoint to obtain the output error estimate
6. The computed output error δJ is an estimate of the true error, <u>not a bound</u>

Outline

Mesh adaptation



Error localization

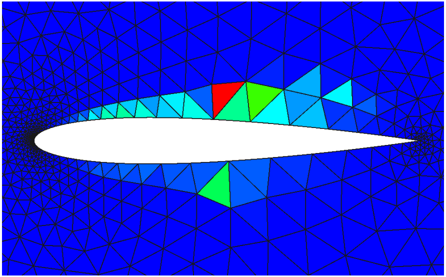
- Recall that the adjoint-weighted residual expression for the output error involves a sum over elements (e)

$$J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h) \approx -\mathbf{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H) \quad = \quad -\sum_e \mathbf{\Psi}_{he}^T \mathbf{R}_{he}(\mathbf{U}_h^H)$$

- The absolute-value of each element's contribution to the error is the **error indicator** on that element

$$\epsilon_e \quad \equiv \quad \left|\mathbf{\Psi}_{he}^T \mathbf{R}_{he}(\mathbf{U}_h^H)\right|$$

Right : plot of error indicator for a viscous DG simulation, $p_H = 1$, $p_h = 2$



Output-based mesh adaptation

Motivating ideas

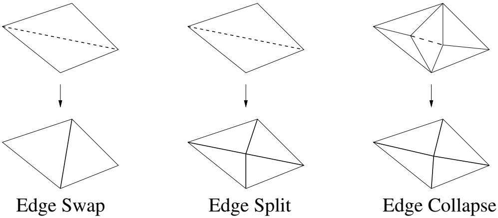
- The error indicator (ϵ_e) identifies elements with large adjoint-weighted residuals
- Locally refining a mesh reduces local residuals
- So we can reduce the output error by refining those elements that have a high ϵ_e

Adaptation choices

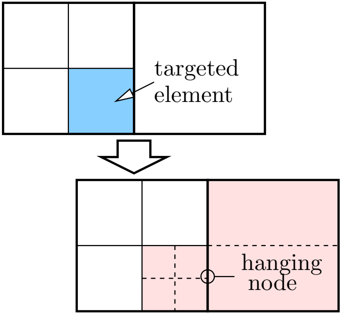
- Local refinement versus global re-meshing
- Which/how many elements should be targeted?
- Isotropic versus anisotropic refinement
- h, p, or hp mechanics
- Should coarsening be allowed?

Local mesh modification

- Modify the mesh incrementally (mesh generation is hard)
- Often more robust than global re-meshing
- With node movement, can be flexible for unstructured meshes
- Hanging nodes easily supported in DG



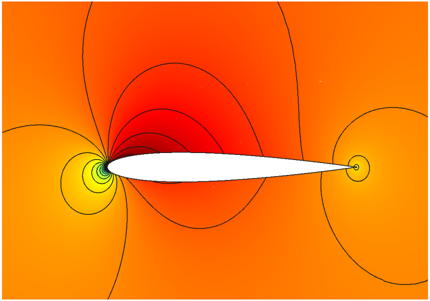
Unstructured local mesh operators



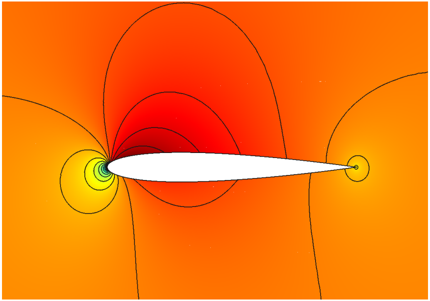
Hanging-node refinement

Example: inviscid flow over an airfoil

NACA 0012, Euler, $M_\infty = 0.5$, $\alpha = 2^\circ$



Mach number contours

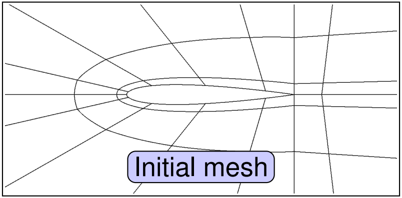


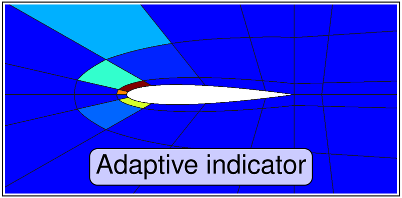
Drag adjoint ($x-$momentum)

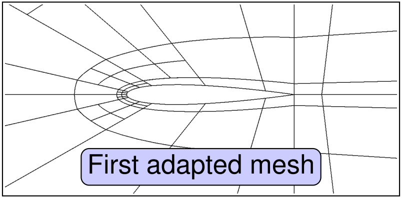
- Output $J =$ drag (expect ~ 0)
- Compare hanging-node adaptation to uniform refinement
- Look at approximation orders $p = 1$ and $p = 2$

Inviscid flow: hanging-node adaptation

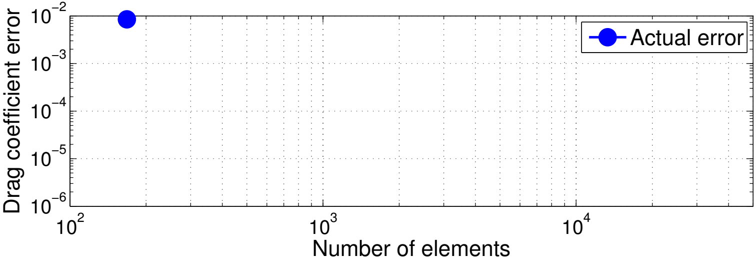
- Isotropic hanging-node refinement
- Fine space = $p+1$
- Fixed fraction $f^{\mathrm{adapt}} = 5\%$
- 20 adaptive iterations
- No coarsening
- Use adjoint-weighted residual δJ as correction
- "Exact" output from a $p = 3$ fine-mesh solve
- *Right:* $p = 2$ first adaptation

