Introduction to CFD Basics

Rajesh Bhaskaran
Lance Collins

This is a quick-and-dirty introduction to the basic concepts underlying CFD. The concepts are illustrated by applying them to simple 1D model problems. We'll invoke these concepts while performing "case studies" in FLUENT. Happily for us, these model-problem concepts extend to the more general situations in the case studies in most instances. Since we'll keep returning to these concepts while performing the FLUENT case studies, it's worth your time to understand and digest these concepts.

We discuss the following topics briefly. These topics are the minimum necessary to perform and validate the FLUENT calculations to come later.

1. The Need for CFD
2. Applications of CFD
3. The Strategy of CFD
4. Discretization Using the Finite-Difference Method
5. Discretization Using The Finite-Volume Method
6. Assembly of Discrete System and Application of Boundary Conditions
7. Solution of Discrete System
8. Grid Convergence
9. Dealing with Nonlinearity
10. Direct and Iterative Solvers
11. Iterative Convergence
12. Numerical Stability
13. Turbulence modeling

The Need for CFD

Applying the fundamental laws of mechanics to a fluid gives the governing equations for a fluid. The conservation of mass equation is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0$$

and the conservation of momentum equation is

$$\rho \frac{\partial \vec{V}}{\partial t} + \rho(\vec{V} \cdot \nabla)\vec{V} = -\nabla p + \rho \vec{g} + \nabla \cdot \tau_{ij}$$

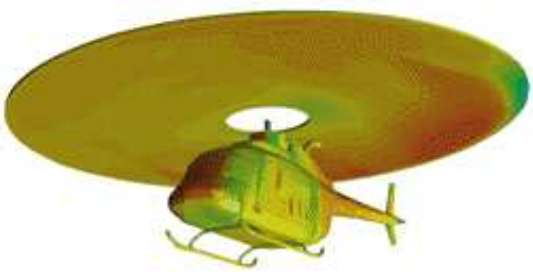
These equations along with the conservation of energy equation form a set of coupled, non-linear partial differential equations. It is not possible to solve these equations analytically for most engineering problems.

However, it is possible to obtain approximate computer-based solutions to the governing equations for a variety of engineering problems. This is the subject matter of Computational Fluid Dynamics (CFD).

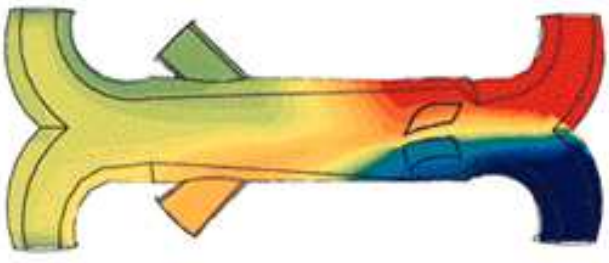
Applications of CFD

CFD is useful in a wide variety of applications and here we note a few to give you an idea of its use in industry. The simulations shown below have been performed using the FLUENT software.

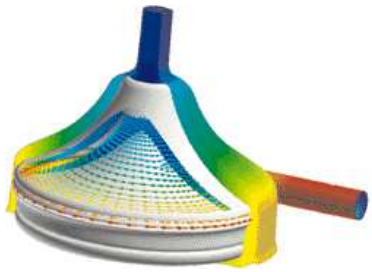
CFD can be used to simulate the flow over a vehicle. For instance, it can be used to study the interaction of propellers or rotors with the aircraft fuselage The following figure shows the prediction of the pressure field induced by the interaction of the rotor with a helicopter fuselage in forward flight. Rotors and propellers can be represented with models of varying complexity.



The temperature distribution obtained from a CFD analysis of a mixing manifold is shown below. This mixing manifold is part of the passenger cabin ventilation system on the Boeing 767. The CFD analysis showed the effectiveness of a simpler manifold design without the need for field testing.



Bio-medical engineering is a rapidly growing field and uses CFD to study the circulatory and respiratory systems. The following figure shows pressure contours and a cutaway view that reveals velocity vectors in a blood pump that assumes the role of heart in open-heart surgery.



CFD is attractive to industry since it is more cost-effective than physical testing. However, one must note that complex flow simulations are challenging and error-prone and it takes a lot of engineering expertise to obtain validated solutions.

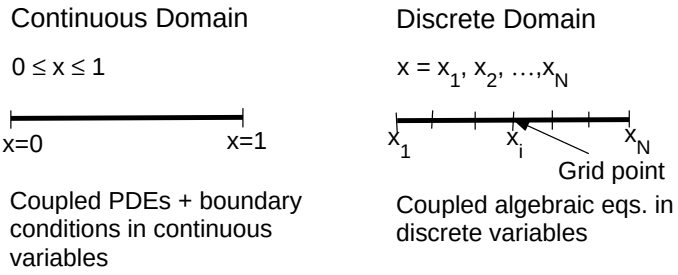
The Strategy of CFD

Broadly, the strategy of CFD is to replace the continuous problem domain with a discrete domain using a grid. In the continuous domain, each flow variable is defined at every point in the domain. For instance, the pressure p in the continuous 1D domain shown in the figure below would be given as

$$p = p(x),\ 0 < x < 1$$

In the discrete domain, each flow variable is defined only at the grid points. So, in the discrete domain shown below, the pressure would be defined only at the N grid points.

