Delft University of Technology

Manual: MoC / Supersonic Stator / Design Chain Code

Authors	: Nitish Anand & Jozef Stuijt
Electronic mail	: nitish.ug2@gmail.com
Date	: 16th August 2016 (Last update 01-11-2018)

Abstract

Method of characteristics(MoC) is a marching type method which solves hyperbolic partial differential equations. This MoC code designs a de Laval Nozzle. The output of the MoC has been used to design a radial stator for ORC application. Details on the method can be found in the Master Thesis [1].

Contents

[1] email author

1 MoC Code

MOC_Config.in is the default configuration file for the MOC code. However, a different configuration file name can be given as argument during function call:

1.1 RUN SYNTAX:

```
nitish@NITISH:./MAIN <input_file_name>
```

1.2 Configuration file inputs:

1. **PLANAR/AXISSYM:** Specify if the nozzle is Axis-symmetric or Planar (Note: At the moment the code is only for Planar cases, but axis-symmetric is open for future application)

```
NozzleType <args>
#<args> = PLANAR / AXISSYM
```

2. **EoS:** Equation of state to be used for thermodynamic calculations.

PRFT	:Perfect Gas
EoS	:Van der Waals
EoS_TAB	:Van der Waals with tables
RefProp	:FluidProp
RefProp_TAB	:FluidProp with tables (not recommended)
CoolProp	:CoolProp

```
GAS_EQU <args>
# <args> = PRFT, EoS, EoS_TAB, RefProp, RefProp_TAB
```

3. **Total Conditions:** Specify total conditions. Required in all Gas Models
To: Total temperature in Kelvin
Po: Total pressure in Pa

```
To = <args>              # K
Po = <args>              # Pa
```

4. **GAS_EQN = PRFT:** Parameters required to use Perfect Gas Model.

```
gamma = <args>           # cnst
R = 8.314                # mol/Kg/K
M = <args>               # kg/mol
```

5. **GAS_EQN = RefProp:** Specify Fluid name from FluidProp file. Make sure *.FLD file is available in FluidProp library. Also specify from where the Throat Properties should be calculated, from 'FILE' or by 'ITER' (iteration).

```
FLDNAME <args>              # Specify *.FLD name
THROAT_PROP <args>          # <args> = ITER / FILE
```

6. **GAS_EQN = CoolProp:** Specify Fluid name from available fluids (see list on http://www.coolprop.org/) . Also specify from where the Throat Properties should be calculated, from 'FILE' (specify filename) or by 'ITER' (iteration).

```
FLDNAME <args>              # Specify FLD name
THROAT_PROP <args>          #<args> = ITER / file_name
```

7. **GAS_EQN = EoS :** Specify properties for Van der Waals equation of State.
()Guess = Specify guess value for the properties (specific volume, Temperature, Pressure)
()c = Specify critical temperature, pressure and specific volume.
()o = Specify Total Enthalpy and Entropy.

```
VrhoGuess = <args>          # m^3/mol
TGuess = <args>             # K
PGuess = <args>             # Pa

Tc = <args>                 # K
Pc = <args>                 # Pa
Vc = <args>                 # m^3/mol

Ho = <args>                 # J/Kg/K
so = <args>                 # J/Kg
```

8. **Table Control:** Specify the limit and delta for the table. LLim = Lower Limit velocity LLim = Upper Limit velocity dV = step of the velocity

```
Table_Oper Calc
LLim = <args>
ULim = <args>
dV = <args>
```

9. **Design Variables:** Refer the Figure 1 for details about the parameters.

```
rho_t = <args>
y_t = <args>
rho_d = <args>
Noz_Design_Mach = <args>
```

