GridGen: an Elliptical Grid Generator

Release

Marc Salvadori

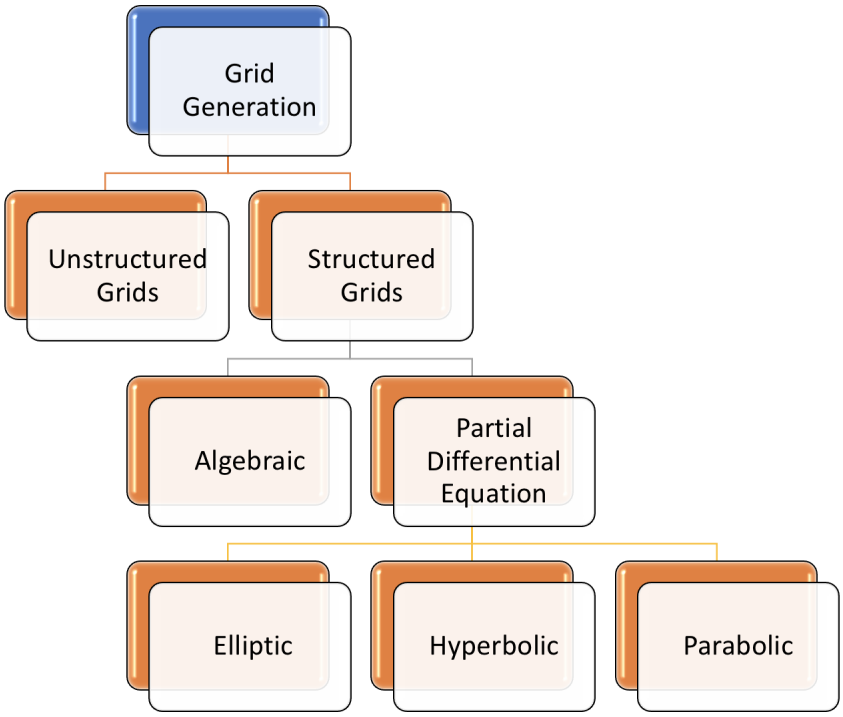
Feb 28, 2018

CONTENTS

Author: Marc Salvadori

Email: msalvadori3@gatech.edu

The basic idea behind grid generation is the creation of the transformation laws between the physical space and the computational space. The different techniques to generate a grid can be summarized in the following diagram.



Elliptic grid generation is one of several methods used to generate structured grids for complex geometries. Algebraic methods are one commonly used alternative, and hyperbolic systems of equations are sometimes used, particularly for external flows. In this project it is required solve a pair of Laplace/Poisson equations to generate the mesh. Rather than work in the physical domain where the geometry is complicated and the equations are simple, we prefer to work in the computational domain where just the opposite is true. Once we have done our work in the computational domain, we bring the results back to the physical domain for viewing.

The overall procedure to generate a grid is as follows:

1. Establish the transformation relations between the physical space and the computational space.
2. Transform the governing equations and the boundary conditions into the computational space.
3. Solve the equations in the computational space using the uniformly spaced rectangular grid.
4. Perform a reverse transformation to represent the flow properties in the physical space.

CHAPTER

ONE

CONTENTS

1.1 Project Description

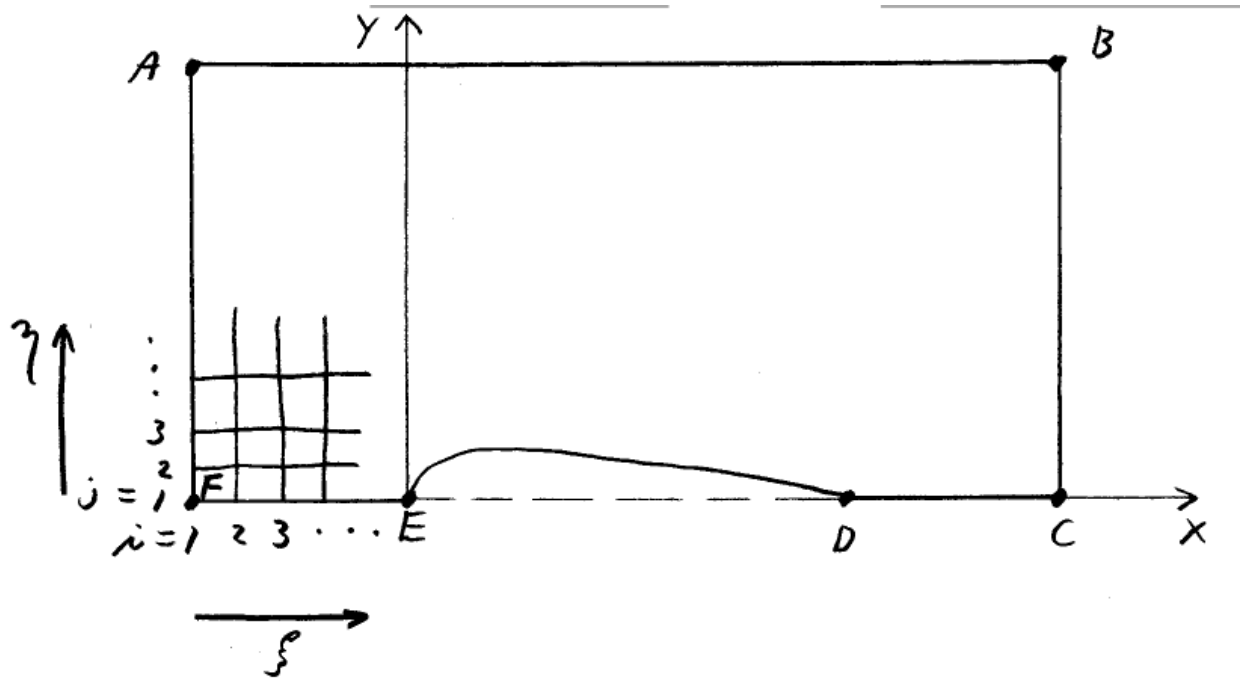
1.1.1 Given task

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 Deliverables

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 Setup

1.2.1 Setting up a Grid

1.2.2 Code Structure

The **|GridGen|** code is divided into three major parts:

1. **Input**: User input is entered into a file called `input_file.xml`.
2. **Main Program**: The main program for solving the elliptical grid is `gridgen.F90` in the `<parent directory>/src/main` folder.
3. **Output**: Once the code is run, the results are stored in the `<parent directory>/output/` folder.

1.2.3 Input File

The inputs for GridGenerator are specified in the `input_file.xml`. The input file is accessed as follows:

1. Go to the parent directory:

```
$ cd <path to |GridGen|>
```

Make sure that you are in the directory that contains the files **setup.py**, **input_file.xml**, and the folder **src**.

