FUN3D
Fully Unstructured Navier-Stokes

Current Release: **12.4-70371**
Site Last Updated: **Wed Jun 04 16:41:01 -0400 2014**

Previous (C7: Adaptation and Error Estimation) | Up | Next (C9: Appendix)

8. Design

8.1. Getting Started

The set of FUN3D design tools has been under heavy development since 1995. A gradient-based approach to the design methodology has been taken, as the cost of computing high number of functions for other techniques is typically prohibitive for CFD simulation-based design. Work within the FUN3D research group has been aimed primarily at robust and efficient analysis and sensitivity analysis capabilities, rather than the optimization schemes themselves. However, others at Langley have done numerous optimization studies using FUN3D as the simulation package.

At this point, all design work must be done on fully tetrahedral grids; mixed element capabilities have not yet been propagated through all of the design tools. **It is assumed that the user has a very good knowledge of using FUN3D for analysis purposes; beginners should get plenty of experience running the flow solver as a stand-alone tool before jumping into design work.** Moreover, please read this documentation before contacting the FUN3D Software Development Team.

One last note before we get started—don't even think about trying optimization without a fair number of high-end processors available for 24/7 use. The algorithms are very efficient, but a typical optimization will still usually cost you O(10) flow solves.

Setting Up The Directory Structure

The FUN3D design tools rely on a specific directory structure that must be present exactly as described here. The optimization driver can set this tree up for you if you want. Simply execute the following command from the Design directory of your build location (this assumes your repository resides on the filesystem where you will be running):

```
./opt_driver --setup_design 1
```

The "1" argument specifies that you wish to only do a single-point design. If you want to set things up for multi-point design, change the "1" to the number of design points you'll be running.

The program will prompt you for the absolute directory name in which your FUN3D source code resides. Enter it in this format, with no trailing slash, and *with* the single quotation marks:

```
'/path/to/my/FUN3D/installation'
```

The program will also prompt you for the absolute directory name in which your FUN3D build resides (executables). Enter it in this format, with no trailing slash, and *with* the single quotation marks:

```
'/path/to/my/FUN3D/installation/build_directory'
```

Finally, the code will ask you for the directory in which you wish to set up your design run. This

directory must exist, must also be an absolute path with no trailing slash, and don't forget the single quotation marks.

```
'/path/to/my/design/case'
```

After entering the information, the code will populate the latter directory with the structure of files required to do a FUN3D-based design optimization. As the code terminates, it will print to the screen the remaining steps necessary to set up your design case. Pay close attention to this information: It serves as a very helpful reminder and will specify exactly which files need to be where and what parameters to set ahead of time! I still use it to help me on every case I run.

The design directory structure is primarily divided into three parts. The first of the three is called `description.1`. This directory contains the baseline files for the model, none of which are modified during the course of the design. It is simply a safe place where the code always knows that the original model resides. Prior to executing the design, the user will set up the baseline files in this directory. During execution, the optimization driver will pull files as needed. The trailing `1` on the directory name is used as a design point ID. If you requested more than one design point to be established when you set up the directory structure as described above, you will have additional `description.n` directories, where the files describing other design points will be located. During setup, templates of the required input decks will be placed in the `description.n` directories.

The second directory created in the location you provided will be called `model.1`. The CFD codes will perform all of their work in these subdirectories during the design. Beneath the `model.1` directory, you will also see subdirectories called `Adjoint`, `Flow`, and `Rubberize`. These directories are populated during the initialization phase with softlinks to each of the executables in the FUN3D installation on your machine. Details of each of the components are given on subsequent pages of this manual. Again, if you requested more than one design point during the setup step, you will also see other `model.n` directories, each of which will be used to perform the CFD steps for each design point.

The third directory created will be called `ammo`. This directory will contain files related to the optimization procedure itself.

Note that you should not have to change anything in the `model.n` directories prior to a design run. Everything you need to set up is in the `description.n` and `ammo` directories.

SETTING UP THE FILES REQUIRED TO DO DESIGN

The user must provide a set of files in the `description.n` directory to begin the design. These files are related to the CFD model, the geometric parameterization, the optimization, and the environment in which you are running. The files described below must be present, unless noted otherwise. Again, follow the steps shown at the end of running `opt_driver --setup_design 1` as outlined above.

CFD GRID FILES

The user must provide the baseline grid (and associated boundary condition files) to be used for the optimization. This can be in any of the grid formats currently supported by FUN3D.