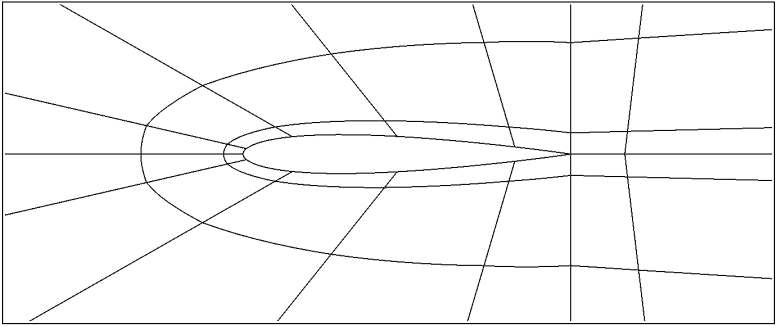




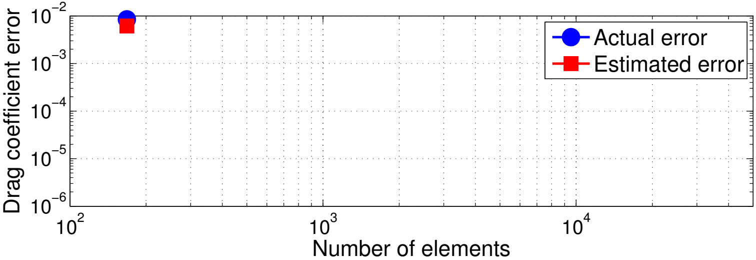


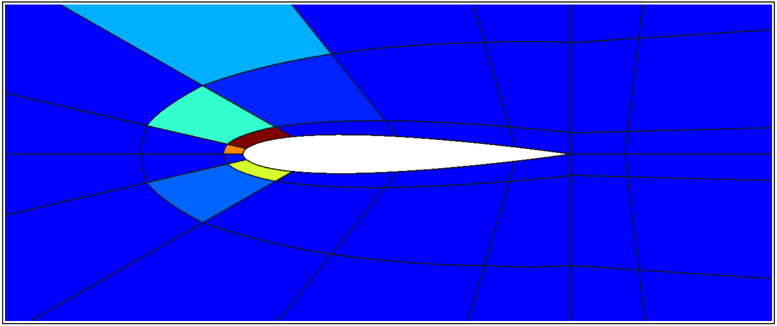
Inviscid flow: $p = 2$ mesh sequence



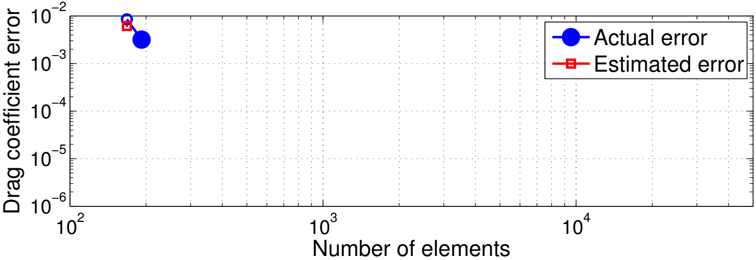


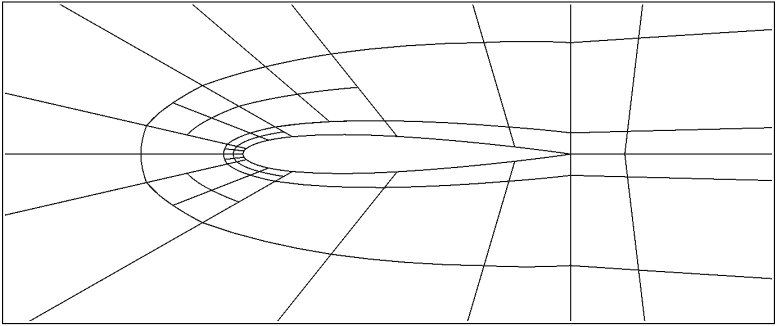
Inviscid flow: $p = 2$ mesh sequence



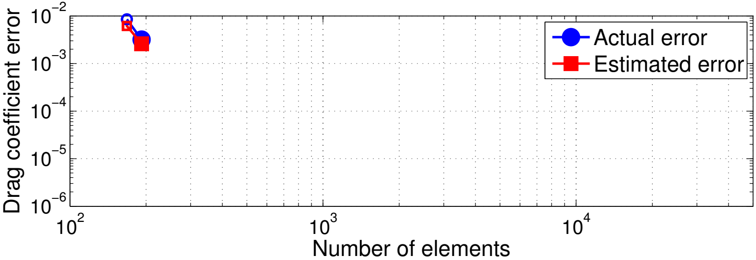


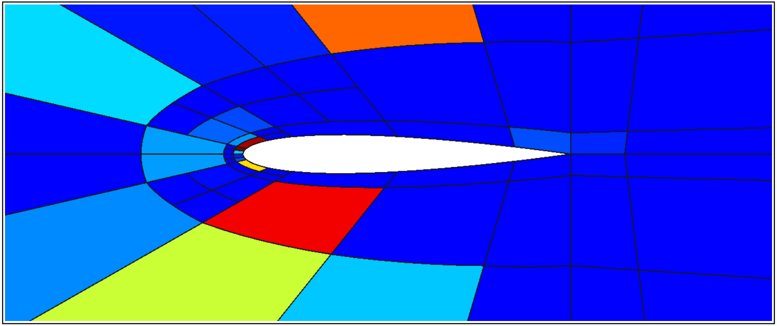
Inviscid flow: $p = 2$ mesh sequence



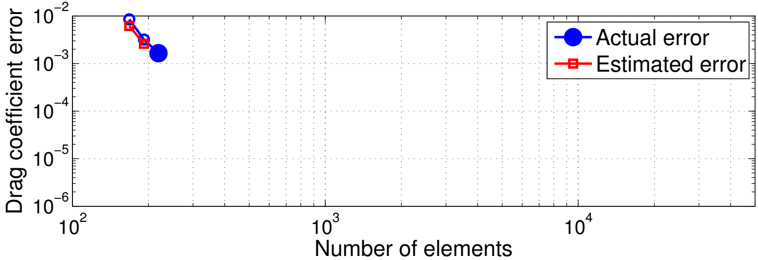


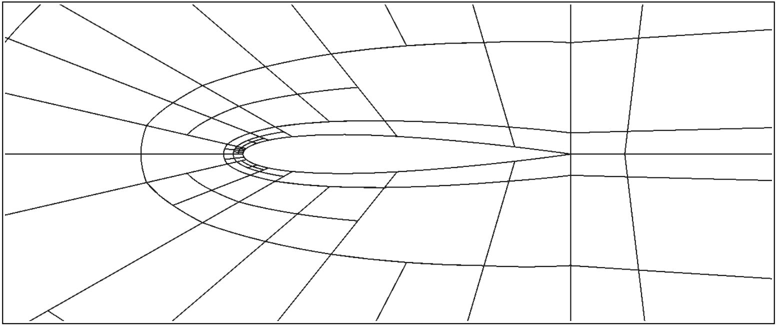
Inviscid flow: $p = 2$ mesh sequence



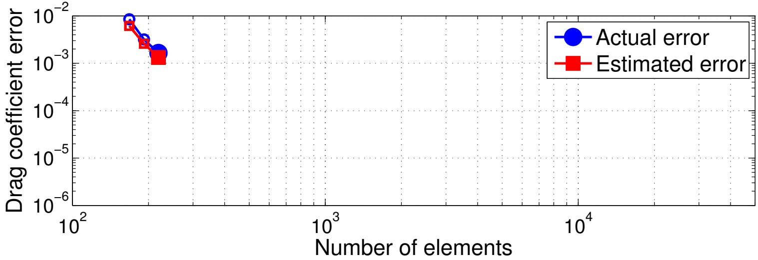


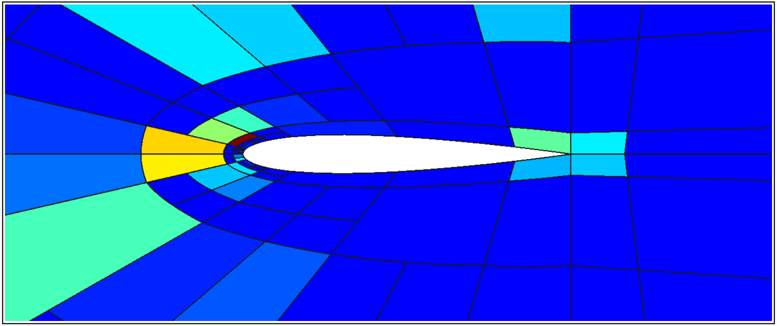
Inviscid flow: $p = 2$ mesh sequence



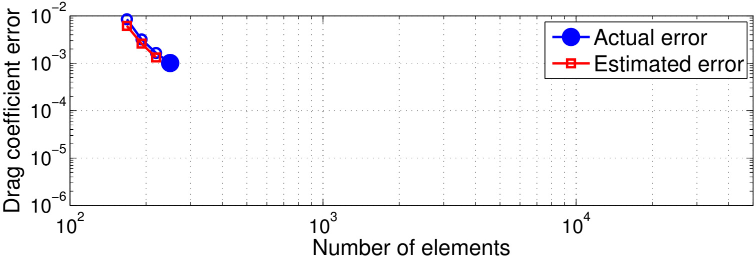


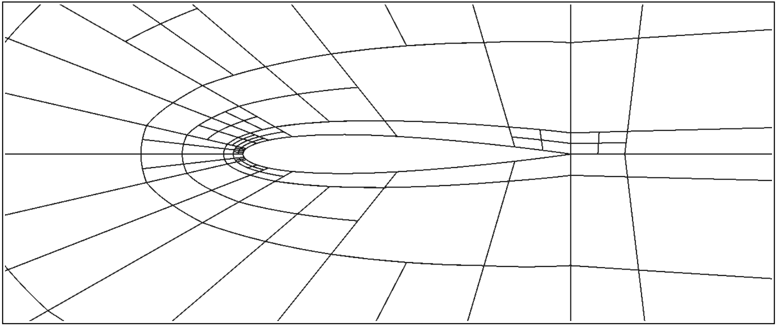
Inviscid flow: $p = 2$ mesh sequence



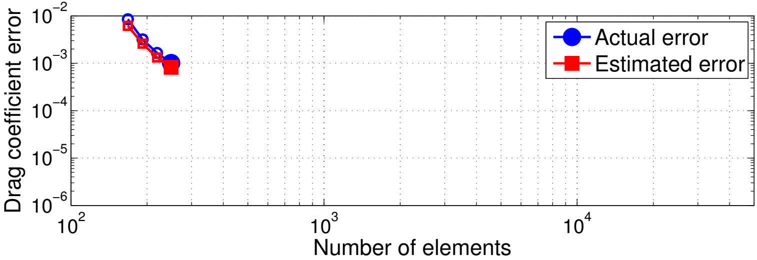


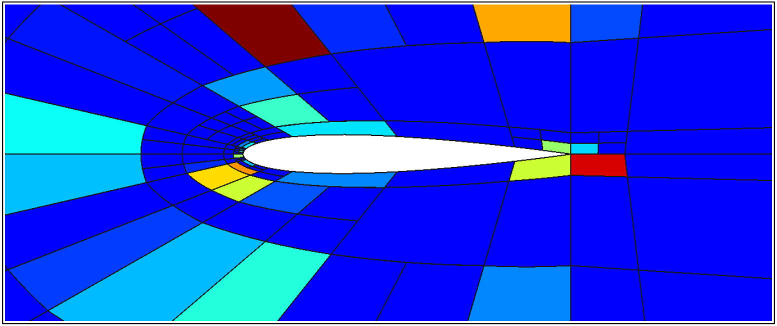
Inviscid flow: $p = 2$ mesh sequence



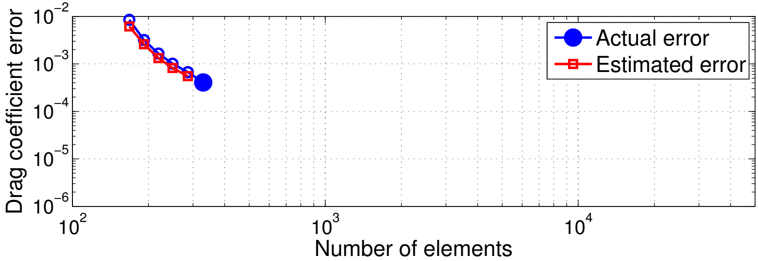


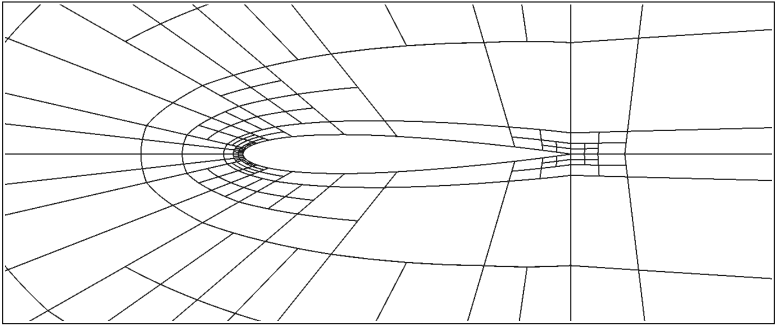
Inviscid flow: $p = 2$ mesh sequence



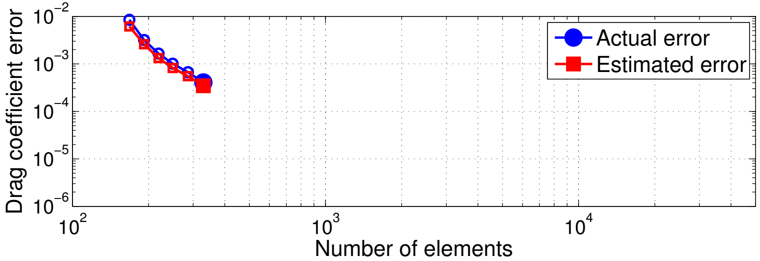


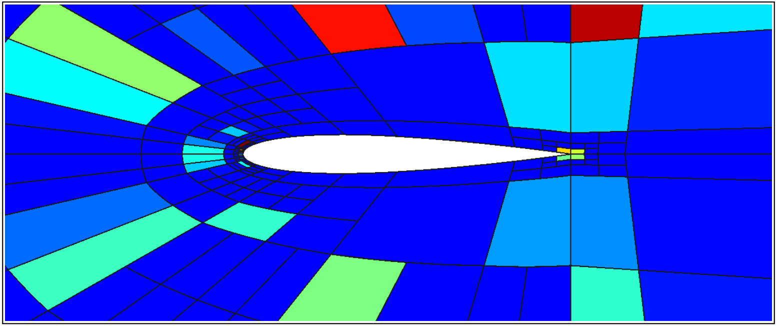
Inviscid flow: $p = 2$ mesh sequence



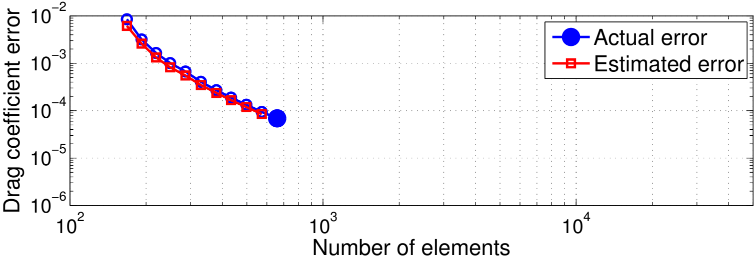


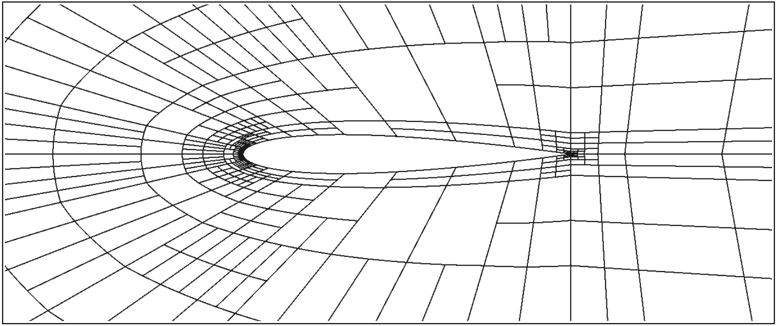
Inviscid flow: $p = 2$ mesh sequence



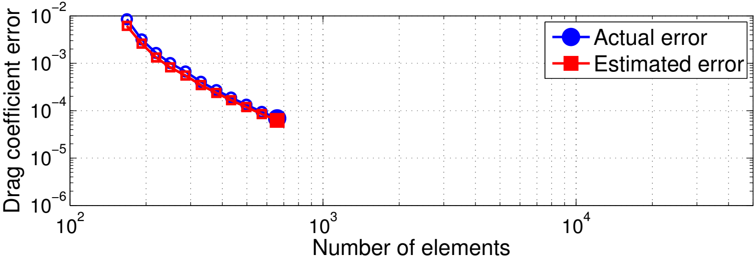


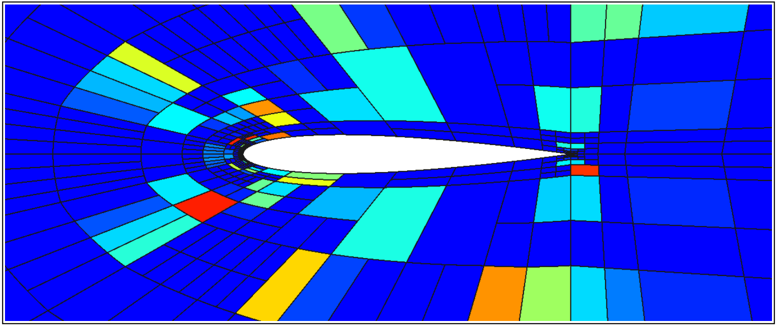
Inviscid flow: $p = 2$ mesh sequence



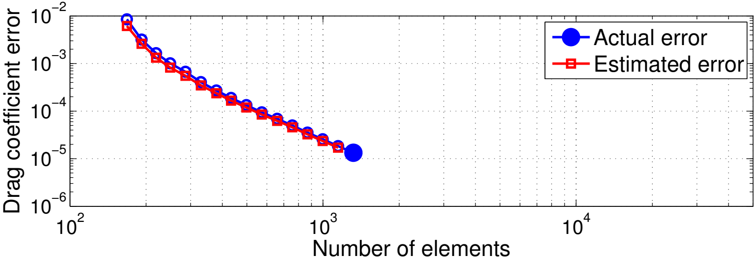


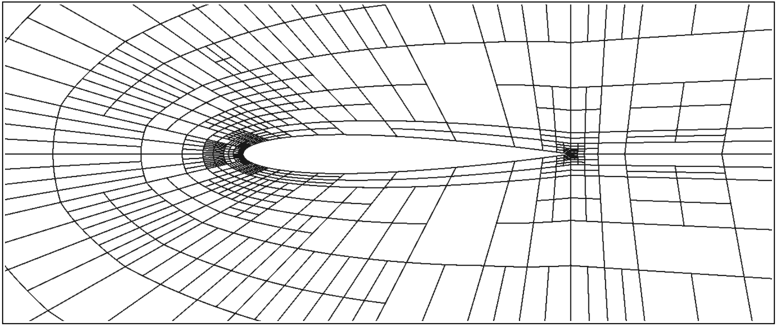
Inviscid flow: $p = 2$ mesh sequence



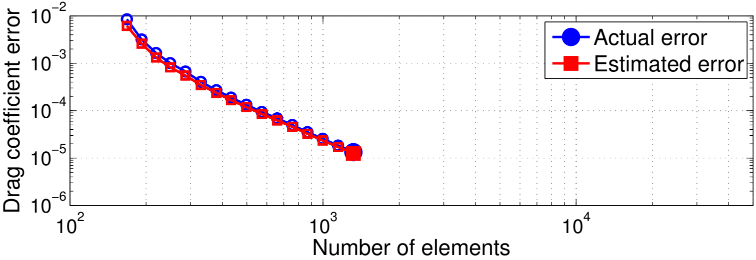


