A detailed look at fvSchemes and fvSolution

13th OpenFOAM Workshop, Shanghai

Prof Gavin Tabor

25th June 2018

Numerics

Crucial part of CFD – three aspects :

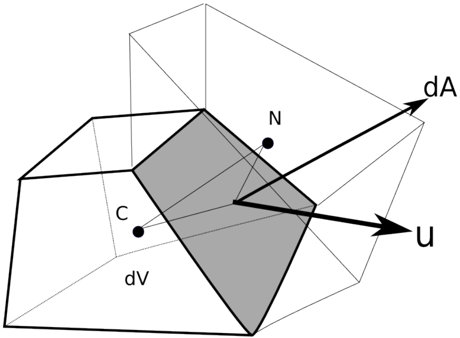
1. Differencing schemes – representation of individual derivatives $\frac{\partial}{\partial t}$, $\nabla.$ etc
2. Matrix inversion – iterative solution of individual equations
3. Algorithms – SIMPLE, PISO, Pimple (etc)

Numerical methods for all these need to be specified; scheme parameters typically also need to be provided. 1. is in `fvSchemes`; 2, 3 in `fvSolution`

Aim of the training session to review all of this.

Examples tested using OF5 (Foundation version)

Differencing Schemes: Overview



OF is a Finite Volume code– discretise by dividing space into cells, integrating equations over each cell.

Use Gauss' theorem to convert (spatial) derivatives into fluxes on faces – need to interpolate from cell centres to evaluate variables.

The interpolation process gives the derivatives (still talk about differencing schemes though).

Derivatives

Navier-Stokes equations ;

$$\nabla . \underline{u} = 0$$

$$\frac{\partial \underline{u}}{\partial t} + \nabla . \underline{u}\ \underline{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \underline{u}$$

Several types of derivative here :

$$\frac{\partial}{\partial t} \quad , \quad \frac{\partial^2}{\partial t^2}$$ time derivatives

$$\nabla p = \underline{i} \frac{\partial p}{\partial x} + \underline{j} \frac{\partial p}{\partial y} + \underline{k} \frac{\partial p}{\partial z}$$ Gradient

UNIVERSITY OF EXETER

$$\nabla.\underline{u}\ \underline{u} \qquad \text{Transport (divergence) term}$$

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \qquad \text{Laplacian}$$

Each has its own peculiarities when being evaluated.

Details of discretisation methods contained in sub-dictionaries in `system/fvSchemes` : `timeScheme`, `gradSchemes`, `divSchemes`, `laplacianSchemes`

`fvSchemes` also contains `interpolationSchemes`, `snGradSchemes` and `wallDist` dictionaries

fvSchemes file

Each entry in an equation needs its discretisation scheme specified. Keywords use OpenFOAM's top level programming syntax.

```
fvScalarMatrix TEqn
(
    fvm::ddt(T)
  + fvm::div(phi, T)
  - fvm::laplacian(DT, T)
    ==
    fvOptions(T)
);
```

```
divSchemes
{
    default       none;
    div(phi, T) Gauss limitedLinear 1.0;
}
```

Default case can also be specified with the `default` keyword. Most terms will discretise in the same way throughout a code (eg. same time discretisation) – however divergence term more diverse; `divSchemes` entries likely to be different (use `default none`).

UNIVERSITY OF EXETER

Transport equation

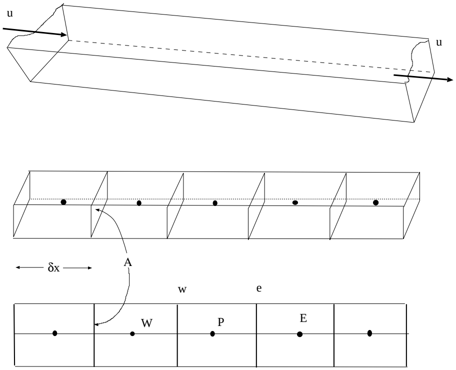
Mathematical form

$$\frac{\partial q}{\partial t} + \nabla . q\underline{u} = \Gamma \nabla^2 q$$

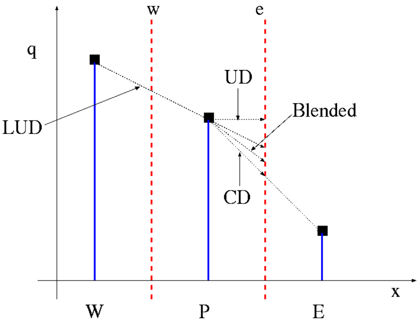
Simple problem (1-d) of contaminant flow in a channel : discretise to give