SOLVER INPUT DECK(S)

The user must provide the baseline input deck `fun3d.nml` for the flow solve. It is strongly recommended that you manually run a flow solve and an adjoint solve (details on how to execute the adjoint solver as a standalone execution will come later) on your baseline configuration first, prior to doing any design, to get a feel for how the stopping tolerance, number of timesteps, and so forth should be set. Note the desired values may be different for the flow solve versus the adjoint solve; I typically place the flow solve values in fun3d.nml and override them via command-line options for the adjoint solver. More details on how to provide command line options will come below.

Depending on the input options you need to perform a solution, you may also need to provide optional files such as `moving_body.input`, etc. If any of these files are present in the `description.n` directory, the optimization procedure will use them when running FUN3D on that design point.

GEOMETRY PARAMETERIZATION FILES

If doing shape optimization, the user must provide a MASSOUD, bandaids, or Sculptor parameterization for each body in the mesh to be modified. Currently, the set of desired bodies must be entirely parameterized with a single one of the packages, but not combinations. (Future versions of the codes will hopefully allow arbitrary combinations.) The documentation here is pretty sparse; it is assumed that the user has obtained MASSOUD or the bandaids package from Jamshid Samareh or Sculptor and has already become familiar with their inputs and outputs.

For MASSOUD parameterizations, the MASSOUD parameter files should be named `design.gp.1`, `design.gp.i`, ..., `design.gp.n` for each of the `n` bodies to be designed. The files specifying the raw MASSOUD variables should be called `design.1`, `design.i`, ..., `design.n` for each of the `n` bodies to be designed. Note, however, that in the current implementation, you **must use** the custom design variable linking feature of MASSOUD. If you wish to use the raw MASSOUD variables as is, simply define the linking matrix as the identity matrix. These files specifying the design variable linking for each body should be named `design.usd.1`, `design.usd.i`, ..., `design.usd.n`.

Finally, if running MASSOUD, the MASSOUD input file specifies the names of the files described above and must be provided as `massoud.1`, `massoud.i`, ..., `massoud.n`. **The files listed in the MASSOUD input file must reflect the names given in the above paragraph.** In addition, the first line of these files must have a positive integer in them equal to the number of user-custom design variables. If you just want to use the raw MASSOUD variables and have specified the identity matrix in the linking file, this number is simply the number of raw MASSOUD variables for that body. For the in/out-of-core parameter, just use in-core (`0`). The filename for Tecplot output viewing must be named `model.tec.i` for the `ith` body. The remaining FAST output filename can be named to anything the user wishes; the FUN3D tools do not use this file. A `massoud.i` file should look like:

```
#MASSOUD INPUT FILE
# runOption 0-analysis, >0-sd users dvs, -1-sd massouds    dvs
52
# 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
newframe.fast.1
# output tecplot file for viewing
model.tec.1
# file containing the design variables group
designVariableGroups.1
# user design variable file
customDV.1
```

Also for MASSOUD, if you are using body transforms to reorient the MASSOUD parameterization into a more suitable reference frame for a body (such as rotating a rotor blade or vertical tail parameterization into position), the file describing the transform for the i-th body should be included as `transforms.i`. The format of a typical `transforms.i` file is as follows:

```
ROTATE 0.0 0.0 1.0 -120.0
```

This would rotate the MASSOUD parameterization for the i-th body by -120 degrees about a unit vector in the +z direction. Get in touch for how to use other transforms available, such as TRANSLATE and SCALE. More on specifying body transforms later when we get to the optimization input deck.

For bandAids parameterizations, the `.bandaid` files created by Jamshid's bandAids tool should be renamed `bandaid.data.1`, `bandaid.data.i`, ..., `bandaid.data.n`. Because bandaids are linear, these files are all that is required; no executable is needed. (FUN3D performs everything internally.)

For Sculptor parameterizations, the user must provide `[project].mdf`, `[project].sd1`, `[project].vol`, `[project].vsp`, `[project].stu`, and `[project].def` files. See Sculptor

documentation for more details on these files.

Machine File

This file named `machinefile` is optional, and provides a list of machines to run the MPI applications across. If you are running in a queue environment where the machines are chosen for you at runtime, you do not need this file.

