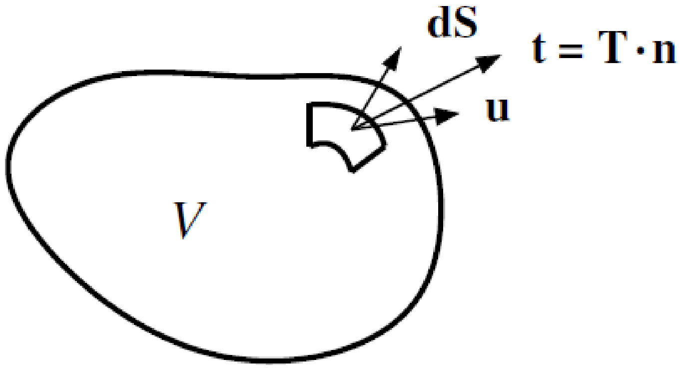
# *Conservation Equations of Compressible Flows*

(Integral Form)




Cauchy Stress Tensor: $\mathbf{T} = -p\mathbf{I} + \boldsymbol{\tau}$

Stokes: $\boldsymbol{\tau} = -\frac{2}{3}\mu(\nabla \cdot \mathbf{u})\mathbf{I} + \mu\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)$

Fourier heat flux: $\mathbf{q} = -\kappa\nabla T$

Mass:

$$\frac{\mathrm{d}}{\mathrm{dt}}\int_V \rho \mathrm{dV} + \int_S \rho\mathbf{u}\cdot\mathrm{d}\mathbf{S} = 0$$

Momentum (**f** body forces) :

$$\frac{\mathrm{d}}{\mathrm{dt}}\int_V \rho\mathbf{u}\mathrm{dV} + \int_S \rho\mathbf{u}(\mathbf{u}\cdot\mathrm{d}\mathbf{S}) = \int_V \rho\mathbf{f}\mathrm{dV} - \int_S \mathbf{T}\cdot\mathrm{d}\mathbf{S}$$

Energy (**Q** heat sources):

$$\frac{\mathrm{d}}{\mathrm{dt}}\int_V \rho E\mathrm{dV} + \int_S \rho E\mathbf{u}\cdot\mathrm{d}\mathbf{S} = \int_S \mathbf{u}\mathbf{T}\cdot\mathrm{d}\mathbf{S} + \int_V \rho\mathbf{f}\cdot\mathbf{u}\mathrm{dV} - \int_S \mathbf{q}\cdot\mathrm{d}\mathbf{S} + \int_V \rho Q\mathrm{dV}$$

# Navier-Stokes Eqns for Compressible Flows

(Differential Form)

Mass: $$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

Momentum: $$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot (\boldsymbol{\tau}) + \mathbf{f}$$

Energy: $$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho E \mathbf{u}) = -\nabla \cdot (p \mathbf{u}) + \nabla \cdot (\mathbf{u}\boldsymbol{\tau} - \mathbf{q}) + \rho(\mathbf{f} \cdot \mathbf{u}) + \rho Q$$

**Boundary Conditions:** *no-slip* on wall, ….

**Constitutive Laws:**

Stokes Viscous stress: $$\boldsymbol{\tau} = -\frac{2}{3}\mu(\nabla \cdot \mathbf{u})\mathbf{I} + \mu\left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right)$$

Total Energy: $$E = e + \frac{\mathbf{u}^2}{2}$$

Ideal gas: $$p = R\rho T$$

Polytropic gas: $$e = c_v T$$ constant specific heat

Fourier heat flux: $$\mathbf{q} = -\kappa \nabla T$$

# 4A2: Euler Equations in 2D: $\mu = 0$ and $\kappa = 0$

Mass:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \ ,$$

Momentum:

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho u v)}{\partial y} = -\frac{\partial p}{\partial x} \ ,$$

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho u v)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} = -\frac{\partial p}{\partial y} \ ,$$

Energy:

$$\frac{\partial(\rho E)}{\partial t} + \frac{\partial\left((\rho E + p)u\right)}{\partial x} + \frac{\partial\left((\rho E + p)v\right)}{\partial y} = 0 \ .$$

**Boundary Conditions:** *slip* on wall, ….

Ideal gas: $p = R\rho T$

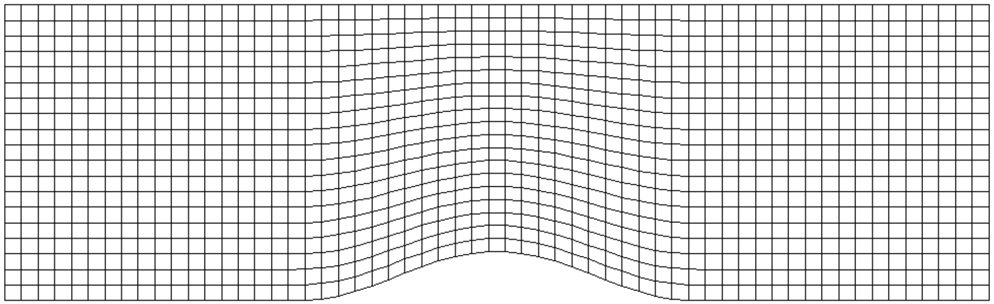
Polytropic gas: $e = c_v T$ constant specific heat

Speed of Sound: $a^2 = \dfrac{\gamma p}{\rho} = \gamma R T$

# What is CFD?

CFD is the art of replacing the integral or the differential derivatives in the governing equations with discetized algebraic forms.

