eddy Users Manual

Scott M. Murman, Laslo T. Diosady, Anirban Garai, Corentin Carton de Wiart, Patrick J. Blonigan, and Dirk Ekelschot

NASA Ames Research Center, Moffett Field, CA, USA

June, 2018

Contents

Chapter 1

Install

1.1 Introduction

eddy is a collection of tools - nonlinear solvers, meshing, post-processing, visualization, optimization, *etc.* - for performing scale-resolving simulations of multi-physics applications. The framework is designed to enable advanced R&D on a variety of topics by leveraging a mature capability for scale-resolving simulations, and simultaneously be an appropriate tool for application analysis and support. Currently, *eddy* is at a relatively low technical-readiness level (TRL), and users and developers should maintain appropriate expectations.

The technical details behind *eddy* are outlined in several publications which can be consulted for more information [1–10]. The solvers are built around an unstructured high-order capability, and heavily utilize the tensor-product sum-factorization approach for efficiency. The unsteady formulation utilizes a fully implicit space-time approach with a matrix-free Newton-Krylov method. A primitive steady-state solver is available for testing purposes, but is not expected to converge for all but simple verification cases. The Navier-Stokes fluid solvers do not support either RANS or hybrid-RANS capability, only LES and wall-modeled LES approaches.

All of the solvers within *eddy* support three modes of operation: a *primal* solve of the full nonlinear problem, and two linearization approaches of the primal solve - the *adjoint* and the *tangent* solution. Details on how to select and use these three modes are outlined in Sec. 3.

1.2 Compiling

eddy is written in C, and relies upon several 3rd-party libraries listed below,

- MPI
- hdf5 - must set -enable-parallel
- parmetis
- glib
- libyaml
- cgns - must set CGNS_ENABLE_64BIT, CGNS_ENABLE_HDF5, CGNS_ENABLE_SCOPING, and HDF5_NEED_MPI
- visit (optional)

In addition, *eddy* builds an optimized library to support the tensor-product operations. This is located in the *gen* directory and must be compiled and installed before building *eddy* .

eddy uses a standard Makefile build system. Machine-dependent customizations, such as the location of the libraries listed above, compiler options, *etc.* , are specified in a file *machine.mk* in the *src* sub-directory. Examples of several customizations are provided for OSX, linux, *etc.* and these can be copied or linked to *machine.mk*. Running make install will locate binaries and libraries in standard *bin*, *lib*, *include* paths relative to the top-level directory.

All of the binaries built within eddy support the --help command-line option which will list a description of the program, the available command-line options, and immediately exit.

1.3 Conventions and Nomenclature

All of the input files for *eddy* are in YAML format, to support syntax highlighting and provide a structure for the input data.

All of the inputs in the YAML file can be parsed with a (relatively) general function parser. For example, it is possible to define variables, and then use those variables in mathematical expressions, as in

```
Mach: 0.69
Alpha: 5*pi/180.0
Velocity:
Uref: Mach*cos(Alpha)
Vref: Mach*sin(Alpha)
Wref: 0
```

The output that the executable translates this to, ignoring the temporary variables, is printed to standard output. All of the variables that can be set in the YAML input file, including any unset defaults, are printed to standard output.

eddy is a multi-physics solver, and each set of different physics (governing equations) is termed a *region*. Thus the computational *domain* is split into multiple *regions*, and then the separate *regions* are coupled together. Further, each *region* may require the specification of physical boundary conditions. A single physics simulation follows this same paradigm, specifying a single *region* without any couplings specified.

The binary data for the mesh, simulations reults, *etc.* is stored in specialized HDF5 files. Currently, a single mesh, flow solution, *etc.* is created for each physics *region*, as outlined in Sec. 2.

Chapter 2

Pre-processing

2.1 Defining a Mesh

In order to define a mesh we need:

1. information about the topology/connectivity which describes how difference entities (vols/faces/edges/vertices) are connected to one another

2. the coordinates which describe the actual geometry of the problem at hand

For high-order meshing it is convenient to think of these as two somewhat separate things.

Within eddy we describe the mesh in terms of a topology/connectivity and an *oracle* which we can query to obtain geometry information to whatever fidelity is required (i.e. high-order geometry representation).

There are currently two paths for generating meshes that eddy can understand

- using a 2nd-order unstructured mesh

- building a higher-order mesh based on a structured *oracle*

2.1.1 Unstructured Mesh

Currently the only support for general unstructured meshes is through reading unstructured CGNS files. In the eddy/bin directory there is a utility

cgns2mesh which converts an unstructured cgns mesh to an eddy supported mesh.h5 file. Currently only single zone, linear, 3D, unstructured meshes are supported at this time. There is no oracle (i.e. the oracle is an identity mapping).

2.1.2 Higher-order meshes

In order to generate curved higher-order meshes we use an oracle based on a structured multi-block mesh. The process of generating an oracle involves two steps:

1. calling **cgns2oracle** to generate an oracle.h5
2. calling **oracle2mesh** to generate a mesh.h5.

cgns2oracle takes the following arguments

- -i inputfile : the filename of the input cgns file containing the multiblock grid
- -o outputfile: the file to write the oracle to (usually oracle.h5)
- –PerJ/–PerK/–PerL : flags indicating periodic boundary conditions on all blocks in the J/K/L direction.