Body grouping input

The file `body_grouping.data` is optional, and provides body grouping information. For example, if you are optimizing the Figure of Merit of a 3-bladed rotor, then you would want to associate the 3 blades (each typically specified as a separate parameterized body in MASSOUD files and `rubber.data`) into one group, so that your sensitivity derivatives would reflect a composite d(thrust)/d(DV) for all three blades. This capability requires that the bodies to be associated all have the exact same parameterization (same number of DVs on each body, etc). The format of this `body_grouping.data` file is as follows:

```
Number of groups to create
1
Number of bodies in group, list of bodies
3
1 2 3
```

Design Control File

The design is ultimately controlled by the data contained in a file called `rubber.data`. See the section below describing this file for information on its details.

Command Line Options File

This file specifies the command line options to be used with each code in the suite, as well as with `mpirun`. A template is provided in the `Design` directory of the source code distribution. The first line of the file specifies the number of codes for which you are specifying command line options. The subsequent line must contain an integer followed by a keyword. The integer specifies how many command line options you are providing for the code identified by the keyword. The valid keywords are `flow`, `adjoint`, `party`, and `mpirun`. This line is followed by a line for each of the command line options you wish to provide for the code identified by the keyword. Each command line option should appear in single quotation marks on its own line.

The optimization driver will append each of the options for the relevant code to the command line it uses to invoke the code. Note that this mechanism is also used for `mpirun` (or `mpiexec`, etc, described later), so that options such as `-nolocal` and `-machinefile` can be specified. If a host file is required of your MPI installation, you should add it as the file `machinefile` in the `description.i` directory. If present, the optimization driver will automatically use it, and the argument specified for `mpirun` (or `mpiexec`, etc) should be `-machinefile ../machinefile`. If you are running in a queue environment where the number of processors is set according to your queuing script and the specific machines are chosen for you at runtime, then you need not provide any command line options for `mpirun` (`mpiexec`, etc).

Target Pressure Data Files

FUN3D does have an inverse design capability where the cost function is composed of target pressures. But the implementation/execution is sort of messy, so it is not described at length here. The files are optional and need only be present for target pressure designs. The files must be named `cpstar.data.1`, `cpstar.data.i`, ..., `cpstar.data.n`. Contact FUN3D Support if you really want to pursue inverse design (target pressures). It's a bit of a pain.

Input Data File Summary

In summary, you should have a set of files in your `description.1` directory similar to:

`[project].fgrid`	FAST CFD grid file (could also be VGRID, etc)
`[project].mapbc`	FAST CFD boundary conditions file (could also be VGRID, etc)
`fun3d.nml`	Solver input deck
`namelist.input`	Optional solver input deck (obsolete in 10.9.0 and later)
`moving_body.input`	Optional solver input deck
`bandaid.data.i`	Bandaid parameterization for ith body (optional)
`design.gp.i`	MASSOUD parameterization for ith body (optional)
`design.i`	MASSOUD raw variables file for ith body (optional)
`design.usd.i`	MASSOUD linking matrix file for ith body (optional)
`massoud.i`	MASSOUD filenames file for ith body (optional)
`transforms.i`	Spatial transform input file for ith body (optional)
`body_grouping.data`	Body grouping information (optional)
`[project].mdf`	Sculptor project file (optional)
`[project].sd1`	Baseline surface info from `--write_massoud_file` (optional)
`[project].vol`	Sculptor project file (optional)
`[project].vsp`	Sculptor project file (optional)
`[project].stu`	Sculptor project file (optional)
`[project].def`	Sculptor project file (optional)
`machinefile`	List of cluster nodes on which to run the codes (optional)
`rubber.data`	Main control file for the design
`command_line.options`	The set of command line parameters to be used for every code, as well as mpirun
`cpstar.data.i`	Target pressure distribution for slice i (optional)

Optimizer Input: `ammo.input`

Now that the `description.1` directory has been populated and all of the necessary parameters have been set (except for `rubber.data`, described below), head over to the `ammo` subdirectory. You will need to specify the parameters in the `ammo.input` file. This will control the actual optimization procedure. For the optimization package, use either a `1` (DOT/BIGDOT), a `3` (KSOPT), a `4` (PORT), a `5` (NPSOL), or a `6` (SNOPT). Note you may only use optimization packages which you have installed and configured FUN3D to use (such as—with-PORT=/path/to/PORT). If you choose an unavailable package, the code will quit and tell you so. Specify the base directory for the optimization case using an absolute path in single quotation marks on the next line—no trailing slash. This should be identical to the path you specified earlier that pointed to where you wanted to run the design. Next, put the number of design points you plan on running. The next input is the weight to be applied when combining the design points if using a simple linear combination of functions. Otherwise, these inputs don't matter, but you must specify as many values as you have design points.

