

# FUN3D v12.4 Training

## Session 9: Adjoint-Based Design for Steady Flows

Eric Nielsen



# Learning Goals

- Introduction and basic approach taken in FUN3D
- Some lingo/nomenclature
- What is an adjoint, and what is it used for?
  - Error estimation and mesh adaptation
  - Sensitivity analysis for design optimization
- Design variables
- Objective/constraint functions
- Geometry parameterizations
- Setup and execution of a simple unconstrained problem
- Things to watch out for
- How to interpret results

## What we will *not* cover

- Body transforms, body grouping
- Overset grid details
- Multipoint/multiobjective/constrained optimization
- Hooking in your own optimizer, parameterization tools
- Forward-mode differentiation using complex variables
- Design of unsteady flows
  - Later session



# What to Expect

- Cost of design optimization is very problem-dependent, but in general you can expect to spend ~20 times the cost of a flow solution to get reasonable improvements, depending on how “good” the baseline is
- Generally see very rapid improvements initially, followed by diminishing returns
- We will cover the bare essentials here; also see the manual
  - There are many aspects we will not have time to cover here
- Hands-off design is challenging – be patient, send in questions, and we’ll try to help you through
  - There are a lot of pieces involved, and getting things running smoothly always involves stumbling blocks along the way



# Design Optimization Using FUN3D

- Based on a gradient-based approach
- FUN3D is distributed with support for several COTS gradient-based optimization packages
  - You must download and install your choice of these third-party libraries
    - DOT/BIGDOT (Vanderplaats R&D)
    - KSOPT (Greg Wrenn @ Langley)
    - PORT (Bell Labs)
    - NPSOL (Stanford)
    - SNOPT (Stanford)
    - Other packages are generally straightforward to hook up – couple of hours
- These optimizers are based on the user supplying functions and gradients (and perhaps constraints and their gradients also)
  - Optimizers know nothing about CFD, all they see are  $f$  and  $\nabla f$
- In CFD, objective/constraint functions are generally based on things like lift, drag, pitching moment, etc.
  - But can be anything you code up, generally speaking



# Design Optimization Components

## Functions

- When the optimizer requests a function value, it requires a flow solution with inputs and a grid corresponding to the current design variables

## Gradients

- When the optimizer requests a gradient value, it requires a sensitivity analysis with inputs and a grid corresponding to the current design variables
  - The most straightforward way to generate sensitivity information is to perturb each design variable independently and run black-box finite differences
    - This is prohibitively expensive when each finite difference requires a new CFD simulation (or two) – cost scales linearly with the number of design variables
  - The most efficient sensitivity analysis approach for CFD simulations based on large numbers of design variables (hundreds or thousands) is the adjoint method



# Notation and Governing Equations

We wish to perform rigorous adaptation and design optimization based on the steady-state Euler/Navier-Stokes equations, *without requiring any a priori knowledge of the problem:*

$$\frac{\partial \mathbf{Q}}{\partial t} + \mathbf{R}(\mathbf{D}, \mathbf{Q}, \mathbf{X}) = 0$$

**R** = Spatial residual

**Q** = Dependent variables

**D** = Design variables

**X** = Computational grid

- Incompressible through hypersonic flows
- May include turbulence models and various physical models from perfect gas through thermochemical nonequilibrium



# What is an Adjoint?

Combine cost function with Lagrange multipliers  $\Lambda$ :

$$L(\mathbf{D}, \mathbf{Q}, \mathbf{X}, \Lambda_f, \Lambda_g) = \underbrace{f(\mathbf{D}, \mathbf{Q}, \mathbf{X})}_{\text{Cost Function}} + \Lambda_f^T \underbrace{\mathbf{R}(\mathbf{D}, \mathbf{Q}, \mathbf{X})}_{\text{Flowfield Equations}} + \Lambda_g^T \underbrace{(\mathbf{K}\mathbf{X} - \mathbf{X}_{surf})}_{\text{Mesh Movement Equations}}$$

$f$  = Cost function (lift/drag/boom/etc)

$\Lambda_f$  = Flowfield adjoint variable

$\mathbf{K}$  = Mesh movement elasticity matrix

$\Lambda_g$  = Grid adjoint variable

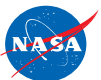
Differentiate with respect to  $\mathbf{D}$ :

$$\frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \left[ \frac{\partial \mathbf{R}}{\partial \mathbf{D}} \right]^T \Lambda_f + \left[ \frac{\partial \mathbf{Q}}{\partial \mathbf{D}} \right]^T \left\{ \frac{\partial f}{\partial \mathbf{Q}} + \left[ \frac{\partial \mathbf{R}}{\partial \mathbf{Q}} \right]^T \Lambda_f \right\} + \left[ \frac{\partial \mathbf{X}}{\partial \mathbf{D}} \right]^T \left\{ \frac{\partial f}{\partial \mathbf{X}} + \left[ \frac{\partial \mathbf{R}}{\partial \mathbf{X}} \right]^T \Lambda_f + \Lambda_g^T \mathbf{K} \right\} - \Lambda_g^T \left[ \frac{\partial \mathbf{X}}{\partial \mathbf{D}} \right]_{surf}$$

$$\longrightarrow \underbrace{\left[ \frac{\partial \mathbf{R}}{\partial \mathbf{Q}} \right]^T}_{\text{Governing Eqns}} \Lambda_f = - \underbrace{\frac{\partial f}{\partial \mathbf{Q}}}_{\text{Engineering Output}}$$

This adjoint equation for the flowfield has powerful implications for:

- Error estimation & mesh adaptation
- Sensitivity analysis



# Adjoint for Error Estimation and Mesh Adaptation

It is apparent that:

$$\Lambda_f \equiv \frac{\partial f}{\partial \mathbf{R}}$$



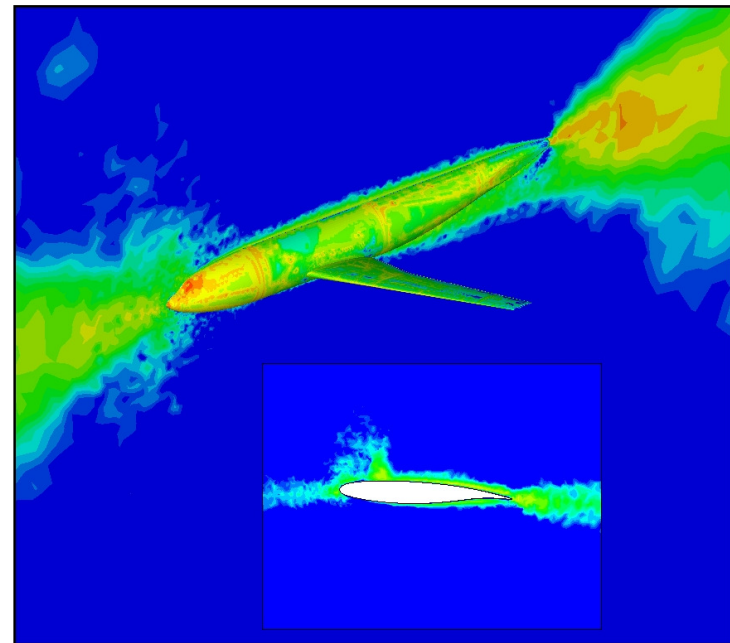
*Direct relationship between local equation error and the output we are interested in*

- These relationships can be used to get error estimates on  $f$
- Also used to compute a scalar field explicitly relating local point spacing requirements to output accuracy for a user-specified error tolerance
- Often yields non-intuitive insight into gridding requirements
- Relies on underlying mathematics to adapt, rather than heuristics such as solution gradients

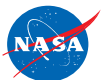
User no longer required to be a CFD expert to get the right answer

Transonic Wing-Body:

*“Where do I need to put grid points to get 10 drag counts of accuracy?”*



**Blue=Sufficient Resolution**  
**Red=Under-Resolved**

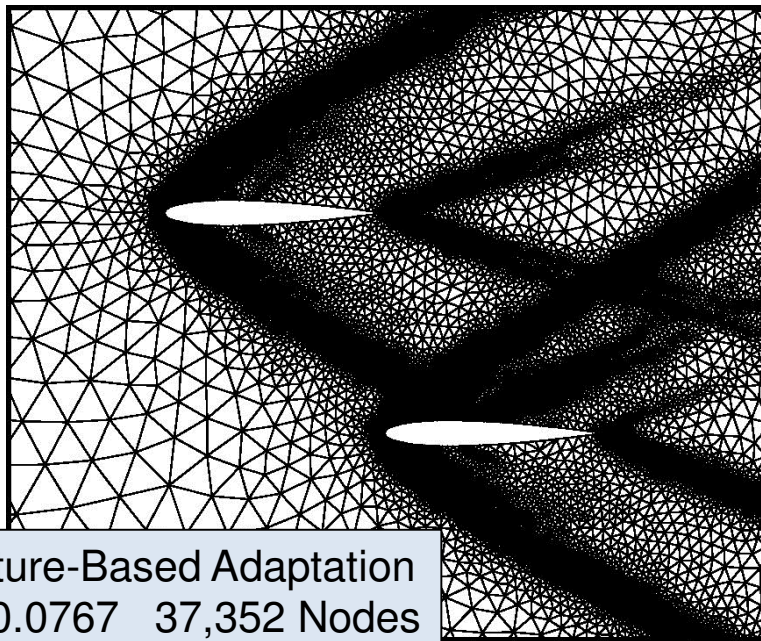
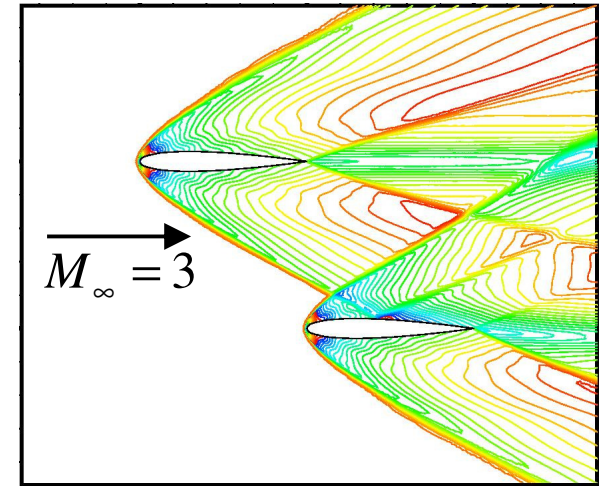




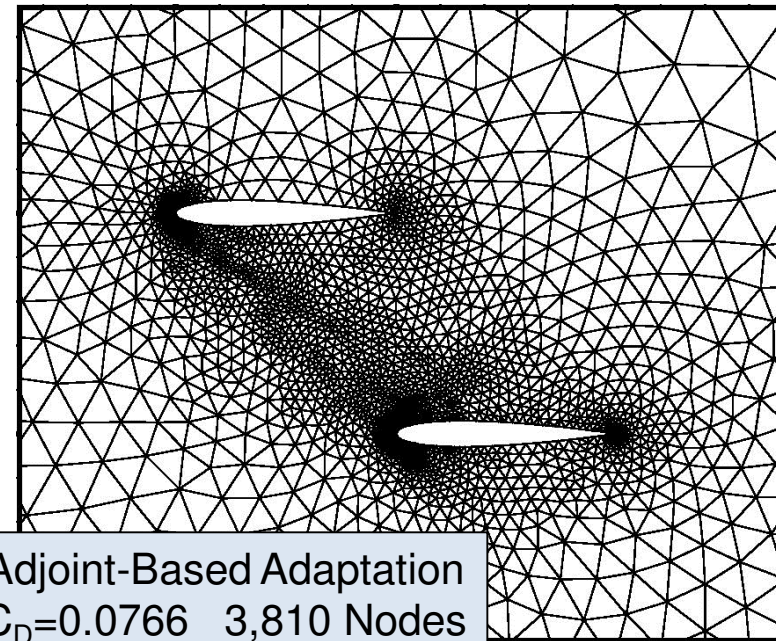
# Supersonic Adjoint-Based Mesh Adaptation

*Collaboration with Venditti/Darmofal of MIT using FUN2D*

- **Objective:** Adapt grid to compute drag on lower airfoil as accurately as possible
- **Result of adjoint-based adaptation:**
  - Uniformly-resolved shocks are not required
  - Drag is computed accurately with a 90% smaller grid