$$\frac{dq}{dt} + \frac{u}{2\delta x}(q_E - q_W) = \frac{\Gamma}{\delta x^2}(q_E - 2q_P + q_W)$$

(explicit scheme → spreadsheet)



Interpolation



Upwind, central extremes; possess undesirable effects

More sophisticated blending to improve results

Further complications if face not at 90° or not evenly between cell centres.

Advanced interpolation

Hybrid differencing schemes blend UD and CD according to some function, generally based on low Peclet number (relates advection to diffusion). Problem is that accuracy of solution limited by lowest order scheme (UD).

Alternative; utilise further points to provide higher order schemes. Eg. Linear Upwind Differencing (LUD), Quadratic Upstream Interpolation for Convective Kinetics (QUICK). More difficult to implement on arbitrary meshes.

Interpolation – requirements

UD introduces excessive numerical viscosity; CD introduces oscillation. Discretisation scheme should be :

Conservative – requires flux through common face represented in a *consistent* manner.

Bounded – in the absence of sources, the internal values of q should be bounded by the boundary values of q

Transportive – relative importance of diffusion and convection should be reflected in interpolation scheme

The boundedness criteria is violated for CD. To improve this, we consider *total variation*

$$TV(q) = \sum_i q_i - q_{i-1}$$

TVD schemes

Desirable property for an interpolation scheme is that it

1. should not create local maxima/minima
2. should not enhance existing local maxima/minima

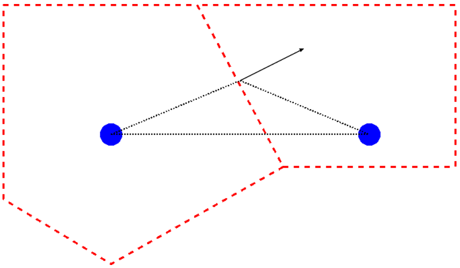
If so, the scheme is said to be *monotonicity-preserving*.

For monotonicity to be satisfied the total variation must not increase. Schemes which decrease TV are called Total Variation Diminishing (TVD) schemes.

Non-uniform meshes

Non-uniform meshes may introduce complications. In particular :

1. Face e may not bisect P–E
2. Face centre not in line with P–E
3. Face normal not in line with P–E



Modifications/corrections can be made to differencing schemes to account for these issues.

1d Transport Eqn in OpenFOAM

We can implement the same case (1-d transport of a contaminant) in OpenFOAM using `scalarTransportFoam` (in `tutorials/basic`) – see how the different options affect the solution.

Two main terms to consider : `ddtSchemes` and `divSchemes`

`ddtSchemes`; options include `steadyState`, `Euler` (1st order implicit), `backward` (2nd order implicit, unbounded), `CrankNicolson` (2nd order bounded) and `localEuler` (pseudoTransient).

So :

$$\frac{\partial q}{\partial t} = \frac{q^n - q^o}{\delta t} \qquad \text{Euler differencing}$$

To improve accuracy towards 2nd order; increase the number of past timesteps :

$$\frac{\partial q}{\partial t} = \frac{1}{\delta t}\left(\frac{3}{2}q^n - 2q^o + \frac{1}{2}q^{oo}\right) \qquad \text{backward differencing}$$

(Adams-Bashforth 2nd order scheme)

Alternatively; evaluate scheme at the mid-point of the timestep – Crank-Nicholson. So if

$$\frac{\partial q}{\partial t} = f(q) \qquad \text{discretise rhs as} \quad \frac{1}{2}\left[f(q)^n + f(q)^o\right]$$

