Dual Degree Project

Generated by Doxygen 1.8.11

Contents

Chapter 1

My Personal Index Page

1.1 Introduction

This is the C++ code to solve the high speed fluid flow. Currently, Euler flow is being solved but this code has been designed in moulder way so to solve the viscus flow additional viscus flux class can be added very easily. This code has been written to fulfill the requirement of the Dual Degree Project(DDP).

1.2 Installation & Use

To use the solver. Follow these simple steps.

- Download form here : `https://github.com/singh-kuldeep/DDP2` or `click here`
- Go to the folder DDP2 and compile and run the file TVD.cpp (ex. g++ TVD.cpp && ./a.out)
- Nozzle has been set up as a default geometry but it can be changed from "run.h" file by uncommenting the header file
- Currently there are two different geometry options are available
  1. Curved wall high area ratio diverging nozzle
  2. Triangular bump inside straight duct

1.3 Brief about the solver

- 3D Cartesian (x,y,z)
- Roe scheme based
- C++
- Exact theory can be found `here`

1.4 Input to the solver

- Grid points
- Boundary condition
- Some initial condition

1.5 Output files.

Here are the list of files which will come as the output of the solver.

- Residual_Nozzle.csv : This file contains the all the residuals (Mass, Momentum, Energy).
- grids_Nozzle_2D.csv : This file contains the grid point (x,y) coordinates.
- 2D_parameters_B.csv : This file contains all the conserved parameters at the 2D plane.

1.6 Results & Plots

Same older contains the MATLAB script "plot_data.m". Once the simulation has started and the output files are generated, one can simply run the MATALB script and can see the plots which are listed below.

- Density Residual
- X Momentum Residual
- Y Momentum Residual
- Z Momentum Residual
- Energy Residual
- Mach Number
- Density
- Velocity
- Temperature
- Pressure
- Geometry 2D cross section

Chapter 2

Bug List

Member diffusionfluxinterface::diffusionfluxinterface (vector< double > &ConservedVariableLeftMinus, vector< double > &ConservedVariableLeft, vector< double > &ConservedVariableRight, vector< double > &ConservedVariableRightPlus, vector< double > &FaceAreaVectorLeft, vector< double > &FaceAreaVectorRight, vector< double > &FaceAreaVectorRightPlus, double CellVolumeLeftMins, double CellVolumeLeft, double CellVolumeRight, double CellVolumeRightPlus, double DeltaT)

Here syntax needs to be changed for gvactor[i] calculation

Here I have doubt about "not equal to sign" because it can't be exactly equal to 0.00000 so most of the time we end up choosing theta i = 0.0

File dt.h

Currently not using this, because grid() is not calculating ds value properly. So recheck this function as well after fixing the grid() function.

File eulerflux.h

Not all memory is freed when deleting an object of this class.

Member grid (vector< vector< vector< vector< double > > > > &iFaceAreaVector, vector< vector< vector< vector< double > > > > &jFaceAreaVector, vector< vector< vector< vector< double > > > > &kFaceAreaVector, vector< vector< vector< double > > > &CellVolume, vector< vector< vector< double > > > &delta_s, int &Ni, int &Nj, int &Nk)

Yet to calculate the ds value properly

Member grid (vector< vector< vector< vector< double > > > > &iFaceAreaVector, vector< vector< vector< vector< double > > > > &jFaceAreaVector, vector< vector< vector< vector< double > > > > &kFaceAreaVector, vector< vector< vector< double > > > &CellVolume, vector< vector< vector< double > > > &delta_s, int &Ni, int &Nj, int &Nk)

Yet to calculate the ds value properly

File netfluxinterface.h

Not all memory is freed when deleting an object of this class.