Additionally cgns2oracle reads inputs from the file oracle.yaml. The additional inputs that can be read are additional periodic connectivities which can be specified using the following:

```
- connection
  block_one: name1
  con_plane_one: 1
  block_two: name2
  con_plane_one: -1
  orientation: 0
  is_periodic: 1
```

A block of this form must be added for every desired additional connection. Here name1/name2 correspond to the zone names in the cgns file, plane1/plane2 correspond to the direction index of the normal (i.e. 1/2/3 -> J/K/L) while the sign denotes the start/end. Orientation is a number from

[0,7] which denote the relative orientations of the two faces of the blocks. Periodic is set to 1 if this is a periodic boundary, or 0 if this involve a one-way coupling from block1 to block2.

oracle2mesh creates mesh.h5 starting from an oracle by splitting each zone into a number of hexahedral elements. The number of elements on each zone is specifies in the file mesh.yaml which has the following form:

```
- Zone : name1
  Nj : 2
  Nk : 3
  Nl : 4

- Zone : name2
  Nj : 3
  Nk : 4
  Nl : 1
```

where a "-Zone" must be specified for each block in the multi-block mesh, corresponding to the given name. Nj/Nk/Nl correspond to the number of elements in each direction. Hanging nodes are not allowed and some minimal error checking occurs to ensure that you have specified a valid mesh.

Note that at this point the coordinates of the higher-order mesh (and geometry order) have not been specified. These are determined at run time.

2.1.3 Mesh2Domain2Mesh

For multiphysics simulations we currently generate a separate mesh.h5 file for each region. Typically we start from a single mesh.h5 which we want to split. The utility which can accomplish this is called mesh2domain2mesh. The utility has two functions:

1. allows you to name topologies within a domain (which Pointwise can't seem to do for us)

2. write a mesh file for a single region.

The input for **mesh2domain2mesh** is the file mesh2domain2mesh.yaml and has the following form:

```
MeshFile: mesh.h5

Maps:
  - Map: surfacename1
    Parent1 : volumename1
    Parent2 : volumename2

  - Map: surfacename2
    Parent1 : volumename2
    Parent2 : volumename3

Regions:
  - Region: volumename2
```

MeshFile denotes the input file. The input file will read all "volume conditions" and "boundary conditions" that were specified in the cgns file. However, we may need to name certain interior boundaries or lower order entities (i.e. edges) which pointwise does not allow us to do. We name these entities by creating "maps" where each "map" corresponds to a newly named entity, which are defined as the intersection of two parents. A region can correspond to any map provided we have also created maps (i.e. named) all of its boundaries. For each region we construct a mesh_regionname.h5 file which is read by eddy. For example, the above input would create a file named mesh_volumename2.h5.

Chapter 3

Running eddy

The following section discusses how to run a case with *eddy* for a given mesh. The working directory in which you wish to run eddy should contain the mesh.h5 file(s) and the necessary YAML input files.

3.1 Input files

Running *eddy* requires at least two YAML input files. The first file needs to be named eddy.yaml and it should contain global information about the case you are running, such as the number of time iterations, time stepping scheme, and the name of the region(s) included in the domain.

Firstly, the temporal discretization needs to be specified in the **UnsteadyControl** section:

```
UnsteadyControl:
  Scheme: # Time Stepping Scheme.
  TimeStep: # Time step size.
  Iterations: # Number of time iterations.
```

The time stepping scheme options are currently **Steady**, **ExplicitRK**, **ImplicitRK**, or **SpaceTime**. The option **Steady** corresponds to a steady state solver and it ignores the **TimeStep** and **Iterations** inputs. Note that the space-time scheme performs an integral over a polynomial basis in the temporal direction of the space-time element, akin to the typical spatial finite-element scheme. As such, there is a distinction between data stored within the temporal projection of the space-time element, which we refer to

as a time slab and contains **Nt** volume states corresponding to the temporal dof, and the projection in the temporal direction to the end of the time slab, which contains a single volume state.

The other mandatory entry in eddy.yaml is the **Regions** section:

```
Regions:
  - Region: # Region Name (Mandatory)
```

Additional options under **Partition** depend on whether the domain is a single region or contains multiple regions. These will be discussed in sections 3.1.1 and 3.1.2.

In addition to the eddy.yaml file, one input file is needed for each region. This file should be named region-name.yaml, where region-name is the region name specified in eddy.yaml. The region input file region-name.yaml should contain information specific to a given region, such as the discretization scheme, the physics module to be used, and any boundary conditions.

The first entry to region-name.yaml should be the mesh file name:

```
MeshFile: # mesh file name
```

The discretization scheme is specifed as follows:

```
Discretization:
  DiscretizationType:
  BasisType:    # LagrangeGauss (default) or LagrangeGLL
  SpatialDealias: # 1.0 collocation (default), > 1.0 dealiasing
  TemporalDealias: # 1.0 collocation (default), > 1.0 dealiasing
  N: # number of 1D basis functions (p+1) ,
     # a/k/a spatial order of accuracy
  Nt: # number of temporal basis functions (only used for space-time)
  N_geom: # N (default) or some other order of accuracy
          # for the mesh curvature
  Nt_geom # Nt (default) or some other order of accuracy
          # for the mesh curvature in time
```

Note that the spatial and temporal orders of accuracy **N** and **Nt** are typically set to multiples of 2 since the low-level matrix math kernels used by *eddy* are optimized for these accuracy orders.