Both of these can be unstable (+ unbounded). OF Crank-Nickolson scheme uses blending with Euler; blending function ψ :

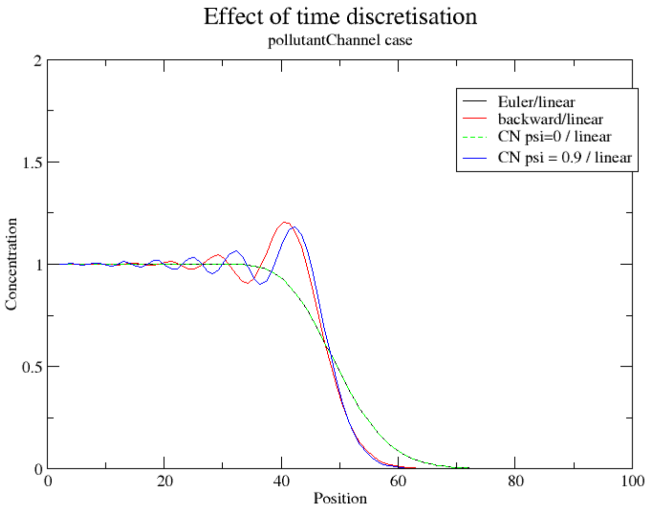
$$\text{rhs} = \frac{\psi}{2}f(q)^n + \left(1 - \frac{\psi}{2}\right)f(q)^o$$

$$\psi = \begin{cases} 1 & \text{pure Crank-Nicholson} \\ 0 & \text{recovers Euler} \\ 0.9 & \text{Recommended for practical problems} \end{cases}$$

Case `polutantChannel` illustrates this :

1. Run `scalarTransportFoam`
2. `sampleDict` provided to sample along mid-line. Run
   `postProcess -func sampleDict`
   to generate results
3. `xmGrace postProcessing/sampleDict/500/lineX1_T.xy`

UNIVERSITY OF EXETER


Effect of time discretisation
pollutantChannel case
Concentration
Position
0
0.5
1
1.5
2
0
20
40
60
80
100
Euler/linear
backward/linear
CN psi=0 / linear
CN psi = 0.9 / linear

Other terms

A useful utility is `foamSearch` :

```
foamSearch $FOAM_TUTORIALS fvSchemes ddtSchemes
```

will list all `ddtSchemes` used in the tutorials. `ddtSchemes.default` will give defaults.

Source code found in

```
$WM_PROJECT_DIR/src/finiteVolume/finiteVolume
```

and appropriate sub-directories

divSchemes

divSchemes probably most tricky for CFD. interpolationSchemes relates to the evaluation of the flux ϕ (phi), but is almost always linear.

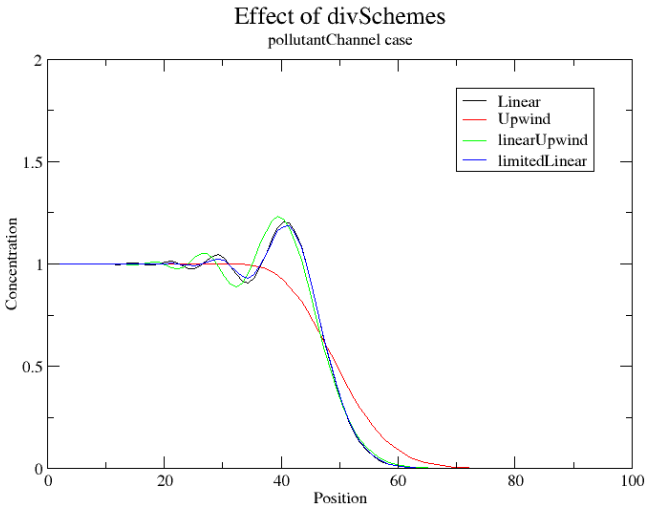
divSchemes entries are of the form :

```
div(phi,U)        Gauss linear;
```

Usually default none; is used as the schemes will vary between equations.