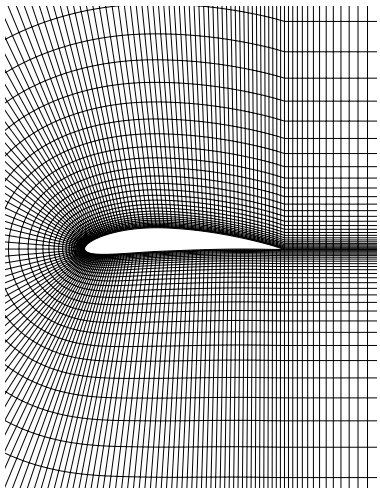
$$p_i = p(x_i), \quad i = 1, 2, \ldots, N$$



In a CFD solution, one would directly solve for the relevant flow variables only at the grid points. The values at other locations are determined by interpolating the values at the grid points.

The governing partial differential equations and boundary conditions are defined in terms of the continuous variables p, $\vec{V}$ etc. One can approximate these in the discrete domain in terms of the discrete variables p_i, $\vec{V}_i$ etc. The discrete system is a large set of coupled, algebraic equations in the discrete variables. Setting up the discrete system and solving it (which is a matrix inversion problem) involves a very large number of repetitive calculations, a task we humans palm over to the digital computer.

This idea can be extended to any general problem domain. The following figure shows the grid used for solving the flow over an airfoil. We'll take a closer look at this airfoil grid soon while discussing the finite-volume method.



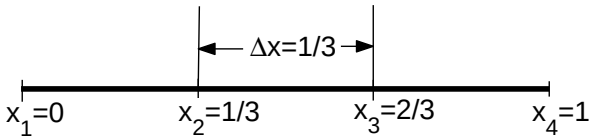
Discretization Using the Finite-Difference Method

To keep the details simple, we will illustrate the fundamental ideas underlying CFD by applying them to the following simple 1D equation:

$$\frac{du}{dx} + u^m = 0; \quad 0 \le x \le 1; \quad u(0) = 1 \tag{1}$$

We'll first consider the case where $m = 1$ when the equation is linear. We'll later consider the $m = 2$ case when the equation is nonlinear.

We'll derive a discrete representation of the above equation with $m = 1$ on the following grid:



This grid has four equally-spaced grid points with Δx being the spacing between successive points. Since the governing equation is valid at any grid point, we have

$$\left(\frac{du}{dx}\right)_i + u_i = 0 \tag{2}$$

where the subscript i represents the value at grid point x_i. In order to get an expression for $(du/dx)_i$ in terms of u at the grid points, we expand u_{i-1} in a Taylor's series:

$$u_{i-1} = u_i - \Delta x \left(\frac{du}{dx}\right)_i + O(\Delta x^2)$$

Rearranging gives

$$\left(\frac{du}{dx}\right)_i = \frac{u_i - u_{i-1}}{\Delta x} + O(\Delta x) \tag{3}$$

The error in $(du/dx)_i$ due to the neglected terms in the Taylor's series is called the truncation error. Since the truncation error above is $O(\Delta x)$, this discrete representation is termed first-order accurate.

Using (3) in (2) and excluding higher-order terms in the Taylor's series, we get the following discrete equation:

$$\frac{u_i - u_{i-1}}{\Delta x} + u_i = 0 \tag{4}$$

Note that we have gone from a differential equation to an algebraic equation!

This method of deriving the discrete equation using Taylor's series expansions is called the finite-difference method. However, most commercial CFD codes use the finite-volume or finite-element methods which are better suited for modeling flow past complex geometries. For example, the FLUENT code uses the finite-volume method whereas ANSYS uses the finite-element method. We'll briefly indicate the philosophy of the finite-volume method next but will keep using the finite-difference approach to illustrate the underlying concepts which are very similar between the different approaches with the finite-difference method being easiest to understand.

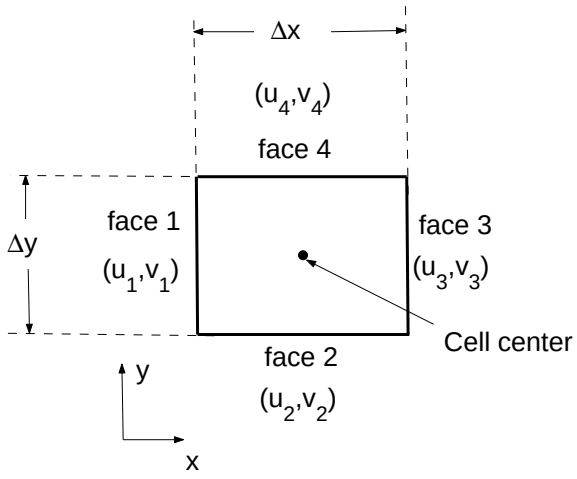
Discretization Using The Finite-Volume Method

If you look closely at the airfoil grid shown earlier, you'll see that it consists of quadrilaterals. In the finite-volume method, such a quadrilateral is commonly referred to as a "cell" and a grid point as a "node". In 2D, one could also have triangular cells. In 3D, cells are usually hexahedrals, tetrahedrals, or prisms. In the finite-volume approach, the *integral form* of the conservation equations are applied to the control volume defined by a cell to get the discrete equations for the cell. The integral form of the continuity equation for steady, incompressible flow is

$$\int_S \vec{\mathbf{V}} \cdot \hat{\mathbf{n}}\, dS = 0 \tag{5}$$

The integration is over the surface S of the control volume and $\hat{\mathbf{n}}$ is the outward normal at the surface. Physically, this equation means that the net volume flow into the control volume is zero.