On the next line, you can choose the operation to perform. Simply putting a `1` here will just do an analysis on your configuration, i.e., place the surface grid according to the current design variables, move the mesh into place, then run a flow solve. Setting this parameter to a `2` will perform an analysis as just described, followed by a sensitivity analysis. Finally, setting this value to a `3` will perform the actual optimization. It is often useful to perform just an analysis first to see if you have that set up right, then do a sensitivity analysis to see if all of that is ready to go, then attempting the actual optimization last, if the analysis and sensitivity analysis worked correctly.

The following line allows for restarting the optimization from the last design point. If this is a `0`, the optimization will be started fresh and use the baseline model files from the `description.1` directory. If this is a `1`, the driver will just forge ahead with a new optimization, using the most recent set of data files that have been placed throughout the `model.1` directory by a previous execution. The appropriate files should have already been placed in the `description.1` directory as described above. For the diagnostics flag, just leave this as a `0`. For the max low-fidelity functions input, this should be set to the maximum number of flow solves that you are willing to execute. The max number of low-fidelity iterations should be set to the maximum number of design cycles you are willing to execute. Next, set the relative convergence criterion. This is basically a convergence criteria for the design procedure. The next input is the absolute feasibility tolerance for constraint violations. This is only relevant when using explicit constraints and represents the violation you are willing to

tolerate on your constraint functions specified in rubber.data. For the number of bodies with spatial transforms, this is the number of bodies for which you have provided a `transforms.i` file, followed by a list of the bodies themselves. If you enter 0 bodies with transforms, then the line containing the list of bodies should not be present. The next line states whether or not you will be performing body grouping (for which you need to provide a `body_grouping.data` input file). The next input tells the optimization driver how to initiate MPI processes. This is usually either `mpirun` or `mpiexec`, depending on your MPI flavor. After that, you need to tell FUN3D how many processors to use for the adjoint solver. In most cases, this is the same number of processors you request for your job; however, in the event one is doing design on overset grids, there may be a node reserved for SUGGAR++ in the flow solver, in which case the adjoint solver must also use one less processor, even though SUGGAR++ is not required for the adjoint solver. The final input is the optimization method to be used if DOT/BIGDOT is being used for the optimization. FUN3D interfaces with DOT/BIGDOT through the general ALLDOT API; see the ALLDOT documentation for valid values of this parameter.

Having fun yet? That should do it for the majority of the problem setup. We still need to deal with `rubber.data` and discuss some command line options for each code that you may want to list in `command_line.options`. Read on for more information on how to get your design up and running…

8.2. Setting Up rubber.data

This section describes how to set up each section of the main design control file, `rubber.data`. A template is provided in the `Adjoint` directory of the source code distribution.

Code Status Section

This section of the file is currently not in use and should not be altered by the user. (This block of data has been eliminated as of v11.5.)

Design Variable Information

This section of `rubber.data` lays out the design variables for the computation. The section is divided into global variables such as Mach number and angle of attack, as well as shape optimization variables. Each design variable appears on its own row in the file, and has several attributes that must be set by the user. The first column is just a dummy index and is merely to assist the user in quickly navigating through the file. The second column is a toggle to activate the design variable. If this value is a `1`, the variable will be allowed to change during the design. If the value is assigned a `0`, this variable will be held constant at the value specified. The third column is the initial value for the current design variable. Columns four and five specify the upper and lower bounds for the current design variable. Be very careful in choosing upper and lower bounds for shape variables. The optimizers tend to do the most drastic changes possible during the design run and you can wind up with some very infeasible shapes (or shapes the mesh movement/solvers cannot handle robustly) if you are not careful. Err on the side of conservatism – you can always restart a design later with larger bound constraints.

Global Design Variables

The freestream Mach number and angle of attack are available for use as design variables. Mach number is in the first row; alpha is in the second. The angle of attack is in degrees. Both the Mach number and alpha specified here will override whatever is present in the initial `fun3d.nml` file provided by the user. The freestream Mach number may not be activated for incompressible design calculations.

Shape Variables

The first row following the Mach number and angle of attack entries specifies the number of bodies that the user has parameterized separately using MASSOUD, or alternatively, the number of bandaids present in the problem. If using Sculptor, the number of bodies must be one, but the Sculptor project may internally include parameterizations of multiple bodies.