Gauss indicates derivatives are evaluated via Gauss' theorem (no real choice there).

upwind is standard 1st order upwind interpolation (usually too diffusive). linear is standard 2nd order interpolation – unbounded.


Effect of divSchemes
pollutantChannel case
Linear
Upwind
linearUpwind
limitedLinear
Concentration
Position
0
0.5
1
1.5
2
0
20
40
60
80
100

Other options

`linearUpwind`	2nd order upwind (Warming and Beam 1976)using upwind interpolation weights with correction based on local cell gradient. Unbounded but better than `linear`. Need to specify discretisation of velocity gradient.
`limitedLinear`	2nd order, uses limiter function to avoid non-physical values. $\psi = 1$ strongest limiter; $\psi = 0 \rightarrow$ linear
`...V`	Schemes designed for vector fields – limiter functions calculated globally (across all components of the vector)

snGradSchemes

Surface normal gradients = gradient of quantity calculated at face centre (from cell centre quantities) projected normal to the face.

- Important in calculating Laplacian term (next slide), but also used elsewhere
- Affected by non-orthogonality of mesh

`orthogonal`	Simple 2nd order interpolation; no corrections for non-orthogonal mesh
`corrected`	Includes an explicit non-orthogonal correction term dependent on angle α
`limited corrected <psi>`	Stabilised scheme with coefficient ψ Generally $\psi = 0.33$ (stability) or $\psi = 0.5$ (accuracy)

UNIVERSITY OF EXETER

laplacianSchemes

In FVM, the laplacian derivative is specified as

$$\nabla.\Gamma\nabla(q)$$

where q is some physical quantity $(U,\ T)$ and Γ a diffusion coefficient $(\nu,\ \alpha)$. This means we have two interpolations to arrange

1. Interpolating Γ from cell centres to faces (possibly)
2. Interpolating gradients of q

Entries in `laplacianSchemes` thus have two entries, in this order. Eg.

```
    default Gauss linear corrected;
```

Matrix Inversion: Review

FVM converts individual equations into matrix equation of the form

$$\mathcal{M}x = y$$

with $\mathcal{M}$, y known matrix, source vector.

We invert $\mathcal{M}$ to advance solution one (1) computational step. Note this is a linearisation process.

$$x = \mathcal{M}^{-1}y$$

$\mathcal{M}$ is a large, sparse matrix of known form – use approximate, iterative methods to invert this.

Definitions

Matrix $\mathcal{M}$ is symmetric if it is equal to its transpose :

$$\mathcal{M} = \mathcal{M}^T$$

and antisymmetric if

$$\mathcal{M} = -\mathcal{M}^T$$

A matrix is said to be *positive definite* if, for any vector x, the product

$$x^T \mathcal{M} x > 0$$

Components on the leading diagonal $m_{i,i}$ prove to be quite important in determining the stability. Define a *diagonally dominant* matrix

$$|m_{i,i}| > \sum_{j \neq i} |m_{i,j}| \qquad \text{for any row } i.$$

Direct Solvers: Gauss-Seidel

Jacobi method : any particular line in the matrix equation can be written

$$\sum_{j=1}^{N} m_{i,j} x_j = q_j$$

If we keep the other entries in x constant, we can invert this;

$$x_i = \frac{q_i - \sum_{j \neq i} m_{i,j} x_j}{m_{i,i}}$$

This can be used as an iterative method, where the x_ith component is updated (store x_i^{k+1}). If we update x_i as we go along – *Gauss-Seidel* method

Smoothing and Roughening

Iterative methods which can be expressed in the form

$$x^{k+1} = \mathcal{B}x^k + c$$

where neither B nor c depend on the iteration, are referred to as stationary iterative methods.

The convergence of iterative methods closely related to the eigenvalues of $\mathcal{B}$. The magnitude of the largest eigenvalue is referred to as the spectral radius of the matrix.

