GridGen Documentation

Release 0.0.1

Sayop Kim

November 05, 2015

Contents

This documentation pages are made for CFD class at Geogia Tech in 2014 Spring. This is online available at http://gridgen.readthedocs.org

Author: Sayop Kim(sayopkim@gatech.edu)

Affilation: School of Aerospace Engineering, Georgia Institute of Technology

Contents

1.1 Project description

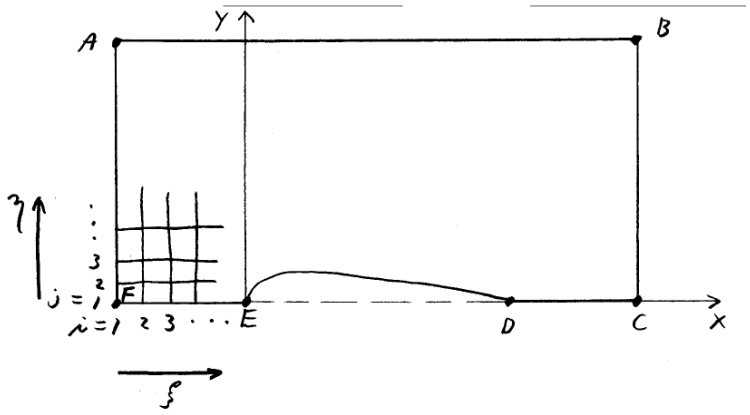
1.1.1 Given task

In this exercise you will generate an inviscid, 2-D computational grid around a modified NACA 00xx series airfoil in a channel. The thickness distribution of a modified NACA 00xx series airfoil is given by:

$$y(x) = \pm 5t[0.2969\sqrt{x_{int}x} - 0.126x_{int}x - 0.3516(x_{int}x)^2 + 0.2843(x_{int}x)^3 - 0.1015(x_{int}x)^4]$$

where the "+" sign is used for the upper half of the airfoil, the "−" sign is used for the lower half and $x_{int} = 1.008930411365$. Note that in the expression above x, y, and t represent values which have been normalized by the airfoil by the airf oil chord.

A sketch of the computational domain is shown below:



Each grid point can be described by (x, y) location or (i, j) location where i is the index in the ξ direction and the j index is in the η direction. The grid should have imax=41 points in the ξ direction and jmax=19 points in the η direction. The coordinates of points A-F shown in the figure are given in the following table:

Point	(i, j)	(x, y)	
A	(1,19)	(-0.8,1.0)	
B	(41,19)	(1.8,1.0)	
C	(41,1)	(1.8,0.0)	
D	(31,1)	(1.0,0.0)	(trailing edge)
E	(11,1)	(0.0,0.0)	(leading edge)
F	(1,1)	(-0.8,0.0)	

Algebraic Grid

To complete this project you will first generate a grid using algebraic methods. Use uniform spacing in the x direction along FE, along ED, and along DC. (However, note that the spacing in the x direction along FE and DC will be different from the spacing along ED). Use uniform spacing in the x direction along AB. (However, note that the spacing in the x direction along AB will be different from that along FE, ED, and DC). For the interior points of the initial algebraic grid use a linear interpolation (in computational space) of the boundary x values:

$$x(i,j) = x(i,1) + \left(\frac{j-1}{jmax-1}\right)[x(i,jmax) - x(i,1)]$$

Use the following stretching formula to define the spacing in the y direction:

$$y(i,j) = y(i,1) - \frac{y(i,jmax) - y(i,1)}{C_y} ln\left[1 + (e^{-C_y} - 1)\left(\frac{j-1}{jmax-1}\right)\right]$$

where C_y is a parameter that controls the amount of grid clustering in the y-direction. (If nearly uniform spacing were desired we would use C_y = 0.001).

The algebraic grid generated now serves as the initial condition for the subroutines which generate the elliptic grid. The bounday values of the initial algebraic grid will be the same as those of the final elliptic grid.

Elliptic Grid

The elliptic grid will be generated by solving Poisson Equations:

$$\xi_{xx} + \xi_{yy} = P(\xi,\eta)$$
$$\eta_{xx} + \eta_{yy} = Q(\xi,\eta)$$

where the source terms,

$$A_1(x_{\xi\xi} + \phi x_\xi) - 2A_2 x_{\xi\eta} + A_3(x_{\eta\eta} + \psi x_\eta) = 0$$
$$A_1(y_{\xi\xi} + \phi y_\xi) - 2A_2 y_{\xi\eta} + A_3(y_{\eta\eta} + \psi y_\eta) = 0$$

where the A's must be defined by mathmatical manipulation.

On the boundaries, ϕ and ψ are defined as follows:

$$\text{On } j = 1 \text{ and } j = jmax: \ \phi = \begin{cases} -\frac{x_{\xi\xi}}{x_\xi} & \text{if } |x_\xi| > |y_\xi| \\ -\frac{y_{\xi\xi}}{y_\xi} & \text{if } |x_\xi| \le |y_\xi| \end{cases}$$

$$\text{On } i = 1 \text{ and } i = imax: \ \psi = \begin{cases} -\frac{x_{\eta\eta}}{x_\eta} & \text{if } |x_\eta| > |y_\eta| \\ -\frac{y_{\eta\eta}}{y_\eta} & \text{if } |x_\eta| \le |y_\eta| \end{cases}$$

At interior points, ϕ and ψ are found by linear interpolation (in computational space) of these boundary values. For example,

$$\psi_{i,j} = \psi_{1,j} + \frac{i-1}{imax-1}(\psi_{imax,j} - \psi_{1,j})$$

1.1.2 Challenges

Demonstrate your solver by generating 5 grids (each with 41x19 grid points):

Grid #1

Initial algebraic grid, non-clustered (C_y = 0.001)

Grid #2

Initial algebraic grid, clustered (C_y = 2.0)

Grid #3

Elliptic grid, clustered (C_y = 2.0), no control terms ($\phi = \psi = 0$)

Grid #4

Elliptic grid, clustered (C_y = 2.0), with control terms

Grid #5

Now, use your program to generate the best grid you can for inviscid , subsonic flow in the geometry shown. You must keep imax = 41, jmax = 19 and not change the size or shape of the outer and wall boundaries. You may, however, change the grid spacing along any and all of the boundaries and use different levels of grid clustering wherever you think it is appropriate.

1.2 Code development

The current project is for developing elliptic grid generator in 3-dimensional domain. Hereafter, the program developed in this project is called 'GridGen'.

1.2.1 GridGen Code summary

The present project is to make a grid-generator for 3-D computational domain around a modified NACA 00xx series airfoil in a channel. The assigned project is inherently aimed at 2-D grid. However, the currently built GridGen code has a capability of 3-D grid generation.

The source code contains two directories, 'io', and 'main', for input/output related sources and grid-setup related sources, respectively. 'CMakeLists.txt' file is also included for cmake compiling.

```
$ cd GridGen/CODEdev/src/
$ ls
$ CMakeLists.txt  io  main
```

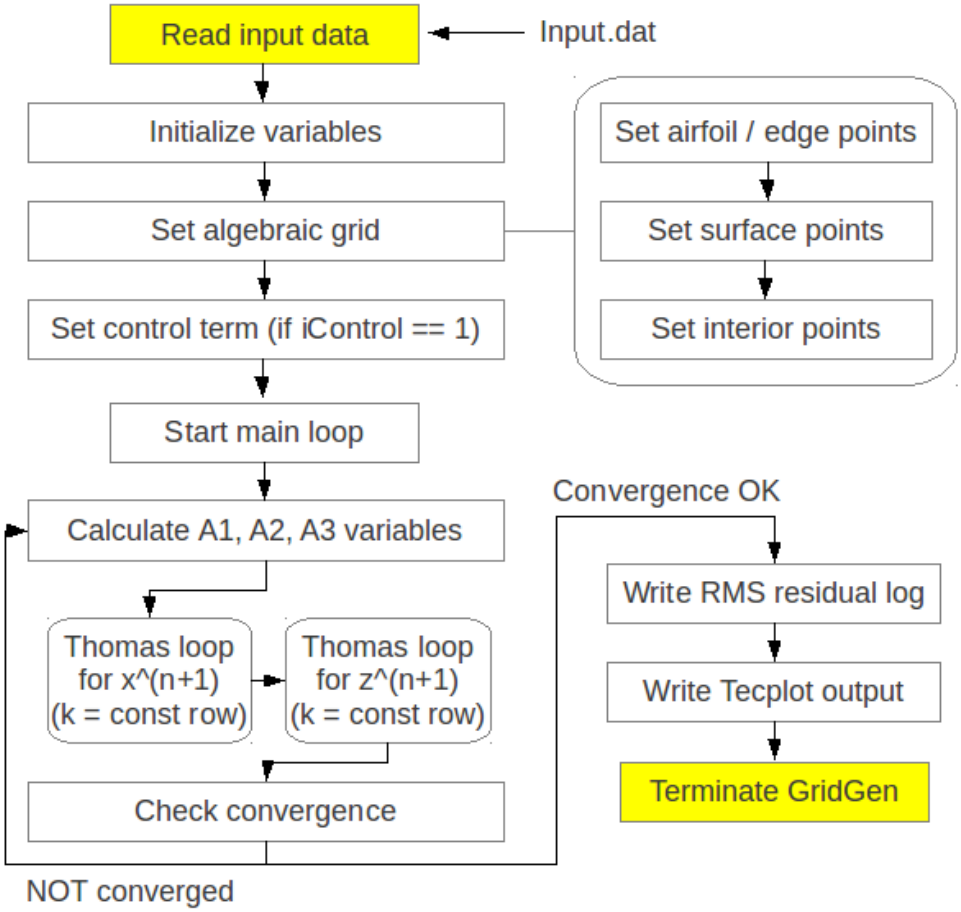
The **io** folder has **io.F90** file which contains **ReadGridInput()** and **WriteTecPlot()** subroutines. It also includes **input** directory which contains default **input.dat** file.

The **main** folder is only used for containing grid-setup related source files. The main routine is run by **main.F90** which calls important subroutines from **main** folder itself and **io** folder when needed. All the fortran source files **main** folder contains are listed below:

```
> GridSetup.F90
> GridTransform.F90
> GridTransformSetup.F90
> main.F90
> Parameters.F90
> SimulationSetup.F90
> SimulationVars.F90
```

1.2.2 Details of GridGen development

The GridGen code is made for creating 3-D computational domain with pre-described points value along the 2D airfoil geometry. The schematic below shows the flow chart of how the GridGen code runs.



The source code shown below is **main.F90** and it calls skeletal subroutines for generating grid structure. The main features of the main code is to (1) read input file, (2) make initialized variable arrays, (3) set initial algebraic grid points, (4) create elliptic grid points, and (5) finally write output files:

```
PROGRAM main
   USE SimulationSetup_m, ONLY: InitializeCommunication
   USE GridSetup_m, ONLY: InitializeGrid
   USE GridTransform_m, ONLY: GridTransform
   USE io_m, ONLY: WriteTecPlot, filenameLength
   USE Parameters_m, ONLY: wp
```

```
   IMPLICIT NONE

   CHARACTER(LEN=filenameLength) :: outputfile = 'output.tec'

   CALL InitializeCommunication
   ! Make initial condition for grid point alignment
   ! Using Algebraic method
   CALL InitializeGrid
   ! Use Elliptic grid points
   CALL GridTransform
   CALL WriteTecPlot(outputfile,'"I","J","K","Jacobian"')
END PROGRAM main
```

Creation of algebraic grid points

The code starts to run by reading the important input parameters defined in **input.dat** file. The input data file first contains the number of i, j, k directional grid points. Then the code reads airfoil geometry data from this input file, which provides the bottom edge points of the domain. The input file also contains four vertex points in (x, y, z) coordinates. Thus those points forms a 2-dimensional surface, which is supposed to be created in this project. Next, the code clones these grid points and locates them away from this surface in j-direction, resulting in 3-dimensional computational domain. Based on these boundary grid points, the code runs with Algebratic grid generating subroutine and gives initial conditions for elliptic solution for grid transformation.