Following the number of bodies, there should be two sections of inputs for each body. **The bodies present in the computation may be listed in any order, but the order of their appearance in this control file must match the integer suffix on their parameterization files that you provide in the `description.1` directory, as well as files such as `body_grouping.data`, `transforms.i`, etc.** The first section for the current body concerns design variables governing rigid mesh motion and is only applicable for time-dependent problems (but must be present for steady cases too). The variables correspond to the rigid motion parameters used to specify body motion in the optional `moving_body.input` file. If active, the optimizer will compute their sensitivities and change them appropriately, just like any other design variable. The next line specifies the number of parameterized variables on the current body, and the subsequent lines lay out the design variable information for that body. A row of data must be provided for every variable in the parameterizations, whether you are using them or not. If you have 25 variables parameterized in a bandaid, then 25 rows must appear in the corresponding data block of `rubber.data`, even if only a subset is active. The same applies to design variables for MASSOUD or Sculptor. And if you have used the design variable linking feature in MASSOUD to create additional variables, they will also appear here. (Basically the number of rows has to equal the number of derived variables in the `design.usd.i` MASSOUD file for that body.)

Cost Function/Constraint Specification

The first line following the design variable sections specifies the number of functions and constraints to be used for the current design point. For a single unconstrained cost function, this value should be `1`. For a single-objective, single-constraint problem, this value should be a `2`. And so forth.

Following the value specifying the total number of functions and constraints, each function and/or constraint will have a block of data associated with it. The cost functions and constraints may be specified in any order.

The first line in the block specifies whether the current function block is an objective function or constraint function. The next line specifies the lower and upper bounds (in that order) for the function, if it is a constraint. If it is not a constraint, these 2 values do not matter. The CFD codes themselves do not care if a function is an objective or a constraint—they will provide values and derivatives for them regardless. The optimization driver is the only thing that cares for what the function is actually being used.

The next line states how many components compose the current function/constraint. The current form of these functions takes the form

$$f = SUM [omega x (C - C_{star}) ^ power]$$

Here, f is the current cost (or constraint), omega is a weighting factor that may be assigned any value the user desires, C is a generic variable representing an aerodynamic quantity (to be described below), C_{star} is the target value for the aero quantity, and power is an exponent to be applied to the difference $C-C_{star}$. The summation indicated by SUM is taken over the number of components.

Following the number of components, for unsteady flows, the user must specify the physical timestep interval over which the function applies. For steady flows, these two inputs can be anything.

Following the timestep interval, each component has a line in the file containing several pieces of data. The first column is the boundary condition ID over which to apply the current component. These correspond directly to the boundary conditions in your baseline grid. If you wish to apply a component over the whole grid (total drag, for example), simply put a `0` in this column. Alternatively, if you know you have a strong shock sitting on a flap in a drag minimization problem, you might specify your cost function as the pressure drag acting on just that boundary group, as opposed to the entire vehicle, so that the optimizer will really hone in on the flap. The next column is the keyword for the aero quantity to be used for the current function component. For a list of available keywords, see the module header of the file `forces.f90` in the LibF90 directory. Several of the quantities have not been differentiated (pretty much the more obscure ones). The adjoint solver will check your input to make sure the components you've requested are available. Contact FUN3D Support if a component you need is not available or if you're not sure. The next column contains the current value of the current function component. This is an output and need not be set by the user. The final three columns in the row correspond to the weight, target value, and power shown in the summation function above.

The next line in the file lists the current value of the composite function built up of its components. This is an output and need not be set by the user.

The remaining lines in the current function block contain the sensitivity derivatives with respect to all of the design variables listed in the top half of the file. These are outputs that need not be set by the user, however, **the user must provide a line for each design variable in each function block**. The values do not matter, but the codes will need positions to place the latest values. Note that this section is divided into derivatives with respect to the global design variables, the rigid motion variables, as well as the design variables on each of the bodies laid out in the top of the file.

8.3. Geometry Parameterizations

The FUN3D suite is currently set up to interface directly with geometry parameterizations processed by MASSOUD, bandaids, or Sculptor. MASSOUD and bandaids are capabilities developed by Jamshid Samareh of NASA Langley. Users are encouraged to contact him for copies of the software and detailed instructions on how to use them. His packages allow the user to parameterize completely arbitrary shapes using a free-form deformation technique. The packages are extremely efficient and robust, and also provide analytic derivatives for the parameterizations, necessary for FUN3D-based design. Sculptor is a popular commercial package developed by Optimal Solutions . FUN3D v11.5 and higher can utilize Sculptor parameterizations for design optimization.