First step: discretization of the physical domain



Test 0 : mesh for subsonic flow over bump

- Differential equation: finite-difference
- Integral equation : finite-volume
- Weak formulation : finite-element

- **Finite Difference** Methods (solving differential equations by approximating them with difference equations, in which finite differences approximate the derivatives). For example,

$$\frac{df}{dt} \approx \frac{f(t+\Delta t) - f(t)}{\Delta t}$$

- **Finite Volume** Methods (based on conservation laws. **Volume integrals** in a partial differential equation that contain a **divergence term** are converted to **surface integrals**, popular in Fluid Mechanics, especially with **hyperbolic** equations).

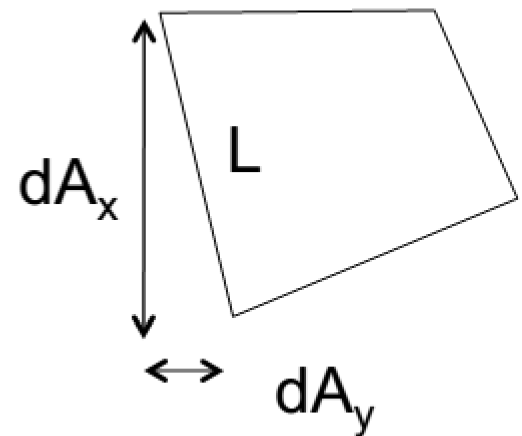
$$\int_{\Omega} \nabla \cdot \boldsymbol{q} dV = \int_{\partial\Omega} \boldsymbol{q} d\boldsymbol{S}$$

The method has the important physical property that certain **conservation laws** are maintained.

- **Finite Element methods are based on weak form of PDE, require a lot of maths (*module 3D7*)**

# Conservation Laws and Finite Volume method

$$Vol\frac{\Delta\rho[?]}{\Delta t}=\sum_{cvs}Flux$$




- Where for edge L:

$Flux_{mass}$ = $\dot{m}$ = ρ **V.dA** = $\rho V_x dA_x + \rho V_y dA_y$

$Flux_{mom}$ = $\dot{m}u$ = (ρ **V.dA**)**V**

or

$Flux_{x,mom} = (\rho V_x dA_x + \rho V_y dA_y) V_x + P\, dA_x$

$Flux_{y,mom} = (\rho V_x dA_x + \rho V_y dA_y) V_y + P\, dA_y$

Flux + Pressure Source
-> Strong Conservation Eqn.

# FINITE VOLUME METHOD

$$\text{Flux}_{\text{enthalpy}} = mh_o = (\rho\ \mathbf{V}\mathbf{.}d\mathbf{A})h_o$$

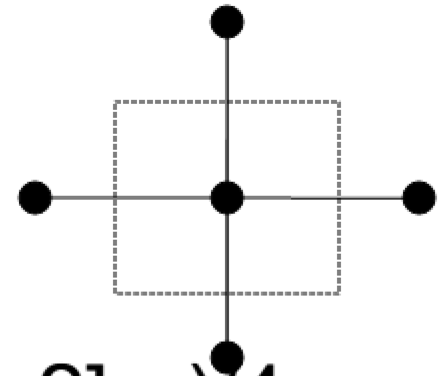
or

$$\text{Flux}_{\text{enthalpy}} = (\rho\ V_x dA_x + \rho\ V_y dA_y)h_o$$

For the RHS terms we have taken a differential eqn and solved it by summing around a control volume – this is the FINITE VOLUME METHOD

# SMOOTHING -> STABILITY

- Averaging surrounding data



$$[\rho ?]_i = ([\rho ?]_{i-1} + [\rho ?]_{i+1} + [\rho ?]_{j-1} + [\rho ?]_{j+1})/4 \qquad \text{(II)}$$

- Discrete equivalent of

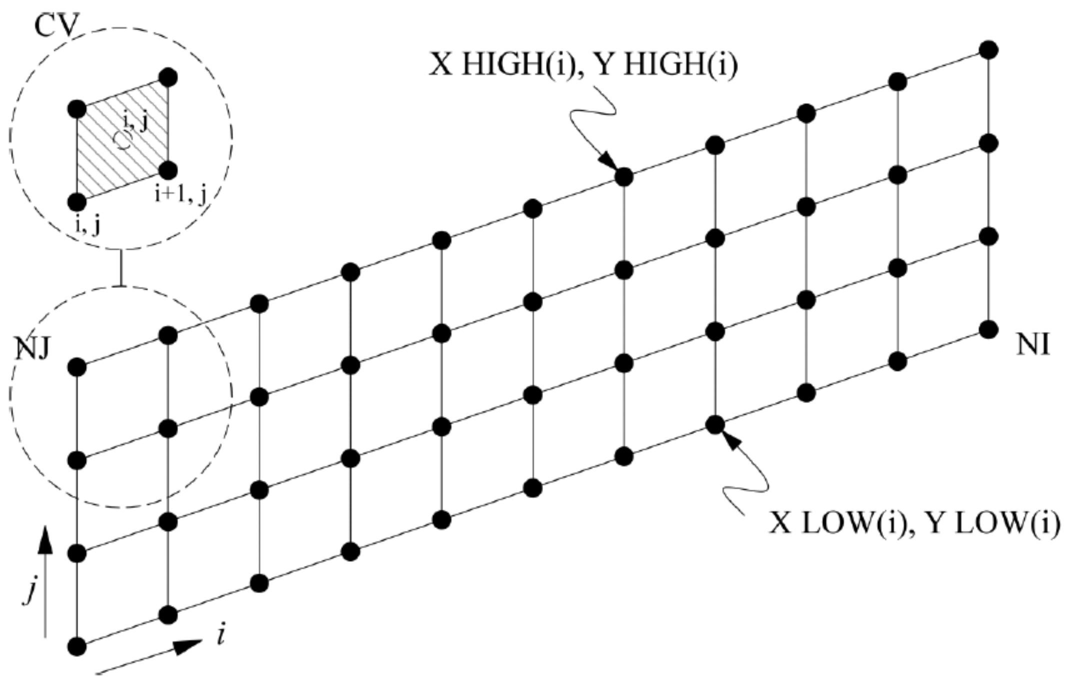
$$\nabla^2 [\rho ?] = \frac{\partial^2 [\rho ?]}{\partial x^2} + \frac{\partial^2 [\rho ?]}{\partial y^2}$$

