Appendix
uFVM

A.1 Introduction

uFVM is a general three dimensional unstructured finite volume-based code developed in Matlab® for the solution of general single fluid flow and transport phenomena problems. The code is written for academic purpose and emphasizes programming simplicity and readability over performance. The code is capable of dealing with a wide range of flow problems and is easily extensible.

A.2 The Base Structure

The code is composed of task specific functions that mimic the numerical functions used in the finite volume method. The user can setup a case by writing a script to read the geometry and define the mathematical model with its associated initial and boundary conditions. An illustrative script file is presented in Listing A.1 showing a test case that involves the solution of a simple scalar equation.
% Convection-Diffusion Problem Solved on Static Grid

1. clear all;
2. clc;
3. global Domain
4. cfdSetupDomain;

%Reading the Geometry from OpenFOAM
5. cfdReadOpenFoamMesh('Domain25CV');

% setup Fluid
6. cfdSetupFluid('water','MW',18);
7. cfdSetIsTransient

% Creating the Property Fields
8. cfdSetupProperty('Density:water','constant','1000');
9. cfdSetupProperty('SpecificHeat:water','constant','4.186');
10. cfdSetupProperty('conduction:water','constant','4.186');
11. cfdSetupProperty('Velx:water','constant','10');
12. cfdSetupProperty('Vely:water','constant','10');
13. cfdSetupProperty('Velz:water','constant','0');

% Setting the equation:
% ========================
14. cfdSetupEquation('T:water','ic','0','surf',1);
% Adding the terms constituting the equation with appropriate coefficients
15. cfdAddTerm('T:water','Transient','coefficientName','Density:water',...
              'coefficientName','SpecificHeat:water');
16. cfdAddTerm('T:water','Diffusion','coefficientName','conduction:water');
17. cfdAddTerm('T:water','Convection','coefficientName','Density:water',...
              'scheme','UPWIND');
% Specifying the Boundary Conditions for the equation
18. cfdSetBC('T:water',1,'type','Specified Value','value','10'); % Inlet BC
19. cfdSetBC('T:water',2,'type','Outlet'); % Outlet BC
20. cfdSetBC('T:water',3,'type','Specified Value','value','0'); % Side Walls
21. cfdSetBC('T:water',4,'type','empty'); % Front & Back

% Creating an Mdot Field
22. cfdSetupMdotFields;

% Initializing the special array for each field
23. cfdInitializeFields;

24. time_i=0;
25. time_f=7;
26. dt=1;

Listing A.1 A script for solving a convection-diffusion problem using uFVM
A.3 Reading the Mesh

The uFVM code can read an OpenFOAM® mesh directly from an OpenFOAM® case directory. Specifically from the polyMesh subfolder that contains the mesh geometric and topological information. The standard structure of an OpenFOAM® case folder is presented in Fig. A.1.

The ‘0’ directory is the initialization sub-folder containing the initial and boundary conditions of the fields defined in the problem model. The ‘constant’ directory contains the dynamicMeshDict, which is a dictionary for dynamic meshes, and a ‘polyMesh’ directory special for the description of the problem’s geometry. The ‘system’ directory contains dictionaries that define the case setup. The controlDict is concerned with the general control parameters of the test case, the fvSchemes defines the discretization schemes, while the fvSolution contains information about the solution algorithms and relaxations to be used in the simulation.

uFVM reads and processes the polyMesh folder information in the \texttt{cfdReadOpenFoamMesh} function. The function starts by reading the points file and storing the \(x, y, z\) coordinates into a structure array called the nodes array, then the geometric information is read from the faces file in the form of a list of node indices for each face. This information is processed and stored in a structure array.
called faces. Information about the patches and the associated patch faces is then read from the boundary file and stored in a structure array denoted by boundary. Finally, the owners and neighbors files are read and the elements structure array is composed. Then, the topological information for elements, faces, nodes and their connectivities are processed in function `cfdProcessOpenFoamMesh`. This is in addition to secondary geometric information such as the volume and centroids of
elements, centroids and area of faces, as well as interpolation factors, and other geometrical entities.

At this stage, the mesh contains: the nodes array storing for each node its coordinates, index, and the element/s and faces to which it belongs; the faces array storing for each face its index, the coordinates of its centroid, area, interpolation factor, area vector, the owner and the neighbor, patch index and the nodes that construct it; the elements array storing for each element the index, the indices of the

Fig. A.2 The internal data structure in uFVM
neighbors, the indices of the faces and the nodes constructing the element, volume, face sign and the coordinates of its centroid. Moreover, a boundary structure array is created that contains the boundary types included in the problem and read from the OpenFoam® boundary file. The array contains an index of the boundary patches and their types, their number of faces and the starting face of each patch. An illustration of the above described structure is displayed in Fig. A.2.