2. Open the input file:

```
$ vi input_file.xml
```

3. The main parts of the input file are shown below

The input file is divided into different parts. The geometry is set in the <geometry> module, which is shown below:

```
<geometry>
  <x1>-0.8</x1>
  <y1>0.0</y1>
  <x2>1.8</x2>
  <y2>0.0</y2>
  <x3>-0.8</x3>
  <y3>1.0</y3>
  <x4>1.8</x4>
  <y4>1.0</y4>
  <FEsize>11</FEsize>
  <DCsize>11</DCsize>
  <Geosize>21</Geosize>
  <Geoptx1>0.0</Geoptx1>
  <Geopty1>0.0</Geopty1>
  <Geoptx2>1.0</Geoptx2>
  <Geopty2>0.0</Geopty2>
  <imax>41</imax>
  <jmax>19</jmax>
</geometry>
```

For setup of the solver, most of the inputs are set in the <setup> module. A snippet of this module is shown below:

```
<setup>
  <Project>2D_Grid_Generator</Project>
  <nmax>500</nmax>
  <iControl>1</iControl>
  <Cy>2.0</Cy>
  <RMSres>1.0e-6</RMSres>
</setup>
```

1.2.4 Compilation

The following sequence of commands is used to compile a single simulation.

1. Go to the parent directory:

```
$ cd <path to |GridGen|>
```

Make sure that you are in the directory that contains the files **setup.py**, **input_file.xml**, and the folder **src**.

2. Clean existing results:

```
$ ./setup.py -u clean heavy
```

This command removes the existing files in the output folder |GridGen|/output/ and deletes the object files from previous compilations. **Backup any required results before using this command.**

3. Set working directory path:

```
$ ./setup.py -u set_path
```

   This command sets the working directory path

4. Set the output directory:

```
$ ./setup.py -u create_subdirectories
```

5. Compile the build:

```
$ ./setup.py -e configure
```

   An empty CMake window opens. Press `[c]` on the keyboard to configure the program.

   This brings up the CMake window. There are two options for the `CMAKE_BUILD_TYPE` :

   - `Release`: This compiles the program in regular mode; debugging flags are disabled.
   - `Debug`: This compiles the program in debug mode; errors and warnings are displayed on the terminal.

   Press `[Enter]` on the keyboard to edit the option (to change from `Release` to `Debug` or vice versa)

   The file **`grid.x`** will now be generated in the parent directory

6. Execute the program:

```
$ ./grid.x
```

   This command runs the program. If Debug mode is enabled in **`GRIDGEN_COMPILE_DEFS`**, appropriate output is printed on the Terminal screen.

1.2.5 Results

The results are stored in the `output/` folder inside the parent directory. The output directory contains several files **.dat** and **.tec** where the calculations are written. In addition, there is also a `output/plot` folder, where figures from the calculated data are plotted. To plot the results. open the inputfile and enter the name of the **.dat** file that was generated in the `files` entry (as shown below).

```
<PostProcessing>
    <plot>
       <files>RESULTDATFILE</files>
       ...
    </plot>
</PostProcessing>
```

Then, from the parent directory execute the following command to plot the results using **|GridGen|**'s inbuilt plotting utility:

```
$ ./setup.py -p single_plot
```

This will generate the RMS line plots from the results, which will be stored in the `output/plot` folder.

1.3 Input File

The input file is central location for setting any and all parameters for all the features in |GridGen|. Depending on the type of operation being performed on |GridGen|, the relevant input options are set in the input file.

1.3.1 Structure of the Geometry Module

```
<geometry>
    <x1>-0.8</x1>
    <y1>0.0</y1>
    <x2>1.8</x2>
    <y2>0.0</y2>
    <x3>-0.8</x3>
    <y3>1.0</y3>
    <x4>1.8</x4>
    <y4>1.0</y4>
    <FEsize>11</FEsize>
    <DCsize>11</DCsize>
    <Geosize>21</Geosize>
    <Geoptx1>0.0</Geoptx1>
    <Geopty1>0.0</Geopty1>
    <Geoptx2>1.0</Geoptx2>
    <Geopty2>0.0</Geopty2>
    <imax>41</imax>
    <jmax>19</jmax>
 </geometry>
```

1.3.2 Structure of Setup Module

```
<setup>
   <Project>2D_Grid_Generator</Project>
   <nmax>500</nmax>
   <iControl>1</iControl>
   <Cy>2.0</Cy>
   <RMSres>1.0e-6</RMSres>
 </setup>
```

1.3.3 Structure of Postprocessing Modules

|GridGen| also allows the user to graphically visualize the results through the use of graphs and contours. Inputs to this plotting utility are also provided through the input file. The relevant block for this utility is shown below, and linked to the dedicated plotting utility page.

```
<PostProcessing>
  <plot>
      <iPost>1</iPost>
      <Method>TecPlot</Method>
      <files>rmslog.dat</files>
      <style>k +r</style>
      <label>Grid</label>
      <LegendFontSize>16</LegendFontSize>
      <FigureSize>10 7</FigureSize>
```

```
      <AxisLabelSize>21</AxisLabelSize>
      <AxisTitleSize>22</AxisTitleSize>
      <XTickSize>23</XTickSize>
      <YTickSize>24</YTickSize>
  </plot>
</PostProcessing>
```

1.4 Code development

The current project is for developing elliptic grid generator in 2-dimensional domain. Hereafter, the program developed in this project is called 'GridGen'.

1.4.1 GridGen Code summary

The present project is to make a grid-generator for 2-D computational domain around a modified NACA 00xx series airfoil in a channel.

The source code contains the following directories:

- io - input/output related routines
- main - main program driver
- math - thomas algorithm
- modules - main grid generator solver routines
- utils - list of useful FORTRAN utilities used within the program
- gridpy - python wrapper for gridgen main program

  Also a 'CMakeLists.txt' file is also included for cmake compiling.

```
$ cd GridGen/src/
$ ls
$ CMakeLists.txt  io  main math modules utils gridpy
```

The **io** folder has **io.F90** file which contains **ReadInput(inputData)** subroutine. It also includes **input_file_xml** which describes the structure of the user run-time input file located in the main 'src' directory, and **output.F90** for storing data in bothb Tecplot and Python format.

The **main** folder is only used for containing the code driver file. The main routines is run by **gridgen.F90** which calls important subroutines from the rest of folders.

1.4.2 Details of GridGen development

The GridGen code is made for creating 2-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 **gridgen.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 xml_data_input_file
      USE GridSetup_m,ONLY:InitGrid,GridInternal,EndVars
      USE parameters_m,ONLY:wp
      USE SimVars_m,ONLY:fileLength
      USE GridTransform_m,ONLY: InitArrays,CalcCoeff,PhiPsi,TriDiag,Jacobian,&
            EndArrays
      USE output_m,ONLY:WritePlotFile,WriteRMS

      IMPLICIT NONE

      TYPE(input_type_t) :: inputData
      CHARACTER(LEN=fileLength) ::output= 'none'
      CHARACTER(LEN=fileLength) :: rmsout = 'rmslog.dat'
      INTEGER :: i
      REAL(KIND=wp) :: rms

      CALL InitGrid(inputData)
      CALL InitArrays
      CALL Jacobian
      CALL WritePlotFile('InitGrid','"x","y","Jacobian","Phi","Psi"',inputData)
      IF (inputData%setup%iControl == 1) CALL PhiPsi
      WRITE(*,*) 'Solving Elliptical Grid.............'
      DO i=1,inputData%setup%nmax
                WRITE(*,*) 'IT = ',i
                CALL CalcCoeff
                CALL TriDiag(rms)
                CALL WriteRMS(i,rms,rmsout)
                IF (rms <= inputData%setup%RMSres) EXIT
      ENDDO
      CALL Jacobian
      CALL WritePlotFile(output,'"x","y","Jacobian","Phi","Psi"',inputData)
      CALL EndArrays
      CALL EndVars