From this, can show Jacobi, Gauss-Seidel

- Converge for diagonally dominant matrices
- Damp high frequency modes – act as *smoothing* step

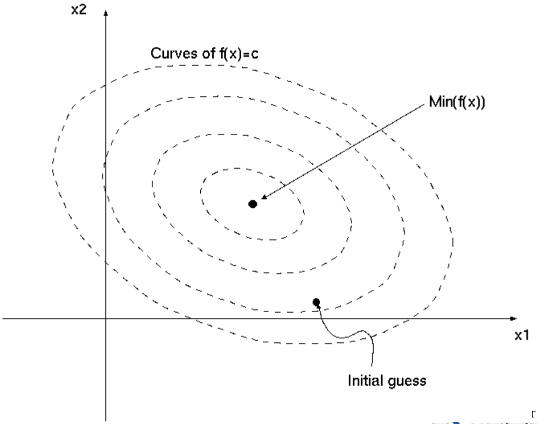
Conjugate Gradient

If $\mathcal{M}$ is symmetric and positive definite, we can examine

$$f(x) = \frac{1}{2}x^T\mathcal{M}x - x^Tq$$

Minimizing this gives us the solution

Close to the solution, $f(x)$ can be drawn as a set of ellipses



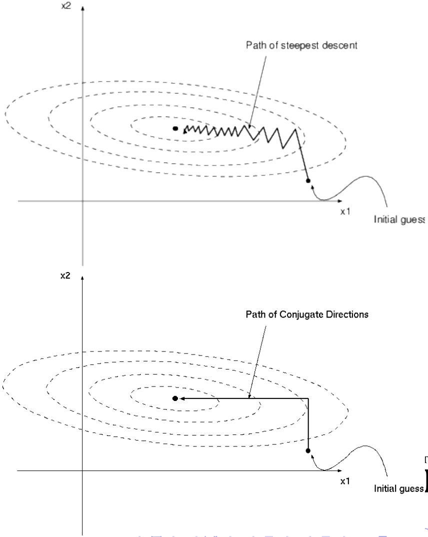
Conjugate Gradient cont

Steepest descent method would find the minimum – but fails for pathological cases

Better to go along x_2 to minimise in this direction, then along x_1 to minimise in that direction – method of *conjugate directions*

Conjugate Gradients method is a special case of this. Only applicable for *symmetric* matrices (but *bi-Conjugate gradient* method for antisymmetric matrices)

Both can be combined with *preconditioners* – multiply $\mathcal{M}$ by $\mathcal{P}$ to improve its numerical behaviour



Multigrid

Methods such as Gauss-Seidel :

- Good at smoothing out short wavelength errors; less good at longer wavelength errors
- Good at fixing solution locally; information propagated too slowly across domain

Unfortunately many equations in CFD (eg. pressure equation) are *elliptic* – solution at any point depends on all points in the domain.

Could speed up convergence on coarser mesh (but would not resolve finer structure)

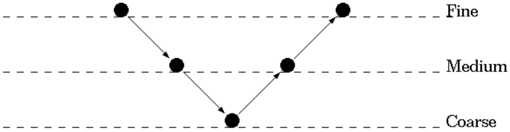
Solution – *Multigrid*. Construct sequence of meshes finer $\rightarrow$ coarser; alternate solution between different levels (e.g. finer $\rightarrow$ coarser $\rightarrow$ finer).

Construction of coarser meshes can be done algebraically

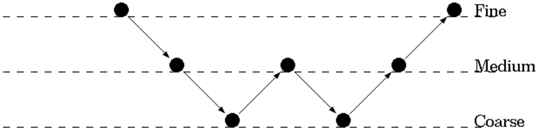
UNIVERSITY OF EXETER

V-cycle and W-cycle

a.



b.



Utilisation

OpenFOAM implements 3 types of solver;

- Solver with smoothing `smoothSolver`
- Preconditioned (bi-)conjugate gradient `PCG/PBiCG`
- Algebraic MultiGrid `GAMG`

Solvers distinguish between symmetric and asymmetric matrices – return error if incorrect.