Feature-Based Adaptation  
 $C_D=0.0767$  37,352 Nodes



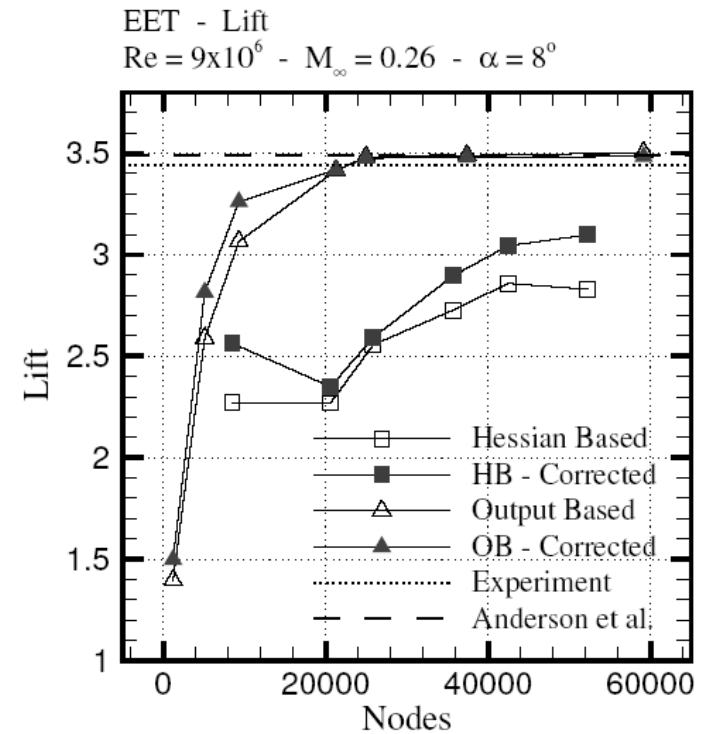
Adjoint-Based Adaptation  
 $C_D=0.0766$  3,810 Nodes



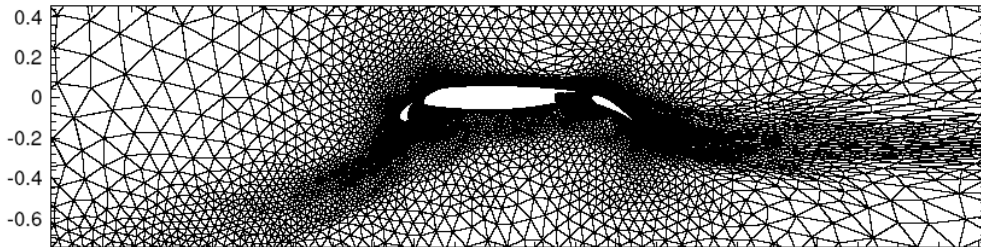
# Adjoint-Based Mesh Adaptation for High Lift

Collaboration with Venditti/Darmofal of MIT using FUN2D

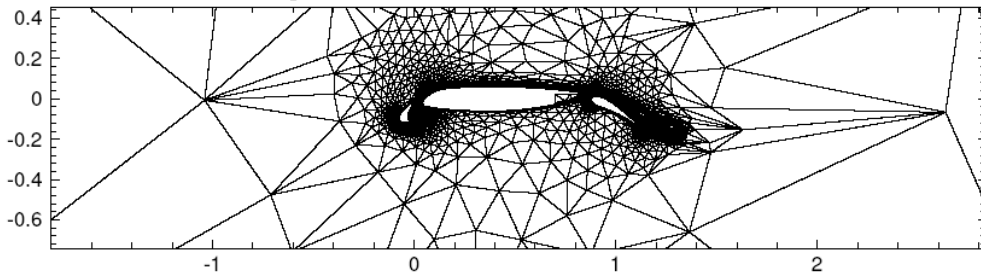
- Initial grid was coarse Euler mesh
- Pressure-based indicator only resolves strong flow curvature
- Adjoint-based indicator also includes important smooth regions, stagnation streamline and wakes



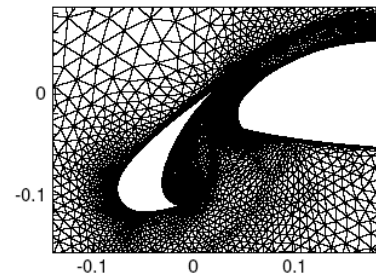
Output-based Adaptation (Lift) 24965 Nodes



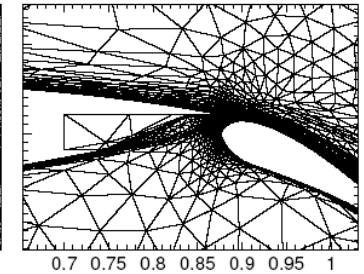
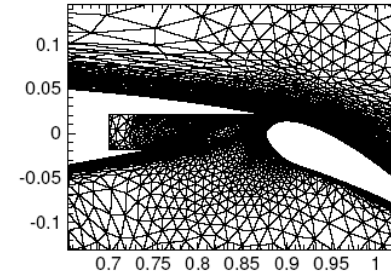
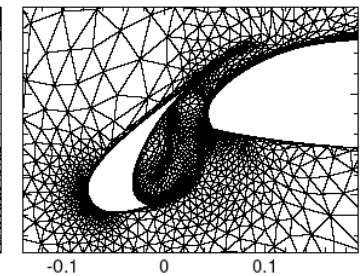
Pure Hessian-Based Adaptation 52235 Nodes



Output-based Adaptation (Lift) 24965 Nodes



Pure Hessian-Based Adaptation 52235 Nodes



# Adjoint for Sensitivity Analysis

Examine the remaining terms in the linearization:

$$\frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \left[ \frac{\partial \mathbf{R}}{\partial \mathbf{D}} \right]^T \Lambda_f + \left[ \frac{\partial \mathbf{X}}{\partial \mathbf{D}} \right]^T \left\{ \frac{\partial f}{\partial \mathbf{X}} + \left[ \frac{\partial \mathbf{R}}{\partial \mathbf{X}} \right]^T \Lambda_f + \Lambda_g^T \mathbf{K} \right\} - \Lambda_g^T \left[ \frac{\partial \mathbf{X}}{\partial \mathbf{D}} \right]_{surf}$$

$$\longrightarrow \mathbf{K}^T \Lambda_g = - \left\{ \frac{\partial f}{\partial \mathbf{X}} + \left[ \frac{\partial \mathbf{R}}{\partial \mathbf{X}} \right]^T \Lambda_f \right\} \quad \text{Discrete adjoint equation for mesh movement}$$

$$\longrightarrow \frac{dL}{d\mathbf{D}} = \frac{\partial f}{\partial \mathbf{D}} + \Lambda_f^T \frac{\partial \mathbf{R}}{\partial \mathbf{D}} - \Lambda_g^T \left[ \frac{\partial \mathbf{X}}{\partial \mathbf{D}} \right]_{surf} \quad \text{Sensitivity equation}$$