END PROGRAM main
```

Creation of algebraic grid points

The code starts to run by reading the important input parameters defined by the user in the **input_file.xml** file. The input data file first contains all the details geometrical description of our 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) coordinates. Thus those points forms a 2-dimensional surface. 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 **gridgen.F90** file first refers to **InitGrid** 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 InitGrid(inputData)

        USE xml_data_input_file
        USE io_m, ONLY: ReadInput
```

```
        IMPLICIT NONE

        TYPE(input_type_t) :: inputData

        CALL ReadInput(inputData)
        CALL InitVars()
        CALL BottomEdge()
        CALL SetBCs()
        CALL GridInternal()

END SUBROUTINE InitGrid
```

- **ReadGridInput**: Reads user defined variables and parameters for grid configuration.
- **InitVars**: Initialize the single- and multi-dimensional arrays and set their size based on the input parameters.
- **BottomEdge**: Generate point values for airfoil geometry.
- **SetBcs**: Generate grid points along 4 edges of the computational domain.
- **GridInternal**: 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:

$$\begin{aligned}\alpha &= x_\eta^2 + y_\eta^2 \\ \beta &= x_\xi x_\eta + y_\xi y_\eta \\ \gamma &= x_\xi^2 + y_\xi^2\end{aligned}$$

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.

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

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

$$\begin{aligned}a_{i,j} = e_{i,j} &= \alpha_{i,j}^n \left(1 - \frac{\phi_{i,j}^n}{2}\right) \\ b_{i,j} = f_{i,j} &= -2\left(\alpha_{i,j} + \gamma_{i,j}\right) \\ c_{i,j} = g_{i,j} &= \alpha_{i,j}^n \left(1 + \frac{\phi_{i,j}^n}{2}\right)\end{aligned}$$

$$e_{i,j} = \frac{\beta_{i,j}^n}{2}\left(x_{i+1,j}^n - x_{i+1,j-1}^{n+1} - x_{i-1,j+1}^n - x_{i-1,j-1}^{n+1}\right) - \gamma_{i,j}^n\left(x_{i,j+1}^n + x_{i,j-1}^{n+1}\right) - \frac{\beta_{i,j}^n}{2}\psi_{i,j}^n\left(x_{i,j+1}^n - x_{i,j-1}^{n+1}\right)$$

$$h_{i,j} = \frac{\beta_{i,j}^n}{2}\left(y_{i+1,j}^n - y_{i+1,j-1}^{n+1} - y_{i-1,j+1}^n - y_{i-1,j-1}^{n+1}\right) - \gamma_{i,j}^n\left(y_{i,j+1}^n + y_{i,j-1}^{n+1}\right) - \frac{\beta_{i,j}^n}{2}\psi_{i,j}^n\left(y_{i,j+1}^n - y_{i,j-1}^{n+1}\right)$$

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. **gridgen.F90** file contains the necessary subroutine calls to evaluate the elipptical grid as shown below:

```
IF (inputData%setup%iControl == 1) CALL PhiPsi
WRITE(*,*) 'Solving Elliptical Grid.............'
DO i=1,inputData%setup%nmax
          WRITE(*,*) 'IT = ',i
          CALL CalcCoeff
          CALL TriDiag(rms)
          CALL WriteRMS(i,rms,rmsout)
          IF (rms <= inputData%setup%RMSres) EXIT
ENDDO
```

Before going into the main loop for solving poisson equations, the code calculate control terms with ϕ and ψ only if the user defines the option in the **input_file.xml** file. 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.

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 j = constant line.

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.

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

where $N = 2x(\mathrm{imax}-2)x(\mathrm{jmax}-2)$ and the RMS criterion is user-specified as as small number or by default it is set as: $1\mathrm{x}10^{-6}$. In this code, the convergend is assumed to be achieved when RMS residual is less than the RMS criterion.

1.5 GridPy

GridPy is a python-based library of |GridGen| that is developed to assist the user with i/o procedures, utilities and code options/testing. In the following, the commands are listed by category and discussion is provided in each respective section.

1.5.1 Compilation Options

|GridGen| has several builtin compilation options, that can be accessed through the command **`./setup.py -e`** and the utilities that can be accessed using **`./setup.py -u`**. These are described here.

Configure

The default method for compiling |GridGen| from scratch is using the command:

```
$ ./setup.py -e configure gridgen
```

This generates a CCMake window with configuration options that can be chosen by the user. Refer to the *compilation section* of the *|GridGen| Setup page* for details on using this method.

Compile

This option should be used only if |GridGen| has been configured first (using `-e configure`). This recompiles the code with any changes, while retaining the build directory and related objects. It can be executed using the command:

```
$ ./setup.py -e compile gridgen
```

1.5.2 Setting the path

|GridGen| requires the path to the working directory be set every time a run is executed from scratch (i.e. after clearing all the compilations). The |GridGen| executable `grid.x` will not run without this path set. To set the path, compile/configure |GridGen| using any of the options, and then run:

```
$ ./setup.py -u set_path
```

1.5.3 Cleaning commands

After completion of a simulation, or before running a fresh simulation, |GridGen| can be cleared of compiled objects, results and other files. There are three variants to clean |GridGen|. The first is to perform a complete clean, which removes the build directories, the executables and any generated results. This can be accomplished by running:

```
$ ./setup.py -u clean heavy
```

On the other hand, the build directories and executables can be retained while deleting only the results by running the command:

```
$ ./setup.py -u clean results
```

The last variant is where the build directories alone are cleared, retaining the results and the executables, which is done using the command:

```
$ ./setup.py -u clean
```

1.5.4 Plotting Utility

|GridGen| has a builtin plotting utility that allows for plotting of the generated data without the use of external tools. Similar to all the other features, the plotting utility is also accessed through the input file, which is accessed as follows.

1. Go to the parent directory:

   ```
   $ cd <path to |GridGen|>
   ```

Make sure that you are in the directory that contains the files **setup.py**, **input_file.xml**, and the folder **src**.

2. Open the input file:

```
$ vi input_file.xml
```

The section of the inputfile devoted to post-processing is shown below:

```
<PostProcessing>
  <plot>
    <iPost>1</iPost>
    <Method>TecPlot</Method>
    <files>file1.dat</files>
    <style>k +r -c :g -.y --m</style>
    <label>Var1</label>
    <LegendFontSize>16</LegendFontSize>
    <FigureSize>10 7</FigureSize>
    <AxisLabelSize>21</AxisLabelSize>
    <AxisTitleSize>22</AxisTitleSize>
    <XTickSize>23</XTickSize>
    <YTickSize>24</YTickSize>
  </plot>
</PostProcessing>
```

Note: Before running the plotting tool it is required to run create the subdirectory output.

Options in the plotting module

1. The data files generated from a |GridGen| run (or any external data file) is placed in the **output** folder. The name of the file should not have any spaces or special characters (like colon :, quotation marks “” or ”, brackets or parenthesis () [] etc). The file name is entered into the `<files>` field in the input file. If there are more than 1 file, they are entered one after another.
2. The `<style>` entry refers to the line style and color used. There are seven available colors in Python by default: RGBCMYK (Red, Green, Blue, Cyan, Magenta, Yellow and Black). Markers can be placed on the lines using the marker symbols (eg. `+r` generates a red line with + shaped markers). Line styles can also be changed using appropriate symbols (eg. `--m` generates a dashed magenta line, while `-.y` generates a dot-dashed yellow line).
3. The `<label>` entry is to populate the legend. If there are N files for N different variables, then the legend is populated according to the entries in this field.
4. The remaining entries are to adjust the figure parameters, like font size and figure size.