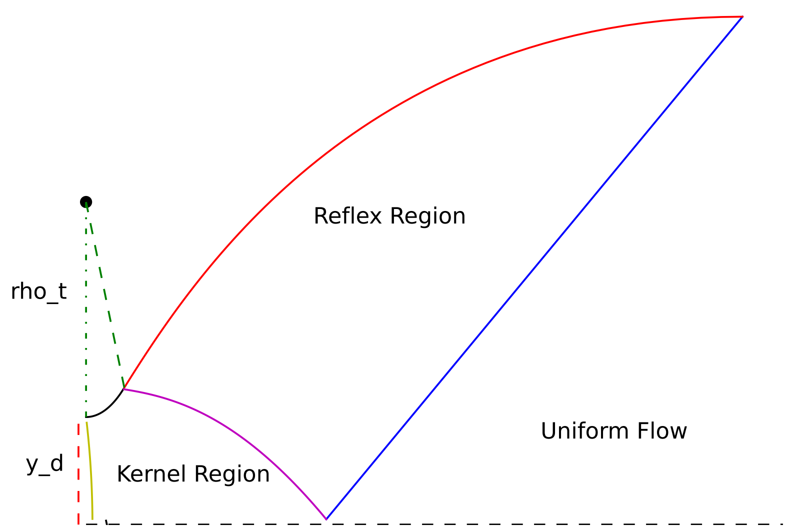

Figure 1: Nozzle Parametrization

- - - - - = Axis of symmetry
——— = Expansion Curvature
——— = Reflex Curvature
——— = Uniform property line
——— = Sauer Line
——— = Kernel to Reflex
- - - - - -= Radius along which the expansion region is defined
● = Center of rotation for creating expansion region

10. **Throat Parameter:** Specify the number of points on the throat. (Recommended: 20)

```
n = <args>
```

11. **Kernel Parameter:** Specify the differential angle at which the wall pressure waves should be calculated. (Recommended: 0.25)

```
dtau = <args>
```

12. Reflex Parameter: Specify the number of divisions in the reflex zone.

```
n_ref = <args>
```

13. **Convergence Criteria:** Specify the space and velocity convergence values. (Recommended: 1e-6)

```
tolerance_v <args>
tolerance_x <args>
```

14. **File Writing:** Specify the co-ordinate and property file name.

```
File_Name_NProp <name>.out
File_Name_NCods <name>.out
```

Sample Configuration file is available at: ”Link: Coming Soon”

1.3 Special Comments:

1.3.1 RefProp Runs:

When doing a RefProp run the properties at the Throat are determined using an iteration. Hence, it might take longer to initialize MoC compared to the other gas models.

2 Stator Design Code:

2.1 RUN SYNTAX:

SST_Config.in is the default configuration file for the SST code. However, a different configuration file name can be given as argument during function call:

```
nitish@NITISH:./SST <input_file_name>
```

2.2 Configuration file inputs:

The list of possible arguments which can be specified in configuration file to run the Stator tool are presented below.

1. **Blade Architecture:** Specify the blade architecture (axial or radial)

```
STATOR\_KIND <args>                 # <args> = AXIAL/RADIAL
```

2. **STATOR_KIND = RADIAL:** For a radial architecture the number of blades (to specify the pitch angle), inlet, blade outlet and rotor inlet radii have to be specified

```
nBlades     <args>        # <args> = Number of blades
ActualRin   <args>        # m
ActualRout  <args>        # m
RealRout    <args>        # m
```

3. **STATOR_KIND = AXIAL:** For an axial architecture the pitch angle needs to be specified

```
AxialPitch <args>        # deg
```

4. **Flow Angle:** Specify the gauging angle of the blades

```
flowAngle <args>         # deg
```

5. **Nozzle coordinate file:** Specify the MoC nozzle coordinates file

```
outFileMoc <args>        # file_name
```

6. **Trailing edge thickness:** Specify the thickness of the blade trailing edge

```
TEminT <args>   # m
```

7. **Coordinate file name:** Specify the Eucledian coordinate file output

```
CoordsName <args>        # file_name
```

8. **Specification file name:** Specify the blade specification file name

```
SpecsName <args>         # file_name
```

9. **Design mode:** Specify the program output (plotting or outfiles)

```
Mode <args>     # <args> = BLADE/SIM
```

10. **Mode = Sim:** In simulation mode outfiles are generated which can be used for CFD simulations. Currently only coordinate file for UMG2 generator is supported. It is necessary to specify a file name, scaling for the mesh generation and a reduction in coordinate points

```
ScaleMesh     <args>    # Scaling factor (Recommended 10)
nPointsScale  <args>    # Point reduction (Recommended 2)
UMG2Name      <args>    # file_name
```

11. **Mode = Blade:** In blade mode the output of the code will be plotted. This is useful when studying the geometry of the blades or making changes to the source code. Plots can be saved as image or shown on screen. The number of plotted blades can also be specified.

```
plotName     <args>     # <args> = show/file_name
nBldPlt      <args>     # number of blades shown in plot
```

Sample Configuration file is available at: ”Link: Coming Soon”

2.3 Special Comments:

UMG2 only seems to work when the blade coordinates are in the I and/or IV quadrant. Therefore depending on the original blade location, the angle argument in the function should be ”writeUMG2out” specified in order to rotate the blade to the correct region.