The **Equation** section specifies the physics solver for a region, sets its important parameters (under **ReferenceConditions**), and sets the initial conditions.

```
Equation:
  EquationType: # Physics module to be used
  ReferenceConditions: # Parameters for physics module
                       # (e.g. Reynolds Number)
  InitialConditions: # Specify Parameters and/or spatial functions,
                     # depending on physics module
  VolumeOutputs: # quantities of interest to be computed
                 # on the region volume (e.g. kinetic energy)
```

The entries in the **ReferenceConditions**, **InitialConditions** and **VolumeOutputs** sections depend on the physics module specified in **EquationType**. Consult chapter 5 for specific details on each physics module/solver. Finally, Boundary conditions are specified in the **BCs** section:

```
BCs:
  - Boundary: # Boundary name
    Type: # Boundary type (e.g. Dirichlet)
          # depends on physics module
    Outputs: # outputs to be computed on this boundary
             # (e.g. shear stess)
```

The boundary name in the YAML file needs to correspond to the boundary name in the mesh file. There should be a **Boundary** entry for each boundary in the region. Each boundary type may have additional input parameters. For example, the NavierStokes boundary condition **FullState** can specify all five primitive states (Rho,Uref,Vref,Wref,P) to over-ride the default values specified in the **ReferenceConditions** subsection of **Equation**.

3.1.1 Single region

To run *eddy* for a single region domain, only two input files are needed, eddy.yaml and region-name.yaml.

```
Regions:
  - Region: # Region Name (Mandatory)
    Partition: # Optional
      NProcs: # Number of cores to be used by this region
      DedicatedIO: # (Optional) Number of cores dedicated
                   # to concurrent IO (Default 0)
```

```
    DedicatedDiag: # (Optional) Number of cores dedicated
                   # to concurrent diagnostics (Default 0)
```

It is not necessary to specfy anything other than the region name after **- Region:** for a single region domain. However, if one wants to used dedicated cores for concurrent IO or diagnostics, the fields under **Partition** need to be specified. **NProcs** should be the total number of core to be used and must match the number of cores specified for MPI at run time. **DedicatedIO** and **DedicatedDiag** are the number of cores out of the **Nprocs** cores specified for the region that will run concurrent IO and diagnostics while the solver runs on the remaining cores.

3.1.2 Multiple regions

To run *eddy* for a domain with multiple regions, one eddy.yaml file is required, along with one region-name.yaml for each region. In eddy.yaml', the **Regions** section must have a **- Region** entry for each region with the following:

```
Regions:
  - Region: # Region Name (Mandatory)
    Partition:
      NProcs: # Number of cores to be used by this region
      DedicatedIO: # (Optional) Number of cores dedicated
                   # to concurrent IO (Default 0)
      DedicatedDiag: # (Optional) Number of cores dedicated
                     # to concurrent diagnostics (Default 0)
```

Each region must have a number of cores **NProcs** specified and the sum of the **NProcs** entry for each region must match the number of cores specified for MPI at run time. The entries **DedicatedIO** and **DedicatedDiag** are the number of cores out of the **Nprocs** cores specified for the region that will run concurrent IO and diagnostics while the solver runs on the remaining cores.

In addition to the entries discussed previously, eddy.yaml needs to include a **Couplings** section that specifies how the regions are coupled:

```
Couplings:
  - Coupling: # specify name here
    Type: # type of coupling
```

```
        # (e.g. Riemann for NavierStokes to NavierStokes)
    Region1: # Pair of regions to be coupled
    Region2: #
    Interface: # name of boundary or volume
               # on which coupling takes place
```

There should be one **- Coupling:** entry for each coupling between two regions.

Note that any boundaries that act as interfaces for a coupling should be included in the **BCs** section of the region-name.yaml file with **Type** set to **Coupling**:

```
BCs:
  - Boundary: # Interface boundary name
    Type: Coupling
```

3.1.3 Optional Inputs

Nonlinear and linear solver parameters can be specified in the **NonlinearSolver** section:

```
NonlinearSolver:
  LinearSolver: # linear solver type (Default GMRES)
  nNonlinearIter: # Maximum number of Newton iterations
                  # (Default 35)
  nGMRESInner: # Number of Linear solver iterations for
               # each Newton iteration (Default 100)
  NonlinearTol: # Absolute residual tolerance
                # (Default 1E-14)
  InitialGlobPar: # Initial values of globalization parameter
                  # (Default 1.0)
  LambdaIncrease: # Rate to increase globalization parameter by
                  # (Default 10.0)
  ForceUpdate: # Ignore convergence status of linear solver
               # (Default False)
```

Current linear solver options include GMRES, Conjugate Gradient (**ConjugateGradient**), and the Biconjugate gradient stabilized method (**BiCGStab**).

Parameters for computing and outputing Volume outputs, boundary outputs, and mean files (with time-averaged variables) can be specified in the **CheckpointControl** section:

```
CheckpointControl:
  WriteInterval: # Frequency that the solution at the end
                 # of a time slab should be written to disk
  TimeSlab: # Boolean: true if all time slabs should saved to disk
            # (Need to save these to run tangent or adjoint)
  AverageStart: # end of averaging window in eddy time units
  AverageEnd: # end of averaging window in eddy time units
  AverageWindow: # type of averaging window:
                 # Square, Hann, or HannSquare
```

Note that "*eddy* time units" refers to the time units used by the solver and should be consistent with the time step size specified in eddy.yaml under **TimeStep**. If **AverageStart** is specified, *eddy* will compute and save an hdf5 file to disk containing the mean of the quatities specified in the **VolumeOutputs** field of the **Equation** section in each region-name.yaml. If no **AverageEnd** is specified, the mean will be a running average.

3.2 Running a solver

The *eddy* solver executable must be run with MPI in the working directory containing the input files and the mesh file.