Using the plotting module

1. Enter the names of the data files in the input file
2. Set the required number of legend entries and line styles, depending on how many files/variables are being plotted.
3. Adjust any other plot parameters as required.
4. The plotting module also provides a generalized feature to make plots from input data. This feature is also accessed through the `input_file.xml`. The input data (.dat) files are entered in the `files` section of the input file. The plotting utility is then accessed using the command:

```
$ ./setup.py -p single_plot options "{<plot options>}"
```

An example of this input for the plotting utility is:

```
$ ./setup.py -p single_plot options "{'title':['RMS Residual for Grid#5'],
  'xlabel':['Number of Iterations'],'ylabel':['RMS'],
  'grid':['on'],'legend':['False'],'legend_frame':['True'],
  'legend_loc':['lower right']}"
```

The options give flexibility to set the graph title, the labels for the axes, select the columns of data from the input .dat file t

Note that if no options are provided as in the example:

```
$ ./setup.py -p single_plot options
```

default options wil be used for the picture name and axes/title labels. Note that the file and picture options must be provided in the input file, in the `<PostProcessing>` section.

Note: The above command is meant to be used with a generic .dat file, without header, where data is distributed column-wise.

1.5.5 Documentation

The |GridGen| documentation is written using restructured text (reST) Sphinx, and can be manipulated using python routines in Gridpy.

Building

The documentation can be built using the command:

```
$ ./setup.py -u doc build
```

This runs Sphinx in the documentation directory, and creates the html files using a make command. The command runs Sphinx twice to make sure references are interlinked properly.

Viewing

This command is used for viewing the documentation. It is only available after running the build command above, so that the documentation is created. The documentation can be opened using the command:

```
$ ./setup.py -u doc open
```

The documentation requires a compatible version of a web broswer (preferably Google Chrome or Mozilla Firefox). If a compatible browser is installed, the documentation will open when the above command is executed. Otherwise, an error message will be displayed. In case an error message is displayed, the documentation can manually be opened on a web browser from the directory **<GridGen parent directory>/doc/_build/html/index.html**

Note: If using MAC OS X please be aware to simply run $ open **<GridGen parent directory>/doc/_build/html/index.html** from the terminal.

Cleaning

The built documentation can be cleaned (i.e. all the compiled files can be deleted while retaining the source content) using the command:

```
$ ./setup.py -u doc clean
```

Closing Notes

1. As with any browser-based content, the appearance of the content is dependent on the capability of the browser to render the elements on the webpage. Depending on the bworser present on the machine, the content may appear different.

1.6 Evaluation Cases

1.6.1 Grid testcases

2D Airfoil Grid Results Summary

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

The figure below shows the grid points 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 on 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 in leading and trailing edges of the airfoil.

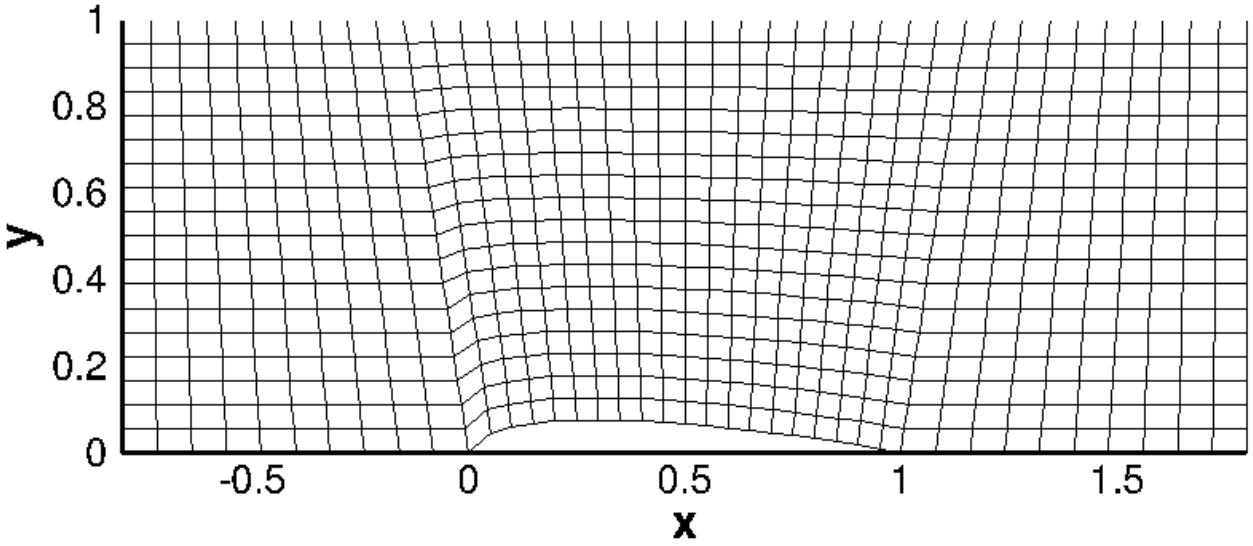


Fig. 1.1: :Grid points alignment of Grid #1

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.

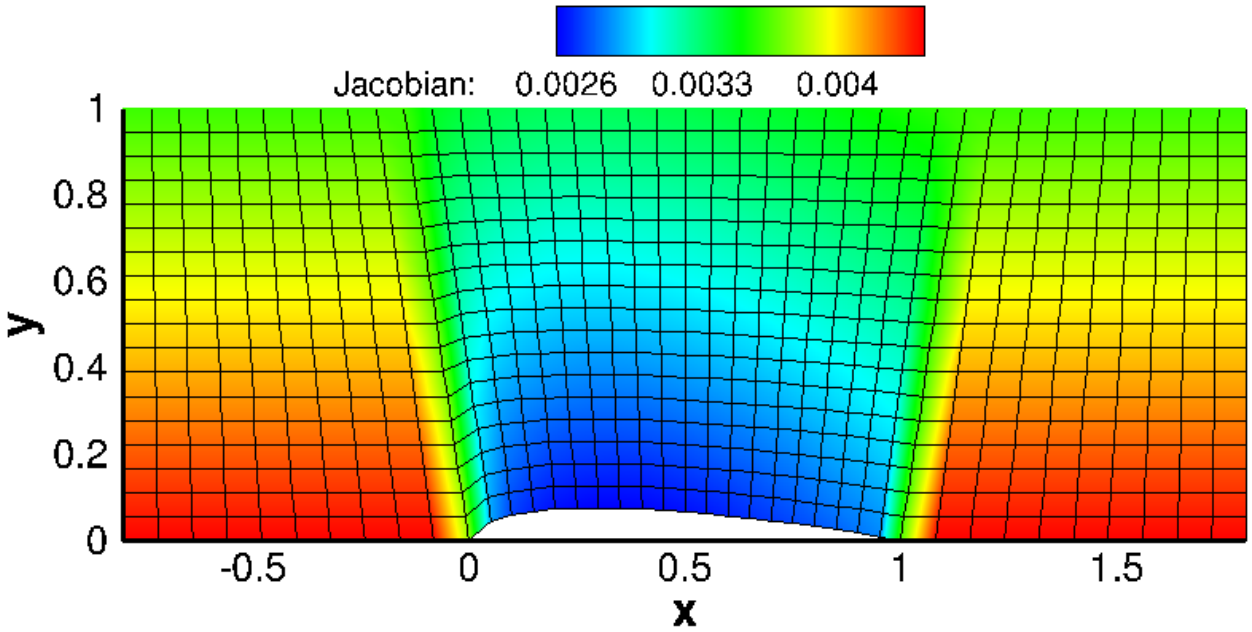


Fig. 1.2: :Inverse Grid Jacobian distribution of Grid #1

Grid #2: Algebraic grid with clustered points in y

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.