To parameterize your surface grids using any of these tools, you will need to extract them to Tecplot files in a pre-established format. The easiest way to do this is to run a single iteration of the flow solver with the `--write_massoud_file` command line option. You will need to include a &massoud_output namelist in your fun3d.nml file that groups all of the required boundary patches for a body you wish to parameterize into a single body (note the boundary indices here must reflect any patch lumping that may have been requested in the solver):

```
&massoud_output
  n_bodies = 2                 ! wish to parameterize 2 bodies: a wing and a t
  nbndry(1) = 6                ! number of boundaries that comprise the wing
  boundary_list(1) = '3-8'     ! wing group boundaries (account for lumping!)
  nbndry(2) = 3                ! number of boundaries that comprise the tail
  boundary_list(2) = '9 10 12' ! tail group boundaries (account for lumping!)
/
```

This will generate a [project]_massoud_bodyN.dat file for each of the N body groups present. These files contain the information necessary to parameterize the surface grid(s)—see the documentation for the specific parameterization packages for further instructions on how to proceed from here.

Details for MASSOUD

If you use MASSOUD for your parameterizations, the MASSOUD executable must be visible in your environment's path, and must be named `massoud`. The optimization driver supplied with FUN3D will attempt to call this executable if MASSOUD parameterizations are present. If you feel the optimization driver is not seeing the MASSOUD executable in your path, try placing a copy of the executable in the `model.1/Rubberize` subdirectory. This is where MASSOUD is ultimately executed by the driver.

Details for bandaids

If you are using bandaids, no additional executables need be supplied— all parameterization manipulations are handled internally by the FUN3D optimization driver. (All of the relationships for bandaids are linear, so that the initial sensitivities remain constant and need only be read in to determine new coordinates for the surfaces.)

Details for Sculptor

If you use Sculptor for your parameterizations, the Sculptor executable must be visible in your environment's path, and must be named `sculptor`. The optimization driver supplied with FUN3D will attempt to call this executable if Sculptor parameterizations are present. If you feel the

optimization driver is not seeing the Sculptor executable in your path, try placing a copy of the executable in the `model.1/Rubberize` subdirectory. This is where Sculptor is ultimately executed by the driver.

If you choose to use Sculptor to parameterize your surface grid(s), the parameterization of all bodies must be bookkept within a single set of Sculptor input files. I.e., in our wing-tail example above, both bodies must be contained in a single instance of Sculptor files. Therefore, your &massoud_output namelist described above should group ALL of the desired boundaries necessary to describe the geometry(s) of interest into a SINGLE body (i.e., all of the boundaries contained in the wing and tail should be used to define a single body in the &massoud_output namelist):

```
&massoud_output
  n_bodies = 1                  ! wing and tail grouped into one body
  nbndry(1) = 9                 ! number of boundaries that comprise the wing,
  boundary_list(1) = '3-10, 12'! wing, tail boundaries (account for lumping!)
/
```

You can work independently on each of the desired bodies (wing, tail) within Sculptor, but they must appear as a single body to FUN3D. Finally, when setting up the design variables in rubber.data for FUN3D, the variables for all three bodies will appear in a single concatenated list as a single body.

After you have parameterized your model using Sculptor and you are setting up all of the files required for FUN3D-based design, you will need to place a copy of the original file that resulted from your `--write_massoud_file` command in the `description.i` directory, **but it must be renamed `[project].sd1`**. Sculptor requires this baseline file during the optimization.

Sculptor usage: Note that before performing design, the `[project].sd1` file must be first read into Sculptor in GUI mode as “Import Mesh/CFD as Tecplot Point FE”. Following this, the Sculptor volumes need to be imported onto the `[project].sd1` file, and then the model needs to be saved again. Once this is done, then the command “export model.tec.1” within a .def batch script generates a `model.tec.1.sd1` file as needed for FUN3D optimization.

During the course of a design optimization, FUN3D will invoke Sculptor in batch (non-GUI) mode. However, current versions of Sculptor will still attempt to communicate with an X server, even when run in this fashion. If your system does not run an X server (such as compute nodes on a cluster), then a fake X server such as Xvfb is recommended. You will then need to execute the fake server prior to running the design optimization. For example, a run script may have the following lines present:

```
.
.
.
Xvfb :1 &
setenv DISPLAY :1.0
.
.
.
<execute FUN3D design command>
.
.
.
```

The syntax here may vary; if this does not allow you to run Sculptor in batch mode on your target system, it is suggested to get in touch with Sculptor support for assistance.