The **main.F90** file first refers to **InitializeGrid** subroutine defined in **GridSetup.F90** file. The main function of this routine is to call again multiple subroutines defined in same file. The subroutine definition shown below summarizes the how the code runs for the grid initialization:

```
!------------------------------------------------------------------------------!
   SUBROUTINE InitializeGrid()
!------------------------------------------------------------------------------!
   USE io_m, ONLY: ReadGridInput
   USE SimulationVars_m, ONLY: imax, jmax, kmax,&
                               xblkV, cy
   IMPLICIT NONE

   ! Create Bottom Edge coordinate values
   CALL ReadGridInput
   CALL InitializeGridArrays
   CALL CreateBottomEdge
   CALL SetEdgePnts
   CALL GridPntsAlgbra
   CALL GenerateInteriorPoints

   END SUBROUTINE
```

- **ReadGridInput**: Reads important user defined variables and parameters for grid configuration.
- **InitializeGridArrays**: Initialize the single- and multi-dimensional arrays and set their size with input parameters(for example, imax, jmax, kmax).
- **CreateBottomEdge**: Generate point values for airfoil geometry.
- **SetEdgePnts**: Generate grid points along 8 edges of the computational domain.
- **GridPntsAlgbra**: Based on the edge points, this routine will distribute grid points located on each 6 surfaces of the computational domain.
- **GenerateInteriorPoints**: Based on grid points along the edges and surfaces, this routine will create interior grid points that are aligned with user-defined grid point interpolations.

Creaction of elliptic grid points

In order to determine the elliptic grid points with the pre-specified boundary points, the following Poisson equations, which is given in previous **Project description** section, have to be resolved numerically. The coefficients of the equations can be determined by:

$$A_1 = x_\eta^2 + y_\eta^2$$
$$A_2 = x_\xi x_\eta + y_\xi y_\eta$$
$$A_3 = x_\xi^2 + y_\xi^2$$

Then, applying finite difference approximation to the governing equations can be transformed into the linear system of equations. The arranged matrix form of equations shown below can be solved for unknown implicitly at every pseudo-time level. At every time loop, the code updates the coefficients composed of ϕ and ψ, and adjacent points. The detailed relations of each coefficients are not shown here for brevity.

$$a_{i,j}x_{i-1,j}^{n+1} + b_{i,j}x_{i,j}^{n+1} + c_{i,j}x_{i+1,j}^{n+1} = d_{i,j}$$
$$e_{i,j}y_{i-1,j}^{n+1} + f_{i,j}y_{i,j}^{n+1} + g_{i,j}y_{i+1,j}^{n+1} = h_{i,j}$$

Above equations can be numerically evaluated by the following descritized expressions:

$$a_{i,j} = e_{i,j} = A_{1\,i,j}^n \left(1 - \frac{\phi_{i,j}^n}{2}\right)$$
$$b_{i,j} = f_{i,j} = -2\left(A_{1\,i,j} + A_{3\,i,j}\right)$$
$$c_{i,j} = g_{i,j} = A_{1\,i,j}^n \left(1 + \frac{\phi_{i,j}^n}{2}\right)$$

$$e_{i,j} = \frac{A_{2\,i,j}^n}{2}\left(x_{i+1,j}^n - x_{i+1,j-1}^{n+1} - x_{i-1,j+1}^n - x_{i-1,j-1}^{n+1}\right) - A_{3\,i,j}^n\left(x_{i,j+1}^n + x_{i,j-1}^{n+1}\right) - \frac{A_{2\,i,j}^n}{2}\psi_{i,j}^n\left(x_{i,j+1}^n - x_{i,j-1}^{n+1}\right)$$
$$h_{i,j} = \frac{A_{2\,i,j}^n}{2}\left(y_{i+1,j}^n - y_{i+1,j-1}^{n+1} - y_{i-1,j+1}^n - y_{i-1,j-1}^{n+1}\right) - A_{3\,i,j}^n\left(y_{i,j+1}^n + y_{i,j-1}^{n+1}\right) - \frac{A_{2\,i,j}^n}{2}\psi_{i,j}^n\left(y_{i,j+1}^n - y_{i,j-1}^{n+1}\right)$$

where n and $n+1$ indicate pseudo time index. Thus above equations will update grid point coordinates for $n+1$ time level by referring to already resolved n time level solution. Note that the pseudo time looping goes along the successive j-constant lines. Therefore, when writing the code, time level index in above equations was not considered as a separate program variable because $j-1$ constant line is already updated in the previous loop.

The expressions above are only evaluted in the interior grid points. The points on the boundaries are evaluated seprately by applying given solutions as problem handout.

Once initial algebraic grid points are created, the code is ready to make elliptic grid points with some control terms in terms of ϕ and ψ. **GridTransform.F90** file contains a subroutine named by **GridTransform** as shown below:

```
!------------------------------------------------------------------------------!
   SUBROUTINE GridTransform()
!------------------------------------------------------------------------------!
   IMPLICIT NONE
   INTEGER :: n

   CALL InitializeArrays
   IF ( iControl == 1) CALL CalculatePiPsi
   DO n = 1, nmax
      CALL CalculateA123
      CALL ThomasLoop
      CALL WriteRMSlog(n,RMSlogfile)
      IF (RMSres <= RMScrit) EXIT
   ENDDO
   CALL CopyFrontTOBack
```

```
   CALL GenerateInteriorPoints
   CALL CalculateGridJacobian
   END SUBROUTINE GridTransform
```

Before going into the main loop for solving poisson equations, the code calculate control terms with ϕ and ψ. Even though the assigned project made an assumption of linear interpolated distribution of ϕ and ψ at interior points, the GridGen code is designed to allow ϕ and ψ be weighted in j and i directions, respectively. This effect is made by the grid stretching formula. This will be revisited for discussion on **Grid 5**.

Here, main DO-loop routine goes with setup of coefficients of governing equations and Thomas loop. The Thomas loop operates with line Gauss-Siedel method for resolving unknown variables, x and y, with tri-diagonal matrix of coefficients of finite difference approximation equation in a k = constant line. Note that the GridGen code transforms the grid points with elliptic solution only in front surface, then clones the grid points to the back surface and finally creates interior points. The front surface is made up of i and k coordinates.

Write Convergence history: RMS residual

In order to avoid infinite time-looping for the Thomas method, the GridGen code employs the following definition of RMS residual based on the new $(n+1)$ and old(n) values of grid point coordinates.

$$\text{RMS}^n = \sqrt{\frac{1}{N}\sum_{i=2}^{imax-1}\sum_{jmax-1}^{j=2}\left[\left(x_{i,j}^{n+1}-x_{i,j}^{n}\right)^2+\left(y_{i,j}^{n+1}-y_{i,j}^{n}\right)^2\right]}$$

where $N = 2x(\text{imax}-2)x(\text{jmax}-2)$ and the RMS criterion is pre-specified as: $1\text{x}10^{-6}$. In this code, the convergend is assumed to be achived when RMS residual is less than the RMS criterion.

1.3 How to run the code

1.3.1 Machine platform for development

This Grid Generation code has been developed on personal computer operating on linux system (Ubuntu Linux 3.2.0-38-generic x86_64). Machine specification is summarized as shown below:

vendor_id : GenuineIntel

cpu family : 6

model name : Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz

cpu cores : 4

Memory : 16418112 kB

1.3.2 Code setup

The GridGen source code has been developed with version management tool, GIT. The git repository was built on 'github.com'. Thus, the source code as well as related document files can be cloned into user's local machine by following command:

```
$ git clone http://github.com/sayop/GridGen.git
```

If you open the git-cloned folder **GridGen**, you will see two different folder and README file. The **CODEdev** folder contains again **bin** folder, **Python** folder, and **src** folder. In order to run the code, user should run **setup.sh** script in the **bin** folder. **Python** folder contains python script that is used to postprocess RMS residual data. It may contain **build** folder, which might have been created in the different platform. Thus it is recommended that user should remove **build** folder before setting up the code. Note that the **setup.sh** script will run **cmake** command. Thus, make sure to have cmake installed on your system:

```
$ rm -rf build
$ ./setup.sh
-- The C compiler identification is GNU 4.6.3
-- The CXX compiler identification is GNU 4.6.3
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- The Fortran compiler identification is Intel
-- Check for working Fortran compiler: /opt/intel/composer_xe_2011_sp1.11.339/bin/intel64/ifort
-- Check for working Fortran compiler: /opt/intel/composer_xe_2011_sp1.11.339/bin/intel64/ifort  -- w
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Checking whether /opt/intel/composer_xe_2011_sp1.11.339/bin/intel64/ifort supports Fortran 90
-- Checking whether /opt/intel/composer_xe_2011_sp1.11.339/bin/intel64/ifort supports Fortran 90 -- y
-- Configuring done
-- Generating done
-- Build files have been written to: /data/ksayop/GitHub.Clone/GridGen/CODEdev/bin/build
Scanning dependencies of target cfd.x
[ 12%] Building Fortran object CMakeFiles/cfd.x.dir/main/Parameters.F90.o
[ 25%] Building Fortran object CMakeFiles/cfd.x.dir/main/SimulationVars.F90.o
[ 37%] Building Fortran object CMakeFiles/cfd.x.dir/io/io.F90.o
[ 50%] Building Fortran object CMakeFiles/cfd.x.dir/main/SimulationSetup.F90.o
[ 62%] Building Fortran object CMakeFiles/cfd.x.dir/main/GridSetup.F90.o
[ 75%] Building Fortran object CMakeFiles/cfd.x.dir/main/GridTransformSetup.F90.o
[ 87%] Building Fortran object CMakeFiles/cfd.x.dir/main/GridTransform.F90.o
[100%] Building Fortran object CMakeFiles/cfd.x.dir/main/main.F90.o
Linking Fortran executable cfd.x
[100%] Built target cfd.x
$ ls
$ build  cfd.x  input.dat  setup.sh
```

If you run this, you will get executable named **cfd.x** and **input.dat** files. The input file is made by default. You can quickly change the required options.

1.3.3 Input file setup

The GridGen code allows user to set multiple options to generate grid by reading **input.dat** file at the beginning of the computation. Followings are default setup values you can find in the input file when you run **setup.sh** script:

```
# Input file for tecplot print
Flow in a channel
imax            41
jmax            2
kmax            19
# domain input (Corner points: x,y coordinates)
```

```
p1              -0.8    0.0     0.0
p2              1.8     0.0     0.0
p3              -0.8    0.0     1.0
p4              1.8     0.0     1.0
GeoStart        0.0     0.0     0.0
GeoEnd          1.0     0.0     0.0
FEsize          11
GeoSize         21
DCsize          11
width           0.1
# Grid clustering:
# cy1: stretched grid in z
# cy2: stretched Pi in z
# cy3: stretched Psi in x
# cy4: stretched grid along FE
# cy5: stretched grid along ED
# cy6: stretched grid along DC
cy1             2.0
cy2             -5.001
cy3             0.001
cy4             -1.2
cy5             1.0
cy6             0.001
# Iteration max: If nmax == 0, elliptic grid won't be calculated
nmax            500
# RMS Criterion
RMScrit         1.0E-6
# Calculate control terms: Pi, Psi
iControl        1
```

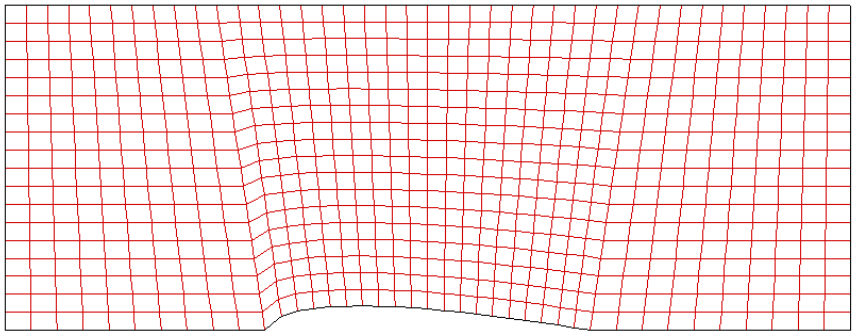
- **imax, jmax, kmax**: These three parameters set the size of grid points in x, y, and z direction, respectively.
- **p1, p2, p3, p4**: Define the corner points that form the front surface of the 3-dimensional computational domain.
- **GeoStart, GeoEnd**: Start and end points of airfoil geometry
- **FEsize, GeoSize, DCsize**: Number of grid points along FE, airfoil shape, and DC
- **width**: Depth of 3D computational domain in y-direction.
- **cy1 ~ cy6**: Stretching parameters used in the stretching formula, which is inherently defined for the grid point spacing in the z direction. In this code, this formula is applied to control terms and bottom edge spacing to define a new grid alignment for Grid #5.
- **nmax**: Maximum number of main loop. If the residual criterion is met before this maximum number is reached, the code will be terminated. If nmax is set to 0, the code will only run for the algebraic grid.
- **RMScrit**: Minimum RMS residual value to obtain the coverged Thomas method calculation.
- **iControl**: If it is 1, the code runs with pre-specified ϕ and ψ at the boundary points.