3.2.1 Eddy Command Line Arguments

By default, eddy will run from the initial condition specified in your input files for the number of iterations specified in eddy.yaml. These defaults can be overwritten using the arguments in table 3.1.
For example, to run *eddy* on 4 cores starting from step 10 and finishing on step 20:

```
mpiexec -np 4 eddy -r 10 -t 20
```

Flag	Short Flag	Description
ndt	t	Specify time step to run to
reload	r	Specify time step to restart from
adjoint	N/A	Run adjoint solver
tangent	N/A	Run tangent solver

Table 3.1: Command line arguments for *eddy* . Note that the *ndt* (*t*) flag overrides the number of iterations specified in the eddy.yaml file.

3.3 Test Cases

There are two main types of software testing distributed with eddy: unit tests which verify the implementation, and regression (QA) tests which examine algorithm behavior, model assumptions, timing, *etc.*

The units tests are run using the command **make check** from the src directory.

The regression tests use a script system to run the jobs in parallel. The cases directory contains the QA tests which are run. The directory utils/regression_test contains scripts which can run these cases interactively, or on NASA's pleiades supercomputer using PBS. Note that different hardware and compiler optimizations are likely to lead to floating-point differences in the results from these regression tests, but these machine precision differences are not of concern.

Chapter 4

Post-processing

This chapter describes the post-processing tools attached to *eddy* and how to extract data from your simulations. The first section lists all the ouputs and diagnostics available in eddy and how to specify them in the input file. The second section is dedicated to flow visualization and how to transform the high-order output fields into a format that can be read by standard flow visualization tools, such as FieldView, VisIt or ParaView.

4.1 Diagnostics

This section describes how to extract data from the simulation and write them to disk. These files can be monitor files (residual, volume integral, etc.) or files containing the full field (full time slab, instantaneous, or mean data).

4.1.1 Instantaneous data, restart files and full space-time solution

To export the solution at a given time, you need to specify the output frequency in the region yaml file, in the **CheckPointControl** section.

If you want to output the solution every 100 steps, it will give:

```
CheckpointControl:
  WriteInterval: 100
```

The solution obtained at the end of the time-slab will be stored using the following naming convention: *soln_region_000100.h5*, with **region** the name of the region. This file can then be use to restart the simulation using the *-r step* argument.

To export the full space-time solution (slab) at every time step, you need to add the following line in the eddy yaml file:

```
CheckpointControl:
  WriteInterval: 100
  TimeSlab: True
```

This for instance, will export the full slab every time step and the instantaneous solution every 100 steps. The full slab solution will be stored using the following naming convention: *slab_region_000001.h5*.

4.1.2 Data integrated in space and/or in time

Integral of data in space and/or in time on the volume or at a boundary can be specified in the region yaml file. When available for your equation type, you can specify the volume output groups you want to export using the **VolumeOutputs** key. For instance:

```
Equation:
  EquationType: NavierStokes
  ReferenceConditions:
    ...
  InitialConditions:
    ...
  VolumeOutputs: "Primitive, ReynoldsStress, Vorticity"
```

If available for your boundary conditions, you can also export quantities of interest on your boundaries by specifying them in the *region.yaml* file as well, by using the **Outputs** key in the boundary block. For instance:

```
BCs:
  - Boundary: wall
    Type: AdiabaticWall
    Outputs: "Geometry, Flux, y+"
```

For data integrated in time (objectives, mean fields, etc.), a windowing can be specified in the *eddy.yaml* file. Here is an example of input to define a window in the yaml file:

```
CheckpointControl:
  AverageStart: 10.5
  AverageEnd: 15.5
  AverageWindow: Square
```

When no window is specified, only data integrated in space will be exported.

For volume outputs, the temporal evolution of the integrated quantities will be written to a file *region.volume*, with *region* the name of your region. For boundary outputs, the data will be exported to a file *wall.boundary*, where in this case *wall* being the name of the boundary. For both the boundary and the volume outputs, the name of the quantities in the file will be written in the header. As we are using a space-time finite element method, the space integral quantities are computed at every temporal Gauss points of each time slab (using $2N_t$ pionts).

Specifying a window in the input file will give access to two more outputs. Firstly, a file containing quantities integrated in time and space will be created. The results are stored in a *.tavg* file (*e.g. region.volume.tavg, wall.boundary.tavg*, etc.), containing one sample per time slab (corresponding at the total integral at the end of the time slab). Note that the objective computed is the integral of the quantity of interest in time, not the average. Secondly, the temporal average of the quantities of interest will be exported using the same frequency as the solution/restart file. The average data, both on the volume and on the boundaries, will be stored in files using the following name convention: *mean_region_000100.h5* (here for the timestep 100). You can then visualize the average data using our post-processing tools (see next section).

4.1.3 Residual norm

When running a case, a residual file, called *eddy.resid* will systematically be created in order to check the convergence of the system. The file is structured as follow:

```
step 1 time 0.1
```

```
outer =   0,  Rnorm = 3.5717752097e-05, ...
gmres-iter = 0, Rnorm = 3.5717752097e-05
gmres-iter = 1, Rnorm = 2.3685488690e-05
...

outer =   1,  Rnorm = 3.4586197307e-05, ...
gmres-iter = 0, Rnorm = 3.4586197307e-05
gmres-iter = 1, Rnorm = 1.7008026260e-05
...

step 2 time 0.2
...
```

The *outer* lines represent the non-linear residual, the *gmres-iter* the linear residual. Some tools are available in the *eddy/utils* directory to process the residual file. For instance, the gnuplot script *liveplot.gnu* allows to plot in real time the evolution of the residual.

