2DEuler Documentation

Release 0.0.1

Sayop Kim

January 05, 2015

CHAPTER 1

Contents

1.1 Project description

1.1.1 Given task

In this project, 2-D Euler solver which utilizes the AUSMPW+ scheme (Advection Upstream Splitting Method - Pressure Weighted) has been developed. The developed 2-D Euler code was verified for a test case of a flowfield over the top half of a 10 deg. diamond airfoil.

1.1.2 Governing Equations

The 2-D, unsteady Euler Equations will be solved. The equations will be marched forward in time until a steady state solution is achieved. The transformed 2-D Euler equations can be written:

$$\frac{\partial \left(\bar{U}/J\right)}{\partial t} + \frac{\partial \vec{F'}}{\partial \xi} + \frac{\partial \vec{G'}}{\partial \eta} = 0$$

where the transformed state and inviscid flux vectors are

$$\vec{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E_t \end{bmatrix}$$

and

$$\vec{F'} = \frac{1}{J}\left(\xi_x \vec{F}_I + \xi_y \vec{G}_I\right), \quad \vec{G'} = \frac{1}{J}\left(\eta_x \vec{F}_I + \eta_y \vec{G}_I\right)$$

Here the total energy per unit volume is defined by:

$$E_t = \frac{p}{\gamma - 1} + \frac{\rho}{2}\left(u^2 + v^2\right)$$

and the stagnation enthalpy per unit mass is:

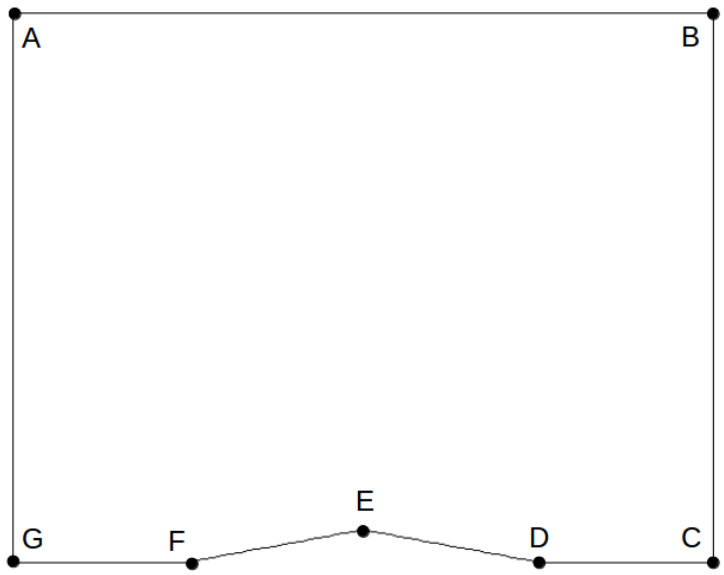
$$h_0 = \frac{p\gamma}{\rho(\gamma - 1)} + \frac{1}{2}\left(u^2 + v^2\right)$$

The transformed velocity component (Contravariant velocities) in the generalized coordinates is determined as a function of u and v with the grid metrics as follows:

$$\tilde{U} = \xi_x u + \xi_y v$$
$$\tilde{V} = \eta_x u + \eta_y v$$

1.1.3 Computational Domain

This project analyze the top half of a 10 deg. diamond airfoil so the location of point E is (x,y) = (0.5, 0.0882). Each grid point can be describd by (x,y) location or (i, j) location where the i index is in the ξ direction and the j index is in the η direction. The grid will consist of 71 points in the "i" direction and 48 points in the "j" direction. The inverse grid metrics must be evaluated at every grid point in the computational domain (including the boundaries). Use 2nd order accurate, central differences for interior points and 2nd order accurate, one-sided differences for boundary points. After the inverse metrics are computed, the grid Jacobian and grid metrics must be computed and stored at every location (including the boundaries)



1.2 Code development

The present project is aimed to develop a computer program for solving 2-D unsteady Euler equations for a supersonic problem. Hereafter, the program developed here in this project is called '2DEuler'.