1.4 Results summary

The GridGen code builds 3-dimensional computational domain. Note that the 3-D domain is made with 2-dimensional front surface composed of x and z coordinates. In the given handout, the coordinate is inherently based on x and y coordinates. In this code, however, the vertical alignment is defined in z, then the 'width' of the 3-D domain is defined along the y direction.

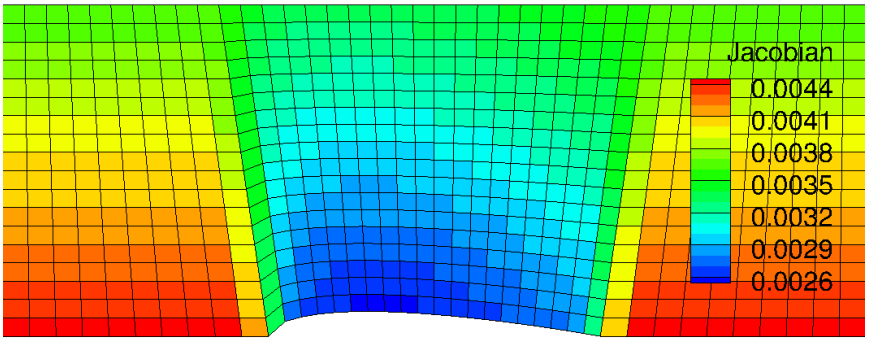
1.4.1 Grid #1: Algebraic grid with non-clustered points in z

The figure below shows the grid point alignments made by the GridGen code with algebraic grid and uniform grid spacing assumptions at every boundary edges. The interior points were generated by applying linear interpolation based two opposed pre-specified grid points. Thus the current grid has almost straight lines but with normally inclined angles, which makes a little skewed cells in the leading edge of the air foil. Also we can find a sudden change in cell volume across two grid lines anchored in leading and trailing edges of the airfoil.



<Figure: Grid points alignment of Grid #1>

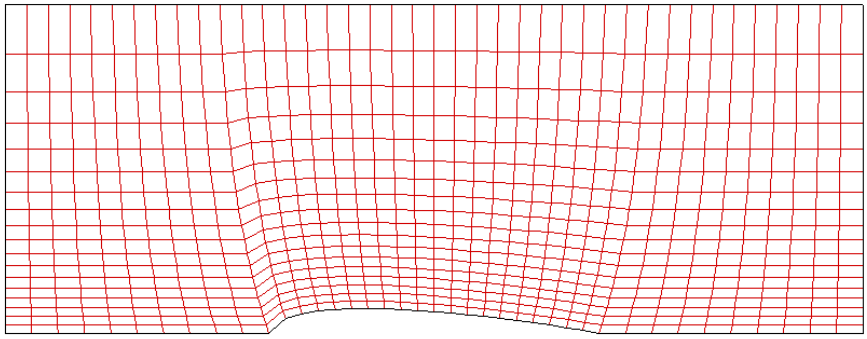
The more quantitative analysis is available with grid Jacobian contour on the current mesh. The 'Jacobian' here is inherently defined as determinant of inverse grid Jacobian matrix at every single grid point. Thus, it indicates a grid cell volume in 3D and cell area in 2D. Here, since the currently used Jacobian is defined at 3-dimensional coordinates, the grid shown below was made with a width of 0.1 m in y direction , however, it does not have grid resolution in this direction.



<Figure: Inverse Grid Jacobian distribution of Grid #1>

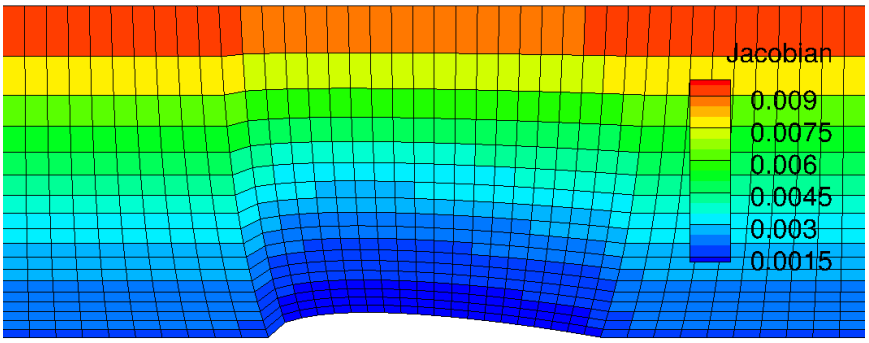
1.4.2 Grid #2: Algebraic grid with clustered points in z

The second trial was made on the point spacing stretching with algebraic grid alignment. This grid is based on the same approach for Grid #1. The only change in this grid was to apply gradually clustered grid points downward at left and right boundaries. Note that the linear interpolation of x-coordinates along the each vertical line is made only on the basis of j-index as formulated earlier. The effect of this is to make x coordinate shifting along the vertical line is identical for every point. Thus it leads to the somewhat much shifting for concentrated grid points in y-direction. Now we can observe non-linear grid lines in j-direction. This makes grid less skewer in the leading edge of the airfoil.



<Figure: Grid points alignment of Grid #2>

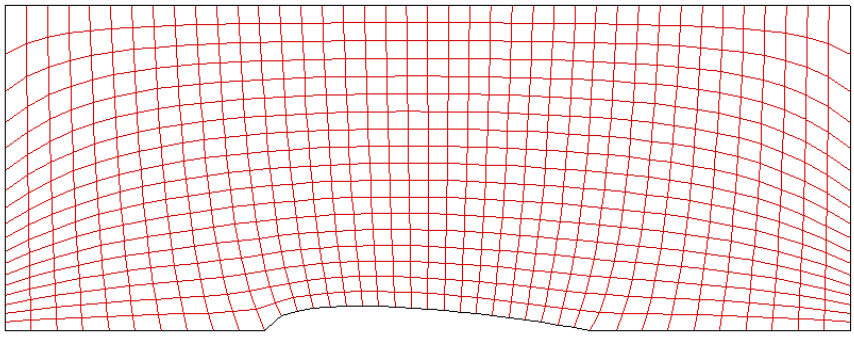
The grid Jacobian contour is shown below. Applying grid stretching along the y direction gives big cell volume distribution gradually upper. Change in volume along the bottom edge looks more less significant even in the leading edge. Since, however, the grid spacing is not changed in x direction from Grid #1 alignment, we could expect some error in flux throught the cell face at leading edge achored point. The same situation happens at the trailing point of the airfoil. In some point, this grid alignment is more reliable for this geometry because the significantly high gradient of flow velocity will only take place in the leading edge so that we need more dense grid points in this reagion.



<Figure: Inverse Grid Jacobian distribution of Grid #2>

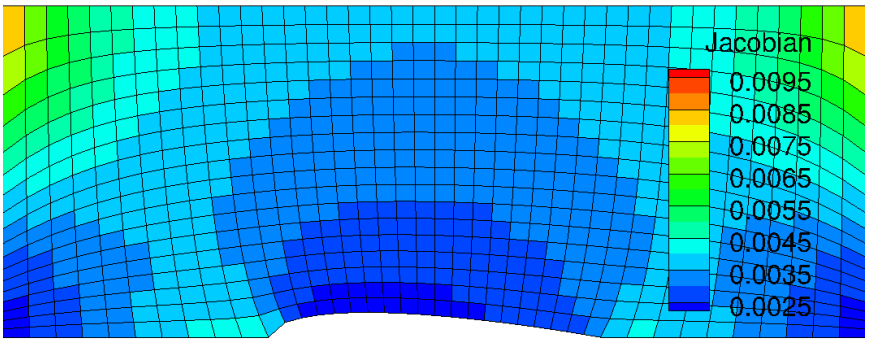
1.4.3 Grid #3: Elliptic grid with clustered points in z & no control terms

The grid shown below is made by the elliptic Poisson equations with clustered grid points in vertical direction. As expected, the Poisson equation with no control terms draws grid aliments resembled with iso-stream lines and iso-potention lines around the airfoil body. This is because the set of Posson equation is exactly same as a set of stream function and potention function when the control terms are ignored.



<Figure: Grid points alignment of Grid #3>

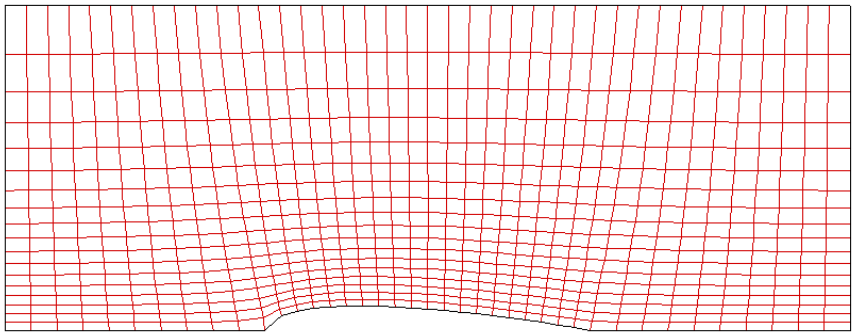
However, it is expected that curved lines right at the inlet edge and outlet edge are not aligned with the inlet flow. This misaligment could cause the flux of flow properties across the k-constant lines and thus it would make numerical errors. From the grid Jacobian contour result, sudden change in cell volume along the flow direction can be found. Maximum and minimum cell volume are found at left and right top edge and bottom edge, respectively.



<Figure: Inverse Grid Jacobian distribution of Grid #3>

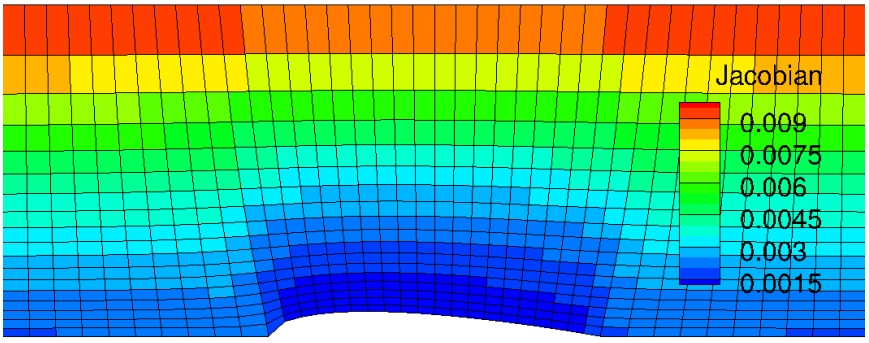
1.4.4 Grid #4: Elliptic grid with clustered points in z & control terms

The problem that arise in Grid #3 case was able to be resolved by adding control terms for Poisson equation. From the mesh shape of Grid #4 shown below, it can be found that adding control terms plays an important role in improving grid orthogonality. Thus now we have better grid aligment especially along the flow stream lines that can be expected intuitively. Even though there is a significant change in grid size along the vertical line, it may not act as a critical issue for numerical accuracy because the flux in vertical direction will be quite important.



<Figure: Grid points alignment of Grid #4>

In this grid, we can find a severely skewed cell in the leading edge of airfoil. This is more severe than Grid #3. Making orthogonality for the vertical lines cause more vertically stand i-constant lines, hence it leads to the sharp angle between airfoil arc and i-constant line anchored at the leading edge.