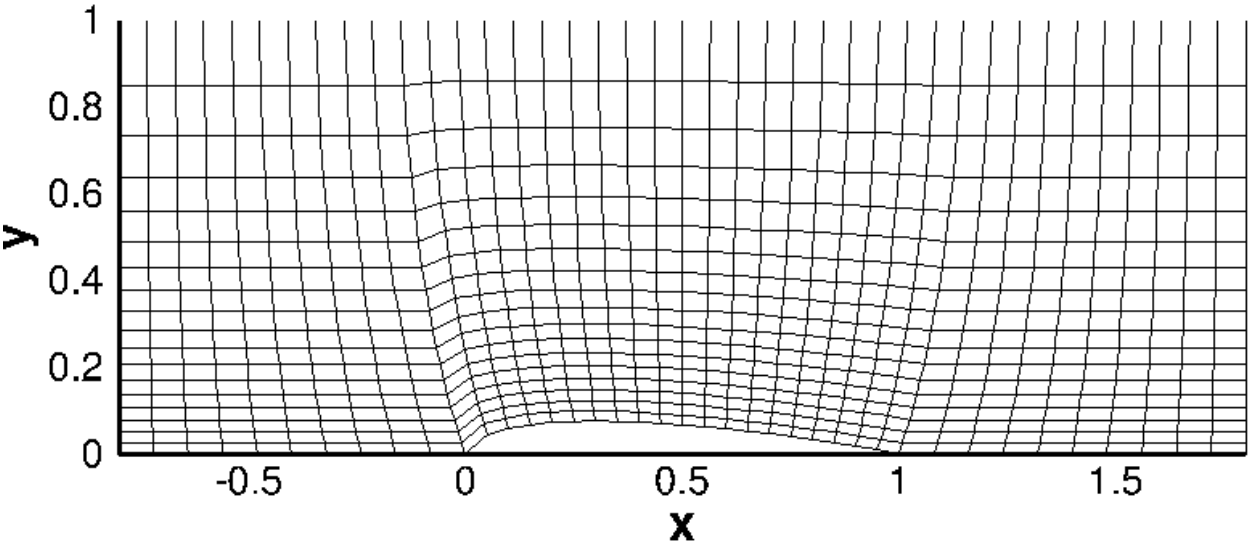


Fig. 1.3: :Grid points alignment of Grid #2

The grid Jacobian contour is shown below. Applying grid stretching along the y direction gives big cell volume

distribution. Change in volume along the bottom edge looks 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. The same situation happens at the trailing point of the airfoil. In some points, 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 and therfeore we need more dense grid points in this region.

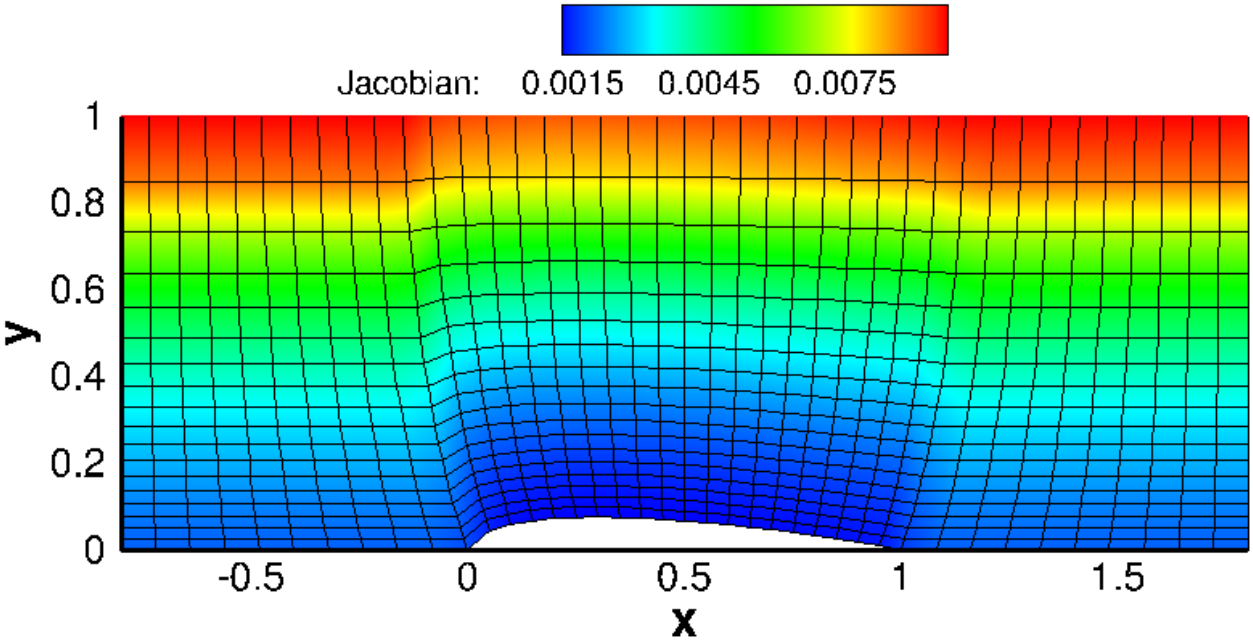


Fig. 1.4: :Inverse Grid Jacobian distribution of Grid #2

Grid #3: Elliptic grid with clustered points in y & 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 alignments that resemble iso-stream lines and iso-potential lines around the airfoil body. This is because the set of Poisson equation is exactly the same as a set of stream function and potention function when the control terms are ignored.

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 issues in the flux evaluation of flow properties across the j-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.

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

The problem that arise in Grid #3 case was able to be resolved by adding control terms in the 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 not be critical.