4.2 Flow visualization

To visualize a field, two utilities are available. The first one is called *plotter* and will convert a restart file or a mean file into the OVERFLOW file format, provided that the case has a structured multi-block *oracle*. The second tool, *plotvisit*, allows you to connect to a visit client, even remotely. You can then directly visualize the data on the supercomputer without downloading the files. Currently *plotvisit* is exceedingly slow for cutting planes, hence the continued support for the legacy *plotter* tool.

4.2.1 plotter

The *plotter* tool is part of the eddy executables. The options to run

```
Usage:
  plotter [OPTION?] Plotter

Help Options:
  -h, --help          Show help options
```

```
Application Options:
  -s, --SaveFiles     Save window to outputfile when timestepping
  -r, --reload        reload step (-1 for outputting mesh only)
  -m, --mean          process mean flowfied
  -a, --adjoint       load adjoint if it exists
  --output            output you desire to save in p3d format
  --dealias           output you desire to save in p3d format
```

The *–output* allows you to specify which type output group you want to export. Typically, for Navier-Stokes, the *Conservative* output group is specified and then the other quantities are reconstructed in FieldView or Paraview.

4.2.2 plotvisit

```
Usage:
  plotvisit [OPTION?] PlotVisit

Help Options:
  -h, --help          Show help options

Application Options:
  -s, --SaveFiles     Save window to outputfile when timestepping
  -r, --reload        reload step (-1 for outputting mesh only)
  -f, --frequency     frequency of outputs
  -m, --mean          process mean flowfied
  -a, --adjoint       load adjoint if it exists
  -w, --wait          wait 10 seconds in order to attach debugger
```

Chapter 5

Available solvers in eddy

This chapter briefly describes available solvers and their inputs in *eddy* .

5.1 Advection-diffusion

An advection-diffusion solver using a space–time DG and CG spectral–element method has been implemented.

The definitions of input parameters are as followed:

```
Equation:
  EquationType: #Available: AdvectionDiffusion_CG, AdvectionDiffusion_DG
  ReferenceConditions:
    U-Velocity: #Spatial and temporal expression of velocity in x-direction.
    V-Velocity: #Spatial and temporal expression of velocity in y-direction.
    W-Velocity: #Spatial and temporal expression of velocity in z-direction.
    Viscosity: #Value of viscosity
  InitialConditions: #Initial condition for the simulation.
    InitialSolution: #Spatial expression of initial solution. Default: 0.0
  VolumeOutputs: #Volume outputs. Available: State, Gradient, Error
BCs:
  - Boundary: #Name of the boundary. Set up at the mesh generation process.
    Type: #Boundary condition. Available: Dirichlet, Neumann, Periodic.
              #Default: Periodic
    Outputs: #Outputs of the boundary. Available: Geometry, Flux, BCState.
```

Definitions and inputs of available boundary conditions are as followed:

1. Dirichlet: It imposes solution at the boundary. Inputs are:

```
State: #Spatial and temporal expression of the solution.
    #Default: 0.0
```

2. Neumann: It imposes flux at the boundary. Inputs are:

```
Flux: #Spatial and temporal expression of the flux.
    #Default: 0.0
```

5.2 Navier-Stokes

The conservative form of compressible Navier-Stokes equations are solved using a space–time DG spectral–element method. By default, entropy–variable formulation is used to satisfy the second law of thermodynamics (under exact integration) discretely. A conservative–variable formulation is also available, but not all the functionality has been implemented. By default, inviscid fluxes are computed using the entropy–stable approach of Ismail and Roe, and the viscous fluxes are computed using an interior penalty method following Bassi and Rebay. Roe flux, Lax–Friedrichs flux, Central–difference flux etc. are also available. Further details can be found at [3].

The definitions of input parameters are as followed:

```
Equation:
  EquationType: NavierStokes
  ReferenceConditions:
    Dimension: #Dimension of the problem (2 or 3 dimensional).
          #Default: 3.
    Velocity: # Reference velocity. Ma_{ref} is computed using these.
    Uref: #x-component of reference velocity
    Vref: #y-component of reference velocity
    Wref: #z-component of reference velocity
    Reynolds: #Reynolds number per unit length (Ma_{ref}/\nu).
             #Required to compute kinematic viscosity.
             #For inviscid flow set it to 0.
    Prandtl: #Prandtl number. Default: 0.71
    Gamma: #Specific heat ratio. Default: 1.4
    Temperature: #Reference temperature.
```

```
            #Required to compute Sutherland Constant. Default: 288.
  TemperatureUnits: #Unit of Reference Temperature. Default: Kelvin
  VariableType: #Type of variable used to solve NS.
            #Available: Entropy, Conservative. Default: Entropy.
  InvJumpFluxType: #Type of inviscid flux.
            #Available: IsmailRoe, IsmailRoeNoDiss,LaxFriedrichs,
                #Central,Roe,DSMV_1,DSMV_3,DSMV_5.
                #Note all the fluxes are not available for Conservative variable
            # Default: IsmailRoe.
SourceFunction: #Volumetric source terms.
            #Available: Channel, Poiseuille. Default: None.
Re_tau: #Turbulent Reynolds number. Needed if SourceFuction: Channel.
InitialConditions: #Initial condition for the simulation.
       #Note that the function purser can handle simple function.
       #For complicated functions use InitialSolution.
  Density: #Spatial profile for density. Default: 1.0
  XVel: #Spatial profile for x-velocity. Default: 1.0
  YVel: #Spatial profile for x-velocity. Default: 0.0
  ZVel: #Spatial profile for x-velocity. Default: 0.0
  Pressure: #Spatial profile for pressure. Default: 1.0/1.4
  InitialSolution: #Hardcoded initial condition for specific profiles.
          # Available: Channel, Jet.
  JetDiameter: #Diameter of the jet at the inlet.
          #See "Turbulence" by Pope, Section 5.1
  JetVelocity: #Centerline velocity of jet at the inlet.
          #See "Turbulence" by Pope, Section 5.1
  JetOriginX:  #Self-similarity origin of jet in x-direction.
          #See "Turbulence" by Pope, Section 5.1
VolumeOutputWeight: #Weight function on the volume. Default: 1.0
VolumeOutputs: #Volume outputs. Available: Geometry, Conservative,
      #Entropy, Primitive, HITBudget, Isentropic, Vorticity, SecondMoment,
      #ThirdMoment, ReynoldsStress, VelocityGradients, Gradients1,
      #Gradients2, PressureStrain, DissipationTerms, PressureTransport,
      #KineticEnergy, AdjointL2Norm
TemporalOutputWeight: #Weight function on the volume. Default: 1.0
TemporalOutputs: #Temporal outputs. Available: Geometry,
      #Conservative, Entropy, Primitive, KineticEnergy
OutputXdir: #x-component of normal vector for Force output
```

```
      #computation of AdiabaticWall boundary condition. Default: 0.0
  OutputYdir: #y-component of normal vector for Force output
      #computation of AdiabaticWall boundary condition. Default: 0.0
  OutputZdir: #z-component of normal vector for Force output
      #computation of AdiabaticWall boundary condition. Default: 0.0
BCs:
  - Boundary: #Name of the boundary. Set up at the mesh generation process.
    Type: #Boundary condition. Available: FullState, FullStateRoundJet,
      #SlipWall, AdiabaticWall, IsothermalWall, PressureOutflow,
      #Riemann, RiemannRoundJet, Coupling, WallModel,
      #Periodic. Default: Periodic
    Outputs: #Outputs of the boundary. Available: Geometry,
      #Flux, y+, Force, FrictionForce, ForceVector for
      #AdiabaticWall and WallModel.
```