# Writing the basic solver Pt 1

Tom Hynes

# Task at Hand

Solve for $\rho, \rho V_x, \rho V_y, \rho E$ where $E = c_v T + \frac{1}{2}V^2$



# Initial Calls

```
program euler
    call read_data
    call generate_grid
    call check_grid
    call crude_guess  !flow_guess
    call set_timestep
```

# Main Loop Calls

```
do nstep = 1,nsteps
  ro_start = ro
  call set_others
  call apply_bconds
  call set_fluxes
  call sum_fluxes
  ro = ro_start + ro_inc
  call smooth
  call check_conv
 end do
```

# Read Data

Need to be able to get information into our program – geometry and flow data

Geometry from file test0_geom, and flow from file test0_flow

Geometry (Define the shape of the computational domain):
NI, NJ, xhigh(NI), yhigh(NI), xlow(NI), ylow(NI)

Flow (Fluid properties and boundary conditions):
R, γ, CFL, NSTEPS, $P_{0,IN}$, $T_{0,IN}$, $P_{OUT}$, α

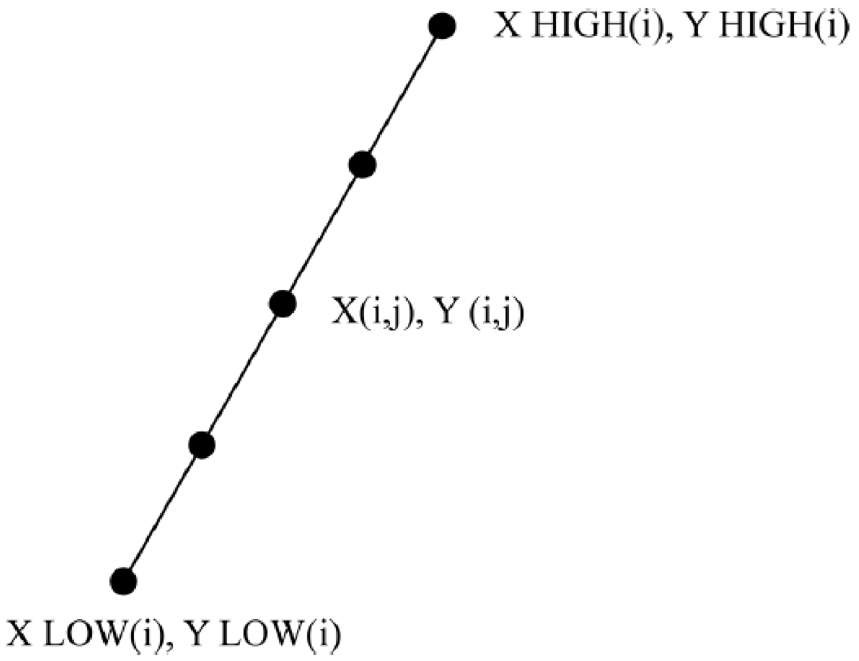
```
-bash-4.2$ more test0_flow
   287.1      1.4
   100000.    300.      00.0    85000.
   0.50       0.50
   3000      0.0001
```

Can use FORTRAN's free-format READ command:

```
READ(2,*) RGAS,GAMMA
READ(2,*) PSTAGIN,TSTAGIN,ALPHA1,PDOWN
```

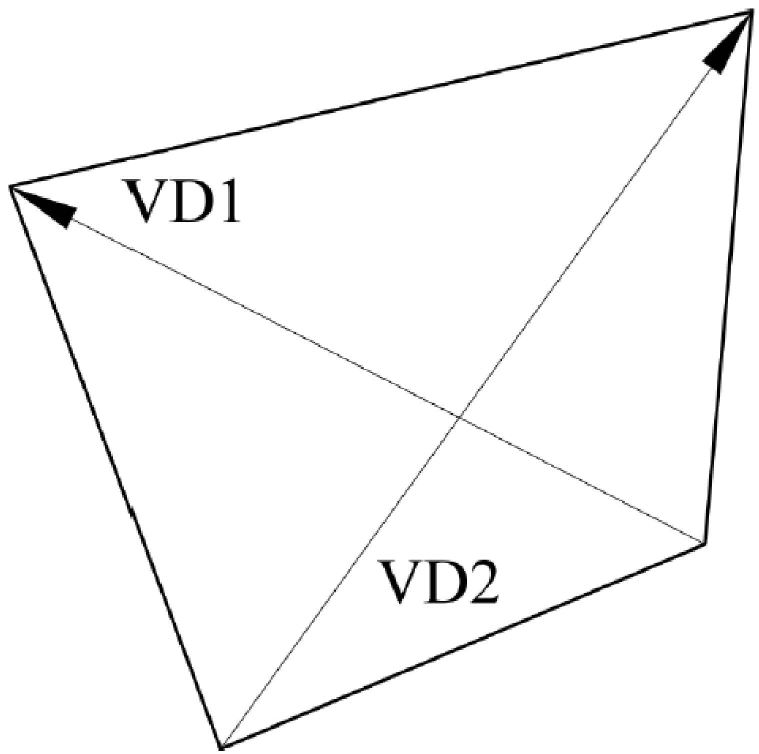
# Generate Grid

First thing `GENERATE_GRID` does is interpolate `X(NI,NJ)`, `Y(NI,NJ)` from `XHIGH(NI)`, `YHIGH(NI)`, `XLOW(NI)`, `XHIGH(NI)`