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

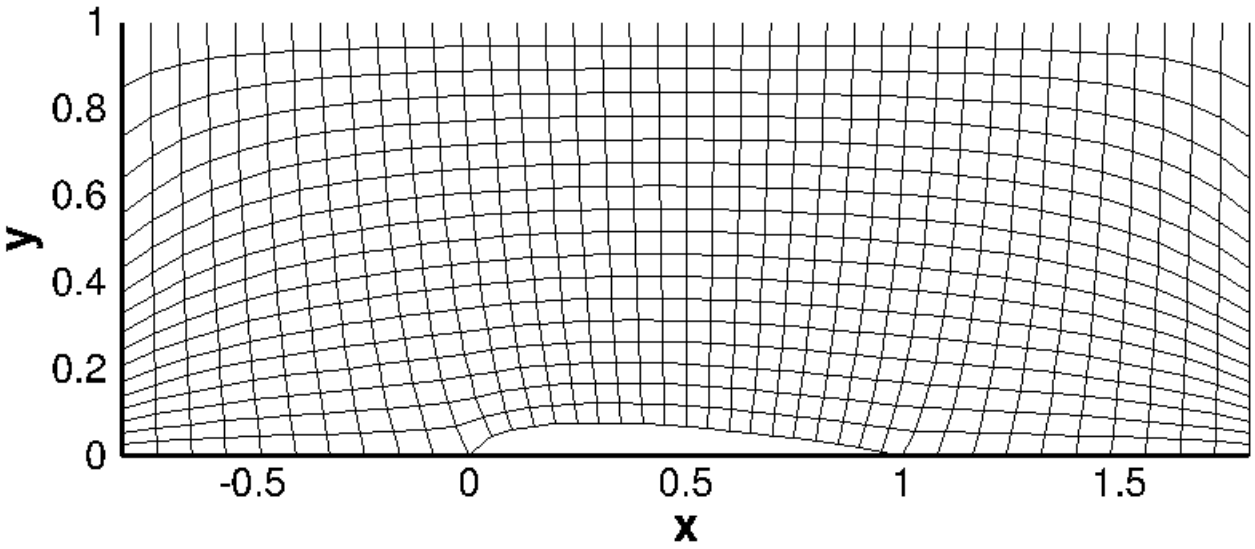


Fig. 1.5: :Grid points alignment of Grid #3

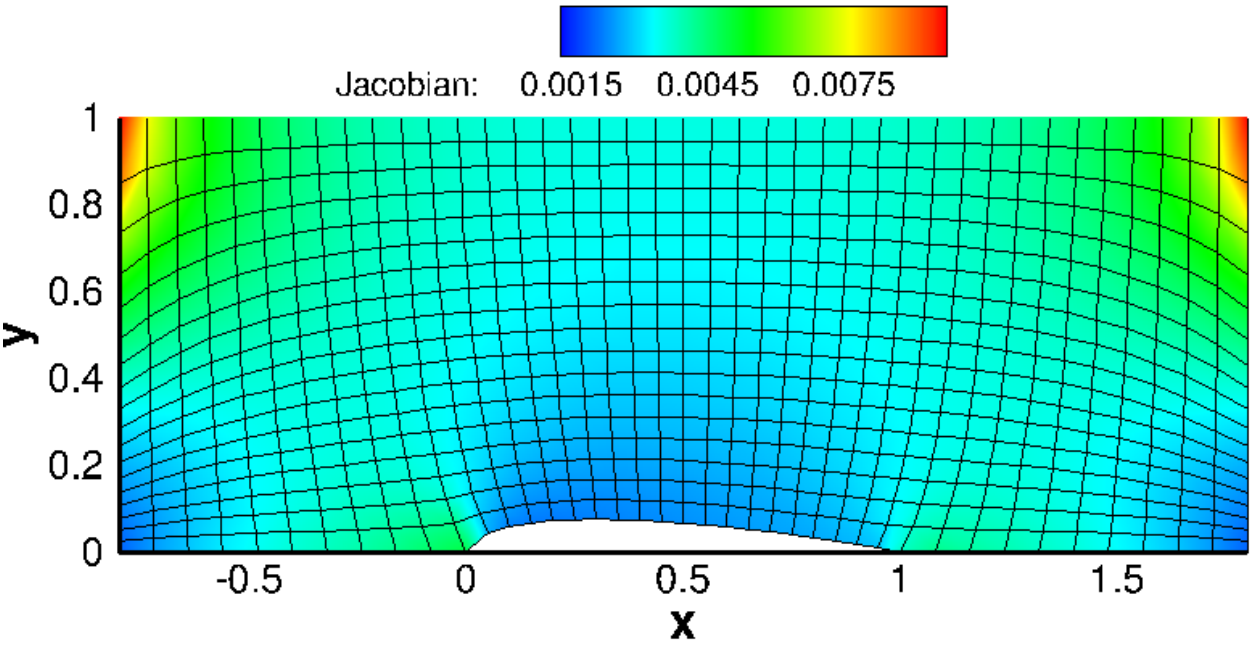


Fig. 1.6: :Inverse Grid Jacobian distribution of Grid #3

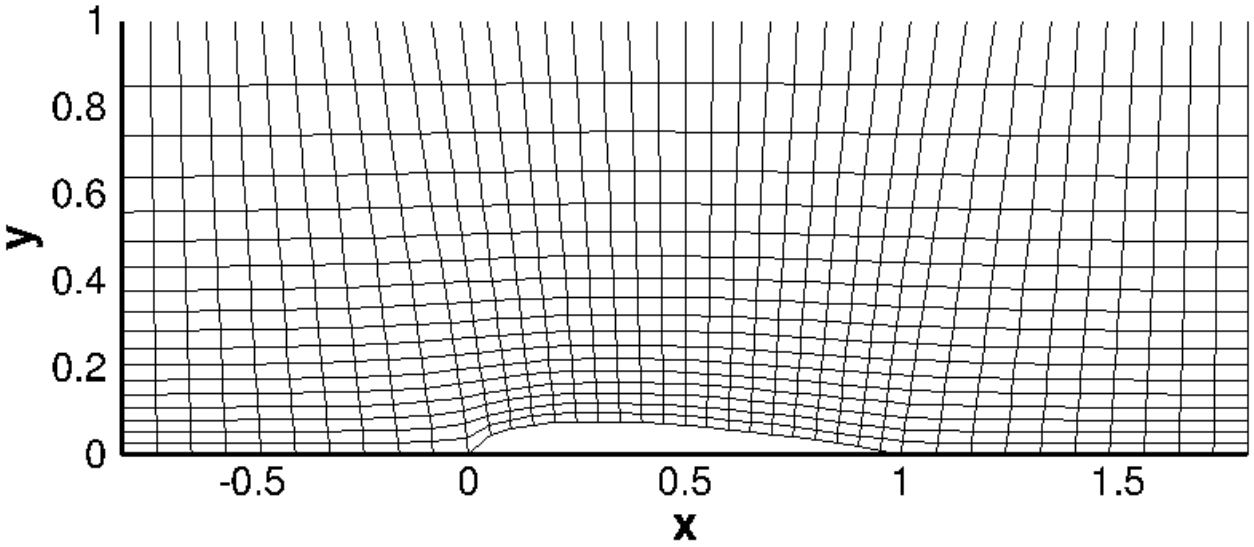


Fig. 1.7: :Grid points alignment of Grid #4

angle between airfoil arc and i-constant line anchored at the leading edge.

Grid #5: Improved elliptic grid

Based on the above grid results we can conclude that the best grid quality is Grid#4. Although Grid#4 shows promixing results, the following issues are still to be resolved:

1. At both the leading edge and trailing edge of the airfoil there is still some sudden variation in the cell size.
2. Large variation in skeweness of the grid is observed when control terms are applied.

To handle the above problems the following approach is implemented:

1. Maintain the grid stretched along the y direction with the same Cy.
2. Provide smooth cell size transition along the segments FE,ED, and DC by employing testing varying values of Cy.
3. Provide new Cy parameters for stretching in Psi and Phi

The following values of Cy were chosen for the best grid quality:

Method	Cy
Stretching along y	2.0
Stretching along FE	-1.0
Stretching along ED	1.0
Stretching along DC	0.001
Stretching in Phi	-11.0
Stretching in Psi	0.001

According to the above table a stretching method was applied to the bottom segment of the grid in order to improve the smoothness of the cell variation in area. This adjustment in the values of Cy allowed to increase the number of grid points near the leading edge and trailing edge of the airfoilf as we expect to obtain greater variation in flow properties along this section of the flow field. A note must be made regarding the segment AB. This portion of the grid was left uniform as we do not expect to have large variation in flow properties in such region of the grid.