<Figure: Inverse Grid Jacobian distribution of Grid #4>

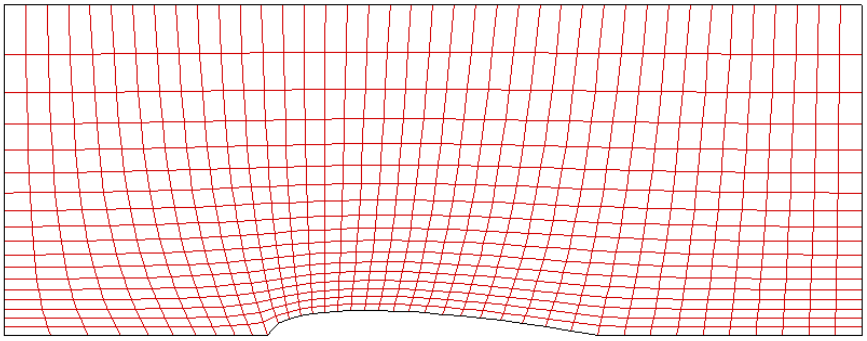
1.4.5 Grid #5: Improved grid quality

We observed several issues in grid quality stepping through the Grid #1 ~ #4. Since Grid #4 shows better quality than others, the new approach started with the method employed in Grid #4. The unresolved issues in Grid #4 can be summarized as followings:

- Sudden chanege in grid cell size at the leading edge point and trailing edge point.
- Skewness becomes more severe when applying control terms especially at leading edge point.

In this approach, an effort was made to resolve the above issues. First of all, to make the smooth change in grid cell size, stretching formula was employed along the FE, ED, and DC lines. As already mentioned earlier, this can be controlled by adding 'cy' values in 'input.dat' file. The following shows a part of 'input.dat' which is applied to Grid #5:

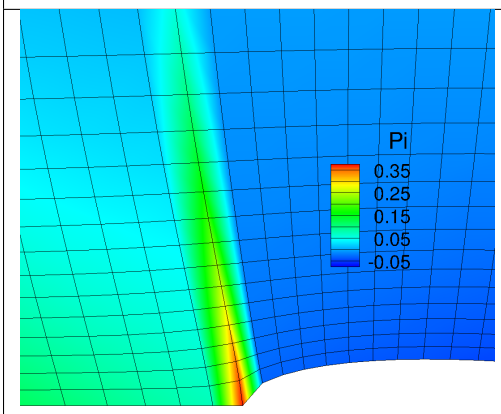
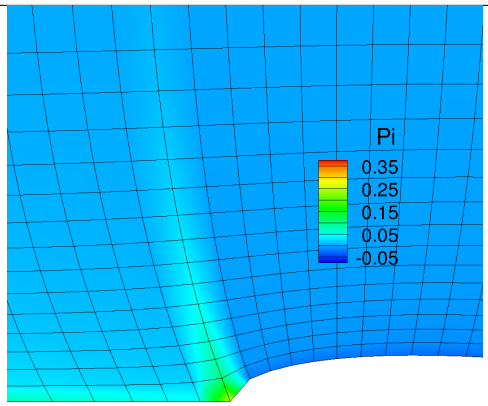
```
# Grid clustering:
# cy1: stretched grid in z
# cy2: stretched Pi in z
# cy3: stretched Psi in x
# cy4: stretched grid along FE
# cy5: stretched grid along ED
# cy6: stretched grid along DC
cy1             2.0
cy2             -10.0
cy3             0.001
cy4             -1.2
cy5             1.0
cy6             0.001
```



<Figure: Grid points alignment of Grid #4>

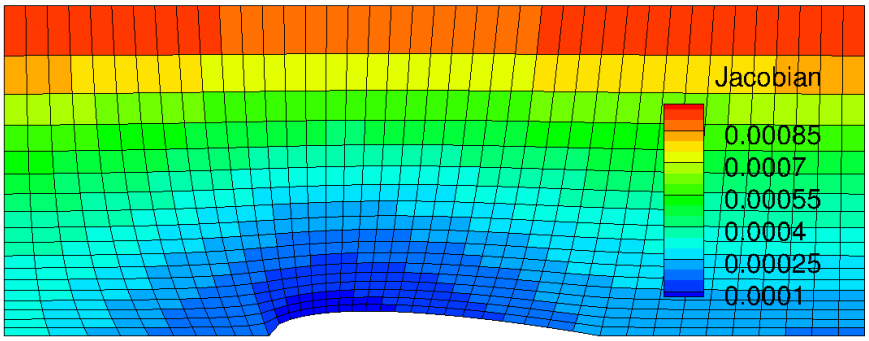
The 'cy1' remains unchanged but 'cy4', 'cy5', and 'cy6' are aditionally defined to change the grid spacing along the FE, airfoil arc, and DC, respectively. Here, negative value makes the grid points more concentrated towards the right corner. As a result, by adding proper values for these parameters, sudden change in grid size was avoided. Moreover, this results in more grid points near the leading edge. This is better grid alignment because we can intuitively expect that there is more significant change in flow properties when flow meet the leading edge.

In this approach, the grid spacing along the top edge (A-B) is left uniform because the flow properties will not experience significant change. Only significant change we care about will take place only in the leading edge.

• No stretched ‘cy = 0.001’	• Stretching factor ‘cy2 = -10’ applied
	

<Figure: Change in ϕ by stretching factor ‘cy3’>

As can be found above, control terms can be aditionally controlled by changing ‘cy2’ and ‘cy3’. The zoomed-in grid shown below confirmed an effect of changing ‘cy3’ value on the distribution of ϕ value. Less ϕ value helps the grid alignment resemble with the Grid #3, which shows the less skew cell in the leading edge.

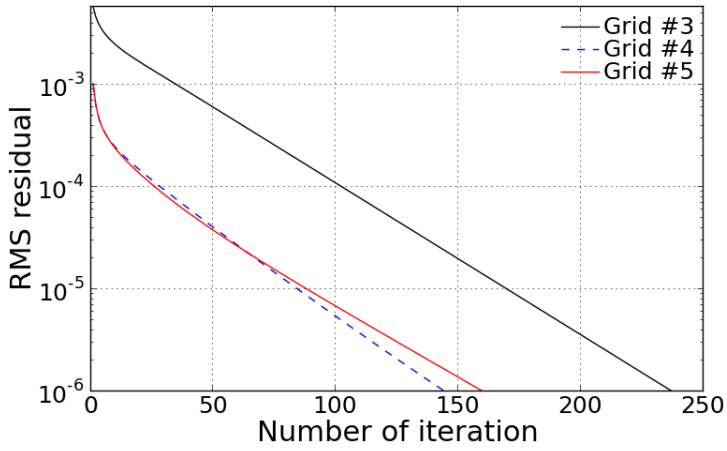


<Figure: Inverse Grid Jacobian distribution of Grid #4>

From the Jacobian contour, we can find that the smallest Jacobian value has been shifted towards the leading edge. This is because the Grid #5 has more grid points near this region. It is expected that the significant flow property change will be covered by blue and dark blue colored region in the above grid.

Convergence check: RMS residual log

A figure shown below illustrates the convergence history as a function of iteration number. This log is made only for the Elliptic grid solution because it is stored while Thomas method is being looped. Every cases meet the pre-specified RMS criterion. Here we can find that adding control terms helps fast convergence.


Grid #3
Grid #4
Grid #5
RMS residual
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
50
100
150
200
250
Number of iteration

CHAPTER 2

FORTRAN 90 Source code

2.1 CMakeList.txt

```
cmake_minimum_required(VERSION 2.6)

project(CFD)

enable_language(Fortran)

#
# add sub-directories defined for each certain purpose
#
add_subdirectory(main)
add_subdirectory(io)

#
# set executable file name
#
set(CFD_EXE_NAME cfd.x CACHE STRING "CFD executable name")

#
# set source files
#
set(CFD_SRC_FILES ${MAIN_SRC_FILES}
                  ${IO_SRC_FILES})

#
# define executable
#
add_executable(${CFD_EXE_NAME} ${CFD_SRC_FILES})
```