# Generate Grid

`GENERATE_GRID` also computes the volume (to unit depth) of the control volumes




$$\vec{A} = \frac{1}{2}\left(\overrightarrow{VD_2} \times \overrightarrow{VD_1}\right)$$

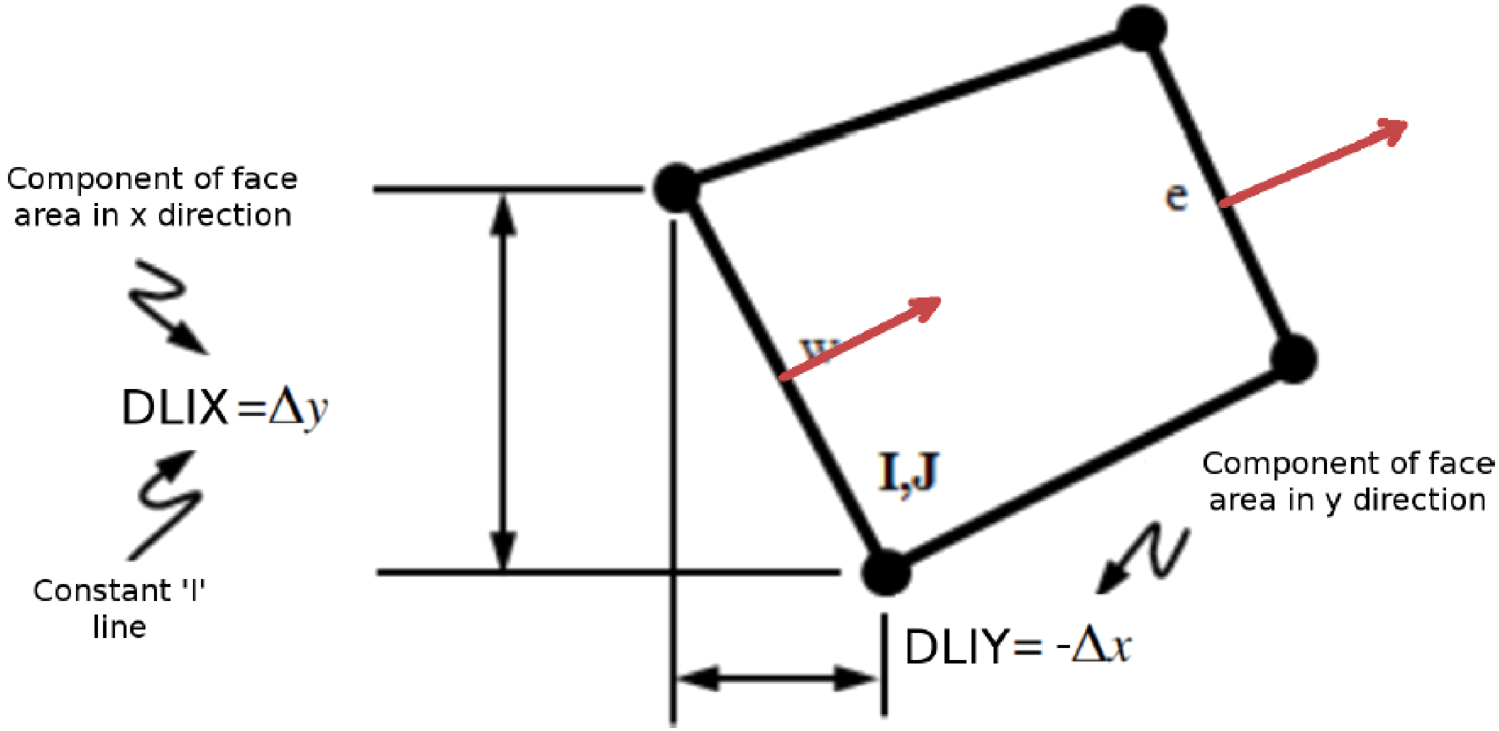
$$\overrightarrow{VD_1} = a_1 \vec{i} + b_1 \vec{j}$$
$$\overrightarrow{VD_2} = a_2 \vec{i} + b_2 \vec{j}$$

```
A1 = X(I,J+1) - X(I+1,J)
B1 = Y(I,J+1) - Y(I+1,J)

A2 = X(I+1,J+1) - X(I,J)
B2 = Y(I+1,J+1) - Y(I,J)
```

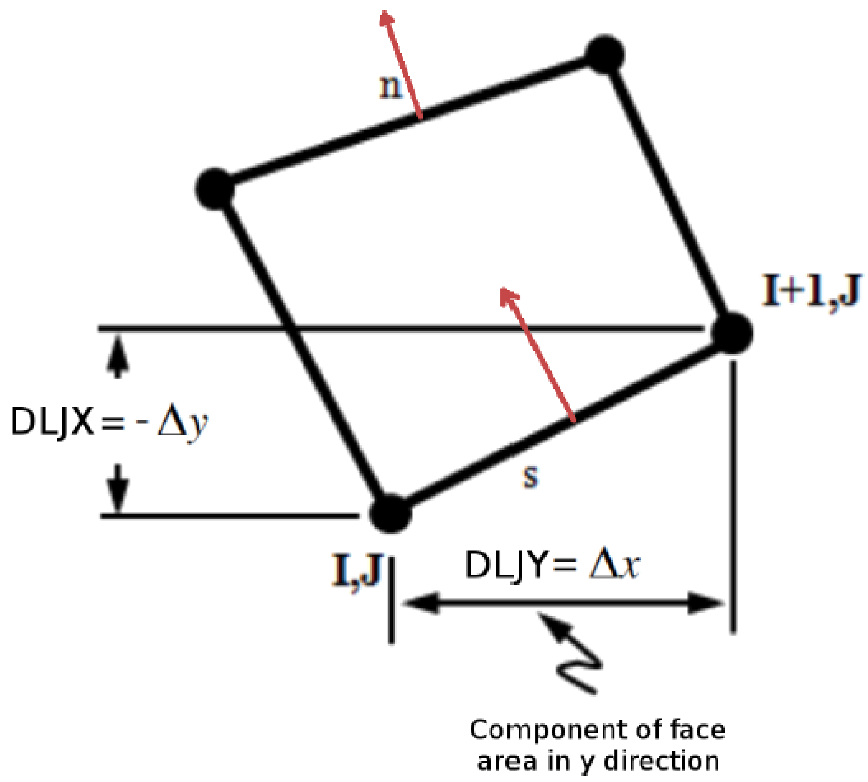
# Generate grid

GENERATE_GRID also computes the face areas (per unit depth) of the control volumes