Member run ()

Every time simulation starts from first iteration. So, to save the simulation it is good to start from the last solution as the initial condition

Local time step needs to be used to reduce the simulation time

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 diffusionfluxinterface Class Reference

Public Member Functions

- diffusionfluxinterface (vector$<$ double $>$ &ConservedVariableLeftMinus, vector$<$ double $>$ &Conserved↩ VariableLeft, vector$<$ double $>$ &ConservedVariableRight, vector$<$ double $>$ &ConservedVariableRight↩ Plus, vector$<$ double $>$ &FaceAreaVectorLeft, vector$<$ double $>$ &FaceAreaVectorRight, vector$<$ double $>$ &FaceAreaVectorRightPlus, double CellVolumeLeftMins, double CellVolumeLeft, double CellVolumeRight, double CellVolumeRightPlus, double DeltaT)

Public Attributes

- double **DiffusionFluxVector** [5]

5.1.1 Constructor & Destructor Documentation

5.1.1.1 diffusionfluxinterface::diffusionfluxinterface (vector$<$ double $>$ & *ConservedVariableLeftMinus,* vector$<$ double $>$ & *ConservedVariableLeft,* vector$<$ double $>$ & *ConservedVariableRight,* vector$<$ double $>$ & *ConservedVariableRightPlus,* vector$<$ double $>$ & *FaceAreaVectorLeft,* vector$<$ double $>$ & *FaceAreaVectorRight,* vector$<$ double $>$ & *FaceAreaVectorRightPlus,* double *CellVolumeLeftMins,* double *CellVolumeLeft,* double *CellVolumeRight,* double *CellVolumeRightPlus,* double *DeltaT*) `[inline]`

Parameters

	DiffusionFluxVector	Numerical diffusion flux vector at the interface
`in`	*ConservedVariable*	Conserved variable vector ([Density , x-momentum, y-momentum, z-momentum, Energy])
`in`	*CellVulume*	Pointer to the cell volume vector
`in`	*LeftMinus*	Cell just previous to the left
`in`	*RightPlus*	Cell just Next to the right
`in`	*DeltaT*	Time step

Bug Here syntax needs to be changed for gvactor[i] calculation

Bug Here I have doubt about "not equal to sign" because it can't be exactly equal to 0.00000 so most of the time we end up choosing theta i = 0.0

The documentation for this class was generated from the following file:

- diffusionfluxinterface.h

5.2 eulerflux Class Reference

Public Member Functions

- eulerflux (vector< double > &ConservedVariable)

Public Attributes

- double **EulerFluxX** [5]
- double **EulerFluxY** [5]
- double **EulerFluxZ** [5]

5.2.1 Constructor & Destructor Documentation

5.2.1.1 eulerflux::eulerflux (vector< double > & *ConservedVariable*) `[inline]`

Parameters

	EulerFluxX	x direction euler flux vector (Ee) at interface
	EulerFluxY	y direction euler flux vector (Fe) at interface
	EulerFluxZ	z direction euler flux vector (Ge) at interface
`in`	*ConservedVariable*	Conserved variable vector ([Density , x-momentum, y-momentum, z-momentum, Energy])
	Pressure	Satic pressure (p)

The documentation for this class was generated from the following file:

- eulerflux.h

5.3 interface Class Reference

Public Member Functions

- interface (vector< double > &ConservedVariableLeft, vector< double > &ConservedVariableRight, vector< double > &FaceAreaVectorInterface, double CellVolumeLeft, double CellVolumeRight, double DeltaT)

Public Attributes

- double **DensityInterface**
- double VelocityXInterface
- double VelocityYInterface
- double VelocityZInterface
- double EnthalpyInterface
- double VectorJumpInterface [5]
- double EigenValue [5]
- double EigenVectorMatrix [5][5]
- double EigenVectorMatrixInverse [5][5]
- double AlphaVectorInterface [5]
- double MuVectorInterface [5]
- double ZVectorInterface [5]
- double PshiVectorInterface [5]
- double **GVectorInterface** [5]

5.3.1 Constructor & Destructor Documentation

5.3.1.1 interface::interface (vector$<$ double $>$ & *ConservedVariableLeft*, vector$<$ double $>$ & *ConservedVariableRight*, vector$<$ double $>$ & *FaceAreaVectorInterface*, double *CellVolumeLeft*, double *CellVolumeRight*, double *DeltaT*) [inline]

Parameters

CellVolume	3D vector which has the cell volume of all cells inside the domain

5.3.2 Member Data Documentation

5.3.2.1 double interface::AlphaVectorInterface[5]

Parameters

EigenVectorMatrixInverse	Inverse of the Jacobian matrix

5.3.2.2 double interface::EigenValue[5]