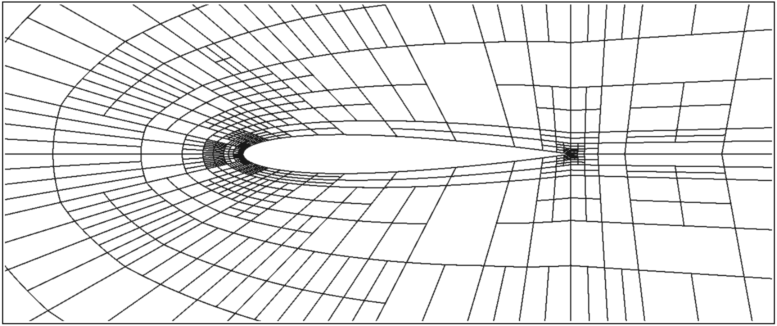
Inviscid flow: $p = 2$ mesh sequence



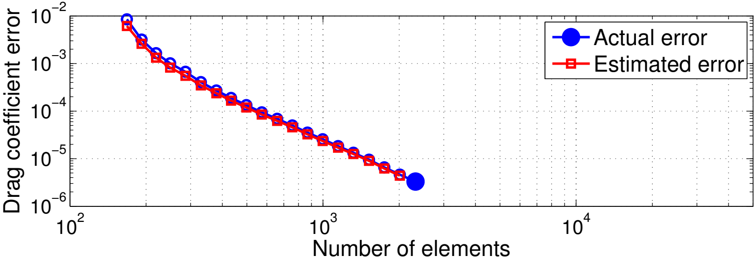


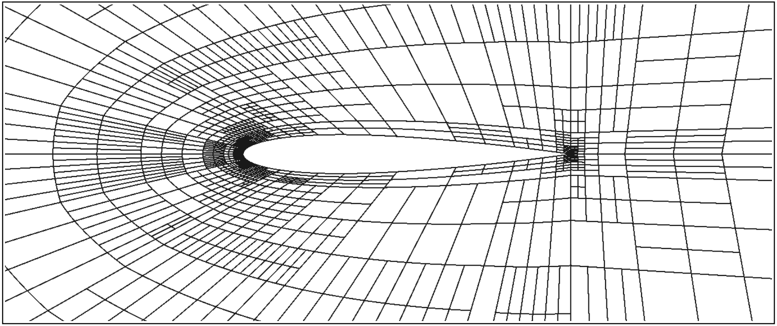
Inviscid flow: $p = 2$ mesh sequence



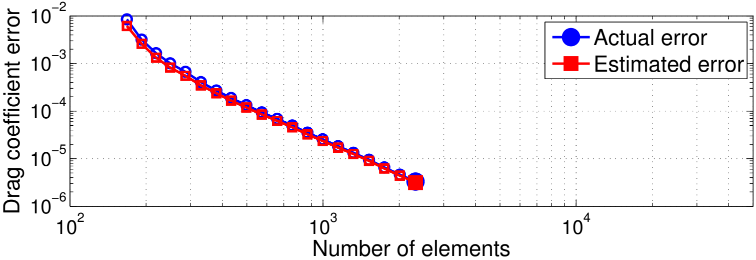


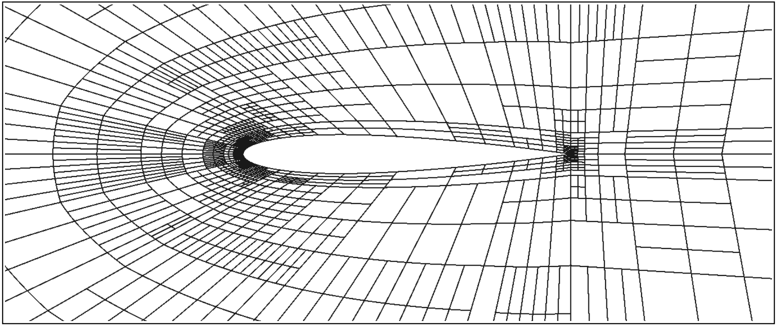
Inviscid flow: $p = 2$ mesh sequence



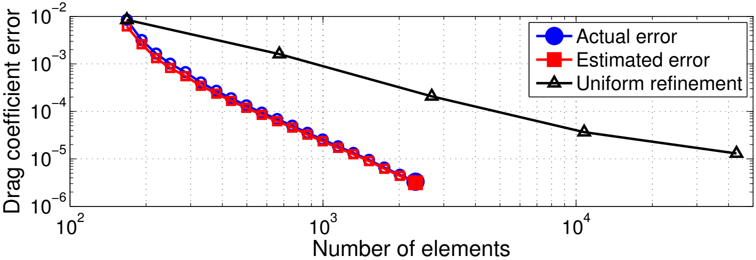


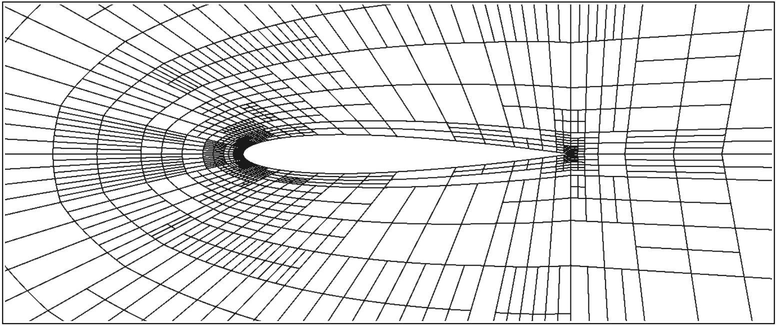
Inviscid flow: $p = 2$ mesh sequence



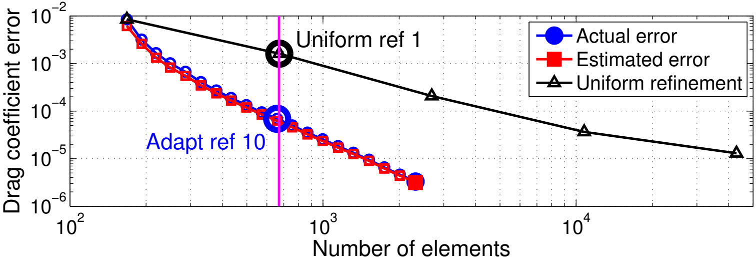


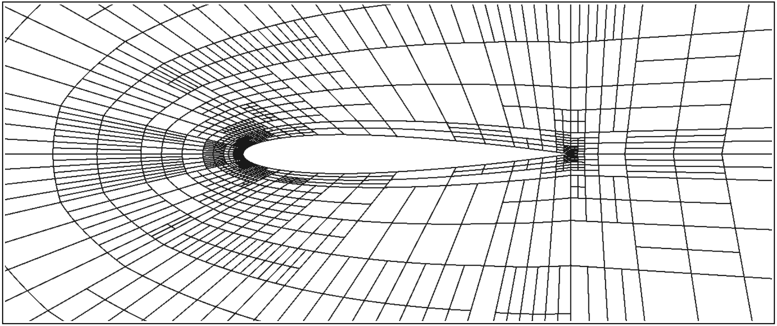
Inviscid flow: $p = 2$ mesh sequence





Inviscid flow: $p = 2$ mesh sequence

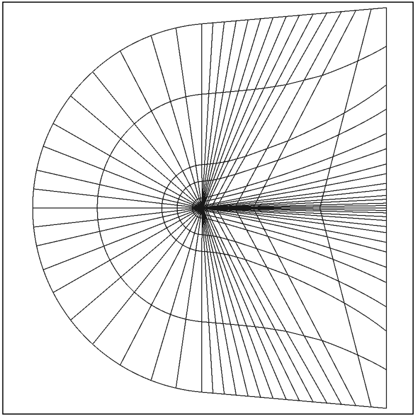




Inviscid flow: adapted/uniform comparison

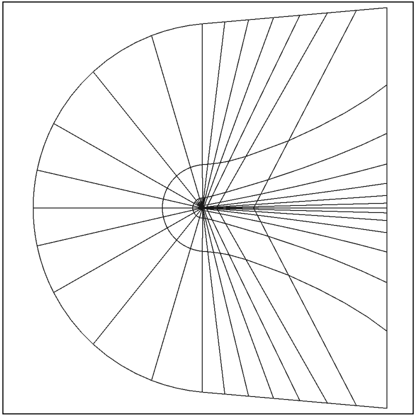
Farfield region

Uniform refinement 1



672 elements

Adapt refinement 10

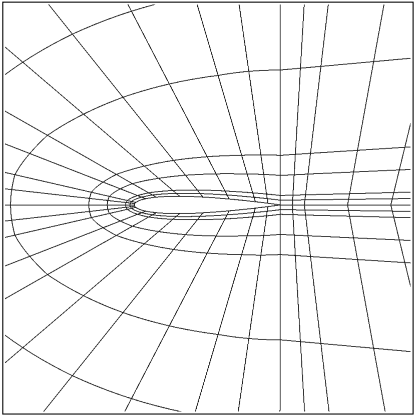


659 elements

Inviscid flow: adapted/uniform comparison

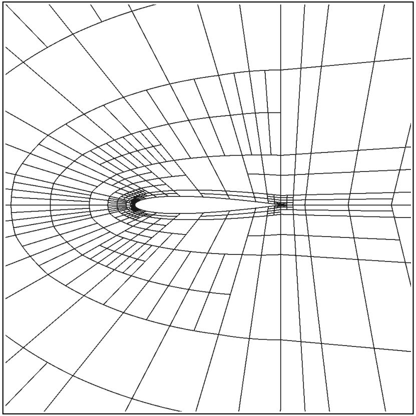
Near-field region

Uniform refinement 1



672 elements

Adapt refinement 10

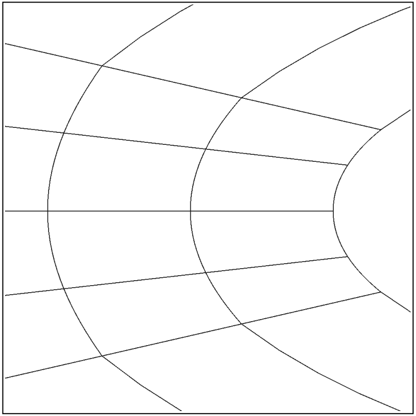


659 elements

Inviscid flow: adapted/uniform comparison

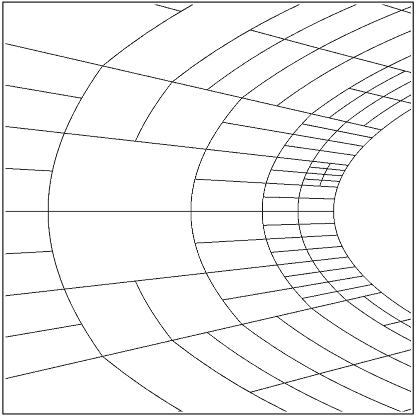
Leading edge

Uniform refinement 1



672 elements

Adapt refinement 10

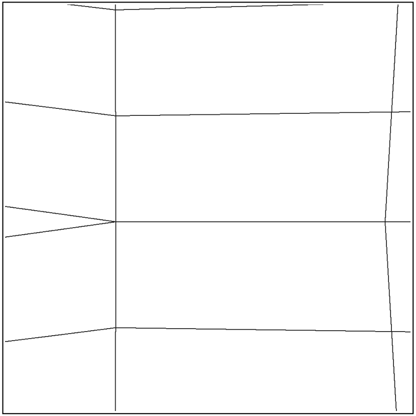


659 elements

Inviscid flow: adapted/uniform comparison

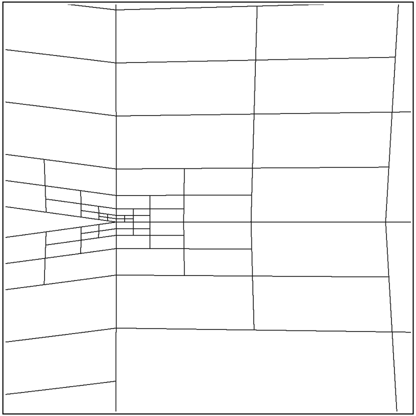
Trailing edge

Uniform refinement 1



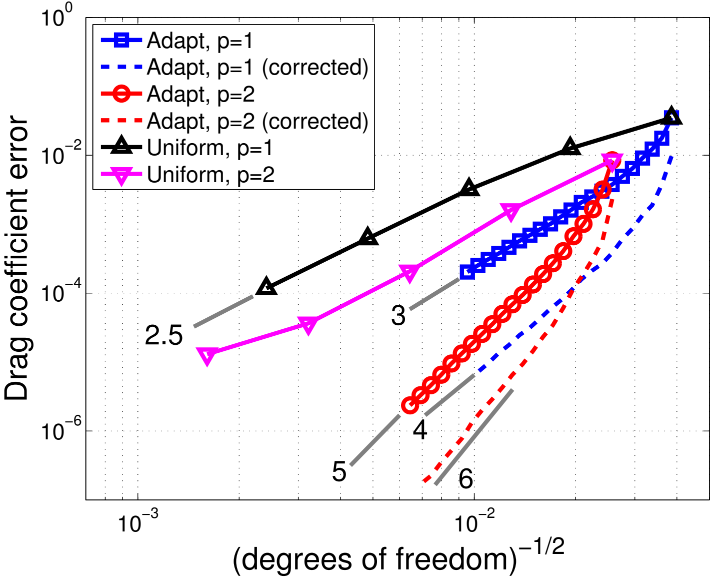
672 elements

Adapt refinement 10

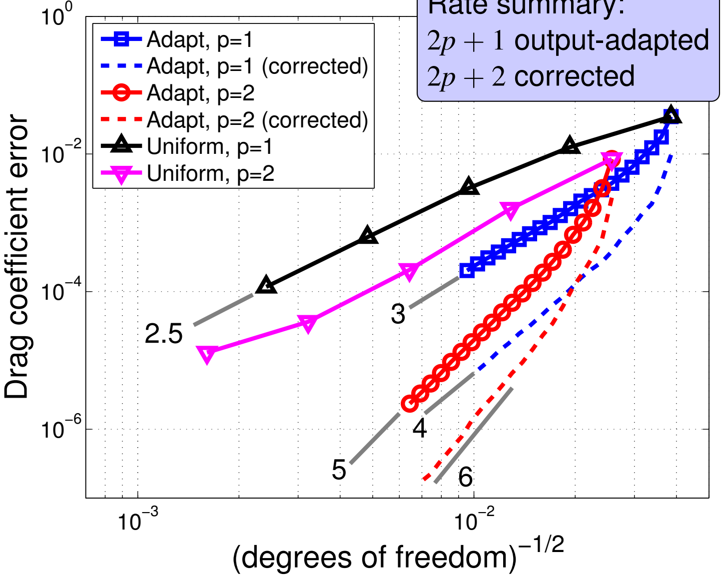


659 elements

Inviscid flow: drag convergence (exact ψ_h)