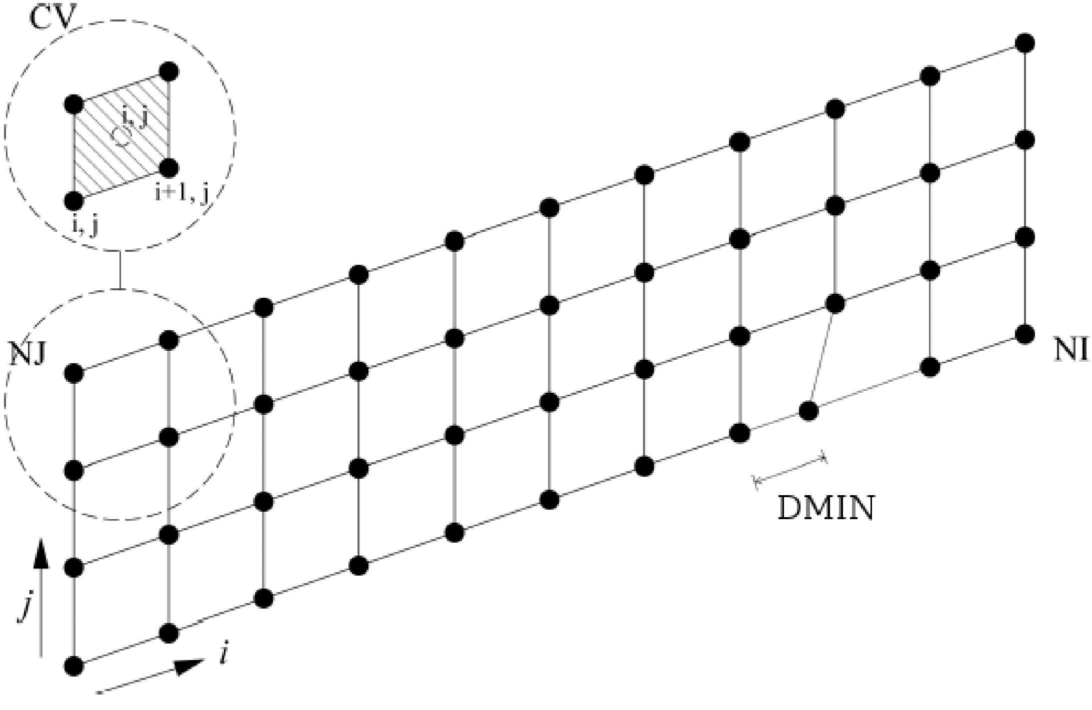
# Generate Grid

`GENERATE_GRID` also computes the face areas (per unit depth) of the control volumes



# Generate Grid

`GENERATE_GRID` also computes the minimum distance between any two grid points



# Check Grid

If the grid is wrong, nothing else will work properly!

Subroutine check_grid carries out some tests on your grid to make sure things are right

First – check all the areas (control volumes) are positive

Second – check vectorial sum of control volume faces is effectively zero

```
IF (abs(totx).GT.small_number) THEN
WRITE(6,*) 'X-Grid Not Zero! '
STOP
ENDIF
IF (abs(toty).GT.small_number) THEN
WRITE(6,*) 'Y-Grid Not Zero! '
STOP
ENDIF
```

# Set Time Step

The Courant-Freidrichs-Lewy (CFL) number is defined by information movement:

$$CFL = \frac{\Delta t |\Lambda_{max}|}{\Delta x_{min}}$$

Largest eigenvalue of flux jacobian:

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial \mathbf{Q}} \frac{\partial \mathbf{Q}}{\partial x} = 0$$

For the 2D Euler equations, these are:

$$|u|, |u|, |u+a|, |u-a|$$

Max eigenvalue is the forward running acoustic wave

# Set Time Step

Need to estimate the speed of the forward running acoustic wave:

$$|\Lambda_{max}| = |u + a|$$

For subsonic flows, can estimate u pessimistically as the speed of sound:

$$u = a = \sqrt{\gamma R T_0}$$

The CFL number is read from the flow file

Smallest grid spacing is calculated in `GENERATE_GRID` as variable `DMIN`

So:

```
A = SQRT(GAMMA * RGAS * TSTAGIN)
U = A
DELTA_T = CFL * DMIN / (U + A)
```

# Set Others

The `SET_OTHERS` subroutine calculates useful secondary variables from the four conserved ones: $\rho, \rho v_x, \rho v_y, \rho E \rightarrow v_x, v_y, p, h_0$