Parameters

VectorJumpInterface	Change in the conserved parameters at the interface

5.3.2.3 double interface::EigenVectorMatrix[5][5]

Parameters

EigenValue	Eigenvalue of the Jacobian matrix

5.3.2.4 double interface::EigenVectorMatrixInverse[5][5]

Parameters

EigenVectorMatrix	Eigenvector of the Jacobian matrix

5.3.2.5 double interface::EnthalpyInterface

Parameters

VelocityZInterface	z velocity at interface

5.3.2.6 double interface::MuVectorInterface[5]

Parameters

AlphaVectorInterface[5]	EigenVectorMatrixInverse[5][5]∗VectorJumpInterface

5.3.2.7 double interface::PshiVectorInterface[5]

Parameters

ZVectorInterface	This is same as MuVectorInterface

5.3.2.8 double interface::VectorJumpInterface[5]

Parameters

EnthalpyInterface	Enthalpy at interface

5.3.2.9 double interface::VelocityXInterface

Parameters

DensityInterface	Roe density at interface

5.3.2.10 double interface::VelocityYInterface

Parameters

VelocityXInterface	x velocity at interface

5.3.2.11 double interface::VelocityZInterface

Parameters

VelocityYInterface	y velocity at interface

5.3.2.12 double interface::ZVectorInterface[5]

Parameters

MuVectorInterface	= delta t $*$ EigenValue

The documentation for this class was generated from the following file:

- interface.h

5.4 netfluxinterface Class Reference

Public Member Functions

- netfluxinterface (vector< double > &ConservedVariableLeftMinus, vector< double > &ConservedVariable↩ Left, vector< double > &ConservedVariableRight, vector< double > &ConservedVariableRightPlus, vector< double > &FaceAreaLeft, vector< double > &FaceAreaVectorRight, vector< double > &FaceArea↩ VectorRightplus, double CellVolumeLeftMins, double CellVolumeLeft, double CellVolumeRight, double Cell↩ VolumeRightPlus, double DeltaT)

Public Attributes

- double **NetFlux** [5]

5.4.1 Constructor & Destructor Documentation

5.4.1.1 netfluxinterface::netfluxinterface (vector< double > & *ConservedVariableLeftMinus,* vector< double > & *ConservedVariableLeft,* vector< double > & *ConservedVariableRight,* vector< double > & *ConservedVariableRightPlus,* vector< double > & *FaceAreaLeft,* vector< double > & *FaceAreaVectorRight,* vector< double > & *FaceAreaVectorRightplus,* double *CellVolumeLeftMins,* double *CellVolumeLeft,* double *CellVolumeRight,* double *CellVolumeRightPlus,* double *DeltaT*) `[inline]`

Parameters

	DiffusionFluxVector	Numerical diffusion flux vector at the interface
`in`	*ConservedVariable*	Conserved variable vector ([Density , x-momentum, y-momentum, z-momentum, Energy])
`in`	*CellVulume*	Pointer to the cell volume vector
`in`	*LeftMinus*	Cell just previous to the left
`in`	*RightPlus*	Cell just Next to the right
`in`	*DeltaT*	Time step

Parameters

left	This object is euler flux calculated using the left cell conserved variables
right	This object is euler flux calculated using the right cell conserved variables
CellVolumeInterface	Average of left and right cell volume

See also

diffusionfluxinterface()
eulerflux()

The documentation for this class was generated from the following file:

- netfluxinterface.h

Chapter 6

File Documentation

6.1 BC.h File Reference

This header file implements all three boundary conditions.

```
#include "math.h"
#include <vector>
```

Include dependency graph for BC.h:

6.2 diffusionfluxinterface.h File Reference

This class calculates the numerical diffusion flux.

```
#include "math.h"
#include "interface.h"
```

Include dependency graph for diffusionfluxinterface.h: This graph shows which files directly or indirectly include this file:

Classes

- class diffusionfluxinterface

6.2.1 Detailed Description

This class calculates the numerical diffusion flux.

Author

Kuldeep Singh

Date

2017

Copyright

GNU Public License.