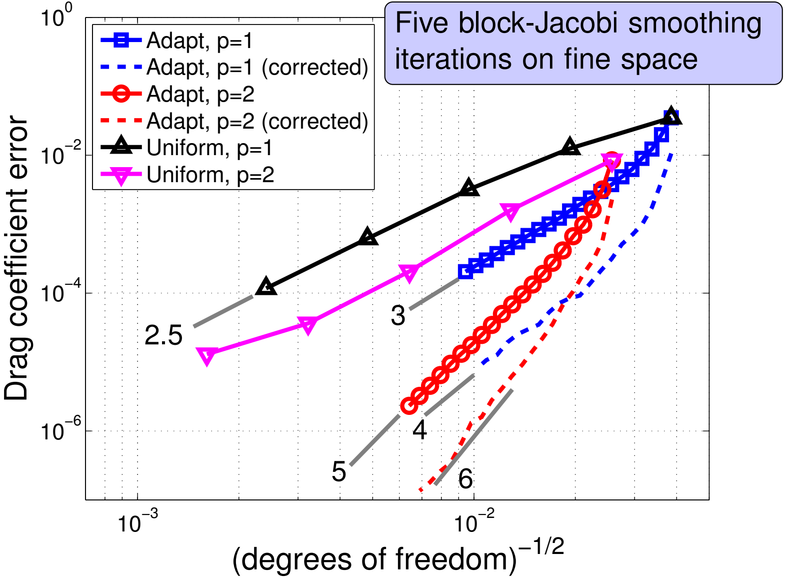


Inviscid flow: drag convergence (exact ψ_h)

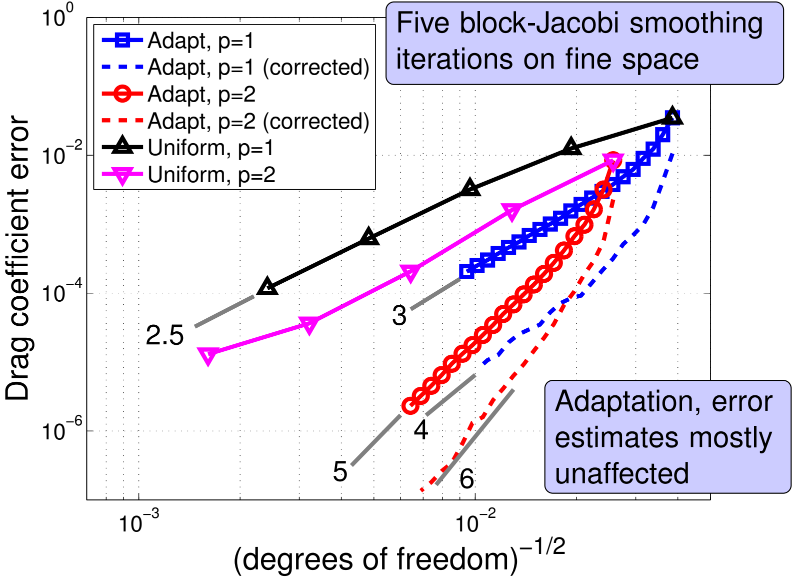


Rate summary:
$2p+1$ output-adapted
$2p+2$ corrected

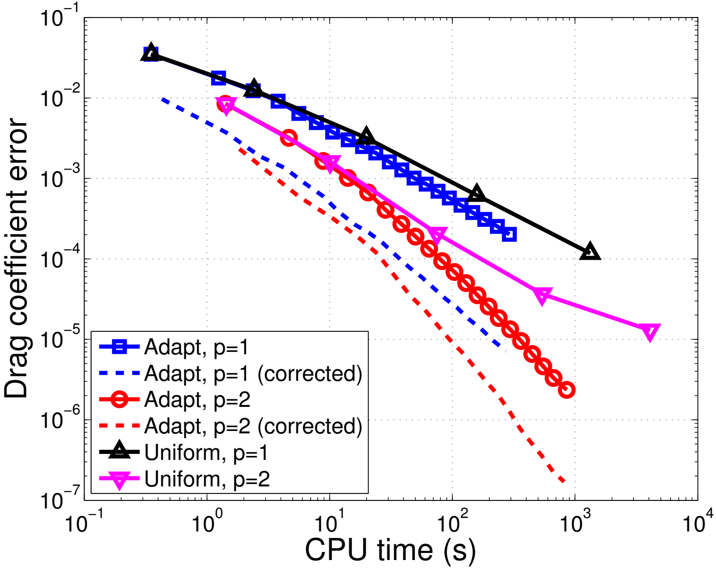
Inviscid flow: drag convergence (approx ψ_h)



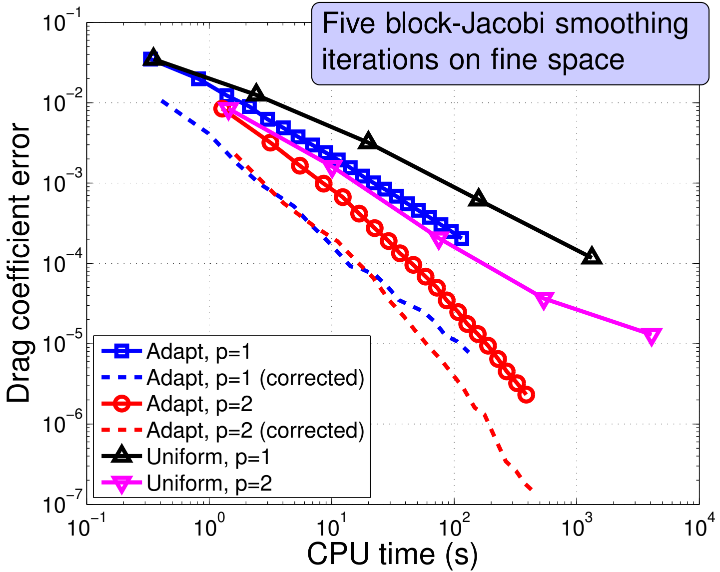
Inviscid flow: drag convergence (approx ψ_h)



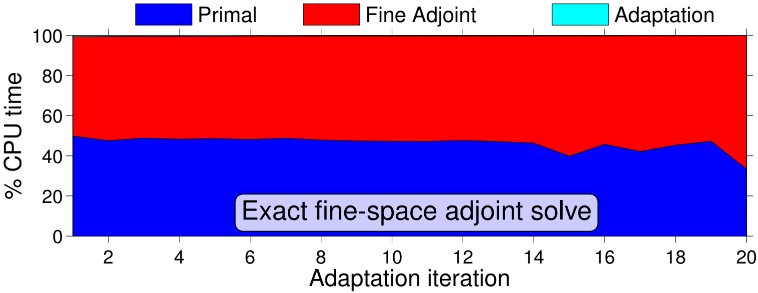
Inviscid flow: timing comparison (exact ψ_h)

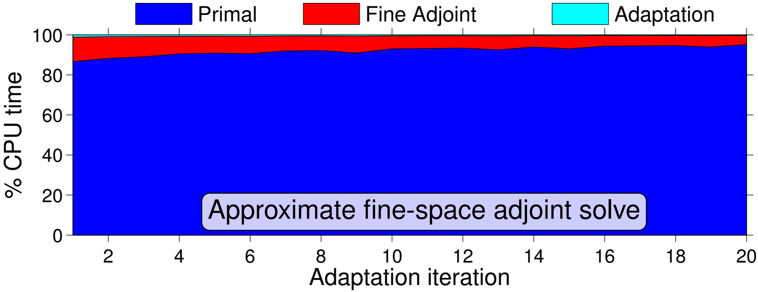


Inviscid flow: timing comparison (approx ψ_h)



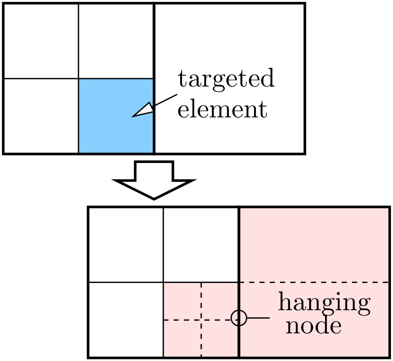
Inviscid flow: timing percentages



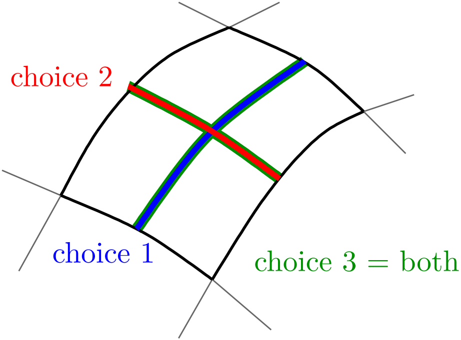


Incorporating anisotropy with hanging nodes

- Crucial for high-Reynolds number simulations, esp. in 3D
- Create anisotropy by cutting in only one direction
- Solve local sub-problems to determine impact of anisotropy directly on the output error



Hanging-node refinement



Discrete choices

Choosing the right cut [6: Ceze + Fidkowski, 2012]

- On an element, pick the cut i with the highest **merit**
- For each cut i define,

$$\mathrm{merit}(i) = \frac{\mathrm{benefit}(i)}{\mathrm{cost}(i)}$$

benefit(i)

- error addressed by cut i
- estimated using adjoint-weighted residual:

$$\mathrm{benefit}(i) \;=\; \sum_{\kappa_h \in \kappa_H} |\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H)|_{\kappa_h}$$

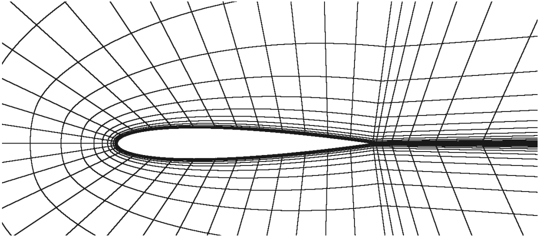
h denotes the error estimation fine space, e.g. $p+1$

cost(i)

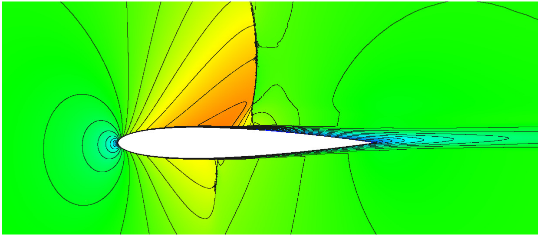
- degrees of freedom (may be too simple)
- number of nonzeros in Jacobian ($\sim$ cost of linear solve)

Transonic RANS airfoil

- NACA 0012
- $M = 0.8$, $\alpha = 1.25°$
- $Re = 10^5$
- 10% fixed fraction
- $q = 3$ curved geometry
- $p = 2$ solution approximation
- RANS-SA model
- Adapt on drag, lift
- Discrete-choice h cut optimization vs. isotropic

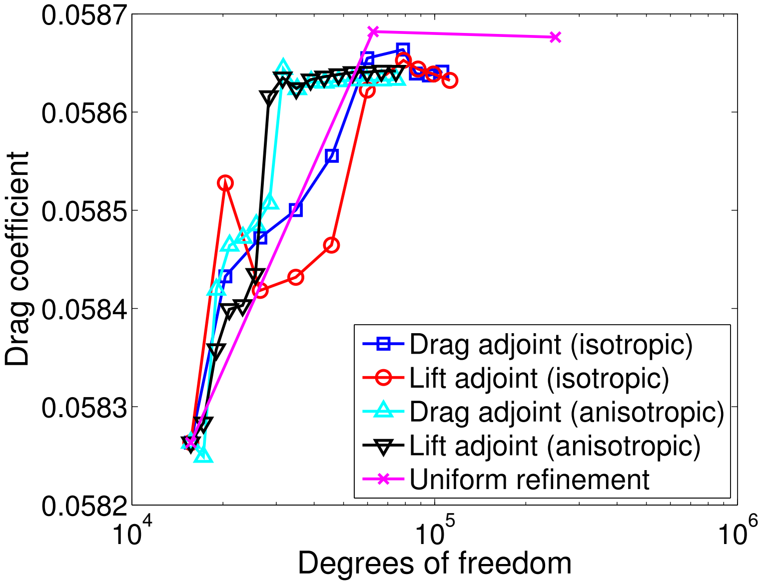


Initial mesh (1740 elements)

