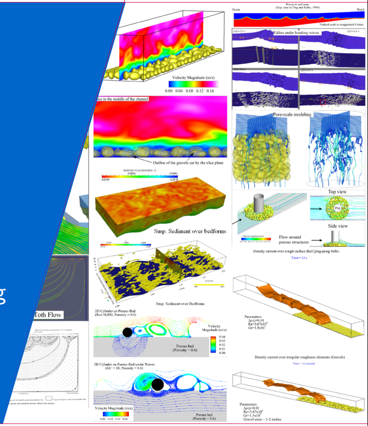
PENNSTATE

1 8 5 5

CE597D Special Topics, Fall, 2014

Chapter 4, Part 3: Unsteady Problems

Xiaofeng Liu, Ph.D., P.E.
Assistant Professor
Department of Civil and Environmental Engineering
Pennsylvania State University
xliu@engr.psu.edu



Outline

Time Advancement

Advancing the Solution in Time

$$\underbrace{\frac{\partial \rho\phi}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla\cdot(\rho\phi\mathbf{u})}_{\text{convection term}} - \underbrace{\nabla\cdot(\Gamma\nabla\phi)}_{\text{diffusion term}} = \underbrace{S_\phi}_{\text{source term}}$$

- Two basic types of time advancement: Implicit and explicit schemes. Properties of the algorithm critically depend on this choice, but both are useful under given circumstances
- There is a number of methods, with slightly different properties, *e.g.* fractional step methods,
- Temporal accuracy depends on the choice of scheme and time step size
- Steady-state simulations
  - If equations are linear, this can be solved directly. We have done this for diffusion and advection-diffusion equations.
  - For non-linear equations, **relaxation methods** are used. A pseudo time is defined.

Time Advancement

Explicit Schemes

- The time derivatives are evaluated using the currently available ϕ and the new ϕ is obtained from the time term
- All other terms (advection, diffusion, source, etc.) are evaluated using old time values.
- **Courant number limit** is the major limitation of explicit methods: in each time step, information can only propagate at the order of cell size; otherwise the algorithm is unstable.

$$CFL = \frac{U\Delta t}{\Delta x} < CFL_{max}$$

- Quick and efficient, no additional storage
- Very bad for elliptic behaviour

$$\underbrace{\frac{\partial \rho\phi}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla \cdot (\rho\phi\mathbf{u})}_{\text{convection term}} - \underbrace{\nabla \cdot (\Gamma\nabla\phi)}_{\text{diffusion term}} = \underbrace{S_\phi}_{\text{source term}}$$

Time Advancement

Implicit Schemes

- The algorithm is based on the method: each term is expressed in matrix form and the resulting linear system is solved
- A new solution takes into account the new values in the complete domain: ideal for elliptic problems
- Implicitness removed the Courant number limitation: we can take larger time-steps (but Δt is still subject to other things such as accuracy requirement)
- Substantial additional storage: matrix coefficients.

Time Advancement

Summary of the temporal schemes in the book:

Scheme	Stability	Accuracy	Positive coefficient criterion
Explicit	Conditionally stable	First-order	$\Delta t < \frac{\rho(\Delta x)^2}{2\Gamma}$
Crank-Nicolson	Unconditionally stable	Second-order	$\Delta t < \frac{\rho(\Delta x)^2}{\Gamma}$
Implicit	Unconditionally stable	First-order	Always positive

Time Schemes in OpenFOAM

From the point of view of temporal discretization, we need to treat both the time derivative term ($\partial/\partial t$) and the spatial derivative terms (such as gradient, laplacian, divergence, etc.), as well as the source term.

$$\underbrace{\frac{\partial \rho \phi}{\partial t}}_{\text{temporal derivative}} + \underbrace{\nabla \cdot (\rho \phi \mathbf{u})}_{\text{convection term}} - \underbrace{\nabla \cdot (\Gamma \nabla \phi)}_{\text{diffusion term}} = \underbrace{S_\phi}_{\text{source term}}$$

We can write this equation in a more compact form as

$$\underbrace{\frac{\partial \rho \phi}{\partial t}}_{\text{temporal derivative}} + \underbrace{A\phi}_{\text{spatial derivatives + source}} = 0$$

where A is an operator including everything else except the temporal derivative.