6.3 dt.h File Reference

This header file conditions the function TimeStep() which calculate the local time step for each cell at every iteration.

```
#include <vector>
#include <math.h>
```
Include dependency graph for dt.h:

Functions

- double **TimeStep** (int i, int j, int k, vector< vector< vector< double > > > delta_s, vector< vector< vector< vector< double > > > > ConservedVariables)

6.3.1 Detailed Description

This header file conditions the function TimeStep() which calculate the local time step for each cell at every iteration.

Author

Kuldeep Singh

Date

2016

See also

grid()

Bug Currently not using this, because grid() is not calculating ds value properly. So recheck this function as well after fixing the grid() function.

Parameters

in	*i,j,k*	Cell location for which TimeStep is to be calculated
in	*delta_s*	ds value of the cell for which TimeStep is to be calculated
	[IN]	ConservedVariables Conserved variables vector
	CFL	Courant–Friedrichs–Lewy number
	Pressure	Static Pressure
	VelocityMagnitude	Magnitude of the velocity
	VelocitySound	Speed of sound
out	*TimeStep*	Time step (dt)

Returns

double

6.4 eulerflux.h File Reference

This class calculates the euler flux vectors(Ee,Fe,Ge) at the interface.

```
#include "math.h"
#include "iostream"
```

Include dependency graph for eulerflux.h: This graph shows which files directly or indirectly include this file:

Classes

- class eulerflux

Macros

- #define SpecificHeatRatio 1.4

6.4.1 Detailed Description

This class calculates the euler flux vectors(Ee,Fe,Ge) at the interface.

Author

Kuldeep Singh

Date

2017

Bug Not all memory is freed when deleting an object of this class.

Copyright

GNU Public License.

6.4.2 Macro Definition Documentation

6.4.2.1 #define SpecificHeatRatio 1.4

This is gas constant (Gamma). For air at room temperature it is almost equal to 1.4. If you are using some other gas at some other temperature then change it

6.5 grid_nozzle.h File Reference

This header file functions find the grid points, cell area vectors and the cell volumes.

```
#include <iostream>
#include "math.h"
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
```

Include dependency graph for grid_nozzle.h: This graph shows which files directly or indirectly include this file:

Functions

- double findY (double x, std::vector< std::vector< double > > UpperCoordinates)

  This function finds the value of the y coordinate of the upper wall at x.
- double finddz (std::vector< std::vector< double > > DownCoordinatesNew)

  Find the cell side in z direction by taking average of all dx for dz.
- double distance (std::vector< double > p1, std::vector< double > p2)

  Calculates the distance between the two points in 3D space.
- double min (double d1, double d2, double d3, double d4, double d5, double d6, double d7, double d8, double d9, double d10, double d11, double d12)

  Find the minimum out of the all input parameters.
- void takeMirror (double &x, double &y, double x1, double y1, double x2, double y2, double l, double m)

  This function will take boundary cell grid points and will calculate the ghost cell grid points by taking the mirror image about the boundary.
- void grid (vector< vector< vector< vector< double > > > > &iFaceAreaVectorIn, vector< vector< vector< vector< double > > > > &jFaceAreaVectorIn, vector< vector< vector< vector< double > > > > &kFace↩ AreaVectorIn, vector< vector< vector< double > > > &CellVolumeIn, vector< vector< vector< double > > > &dsIn, int &Ni, int &Nj, int &Nk)

  This function calculates the cell area and the cell volumes of all cells including the ghost cells.

6.5.1 Detailed Description

This header file functions find the grid points, cell area vectors and the cell volumes.

Author

Kuldeep Singh

Date

2017

Warning

For different geometries change this file accordingly.

6.5.2 Function Documentation

6.5.2.1 double distance (std::vector< double > *p1*, std::vector< double > *p2*)

Calculates the distance between the two points in 3D space.

Parameters

in	*p1*	First point.
in	*p2*	Second point.

Returns

Distance between the two points

6.5.2.2 double finddz (std::vector< std::vector< double > > *DownCoordinatesNew*)

Find the cell side in z direction by taking average of all dx for dz.

Parameters

in	*DownCoordinatesNew*	(x,y) coordinates of the down wall of the nozzle.

Returns

double

6.5.2.3 double findY (double *x,* std::vector< std::vector< double > > *UpperCoordinates*)

This function finds the value of the y coordinate of the upper wall at x.

Parameters

in	*UpperCoordinates*	(x,y) coordinates of the upper wall of the nozzle.
in	*x*	X location.