## Function Evaluation

1. Compute surface mesh at current  $\mathbf{D}$
2. Solve mesh movement equations
3. Solve flowfield equations

## Sensitivity Evaluation

3. Solve flowfield adjoint equations
2. Solve mesh adjoint equations
1. Matrix-vector product over surface

**Analysis Cost = Sensitivity Analysis Cost**  
 Even for *1000's* of design variables



# Design Variables in FUN3D

- Global flowfield variables
  - Mach number, angle of attack
- Shape variables
  - These depend entirely on the geometric parameterization being supplied to FUN3D
  - FUN3D has no native shape variables, other than the grid points themselves
- Additional variables related to unsteady simulations



# Objective/Constraint Functions in FUN3D

$$f_i = \sum_{j=1}^{J_i} \omega_j (C_j - C_j^*)^{p_j}$$

$\omega$  = weight     $C$  = aero coeff  
 $p$  = power     $C^*$  = target aero coeff

- We call each term in the summation a *component* function and the summation  $f_i$  a *composite* function
- User may specify which boundary patch in the grid (or all) to which each component function applies
- Constraints may be explicit or added as “penalties”
- Multipoint/multiobjective: as many composite functions/constraints as desired
  - Only limited by particular optimization package
  - Adjoints for multiple functions/constraints computed simultaneously
- The optimization always seeks to *minimize* the objective function(s), so pose them accordingly
- This general form leads to numerous ways to pose an optimization problem; each optimizer has its own limitations though
  - Extensive discussion in manual



# Objective/Constraint Functions Examples

Unconstrained Drag Minimization

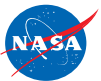
$$f = C_D^2$$

Drag Minimization with  $C_L=0.5$  Lift Penalty

$$f = 10C_D^2 + (C_L - 0.5)^2$$

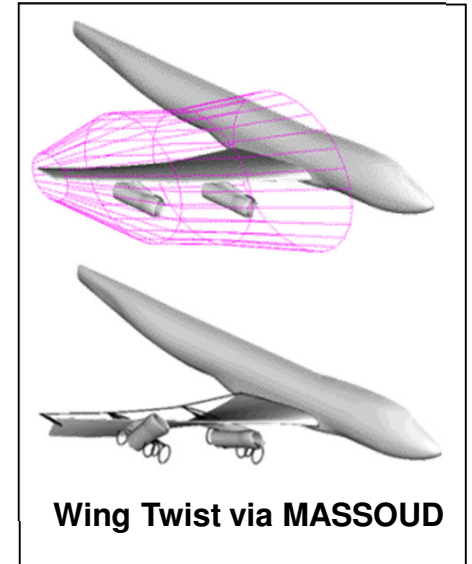
Drag Minimization with Explicit  $C_L=0.5$  Lift Constraint

$$f_1 = C_D^2 \quad f_2 = C_L$$



# Geometry Parameterizations

- FUN3D relies on a pre-defined relationship between a set of parameters, or design variables, and the discrete surface mesh coordinates
- Given  $\mathbf{D}$ , surface parameterization determines  $\mathbf{X}_{surf}$  (surface mesh)
- For example, given the current value of wing thickness at a location, what are the corresponding xyz-coordinates of the mesh?
- This narrows down the number of design variables from hundreds of thousands (raw grid points) to dozens or hundreds
  - Optimizers will perform more efficiently
  - Smoother design space
- The other requirement of the parameterization package is that it provides the Jacobian of the relationship between the design variables and the surface mesh,  $\partial\mathbf{X}_{surf}/\partial\mathbf{D}$
- While users may provide their own parameterization scheme, FUN3D is set up to handle three common packages:
  - MASSOUD: Aircraft-centric design variables (thickness, camber, planform, twist, etc)
  - Bndaids: General patching tool to handle fillets, winglets, and other odd shapes
  - Sculptor: Commercial package from Optimal Solutions
- To dump out the surface grids in the Tecplot format necessary for these tools, run the flow solver with `'--write_massoud_file'`
  - This procedure generates a `[project]_massoud_bndryN.dat` file for the  $i^{\text{th}}$  solid boundary



# Directory Tree for FUN3D-Based Design

## Design

- Main directory for design execution
- The only directory here without a hardwired name

## Design/ammo

- Design is executed from here using the `opt_driver` executable
- `ammo.input` resides here

## Design/description.i

- `i` suffix is an integer referring to the design point (to accommodate multipoint design)
- Contains all of the baseline files describing this design point (CFD model and all input decks specific to it)
- **The optimization never changes anything in here**; this is where the optimizer can always find the problem definition
- **You provide the problem description for the  $i^{\text{th}}$  design point here**

## Design/model.i

- `i` suffix is an integer referring to the design point (to accommodate multipoint design)
- All CFD runs are performed here
- **You never change anything in here**; it only contains outputs

## Design/model.i/Flow

- All flow solutions are performed here

## Design/model.i/Adjoint

- All adjoint solutions are performed here

## Design/model.i/Rubberize

- All parameterization evaluations are performed here

**You need not set up this tree manually; the code will do it for you, provided some basic pathnames**

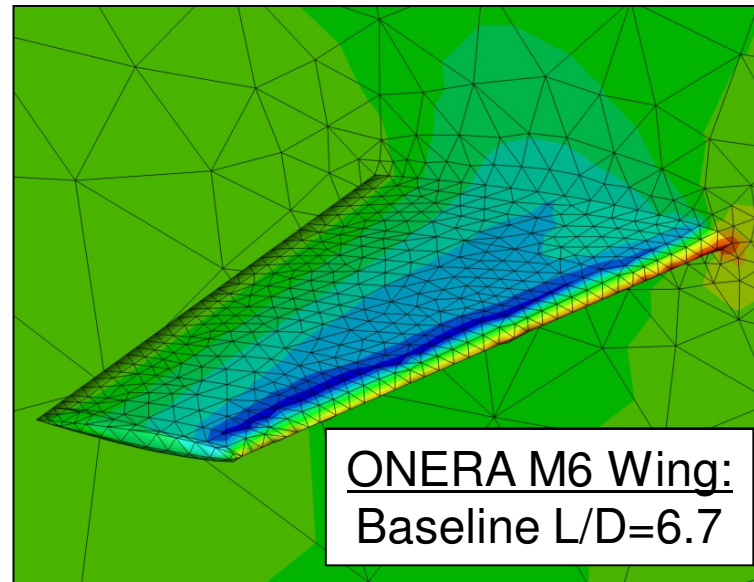
## Design/model.i/Rubberize/surface\_history