Various control parameters must be specified; some solvers have additional options (eg. range of possible smoothers for `smoothSolver`

Runtime; solver prints out useful information:

```
smoothSolver: Solving for Ux, Initial residual = 0.00213492, Final residual = 0.000175659, No Iterations 8
smoothSolver: Solving for Uy, Initial residual = 0.0436924, Final residual = 0.00402266, No Iterations 7
smoothSolver: Solving for Uz, Initial residual = 0.0446746, Final residual = 0.00409408, No Iterations 7
GAMG: Solving for p, Initial residual = 0.0148994, Final residual = 6.73606e-05, No Iterations 4
```

Settings (general)

All `solvers` entries have some common elements :

`solver`	
`tolerance`	Cutoff tolerance for absolute residual
`relTol`	Tolerance for atio of final to initial residuals
`maxIter`	Maximum allowable number of iterations (defaults to 1000)

Solver stops iterating if *any* of these is satisfied.

Settings (Conjugate Gradient)

Conjugate Gradient/BiConjugate gradient methods have a selection of possible preconditioners available :

`DIC`	diagonal incomplete Cholesky for symmetric matrices; (paired with PCG)
`FDIC`	faster diagonal IC (uses caching)
`DILU`	incomplete LU preconditioner for asymmetric matrices (pair with PBiCG)
`diagonal`	diagonal preconditioning

Can also specify `none`

smoothSolver, GAMG options

Smoothsolvers involve a choice of smoother – related to preconditioner, so `DIC/DICU` are available, as well as `GaussSeidel` and `symGaussSeidel`, and combinations.

Can also specify `nSweeps` between recalculation of residual (defaults to 1)

`GAMG` options include the choice of smoother (as for `smoothSolver`) and a range of options for controlling the multigrid process; particularly the agglomeration strategy and number of sweeps of the smoother at different levels of refinement.

Comments – matrix inversion

Matrix inversion – equation solving. Typical pairing

- For speed : `GAMG` (pressure equation)
- For stability : `PCG` (pressure equation)
- `smoothsolver` (other equations)

Individual pass through solver should reduce residual by at least 1 (often more) orders of magnitude. Final target residual can be absolute or relative (to initial residual)

Pressure equation is usually most difficult. Failure to solve this (eg. too many iterations) often first sign of problems.

Lid Driven Cavity

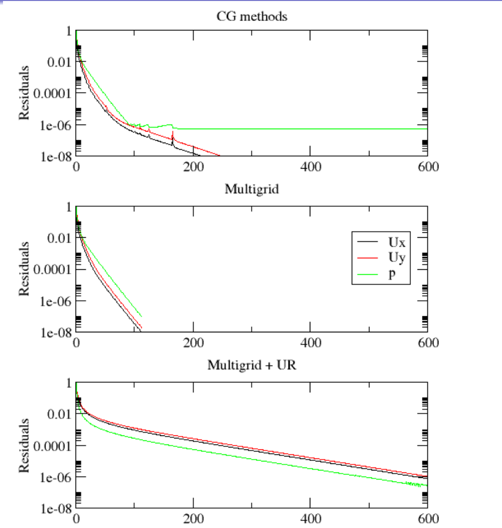
Modified from tutorials to run with `simpleFoam`, $Re = 10$.

3 different `fvSolution` files provided :

1. Conjugate Gradient solver for p
2. Multigrid solver for p
3. Lower underrelaxation parameters (0.3 for U) – shows effect of this in SIMPLE loop.

Uses `residuals` function object to output initial solver residuals for iteration – plot using xmgrace (or similar). (Could use `foamLog` instead).

Residuals


CG methods
Residuals
1
0.01
0.0001
1e-06
1e-08
0
200
400
600
Multigrid
Ux
Uy
p
Multigrid + UR

Conclusions

We have reviewed (most of) the content of `fvSchemes` and `fvSolution` – two very important control dictionaries.

Homework : try the examples, try some of the other options.

Thanks : Prof Hrv Jasak; my research group

Contact me : g.r.tabor@ex.ac.uk