1.2.1 2DEuler Code summary

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

```
$ cd 2DEuler/CODEdev/src/
$ ls
$ CMakeLists.txt  io  main
```

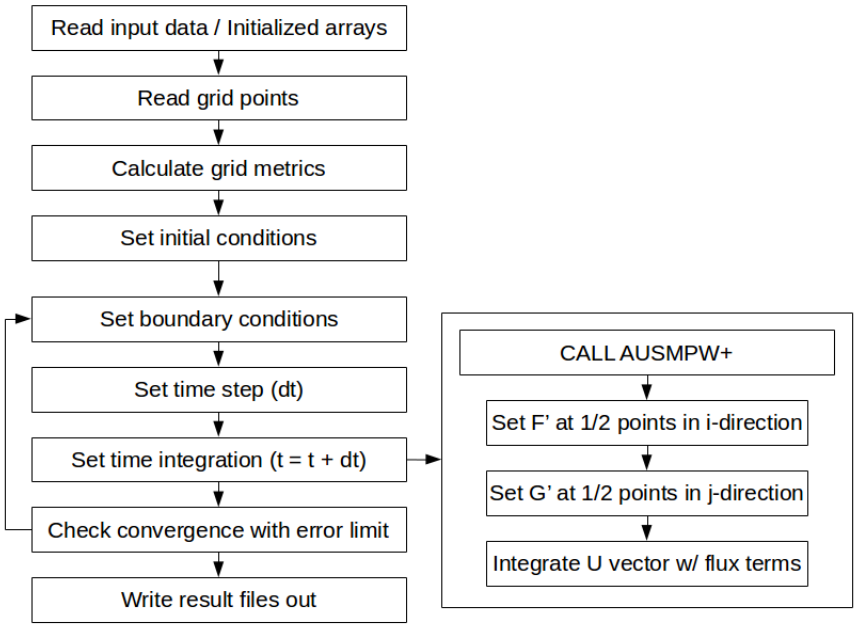
The **io** folder has **io.F90** and **ReadGrid.F90** files which contains subroutines for reading input/output data and grid info. It also includes **input** directory which contains a default **input.dat** file.

The **main** folder is only used for calculating essential subroutines required to solve the '2DEuler' equation by using 'AUSMPW+' scheme for solving flux reconstruction. The main routine is run by **main.F90** which calls important subroutines from **main** folder itself and **io** folder when needed.

1.2.2 Flow chart of 2DEuler code running

The schematic below shows the flow chart of how the 2DEuler code runs. At the beggining of case running, the code reads the important parameters for grid information and numerical schemes. After then, it applies the proper initial conditions and set the variables to be applied to the main time loops.

In the main loop, the time-integration for updating flux vector $\vec{U}$ is made. It begins with applying the pre-specified boundary conditions and calculate the time-step (dt) for time-level update. The main part of this loop is made by calling 'AUSMPW+' flux splitting scheme.



1.3 Numerical Method

During each time-integration step, the code calculate the fluxes $\vec{F'}_{i+1/2,j}$ at every "i" half-point locations, and $\vec{G'}_{i,j+1/2}$ at every "j" half-point locations. In order to obtain the flux terms properly treated with consideration of characteristics of wave propagation, MUSCL differencing should first be used to extrapolate the state vectors to every half point locations. After then AUSMPW+ scheme applies to those points for solving the flux terms.

1.3.1 MUSCL with limiter

The extrapolated state vector can be written:

$$\vec{U}^L_{i+\frac{1}{2}} = \vec{U}_i + \frac{\epsilon}{4}\Delta\vec{U}_{i-\frac{1}{2}}\left\{(1+\kappa)\left[\varphi\left(\mathrm{r}^L\right)\right] + (1+\kappa)\mathrm{r}^L\left[\varphi\left(\frac{1}{\mathrm{r}^L}\right)\right]\right\}$$

where

$$\mathrm{r}^L = \frac{\Delta\vec{U}_{i+\frac{1}{2}}}{\Delta\vec{U}_{i-\frac{1}{2}}} = \frac{\vec{U}_{i+1} - \vec{U}_i}{\vec{U}_i - \vec{U}_{i-1}}$$

$$\vec{U}^R_{i+\frac{1}{2}} = \vec{U}_{i+1} - \frac{\epsilon}{4}\Delta\vec{U}_{i+\frac{3}{2}}\left\{(1+\kappa)\left[\varphi\left(\frac{1}{\mathrm{r}^R}\right)\right] + (1+\kappa)\,\mathrm{r}^L\left[\varphi\left(\mathrm{r}^R\right)\right]\right\}$$

where

$$\mathrm{r}^R = \frac{\Delta\vec{U}_{i+\frac{1}{2}}}{\Delta\vec{U}_{i+\frac{2}{3}}} = \frac{\vec{U}_{i+1} - \vec{U}_i}{\vec{U}_{i+2} - \vec{U}_{i+1}}$$