2.2 io directory

2.2.1 CMakeLists.txt

```
set(IO_SRC_FILES
    ${CMAKE_CURRENT_SOURCE_DIR}/io.F90 CACHE INTERNAL "" FORCE)
```

2.2.2 io.F90

```
!> \file: io.F90
!> \author: Sayop Kim
!> \brief: Provides routines to read input and write output

MODULE io_m
   USE Parameters_m, ONLY: wp
   USE SimulationVars_m, ONLY: nmax
   USE GridTransformSetup_m, ONLY: RMScrit
   IMPLICIT NONE

   PUBLIC :: filenameLength, Gpnts, FEsize, GeoSize, DCsize, &
             ReadGridInput, WriteTecPlot, WriteRMSlog, width, &
             iControl

   REAL(KIND=wp), DIMENSION(3,2) :: Gpnts    ! Geometry points(start,end)
   REAL(KIND=wp) :: width   ! width: domain width
   INTEGER :: FEsize, GeoSize, DCsize
   INTEGER :: iControl
   INTEGER, PARAMETER :: IOunit = 10, filenameLength = 64
   CHARACTER(LEN=50) :: prjTitle

CONTAINS

!------------------------------------------------------------------------------!
   SUBROUTINE ReadGridInput()
!------------------------------------------------------------------------------!
! Read input files for transformation 1:
!------------------------------------------------------------------------------!
   USE SimulationVars_m, ONLY: imax, jmax, kmax,&
                               xblkV, cy1, cy2, cy3, cy4, cy5, cy6
   IMPLICIT NONE
   INTEGER :: ios, i, j
   CHARACTER(LEN=8) :: inputVar

   OPEN(IOunit, FILE = 'input.dat', FORM = 'FORMATTED', ACTION = 'READ', &
         STATUS = 'OLD', IOSTAT = ios)
   IF(ios /= 0) THEN
      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "Fatal error: Could not open the input data file."
      RETURN
   ELSE
      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "Reading input file for transformation 1"
   ENDIF

   READ(IOunit,*)
   READ(IOunit,'(a)') prjTitle
   WRITE(*,'(4a)') 'Project Title:', '"',TRIM(prjTitle),'"'
   READ(IOunit,*) inputVar, imax
   WRITE(*,'(a,i6)') inputVar,imax
   READ(IOunit,*) inputVar, jmax
   WRITE(*,'(a,i6)') inputVar,jmax
   READ(IOunit,*) inputVar, kmax
   WRITE(*,'(a,i6)') inputVar,kmax
   READ(IOunit,*)
   READ(IOunit,*) inputVar, xblkV(1,1), xblkV(2,1), xblkV(3,1)
```

```
WRITE(*,'(a,3f6.3)') inputVar, xblkV(1,1), xblkV(2,1), xblkV(3,1)
READ(IOunit,*) inputVar, xblkV(1,2), xblkV(2,2), xblkV(3,2)
WRITE(*,'(a,3f6.3)') inputVar, xblkV(1,2), xblkV(2,2), xblkV(3,2)
READ(IOunit,*) inputVar, xblkV(1,3), xblkV(2,3), xblkV(3,3)
WRITE(*,'(a,3f6.3)') inputVar, xblkV(1,3), xblkV(2,3), xblkV(3,3)
READ(IOunit,*) inputVar, xblkV(1,4), xblkV(2,4), xblkV(3,4)
WRITE(*,'(a,3f6.3)') inputVar, xblkV(1,4), xblkV(2,4), xblkV(3,4)
! Set remaining corner points for 3D
DO i = 1, 4
   DO j = 1, 3, 2
      xblkV(j,i+4) = xblkV(j,i)
   ENDDO
   xblkV(2,i+4) = width
ENDDO
! Read Airfoil corner point data
READ(IOunit,*) inputVar, Gpnts(1,1), Gpnts(2,1), Gpnts(3,1)
WRITE(*,'(a,3f6.3)') inputVar, Gpnts(1,1), Gpnts(2,1), Gpnts(3,1)
READ(IOunit,*) inputVar, Gpnts(1,2), Gpnts(2,2), Gpnts(3,2)
WRITE(*,'(a,3f6.3)') inputVar, Gpnts(1,2), Gpnts(2,2), Gpnts(3,2)
! Read Airfoil grid resolution data (at bottom edge)
READ(IOunit,*) inputVar, FEsize
WRITE(*,'(a,i6)') inputVar, FEsize
READ(IOunit,*) inputVar, GeoSize
WRITE(*,'(a,i6)') inputVar, GeoSize
READ(IOunit,*) inputVar, DCsize
WRITE(*,'(a,i6)') inputVar, DCsize
READ(IOunit,*) inputVar, width
WRITE(*,'(a,f6.3)') inputVar,width
READ(IOunit,*)
READ(IOunit,*)
READ(IOunit,*)
READ(IOunit,*)
READ(IOunit,*)
READ(IOunit,*)
READ(IOunit,*)
READ(IOunit,*) inputVar, cy1
WRITE(*,'(a,f6.3)') inputVar, cy1
READ(IOunit,*) inputVar, cy2
WRITE(*,'(a,f6.3)') inputVar, cy2
READ(IOunit,*) inputVar, cy3
WRITE(*,'(a,f6.3)') inputVar, cy3
READ(IOunit,*) inputVar, cy4
WRITE(*,'(a,f6.3)') inputVar, cy4
READ(IOunit,*) inputVar, cy5
WRITE(*,'(a,f6.3)') inputVar, cy5
READ(IOunit,*) inputVar, cy6
WRITE(*,'(a,f6.3)') inputVar, cy6
READ(IOunit,*)
READ(IOunit,*) inputVar, nmax
WRITE(*,'(a,i6)') inputVar, nmax
READ(IOunit,*)
READ(IOunit,*) inputVar, RMScrit
WRITE(*,'(a,f6.3)') inputVar, RMScrit
READ(IOunit,*)
READ(IOunit,*) inputVar, iControl
WRITE(*,'(a,i6)') inputVar, iControl
! Set remaining corner points for 3D
DO i = 1, 4
```

```
      DO j = 1, 3, 2
         xblkV(j,i+4) = xblkV(j,i)
      ENDDO
      xblkV(2,i+4) = width
   ENDDO

   CLOSE(IOunit)
   END SUBROUTINE ReadGridInput


!-------------------------------------------------------------------------------!
   SUBROUTINE WriteTecPlot(fileName,varList)
!-------------------------------------------------------------------------------!
! Write Tecplot file
!-------------------------------------------------------------------------------!
   USE SimulationVars_m, ONLY: imax, jmax, kmax,&
                               xp, inverseJacobian
   USE GridTransformSetup_m, ONLY: Pi, Psi
   IMPLICIT NONE
   CHARACTER(LEN=filenameLength), INTENT(IN) :: fileName
   CHARACTER(LEN=*), INTENT(IN) :: varList
   INTEGER :: i, j, k

   OPEN(IOunit, File = fileName, FORM = 'FORMATTED', ACTION = 'WRITE')
   ! writes the two line TECPLOT header
   WRITE(IOunit,'(a)') 'Title="' // TRIM(prjTitle) // '"'
   WRITE(IOunit,'(a)') 'Variables=' // TRIM(varList)

   WRITE(IOunit,'(a)') ""
   WRITE(IOunit,'(a,i6,a,i6,a,i6,a)') 'Zone I=', imax, ', J=', jmax, ', K=', kmax, ', F=POINT'

   DO k = 1, kmax
      DO j = 1, jmax
         DO i = 1, imax
            WRITE(IOunit,'(6g15.6)') xp(1,i,j,k), xp(2,i,j,k), xp(3,i,j,k), &
                                     inverseJacobian(i,j,k), Pi(i,j,k), Psi(i,j,k)
         ENDDO
      ENDDO
   ENDDO
   CLOSE(IOunit)

   END SUBROUTINE WriteTecPlot

!-------------------------------------------------------------------------------!
   SUBROUTINE WriteRMSlog(nIter,fileName)
!-------------------------------------------------------------------------------!
! Write Tecplot file
!-------------------------------------------------------------------------------!
   USE GridTransformSetup_m, ONLY: RMSres
   IMPLICIT NONE
   CHARACTER(LEN=filenameLength), INTENT(IN) :: fileName
   INTEGER :: nIter

   IF ( nIter == 1 ) THEN
      OPEN(IOunit, File = fileName, FORM = 'FORMATTED', ACTION = 'WRITE')
   ELSE
      OPEN(IOunit, File = fileName, FORM = 'FORMATTED', ACTION = 'WRITE', &
          POSITION = 'APPEND')
```

```
   ENDIF
   write(IOunit,'(i6,g15.6)') nIter, RMSres
   CLOSE(IOunit)
   END SUBROUTINE WriteRMSlog
END MODULE io_m
```

2.3 main directory

2.3.1 CMakeLists.txt

```
set(MAIN_SRC_FILES
   ${CMAKE_CURRENT_SOURCE_DIR}/main.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/SimulationSetup.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/SimulationVars.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/GridSetup.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/GridTransform.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/GridTransformSetup.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/Parameters.F90 CACHE INTERNAL "" FORCE)
```

2.3.2 main.F90

```
!> \file: main.F90
!> \author: Sayop Kim

PROGRAM main
   USE SimulationSetup_m, ONLY: InitializeCommunication
   USE GridSetup_m, ONLY: InitializeGrid
   USE GridTransform_m, ONLY: GridTransform
   USE io_m, ONLY: WriteTecPlot, filenameLength
   USE Parameters_m, ONLY: wp

   IMPLICIT NONE

   CHARACTER(LEN=filenameLength) :: outputfile = 'output.tec'

   CALL InitializeCommunication
   ! Make initial condition for grid point alignment
   ! Using Algebraic method
   CALL InitializeGrid
   ! Use Elliptic grid points
   CALL GridTransform
   CALL WriteTecPlot(outputfile,'"I","J","K","Jacobian","Pi","Psi"')
END PROGRAM main
```

2.3.3 SimulationVars.F90

```
!> \file: SimulationVars.F90
!> \author: Sayop Kim

MODULE SimulationVars_m
   USE parameters_m, ONLY : wp
   IMPLICIT NONE
```

```
   INTEGER :: imax, jmax, kmax, nmax
   REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:,:,:) :: xp
   REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:) :: BOTedge
   REAL(KIND=wp) :: cy1, cy2, cy3, cy4, cy5, cy6
   REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:,:) :: inverseJacobian
   REAL(KIND=wp), DIMENSION(3,8) :: xblkV    ! x,y,z points at 8 vertices of block
END MODULE SimulationVars_m
```

2.3.4 parameters.F90

```
!> \file parameters.F90
!> \author Sayop Kim
!> \brief Provides parameters and physical constants for use throughout the
!! code.
MODULE Parameters_m
   INTEGER, PARAMETER :: wp = SELECTED_REAL_KIND(8)

   CHARACTER(LEN=10), PARAMETER :: CODE_VER_STRING = "V.001.001"
   REAL(KIND=wp), PARAMETER :: PI = 3.14159265358979323846264338_wp

END MODULE Parameters_m
```