- A Tecplot file for every surface grid evaluated during the design is stored here



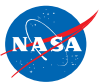


# Maximize L/D for Transonic Flow Over a Wing



- To create the directory structure necessary for performing the optimization, issue the following command:  

```
'/path/to/your/FUN3D/installation/Design/opt_driver --setup_design 1'
```
- The trailing integer represents the number of design points desired
- This command will prompt you for several paths and then will set up the required directory structure
- First we will discuss the files that must be provided in the `description.1` directory



# Maximize L/D for Transonic Flow Over a Wing

*Files Required in description.1 Directory*

```
command_line.options
```

```
3
0 flow
1 adjoint
`--rmstol 1.e-3'
0 mpirun
```

- This file is used to specify any command line options (CLOs) required by the FUN3D executables, as well as MPI
- The first line specifies the number of executables for which you are providing CLOs
- This is followed by a line containing an integer and a keyword
  - The integer specifies the number of CLOs you are providing for the code identified by the keyword
- This is followed by the actual CLOs for the current executable
- Note ‘mpirun’ is an available keyword: this provides a mechanism to feed your mpirun executable any options it may require (`-nolocal`, `-machinefile filename`, etc.)
  - Depends on your environment, queue structure, etc.



# Maximize L/D for Transonic Flow Over a Wing

*Files Required in description.1 Directory*

*We are assuming the use of a MASSOUD parameterization for this example*

**design.1, design.gp.1**

- These files are input files for MASSOUD for the 1<sup>st</sup> body; the MASSOUD setup tool provides these when you set up your parameterization
- Do not change these files

**design.usd.1**

- This file is an input file for MASSOUD for the 1<sup>st</sup> body; the MASSOUD setup tool provides this template when you set up your parameterization
- Depending on how you choose to “link” raw MASSOUD variables to create new variables, this defines the linking weights (see MASSOUD documentation)
- When using MASSOUD with FUN3D, you must always use the design variable linking option, even if simply set to the identity matrix



# Maximize L/D for Transonic Flow Over a Wing

*Files Required in description.1 Directory*

design.usd.1

```
# this is input sd file for MASSOUD
# number of row == number dvs within MASSOUD
# number of col == final number dvs
#(row) (col) (#of nonzero rows)
10 11 10
  d   1d   2d   3d   4d   5d   6d   7d   8d   9d  10d  11d
  1   1   0   0   0   0   0   0   0   0   0   0
  2   0   1   0   0   0   0   0   0   0   0   0
  3   0   0   1   0   0   0   0   0   0   0   0
  4   0   0   0   1   0   0   0   0   0   0   0
  5   0   0   0   0   1   0   0   0   0   0   0
  6   0   0   0   0   0   1   0   0   0   0   0
  7   0   0   0   0   0   0   1   0   0   0   1
  8   0   0   0   0   0   0   0   1   0   0   1
  9   0   0   0   0   0   0   0   0   1   0   1
 10   0   0   0   0   0   0   0   0   0   1   1
```

- Our demo problem uses 166 variables; this sample file only shows 10 raw variables plus 1 linked variable for clarity
- Linked variable is equal combination of raw DV's 7-10



# Maximize L/D for Transonic Flow Over a Wing

## *Files Required in description.1 Directory*

### massoud.1

```
#MASSOUD INPUT FILE
# runOption (0 analysis), (> 0 sd using user's dvs ) (-1, sd using massoud's dvs)
166
# core (0 incore solution)(1 out of core solution)
0
# input parameterized file
design.gp.1
# design variable input file
design.1
# input sensitivity file (used for runOption > 0
design.usd.1
# output file grid file
new1.plt
# output tecplot file for viewing
model.tec.1
# file containing the design variables group
designVariableGroups.1
# user design variable file
customDV.1
```

- This file tells MASSOUD the names of its input/output files for the 1<sup>st</sup> body
- The first value specifies the number of linked MASSOUD design variables
  - If linking matrix is identity, this is just the number of raw MASSOUD design variables
- The remainder of the inputs are filenames; they should remain as is, but with the integer value in each name set to the index of the current body



# Maximize L/D for Transonic Flow Over a Wing

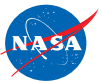
*Files Required in description.1 Directory*

## `fun3d.nml`

- This is the nominal solver input deck for your case
- The adjoint solver also uses this input
  - If the adjoint requires different values (e.g., stopping tolerance), you can override these values with CLOs given in `command_line.options`
- It should contain the necessary inputs to run the baseline case
- The optimization will override values as needed using CLOs (e.g., angle of attack, etc)

## `[project].fgrid, [project].mapbc`

- This is the nominal mesh for your baseline case in whatever grid format is convenient

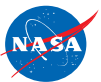


# Maximize L/D for Transonic Flow Over a Wing

*Files Required in description.1 Directory*

## **rubber.data**

- This is the main design control file used to define the design variables and their bounds, objective functions, and constraints for the current design point
- It also stores current values of functions and sensitivities
- A copy of this file is placed in the `model.1` directory at the beginning of an optimization and is continuously updated with the current values of the design variables, objective/constraint functions, and all gradient information
  - If you want to know the latest info during a design, it's probably in here