Consider the rectangular cell shown below.



The velocity at face i is taken to be $\vec{V}_i = u_i\,\hat{i} + v_i\,\hat{j}$. Applying the mass conservation equation (5) to the control volume defined by the cell gives

$$-u_1\Delta y - v_2\Delta x + u_3\Delta y + v_4\Delta x = 0$$

This is the discrete form of the continuity equation for the cell. It is equivalent to summing up the net mass flow into the control volume and setting it to zero. So it ensures that the net mass flow into the cell is zero i.e. that mass is conserved for the cell. Usually, though not always, the values at the cell centers are solved for directly by inverting the discrete system. The face values u_1, v_2, etc. are obtained by suitably interpolating the cell-center values at adjacent cells.

Similarly, one can obtain discrete equations for the conservation of momentum and energy for the cell. One can readily extend these ideas to any general cell shape in 2D or 3D and any conservation equation. Take a few minutes to contrast the discretization in the finite-volume approach to that in the finite-difference method discussed earlier.

Look back at the airfoil grid. When you are using FLUENT, it's useful to remind yourself that the code is finding a solution such that mass, momentum, energy and other relevant quantities are being conserved for each cell. Also, the code directly solves for values of the flow variables at the *cell centers*; values at other locations are obtained by suitable interpolation.

Assembly of Discrete System and Application of Boundary Conditions

Recall that the discrete equation that we obtained using the finite-difference method was

$$\frac{u_i - u_{i-1}}{\Delta x} + u_i = 0$$

Rearranging, we get

$$-u_{i-1} + (1 + \Delta x)u_i = 0$$

Applying this equation to the 1D grid shown earlier at grid points $i = 2, 3, 4$ gives

$$-u_1 + (1 + \Delta x)\,u_2 = 0 \qquad (i = 2) \tag{6}$$

$$-u_2 + (1 + \Delta x)\,u_3 = 0 \qquad (i = 3) \tag{7}$$

$$-u_3 + (1 + \Delta x)\,u_4 = 0 \qquad (i = 4) \tag{8}$$

The discrete equation cannot be applied at the left boundary (i=1) since u_{i-1} is not defined here. Instead, we use the boundary condition to get

$$u_1 = 1 \tag{9}$$

Equations (6)-(9) form a system of four simultaneous algebraic equations in the four unknowns u_1, u_2, u_3 and u_4. It's convenient to write this system in matrix form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1+\Delta x & 0 & 0 \\ 0 & -1 & 1+\Delta x & 0 \\ 0 & 0 & -1 & 1+\Delta x \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{10}$$

In a general situation, one would apply the discrete equations to the grid points (or cells in the finite-volume method) in the interior of the domain. For grid points (or cells) at or near the boundary, one would apply a combination of the discrete equations and boundary conditions. In the end, one would obtain a system of simultaneous algebraic equations with the number of equations being equal to the number of independent discrete variables. The process is essentially the same as for the model equation above with the details being much more complex.

FLUENT, like other commercial CFD codes, offers a variety of boundary condition options such as velocity inlet, pressure inlet, pressure outlet, etc. It is very important that you specify the proper boundary conditions in order to have a well-defined problem. Also, read through the documentation for a boundary condition option to understand what it does before you use it (it might not be doing what you expect). A single wrong boundary condition can give you a totally wrong result.

Solution of Discrete System

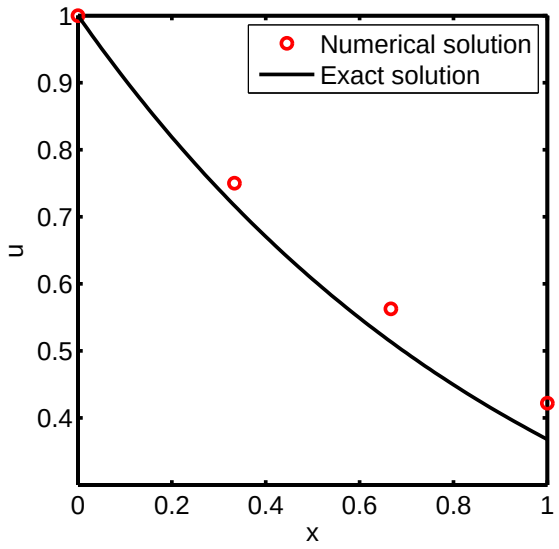
The discrete system (10) for our own humble 1D example can be easily inverted to obtain the unknowns at the grid points. Solving for u_1, u_2, u_3 and u_4 in turn and using $\Delta x = 1/3$, we get

$$u_1 = 1 \quad u_2 = 3/4 \quad u_3 = 9/16 \quad u_4 = 27/64$$

The exact solution for the 1D example is easily calculated to be

$$u_{exact} = \exp(-x)$$

The figure below shows the comparison of the discrete solution obtained on the four-point grid with the exact solution. The error is largest at the right boundary where it is equal to 14.7%.