3 Design Chain

This design chain was implemented by Jozef Stuijt based on the original source code of Stephan Smit for the optimization of rotor blades. It currently supports blade generation and simulation based on the MoC Tool (nozzle generation), SST (blade generation), UMG2 (meshing) and SU2 (CFD solver).

Each process of designing and simulating a blade is called a "job". Simulation "runs", based on several jobs, can be performed in which a design parameter can be varied for a certain number of values within a specified range.

3.1 Setup

The directory structure of the code is as follows: within the main directory the "optimizationcode" directory, containing the scripts used to run the code, and "simulation" directory, where results are saved, are present. Within the optimizationcode directory, the configuration file templates and main code are present.

3.1.1 Configuration files

The design chain produces separate configuration files for each job which are stored in the corresponding job directory. These are produced based on python-script templates which have to be placed in the "config" directory under optimizationcode. Such a script should have the following structure:

```
from config import Config

class ClassName(Config):         # Give class specific name

        Config.__init__(self)
        self.name = name
        self.set_default_properties()
        self.excludeconfig = ['log_filename', 'config_filename',
        'name', 'output_filename', 'File_Name_Plot']

    def set_default_properties(self):
        self.log_filename = 'moc'+self.name+'.log'
        self.config_filename = 'moc'+self.name+'.cfg'

        # Parameters of configuration file can be defined here
        self.parameter = <args>
```

A template for each of the program to be used needs to exist. The vales of ¡args¿ will be the default values in the generated configuration files for the jobs, unless varied by the main script.

UMG2 requires certain configuration files to be run, "options", "topology", "geometry" and "spacingcontrol" (see UMG2 read me). These have to be stored in a directory named

"Db" in the job directory (this is done by the execution plan of the design chain). The SST tool generates the "geometry" file. All other files will be made from templates which have to be stored in "../optimizationcode/Data/Db". These are then changed by the UMG2 python script (see related section below).

3.1.2 Executables

SU2 needs to be installed in the machine in order to be able to run CFD simulations. Either the SU2 commands need to be added to the PATH, or the absolute path needs to be defined in the script "/optimizationcode/executionunits/su2executionunit.py", under class SU2ExecutionUnit as follows:

```
self.installdir="~/absolute/path/to/SU2"
```

A bash file named "UMG2.sh" to run the UMG2 executables in sequential order should be stored in the machine and its location should be added to the PATH.

All other executables should be stored in "executables" directory. Files that should be contained in this directory are:

1. moctool.py (MoC Tool main)
2. srcMOC (MoC Tool source files)
3. SST.py (Supersonic Stator main)
4. srcSST (SST source files)
5. SU2_PER
6. UMG2.py (UMG2 handling script)

3.2 RUN SYNTAX

To run the Design Chain it is necessary to have the terminal open at the main directory. The main script can then be called as follows:

```
nitish@NITISH:~/path/to/DesignChain$ python -m
optimizationcode.main
```

The main script currently has two main functions, mesh_convergence(args) and sim_range_sweep(args). The functions define which jobs will be carried out and these functions should be specified in the "__main__" function of the script. The syntax of both functions are explained in the subsections below.

Running the main script with one or more of the previous functions will produce runs, which will be stored in their own directory under the simulations directory. In turn each of the jobs of a run will be stored in its own subdirectory.

3.2.1 Mesh Convergence Function

This function was designed to easily vary mesh related parameters. The function has the following syntax

```
mesh_convergence(var_rng, var_type, nupr=7, cluster=True,
numit=None, blfac=None, tefac=None, nemel=None, Ma=None)
```

The arguments are:

1. var_rng: All the values of the parameter to be changes. Should be list or numpy array
2. var_type: Parameter to be changed. Currently supported: 'BL' (boundary layer factor), 'TE' (trailing edge radius factor), 'numel' (number of mesh elements).
3. nupr: Number of processors used for calculations
4. cluster: (True/False) Define if calculations are local or using slurm
5. numit: Change the number of iterations for all jobs
6. blfac: Change the boundary layer factor for all jobs
7. tefac: Change the trailing edge radius factor for all jobs
8. nemel: Change the number of elements for all jobs

3.2.2 Parametric Study Function

This function is the main function designed to carry out parametric studies of the design variables. It needs to be defined correctly in the main section of the main script. It has the following syntax:

```
sim_range_sweep(var_rng, var_nam = 'Mach', nupr = 7,
cluster=True, Pback = None, AreaRatio = None, NozMach = None,
RealRout = None, phi= None, Rout=None, mocconfig = None,
ThroatMoc = None, BC = 'Giles', Build = 'full',
filter_dups = True)
```