# Maximize L/D for Transonic Flow Over a Wing

*Files Required in description.1 Directory*

## `rubber.data`: Design Variable Block

- In general, for each design variable, you must set several fields
  - Active (0=no, 1=yes), baseline value, upper and lower bounds (if active)
- First subsection lays out the Mach number and angle-of-attack information
- This is followed by an input stating the number of bodies to be designed
- Then for each body:
  - Fixed number of rigid motion variables – leave these alone (used for unsteady flows)
  - Number of shape variables and their inputs – these correspond directly to the MASSOUD variables previously discussed
    - When setting bounds for shape variables, it pays to be conservative – the optimizer will exploit every radical shape it can dream up
    - You can quickly get into unsolve-able or invalid/crossed-up geometries
    - You can always loosen up the bounds and restart the design if needed





# Maximize L/D for Transonic Flow Over a Wing

## *Files Required in description.1 Directory*

```
#####
##### Design Variable Information #####
#####
Global design variables (Mach number / angle of attack)
Index Active          Value          Lower Bound      Upper Bound
Mach    0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
AOA     0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
Number of bodies
1
Rigid motion design variables for body 1 (name of body 1, less than 80 cols)
Var Active          Value          Lower Bound      Upper Bound
RotRate  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotFreq  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotAmpl  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotOrgx  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotOrgy  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotOrgz  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotVecx  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotVecy  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
RotVecz  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
TrnRate  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
TrnFreq  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
TrnAmpl  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
TrnVecx  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
TrnVecy  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
TrnVecz  0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
Parameterization Scheme (Massoud=1 Bandajids=2 Sculptor=4)
1
Number of shape variables for body 1 (name of body 1, less than 80 cols)
166
Index Active          Value          Lower Bound      Upper Bound
1    0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
2    0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
3    0    0.000000000000000E+00  0.000000000000000E+00  0.500000000000000E+01
.
.
```



# Maximize L/D for Transonic Flow Over a Wing

*Files Required in description.1 Directory*

## **rubber.data: Function Block**

- These sections lay out the objective/constraint function definitions
- First input is the total number of composite functions being specified (sum of objectives + constraints)
- Then, for each function:
  - Is it an objective function (1) or a constraint (2)
  - If it is a constraint, what are the upper and lower bounds (otherwise dummies)
  - How many component functions are used to build up the composite function
  - Time step interval defining the function (leave as dummies – for unsteady design)
  - Composite function weight/target/power: for further generality, described in manual
  - Then the list of component functions:
    - Boundary index it applies to (0 means all boundaries)
    - Keyword identifying the function type (see manual)
    - Value (dummy – this is an output during the optimization)
    - Weight/target/power to be applied to current component function
    - The remainder of the function block is devoted to sensitivity outputs – you can place dummies here, *but there must be a line corresponding to every design variable*



# Maximize L/D for Transonic Flow Over a Wing

## *Files Required in description.1 Directory*

```
#####
##### Function Information #####
#####
Number of composite functions for design problem statement
  1
#####
Cost function (1) or constraint (2)
  1
If constraint, lower and upper bounds
  0.0 0.0
Number of components for function  1
  1
Physical timestep interval where function is defined
  1 1
Composite function weight, target, and power
1.0 0.0 1.0
Components of function  1: boundary id (0=all)/name/value/weight/target/power
  0 clcd          0.0000000000000000          1.000  20.00000 2.000
Current value of function  1
  0.0000000000000000
Current derivatives of function wrt global design variables
  0.0000000000000000
  0.0000000000000000
Current derivatives of function wrt rigid motion design variables of body  1
  0.0000000000000000
  0.0000000000000000
  0.0000000000000000
.
.
.
Current derivatives of function wrt design variables of body  1
  0.0000000000000000
  0.0000000000000000
  0.0000000000000000
.
.
.
```

Our objective function:

$$f = (L/D - 20)^2$$



# Maximize L/D for Transonic Flow Over a Wing

*ammo/ammo.input*

- We are now finished setting things up in the `description.1` directory
- There is one more file that needs to be set up in the `../ammo` directory
- The `ammo.input` file controls the actual optimization procedure

**ammo.input**

- Everything pretty self-explanatory, but a few reminders:
  - Optimization packages: `DOT/BIGDOT=1`, `KSOPT=3`, `PORT=4`, `NPSOL=5`, `SNOPT=6`
  - “Operation to perform”: `analysis=1`, `sensitivity analysis=2`, `optimization=3`
  - Note you can specify the `mpirun` executable name
    - Useful if executable is called ‘`mpiexec`’, ‘`aprun`’, or otherwise on your system
  - “DOT method” corresponds to the “method” variable in the DOT manual



# Maximize L/D for Transonic Flow Over a Wing

*ammo/ammo.input*

```
Optimization package
4
Base directory from which to run optimization
'/workshop/Wing'
Number of design points
1
Weights for each design point
1.0
Operation to perform
1
Restart the optimization
0
Maximum number of flow solves
20
Maximum number of design cycles
10
Relative convergence criterion for subproblem
1.e-5
Absolute feasibility tolerance for constraint violation
110.0
Number of bodies with spatial transforms
0
List of bodies with spatial transforms
Body grouping desired (0=no, 1=yes)
0
Executable for running MPI programs
'mpirun'
Number of processors from which to run adjoint solver
8
DOT method
0
```



# Maximize L/D for Transonic Flow Over a Wing

## *Running a Function Evaluation*

- Things are now ready for execution
- The first thing I typically do is just run a function evaluation to see that the parameterization and all of the inputs are set correctly
- To do this, edit `ammo.input` and set `Operation` to `Perform to 1`
- From the `ammo` directory, the command line that is used to run this case is

```
./opt_driver --sleep_delay 5
```

- The '`--sleep_delay 5`' instructs the design driver to wait 5 seconds in between operations – allows NFS caching to keep up
- Different systems may require more time (or none)



# Maximize L/D for Transonic Flow Over a Wing

## *Running a Function Evaluation*

- The first thing that you will see is MASSOUD evaluating the parameterization for each body, defining the surface grid coordinates at the baseline position
- The flow solver will then start up, but prior to the solve, you will see an auxiliary solution take place that represents the interior mesh movement based on the elasticity equations
  - For this first step at the baseline position, you should see very small numbers for the “Natural Error Est” (close to machine zero): this indicates the current surface mesh is very close to the requested surface mesh
- After the actual flow solution takes place, the solver will evaluate each of the objective and constraint functions you posed:  

<b>Current value of function</b>	<b>1</b>	<b>178.087727950984</b>
----------------------------------	----------	-------------------------
- This marks the end of a successful function evaluation
- Always wise to plot the flow solver convergence – you want to run enough iterations to get a “reasonable” answer (outputs resolved beyond what you are expecting from design changes), but you don’t necessarily need to drive it into the ground



# Maximize L/D for Transonic Flow Over a Wing

## Running a Function Evaluation

[MASSOUD Screen Output]

Sleeping to allow file system time to catch up...