1.3.2 AUSMPW+ scheme: Flux Splitting with pressure weight

General time-integration in CFD solver should follow the algebraic equation of state- and flux-vectors as shown below:

$$\vec{U}^{n+1}_{i,j} = \vec{U}^n_{i,j} - \Delta t^n_{i,j} J_{i,j} \left[\left(\vec{F'}_{i+\frac{1}{2},j} - \vec{F'}_{i-\frac{1}{2},j} \right) + \left(\vec{G'}_{i,j+\frac{1}{2}} - \vec{G'}_{i,j-\frac{1}{2}} \right) \right]^n$$

where the time-step restriction at every grid cell is

$$\Delta t^n_{i,j} = \left\{ \frac{CFL}{|\tilde{U}| + |\tilde{V}| + c\sqrt{\xi_x^2 + \xi_y^2 + \eta_x^2 + \eta_y^2 + 2|\xi_x\eta_x + \xi_y\eta_y|}} \right\}^n_{i,j}$$

In the above equations, AUSMPW+ scheme applies to calculate the transformed flux vectors at every half points. In this scheme, the values of Mach numbers and pressures are calculated at the cell interfaces and split into the left- and right-extrapolated components. The scheme employed flux vectors can be determined as:

$$\vec{F'} = \frac{\widetilde{\overline{M}}^+_L C_{avg} A_1}{J} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ (E_t + p) \end{bmatrix}_L + \frac{\widetilde{\overline{M}}^-_R C_{avg} A_1}{J} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ (E_t + p) \end{bmatrix}_R + \frac{P^+}{J} \begin{bmatrix} 0 \\ \xi_x p \\ \xi_y p \\ 0 \end{bmatrix}_L + \frac{P^-}{J} \begin{bmatrix} 0 \\ \xi_x p \\ \xi_y p \\ 0 \end{bmatrix}_R$$

where

$$A_1 = \sqrt{\xi_x^2 + \xi_y^2}$$

$$\vec{G'} = \frac{\widetilde{\overline{M}}^+_L C_{avg} A_1}{J} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ (E_t + p) \end{bmatrix}_L + \frac{\widetilde{\overline{M}}^-_R C_{avg} A_1}{J} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ (E_t + p) \end{bmatrix}_R + \frac{P^+}{J} \begin{bmatrix} 0 \\ \eta_x p \\ \eta_y p \\ 0 \end{bmatrix}_L + \frac{P^-}{J} \begin{bmatrix} 0 \\ \eta_x p \\ \eta_y p \\ 0 \end{bmatrix}_R$$

$$A_1 = \sqrt{\eta_x^2 + \eta_y^2}$$

One thing to keep carefully in mind is that grid metrics quantities should not be left- and right- extrapolated because these quantities are transportable flow quantities but static grid-point related quantities.

1.3.3 Initial and Boundary Conditions

At the beginning of simulation, the 2DEuler code sets the initial condition. After then the code set the boundary conditions at every time step. The initial conditions at all grid points is set on the basis of following pre-specified flow quantities:

$$\rho = 4, \quad u = 1.0, \quad v = 1.0, \quad \gamma = 1.4, \quad p = \frac{1}{\gamma}$$

The incoming flow and outflow are supersonic. Thus following pre-specified boundary conditions should be employed by

Inflow: $\vec{U}^n_{1,j}$ (determined from the initial condition parameters)

Outflow: $\vec{U}^n_{imax,j} = \vec{U}^n_{imax-1,j}$ (1st order extrapolation)

Solid boundary: No velocity in the η direction. The 2DEuler code uses a 2nd order extrapolation that is described in the project assignment.

1.3.4 Convergence Limit

In order to terminate the code running when the proper steady-state assumption can be made, the 2DEuler code calculates the RMS error at every time step as defined below:

$$\text{RMS}^n = \sqrt{\frac{1}{N}\sum_{m=1}^{4}\sum_{i=1}^{imax}\sum_{j=1}^{jmax}\left[\left(\vec{U}_{i,j}^{n+1} - \vec{U}_{i,j}^{n}\right)^2\right]}$$

The computation will stop when RMS error normalized by the RMS error at first iteration meets the following criteria:

$$\frac{\text{RMS}^n}{\text{RMS}^{n=1}} \leq 1\text{x}10^{-3}$$

1.4 How to run the code

1.4.1 Machine platform for development

This 2DEuler 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

1.4.2 Code setup

The 2DEuler 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/2DEuler.git
```

If you open the git-cloned folder **CouetteFlow**, you will see two different folders and README file. The **CODEdev** folder contains again **bin** folder, **Python** folder, and **src** folder. In order to run the code, use should run **setup.sh** script in the **bin** folder. **Python** folder contains python scripts that are used to postprocess 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 4.8.1
-- The CXX compiler identification is GNU 4.8.1
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- 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 GNU
-- Check for working Fortran compiler: /usr/bin/gfortran
-- Check for working Fortran compiler: /usr/bin/gfortran  -- works
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Checking whether /usr/bin/gfortran supports Fortran 90
-- Checking whether /usr/bin/gfortran supports Fortran 90 -- yes
-- Configuring done
-- Generating done
-- Build files have been written to: /home/sayop/data/Devel/GitHub.Clones/2DEuler/CODEdev/bin/build
Scanning dependencies of target cfd.x
[ 10%] Building Fortran object CMakeFiles/cfd.x.dir/main/Parameters.F90.o
[ 20%] Building Fortran object CMakeFiles/cfd.x.dir/main/SimulationVars.F90.o
[ 30%] Building Fortran object CMakeFiles/cfd.x.dir/main/GridJacobian.F90.o
[ 40%] Building Fortran object CMakeFiles/cfd.x.dir/main/AUSMPWplus/AUSMPWplus.F90.o
[ 50%] Building Fortran object CMakeFiles/cfd.x.dir/main/TimeIntegration.F90.o
[  60%] Building Fortran object CMakeFiles/cfd.x.dir/io/io.F90.o
[ 70%] Building Fortran object CMakeFiles/cfd.x.dir/main/SimulationSetup.F90.o
[ 80%] Building Fortran object CMakeFiles/cfd.x.dir/main/MainLoop.F90.o
[ 90%] Building Fortran object CMakeFiles/cfd.x.dir/io/ReadGrid.F90.o
[100%] Building Fortran object CMakeFiles/cfd.x.dir/main/main.F90.o
Linking Fortran executable cfd.x
[100%] Built target cfd.x
```

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 input options.

1.4.3 Input file setup

The 2DEuler code allows user to set multiple options to solve the unsteady 2-dimensional Euler problem 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:

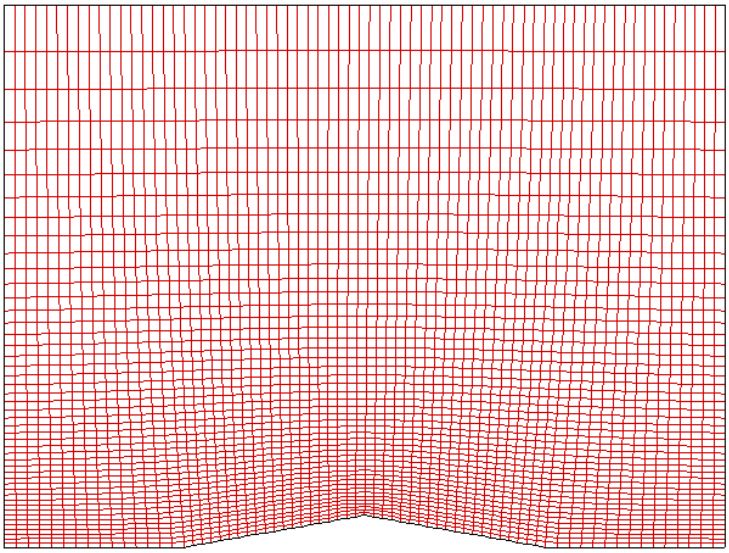
```
#Input file for tecplot print
2-D Euler solver
imax            71
jmax            48
ngl             3
gridFile        grid.fine
#Initial conditions
density         4.0
u               1.0
v               0.0
pressure        0.7142857
gamma           1.4
#Simulation parameters
nmax            20000
CFL             0.8
errorLimit      1e-03
#AUSMPW+ parameters
alpha           0.0
epsil           0
limiter         0
kappa           0.0
```

- **First line** ('2-D Euler solver' by default): Project Name
- **imax**: number of grid points in i-direction
- **jmax**: number of grid points in j-direction
- **ngl**: number of ghost layers (not available in this project)
- **gridFile**: grid file name to be read
- **density**: incoming flow density
- **u**: incoming flow velocity in x-direction
- **v**: incoming flow velocity in y-direction
- **pressure**: incoming flow pressure
- **gamma**: incoming flow heat specific ratio
- **nmax**: maximum number of iteration to be allowed and terminate case running
- **CFL**: CFL number
- **errorLimit**: normalized RMS error limit for convergence
- **alpha**: coefficient in AUSMPW+ (not used in this project)
- **epsil**: switch of second order accurate MUSCL differencing
- **limiter**: switch of MUSCL minmod limiter
- **kappa**: control parameter for 1st/2nd order accurate upwind differencing of MUSCL

1.5 Results and discussions

1.5.1 Computational Grid

The grid used in this project has a resolution of 71 X 48 in i- and j-directions as shown below. In this project, unsteady 2-dimensional Euler solution is being resolved by performing the explicit time-integration with AUSMPW+ flux-splitting scheme.