Returns

double

6.5.2.4 void grid (vector< vector< vector< vector< double > > > > & *iFaceAreaVectorIn,* vector< vector< vector< vector< double > > > > & *jFaceAreaVectorIn,* vector< vector< vector< vector< double > > > > & *kFaceAreaVectorIn,* vector< vector< vector< double > > > & *CellVolumeIn,* vector< vector< vector< double > > > & *dsIn,* int & *Ni,* int & *Nj,* int & *Nk*)

This function calculates the cell area and the cell volumes of all cells including the ghost cells.

This function generates the area vector and cell volumes inside the domain whole domain.

Parameters

in	*iFaceAreaVectorIn*	Input pointer to "i" faces area vector

Parameters

in	*jFaceAreaVectorIn*	Input pointer to "j" faces area vector
in	*kFaceArea←↩ VectorIn*	Input pointer to "k" faces area vector
in	*CellVolumeIn*	Input pointer to cell volumes
in	*dsIn*	Input pointer to minimum distance
	UpperCoordinates	Upper wall coordinates (x,y) of the nozzle geometry
	DownCoordinates	Down wall coordinates (x,y) of the nozzle geometry

Returns

void

Parameters

	N	Total cells in j direction
	N+1	Total grid points in j direction after including the boundary points
in	*Ni*	Input number of cells in in "i" direction.
in	*Nj*	Input number of cells in in "j" direction.
in	*Nk*	Input number of cells in in "k" direction.

Here Nk = 5 because this is 2D-simulation so no need to take large number of cells in z direction

Parameters

Coordinate	4D vector which stores the all coordinates of all cells inside the domain
iFaceAreaVector	4D vector which stores the all "i" face area vectors of all cells inside the domain
jFaceAreaVector	4D vector which stores the all "j" face area vectors of all cells inside the domain
kFaceAreaVector	4D vector which stores the all "k" face area vectors of all cells inside the domain
CellVolume	3D vector which stores the cell volume of all cells inside the domain
(x0,y0)	Live cell coordinates which needs to be mirrored to get the ghost cell coordinates
(x1,y1)	Next live cell coordinates which needs to be mirrored to get the ghost cell coordinates
(l0,m0),(l1,m1)	Line about which reflection needs to be taken
(rx0,ry0)	Ghost cell grid point
(rx1,ry1)	Ghost cell next grid point

Bug Yet to calculate the ds value properly

Structure of grid out put file ("grids_Nozzle_2D.csv")

- First line of the grid file will contain grid points(excluding ghost cells) in x and y direction
- This will exclude the ghost, only live cells or actual geomatry points

6.5.2.5 double min (double *d1*, double *d2*, double *d3*, double *d4*, double *d5*, double *d6*, double *d7*, double *d8*, double *d9*, double *d10*, double *d11*, double *d12*)

Find the minimum out of the all input parameters.

Parameters

`in`	*di*	ith input parameter.

Returns

double

6.5.2.6 void takeMirror (double & *x*, double & *y*, double *x1*, double *y1*, double *x2*, double *y2*, double *l*, double *m*)

This function will take boundary cell grid points and will calculate the ghost cell grid points by taking the mirror image about the boundary.

Parameters

`in`	*&x*	Pointer to x coordinate after taking mirror image
`in`	*&y*	Pointer to y coordinate after taking mirror image
`in`	*l*	x coordinate of the point which is to mirrored
`in`	*m*	y coordinate of the point which is to mirrored
`in`	*(x1,y1)*	Starting point of the line about which mirror is taken
`in`	*(x2,y2)*	End point of the line about which mirror is taken

Returns

void

6.6 interface.h File Reference

This class calculates the interface parameters using Reo scheme flux.

```
#include "math.h"
#include "iostream"
```

Include dependency graph for interface.h: This graph shows which files directly or indirectly include this file:

Classes

- class interface

Macros

- #define SpecificHeatRatio 1.4

6.6.1 Detailed Description

This class calculates the interface parameters using Reo scheme flux.

Author

Kuldeep Singh

Date

2017

Copyright

GNU Public License.