Executing: mpirun nodet\_mpi --animation\_freq -1 --design\_run --irest 0

FUN3D 12.4-69612M Flow started 02/25/2014 at 12:56:31 with 8 processes

[Echo of fun3d.nml]

[Usual preprocessing info]

Using linear elasticity to reposition grid...

reading ../rubber.data ...

reading:../Rubberize/model.tec.1.sdl

Iter	Natural Err Est	Error Estimate	Restarts
0	0.648914658284637E-16	0.000000000000000E+00	0

Iter	density_RMS	density_MAX	X-location	Y-location	Z-location
1	0.725550147064997E-04	0.46595E-03	0.34893E-01	0.60683E-01	0.00000E+00
	Lift	0.657554528793843E-01	Drag	0.319926994134964E-01	
...					
74	0.207836490870309E-09	0.82846E-08	0.22500E+01	0.45000E+01	0.65000E+01
	Lift	0.881383268442809E-01	Drag	0.132438291863532E-01	

Writing boundary output: inviscid\_tec\_boundary.dat

Time step: 74, ntt: 74, Prior iterations: 0

Writing inviscid.flow (version 11.8) lmpi\_io 2

inserting current history iterations 74

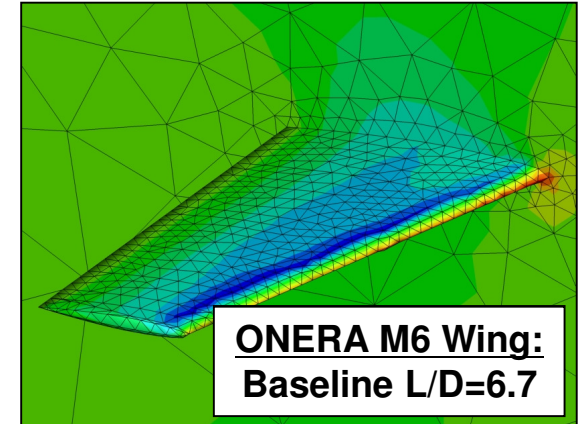
Time for write: .0 s

Current value of function 1 178.087727950984

writing ../rubber.data ...

Done.

Analysis complete.





# Maximize L/D for Transonic Flow Over a Wing

## *Running a Gradient Evaluation*

- Now lets test a sensitivity analysis
- Edit `ammo.input` and set `Operation` to `Perform to 2`
- Submit the job just as before
- The first thing that will take place is a function evaluation, just as before
- After the function evaluation takes place, MASSOUD will fire up again to evaluate the linearizations of the surface mesh coordinates with respect to the design variables
- FUN3D's adjoint solver will then start up:
  - You will see a solution taking place; this is the flowfield adjoint
  - Afterwards, you will see another solution occurring; this is the elasticity adjoint for the mesh
  - The final step is to update the `model.1/rubber.data` file with the sensitivity information
- This marks the end of a successful sensitivity analysis
- Again, it is wise to plot the convergence of the flowfield adjoint system
  - This convergence history is in the `model.1/Adjoint/[project]_hist.tec` file
  - In general, you want 2-3 orders of magnitude convergence; this is usually sufficient for reasonable sensitivity information



# Maximize L/D for Transonic Flow Over a Wing

## Running a Gradient Evaluation

[Function Evaluation]

[MASSOUD Screen Output]

Sleeping to allow file system time to catch up...

Executing: mpirun -np 8 dual\_mpi --rmstol 1.e-3 --getgrad --irest 0  
FUN3D 12.4-69612M Adjoint started 02/25/2014 at 13:30:45 with 8 processes

[Echo of fun3d.nml]

[Usual preprocessing info]

Iter	adjoint RMS	adjoint MAX	X location	Y location	Z location
1	0.707037901636711E+00	0.30235E+01	0.57720E+00	0.95000E+00	0.13288E-01
2	0.221413741319278E+02	0.77671E+03	0.22500E+01	0.45000E+01	0.65000E+01
3	0.252132505507981E+02	0.85665E+03	0.22500E+01	0.45000E+01	0.65000E+01
...					
79	0.108404219416308E-02	0.48685E-01	0.20671E+00	0.43560E+01	0.19196E+01
80	0.961305851711102E-03	0.43086E-01	0.20671E+00	0.43560E+01	0.19196E+01

Performing linear elasticity adjoint...

reading ../rubber.data ...

Using defaults for move\_relaxation.schedule.

Boundary 1 allowed to deform with y=constant constraint

Iter	Natural Err Est	Error Estimate	Restarts
0	0.540562915758561E+04	0.100000000000000E+01	0
1	0.351062487957891E+02	0.649438719756149E-02	0
11	0.426070657988252E-02	0.788198090485649E-06	0

writing ../rubber.data ...

Done.

Sensitivity analysis complete.



# Maximize L/D for Transonic Flow Over a Wing

## *Running the Optimization*

- If you got this far, things are looking pretty good – we’ve checked that everything is set up to run functions and gradients correctly, which is all the optimizer depends on
- Now we’re ready to try an actual optimization
  - Edit `ammo.input` and set `Operation` to `Perform` to 3; submit the job like usual
- Now you will see a lot of function and gradient evaluations going by, as the optimizer starts to change design variables and search for an optimum solution
- One easy way to monitor progress is to grep your screen output:

– `'grep "Current value" screen.output'` :

```
Current value of function      1  178.087727950984
Current value of function      1  137.781365382028
Current value of function      1  109.428433033596
Current value of function      1  95.6295276213557
Current value of function      1  98.1556950554243
Current value of function      1  90.6778838105760
Current value of function      1  90.5396548069062
Current value of function      1  87.6654651887965
Current value of function      1  87.6871379069251
Current value of function      1  87.1318618979881
Current value of function      1  86.8957965299972
Current value of function      1  87.3525553665462
Current value of function      1  86.5144724882407
Current value of function      1  86.8116011407977
Current value of function      1  86.2791108703313
Current value of function      1  86.2399365109043
Current value of function      1  86.2399356879804
```