The arguments are:

1. var_rng: All the values of the parameter to be changes. Should be list or numpy array
2. var_nam: Parameter to be varied. Currently supported:
   - 'Mach': Nozzle exit Mach number (MoC)
   - 'flowAngle': Blade flow angle (SST)
   - 'Rout': Stator blade outlet radius (SST)
   - 'BackPressure': Pressure on the outflow boundary (SU2)

- ’RealRout’: Outflow boundary radial location (SST)
- ’Visc’: Constant viscosity value (SU2)

3. nupr: Number of processors used for calculations
4. cluster: (True/False) Define if calculations are local or using slurm
5. Pback: Change the pressure on the outflow boundary for all jobs
6. AreaRatio: Fix the throat-to-outflow area ratio for all jobs (needs to be checked)
7. NozMach: Change the nozzle exit Mach number for all jobs
8. RealRout: Change the outflow boundary location for all jobs
9. phi: Change the flow angle for all jobs
10. Rout: Change the blade outlet radius for all jobs
11. mocconfig: A MOCConfig object can be created and changed in the main script and used as input for all the jobs
12. ThroatMoc: Fix the throat length for all jobs (needs to be checked)
13. BC: Change the inflow/outflow boundary condition (’Giles’/’Riemann’)
14. Build: Chose the job type, only blade generation or blade and simulation (’full’/’blade’)
15. filter_dups: Avoid re-running existing jobs in an existing job folder (True/False). Use when defining a range which contains jobs which already have been carried out

See main file for example of functions used to carry out the runs laid in the thesis work ”Design Guidelines for Radial Supersonic Stators” by J.H. Stuijt Giacaman (add link)

3.3 Special Comments:

When running in the cluster make sure executables do not perform any plotting as this is usually not supported and will lead to run failure. Also, make sure the correct username and server are specified in the ”optimizationcode/jobs/secrets.py” file.

3.4 UMG2 Python Script

In order to provide flexibility with respect to meshing using UMG2, a specialized Python script was written to handle changes in meshes for the variations in blade geometries. The script can ensure similar meshes are used to maintain constant experimental conditions. Two minimization functions are used: one to change the number of elements of the mesh and another to improve the worst element of the mesh, which is expected to appear in the region the trailing edge.

It is important to have a working set of UMG2 configuration files in the ”Db” directory, as these will be used as templates and adapted to make the desired mesh. If the original files do not produce a mesh, the script will fail.

”Mesh_Config.in” is the default configuration file for UMG2.py. However, a different configuration file name can be given as an argument during function call:

```
nitish@NITISH:./UMG2 <input_file_name>
```

The list of possible arguments which can be specified in the configuration file to run the script are the following:

1. **UMG2 File Names:** Specify the UMG2 configuration files name (file extension). For example, if geometry file is called geometry.stator, stator should be the argument.

   ```
   FILE_NAME <args>          # file_name
   ```

2. **Blade specification file:** Specify if input blade specification file

   ```
   SPECS\_FILE <args>              # <args> = file_name/'NO'
   ```

3. **STATOR_KIND = 'NO':** If no specification file is defined, the trailing edge thickness, trailing edge coordinates and throat width have to be specified

   ```
   TE_thk    <args>        # m
   TE_coords <args>        # <args>=[x_coord,y_coord] (m)
   Throat    <args>        # m
   ```

4. **Boundary layer thickness type:** Define how to specify the boundary layer thickness, using an actual length, or a factor of the throat length

   ```
   BL_IN  <args>   # LEN/FAC
   BL_FAC <args>   # if BL_IN=FAC
   BL_H   <args>   # if BL_IN=LEN
   ```

5. **Trailing edge refinement radius:** Define the radius of the circle used to refine the region around the trailing edge as a factor of the trailing edge width

   ```
   RAD_FAC <args>
   ```

6. **BL thickness correction:** Define a correction scaling (should be removed from config and made an internal parameter)

   ```
   scale <args> # Recomended 10
   ```

7. **Mesh elements:** Define a minimum number of mesh elements. An optimizer will attempt to scale the element sizes in the ratio of the original spacingcontrol file to achieve a number close to this.

   ```
   MIN_NELEM <args>
   ```

Sample Configuration file is available at: ”Link: Coming Soon”

4 Updates

November 1st 2018: Added CoolProp to MoC, changed RST to SST section and added Design Chain section.