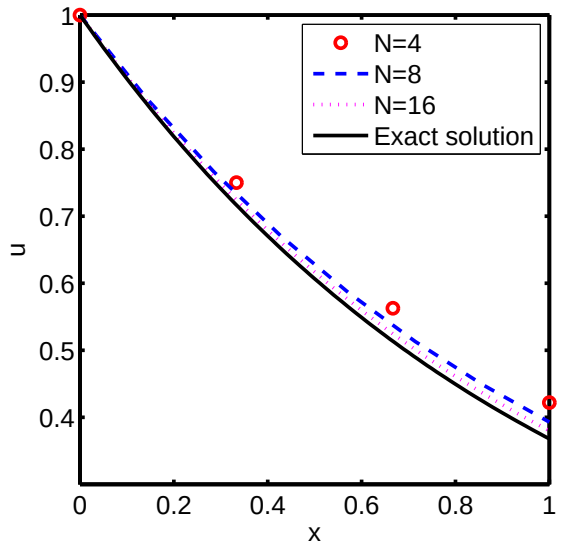
In a practical CFD application, one would have thousands to millions of unknowns in the discrete system and if one uses, say, a Gaussian elimination procedure naively to invert the matrix, you'd graduate before the computer finishes the calculation. So a lot of work goes into optimizing the matrix inversion in order to minimize the CPU time and memory

required. The matrix to be inverted is sparse i.e. most of the entries in it are zeros since the discrete equation at a grid point or cell will contain only quantities at the neighboring points or cells; verify that this is indeed the case for our matrix system (10). A CFD code would store only the non-zero values to minimize memory usage. It would also generally use an iterative procedure to invert the matrix; the longer one iterates, the closer one gets to the true solution for the matrix inversion.

Grid Convergence

While developing the finite-difference approximation for the 1D example, we saw that the truncation error in our discrete system is $O(\Delta x)$. So one expects that as the number of grid points is increased and Δx is reduced, the error in the numerical solution would decrease and the agreement between the numerical and exact solutions would get better.

Let's consider the effect of increasing the number of grid points N on the numerical solution of the 1D problem. We'll consider $N = 8$ and $N = 16$ in addition to the $N = 4$ case solved previously. We repeat the above assembly and solution steps on each of these additional grids. The resulting discrete system was solved using MATLAB. The following figure compares the results obtained on the three grids with the exact solution. As expected, the numerical error decreases as the number of grid points is increased.



When the numerical solutions obtained on different grids agree to within a level of tolerance specified by the user, they are referred to as "grid converged" solutions. The concept of grid convergence applies to the finite-volume approach also where the numerical solution, if correct, becomes independent of the grid as the cell size is reduced. It is very important that you investigate the effect of grid resolution on the solution in every CFD problem you solve. Never trust a CFD solution unless you have convinced yourself that the solution is grid converged to an acceptance level of tolerance (which would be problem dependent).

Dealing with Nonlinearity

The momentum conservation equation for a fluid is nonlinear due to the convection term $(\vec{V} \cdot \nabla)\vec{V}$. Phenomena such as turbulence and chemical reaction introduce additional non-

linearities. The highly nonlinear nature of the governing equations for a fluid makes it challenging to obtain accurate numerical solutions for complex flows of practical interest.

We will demonstrate the effect of nonlinearity by setting $m = 2$ in our simple 1D example (1):

$$\frac{du}{dx} + u^2 = 0; \quad 0 \le x \le 1; \quad u(0) = 1$$

A first-order finite-difference approximation to this equation, analogous to that in (4) for $m = 1$, is

$$\frac{u_i - u_{i-1}}{\Delta x} + u_i^2 = 0 \tag{11}$$

This is a nonlinear algebraic equation with the u_i^2 term being the source of the nonlinearity.

The strategy that is adopted to deal with nonlinearity is to linearize the equations about a *guess value* of the solution and to iterate until the guess agrees with the solution to a specified tolerance level. We'll illustrate this on the above example. Let u_{g_i} be the guess for u_i. Define

$$\Delta u_i = u_i - u_{g_i}$$

Rearranging and squaring this equation gives

$$u_i^2 = u_{g_i}^2 + 2u_{g_i}\Delta u_i + (\Delta u_i)^2$$

Assuming that $\Delta u_i \ll u_{g_i}$, we can neglect the Δu_i^2 term to get

$$u_i^2 \simeq u_{g_i}^2 + 2u_{g_i}\Delta u_i = u_{g_i}^2 + 2u_{g_i}(u_i - u_{g_i})$$

Thus,

$$u_i^2 \simeq 2u_{g_i}u_i - u_{g_i}^2$$

The finite-difference approximation (11) after linearization becomes

$$\frac{u_i - u_{i-1}}{\Delta x} + 2u_{g_i}u_i - u_{g_i}^2 = 0 \tag{12}$$

Since the error due to linearization is $O(\Delta u^2)$, it tends to zero as $u_g \to u$.

In order to calculate the finite-difference approximation (12), we need guess values u_g at the grid points. We start with an initial guess value in the first iteration. For each subsequent iteration, the u value obtained in the previous iteration is used as the guess value.

Iteration 1: $u_g^{(1)} =$ Initial guess
Iteration 2: $u_g^{(2)} = u^{(1)}$
$\vdots$
Iteration l: $u_g^{(l)} = u^{(l-1)}$

The superscript indicates the iteration level. We continue the iterations until they converge. We'll defer the discussion on how to evaluate convergence until a little later.

This is essentially the process used in CFD codes to linearize the nonlinear terms in the conservations equations, with the details varying depending on the code. The important points to remember are that the linearization is performed about a guess and that it is necessary to iterate through successive approximations until the iterations converge.

Direct and Iterative Solvers

We saw that we need to perform iterations to deal with the nonlinear terms in the governing equations. We next discuss another factor that makes it necessary to carry out iterations in practical CFD problems.