$$v_x = \frac{\rho v_x}{\rho}$$

$$v_y = \frac{\rho v_y}{\rho}$$

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho V^2\right)$$

$$h_0 = \frac{\rho E + p}{\rho}$$

There are many ways to compute these variables – this is just a suggestion

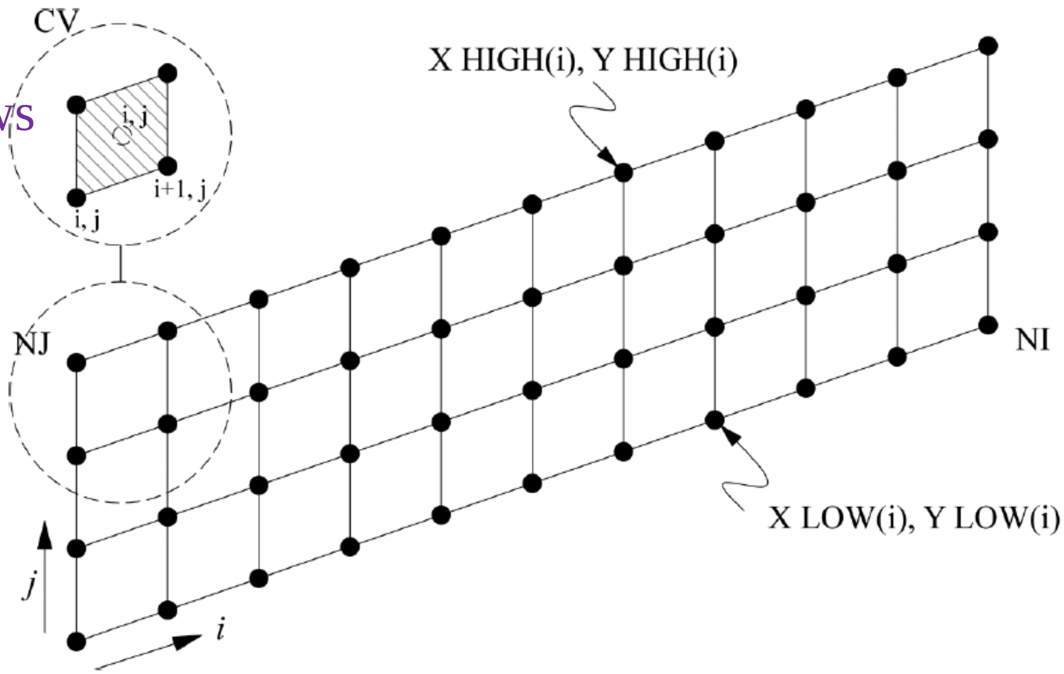
Time-Dependent Euler Solver for Compressible Flows

# Task at Hand

Solve for $\rho, \rho V_x, \rho V_y, \rho E$ where $E = c_v T + \frac{1}{2} V^2$

Conservative Variables defined at Cell Vertices

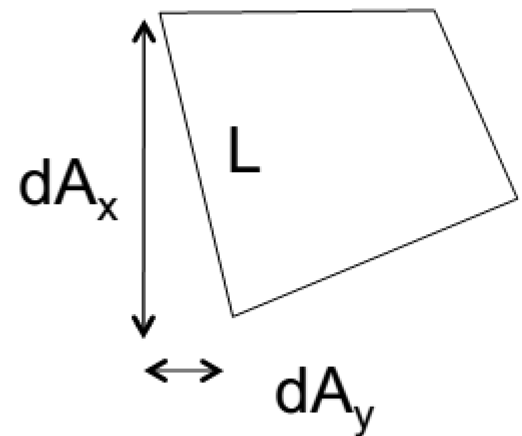
Conservation Laws Applied to CV




Get Your Hand "Dirty"

# Conservation Laws and Finite Volume method

$$Vol\frac{\Delta\rho[?]}{\Delta t} = \sum_{cvs} Flux$$




- Where for edge L:

$Flux_{mass} = m = \rho\, \mathbf{V.dA} = \rho\, V_x dA_x + \rho\, V_y dA_y$

$Flux_{mom} = mu = (\rho\, \mathbf{V.dA})\mathbf{V}$

or

$Flux_{x,mom} = (\rho\, V_x dA_x + \rho\, V_y dA_y)V_x + P\, dA_x$

$Flux_{y,mom} = (\rho\, V_x dA_x + \rho\, V_y dA_y)V_y + P\, dA_y$

Flux + Pressure Source
-> Strong Conservation Eqn.

# FINITE VOLUME METHOD

$$\text{Flux}_{\text{enthalpy}} = mh_o = (\rho\, \mathbf{V}.d\mathbf{A})h_o$$

or

$$\text{Flux}_{\text{enthalpy}} = (\rho\, V_x dA_x + \rho\, V_y dA_y)h_o$$

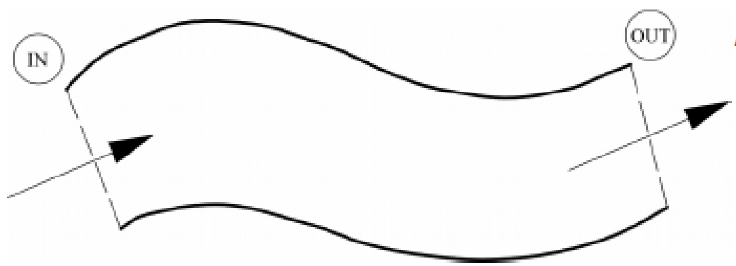
For the RHS terms we have taken a differential eqn and solved it by summing around a control volume – this is the FINITE VOLUME METHOD

# Apply Boundary Conditions

The `APPLY_BCONDS` subroutine enforces the specified boundary conditions at the inflow and outflow to the domain

Apply:

$$p_0, T_0, \alpha$$




Apply:

$$p$$

$$\rho_0 = {}^{p_0}\!/_{RT_0}$$
$$T = T_0\left({}^{\rho}\!/_{\rho_0}\right)^{\gamma-1}$$
$$E = c_v T + {}^{1}\!/_{2} V^2$$
$$V = \sqrt{2c_p\left(T_0 - T\right)}$$
$$V_x = V\cos\alpha$$
$$V_y = V\sin\alpha$$

$$\rho$$
$$\rho V_x$$
$$\rho V_y$$
$$\rho E$$

$$\rho = \alpha\rho^{new} + (1-\alpha)\,\rho^{old}$$
$$\rho < \rho_0$$

Wall boundary conditions are applied implicitly by preventing flux through surfaces where `J=1` or `J=NJ`