Time Schemes in OpenFOAM

The first time derivative ($\partial/\partial t$) term is integrated over a control volume as usual

$$\int_V \frac{\partial \rho \phi}{\partial t} dV = \frac{\partial}{\partial t} \int_V \rho \phi dV$$

This term is discretized by time differencing schemes using

- new values $\phi^n = \phi(t + \Delta t)$ at the time step we are solving for
- old values $\phi^o = \phi(t)$ that were stored from the previous time step
- old-old values $\phi^{oo} = \phi(t - \Delta t)$ stored from a time step previous to the last

Time Schemes in OpenFOAM

Euler implicit scheme (first order accurate in time)

$$\frac{\partial}{\partial t}\int_V \rho\phi dV = \frac{(\rho_P \phi_P V)^n - (\rho_P \phi_P V)^o}{\Delta t}$$

Backward differencing scheme (second order accurate in time):

$$\frac{\partial}{\partial t}\int_V \rho\phi dV = \frac{3\left(\rho_P \phi_P V\right)^n - 4\left(\rho_P \phi_P V\right)^o + \left(\rho_P \phi_P V\right)^{oo}}{2\Delta t}$$

Time Schemes in OpenFOAM

For the second time derivative, the only option is Euler scheme:

$$\frac{\partial}{\partial t}\int_V \rho \frac{\partial \phi}{\partial t} dV = \frac{\left(\rho_P \phi_P V\right)^n - 2\left(\rho_P \phi_P V\right)^o + \left(\rho_P \phi_P V\right)^{oo}}{\Delta t^2}$$

Time Schemes in OpenFOAM

Next we need to consider how to treat the spatial derivatives in a transient problem. Integrate the whole equation over a control volume and over time, we have

$$\int_{t}^{t+\Delta t} \left[\frac{\partial}{\partial t} \int_{V} \rho\phi \, dV + \int_{V} A\phi \, dV \right] dt = 0$$

The discretization of the temporal derivative and operator A is independent. For example, we can use Euler implicit scheme for the temporal,

$$\int_{t}^{t+\Delta t} \left[\frac{\partial}{\partial t} \int_{V} \rho\phi \, dV \right] dt = \frac{(\rho_P \phi_P V)^n - (\rho_P \phi_P V)^o}{\Delta t} \Delta t$$

Time Schemes in OpenFOAM

For the operator A, it can be expressed as

$$\int_t^{t+\Delta t} \left[\int_V A\phi \, dV \right] dt = \int_t^{t+\Delta t} A^* \, dt$$

where A^* represent the spatial discretization of the operator. The time integral on the right hand side of the equation can be discretized in different ways:

- Euler implicit: first order accurate, bounded, unconditionally stable

  $$\int_t^{t+\Delta t} A^* dt = A^* \phi^n \Delta t$$

- Explicit: first order accurate, unstable if the Courant number $C_o = \frac{\mathbf{U}_f \cdot \mathbf{d}}{|\mathbf{d}|^2 \delta t} > 1$

  $$\int_t^{t+\Delta t} A^* dt = A^* \phi^o \Delta t$$

- Crank Nicolson: second order accurate, unconditionally stable, not guaranteed boundedness

  $$\int_t^{t+\Delta t} A^* dt = A^* \left(\frac{\phi^n + \phi^o}{2} \right) \Delta t$$

Time Schemes in OpenFOAM

The first time derivative ($\partial/\partial t$) terms are specified in the *ddtSchemes* sub-dictionary. The discretisation scheme for each term can be selected from those listed in the following table.

Scheme	Description
Euler	First order, bounded, implicit
localEuler	Local-time step, first order, bounded, implicit
CrankNicolson Ψ	Second order, bounded, implicit
backward	Second order, implicit
steadyState	Does not solve for time derivatives

There is an off-centering coefficient Ψ with the CrankNicolson scheme that blends it with the Euler scheme.