Using Other Parameterization Schemes

Although the packages described above are the predominant choices to use, the interface to the parameterization scheme has been coded in a modular fashion, and the user may choose to use his or her own package. See the Customization section for additional details.

8.4. The Adjoint Solver

This section describes how to execute the adjoint solver manually. Normally this is handled for the

user by the perform_sensitivity_analysis wrapper, however, *it is highly recommended that the user perform an adjoint solution on the baseline configuration to get a feel for input parameters, kick-out strategies, and so forth, to use during the actual design optimization*.

Running An Adjoint Solution

To perform an adjoint solution, the user must first perform a flow solution in the `model.1/Flow` directory. Once the flow solve is complete, a copy of the desired `rubber.data` file must be placed in the main `model.1` directory (one level above where the flow solve was run). The adjoint solver will read `rubber.data` to get objective/constraint function information for its right-hand side. The file `fun3d.nml` (`moving_body.input`, etc) must reside in the `Flow` directory, right where it was for the flow solution.

To execute the adjoint solver, change over to the `model.1/Adjoint` directory and enter the following command (`mpirun` arguments may be modified for your environment):

```
mpirun -np ## -nolocal -machinefile ../machinefile ./dual_mpi
```

This will run simultaneous adjoint solutions for each objective/constraint outlined in `rubber.data` for the current flow field. Note that additional command line options may be needed for your case, such as unsteady flows, noninertial, etc. When complete, you should see a set of `[project]_adj.i` files in the `model.1/Adjoint` directory. These are the adjoint solutions on each partition of the grid, and may be repartitioned and/or post-processed using party, in the same manner as regular flow solution files. Another file that will come out of the adjoint solver will be `[project]_hist.dat`, a file similar to that of the same name that comes out of the flow solver. This file is a Tecplot file that contains the residual convergence histories for the adjoint solution. You will want to keep track of how far you wish to converge the density adjoint residual—loosely speaking, your sensitivities will converge at the same rate as your functions (i.e., flow solution), but maybe you don't need your derivatives to be super-accurate for optimization purposes. In this case, you may want to specify a looser `RMSTOL` for the adjoint solver by providing a command-line override for the adjoint solver. Because the magnitudes of the adjoint residuals scale with the cost function/constraint definitions, you will probably want to specify a different `RMSTOL` for the adjoint solution anyway.

Another thing to be aware of, particularly for turbulent flows, is the degree to which you converge your flow solutions. You may find that your forces converge relatively quickly during the flow solve, at which point you may be tempted to terminate the flow solution and fire up an adjoint. However, if the flow field has not reached a sufficiently steady-state, the subsequent adjoint solution may diverge. This is the single most important reason for running an adjoint solution by hand a priori—to get a feel for the convergence levels you need in the flow field to obtain a stable adjoint solution. This is not a major stumbling block for inviscid or laminar flows, but this can be troublesome for RANS problems. This a burden on the user, and we're working on automated ways to detect stability requirements.

8.5. Running the Optimization

When you have finished populating all of the necessary files and setting all of the various input parameters, you are ready to run the design case. When you are running on a cluster system with a remote filesystem, I have found that it is most robust to execute from the first node in the cluster, rather than a "head" node which controls the worker nodes. I have found that the rapid execution of the various codes causes the filesystem to have a hard time keeping up. By running on the first node, it seems to allow the filesystem to keep up. This is just my experience, you may have to fiddle with things, depending on how your hardware is set up. To execute the optimization, go into the `ammo` directory and enter the following command:

```
./opt_driver --sleep_delay 30 > screen.output &
```

The—sleep_delay option will force a sleep of 30 seconds in between code executions in an attempt to allow the file system to keep up. The default (if you do not specify it as above) is 120 seconds, which is probably more than you need.

Redirecting the output into a file will provide a record of what flies by on the screen from each code

as the design progresses. It also lets one see the latest info if you are logged in remotely (from home in the middle of the night, etc). **I would highly suggest watching the output during the first design cycle and from time-to-time during the run.** This file is also very useful in the event you need to contact us and we need to help you debug your run.

Once the run completes, you should have some sort of design history stored in `model.i/dot.history` and `dot.output` (if running DOT/BIGDOT), `model.i/port.output` (if running PORT), `npsol.printfile` and `npsol.summaryfile` (if running NPSOL), or `ksopt.output` (if running KSOPT). Another file that may be of interest is `movie.tec` in the `model.1/Flow` directory. This file is a Tecplot file that is appended with the latest grid and flow solve information after each function evaluation. By using Tecplot's "Animate Zones" option, you can animate the design and see what the grid was doing, how solution contours changed, and so on. Finally, the final set of design variables determined by the optimizer will be available in `model.1/rubber.data`.