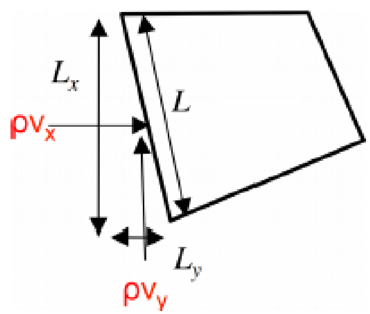
# Set Fluxes

The `SET_FLUXES` subroutine computes $\mathbf{F}_\phi \cdot \mathbf{n}$ for the four Euler equations

For i fluxes:

```
  FLUXI_PHI(I,J) = 0.5*(ROVX(I,J)+ROVX(I,J+1))*0.5*(PHI(I,J)+PHI(I,J+1))*DLIX(I,J) +
&                  0.5*(ROVY(I,J)+ROVY(I,J+1))*0.5*(PHI(I,J)+PHI(I,J+1))*DLIY(I,J) +
&                  0.5*(SOURCE(I,J)+SOURCE(I,J+1))
```

| Conserved Variable | Φ | Source Term |
|---|---|---:|
| Mass | 1 | 0 – no mass source |
| x-Momentum | $v_x$ | $\rho\delta x$ – pressure term |
| y-Momentum | $v_y$ | $\rho\delta y$ – pressure term |
| Energy | $h_0$ | 0 – no internal generation |




Similar expressions apply in j

# Sum Fluxes

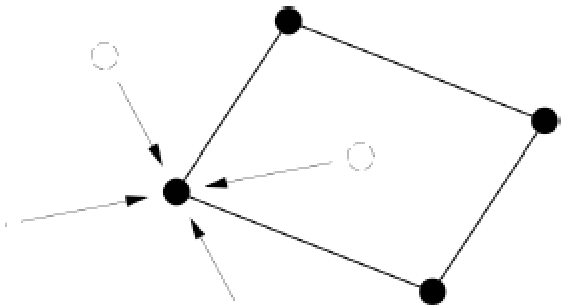
The `SUM_FLUXES` subroutine accumulates the fluxes into and out of each control volume, updates the solution, and scatters data to the vertices

$$\Delta\phi = \frac{\Delta t}{\Delta V} \sum_{CV} (\mathbf{F}_\phi \cdot \mathbf{n})$$

**del_prop(i,j) = (deltat/area(i,j))*(iflux(i,j) – iflux(i+1.j) +jflux(i,j)-jflux(i,j+1))**

# Sum Fluxes

**del_prop(i,j) = (deltat/area(i,j))*(iflux(i,j) – iflux(i+1.j) +jflux(i,j)-jflux(i,j+1))**



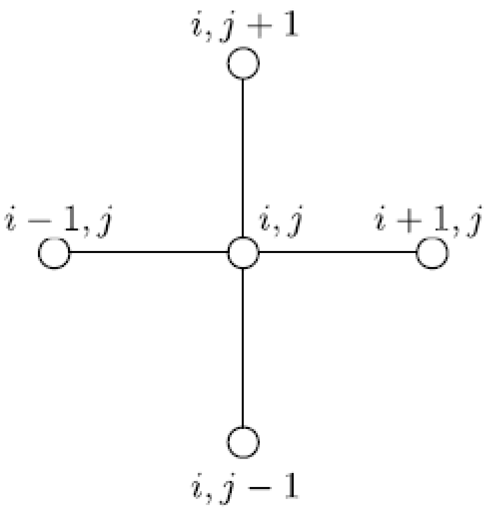
Solution stored at vertices, so the change at each is averaged from surrounding control volumes

**prop_inc(i,j) =**

Numeric Trick:
Variable Changes calculated at Cell Centres
Interpolate to Anywhere you want

# Smooth

The `SMOOTH` subroutine smooths the conserved variable fields using their averages from surrounding vertices




$$\phi_{averaged} = {}^{1}\!/_{4}\left(\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1}\right)$$