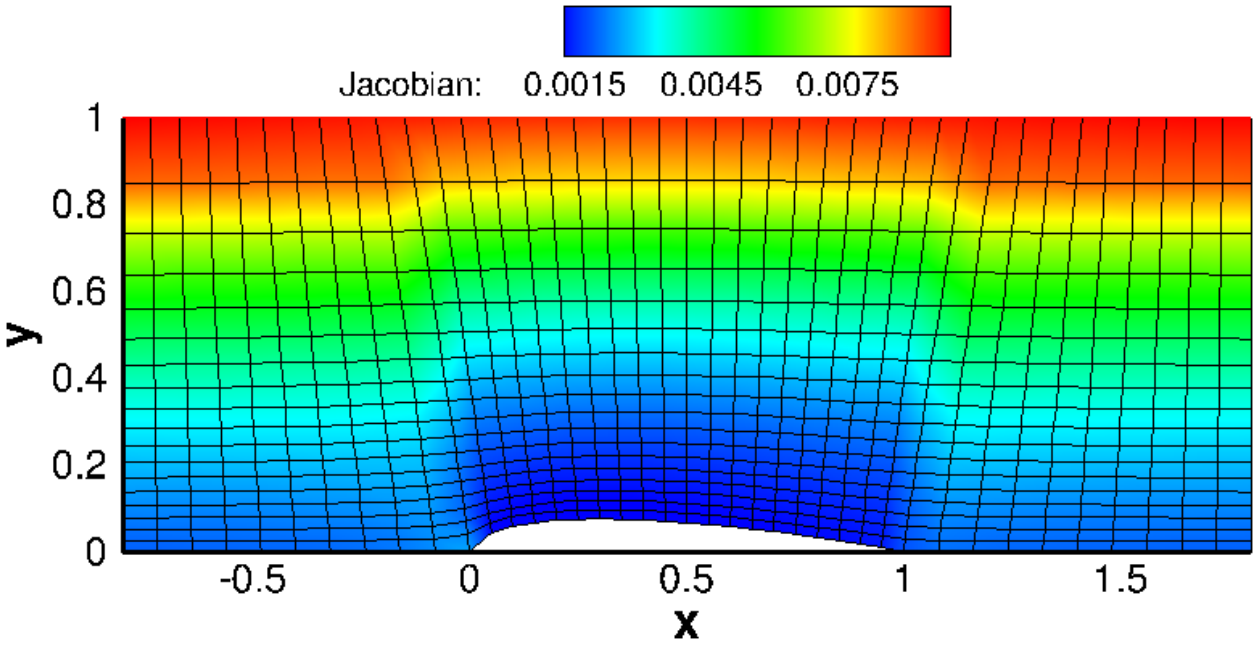


Fig. 1.8: : Inverse Grid Jacobian distribution of Grid #4

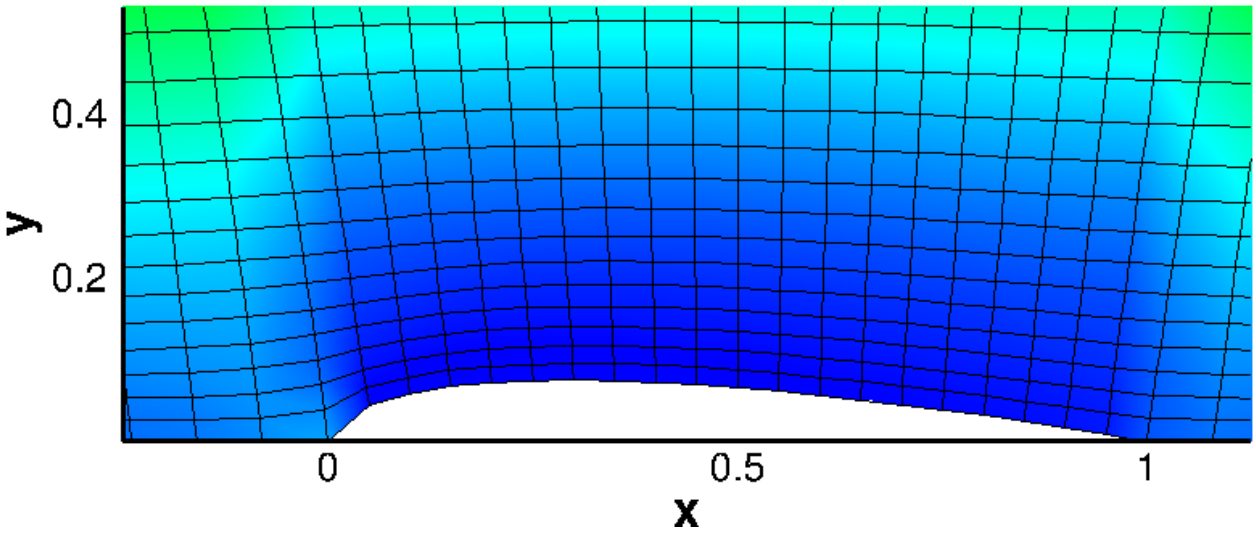
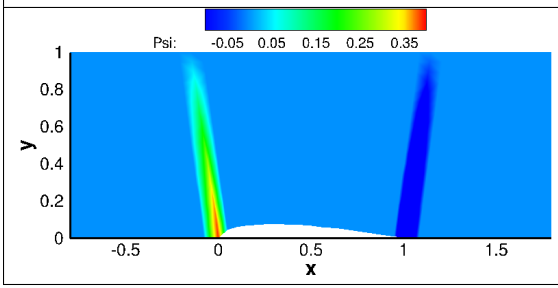


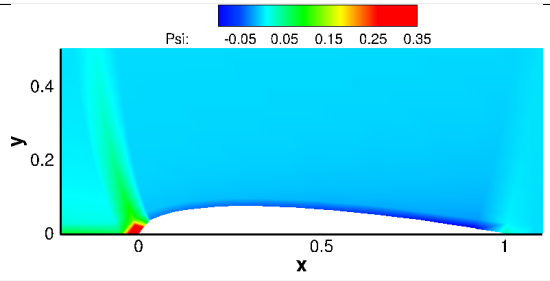
Fig. 1.9: : Zoomed Inverse Grid Jacobian distribution of Grid #4 over airfoil surface

Furthermore, stretching for ϕ and ψ was applied in order to improve the grid alingment along the surface of the airfoil. A negative value for ψ reduces the skewness of nearby cells at the leading and trailing edge. Below the ψ field is compared between the original grid with constant Cy and the new improved grid.