6.6.2 Macro Definition Documentation

6.6.2.1 #define SpecificHeatRatio 1.4

This is gas constant (Gamma). For air at room temperature it is almost equal to 1.4. If you are using some other gas at some other temperature then change it

6.7 netfluxinterface.h File Reference

Calculates the net flux vector(numerical diffusion and euler flux) at the interface.

```
#include "math.h"
#include "eulerflux.h"
#include "diffusionfluxinterface.h"
```

Include dependency graph for netfluxinterface.h: This graph shows which files directly or indirectly include this file:

Classes

- class netfluxinterface

6.7.1 Detailed Description

Calculates the net flux vector(numerical diffusion and euler flux) at the interface.

This class uses the two other class. One Euler for euler fulx calculation and second for numerical diffusion flux calculation.

Author

Kuldeep Singh

Date

2015

Bug Not all memory is freed when deleting an object of this class.

Copyright

GNU Public License(GPL).

6.8 run.h File Reference

This header file contains the run() function which runs the solver.

```
#include "iostream"
#include <vector>
#include <fstream>
#include "math.h"
#include "time.h"
#include "netfluxinterface.h"
#include "grid_nozzle.h"
#include "BC.h"
```

Include dependency graph for run.h:

Functions

- void BC (vector< vector< vector< vector< double > > > > &ConservedVariables, vector< vector< vector< vector< double > > > > &jFaceAreaVector, vector< vector< vector< vector< double > > > > &kFace↩ AreaVector, int Ni, int Nj, int Nk)

  This function implements the boundary condition, iFaceAreaVector is not required Because currently the flow in x direction and 2D flow.
- void grid (vector< vector< vector< vector< double > > > > &iFaceAreaVector, vector< vector< vector< vector< double > > > > &jFaceAreaVector, vector< vector< vector< vector< double > > > > &kFace↩ AreaVector, vector< vector< vector< double > > > &CellVolume, vector< vector< vector< double > > > &delta_s, int &Ni, int &Nj, int &Nk)

  This function generates the area vector and cell volumes inside the domain whole domain.
- void run ()

  This function runs the solver.

6.8.1 Detailed Description

This header file contains the run() function which runs the solver.

Author

Kuldeep Singh

Date

2017

6.8.2 Function Documentation

6.8.2.1 void BC (vector< vector< vector< vector< double > > > > & *ConservedVariables,* vector< vector< vector< vector< double > > > > & *jFaceAreaVector,* vector< vector< vector< vector< double > > > > & *kFaceAreaVector,* int *Ni,* int *Nj,* int *Nk*)

This function implements the boundary condition, iFaceAreaVector is not required Because currently the flow in x direction and 2D flow.

This function implements the boundary condition, iFaceAreaVector is not required Because currently the flow in x direction and 2D flow.

Parameters

in	*ConservedVariables*	This is the pointer to the 4D vector where all the conserved variables of previous time step are stored.
in	*&iFaceAreaVector*	This is a pointer to the 4D vector which has the area vector of all faces which are in "i" direction.
in	*&jFaceAreaVector*	This is a pointer to the 4D vector which has the area vector of all faces which are in "j" direction.
in	*&kFaceAreaVector*	This is a pointer to the 4D vector which has the area vector of all faces which are in "k" direction.
in	*Ni*	Number of cells in in "i" direction.
in	*Nj*	Number of cells in in "j" direction.
in	*Nk*	Number of cells in in "k" direction.

Returns

void

Inlet conditions are user given data. one has to mention the stagnation parameters at inlet (ex. stagnation pressure (P_0), temperature(T_0))

Parameters

TemperatureStagnation	Stagnation temperature at inlet
PressureStagnation	Stagnation pressure at inlet
DensityStagnation	Stagnation density at inlet
Geometry	rotation angle

Inlet ghost cells are being updated using the stagnation quantities(P_0, T_0) and flow direction

Parameters

InletPressure	Static pressure at inlet
Mach	Mach number at inlet
InletTemperature	Static temperature at inlet
InletVelocity	Flow velocity at inlet
InletDensity	Flow density at inlet

At exit updating the i ghost cells (this is true where flow is supersonic)

Updating the ghost cell conserved parameters value at j - wall

6.8.2.2 void grid (vector< vector< vector< vector< double > > > > & *iFaceAreaVectorIn*, vector< vector< vector< vector< double > > > > & *jFaceAreaVectorIn*, vector< vector< vector< vector< double > > > > & *kFaceAreaVectorIn*, vector< vector< vector< double > > > & *CellVolumeIn*, vector< vector< vector< double > > > & *dsIn*, int & *Ni*, int & *Nj*, int & *Nk*)

This function generates the area vector and cell volumes inside the domain whole domain.