2.3.5 GridSetup.F90

```
!> \file: GridSetup.F90
!> \author: Sayop Kim

MODULE GridSetup_m
   USE Parameters_m, ONLY: wp
   USE SimulationSetup_m, ONLY: GridStretching
   IMPLICIT NONE

   PUBLIC :: InitializeGrid, GenerateInteriorPoints

CONTAINS
!------------------------------------------------------------------------------!
   SUBROUTINE InitializeGrid()
!------------------------------------------------------------------------------!
   USE io_m, ONLY: ReadGridInput
   IMPLICIT NONE

   ! Create Bottom Edge coordinate values
   CALL ReadGridInput
   CALL InitializeGridArrays
   CALL CreateBottomEdge
   CALL SetEdgePnts
   CALL GridPntsAlgbra
   CALL GenerateInteriorPoints

   END SUBROUTINE

!------------------------------------------------------------------------------!
   SUBROUTINE InitializeGridArrays()
!------------------------------------------------------------------------------!
```

```
      ! imax: number of grid points in i-drection
      ! jmax: number of grid points in j-direction
      ! kmax: number of grid points in k-direction
      ! xp(3,imax,jmax,kmax): curvilinear coordinates in physical space
      USE SimulationVars_m, ONLY: imax, jmax, kmax, &
                                  xp, inverseJacobian
      IMPLICIT NONE

      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "Initializing data arrays..."
      ALLOCATE(xp(3,imax,jmax,kmax))
      ALLOCATE(inverseJacobian(imax,jmax,kmax))
      xp = 0.0_wp
      inverseJacobian = 0.0_wp
   END SUBROUTINE


!---------------------------------------------------------------------------!
   SUBROUTINE CreateBottomEdge()
!---------------------------------------------------------------------------!
   USE io_m, ONLY: width, FEsize, GeoSize, DCsize, &
                   Gpnts
   USE SimulationVars_m, ONLY: imax, jmax, kmax,&
                               xblkV, cy4, cy5, cy6
   USE SimulationVars_m, ONLY: BOTedge
   USE SimulationSetup_m, ONLY: UniformSpacing
   IMPLICIT NONE
   INTEGER :: i

   ALLOCATE(BOTedge(3,imax))
   WRITE(*,*) ""
   WRITE(*,*) "Creating Bottome edge point values with Airfoil geometry"
   DO i = 2, FEsize
      BOTedge(1,i) = GridStretching(xblkV(1,1), Gpnts(1,1), i, FEsize, cy4)
      !BOTedge(2,i) = UniformSpacing(xblkV(2,1), Gpnts(2,1), i, FEsize)
      BOTedge(3,i) = GridStretching(xblkV(3,1), Gpnts(3,1), i, FEsize, cy4)
   ENDDO
   DO i = FEsize + 1, FEsize + GeoSize - 1
      BOTedge(1,i) = GridStretching(Gpnts(1,1), Gpnts(1,2), i-FEsize+1, GeoSize, cy5)
      !BOTedge(2,i) = UniformSpacing(Gpnts(2,1), Gpnts(2,2), i-FEsize+1, GeoSize)
      !BOTedge(3,i) = UniformSpacing(Gpnts(3,1), Gpnts(3,2), i-FEsize+1, GeoSize)
      BOTedge(3,i) = Airfoil(BOTedge(1,i))
   ENDDO
   DO i = FEsize + GeoSize, imax - 1
      BOTedge(1,i) = GridStretching(Gpnts(1,2), xblkV(1,2), i-FEsize-GeoSize+2, &
                                    DCsize, cy6)
      !BOTedge(2,i) = UniformSpacing(Gpnts(2,2), xblkV(2,2), i-FEsize-GeoSize+2, DCsize)
      BOTedge(3,i) = GridStretching(Gpnts(3,2), xblkV(3,2), i-FEsize-GeoSize+2, &
                                    DCsize, cy6)
   ENDDO

   END SUBROUTINE


!---------------------------------------------------------------------------!
   FUNCTION Airfoil(xx) RESULT(yx)
!---------------------------------------------------------------------------!
   IMPLICIT NONE
```

```
   REAL(KIND=wp) xint, thick, xx, yx
   xint = 1.008930411365_wp
   thick = 0.15_wp
   yx = 0.2969_wp * sqrt(xint * xx) - 0.126_wp * xint * xx - 0.3516_wp * &
        (xint * xx)**2 + 0.2843_wp * (xint * xx)**3 - 0.1015_wp * (xint * xx)**4
   yx = 5.0_wp * thick * yx

   END FUNCTION Airfoil


!---------------------------------------------------------------------------!
   SUBROUTINE SetEdgePnts()
!---------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: imax, jmax, kmax, &
                                  xp, xblkV, BOTedge, cy1
      USE SimulationSetup_m, ONLY: UniformSpacing
      IMPLICIT NONE
      INTEGER :: i

      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "Setting Boundary Conditions..."
      !+++++++++++++++++++++++++++++++++++++++++++++++++++!
      ! Assign coordinates value in xblkV(8,3)           !
      ! Below shows 8 vertices defined in one single block!
      !                                                   !
      !         7--------------8                          !
      !        /|             /|                          !
      !       / |            / |                          !
      !      3--------------4  |    z  y                  !
      !      |  |           |  |    | /                   !
      !      |  5-----------|--6    |/                    !
      !      | /            | /     --- x                 !
      !      |/             |/                            !
      !      1--------------2                             !
      !                                                   !
      !+++++++++++++++++++++++++++++++++++++++++++++++++++!
      ! Vertex (1)
      !xblkV(1,1) = 0.0
      !xblkV(2,1) = 0.0
      !xblkV(3,1) = 0.0
      DO i = 1, 3
         xp(i,1,1,1) = xblkV(i,1)
      ENDDO
      ! Vertex (2)
      !xblkV(1,2) = 0.0
      !xblkV(2,2) = 0.0
      !xblkV(3,2) = 0.0
      DO i = 1, 3
         xp(i,imax,1,1) = xblkV(i,2)
      ENDDO
      ! Vertex (3)
      !xblkV(1,3) = 0.0
      !xblkV(2,3) = 0.0
      !xblkV(3,3) = 0.0
      DO i = 1, 3
         xp(i,1,1,kmax) = xblkV(i,3)
      ENDDO
```

```
      ! Vertex (4)
      !xblkV(1,4) = 0.0
      !xblkV(2,4) = 0.0
      !xblkV(3,4) = 0.0
      DO i = 1, 3
         xp(i,imax,1,kmax) = xblkV(i,4)
      ENDDO
      ! Vertex (5)
      !xblkV(1,5) = 0.0
      !xblkV(2,5) = 0.0
      !xblkV(3,5) = 0.0
      DO i = 1, 3
         xp(i,1,jmax,1) = xblkV(i,5)
      ENDDO
      ! Vertex (6)
      !xblkV(1,6) = 0.0
      !xblkV(2,6) = 0.0
      !xblkV(3,6) = 0.0
      DO i = 1, 3
         xp(i,imax,jmax,1) = xblkV(i,6)
      ENDDO
      ! Vertex (7)
      !xblkV(1,7) = 0.0
      !xblkV(2,7) = 0.0
      !xblkV(3,7) = 0.0
      DO i = 1, 3
         xp(i,1,jmax,kmax) = xblkV(i,7)
      ENDDO
      ! Vertex (8)
      !xblkV(1,8) = 0.0
      !xblkV(2,8) = 0.0
      !xblkV(3,8) = 0.0
      DO i = 1, 3
         xp(i,imax,jmax,kmax) = xblkV(i,8)
      ENDDO
      !+++++++++++++++++++++++++++++++++++++++++++++++++++
      ! Set up boundary point coordinates at every edge
      !
      !
      !          +--------(8)---------+
      !         /|                   /|
      !      (11)|                (12)|
      !       /  |                 /  |
      !      +---------(4)--------+  (6)
      !      |  (5)               |   |
      !      |   |                |   |      z  y
      !      |   |                |   |      | /
      !     (1)  +-------(7)------|---+      |/
      !      |  /                (2) /       ---x
      !      |(9)                 |(10)
      !      |/                   |/
      !      +--------(3)---------+
      !
      !+++++++++++++++++++++++++++++++++++++++++++++++++++
      ! edge (1)
      DO i = 2, kmax - 1
         xp(1,1,1,i) = UniformSpacing(xblkV(1,1), xblkV(1,3), i, kmax)
         xp(2,1,1,i) = UniformSpacing(xblkV(2,1), xblkV(2,3), i, kmax)
```

```
      xp(3,1,1,i) = GridStretching(xblkV(3,1), xblkV(3,3), i, kmax, cy1)
   ENDDO
   ! edge (2)
   DO i = 2, kmax - 1
      xp(1,imax,1,i) = UniformSpacing(xblkV(1,2), xblkV(1,4), i, kmax)
      xp(2,imax,1,i) = UniformSpacing(xblkV(2,2), xblkV(2,4), i, kmax)
      xp(3,imax,1,i) = GridStretching(xblkV(3,2), xblkV(3,4), i, kmax, cy1)
   ENDDO
   ! edige (3)
   DO i = 2, imax - 1
      !xp(1,i,1,1) = UniformSpacing(xblkV(1,1), xblkV(1,2), i, imax)
      xp(2,i,1,1) = UniformSpacing(xblkV(2,1), xblkV(2,2), i, imax)
      !xp(3,i,1,1) = UniformSpacing(xblkV(3,1), xblkV(3,2), i, imax)
      xp(1,i,1,1) = BOTedge(1,i)
      xp(3,i,1,1) = BOTedge(3,i)
   ENDDO
   ! edge (4)
   DO i = 2, imax - 1
      xp(1,i,1,kmax) = UniformSpacing(xblkV(1,3), xblkV(1,4), i, imax)
      xp(2,i,1,kmax) = UniformSpacing(xblkV(2,3), xblkV(2,4), i, imax)
      xp(3,i,1,kmax) = UniformSpacing(xblkV(3,3), xblkV(3,4), i, imax)
   ENDDO
   ! edge (5)
   DO i = 2, kmax - 1
      xp(1,1,jmax,i) = xp(1,1,1,i)
      xp(2,1,jmax,i) = UniformSpacing(xblkV(2,5), xblkV(2,7), i, kmax)
      xp(3,1,jmax,i) = xp(3,1,1,i)
   ENDDO
   ! edge (6)
   DO i = 2, kmax - 1
      xp(1,imax,jmax,i) = xp(1,imax,1,i)
      xp(2,imax,jmax,i) = UniformSpacing(xblkV(2,6), xblkV(2,8), i, kmax)
      xp(3,imax,jmax,i) = xp(3,imax,1,i)
   ENDDO
   ! edge (7)
   DO i = 2, imax - 1
      !xp(1,i,jmax,1) = UniformSpacing(xblkV(1,5), xblkV(1,6), i, imax)
      xp(2,i,jmax,1) = UniformSpacing(xblkV(2,5), xblkV(2,6), i, imax)
      !xp(3,i,jmax,1) = UniformSpacing(xblkV(3,5), xblkV(3,6), i, imax)
      xp(1,i,jmax,1) = BOTedge(1,i)
      xp(3,i,jmax,1) = BOTedge(3,i)
   ENDDO
   ! edge (8)
   DO i = 2, imax - 1
      xp(1,i,jmax,kmax) = xp(1,i,1,kmax)
      xp(2,i,jmax,kmax) = UniformSpacing(xblkV(2,7), xblkV(2,8), i, imax)
      xp(3,i,jmax,kmax) = xp(3,i,1,kmax)
   ENDDO
   ! edge (9)
   DO i = 2, jmax - 1
      xp(1,1,i,1) = UniformSpacing(xblkV(1,1), xblkV(1,5), i, jmax)
      xp(2,1,i,1) = UniformSpacing(xblkV(2,1), xblkV(2,5), i, jmax)
      xp(3,1,i,1) = UniformSpacing(xblkV(3,1), xblkV(3,5), i, jmax)
   ENDDO
   ! edge (10)
   DO i = 2, jmax - 1
      xp(1,imax,i,1) = UniformSpacing(xblkV(1,2), xblkV(1,6), i, jmax)
      xp(2,imax,i,1) = UniformSpacing(xblkV(2,2), xblkV(2,6), i, jmax)
```

```
         xp(3,imax,i,1) = UniformSpacing(xblkV(3,2), xblkV(3,6), i, jmax)
      ENDDO
      ! edge (11)
      DO i = 2, jmax - 1
         xp(1,1,i,kmax) = UniformSpacing(xblkV(1,3), xblkV(1,7), i, jmax)
         xp(2,1,i,kmax) = UniformSpacing(xblkV(2,3), xblkV(2,7), i, jmax)
         xp(3,1,i,kmax) = UniformSpacing(xblkV(3,3), xblkV(3,7), i, jmax)
      ENDDO
      ! edge (12)
      DO i = 2, jmax - 1
         xp(1,imax,i,kmax) = UniformSpacing(xblkV(1,4), xblkV(1,8), i, jmax)
         xp(2,imax,i,kmax) = UniformSpacing(xblkV(2,4), xblkV(2,8), i, jmax)
         xp(3,imax,i,kmax) = UniformSpacing(xblkV(3,4), xblkV(3,8), i, jmax)
      ENDDO
   END SUBROUTINE


!-------------------------------------------------------------------------------!
   SUBROUTINE GridPntsAlgbra()
!-------------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: imax, jmax, kmax, &
                                  xp, xblkV, cy1
      USE SimulationSetup_m, ONLY: UniformSpacing
      IMPLICIT NONE
      INTEGER :: i, j, k

      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "Writing grid points on block surface..."

      !++++++++++++++++++++++++++++++++++++++++
      ! "front plane"
      !     +------------+
      !     |            |   z(k)
      !     | i-k plane  |    |
      !     |  (j = 1)   |    |
      !     1------------+    ---- x(i)
      !
      !k=kmax @---@---@---@---@
      !       |   |   |   |   |  @: edge points (known)
      !       @---o---o---o---@  o: interior points (unknown)
      !       |   |   |   |   |
      !       @---o---o---o---@
      !       |   |   |   |   |
      !   k=1 @---@---@---@---@
      !      i=1             i=imax
      ! x-coordinate is determined along the i=const lines
      ! y-coordinate is same as y of corner (1)
      ! z-coordinate is determined along the k=const lines
      !++++++++++++++++++++++++++++++++++++++++
      DO i = 2, imax - 1
         DO k = 2, kmax - 1
            xp(1,i,1,k) = UniformSpacing(xp(1,i,1,1), xp(1,i,1,kmax), k, kmax)
            xp(2,i,1,k) = UniformSpacing(xp(2,i,1,1), xp(2,i,1,kmax), k, kmax)
            xp(3,i,1,k) = GridStretching(xp(3,i,1,1), xp(3,i,1,kmax), k, kmax, cy1)
         ENDDO
      ENDDO
      !++++++++++++++++++++++++++++++++++++++++
      ! "back plane"
```

```
!     +------------+
!     |            |   z(k)
!     | i-k plane  |    |
!     | (j = jmax) |    |
!     5------------+    ---- x(i)
! x-coordinate is determined along the i=const lines
! y-coordinate is same as y of corner (5)
! z-coordinate is determined along the k=const lines
!+++++++++++++++++++++++++++++++++++++++
DO i = 2, imax - 1
   DO k = 2, kmax - 1
      xp(1,i,jmax,k) = xp(1,i,1,k)
      xp(2,i,jmax,k) = UniformSpacing(xp(2,i,jmax,1), xp(2,i,jmax,kmax), k, kmax)
      xp(3,i,jmax,k) = xp(3,i,1,k)
   ENDDO
ENDDO
!+++++++++++++++++++++++++++++++++++++++
! "left plane"
!                 +
!                /|
!               / |   j-k plane (i = 1)
!              /  |
!             +   +
!             |  /  z(k) y(j)
!             | /     |  /
!             |/      | /
!             1       |/
! x-coordinate is same as x of corner (1)
! y-coordinate is determined along the j=const lines
! z-coordinate is determined along the k=const lines
!+++++++++++++++++++++++++++++++++++++++
DO j = 2, jmax - 1
   DO k = 2, kmax - 1
      xp(1,1,j,k) = UniformSpacing(xp(1,1,j,1), xp(1,1,j,kmax), k, kmax)
      xp(2,1,j,k) = UniformSpacing(xp(2,1,j,1), xp(2,1,j,kmax), k, kmax)
      xp(3,1,j,k) = GridStretching(xp(3,1,j,1), xp(3,1,j,kmax), k, kmax, cy1)
   ENDDO
ENDDO
!+++++++++++++++++++++++++++++++++++++++
! "right plane"
!                 +
!                /|
!               / |   j-k plane (i = imax)
!              /  |
!             +   +
!             |  /  z(k) y(j)
!             | /     |  /
!             |/      | /
!             2       |/
! x-coordinate is same as x of corner (2)
! y-coordinate is determined along the j=const lines
! z-coordinate is determined along the k=const lines
!+++++++++++++++++++++++++++++++++++++++
DO j = 2, jmax - 1
   DO k = 2, kmax - 1
      xp(1,imax,j,k) = UniformSpacing(xp(1,imax,j,1), xp(1,imax,j,kmax), k, kmax)
      xp(2,imax,j,k) = xp(2,1,j,k)
      xp(3,imax,j,k) = xp(3,1,j,k)
```

```
         ENDDO
      ENDDO

      !++++++++++++++++++++++++++++++++++++++++
      ! "bottom plane"
      !           +-------------+
      !          /             /   y(j)
      !         /  i-j plane  /   /
      !        /  (k = 1)    /   /
      !       1-------------+    ---->x(i)
      ! x-coordinate is determined along the i=const lines
      ! y-coordinate is determined along the j=const lines
      ! z-coordinate is same as z of corner (1)
      !++++++++++++++++++++++++++++++++++++++++
      DO i = 2, imax - 1
         DO j = 2, jmax - 1
            xp(1,i,j,1) = UniformSpacing(xp(1,i,1,1), xp(1,i,jmax,1), j, jmax)
            xp(2,i,j,1) = UniformSpacing(xp(2,i,1,1), xp(2,i,jmax,1), j, jmax)
            xp(3,i,j,1) = xp(3,i,1,1)
         ENDDO
      ENDDO
      !++++++++++++++++++++++++++++++++++++++++
      ! "top plane"
      !           +-------------+
      !          /             /   y(j)
      !         /  i-j plane  /   /
      !        /  (k = kmax) /   /
      !       3-------------+    ---->x(i)
      ! x-coordinate is determined along the i=const lines
      ! y-coordinate is determined along the j=const lines
      ! z-coordinate is same as z of corner (3)
      !++++++++++++++++++++++++++++++++++++++++
      DO i = 2, imax - 1
         DO j = 2, jmax - 1
            xp(1,i,j,kmax) = xp(1,i,1,kmax)
            xp(2,i,j,kmax) = xp(2,i,j,1)
            xp(3,i,j,kmax) = UniformSpacing(xp(3,i,1,kmax), xp(3,i,jmax,kmax), j, jmax)
         ENDDO
      ENDDO
   END SUBROUTINE


!-------------------------------------------------------------------------------!
   SUBROUTINE GenerateInteriorPoints()
!-------------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: imax, jmax, kmax, &
                                  xp, xblkV, cy1
      USE SimulationSetup_m, ONLY: UniformSpacing
      IMPLICIT NONE
      INTEGER :: i, j, k

      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "Writing interior grid points..."
      DO i = 2, imax -1
         DO k = 2, kmax - 1
            DO j = 2, jmax - 1
               xp(1,i,j,k) = UniformSpacing(xp(1,i,1,k), xp(1,i,jmax,k), j, jmax)
               xp(2,i,j,k) = UniformSpacing(xp(2,i,1,k), xp(2,i,jmax,k), j, jmax)
```

```
               xp(3,i,j,k) = GridStretching(xp(3,i,1,k), xp(3,i,jmax,k), j, jmax, cy1)
            ENDDO
         ENDDO
      ENDDO

   END SUBROUTINE

END MODULE GridSetup_m
```