Definitions and inputs of available boundary conditions are as followed:

1. FullState: This uses numerical Riemann formulation to compute the fluxes at the freestream or inflow or outflow boundaries. Required inputs are:

```
Density: #Density at the boundary. Default: 1.0
Uref: # x-velocity at the boundary.
    #Default: Uref from ReferenceConditions.
Vref: # y-velocity at the boundary.
    #Default: Vref from ReferenceConditions.
Wref: # z-velocity at the boundary.
    #Default: Wref from ReferenceConditions.
Pressure: # pressure at the boundary.
   #Default: 1.0/Gamma
```

2. Riemann: This uses Riemann invariant formulation to compute the fluxes at the freestream or inflow or outflow boundaries. Required inputs are:

```
Density: #Density at the boundary. Default: 1.0
Uref: # x-velocity at the boundary.
     #Default: Uref from ReferenceConditions.
Vref: # x-velocity at the boundary.
```

```
     #Default: Vref from ReferenceConditions.
Wref: # x-velocity at the boundary.
     #Default: Wref from ReferenceConditions.
Pressure: # pressure at the boundary.
     #Default: 1.0/Gamma
```

3. FullStateRoundJet: This is similar to the FullState boundary condition with the jet profiles. Required inputs are:

```
Density: #Density at the boundary. Default: 1.0
JetOriginX: #Self-similarity origin of jet in x-direction.
       #Default: 1.0.
JetVelocity: #Jet centerline velocity at the boundary.
       #Default: Uref from ReferenceConditions.
JetDiameter: #Jet diameter at the boundary. Default:1.0.
Pressure: #pressure at the boundary. Default: 1.0/Gamma
IsInlet: #Boolean to flag to use similarity solution (FALSE)
     #or tanh profile (TRUE).
Epsilon: #Parameter for tanh profile
```

4. PressureOutflow: This uses Riemann invariant formulation and enforce prescribed pressure to compute the fluxes at the outflow boundaries. Required inputs are:

```
Density: #Density at the boundary. Default: 1.0
Uref: # x-velocity at the boundary.
    #Default: Uref from ReferenceConditions.
Vref: # y-velocity at the boundary.
   #Default: Vref from ReferenceConditions.
Wref: # z-velocity at the boundary.
    #Default: Wref from ReferenceConditions.
Pressure: # pressure at the boundary. Default: 1.0/Gamma
```

5. SlipWall: This enforces slip wall boundary condition with a ramp. Required inputs are:

```
RampTime: #Timescale of the ramp. Default: 0.0 (means don't use ramp)
FreeStreamU: # x-velocity at the boundary.
```

```
     #Default: Uref from ReferenceConditions.
FreeStreamV: # y-velocity at the boundary.
    #Default: Vref from ReferenceConditions.
FreeStreamW: # z-velocity at the boundary.
    #Default: Wref from ReferenceConditions.
```

6. AdiabaticWall: This enforces adiabatic wall boundary condition. Required inputs are:

```
HeatFlux: #Normalized wall heat flux. Default: 0.0
```

7. IsothermalWall: This enforces isothermal wall boundary condition. Required inputs are:

```
WallTemp: #Normalized wall temperature. Default: 1.0
```

5.2.1 Shock-capturing

An artificial viscosity method with a shock sensor for DG spectral–element method is currently getting implemented. Details of the shock capturing scheme can be found at [**?**].

5.2.2 Variational Multiscale Modeling (VMM)

Variational Multiscale Model is a reformulation of the Large Eddy Simulation (LES) methods, in which resolved and unresolved scales are computed by Galerkin projection operation instead of filtering operation. To apply the method in general complex flows a dynamic procedure is developed following Germano procedure from classical LES. The dynamic VMM approach for entropy stable DG spectral–element method is currently getting implemented. Details of the VMM approach can be found at [**?**].