A.4 Setting-Up the Model

Before setting up the equations underlying the physical model and representing the major constituents of any case, the fluid/s involved in the problem should be described. This is done through the function cfdSetupFluid, which is responsible for defining the type of the fluid, its username, molecular weight, type (compressible or incompressible), in addition to some other related information. It should be noted at this stage that all the data that will be used by other functions are saved in a global structure array named Domain. The thermo-physical variables appearing in the governing equations should be defined as well. This is the role of the special function cfdSetupProperty. In this function, the user should insert, among others, the name of the property, its type, and the under relaxation factor needed when updating its value. All data is stored in an element mesh field.

Now the setup of the governing equations can proceed for any problem by using the function cfdSetupEquation. This is where the type of the equation (scalar or vector) is defined along with the initial conditions and the under-relaxation factor. The user can also choose the gradient type to be used for evaluating the derivatives appearing in the conservation equation. A field for each equation is stored on an element mesh size array in the global structure array Domain. For each defined equation, the user can add the various terms (transient, convection, diffusion, pressure gradient, stress, source, electric potential, Darcy, buoyancy, and drag among others) that constitute the equation. Each term is defined in the code using the function cfdAddTerm. This function relates each term along with the coefficients (density, viscosity, and diffusion coefficient among others) to its equation where the information is stored for later use during assembly. The associated boundary conditions of each equation are then added using the function cfdAddBC, where the type of the boundary condition (inlet, outlet, specified value, specified flux, slip, no slip among others) and its value, if needed, are specified. If any of the equations contains a convective term, then the user has to setup a ‘mdot’ (mass flow rate: density multiplied by the dot product of the velocity vector and the area vector) field, which creates a mesh field covering the faces of the domain.
The internal structure of an equation (e.g., ‘Velocity:water’) is shown in Listing A.2.

Listing A.2  Internals of an equation (model) structure

The structure of a term is shown in Listing A.3.

Listing A.3  Internals of the term structure
A.5 Setup the Computational Fields

This section describes how fields are initialized in the code, each on its prescribed locale (elements, faces, nodes). Adding the function `cfdInitializeFields` into the script file will automatically initialize the equation field, the property field, and the ‘mdot’ field. An equation is initialized over elements by computing and distributing the initial conditions onto each interior element along with the previously defined boundary conditions. After that, a property is initialized over the associated mesh field, whether it should be computed from a formula or has a constant value, by evaluating or placing the values over elements or faces. The ‘mdot’ field is initialized as well by calling the density field (which is a property already defined and initialized) along with the velocity field to compute the value over each face of the mesh. Storing all the initialized fields in the appropriate arrays for later communication is done by each associated function (`cfdInitializeEquation`, `cfdInitializeProperty`, `cfdInitializeMdotField`).

A.6 Equation Discretization (Assembly)

After all fields have been initialized and the environment has become suitable to start the solution process, the function `cfdAssembleAndCorrectEquation` is invoked in order to assemble, solve, and correct the equations governing the modeled problem.

A.6.1 Equation Assembly

For each equation, the internal function `cfdAssembleEquation` assembles the terms that have already been associated with the specified equation. A coefficient array containing the coefficients of the variable, of size equal to the number of control volumes, is setup using the `cfdSetupCoefficients` function, as illustrated in Fig. A.3.

A process is initiated in `cfdAssembleEquationTerms` to loop over each term and calculate its fluxes according to the selected or default scheme. The calculated face fluxes and element fluxes are then assembled according to their discretized form into the global coefficient matrix by `cfdAssembleIntoGlobalMatrixFaceFluxes` and `cfdAssembleIntoGlobalMatrixElementFluxes`, respectively. After assembling all the terms of a single equation and obtaining the complete array of coefficients, the residuals of the equation are computed using the function `cfdComputeResidual` according to the prescribed equation of the residuals. After that the under-relaxation factor, already specified for each equation, is applied on the coefficients.
A.6.2 Solving the Equations

Two algebraic solvers are implemented in uFVM, which are the successive over-relaxation method (SOR) and the ILU(0) method. The SOR method is simply the under-relaxed Gauss-Siedel method. With SOR the array containing the coefficients is imported to the solver, which in turn loops over the elements of the domain to solve and update the values of the unknown field $\phi$. The system of equations is solved in residual form according to the following equation:

$$\phi'_C = \frac{b_C - a_C \phi_C - \sum_{F \sim NB(C)} a_F \phi_F}{a_C}$$  \hspace{1cm} (A.1)$$

The ILU(0) method follows the methodology described in Chap. 10, where also the system of equations is solved in residual form.

Whether solving a simple scalar equation or a set of scalar and vector equations (i.e., the Navier-Stokes equations), the solutions of these equations have to be
corrected using the \texttt{cfdCorrectEquation} function along with several internal functions specific for the correction of each type of equations: velocity, scalar or pressure (which has a specific treatment). After the correction field is obtained, the \(\phi\) field at internal nodes is updated. Boundary conditions are corrected, each according to its specified type through the use of any of the following functions: \texttt{cfdCorrectWallZeroFluxBC}, \texttt{cfdCorrectWallSpecifiedFluxBC}, \texttt{cfdCorrectInletInletBC}, and \texttt{cfdCorrectOutletZeroFluxBC} among others. The pressure equation is corrected by executing \texttt{cfdCorrectPPField} and \texttt{cfdCorrectPressureEquation} because of the need to correct the pressure correction and pressure fields. After that, the velocity and the ‘\texttt{mdot}’ fields are corrected at the interior element faces as well as at boundaries.