2.3.6 GridTransform.F90

```
!> \file GridTransform.F90
!> \author Sayop Kim

MODULE GridTransform_m
   USE Parameters_m, ONLY: wp
   USE io_m, ONLY: iControl, WriteRMSlog, filenameLength
   USE SimulationVars_m, ONLY: nmax
   USE GridTransformSetup_m, ONLY: InitializeArrays, CalculateA123, &
                                   CalculatePiPsi, ThomasLoop, &
                                   CopyFrontTOBack, CalculateGridJacobian, &
                                   RMSres, RMScrit
   USE GridSetup_m, ONLY: GenerateInteriorPoints
   IMPLICIT NONE
   CHARACTER(LEN=filenameLength) :: RMSlogfile = 'RMSlog.dat'
CONTAINS

!------------------------------------------------------------------------------!
   SUBROUTINE GridTransform()
!------------------------------------------------------------------------------!
   IMPLICIT NONE
   INTEGER :: n

   CALL InitializeArrays
   IF ( iControl == 1) CALL CalculatePiPsi
   DO n = 1, nmax
      CALL CalculateA123
      CALL ThomasLoop
      CALL WriteRMSlog(n,RMSlogfile)
      IF (RMSres <= RMScrit) EXIT
   ENDDO
   CALL CopyFrontTOBack
   CALL GenerateInteriorPoints
   CALL CalculateGridJacobian
   END SUBROUTINE GridTransform

END MODULE
```