- $\Psi = 1$ corresponds to pure CrankNicolson
- $\Psi = 0$ corresponds to pure Euler

The blending coefficient can help to improve stability in cases where pure CrankNicolson are unstable.

Second time derivative ($\partial^2/\partial t^2$) terms are specified in the *d2dt2Schemes* sub-dictionary. Only the Euler scheme is available for *d2dt2Schemes*.

Time Schemes in OpenFOAM

In OpenFOAM® , the temporal discretization of spatial operator A is actually controlled by the implementation of the spatial derivative in the equation. For example, the transient diffusion equation

$$\frac{\partial \phi}{\partial t} = \Gamma \nabla^2 \phi$$

An Euler implicit treatment would be

*solve(fvm::ddt(phi) == Gamma*fvm::laplacian(phi));*

where the *laplacian* member function in the *fvm* namespace in OpenFOAM® treats the Laplacian term implicitly (Euler implicit). However, you can specify how to discretize the *fvm::ddt(phi)* term differently in the *fvSchemes* file.

Time Schemes in OpenFOAM

An explicit treatment would be

*solve(fvm::ddt(phi) == Gamma*fvc::laplacian(phi))*

where the *laplacian* member function in the *fvc* namespace in OpenFOAM® treats the Laplacian term explicitly.

Time Schemes in OpenFOAM

The Crank Nicolson scheme treatment would be

*solve(fvm::ddt(phi) == Gamma*0.5*(fvm::laplacian(phi) + fvc::laplacian(phi)))*

where it uses the *laplacian* member functions in both the *fvm* and *fvc* namespaces in OpenFOAM® .

For Crank Nicolson scheme, you also have the choice of directly specify the *ddtScheme* as *CrankNicolson* in the *fvSchemes* file.

```
ddtSchemes
{
    default       CrankNicolson 1.0;
}
```

Here the number "1.0" following the keyword *CrankNicolson* means it is a pure *CrankNicolson* scheme. If the number is "0", then it becomes a pure *Euler* implicit scheme. Any number in between "0" and "1.0" means a blending between *Euler* and *CrankNicolson*.

Pseudo-transient approach for steady state problem

In previous chapters, we have focused on steady state physics, i.e., the $\partial\phi/\partial t$ term does not show up in the governing equation.

However, steady state solution can also be simulated as the final converged solution of a transient simulation. We call this approach the "pseudo-transient" approach.

The pseudo-transient approach is attractive for circumstances where for example nonlinearity presents in the problem. It will pose great difficulty to the solver if don't use the pseudo-transient approach.

Pseudo-transient approach for steady state problem

Closely related to this topic is the under-relaxation which controls the change of ϕ, i.e.,

$$\phi_P^n = \phi_P^{n-1} + \alpha \left(\phi_P^{n*} - \phi_P^{n-1} \right)$$

where α is the relaxation factor.

- $\alpha < 1$: under-relaxation. This will slow down the convergence but increase the stability.
- $\alpha = 1$: no relaxation.
- $\alpha > 1$: over-relaxation. This can be used to accelerate the convergence but it will also decrease the stability.

Pseudo-transient approach for steady state problem

More on the relaxation:

- Unfortunately, the specification of the relaxation factor α might be problem-dependent.It needs user's experience
- The general purpose of under-relaxation is to suppress oscillations
- OpenFOAM® comes with some default relaxation factor values which are usually recommended

```
relaxationFactors
{
    p         0.3;
    U         0.7;
    k         0.7;
    omega     0.7;
}
```

- If you write your own code, you need to relax the solution variable explicitly. For example, in the solver simpleFoam, the velocity and pressure are relaxed explicitly by

```
UEqn.relax()
p.relax()
```

Other schemes

OpenFOAM® has temporal schemes to solve ordinary differential equations (ODEs):

- source code located at:

  `src/ODE/ODESolvers`
- Fifth-order Cash-Karp Runge-Kutta for non-stiff systems: `RK`
- Fourth-order semi-implicit Runge-Kutta scheme of Kaps, Rentrop and Rosenbrock for stiff systems: `KRR`
- Semi-Implicit Bulirsh-Stoer: SIBS

Questions?