<Computational grid>

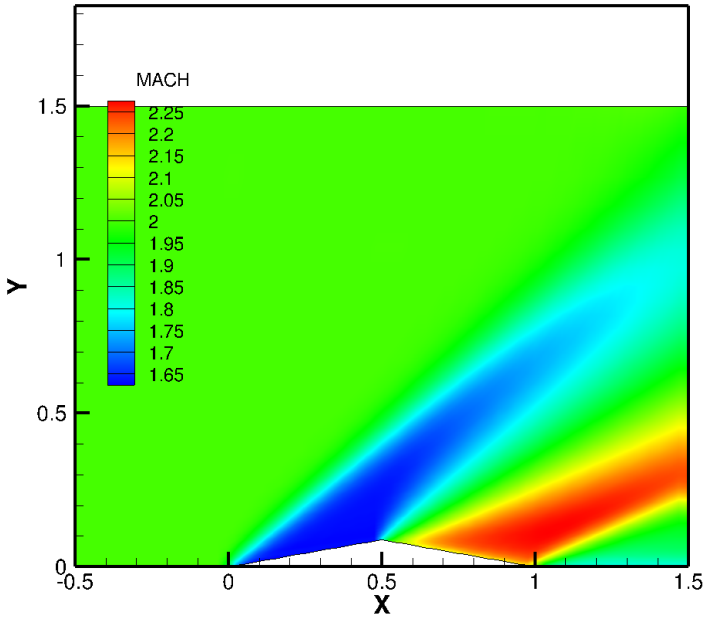
1.5.2 CASE 1

- 1st order accuracy (CFL = 0.8)
- MUSCL differencing setup: $\epsilon = 0$

The first case in this project was set to run with first order accurate with CFL = 0.8. Since this setup employs the first order accuracy, the MUSCL differencing uses neighborting one points from either left or right point. For this setup, ϵ was set to zero in the MUSCL diferencing extrapolation equations in 'Numerical Method' section.

The figure shown below illustrates the fully developed flow field around the airfoil in terms of Mach number. Incoming supersonic flow with Mach = 2 meets the oblique shock so that the flow experiences dramatic change in Mach number. After the flow goes over the top edge of the airfoil, the Mach number goes up again and finally leads to the similar level of Mach number with incoming flow beyond the trailing edge.

As observed near at the top edge, shock surface (Prandtl-Meyer fan) seems to form further downstream than expected theoretically.



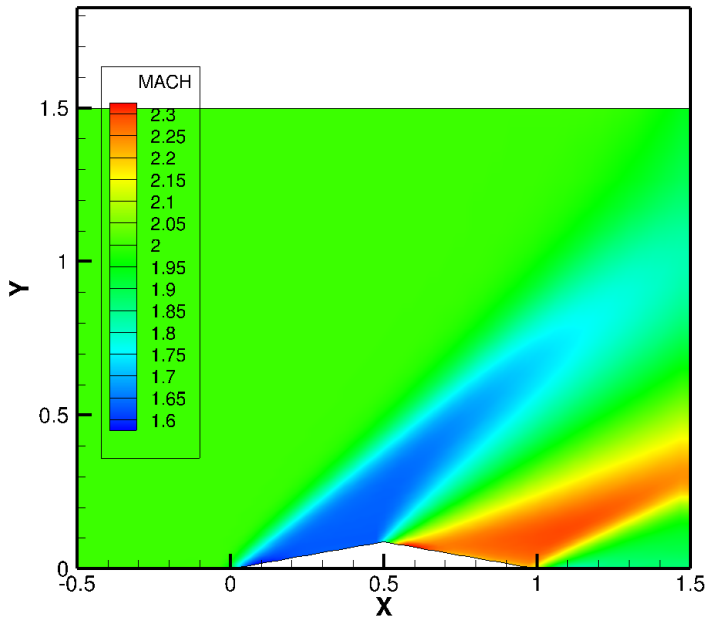
<CASE1: Mach number contour plot>

1.5.3 CASE 2

- 2nd order accuracy MUSCL without flux limiter (CFL = 0.7)
- MUSCL differencing setup: $\epsilon = 0$, $\varphi = 1$, $\kappa = -1$

The second trial for this project is made for running the Euler solver with higher order accurate MUSCL differencing. In this case, by setting φ to 1 and κ to -1, the scheme yields 2nd order accurate, fully upwind differencing.