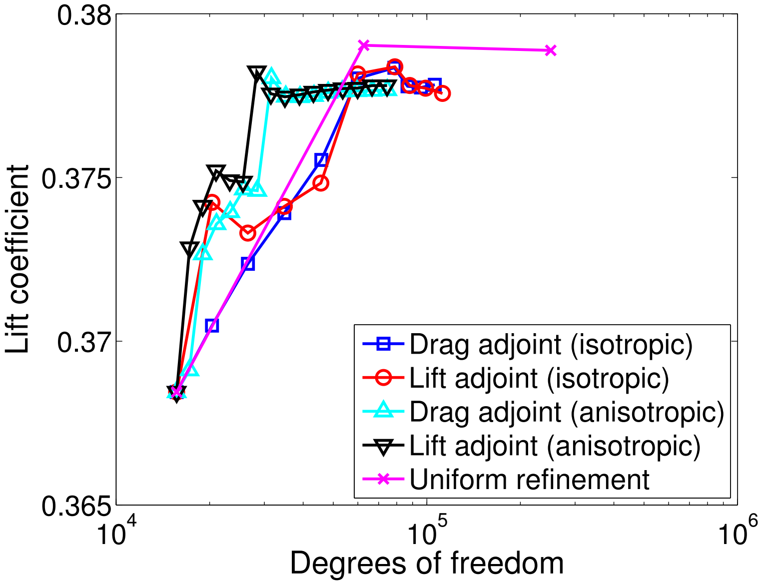


Mach number contours

Transonic RANS airfoil: output convergence

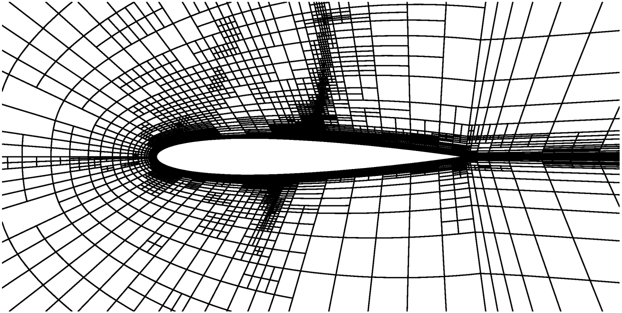


Transonic RANS airfoil: output convergence

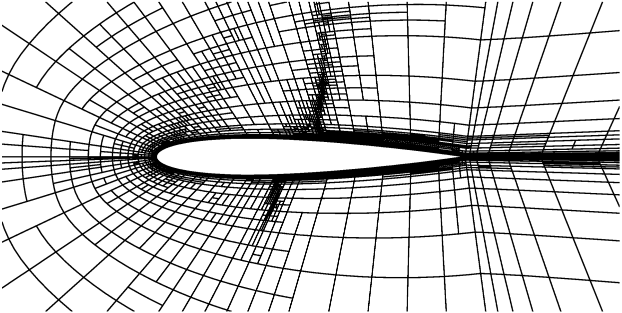


Transonic RANS airfoil: drag-adapted meshes

- Isotropic
- Adapt iter 6
- 8736 elems

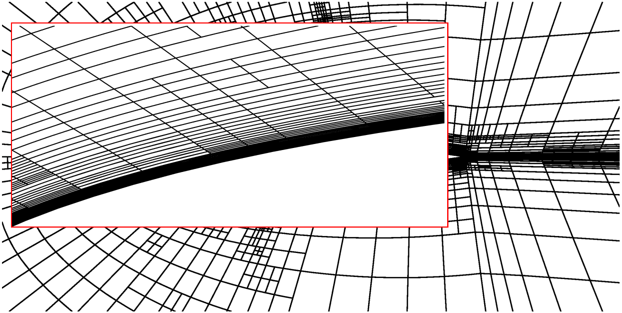


- Anisotropic
- Adapt iter 10
- 4816 elems

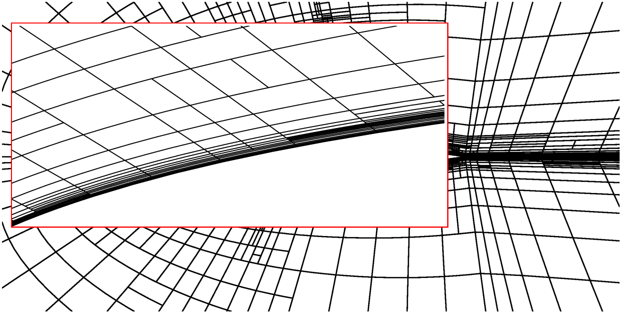


Transonic RANS airfoil: drag-adapted meshes

- Isotropic
- Adapt iter 6
- 8736 elems

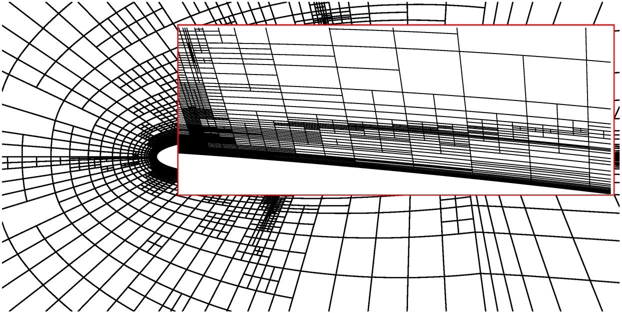


- Anisotropic
- Adapt iter 10
- 4816 elems

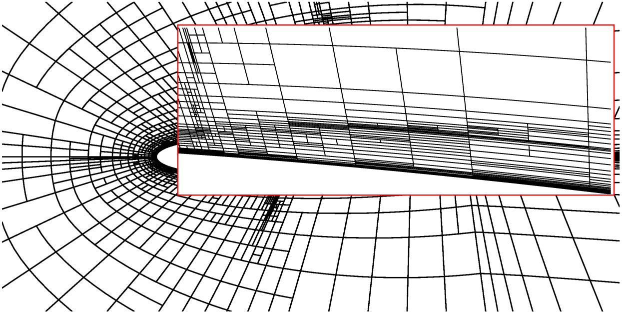


Transonic RANS airfoil: drag-adapted meshes

- Isotropic
- Adapt iter 6
- 8736 elems



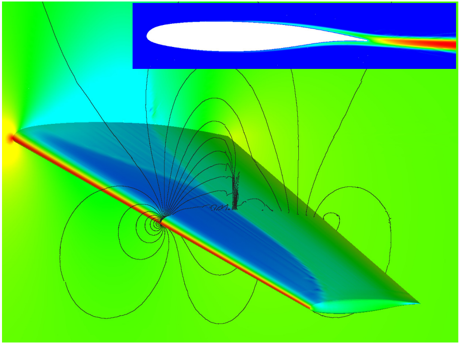
- Anisotropic
- Adapt iter 10
- 4816 elems



Transonic RANS flow over a wing

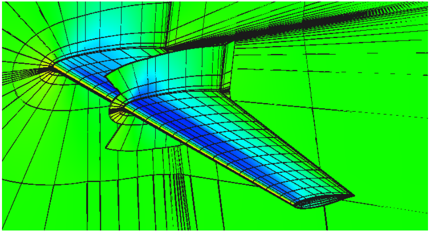
DPW III wing-alone case: $M_\infty = 0.76, Re = 5 \times 10^6$

- Initial mesh: cubic hex elements generated by agglomeration of linear multiblock meshes (first element $y^+ \approx 1$)
- Artificial viscosity shock capturing
- Spalart-Allmaras turbulence model with negative $\tilde{\nu}$ modification [1: Allmaras et al, 2012]
- Drag-adaptive simulation using hp discrete choice algorithm [7: Ceze + Fidkowski, 2013]

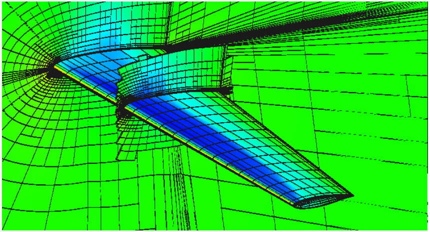


Contours of c_p and $\tilde{\nu}$

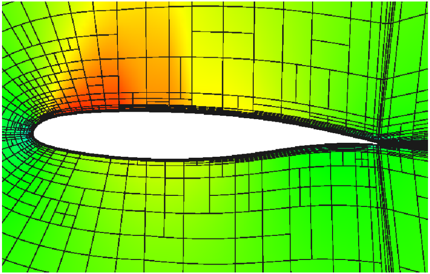
DPW wing: adapted meshes



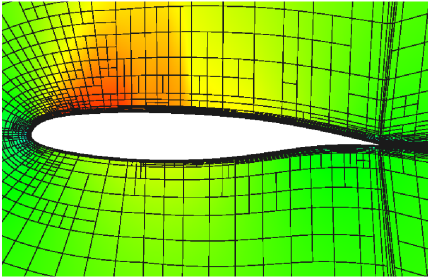
Original mesh, with c_p contours



7^{th} drag-adapted mesh

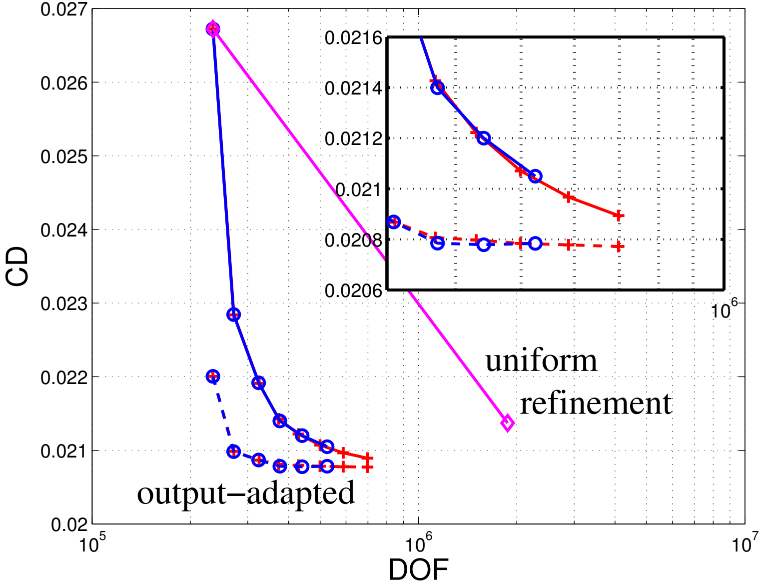


Mach/mesh using DOF cost

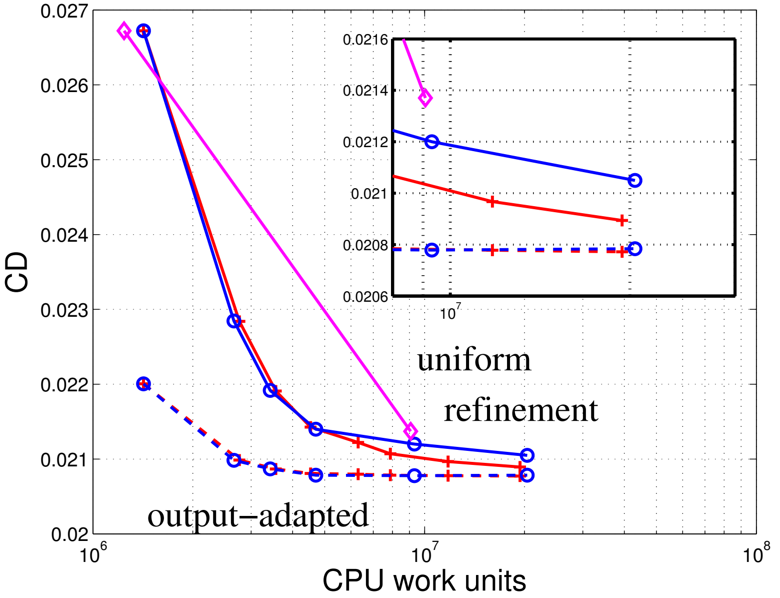


Mach/mesh using non-zero entries cost

DPW wing: comparison to uniform refinement



DPW wing: comparison to uniform refinement



Outline

Mesh adaptation using a metric field

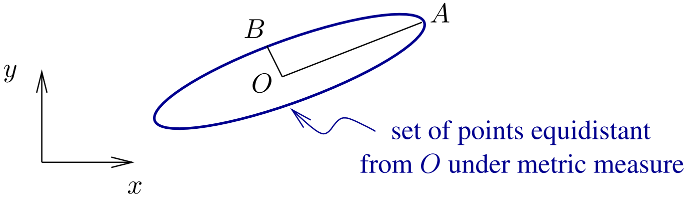
- Unstructured meshes offer more *geometric* and *adaptive* flexibility over structured ones
- Resolution information: size and shape of an element
- This can be encoded in a *metric field* [5: Borouchaki, 1995] [12: Pennec, 2006] over the domain
- We are interested in an adaptive method where the mesh is *regenerated* at each iteration using the current mesh and information from the solution
- Key ingredients:
  1. Metric-conforming mesh generator
  2. Solution-based metric specification

A Riemannian metric field

$\mathcal{M}(\vec{x}) \in \mathbb{R}^{d\times d}$

A symmetric positive definite (SPD) tensor field that provides a "yardstick" for measuring distances in different directions

$$\text{metric distance between } \vec{x} \text{ and } \vec{x}+\delta\vec{x} : \ \delta\ell = \sqrt{\delta\vec{x}^T\mathcal{M}\delta\vec{x}}$$



- Eigenvectors of $\mathcal{M}$ are principal stretching directions
- Eigenvalues: $\lambda_i = 1/h_i^2$; h_i is the "stretching magnitude": the distance along the eigenvector for unit metric measure

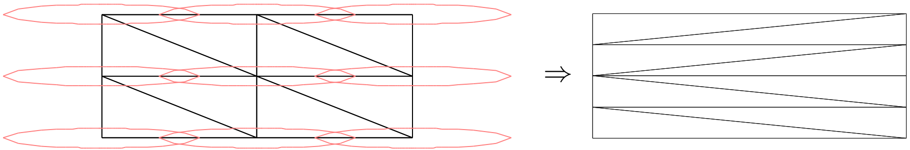
Mesh-conforming mesh generation

Idea

Make mesh in which each edge has the same metric length

$$\text{metric distance from } A \text{ to } B\text{: } \ell_{AB} = \int_A^B d\ell = \int_A^B \sqrt{d\vec{x}^T \mathcal{M} d\vec{x}}$$

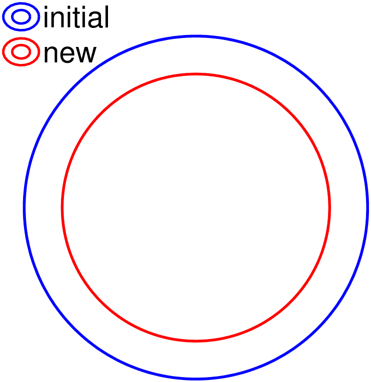
- e.g. BAMG = Bi-dimensional Anisotropic Mesh Generator [5: Borouchaki, 1995]
- Input: background mesh and desired metric at nodes
- Output: metric-conforming mesh