\subsection*{A.6.3 Computing the Residuals}

The residual of each equation is calculated using the \texttt{cfdComputeResidual}. First a scaled \(\phi\) for the equation to be solved is calculated as

\[
\phi_{\text{scale}} = \max(\phi_{\text{max}} - \phi_{\text{min}}, \abs{\phi_{\text{max}}})
\]  

(A.2)

then the residuals are scaled using

\[
R_{C,\text{scaled}}^{\phi} = \sum_{\text{all elements}} \frac{b_C - a_C \phi_C - \sum_{F \sim \text{NB}(C)} a_F \phi_F}{a_C \phi_{\text{scale}}}
\]  

(A.3)

and finally the root-mean square residual is computed over the domain as

\[
R_{C,\text{rms}}^{\phi} = \sqrt{\frac{\sum_{C \sim \text{all cells}} (R_{C,\text{scaled}}^{\phi})^2}{\text{number of elements}}}.
\]  

(A.4)

The obtained value is stored since it is needed as an indicator for the convergence of the solution and can be plotted as explained later.

\subsection*{A.7 Plotting Utilities}

Several plotting functions are available in uFVM allowing results in addition to mesh and geometry to be visualized. A summary of these functions is given below.  

\texttt{cfdPlotMesh} plots the domain under consideration along with the mesh that covers it.
**cfdPlotElements** plots any element the user choose or a set of specified elements using the index of each.

**cfdPlotFaces** plots the faces of the domain using their indices.

**cfdPlotPatches** plots the full boundary patch that the user choose by using the index of the patch already defined in earlier stages of a case problem.

**cfdPlotContours** plots contours of the variable over the domain.

**cfdPlotField** plots any field defined in the solver, as shown in Fig. A.4.

---

Fig. A.4  The $\phi$ field over the domain of interest

---

**cfdPlotResiduals** plots the residual value of each variable per iteration, as shown in Fig. A.5.

---

Fig. A.5  Variation of the residual of $\phi$ with iterations for a transient run
**cfdPlotVelocity** is responsible for plotting the mesh and the velocity vectors on the element centroids and on boundary faces, as shown in Fig. A.6.

**A.8 Interpolation Schemes**

Several interpolation functions are included in uFVM serving different purposes. A summary of these interpolation functions is given below.

1. **cfdInterpolateFromElementsToNodes** function is used to compute the gradient according to the node-based method and in the **cfdPlotField** function. A loop over all nodes is performed and for each node an array is created to store the indices of the elements sharing the node. The value of the variable at the node is computed by applying Eq. (9.18).

2. **cfdInterpolateFromElementsToFaces** function is used in assembling the stress, diffusion, and ‘mdot’ terms as well as in the initialization of fields. Three interpolation schemes (Hyperbolic, Upwind, and Average) are implemented in this function from which the user can choose when computing face values from element values.

3. **cfdInterpolateFromNodesToFaces** function is used in computing the gradient according to the node-based method. For a single face, the value at the face centroid is computed from the values at the face nodes using Eq. (9.18).

4. **cfdInterpolateGradientsFromElementsToInteriorFaces** function is used to interpolate the gradients from elements to interior faces according to the selected interpolation scheme. Four options are available for this function (Average, Upwind, Downwind and Corrected Average). The Average scheme depends on the weighting geometric factor and includes the owner and neighbor of the

Fig. A.6 A plot of the velocity field over the domain

![Image of velocity field over domain]
interior face. The Upwind and Downwind schemes use the value of the upwind and downwind element, respectively, depending on the direction of the \( \text{mdot} \) vector at the specified interior face. The corrected Average scheme resembles the Average scheme but with the introduction of a correction to the interpolated gradient according to Eq. (9.35).

### A.9 Test Cases

uFVM comes with a set of basic test cases that can be used to learn how to setup and run problems using the code. They are also useful as initial templates to setup new problems. The name of files for all test cases start with ‘test’, some of these test cases are listed below.

- `testAdvection.m`
- `testDiffusion.m`
- `testDiffusion01.m`
- `testDiffusion02.m`
- `testDiffusion03.m`
- `testDiffusion04.m`
- `testFlow01.m`
- `testFlow02.m`
- `testGradients.m`
- `testShearRate.m`
- `testSmithHutton.m`
- `testStepProfile.m`
- `testStepProfile2.m`
- `testTemperature.m`
- `testTransient.m`
- `testTurbulence.m`

Any test case can be run simply by calling it from within Matlab\textsuperscript{®} as a script with no input. More information on uFVM can be obtained throughout the book in the various Computational Pointers sections.

### A.10 Closing Remarks

uFVM is mainly used as a teaching tool. It is easy to read but it may take some time to get used to, especially the setting up of cases. However it provides a clear implementation of many of the numerics currently used in industrial type CFD codes. The uFVM code was found to provide good and practical insight to students. It is our hope that you will find it as useful in your classroom as we have in ours.