This function generates the area vector and cell volumes inside the domain whole domain.

Parameters

in	*iFaceAreaVectorIn*	Input pointer to "i" faces area vector
in	*jFaceAreaVectorIn*	Input pointer to "j" faces area vector
in	*kFaceArea←↩ VectorIn*	Input pointer to "k" faces area vector
in	*CellVolumeIn*	Input pointer to cell volumes
in	*dsIn*	Input pointer to minimum distance
	UpperCoordinates	Upper wall coordinates (x,y) of the nozzle geometry
	DownCoordinates	Down wall coordinates (x,y) of the nozzle geometry

Returns

void

Parameters

	N	Total cells in j direction
	N+1	Total grid points in j direction after including the boundary points
in	*Ni*	Input number of cells in in "i" direction.
in	*Nj*	Input number of cells in in "j" direction.
in	*Nk*	Input number of cells in in "k" direction.

Here Nk = 5 because this is 2D-simulation so no need to take large number of cells in z direction

Parameters

Coordinate	4D vector which stores the all coordinates of all cells inside the domain
iFaceAreaVector	4D vector which stores the all "i" face area vectors of all cells inside the domain
jFaceAreaVector	4D vector which stores the all "j" face area vectors of all cells inside the domain
kFaceAreaVector	4D vector which stores the all "k" face area vectors of all cells inside the domain
CellVolume	3D vector which stores the cell volume of all cells inside the domain
(x0,y0)	Live cell coordinates which needs to be mirrored to get the ghost cell coordinates
(x1,y1)	Next live cell coordinates which needs to be mirrored to get the ghost cell coordinates
(l0,m0),(l1,m1)	Line about which reflection needs to be taken
(rx0,ry0)	Ghost cell grid point
(rx1,ry1)	Ghost cell next grid point

Bug Yet to calculate the ds value properly

Structure of grid out put file ("grids_Nozzle_2D.csv")

- First line of the grid file will contain grid points(excluding ghost cells) in x and y direction
- This will exclude the ghost, only live cells or actual geomatry points

6.8.2.3 void run ( )

This function runs the solver.

Warning

Currently not using this, because grid() is not calculating ds value properly. So recheck this function as well after fixing the grid() function.

Returns

double

Parameters

StartTime	Simulation starting time
EndTime	Simulation ending time
DeltaT	Time step
IterationValues	Total iterations = floor(TIME/DeltaT)
Ni	Number of cells in in "i" direction.
Nj	Number of cells in in "j" direction.
Nk	Number of cells in in "k" direction.
&iFaceAreaVector	This is a pointer to the 4D vector which has the area vector of all faces which are in "i" direction.
&jFaceAreaVector	This is a pointer to the 4D vector which has the area vector of all faces which are in "j" direction.
&kFaceAreaVector	This is a pointer to the 4D vector which has the area vector of all faces which are in "k" direction.
CellVolumeIn	Input pointer to cell volumes
delta_s	Minimum distance
ConservedVariables	This is the pointer to the 4D vector where all the conserved variables ([Density , x-momentum, y-momentum, z-momentum, Energy]) of previous time step are stored.
ConservedVariablesNew	This is the pointer to the 4D vector where all the conserved variables ([Density , x-momentum, y-momentum, z-momentum, Energy]) of current/new time step are stored.

Bug Every time simulation starts from first iteration. So, to save the simulation it is good to start from the last solution as the initial condition

Parameters

iCellInterfaceVolume	Average of right and left cell volume in i direction
jCellInterfaceVolume	Average of right and left cell volume in j direction
kCellInterfaceVolume	Average of right and left cell volume in k direction

Bug Local time step needs to be used to reduce the simulation time

Parameters

DensityResidual	Density residual
xMomentumResidual	x Momentum residual
yMomentumResidual	y Momentum residual
zMomentumResidual	z Momentum residual
Energy	residual

Index