Affine-invariant metric modification

Need a systematic way to alter $\mathcal{M}$: must keep SPD

- $\mathcal{M}_0$ = current metric
- $\mathcal{M}$ = new metric = $\boxed{\mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}}$
- $\mathcal{S} \in \mathbb{R}^{d\times d}$ = metric step matrix (symmetric)



Example ($\mathcal{M}_0 = \mathcal{I}$):

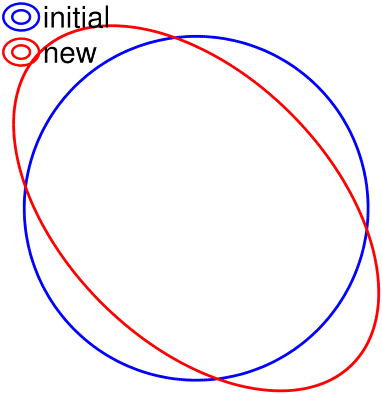
$$\mathcal{S} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Positive values on diagonal produce a contraction

Affine-invariant metric modification

Need a systematic way to alter $\mathcal{M}$: must keep SPD

- $\mathcal{M}_0$ = current metric
- $\mathcal{M}$ = new metric = $\boxed{\mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}}$
- $\mathcal{S} \in \mathbb{R}^{d \times d}$ = metric step matrix (symmetric)



Example ($\mathcal{M}_0 = \mathcal{I}$):

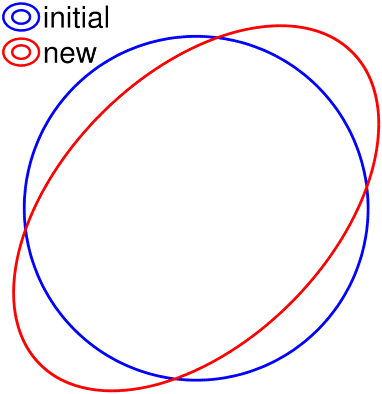
$$\mathcal{S} = \begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix}$$

Positive values off diagonal produce negative shear

Affine-invariant metric modification

Need a systematic way to alter $\mathcal{M}$: must keep SPD

- $\mathcal{M}_0$ = current metric
- $\mathcal{M}$ = new metric = $\boxed{\mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}}$
- $\mathcal{S} \in \mathbb{R}^{d \times d}$ = metric step matrix (symmetric)



Example ($\mathcal{M}_0 = \mathcal{I}$):

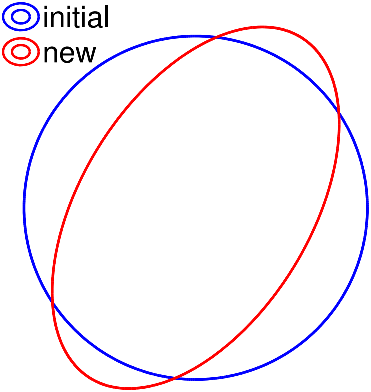
$$\mathcal{S} = \begin{bmatrix} 0 & -0.5 \\ -0.5 & 0 \end{bmatrix}$$

Negative values off diagonal produce positive shear

Affine-invariant metric modification

Need a systematic way to alter $\mathcal{M}$: must keep SPD

- $\mathcal{M}_0$ = current metric
- $\mathcal{M}$ = new metric = $\boxed{\mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}}$
- $\mathcal{S} \in \mathbb{R}^{d\times d}$ = metric step matrix (symmetric)



Example ($\mathcal{M}_0 = \mathcal{I}$):

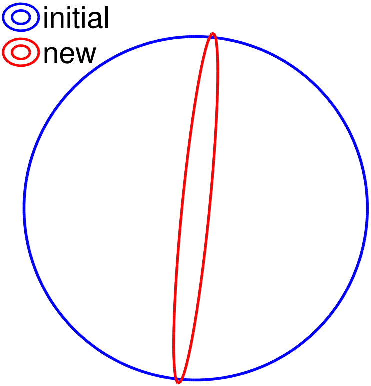
$$\mathcal{S} = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0 \end{bmatrix}$$

Combination of positive shear and contraction in x

Affine-invariant metric modification

Need a systematic way to alter $\mathcal{M}$: must keep SPD

- $\mathcal{M}_0$ = current metric
- $\mathcal{M}$ = new metric = $\boxed{\mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}}$
- $\mathcal{S} \in \mathbb{R}^{d \times d}$ = metric step matrix (symmetric)



Example ($\mathcal{M}_0 = \mathcal{I}$):

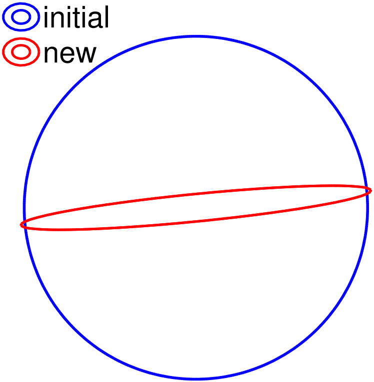
$$\mathcal{S} = \begin{bmatrix} 5 & -0.5 \\ -0.5 & 0 \end{bmatrix}$$

Combination of positive shear and a lot of contraction in x

Affine-invariant metric modification

Need a systematic way to alter $\mathcal{M}$: must keep SPD

- $\mathcal{M}_0$ = current metric
- $\mathcal{M}$ = new metric = $\boxed{\mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}}$
- $\mathcal{S} \in \mathbb{R}^{d \times d}$ = metric step matrix (symmetric)



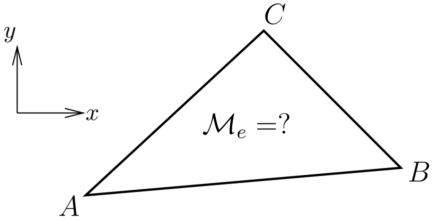
Example ($\mathcal{M}_0 = \mathcal{I}$):

$$\mathcal{S} = \begin{bmatrix} 0 & -0.5 \\ -0.5 & 5 \end{bmatrix}$$

Combination of positive shear and a lot of contraction in y

Mesh-implied metric

- We need to "back out" a metric given a mesh, since we will be prescribing *changes* to the metric
- Calculate elemental metric, $\mathcal{M}_e$, by enforcing that each edge length is of unit measure under the metric



$$
\begin{aligned}
\Delta\vec{x}_{AB}^T \mathcal{M}_e \Delta\vec{x}_{AB} &= 1 \\
\Delta\vec{x}_{BC}^T \mathcal{M}_e \Delta\vec{x}_{BC} &= 1 \\
\Delta\vec{x}_{CA}^T \mathcal{M}_e \Delta\vec{x}_{CA} &= 1
\end{aligned}
$$

- This gives 3 equations for the three independent unknowns in the symmetric matrix representation of $\mathcal{M}_e$
- Note: the elemental metric can be mapped to nodes via an affine-invariant average [12: Pennec et al, 2006]

A mesh optimization algorithm [16: Yano, 2012]

- Given: current mesh, primal and adjoint solutions
- Determine: metric step matrix, $\mathcal{S}_v$, at each mesh vertex, v, that produces a mesh with the smallest output error at a fixed solution cost
- Key ingredients
  1. Error convergence model: $\mathcal{S}_v \rightarrow$ output error
  2. Cost model: $\mathcal{S}_v \rightarrow$ solution cost
  3. Iterative algorithm that equidistributes the marginal error-to-cost ratio
- Expect multiple iterations of optimization until error "bottoms out" at a fixed cost; can then increase allowable cost to further reduce error

Error convergence model

- $\mathcal{E}_{e0}$ = current output error indicator on element e (from AWR)
- $\mathcal{S}_e$ = proposed metric step matrix on element e
- Model for error after metric modification with $\mathcal{S}_e$:

$$\boxed{\mathcal{E}_e = \mathcal{E}_{e0} \exp\left[\mathrm{tr}(R_e \mathcal{S}_e)\right]}$$

- R_e = error convergence rate tensor (identified by sampling)
- Note, this is a generalization to anisotropic shape changes of the more familiar isotropic model,

$$\mathcal{E}_e = \mathcal{E}_{e0} \left(\frac{h}{h_0}\right)^r = \mathcal{E}_{e0} \exp\left[r \log(h/h_0)\right]$$

- Sum over elements to get the total error on the mesh,

$$\boxed{\mathcal{E} = \sum_e \mathcal{E}_e}$$

Cost model

cost = degrees of freedom (`dof`) in solution approximation

- Assume p = approximation order = same for all elements
- $\mathcal{C}_{e0}$ = current cost on element e, e.g. $(p+1)(p+2)/2$
- New cost after application of step matrix $\mathcal{S}_e$,

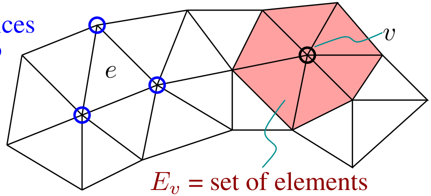
$$\boxed{\mathcal{C}_e = C_{e0} \underbrace{\exp\left[\frac{1}{2}\mathrm{tr}(\mathcal{S}_e)\right]}_{\mathrm{Area}_0/\mathrm{Area}}}$$

- Note, the cost is just scaled by $\mathrm{Area}_0/\mathrm{Area}$ = # new elements occupying the original area of element e
- Sum over elements to get the total cost on the mesh,

$$\boxed{\mathcal{C} = \sum_e \mathcal{C}_e}$$

From elements to vertices

$$\mathcal{S}_e = \frac{1}{|V_e|} \sum_{v \in V_e} \mathcal{S}_v$$



Error

$$\mathcal{E} = \sum_e \mathcal{E}_e$$

$$\frac{\partial \mathcal{E}}{\partial \mathcal{S}_v} = \sum_{e \in E_v} \underbrace{\frac{\partial \mathcal{E}_e}{\partial \mathcal{S}_e}}_{\mathcal{E}_e R_e} \underbrace{\frac{\partial \mathcal{S}_e}{\partial \mathcal{S}_v}}_{1/|V_e|},$$

Cost

$$\mathcal{C} = \sum_e \mathcal{C}_e$$

$$\frac{\partial \mathcal{C}}{\partial \mathcal{S}_v} = \sum_{e \in E_v} \underbrace{\frac{\partial \mathcal{C}_e}{\partial \mathcal{S}_e}}_{\mathcal{C}_e \frac{1}{2}\mathcal{I}} \underbrace{\frac{\partial \mathcal{S}_e}{\partial \mathcal{S}_v}}_{1/|V_e|},$$

Optimization algorithm

Given:

- $\mathcal{M}_0$ = mesh-implied metric (elements $\rightarrow$ nodes)
- $\mathcal{E}_{e0}$ = error indicators on elements (AWR)
- R_e = error rate tensor (sampling)

Calculate:

$\mathcal{S}_v$ = step matrix at vertices that minimizes error at a fixed cost

- We separate $\mathcal{S}_v$ into size (trace) and shape (trace-free) contributions,

$$\mathcal{S}_v = s_v \mathcal{I} + \widetilde{\mathcal{S}}_v$$

- Derivatives of the error with respect to s_v and $\widetilde{\mathcal{S}}_v$ are

$$\frac{\partial \mathcal{E}}{\partial s_v} = \mathrm{tr}\left(\frac{\partial \mathcal{E}}{\partial \mathcal{S}_v}\right), \qquad \frac{\partial \mathcal{E}}{\partial \widetilde{\mathcal{S}}_v} = \frac{\partial \mathcal{E}}{\partial \mathcal{S}_v} - \frac{\partial \mathcal{E}}{\partial s_v}\frac{\mathcal{I}}{d}$$

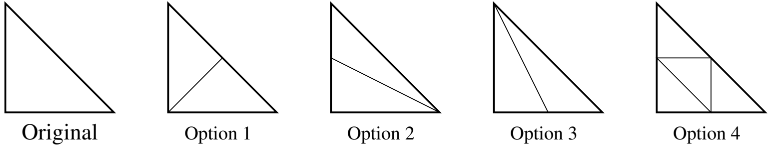
Optimization algorithm (continued)

Use an iterative approach [16: Yano, 2012]

- Initialize $\mathcal{S}_v = 0, \forall v$, set $\delta s = s_{\max}/n_{\text{step}} = 2\log 2/20$
- Loop $i = 1 : n_{\text{step}}$,
  1. $\mathcal{S}_v \to \mathcal{S}_e \to \frac{\partial \mathcal{E}_e}{\partial \mathcal{S}_e}, \frac{\partial \mathcal{C}_e}{\partial \mathcal{S}_e} \to$ linearizations w.r.t s_v and $\widetilde{\mathcal{S}}_v$
  2. Define $\lambda_v = \frac{\partial \mathcal{E}/\partial s_v}{\partial \mathcal{C}/\partial s_v}$ = marginal error to marginal cost ratio of mesh refinement
     - Refine 30% of vertices with the largest $|\lambda_v|$: $\mathcal{S}_v = \mathcal{S}_v + \delta s \mathcal{I}$
     - Coarsen 30% of the vertices with the smallest $|\lambda_v|$: $\mathcal{S}_v = \mathcal{S}_v - \delta s \mathcal{I}$
  3. Update the trace-free part of S_v, $S_v = S_v + \delta s (\partial \mathcal{E}/\partial \widetilde{\mathcal{S}}_v)/(\partial \mathcal{E}/\partial s_v)$
  4. Rescale $S_v \to S_v + \beta \mathcal{I}$, to meet total cost constraint via $\beta = \frac{2}{d} \log \frac{\mathcal{C}_{\text{target}}}{\mathcal{C}}$, where $\mathcal{C}_{\text{target}}$ is the target cost