2.3.7 GridTransformSetup.F90

```
!> \file GridTransformSetup.F90
!> \author Sayop Kim

MODULE GridTransformSetup_m
   USE Parameters_m, ONLY: wp
```

```
   USE SimulationVars_m, ONLY: imax, jmax, kmax, &
                               xp, cy2, cy3
   IMPLICIT NONE

   PUBLIC CalculateA123, CalculatePiPsi, ThomasLoop, &
          RMScrit, RMSres

   REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:,:,:,:) :: InverseGridMetrics
   REAL(KIND=wp), ALLOCATABLE, DIMENSION(:,:,:) :: A1, A2, A3, Pi, Psi
   REAL(KIND=wp) :: RMScrit, RMSres
CONTAINS

!------------------------------------------------------------------------------!
   SUBROUTINE InitializeArrays()
!------------------------------------------------------------------------------!
   IMPLICIT NONE

   ALLOCATE(A1(imax,1,kmax))
   ALLOCATE(A2(imax,1,kmax))
   ALLOCATE(A3(imax,1,kmax))
   A1 = 0.0_wp
   A2 = 0.0_wp
   A3 = 0.0_wp

   ALLOCATE(Pi(imax,1,kmax))
   ALLOCATE(Psi(imax,1,kmax))
   Pi = 0.0_wp
   Psi = 0.0_wp

   END SUBROUTINE InitializeArrays


!------------------------------------------------------------------------------!
   SUBROUTINE CalculateA123()
!------------------------------------------------------------------------------!
! Evaluate A1, A2, A3 coefficients before looping Thomas method
! A1 = (x_k)^2 + (z_k)^2
! A2 = (x_i)*(x_k) + (z_i)*(z_k)
! A3 = (x_i)^2 + (z_i)^2

   IMPLICIT NONE
   INTEGER :: i, j, k
   ! _i: derivative w.r.t ksi
   ! _k: derivative w.r.t zeta
   REAL(KIND=wp) :: x_i, z_i, x_k, z_k

   ! Evaluate only on j=1 surface (2D i-k front plane)
   j = 1

   !WRITE(*,'(a)') ""
   !WRITE(*,'(a)') "Calculating A1, A2, A3 coefficients for the governing equation..."
   DO i = 2, imax - 1
      DO k = 2, kmax - 1
         x_i = 0.5_wp * (xp(1,i+1,j,k) - xp(1,i-1,j,k))
         z_i = 0.5_wp * (xp(3,i+1,j,k) - xp(3,i-1,j,k))
         x_k = 0.5_wp * (xp(1,i,j,k+1) - xp(1,i,j,k-1))
         z_k = 0.5_wp * (xp(3,i,j,k+1) - xp(3,i,j,k-1))
         A1(i,j,k) = x_k**2 + z_k**2
```

```
         A2(i,j,k) = x_i*x_k + z_i*z_k
         A3(i,j,k) = x_i**2 + z_i**2
      ENDDO
   ENDDO

   END SUBROUTINE CalculateA123

!---------------------------------------------------------------------------!
   SUBROUTINE CalculatePiPsi()
!---------------------------------------------------------------------------!
! Initialize Pi and Psy value before moving into pseudo time loop.
   USE SimulationSetup_m, ONLY: UniformSpacing, GridStretching

   IMPLICIT NONE
   INTEGER :: i, j, k
   ! _i: derivative w.r.t ksi
   ! _k: derivative w.r.t zeta
   REAL(KIND=wp) :: x_i, z_i, x_ii, z_ii, &
                    x_k, z_k, x_kk, z_kk

   ! Evaluate only on j=1 surface (2D i-k front plane)
   j = 1

   WRITE(*,'(a)') ""
   WRITE(*,'(a)') "Calculating Pi and Psi variables for controling elliptic grid..."
   ! Evaluate Psi on the boundaries (i=1, i=imax)
   DO i = 1, imax, imax - 1
      DO k = 2, kmax - 1
         x_k = 0.5_wp * (xp(1,i,j,k+1) - xp(1,i,j,k-1))
         z_k = 0.5_wp * (xp(3,i,j,k+1) - xp(3,i,j,k-1))
         x_kk = xp(1,i,j,k+1) - 2.0_wp * xp(1,i,j,k) + xp(1,i,j,k-1)
         z_kk = xp(3,i,j,k+1) - 2.0_wp * xp(3,i,j,k) + xp(3,i,j,k-1)
         IF(abs(x_k) > abs(z_k)) THEN
            Psi(i,j,k) = -x_kk / x_k
         ELSE
            Psi(i,j,k) = -z_kk / z_k
         ENDIF
      ENDDO
   ENDDO
   ! Evaluate Pi on the boundaries (k=1, k=kmax)
   DO k = 1, kmax, kmax - 1
      DO i = 2, imax - 1
         x_i = 0.5_wp * (xp(1,i+1,j,k) - xp(1,i-1,j,k))
         z_i = 0.5_wp * (xp(3,i+1,j,k) - xp(3,i-1,j,k))
         x_ii = xp(1,i+1,j,k) - 2.0_wp * xp(1,i,j,k) + xp(1,i-1,j,k)
         z_ii = xp(3,i+1,j,k) - 2.0_wp * xp(3,i,j,k) + xp(3,i-1,j,k)
         IF(abs(x_i) > abs(z_i)) THEN
            Pi(i,j,k) = -x_ii / x_i
         ELSE
            Pi(i,j,k) = -z_ii / z_i
         ENDIF
      ENDDO
   ENDDO

   ! Evaluate Pi and Psi at interior points
   DO i = 2, imax - 1
      DO k = 2, kmax - 1
         !Psi(i,j,k) = UniformSpacing(Psi(1,j,k), Psi(imax,j,k), i, imax)
```

```
         Psi(i,j,k) = GridStretching(Psi(1,j,k), Psi(imax,j,k), i, imax, cy3)
         !Pi(i,j,k) = UniformSpacing(Pi(i,j,1), Pi(i,j,kmax), k, kmax)
         Pi(i,j,k) = GridStretching(Pi(i,j,1), Pi(i,j,kmax), k, kmax, cy2)
      ENDDO
   ENDDO
   END SUBROUTINE CalculatePiPsi


!--------------------------------------------------------------------------------!
   SUBROUTINE ThomasLoop()
!--------------------------------------------------------------------------------!
! Thomas method for solving tridiagonal matrix system
! This subroutine should be run in a pseudo time loop
   IMPLICIT NONE
   INTEGER :: i, j, k

   REAL(KIND=wp), DIMENSION(imax) :: a, b, c, d
   REAL(KIND=wp) :: x_ik, x_k, z_ik, z_k
   RMSres = 0.0_wp
   j = 1

   DO k = 2, kmax - 1
      ! Calculate governing equation w.r.t x-coordinate
      DO i = 1, imax
         IF( i == 1 .or. i == imax ) THEN
             a(i) = 0.0_wp
             b(i) = 1.0_wp
             c(i) = 0.0_wp
             d(i) = xp(1,i,j,k)
         ELSE
             a(i) = A1(i,j,k) * (1.0_wp - 0.5_wp * Pi(i,j,k))
             b(i) = -2.0_wp * (A1(i,j,k) + A3(i,j,k))
             c(i) = A1(i,j,k) * (1.0_wp + 0.5_wp * Pi(i,j,k))
             x_k = 0.5_wp * (xp(1,i,j,k+1) - xp(1,i,j,k-1))
             x_ik = 0.25_wp * ( xp(1,i+1,j,k+1) - xp(1,i+1,j,k-1) &
                               -xp(1,i-1,j,k+1) + xp(1,i-1,j,k-1) )
             d(i) = 2.0_wp * A2(i,j,k) * x_ik - A3(i,j,k) * ( xp(1,i,j,k+1) + &
                                                              xp(1,i,j,k-1) + &
                                                              Psi(i,j,k) * x_k )
         ENDIF
      ENDDO
      ! Call Thomas method solver
      CALL SY(1, imax, a, b, c, d)
      ! Update values at n+1 pseudo time
      DO i = 1, imax
         RMSres = RMSres + (d(i) - xp(1,i,j,k)) ** 2
         xp(1,i,j,k) = d(i)
      ENDDO

      ! Calculate governing equation w.r.t x-coordinate
       DO i =1, imax
         IF( i == 1 .or. i == imax ) THEN
             a(i) = 0.0_wp
             b(i) = 1.0_wp
             c(i) = 0.0_wp
             d(i) = xp(3,i,j,k)
         ELSE
             a(i) = A1(i,j,k) * (1.0_wp - 0.5_wp * Pi(i,j,k))
```

```
            b(i) = -2.0_wp * (A1(i,j,k) + A3(i,j,k))
            c(i) = A1(i,j,k) * (1.0_wp + 0.5_wp * Pi(i,j,k))
            z_k = 0.5_wp * (xp(3,i,j,k+1) - xp(3,i,j,k-1))
            z_ik = 0.25_wp * ( xp(3,i+1,j,k+1) - xp(3,i+1,j,k-1) &
                              -xp(3,i-1,j,k+1) + xp(3,i-1,j,k-1) )
            d(i) = 2.0_wp * A2(i,j,k) * z_ik - A3(i,j,k) * ( xp(3,i,j,k+1) + &
                                                             xp(3,i,j,k-1) + &
                                                             Psi(i,j,k) * z_k )
         ENDIF
      ENDDO
      ! Call Thomas method solver
      CALL SY(1, imax, a, b, c, d)
      ! Update values at n+1 pseudo time
      DO i = 1, imax
         RMSres = RMSres + (d(i) - xp(3,i,j,k)) ** 2
         xp(3,i,j,k) = d(i)
      ENDDO
   ENDDO
   END SUBROUTINE ThomasLoop


!-------------------------------------------------------------------------------!
   SUBROUTINE SY(IL,IU,BB,DD,AA,CC)
!-------------------------------------------------------------------------------!
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: IL, IU
   REAL(KIND=wp), DIMENSION(IL:IU), INTENT(IN) :: AA, BB
   REAL(KIND=wp), DIMENSION(IL:IU), INTENT(INOUT) :: CC, DD

   INTEGER :: LP, I, J
   REAL(KIND=wp) :: R

   LP = IL + 1

   DO I = LP, IU
      R = BB(I) / DD(I-1)
      DD(I) = DD(I) - R*AA(I-1)
      CC(I) = CC(I) - R*CC(I-1)
   ENDDO

   CC(IU) = CC(IU)/DD(IU)
   DO I = LP, IU
      J = IU - I + IL
      CC(J) = (CC(J) - AA(J)*CC(J+1))/DD(J)
   ENDDO
   END SUBROUTINE SY


!-------------------------------------------------------------------------------!
   SUBROUTINE CopyFrontTOBack()
!-------------------------------------------------------------------------------!
   USE SimulationVars_m, ONLY: imax, jmax, kmax, xp
   USE SimulationSetup_m, ONLY: UniformSpacing

   IMPLICIT NONE
   INTEGER :: i, k

   DO i = 2, imax - 1
```

```
      DO k = 2, kmax - 1
         xp(1,i,jmax,k) = xp(1,i,1,k)
         xp(2,i,jmax,k) = UniformSpacing(xp(2,i,jmax,1), xp(2,i,jmax,kmax), k, kmax)
         xp(3,i,jmax,k) = xp(3,i,1,k)
      ENDDO
   ENDDO

   END SUBROUTINE CopyFrontTOBack


!---------------------------------------------------------------------------------!
   SUBROUTINE CalculateGridJacobian()
!---------------------------------------------------------------------------------!
   USE SimulationVars_m, ONLY: imax, jmax, kmax, xp, inverseJacobian

   IMPLICIT NONE
   INTEGER :: i, j, k
   ! xgst, ygst, zgst: arbitrary ghost cell points
   REAL(KIND=wp) :: x_i, y_i, z_i, x_j, y_j, z_j, x_k, y_k, z_k, &
                    xgst, ygst, zgst

   DO i = 1, imax
      DO j = 1, jmax
         DO k = 1, kmax
            ! calculate x_i, y_i, z_i
            IF ( i == 1 ) THEN
               xgst = xp(1,i,j,k) - (xp(1,i+1,j,k) - xp(1,i,j,k))
               ygst = xp(2,i,j,k) - (xp(2,i+1,j,k) - xp(2,i,j,k))
               zgst = xp(3,i,j,k) - (xp(3,i+1,j,k) - xp(3,i,j,k))
               x_i = 0.5_wp * (xp(1,i+1,j,k) - xgst)
               y_i = 0.5_wp * (xp(2,i+1,j,k) - ygst)
               z_i = 0.5_wp * (xp(3,i+1,j,k) - zgst)
            ELSEIF ( i == imax ) THEN
               xgst = xp(1,i,j,k) + (xp(1,i,j,k) - xp(1,i-1,j,k))
               ygst = xp(2,i,j,k) + (xp(2,i,j,k) - xp(2,i-1,j,k))
               zgst = xp(3,i,j,k) + (xp(3,i,j,k) - xp(3,i-1,j,k))
               x_i = 0.5_wp * (xgst - xp(1,i-1,j,k))
               y_i = 0.5_wp * (ygst - xp(2,i-1,j,k))
               z_i = 0.5_wp * (zgst - xp(3,i-1,j,k))
            ELSE
               x_i = 0.5_wp * (xp(1,i+1,j,k) - xp(1,i-1,j,k))
               y_i = 0.5_wp * (xp(2,i+1,j,k) - xp(2,i-1,j,k))
               z_i = 0.5_wp * (xp(3,i+1,j,k) - xp(3,i-1,j,k))
            ENDIF
            ! calculate x_j, y_j, z_j
            IF ( j == 1 ) THEN
               xgst = xp(1,i,j,k) - (xp(1,i,j+1,k) - xp(1,i,j,k))
               ygst = xp(2,i,j,k) - (xp(2,i,j+1,k) - xp(2,i,j,k))
               zgst = xp(3,i,j,k) - (xp(3,i,j+1,k) - xp(3,i,j,k))
               x_j = 0.5_wp * (xp(1,i,j+1,k) - xgst)
               y_j = 0.5_wp * (xp(2,i,j+1,k) - ygst)
               z_j = 0.5_wp * (xp(3,i,j+1,k) - zgst)
            ELSEIF ( j == jmax ) THEN
               xgst = xp(1,i,j,k) + (xp(1,i,j,k) - xp(1,i,j-1,k))
               ygst = xp(2,i,j,k) + (xp(2,i,j,k) - xp(2,i,j-1,k))
               zgst = xp(3,i,j,k) + (xp(3,i,j,k) - xp(3,i,j-1,k))
               x_j = 0.5_wp * (xgst - xp(1,i,j-1,k))
               y_j = 0.5_wp * (ygst - xp(2,i,j-1,k))
```

```
               z_j = 0.5_wp * (zgst - xp(3,i,j-1,k))
            ELSE
               x_j = 0.5_wp * (xp(1,i,j+1,k) - xp(1,i,j-1,k))
               y_j = 0.5_wp * (xp(2,i,j+1,k) - xp(2,i,j-1,k))
               z_j = 0.5_wp * (xp(3,i,j+1,k) - xp(3,i,j-1,k))
            ENDIF
            ! calculate x_k, y_k, z_k
            IF ( k == 1 ) THEN
               xgst = xp(1,i,j,k) - (xp(1,i,j,k+1) - xp(1,i,j,k))
               ygst = xp(2,i,j,k) - (xp(2,i,j,k+1) - xp(2,i,j,k))
               zgst = xp(3,i,j,k) - (xp(3,i,j,k+1) - xp(3,i,j,k))
               x_k = 0.5_wp * (xp(1,i,j,k+1) - xgst)
               y_k = 0.5_wp * (xp(2,i,j,k+1) - ygst)
               z_k = 0.5_wp * (xp(3,i,j,k+1) - zgst)
            ELSEIF ( k == kmax ) THEN
               xgst = xp(1,i,j,k) + (xp(1,i,j,k) - xp(1,i,j,k-1))
               ygst = xp(2,i,j,k) + (xp(2,i,j,k) - xp(2,i,j,k-1))
               zgst = xp(3,i,j,k) + (xp(3,i,j,k) - xp(3,i,j,k-1))
               x_k = 0.5_wp * (xgst - xp(1,i,j,k-1))
               y_k = 0.5_wp * (xp(2,i,j,k+1) - ygst)
               z_k = 0.5_wp * (xp(3,i,j,k+1) - zgst)
            ELSEIF ( k == kmax ) THEN
               xgst = xp(1,i,j,k) + (xp(1,i,j,k) - xp(1,i,j,k-1))
               ygst = xp(2,i,j,k) + (xp(2,i,j,k) - xp(2,i,j,k-1))
               zgst = xp(3,i,j,k) + (xp(3,i,j,k) - xp(3,i,j,k-1))
               x_k = 0.5_wp * (xgst - xp(1,i,j,k-1))
               y_k = 0.5_wp * (ygst - xp(2,i,j,k-1))
               z_k = 0.5_wp * (zgst - xp(3,i,j,k-1))
            ELSE
               x_k = 0.5_wp * (xp(1,i,j,k+1) - xp(1,i,j,k-1))
               y_k = 0.5_wp * (xp(2,i,j,k+1) - xp(2,i,j,k-1))
               z_k = 0.5_wp * (xp(3,i,j,k+1) - xp(3,i,j,k-1))
            ENDIF
            ! Calculate 1/J: Inverse of grid Jacobian
            inverseJacobian(i,j,k) = x_i * (y_j * z_k - y_k * z_j) - &
                                     x_j * (y_i * z_k - y_k * z_i) + &
                                     x_k * (y_i * z_j - y_j * z_i)
         ENDDO
      ENDDO
   ENDDO

   END SUBROUTINE CalculateGridJacobian


END MODULE
```

2.3.8 SimulationSetup.F90

```
!> \file SimulationSetup.F90
!> \author Sayop Kim

MODULE SimulationSetup_m
   USE Parameters_m, ONLY: wp
   IMPLICIT NONE

   PUBLIC :: InitializeCommunication, UniformSpacing, GridStretching
```

```
CONTAINS

!------------------------------------------------------------------------------!
   SUBROUTINE InitializeCommunication()
!------------------------------------------------------------------------------!
      USE Parameters_m, ONLY: CODE_VER_STRING
      IMPLICIT NONE

      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "CFD code Version: ", CODE_VER_STRING
   END SUBROUTINE InitializeCommunication


!------------------------------------------------------------------------------!
   FUNCTION UniformSpacing(xmin,xmax,indx,indxMax) RESULT(outcome)
!------------------------------------------------------------------------------!
      !Distribute interior grid points based on edge points' coordinates.
      !Linear Interpolateion is made by referring to (i,j,k) indices
      IMPLICIT NONE
      REAL(KIND=wp), INTENT(IN) :: xmin, xmax
      INTEGER, INTENT(IN) :: indx, indxMax
      REAL(KIND=wp) :: outcome, coef
      coef = REAL(indx - 1) / REAL(indxMax - 1)
      outcome = xmin + coef * (xmax - xmin)
   END FUNCTION UniformSpacing

!------------------------------------------------------------------------------!
   FUNCTION GridStretching(xmin,xmax,indx,indxMax,cy) RESULT(outcome)
!------------------------------------------------------------------------------!
      !Distribute interior grid points based on stretching coefficient
      !Interpolateion is made by referring to (i,j,k) indices

      IMPLICIT NONE
      REAL(KIND=wp) :: cy
      REAL(KIND=wp), INTENT(IN) :: xmin, xmax
      INTEGER, INTENT(IN) :: indx, indxMax
      REAL(KIND=wp) :: outcome, coef
      coef = log(1.0_wp + (exp(-cy) - 1.0_wp) * REAL(indx - 1) / REAL(indxMax - 1))
      outcome = xmin - coef * (xmax - xmin) / cy
   END FUNCTION GridStretching

END MODULE SimulationSetup_m
```