Verify that the discrete equation system resulting from the finite-difference approximation (12) on our four-point grid is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1+2\Delta x\, u_{g_2} & 0 & 0 \\ 0 & -1 & 1+2\Delta x\, u_{g_3} & 0 \\ 0 & 0 & -1 & 1+2\Delta x\, u_{g_4} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 1 \\ \Delta x\, u_{g_2}^2 \\ \Delta x\, u_{g_3}^2 \\ \Delta x\, u_{g_4}^2 \end{bmatrix} \tag{13}$$

In a practical problem, one would usually have thousands to millions of grid points or cells so that each dimension of the above matrix would be of the order of a million (with most of the elements being zeros). Inverting such a matrix directly would take a prohibitively large amount of memory. So instead, the matrix is inverted using an iterative scheme as discussed below.

Rearrange the finite-difference approximation (12) at grid point i so that u_i is expressed in terms of the values at the neighboring grid points and the guess values:

$$u_i = \frac{u_{i-1} + \Delta x\, u_{g_i}^2}{1 + 2\,\Delta x\, u_{g_i}}$$

If a neighboring value at the current iteration level is not available, we use the guess value for it. Let's say that we sweep from right to left on our grid i.e. we update u_4, then u_3 and finally u_2 in each iteration. In the m^{th} iteration, $u_{i-1}^{(l)}$ is not available while updating u_i^m and so we use the guess value $u_{g_{i-1}}^{(l)}$ for it instead:

$$u_i^{(l)} = \frac{u_{g_{i-1}}^{(l)} + \Delta x\, u_{g_i}^{(l)^2}}{1 + 2\,\Delta x\, u_{g_i}^{(l)}} \tag{14}$$

Since we are using the guess values at neighboring points, we are effectively obtaining only an approximate solution for the matrix inversion in (13) during each iteration but in the process have greatly reduced the memory required for the inversion. This tradeoff is good strategy since it doesn't make sense to expend a great deal of resources to do an exact matrix inversion when the matrix elements depend on guess values which are continuously being refined. In an act of cleverness, we have combined the iteration to handle nonlinear terms with the iteration for matrix inversion into a single iteration process. Most importantly, as the iterations converge and $u_g \to u$, the approximate solution for the matrix inversion tends towards the exact solution for the inversion since the error introduced by using u_g instead of u in (14) also tends to zero.

Thus, iteration serves two purposes:

1. It allows for efficient matrix inversion with greatly reduced memory requirements.

2. It is necessary to solve nonlinear equations.

In steady problems, a common and effective strategy used in CFD codes is to solve the unsteady form of the governing equations and "march" the solution in time until the solution converges to a steady value. In this case, each time step is effectively an iteration, with the the guess value at any time level being given by the solution at the previous time level.

Iterative Convergence

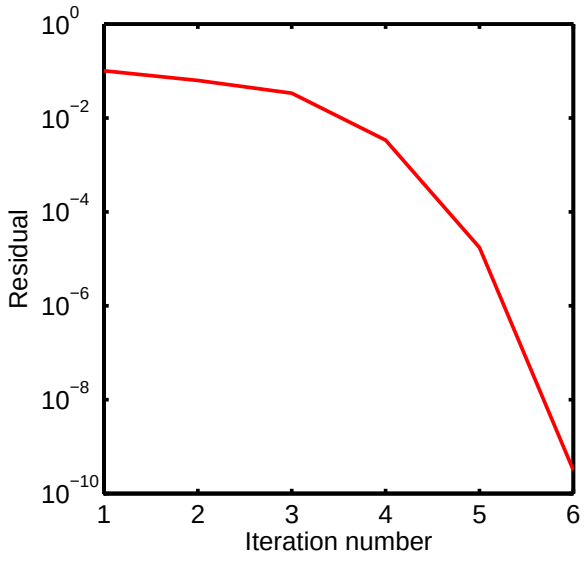
Recall that as $u_g \to u$, the linearization and matrix inversion errors tends to zero. So we continue the iteration process until some selected measure of the difference between u^g and u, refered to as the residual, is "small enough". We could, for instance, define the residual R as the RMS value of the difference between u and u_g on the grid:

$$R \equiv \sqrt{\frac{\sum_{i=1}^{N} (u_i - u_{g_i})^2}{N}}$$

It's useful to scale this residual with the average value of u in the domain. An unscaled residual of, say, 0.01 would be relatively small if the average value of u in the domain is 5000 but would be relatively large if the average value is 0.1. Scaling ensures that the residual is a relative rather than an absolute measure. Scaling the above residual by dividing by the average value of u gives

$$R = \left(\sqrt{\frac{\sum_{i=1}^{N} (u_i - u_{g_i})^2}{N}} \right) \left(\frac{N}{\sum_{i=1}^{N} u_i} \right) = \frac{\sqrt{N \sum_{i=1}^{N} (u_i - u_{g_i})^2}}{\sum_{i=1}^{N} u_i} \tag{15}$$