5.2.3 Perfectly Matching Layer (PML)

Perfectly Matched Layer method solves another set of auxiliary equations to ensure minimal spurious reflections from the inflow and outflow boundary conditions. Details of the the PML technique can be found at [6].

Since PML inherits from Navier-Stokes, its inputs are same as Navier-Stokes with some additions. These additional inputs are defined as follows:

```
Equation:
  EquationType: NavierStokes_PML
  PML type: Type of PML to identify damping direction.
        #Available: x-pml, y-pml, z-pml, x-radial-pml.
        #Default: x-pml.
  Ma_x: Mean Mach number in the x-direction over the PML region.
  Ma_y: Mean Mach number in the y-direction over the PML region.
  Ma_z: Mean Mach number in the z-direction over the PML region.
  ...rest are same as Navier-Stokes
```

5.3 Wall model

An equilibrium wall model based on Reichardt velocity profile has been implemented using a space–time DG spectral–element method. Details can be found at [11].

Definitions of the input parameters are as followed:

```
Equation:
  EquationType: WallModel_Reichardt
  InitialState: #Initial guess for the friction velocity
  VolumeOutputs: #Volume outputs. Available: Geometry, Friction.
```

5.4 Linear Elasticity

A linear elasticity approach has been implemented using a space–time CG spectral–element method for moving domain, FSI and wall roughness applications. Details of the linear elasticity technique can be found at [12].

Definitions of the input parameters are as followed:

```
Equation:
  EquationType: Elasticity
  ReferenceConditions:
    YoungModulus: #Young modulus. Default: 10.0
    PoissonRatio: #Poissom ratio. Default: 0.3
    ScaleByJacobian: #To scale Young modulus by Jacobian.
          #Available: TRUE, FALSE
  InitialConditions: #Initial condition for the simulation.
```

```
    InitialDisplacement-X: #Spatial expression of initial displacement
         #in x-direction. Default: 0.0
    InitialDisplacement-Y: #Spatial expression of initial displacement
         #in y-direction. Default: 0.0
    InitialDisplacement-Z: #Spatial expression of initial displacement
         #in z-direction. Default: 0.0
BCs:
  - Boundary: #Name of the boundary. Set up at the mesh generation process.
    Type: #Boundary condition. Available: SpecifiedDisplacement,
         #SpecifiedNormalDisplacement, SpecifiedNormalDisplacementOnly,
         #SpecifiedTraction, Periodic. Default: Periodic.
    Outputs: #Outputs of the boundary. Available: Geometry, Displace,
        #Displace2 for SpecifiedNormalDisplacement,
        #SpecifiedNormalDisplacementOnly.
```

Definitions and inputs of available boundary conditions are as followed:

1. SpecifiedDisplacement: It enforces boundary displacement strongly. Inputs are:

   ```
   X-Displacement: #Spatial and temporal expression for displacement
       #in x-direction. Default: 0.0
   Y-Displacement: #Spatial and temporal expression for displacement
       #in y-direction. Default: 0.0
   Z-Displacement: #Spatial and temporal expression for displacement
      #in z-direction. Default: 0.0
   ```

2. SpecifiedNormalDisplacement: It enforces boundary normal displacement weekly. Tangential displacements are not enforced. Inputs are:

   ```
   NormalDisplacement: #Spatial and temporal expression for the
           #normal displacement. Default: 0.0
   ```

3. SpecifiedNormalDisplacementOnly: It enforces boundary normal displacement weekly. Tangential displacements are enforced to be zero in week sense. Inputs are:

   ```
   NormalDisplacement: #Spatial and temporal expression for the
        #normal displacement. Default: 0.0
   ```

4. SpecifiedTraction: It enforces traction force at the boundary weekly. Inputs are:

```
X-Traction: #Spatial and temporal expression for traction in
      #x-direction. Default: 0.0
Y-Traction: #Spatial and temporal expression for traction in
      #y-direction. Default: 0.0
Z-Traction: #Spatial and temporal expression for traction in
      #z-direction. Default: 0.0
```

5.5 Linear Shell

A structural solver based on linear-shell model has been implemented using a C^1–DG spectral–element method. Details of the shell model can be found at [13].

Definitions of the input parameters are as followed:

```
Equation:
  EquationType: LinearShell
  MaterialDensity: #Density of the material. Default: 1.25
  YoungModulus: #Young modulus of the material. Default: 178906.23535
  PoissonRatio: #Poisson ratio of the material. Default: 0.3
  Thickness: #Thickness of the material. Default: 0.12556
  InitialConditions: #Initial condition for the simulation.
    X-Velocity: #Spatial expression of velocity in x-direction.
        #Default: 0.0
    Y-Velocity: #Spatial expression of velocity in y-direction.
        #Default: 0.0
    Z-Velocity: #Spatial expression of velocity in z-direction.
        #Default: 0.0
    X-Displacement: #Spatial expression of displacement in x-direction.
        #Default: 0.0
    Y-Displacement: #Spatial expression of displacement in y-direction.
        #Default: 0.0
    Z-Displacement: #Spatial expression of displacement in z-direction.
        #Default: 0.0
  VolumeForcing: #Volumetric force
    X-Force: #Spatial and temporal expression of force in x-direction.
```

```
      #Default: 0.0
    Y-Force: #Spatial and temporal expression of force in y-direction.
      #Default: 0.0
    Z-Force: #Spatial and temporal expression of force in z-direction.
      #Default: 0.0
  VolumeOutputs: #Volume outputs. Available: Displacement,
    #Membrane, Bending, ShellNormal, Error, Loading
BCs:
  - Boundary: #Name of the boundary. Set up at the mesh generation process.
    Type: #Boundary condition. Available: SpecifiedDisplacement,
      #SpecifiedTraction, ClampedEnd, Periodic. Default: Periodic
    Outputs: #Outputs of the boundary. Available: Displacement.
```