Since multiple trials with different CFL number gives limitation of CFL number use for this case, the second case was running with CFL = 0.7. Maximum CFL number criterion will be discussed later. Compared to the first case, the Prandtl-Meyer shock formation seems to be attached more close to the top edge. Thus, it can be conlcuded that the higher accurate scheme is more likely to properly predict the shock formation.

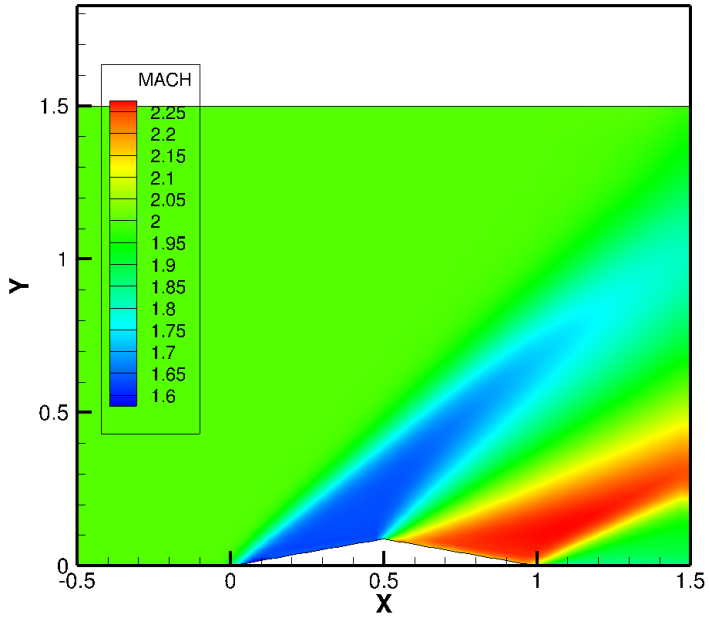


<CASE2: Mach number contour plot>

1.5.4 CASE 3

- 2nd order accuracy MUSCL with minmod limiter (CFL = 0.7)
- MUSCL differencing setup: ϵ = 0, κ = -1, φ: set by minmod limiter

For the final case running, the second order accurate with minmod limiter is employed to introduce the TVD (Total Variation Diminishing) scheme. The figure shown below is the result of Mach number calculated from the CASE #3. This looks qualitatively same as the second result.



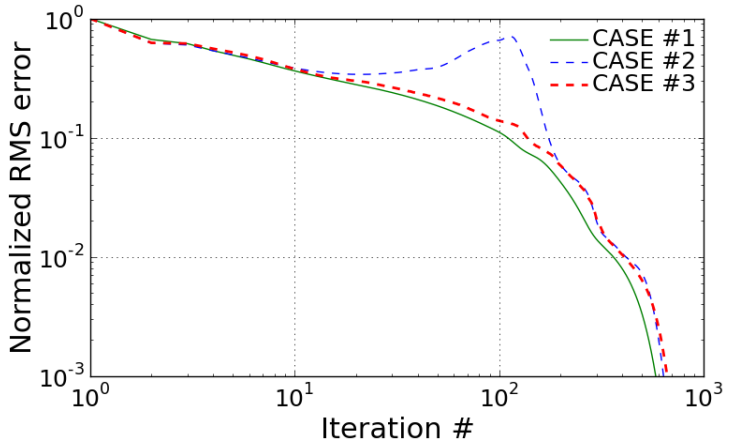
<CASE3: Mach number contour plot>

1.5.5 Comparative Analysis

In this section, three different cases introduced above are compared in terms of convergency and calculated pressure along the bottom wall. The figure shown below illustrates the time history of RMS errors for different cases. The quantitative comparison is made in the table in terms of computational time and required iteration number for convergence. For those cases, the CASE #3 results in the heaviest computational cost. This is because 2nd order accurate needs one more neighboring points to extrapolate every interior points and moreover 'minmod' calculation should be added. The CASE #2 shows the irregular pattern of RMS error around 10^2 iterations level. However, this is confirmed to disappear if the lower CFL number is employed for this case.

The table shown below also tells about applicable maximum CFL number limit for each cases. These numbers were achieved by experimenting the various number of CFL cases.