- Initial Grid 'cy = 0.001'



- Grid 5 Psi Field



After completion, the overall improved grid and inverse Jacobian are presented below:

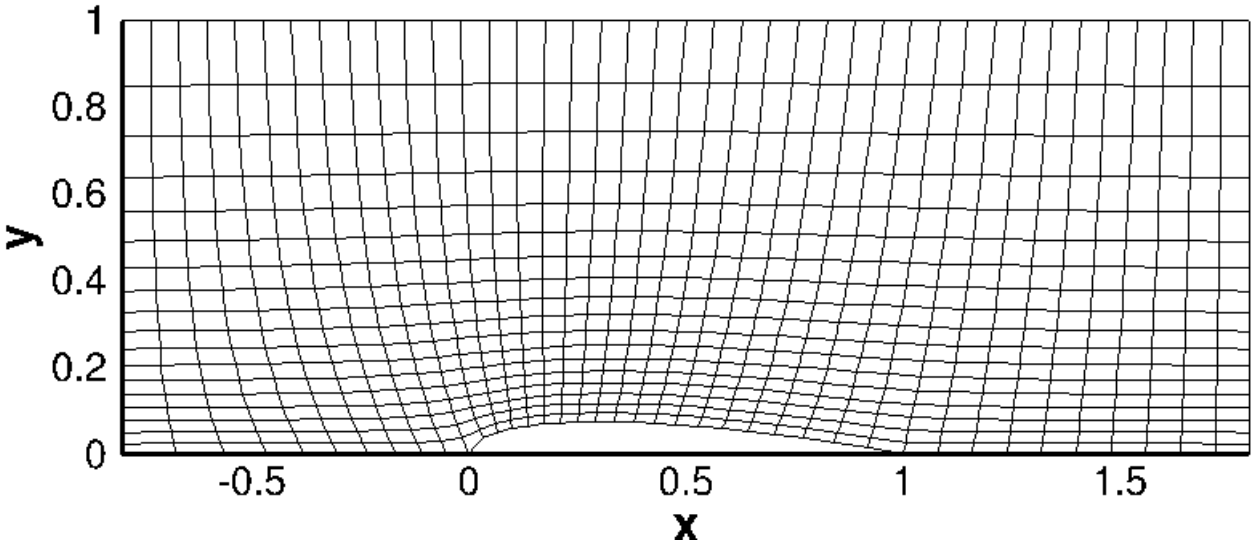


Fig. 1.10: :Grid #5

As shown from the inverse of the Jacobian, most of the smalle area cells are located along the surface of the airfoil where the flow prperties significantly change.

Residual Convergence

The residual is one of the most fundamental measures of an iterative solution's converence, as it directly quantifies the error in the solution. The image below show the RMS residual for the last three grids generated through this code.

It can be noted that as we apply the control terms for the elliptical grid generation, the number of iterations required for the RMS residual to approach the desired limit is reduced.

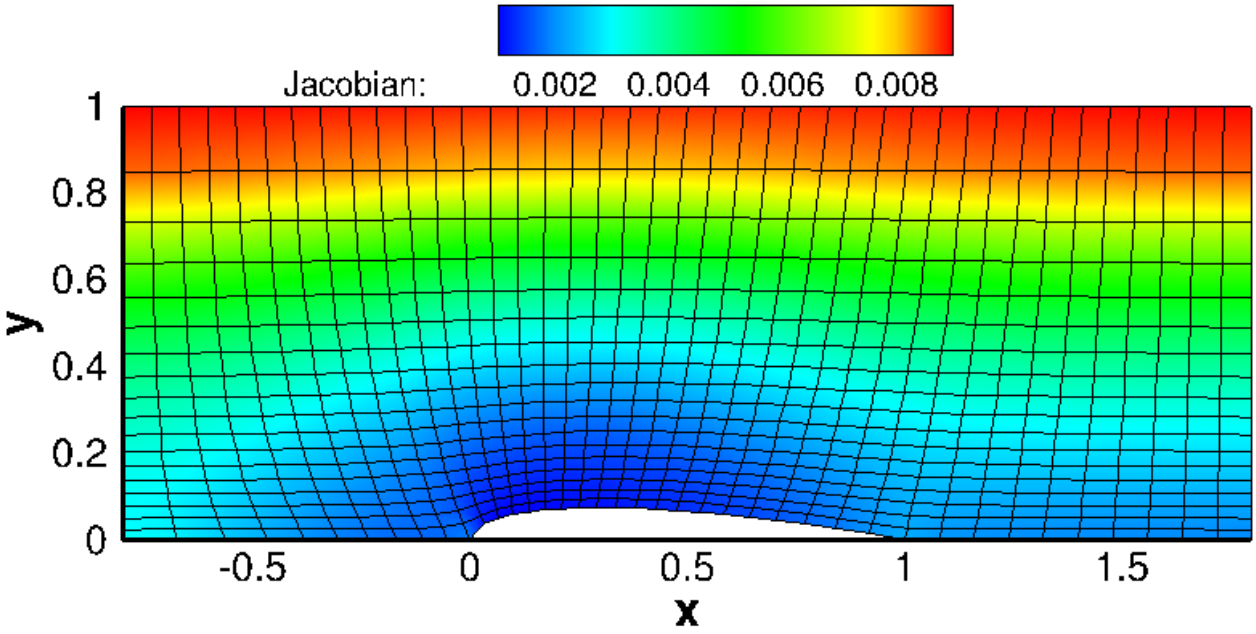


Fig. 1.11: :Inverse Grid Jacobian distribution of Grid #5

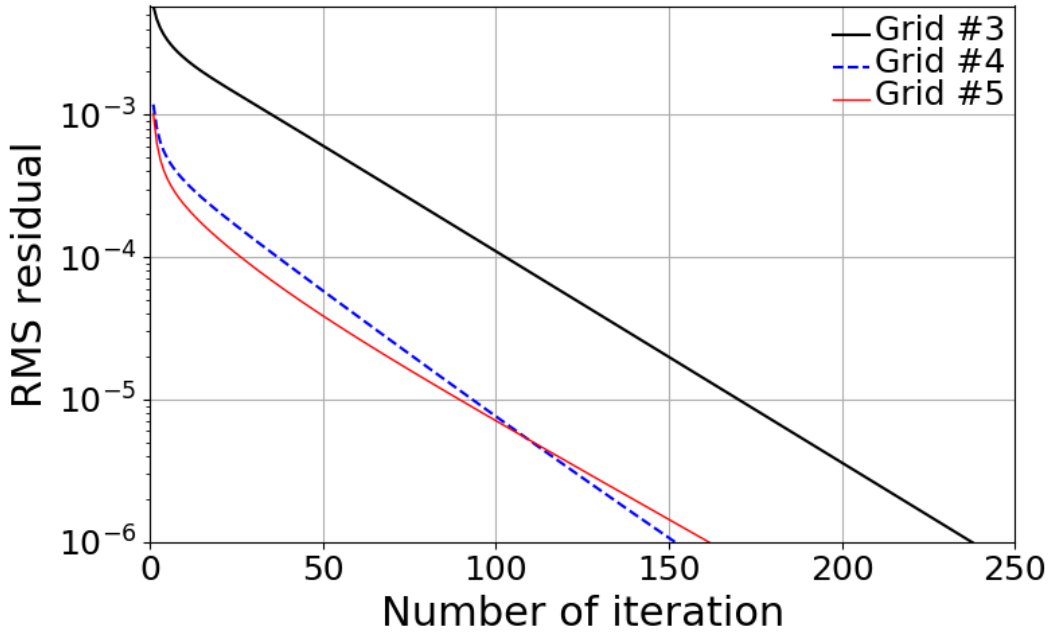


Fig. 1.12: :RMS Residual