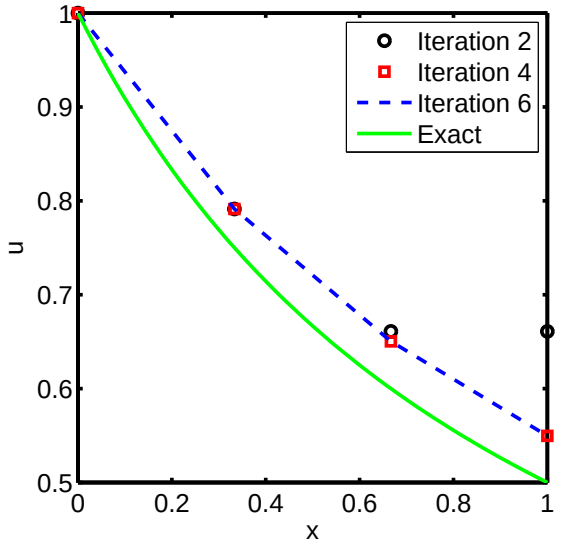
For the nonlinear 1D example, we'll take the initial guess at all grid points to be equal to the value at the left boundary i.e. $u_g^{(1)} = 1$. In each iteration, we update u_g, sweep from right to left on the grid updating, in turn, u_4, u_3 and u_2 using (14) and calculate the residual using (15). We'll terminate the iterations when the residual falls below 10^{-9} (which is referred to as the convergence criterion). Take a few minutes to implement this procedure in MATLAB which will help you gain some familiarity with the mechanics of the implementation. The variation of the residual with iterations obtained from MATLAB is shown below. Note that logarithmic scale is used for the ordinate. The iterative process converges to a level smaller than 10^{-9} in just 6 iterations. In more complex problems, a lot more iterations would be necessary for achieving convergence.



The solution after 2,4 and 6 iterations and the exact solution are shown below. It can easily be verified that the exact solution is given by

$$u_{exact} = \frac{1}{x+1}$$

The solutions for iterations 4 and 6 are indistinguishable on the graph. This is another indication that the solution has converged. The converged solution doesn't agree well with the exact solution because we are using a coarse grid for which the truncation error is relatively large. The iterative convergence error, which is of order 10^{-9}, is swamped out by the truncation error of order 10^{-1}. So driving the residual down to 10^{-9} when the truncation error is of order 10^{-1} is a waste of computing resources. In a good calculation, both errors would be of comparable level and less than a tolerance level chosen by the user. The agreement between the numerical and exact solutions should get much better on refining the grid as was the case for $m = 1$.



Some points to note:

1. Different codes use slightly different definitions for the residual. Read the documentation to understand how the residual is calculated.

2. In the FLUENT code, residuals are reported for each conservation equation. A discrete conservation equation at any cell can be written in the form LHS = 0. For any iteration, if one uses the current solution to compute the LHS, it won't be exactly equal to zero, with the deviation from zero being a mesaure of how far one is from achieving convergence. So FLUENT calculates the residual as the (scaled) mean of the absolute value of the LHS over all cells.

3. The convergence criterion you choose for each conservation equation is problem- and code-dependent. It's a good idea to start with the default values in the code. One may then have to tweak these values.

Numerical Stability

In our previous 1D example, the iterations converged very rapidly with the residual falling below the convergence criterion of 10^{-9} in just 6 iterations. In more complex problems, the iterations converge more slowly and in some instances, may even diverge. One would like to know a priori the conditions under which a given numerical scheme converges. This is determined by performing a stability analysis of the numerical scheme. A numerical method is referred to as being stable when the iterative process converges and as being unstable when it diverges. It is not possible to carry out an exact stability analysis for the Euler or Navier-Stokes equations. But a stability analysis of simpler, model equations provides useful insight and approximate conditions for stability. A common strategy used in CFD codes for steady problems is to solve the unsteady equations and march the solution in time until it converges to a steady state. A stability analysis is usually performed in the context of time-marching.

While using time-marching to a steady state, we are only interested in accurately obtaining the asymptotic behavior at large times. So we would like to take as large a time-step Δt as possible to reach the steady state in the least number of time-steps. There is usually a maximum allowable time-step Δt_{max} beyond which the numerical scheme is unstable. If $\Delta t > \Delta t_{max}$, the numerical errors will grow exponentially in time causing the solution to diverge from the steady-state result. The value of Δt_{max} depends on the numerical discretization scheme used. There are two classes of numerical shemes, explicit and implicit, with very different stability characteristics as we'll briefly discuss next.

Explicit and Implicit Schemes

The difference between explicit and implicit schemes can be most easily illustrated by applying them to the wave equation

$$\frac{\partial u}{\partial t} + c\,\frac{\partial u}{\partial x} = 0$$

where c is the wavespeed. One possible way to discretize this equation at grid point i and time-level n is

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} + c\,\frac{u_i^{n-1} - u_{i-1}^{n-1}}{\Delta x} = O(\Delta t, \Delta x) \tag{16}$$

The crucial thing to note here is that the spatial derivative is evaluated at the $n-1$ time-level. Solving for u_i^n gives

$$u_i^n = \left[1 - \left(\frac{c\Delta t}{\Delta x}\right)\right]\, u_i^{n-1} + \left(\frac{c\Delta t}{\Delta x}\right) u_{i-1}^{n-1} \tag{17}$$

This is an explicit expression i.e. the value of u_i^n at any grid point can be calculated directly from this expression without the need for any matrix inversion. The scheme in (16) is known as an explicit scheme. Since u_i^n at each grid point can be updated independently, these schemes are easy to implement on the computer. On the downside, it turns out that this scheme is stable only when

$$\mathrm{C} \equiv \frac{c\Delta t}{\Delta x} \leq 1$$

where C is called the Courant number. This condition is refered to as the Courant-Friedrichs-Lewy or CFL condition. While a detailed derivation of the CFL condition through stability analysis is outside the scope of the current discussion, it can seen that the coefficient of u_i^{n-1}

in (17) changes sign depending on whether $C > 1$ or $C < 1$ leading to very different behavior in the two cases. The CFL condition places a rather severe limitation on Δt_{max}.