Error sampling to obtain R_e

- An a-posteriori data-driven approach
- Idea: cut an element in different ways, measure change in error indicator, and fit model via least-squares regression



- Determine entries of R_e (symmetric) that minimize misfit between the model and observed errors for reach refinement option i

$$\text{misfit} = \sum_i \left[\log \frac{\mathcal{E}_{ei}}{\mathcal{E}_{e0}} - \text{tr}(R_e \mathcal{S}_{ei}) \right]^2$$

$\mathcal{E}_{ei}$ = output error for refinement option i
$\mathcal{S}_{ei}$ = average metric step matrix for refinement option i

Estimating errors after refinement

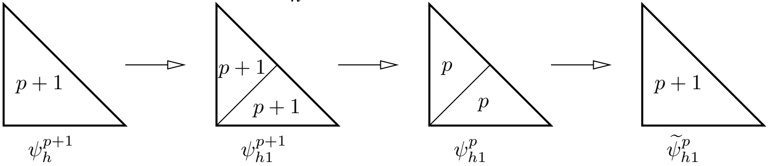
- Error left after refining via option i,

$$\mathcal{E}_{ei} = \mathcal{E}_{e0} - \Delta\mathcal{E}_{ei}$$

- $\Delta\mathcal{E}_{ei}$ = error between coarse solution and refinement option i

$$\Delta\mathcal{E}_{ei} \equiv |\mathcal{R}_h^{p+1}(\mathbf{u}_h^p, \widetilde{\boldsymbol{\psi}}_{hi}^p\big|_{\Omega_e})|$$

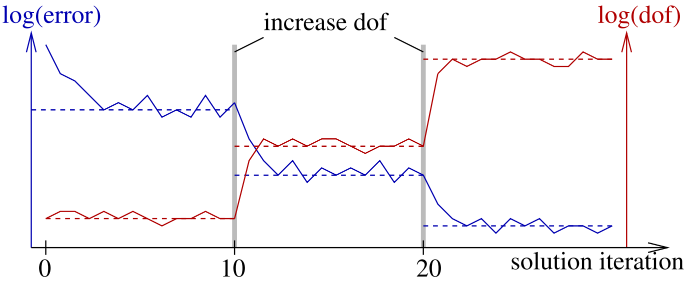
- $\widetilde{\boldsymbol{\psi}}_{hi}^p$ = adjoint after refining via option i, obtained by projecting the fine-space adjoint ψ_h^{p+1}



Note, can use pre-calculated projection matrices for each refinement option

Combining adaptation and optimization

1. Start with a coarse mesh at a certain cost = `dof`
2. Run multiple (~ 10) mesh optimization iterations at fixed cost
   - Each iteration requires primal and adjoint solves
   - Solves are quick since starting from good initial guesses
   - Error will drop, then stagnate/oscillate
   - Use results from final run or average of last few runs

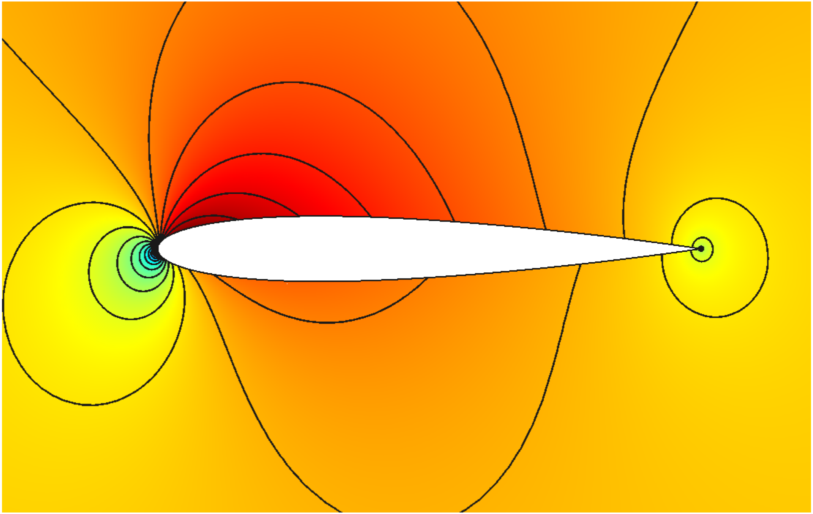


3. Increase `dof` cost by a prescribed factor if need more accuracy and can afford more cost; return to step 2

Example: NACA 0012 in inviscid flow

Euler equations, $M_\infty = 0.5, \alpha = 2^\circ, \gamma = 1.4$, output = drag

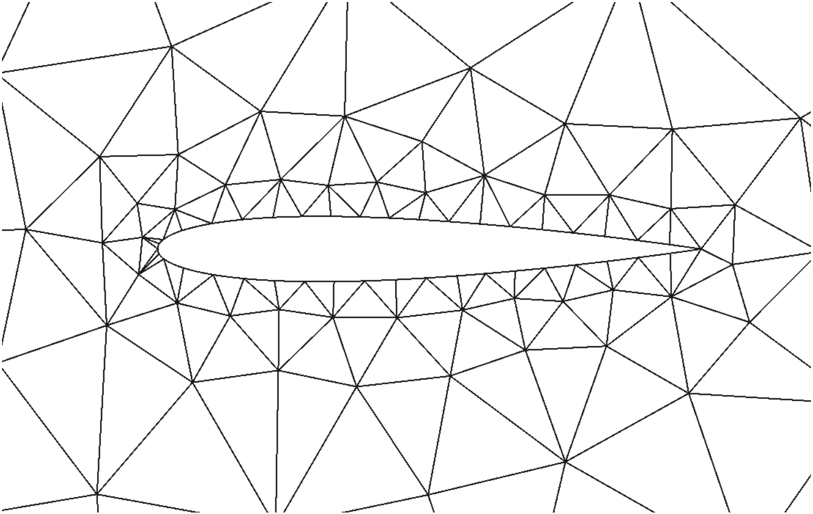
Mach number contours



Example: NACA 0012 in inviscid flow

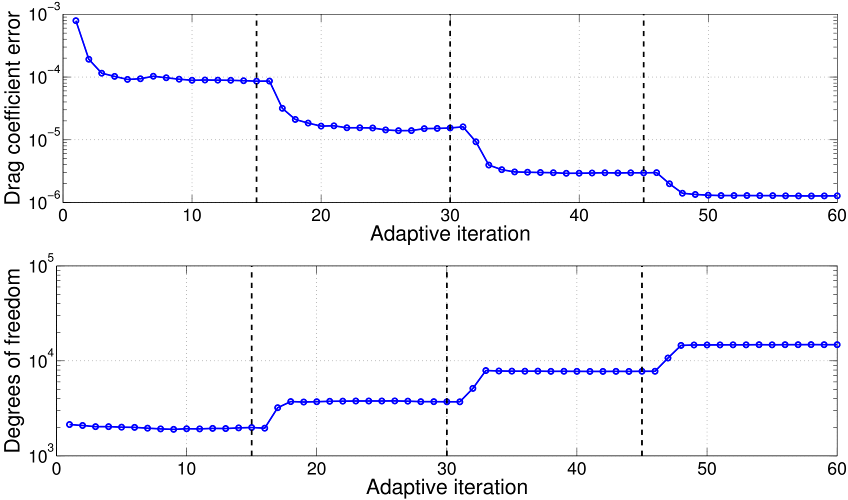
Euler equations, $M_\infty = 0.5, \alpha = 2^\circ, \gamma = 1.4$, output = drag

Initial mesh: 356 triangles, farfield @$2000c$



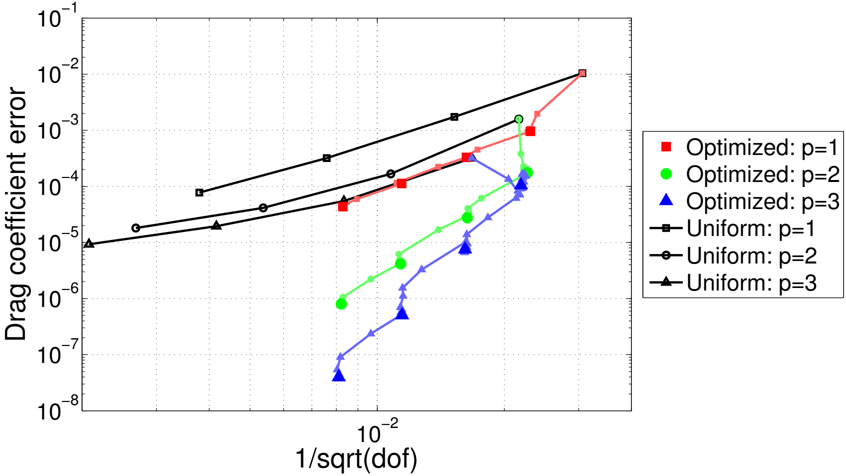
NACA 0012 in inviscid flow: sample run

$p = 2$, 15 optimization iterations at each `dof`

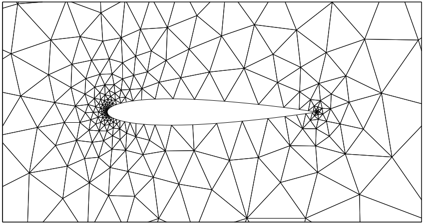


NACA 0012 in inviscid flow: output convergence

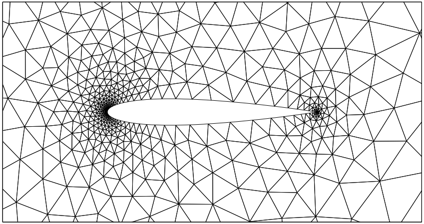
Compare to uniform refinement at different orders p



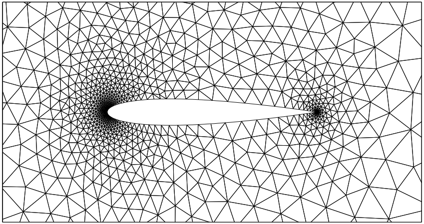
NACA 0012 in inviscid flow: optimized meshes



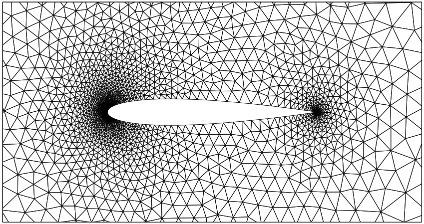
$p = 1$, `dof` = 2000



$p = 1$, `dof` = 4000

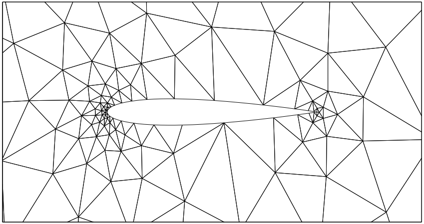


$p = 1$, `dof` = 8000

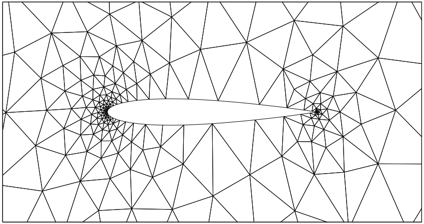


$p = 1$, `dof` = 16000

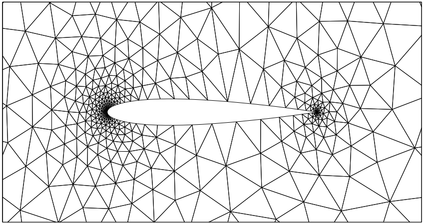
NACA 0012 in inviscid flow: optimized meshes



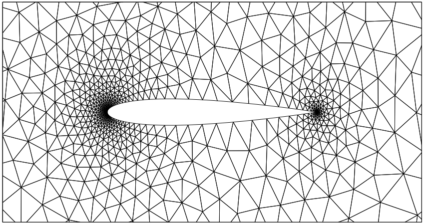
$p = 2$, dof = 2000



$p = 2$, dof = 4000

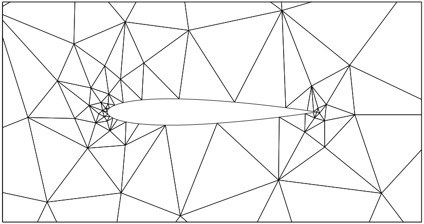


$p = 2$, dof = 8000

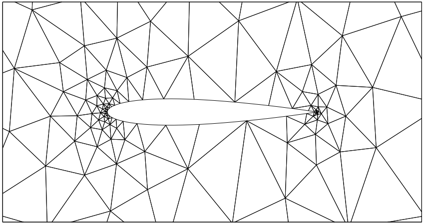


$p = 2$, dof = 16000

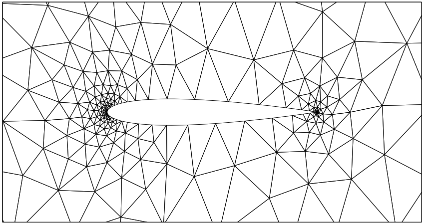
NACA 0012 in inviscid flow: optimized meshes