8.6. Customization

Hooking In Your Own Optimizer

In the design context, the term "function" for CFD computations includes a mesh movement (both surface and volume), a flow solution, and an evaluation of the cost function (and possibly any constraints) at a given set of design variables. For those interested in using the tools at a high-level and do not necessarily need to know what's "under the hood", a wrapper has been provided in the `LibF90` directory of the distribution named `analysis.f90`. This module contains a subroutine called `perform_analysis` which will perform the above operations.

To obtain sensitivities, the FUN3D package relies on a discrete adjoint formulation. This approach yields discretely consistent sensitivity derivatives at the same cost as the baseline analysis described above. A call to the `perform_sensitivity_analysis` routine in the `sensitivity.f90` module will perform an adjoint solution for the flow field, an adjoint solution for the mesh movement scheme, and obtain the final sensitivity derivatives that are requested.

Using the two wrappers provided, users should hopefully be able to hook up to their optimization package of choice with little trouble. Feel free to get in touch for guidance in hooking the wrappers up to your framework.

Using Your Own Parameterization Scheme

Users may use their own parameterization schemes, as long as the file formats match those of FUN3D. Contact FUN3D Support for details on hooking in your own parameterization package. You will need to be able to provide *xyz*-coordinates for the bodies of interest as well as derivatives of these coordinates with respect to the parameterization variables.

Implementing New Cost Functions / Constraints

Implementing new cost functions or constraints will obviously require some low-level coding. Although the codes are continuously being refactored for modularity and ease of maintenance, there is still a pretty steep learning curve for adding new capabilities. The user should be well versed in F95/2003, unstructured grids, programming in a domain-decomposed environment, and CFD in general. The user will need to provide a basic routine to evaluate the function, and routines to evaluate the linearizations of the function with respect to both the flow-field variables (for the flowfield adjoint solver) and with respect to the grid (for mesh adjoint). Users wanting to venture down this path should probably get in touch with FUN3D support ahead of time to get advice on implementation issues they may end up facing. To check your work, we will highly recommend that you verify your linearizations against the complex-variable form of FUN3D—see the accompanying section on this topic.

Other Customizations

Feel free to get in touch for advice on implementing any other modifications. Our team are more CFDers than optimization or aircraft design gurus, so we are interested in hearing what outside groups actually need in order to get the job done. If your application is of mutual interest we will try

to support a joint effort.

8.7. Forward-Mode Differentiation

Although a reverse, or adjoint, mode of differentiation is primarily used for design with FUN3D, a forward-mode of differentiation is also provided. This capability is useful for design problems containing few design variables and many cost functions or constraints. It is also useful for aero-structural optimization, where derivatives of the dependent variables may be needed at every grid point on the surface. Finally, it is invaluable during code development of new linearizations. For a description of the complex-variable "trick" (it's super easy), see some of the FUN3D publications.

To generate a complex-variable formulation of FUN3D, configure your FUN3D installation with the —enable-complex option and compile. You will get complex versions of the flow solver and mesh movement codes in the Complex/ subdirectory of your configuration.

The complex flow solver will read the usual real-valued grid files and can compute derivatives of every variable with respect to Mach number, angle of attack, non-inertial rotation rates, or the x, y, or z coordinate of a single grid point. This choice is controlled by the file `perturb.dat`, a template of which is provided in the `FUN3D_90` directory. This file also specifies the step size to be used for the complex perturbation—a value of 1.e-30 or smaller is recommended. The smallest perturbation size supported by most machines/compilers is on the order of 1.e-308.

To compute derivatives with respect to a parameterized variable (i.e., MASSOUD or band-aid variables), a complex-valued grid must first be generated. Given a set of sensitivities for the surface mesh, the complex mesh movement code will propagate the sensitivity information out into the field, and dump out a complex-variable form of the grid partition files. At this point, the user can then run the complex flow solver to obtain derivatives of flow-field quantities with respect to the original design variable for which sensitivities were provided. For more help in executing the complex form of the codes, contact FUN3D Support.

Previous (C7: Adaptation and Error Estimation) | Up | Next (C9: Appendix)

Today's NASA Official: Dana Hammond, a member of The FUN3D Development Team
Contact: FUN3D-support@lists.nasa.gov
NASA Privacy Statement