In an implicit scheme, the spatial derivative term is evaluated at the n time-level:

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} + c\,\frac{u_i^n - u_{i-1}^n}{\Delta x} = O(\Delta t, \Delta x)$$

In this case, we can't update u_i^n at each grid point independently. We instead need to solve a system of algebraic equations in order to calculate the values at all grid points simultaneously. It can be shown that this scheme is unconditionally stable for the wave equation so that the numerical errors will be damped out irrespective of how large the time-step is.

The stability limits discussed above apply specifically to the wave equation. In general, explicit schemes applied to the Euler or Navier-Stokes equations have the same restriction that the Courant number needs to be less than or equal to one. Implicit schemes are *not* unconditonally stable for the Euler or Navier-Stokes equations since the nonlinearities in the governing equations often limit stability. However, they allow a much larger Courant number than explicit schemes. The specific value of the maximum allowable Courant number is problem dependent.

Some points to note:

1. CFD codes will allow you to set the Courant number (which is also referred to as the CFL number) when using time-stepping. Taking larger time-steps leads to faster convergence to the steady state, so it is advantageous to set the Courant number as large as possible, within the limits of stability, for steady problems.

2. You may find that a lower Courant number is required during startup when changes in the solution are highly nonlinear but it can be increased as the solution progresses.

Turbulence Modeling

There are two radically different states of flows that are easily identified and distinguished: laminar flow and turbulent flow. Laminar flows are characterized by smoothly varying velocity fields in space and time in which individual "laminae" (sheets) move past one another without generating cross currents. These flows arise when the fluid viscosity is sufficiently large to damp out any perturbations to the flow that may occur due to boundary imperfections or other irregularities. These flows occur at low-to-moderate values of the Reynolds number. In contrast, turbulent flows are characterized by large, nearly random fluctuations in velocity and pressure in both space and time. These fluctuations arise from instabilities that grow until nonlinear interactions cause them to break down into finer and finer whirls that eventually are dissipated (into heat) by the action of viscosity. Turbulent flows occur in the opposite limit of high Reynolds numbers.

A typical time history of the flow variable u at a fixed point in space is shown in Fig. 1(a). The dashed line through the curve indicates the "average" velocity. We can define three types of averages:

1. Time average

2. Volume average

3. Ensemble average

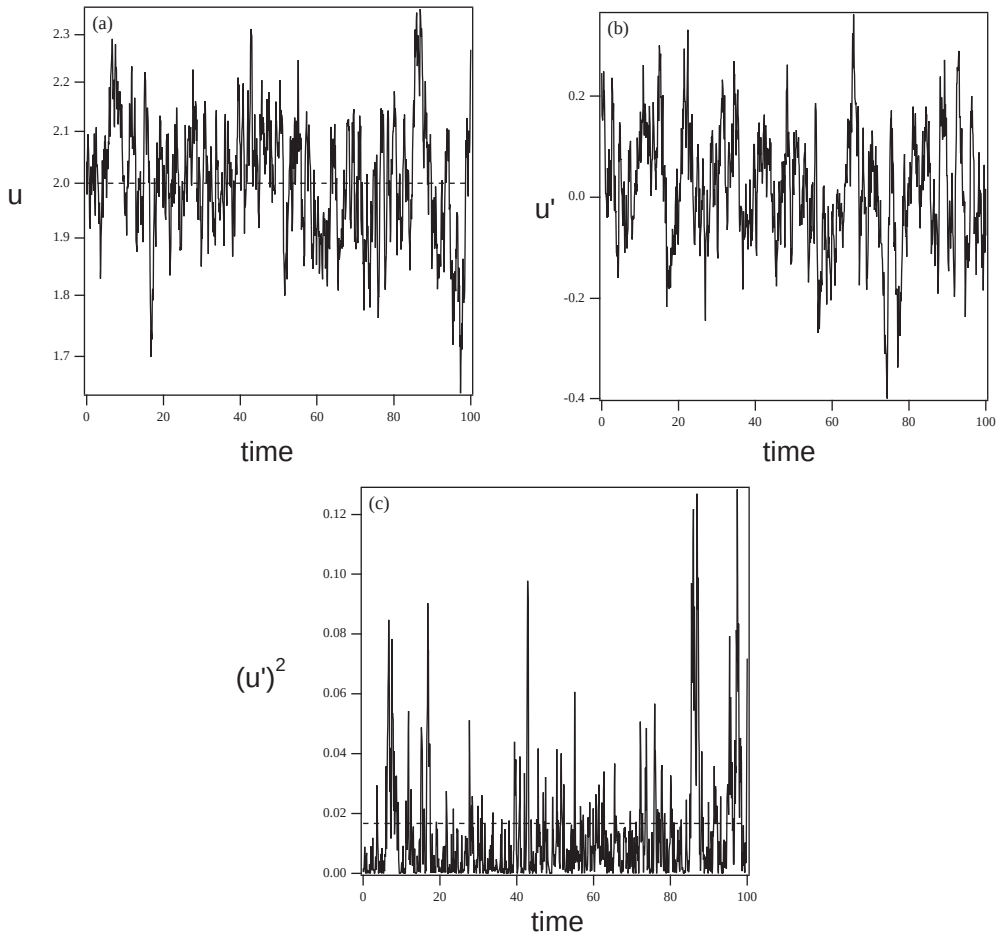


Figure 1: Example of a time history of a component of a fluctuating velocity at a point in a turbulent flow. (a) Shows the velocity, (b) shows the fluctuating component of velocity $u' \equiv u - \overline{u}$ and (c) shows the square of the fluctuating velocity. Dashed lines in (a) and (c) indicate the time averages.