Definitions and inputs of available boundary conditions are as followed:

1. SpecifiedDisplacement: It imposes displacement at the boundary. Inputs are:

   ```
   X-Displacement: #Spatial and temporal expression for displacement
     #in x-direction. Default: 0.0
   Y-Displacement: #Spatial and temporal expression for displacement
     #in y-direction. Default: 0.0
   Z-Displacement: #Spatial and temporal expression for displacement
     #in z-direction. Default: 0.0
   ```

2. SpecifiedTraction: It imposes traction force at the boundary. Inputs are:

   ```
   X-Traction: #Spatial and temporal expression for traction
     #in x-direction. Default: 0.0
   Y-Traction: #Spatial and temporal expression for traction
     #in y-direction. Default: 0.0
   Z-Traction: #Spatial and temporal expression for traction
     #in z-direction. Default: 0.0
   ```

3. ClampedEnd:

5.6 6DoF

A six-dof solver has been implemented using a space–time DG spectral–element method.

Definitions of the input parameters are as followed:

```
Equation:
  EquationType: SixDof
  ReferenceConditions:
    Mass: #Mass. Default: 1.0
    X-MomentofInertia: #x-component of moment of inertia. Default: 1.0
    Y-MomentofInertia: #y-component of moment of inertia. Default: 10.0
    Z-MomentofInertia: #z-component of moment of inertia. Default: 100.0
  InitialConditions: #Initial condition for the simulation.
    X-Position: #x-position. Default: 0.0
    Y-Position: #y-position. Default: 0.0
    Z-Position: #z-position. Default: 0.0
    X-Velocity: #x-velocity. Default: 0.0
    Y-Velocity: #y-velocity. Default: 0.0
    Z-Velocity: #z-velocity. Default: 0.0
    X-AngularVelocity: #x-angularvelocity. Default: 0.0
    Y-AngularVelocity: #y-angularvelocity. Default: 0.0
    Z-AngularVelocity: #z-angularvelocity. Default: 0.0
    X-Axis: #x-axis. Default: 0.0
    Y-Axis: #y-axis. Default: 0.0
    Z-Axis: #z-axis. Default: 1.0
    Angle: #angle. Default: 0.0
  VolumeOutputs: #Volume outputs. Available: State, Error
```

Bibliography

[1] L. Diosady and S. Murman, "Design of a Variational Multiscale Method for Turbulent Compressible Flows," AIAA Paper 2013-2870, June 2013.

[2] L. Diosady and S. Murman, "DNS of Flows over Periodic Hills using a Discontinuous Galerkin Spectral-Element Method," AIAA Paper 2014-2784, June 2014.

[3] L. Diosady and S. Murman, "Higher-Order Methods for Compressible Turbulent Flows Using Entropy Variables," AIAA Paper 2105-0294, 2015.

[4] Diosady, L.T. and Murman, S.M., "General element shapes within a tensor-product higher-order space-time discontinuous-Galerkin formulation," AIAA Paper 2015-3044, 2015.

[5] Ceze, M., Diosady, L.T., and Murman, S.M., "Development of a High-Order Space-Time Matrix-Free Adjoint Solver," AIAA Paper 2016-0833, 2016.

[6] Garai, A., Diosady, L.T., Murman, S.M., and Madavan, N., "Development of a Perfectly Matched Layer Technique for a Discontinuous-Galerkin Spectral-Element Method," AIAA Paper 2016-1338, 2016.

[7] Murman, S.M., Diosady, L.T., Garai, A., and Ceze, M., "A Space-Time Discontinuous-Galerkin Approach for Separated Flows," AIAA Paper 2016-1059, 2016.

[8] Garai, A., Diosady, L.T., Murman, S.M., and Madavan, N., "DNS of Flow in a Low-Pressure Turbine Cascade with Elevated Inflow Turbulence Using a Discontinuous-Galerkin Spectral-Element Method," in *Proceedings of ASME Turbo Expo 2016*, no. GT2016-56700, 2016.

[9] Diosady, L.T. and Murman, S.M., "Tensor-Product Preconditioners for Higher-Order Space-Time Discontinuous Galerkin Methods," *Journal of Computational Physics*, vol. 330, no. 1, pp. 296–318, 2017.

[10] Carton de Wiart, C., Diosady, L.T., Garai, A., Burgess, N.K., Blonigan, P., Ekelschot, D., and Murman, S.M., "Design of a modular monolithic implicit solver for multi-physics applications," AIAA Paper 2018-1400, 2018.

[11] Carton de Wiart, C., and Murman, S.M., "Assessment of Wall-modeled LES Strategies Within a Discontinuous-Galerkin Spectral-element Framework," AIAA Paper 2017-1223, 2017.

[12] Diosady, L.T., and Murman, S.M., "A linear-elasticity solver for higher-order space-time mesh deformation," AIAA Paper 2018-0919, 2018.

[13] Burgess. N. K., Diosady, L.T., and Murman, S.M., "A C^1-discontinuous-Galerkin Spectral-element Shell Structural Solver," AIAA Paper 2017-3727, 2017.