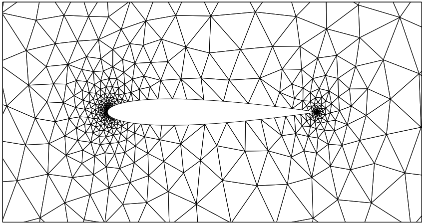
$p = 3$, `dof` = 2000



$p = 3$, `dof` = 4000



$p = 3$, `dof` = 8000

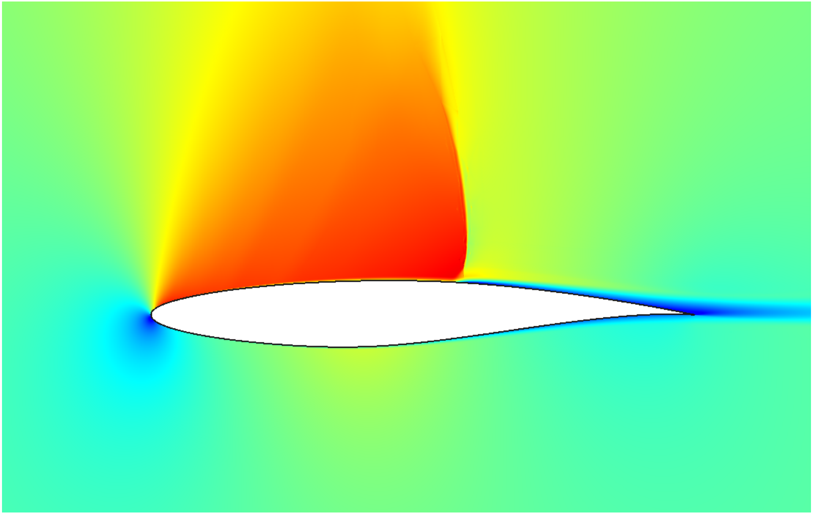


$p = 3$, `dof` = 16000

Example: RAE 2822 in transonic flow

RANS-SA, $M_\infty = 0.73, \alpha = 2.79°, Re = 6.5M$, output = drag

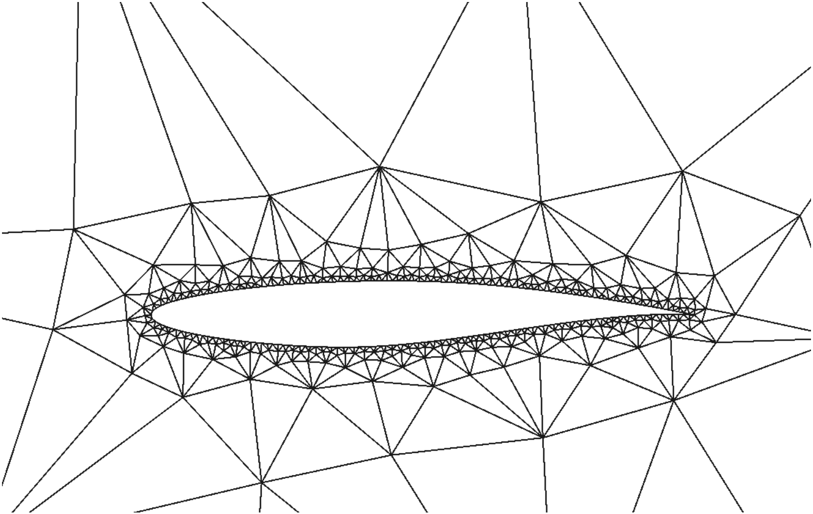
Mach number contours



Example: RAE 2822 in transonic flow

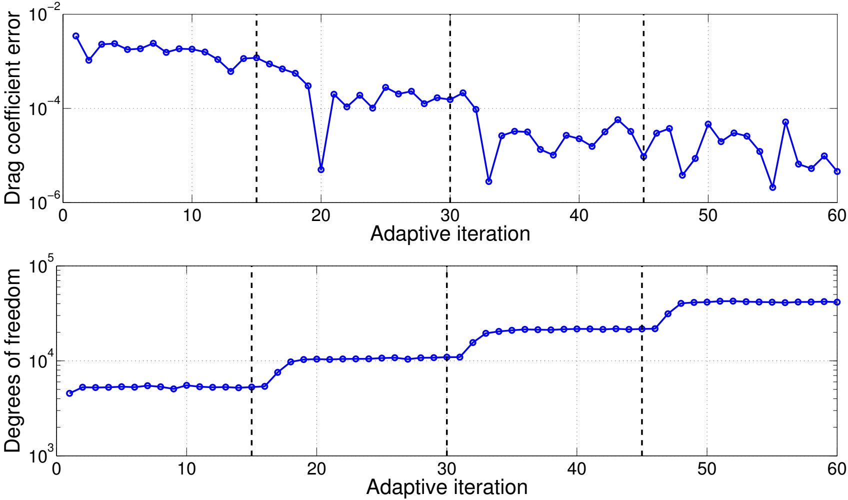
RANS-SA, $M_\infty = 0.73, \alpha = 2.79^\circ, Re = 6.5M$, output = drag

Initial mesh: 758 triangles, farfield @$2000c$



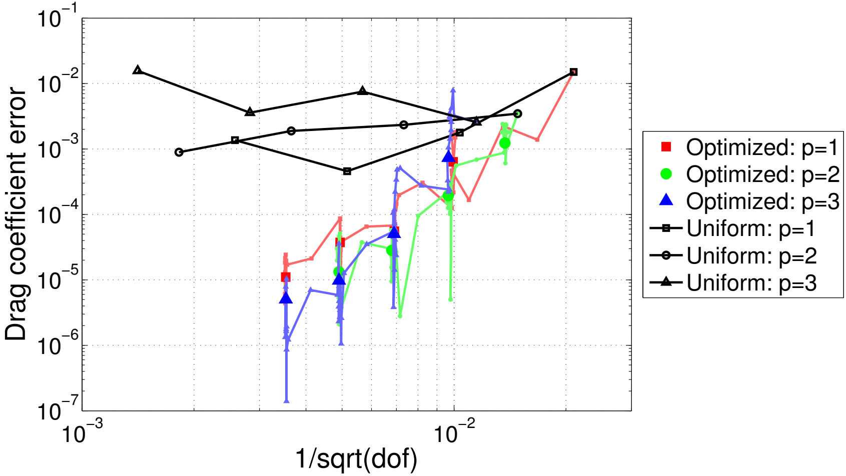
RAE 2822 in transonic flow: sample run

$p = 2$, 15 optimization iterations at each `dof`

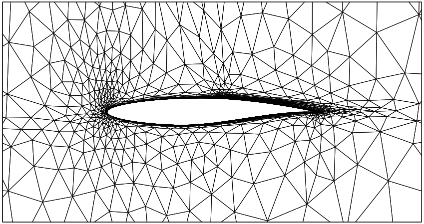


RAE 2822 in transonic flow: output convergence

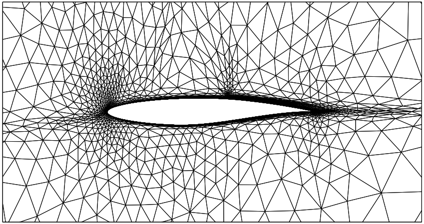
Compare to uniform refinement at different orders p



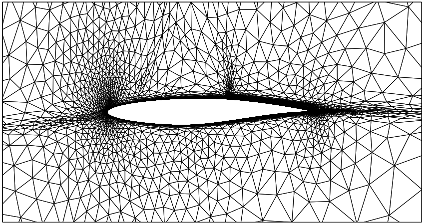
RAE 2822 in transonic flow: optimized meshes



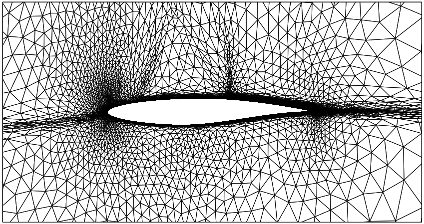
$p = 1$, `dof` = 5000



$p = 1$, `dof` = 10000

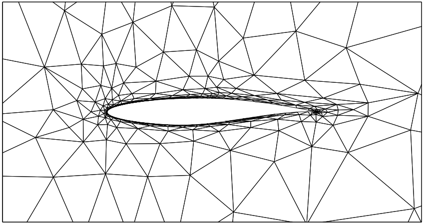


$p = 1$, `dof` = 20000

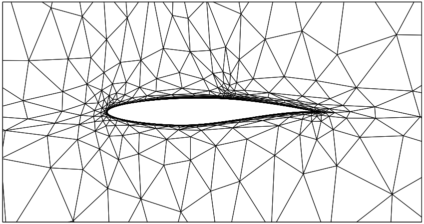


$p = 1$, `dof` = 40000

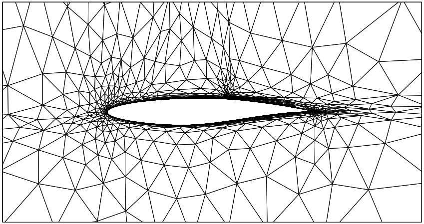
RAE 2822 in transonic flow: optimized meshes



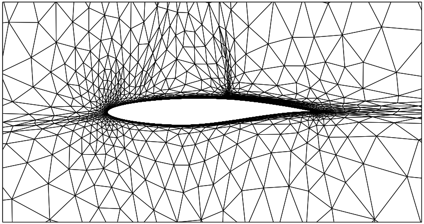
$p = 2$, `dof` = 5000



$p = 2$, `dof` = 10000

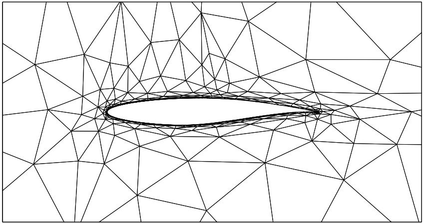


$p = 2$, `dof` = 20000

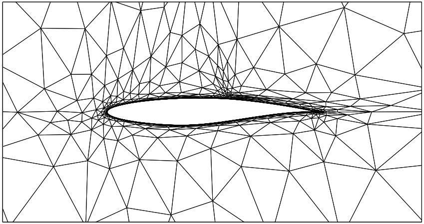


$p = 2$, `dof` = 40000

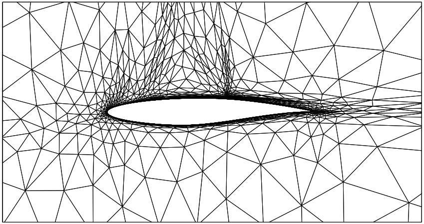
RAE 2822 in transonic flow: optimized meshes



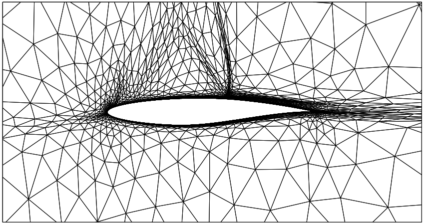
$p = 3$, `dof` = 5000



$p = 3$, `dof` = 10000



$p = 3$, `dof` = 20000



$p = 3$, `dof` = 40000

Summary

- We can quantify numerical error and adapt the mesh to reduce it at its source
- This requires an adjoint and fine-space calculations
- Exact fine-space adjoints yield effective error estimates that we can use as corrections
- Approximate fine-space adjoints (e.g. via Jacobi smoothing) are still good for adaptation
- Mesh anisotropy is critical for efficiently resolving boundary layers
- Hanging-node anisotropy through single cuts is limited by structure of original mesh
- Unstructured metric-based mesh regeneration offers an opportunity to globally optimize meshes for accurate output prediction at minimal cost

Outline

References I

[1] S.R. Allmaras, F.T. Johnson, and P.R. Spalart.
Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model.
Seventh International Conference on Computational Fluid Dynamics (ICCFD7) 1902, 2012.

[2] Garrett E. Barter.
Shock Capturing with PDE-Based Artificial Viscosity for an Adaptive Higher–Order Discontinuous Galerkin Finite Element Method.
PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2008.

[3] Timothy Barth and Mats Larson.
A posteriori error estimates for higher order Godunov finite volume methods on unstructured meshes.
In R. Herban and D. Kröner, editors, *Finite Volumes for Complex Applications III*, pages 41–63, London, 2002. Hermes Penton.

[4] R. Becker and R. Rannacher.
A feed-back approach to error control in finite element methods: Basic analysis and examples.
East-West Journal of Numerical Mathematics, 4(4):237–264, 1996.

[5] H. Borouchaki, P. George, F. Hecht, P. Laug, and E Saltel.
Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie I: Algorithmes.
INRIA-Rocquencourt, France. Tech Report No. 2741, 1995.

[6] Marco A. Ceze and Krzysztof J. Fidkowski.
An anisotropic hp-adaptation framework for functional prediction.
American Institute of Aeronautics and Astronautics Journal, 51:492–509, 2013.

[7] Marco A. Ceze and Krzysztof J. Fidkowski.
Drag prediction using adaptive discontinuous finite elements.
AIAA Paper 2013-0051, 2013.

[8] Krzysztof J. Fidkowski and David L. Darmofal.
An adaptive simplex cut-cell method for discontinuous Galerkin discretizations of the Navier-Stokes equations.
AIAA Paper 2007-3941, 2007.

References II

[9] M. B. Giles and N. A. Pierce.
Adjoint equations in CFD: duality, boundary conditions and solution behavior.
AIAA Paper 97-1850, 1997.

[10] James Lu.
An a Posteriori Error Control Framework for Adaptive Precision Optimization Using Discontinuous Galerkin Finite Element Method.
PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2005.

[11] Todd A. Oliver.
A High–order, Adaptive, Discontinuous Galerkin Finite Elemenet Method for the Reynolds-Averaged Navier-Stokes Equations.
PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2008.

[12] Xavier Pennec, Pierre Fillard, and Nicholas Ayache.
A riemannian framework for tensor computing.
International Journal of Computer Vision, 66(1):41–66, 2006.

[13] R. Rannacher.
Adaptive Galerkin finite element methods for partial differential equations.
Journal of Computational and Applied Mathematics, 128:205–233, 2001.

[14] P. Solín and L. Demkowicz.
Goal-oriented hp-adaptivity for elliptic problems.
Computer Methods in Applied Mechanics and Engineering, 193:449–468, 2004.

[15] D. A. Venditti and D. L. Darmofal.
Grid adaptation for functional outputs: application to two-dimensional inviscid flows.
Journal of Computational Physics, 176(1):40–69, 2002.

References III

[16] Masayuki Yano.
An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes.
PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2012.