- Convergence check with RMS limit



<Comparison of RMS history for three different cases>

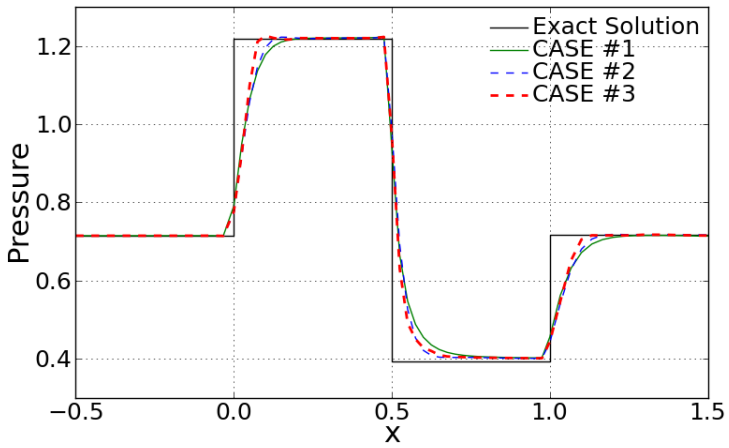
	CPU time (sec)	Iternation # for convergence	Max. CFL to be stable
CASE #1	1.78388	583	1.2
CASE #2	1.93979	637	0.8
CASE #3	2.74088	664	0.75

<Table: required CPU time, total number of iterations, and maximum CFL number>

The quantitative comparison for three different cases with exact invicid solution is made as shown below. The first figure shows the pressure development along the bottom wall. The 2DEuler solver seems to properly follow the theretically resolved solution. However, there is a little difference around the region that shock surface forms.

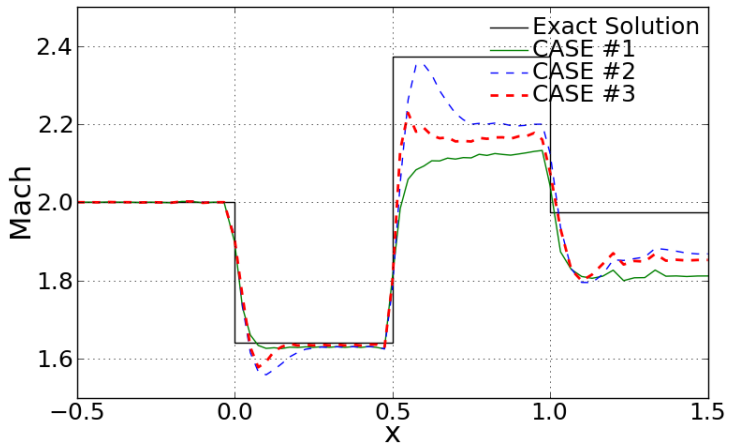
All three cases are somewhat limited to capture the shock surface that brings the infinitesimally small region of sudden pressure change. The first reason of this is because of the limited grid resolution around the shock surface. The second possible reason is that the dissipative errors of the current scheme may lead to smear the high gradient out.

On the other hand, we can find quite meaningful difference between those different cases. The comparison proves that the second order accurate is more likely to follow the high gradient in shock. The more enhanced achievement can be made with the greater resolution of the employed grid.



<Comparison of pressure along the wall>

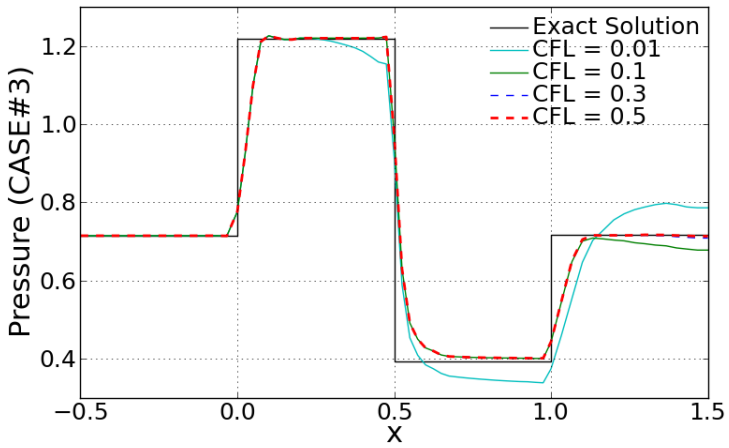
From the comparison of Mach number distribution along the bottom wall, very noticible dispersion errors of second order accurate scheme can be found. When it comes to the TVD scheme, CASE #3 shows the more effective diminishing dispersion compared to the CASE #2 because it adapts the limited extrapolated state vector by using the slope limiter function as defined earlier.