- You can also observe (but don’t change!) the file `model.1/rubber.data`

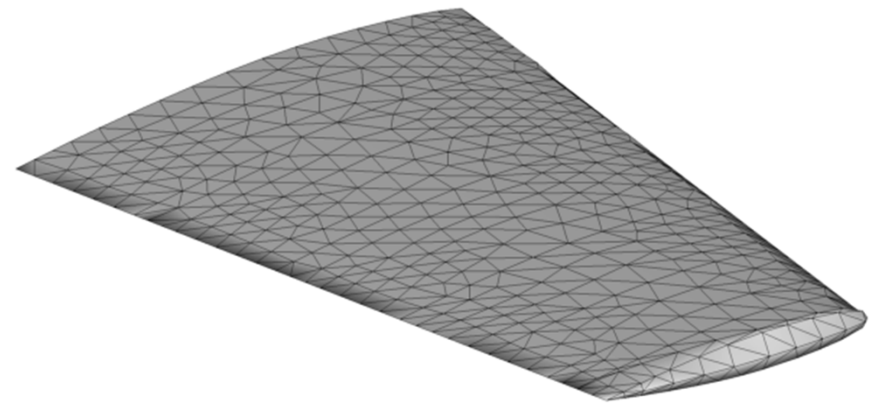
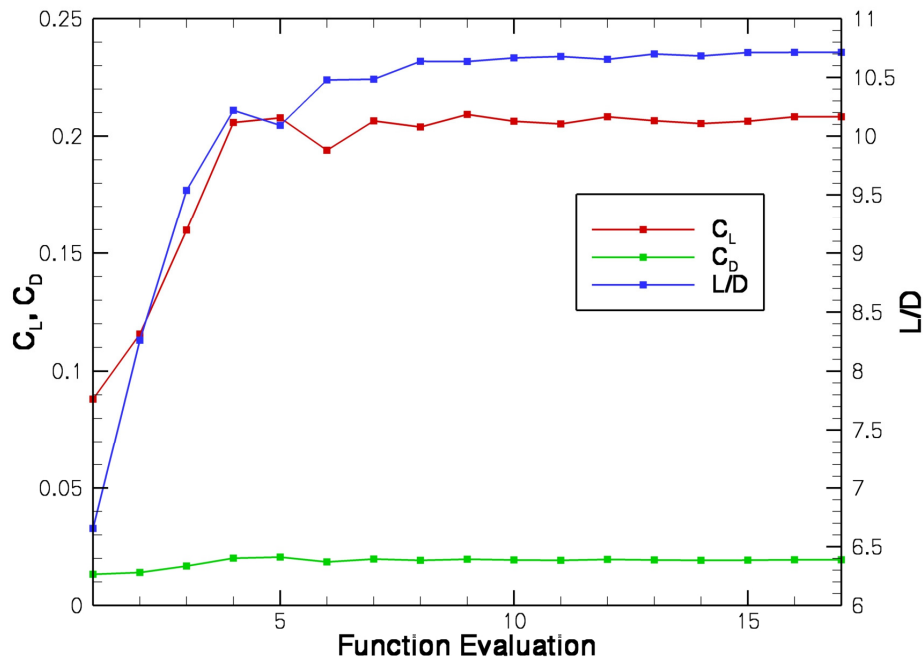


# Maximize L/D for Transonic Flow Over a Wing

## *Post Mortem*

- After the job finishes, PORT will summarize its performance in the file `model.1/port.output`
- Since each solution is a warm start, you can plot the entire flow solution history contained in `model.1/Flow/[project]_hist.tec`
- A history of the surface geometry is stored in `model.1/Rubberize/surface_history/model.tec.1.sd1.iteration.*`

Redesigned Wing:  
L/D=10.7



# What Could Possibly Go Wrong?

- The procedure can terminate due to CFD-related problems:
  - Running into negative volumes during a mesh movement (you can plot the history of the surface(s) using the files in `model.1/Rubberize/surface_history`)
    - Watch for invalid surfaces or unusually large changes
    - Be conservative in your lower/upper bounds!
  - The flowfield or the adjoint solution is unstable
    - Problem-dependent; get in touch for advice
- The procedure can also terminate due to hardware/environment problems
  - You run out of allocated time, a node dies, etc.
- Finally, the procedure can terminate if the optimizer has given up:
  - No more progress can be made due to constraints
  - The optimizer has hit the max number of functions/gradients you allowed
  - An optimal solution has been found



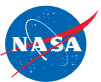
# List of Key Input/Output Files

## Input

- In `description.i` directory:
  - All files necessary to run solutions for  $i$ th design point (grid files, `fun3d.nml`, etc)
  - All parameterization files for  $i$ th parameterized body
  - `command_line.options`
  - `rubber.data`
- `ammo/ammo.input`

## Output

- All files normally associated with running the solver
- `rubber.data`
- `port.output`
- Design history in `model.1/Rubberize/surface_history`



# Summary of Design Optimization for Steady Flows

- That's more or less the basic pieces involved with running an optimization
- Lots of options we did not cover here; see manual or get in touch for help
  - How the wrappers work (`LibF90/analysis.f90`, `LibF90/sensitivity.f90`)
  - Parameterizations other than MASSOUD
  - Multipoint/multiobjective (tutorial on website)
  - Constrained problems (tutorial on website)
  - Running with other optimization packages (tutorial on website)
  - Body grouping, spatial transforms
  - Overset grids
  - Forward-mode sensitivity analysis using complex variables
  - Unsteady design (later session)

## General Advice

- Become very comfortable with the flow solver
- Work the website tutorials
- Learn how to set up parameterizations using MASSOUD and/or bandaids
- Try plugging in your own grids/parameterizations in the tutorials
- Ask questions – it's actually not that bad once you get up the learning curve



# What We Learned

- General approach used by FUN3D for design optimization
- What is an adjoint
- What does a function/gradient evaluation consist of in terms of CFD
- Design variables in FUN3D
- Functions/constraints in FUN3D
- What is required of a geometry parameterization tool
- How to set up the inputs required for design optimization
- How to run function, gradient evaluations
- How to perform a basic design optimization
- What to watch out for and how to interpret results