The most mathematically general average is the ensemble average, in which you repeat a given experiment a large number of times and average the quantity of interest (say velocity) at the same position and time in each experiment. For practical reasons, this is rarely done. Instead, a time or volume average (or combination of the two) is made with the assumption that they are equivalent to the ensemble average. For the sake of this discussion, let us define the time average for a stationary flow[1] as

$$\overline{u}(y) \equiv \lim_{\tau \to \infty} \frac{1}{2\tau} \int_{-\tau}^{\tau} u(y,t) dt \tag{18}$$

The deviation of the velocity from the mean value is called the fluctuation and is usually defined as

$$u' \equiv u - \overline{u} \tag{19}$$

Note that by definition $\overline{u'} = 0$ (the average of the fluctuation is zero). Consequently, a better measure of the *strength* of the fluctuation is the average of the *square* of a fluctuating

[1] A stationary flow is defined as one whose statistics are not changing in time. An example of a stationary flow is steady flow in a channel or pipe.

variable. Figures 1(b) and 1(c) show the time evolution of the velocity fluctuation, u', and the square of that quantity, u'^2. Notice that the latter quantity is always greater than zero as is its average.

The equations governing a turbulent flow are precisely the same as for a laminar flow; however, the solution is clearly much more complicated in this regime. The approaches to solving the flow equations for a turbulent flow field can be roughly divided into two classes. Direct numerical simulations (DNS) use the speed of modern computers to numerically integrate the Navier Stokes equations, resolving all of the spatial and temporal fluctuations, without resorting to modeling. In essence, the solution procedure is the same as for laminar flow, except the numerics must contend with resolving all of the fluctuations in the velocity and pressure. DNS remains limited to very simple geometries (e.g., channel flows, jets and boundary layers) and is extremely expensive to run.[2] The alternative to DNS found in most CFD packages (including FLUENT) is to solve the Reynolds Averaged Navier Stokes (RANS) equations. RANS equations govern the *mean* velocity and pressure. Because these quantities vary smoothly in space and time, they are much easier to solve; however, as will be shown below, they require *modeling* to "close" the equations and *these models introduce significant error into the calculation.*

To demonstrate the closure problem, we consider fully developed turbulent flow in a channel of height $2H$. Recall that with RANS we are interested in solving for the *mean* velocity $\overline{u}(y)$ only. If we formally average the Navier Stokes equations and simplify for this geometry we arrive at the following

$$\frac{d\overline{u'v'}}{dy} + \frac{1}{\rho}\frac{d\overline{p}}{dx} = \nu\frac{d^2\overline{u}(y)}{dy^2} \tag{20}$$

subject to the boundary conditions

$$y = 0 \quad \frac{d\overline{u}}{dy} = 0 \ , \tag{21}$$

$$y = H \quad \overline{u} = 0 \ , \tag{22}$$

The kinematic viscosity $\nu{=}\mu/\rho$. The quantity $\overline{u'v'}$, known as the Reynolds stress,[3] is a higher-order moment that must be modeled in terms of the knowns (i.e., $\overline{u}(y)$ and its derivatives). This is referred to as the "closure" approximation. The quality of the modeling of this term will determine the reliability of the computations.[4]

Turbulence modeling is a rather broad discipline and an in-depth discussion is beyond the scope of this introduction. Here we simply note that the Reynolds stress is modeled in terms of two turbulence parameters, the turbulent kinetic energy k and the turbulent energy dissipation rate ϵ defined below

$$k \equiv \frac{1}{2}\left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}\right) \tag{23}$$

$$\epsilon \equiv \nu\left[\left(\frac{\partial u'}{\partial x}\right)^2 + \left(\frac{\partial u'}{\partial y}\right)^2 + \left(\frac{\partial u'}{\partial z}\right)^2 + \left(\frac{\partial v'}{\partial x}\right)^2 + \left(\frac{\partial v'}{\partial y}\right)^2 + \left(\frac{\partial v'}{\partial z}\right)^2\right.$$

[2]The largest DNS to date was recently published by Kaneda et al., *Phys. Fluids* **15**(2):L21–L24 (2003); they used 4096^3 grid point, which corresponds roughly to 0.5 terabytes of memory per variable!

[3]Name after the same Osborne Reynolds from which we get the Reynolds number.

[4]Notice that if we neglect the Reynolds stress the equations reduce to the equations for laminar flow; thus, the Reynolds stress is solely responsible for the difference in the mean profile for laminar (parabolic) and turbulent (blunted) flows.

$$+\left(\frac{\partial w'}{\partial x}\right)^2+\left(\frac{\partial w'}{\partial y}\right)^2+\left(\frac{\partial w'}{\partial z}\right)^2\Bigg] \tag{24}$$

where (u', v', w') is the fluctuating velocity vector. The kinetic energy is *zero* for laminar flow and can be as large as 5% of the kinetic energy of the mean flow in a highly turbulent case. The family of models is generally known as k–ϵ and they form the basis of most CFD packages (including FLUENT). We will revisit turbulence modeling towards the end of the semester.