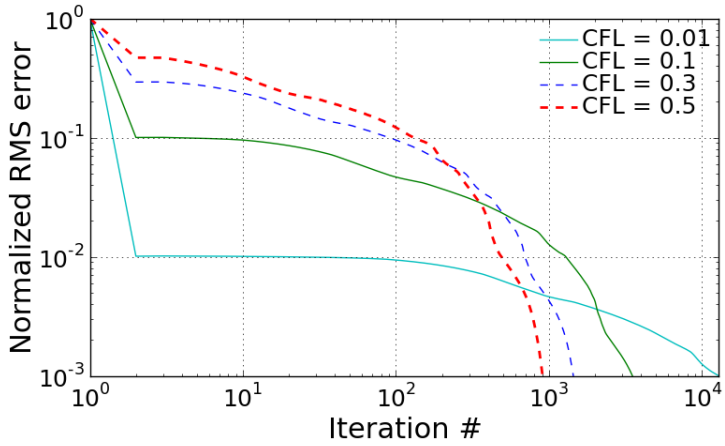
<Comparison of Mach number along the wall>

1.5.6 Effect of CFL (for CASE 3)

Following figures show the effect of employed CFL number in CASE #3. Since the time-step for every iteration is determined on the basis of CFL number, grid size, local contravariant velocities and speed of sound, the effect of CFL number on computational time and convergence history is quite noticible. All these test cases were converged within a same level of RMS limit, which is earlier defined.

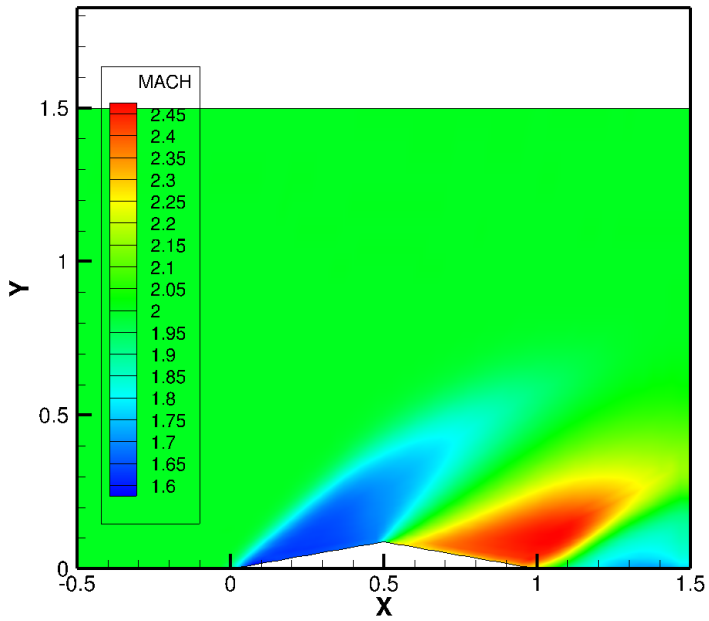


<Effect of CFL number on converged pressure distribution>

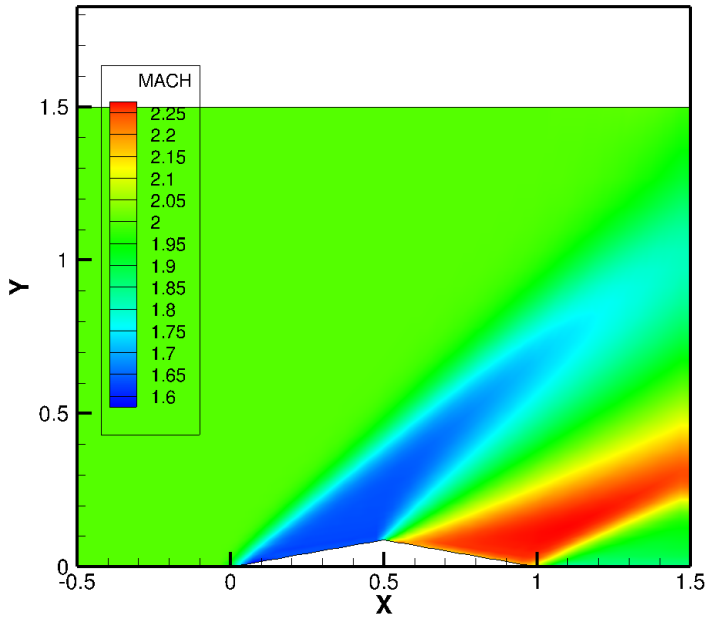


<Effect of CFL number on RMS history>

As noticed from below, the smaller CFL number is, the more far solution is obtained away from the exact solution. The required iteration number gets bigger as CFL number decreases. This is simply because less CFL number reduces the time step and it then results in less change in state vector in every time step. Thus, smaller CFL number case may not be able to show the fully developed steady flow. This is the main reason why the case of CFL = 0.01 shows the far pressure away especially beyond the half of air foil. This can be clearly observed by looking at below two snapshots obtained at the same RMS limit (but at different iteration number).



(A: CFL = 0.01)



(B: CFL = 0.5)

<Mach number contour for different CFL number>