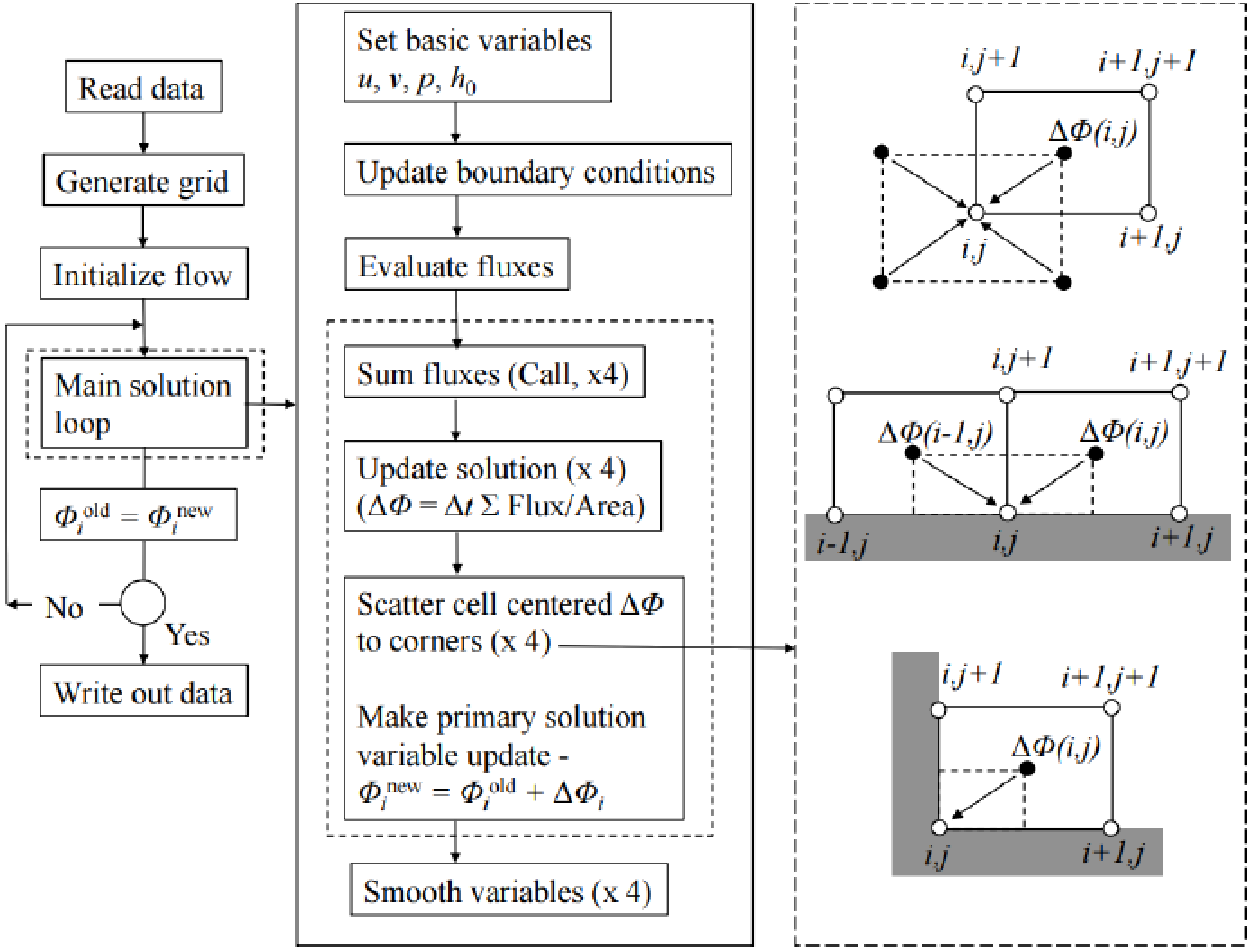
$$\phi_{smoothed} = \varepsilon\phi_{averaged} + (1-\varepsilon)\,\phi_{local}$$

$$\frac{\phi_{i+1} + \phi_{i-1}}{\Delta x^2} = \frac{\frac{\phi_{i+1}-\phi_i}{\Delta x} - \frac{\phi_i - \phi_{i-1}}{\Delta x}}{\Delta x} \approx \frac{\frac{\partial\phi}{\partial x}|_R - \frac{\partial\phi}{\partial x}|_L}{\Delta x} \approx \nabla^2\phi$$

Careful thought needed for smoothing behaviour near the boundaries

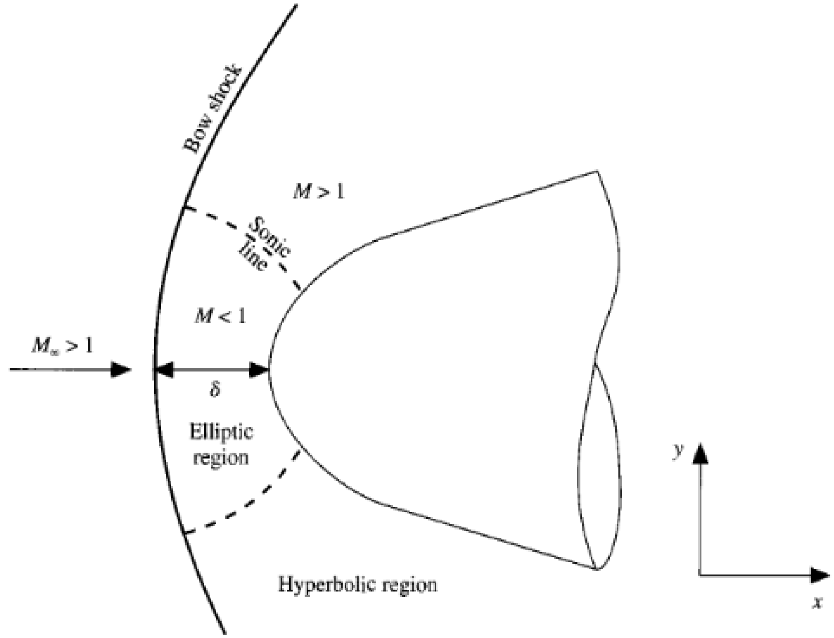
Equivalent to add a viscosity (diffusion) term
Artificial (numeric) viscosity

# Summary



Read data
Generate grid
Initialize flow
Main solution loop
$\Phi_i^{old} = \Phi_i^{new}$
No
Yes
Write out data
Set basic variables
$u, v, p, h_0$
Update boundary conditions
Evaluate fluxes
Sum fluxes (Call, x4)
Update solution (x 4)
($\Delta\Phi = \Delta t\, \Sigma$ Flux/Area)
Scatter cell centered $\Delta\Phi$ to corners (x 4)
Make primary solution variable update -
$\Phi_i^{new} = \Phi_i^{old} + \Delta\Phi_i$
Smooth variables (x 4)
i,j+1
i+1,j+1
$\Delta\Phi(i,j)$
i+1,j
i,j
i,j+1
i+1,j+1
$\Delta\Phi(i\text{-}1,j)$
$\Delta\Phi(i,j)$
i-1,j
i,j
i+1,j
i,j+1
i+1,j+1
$\Delta\Phi(i,j)$
i,j
i+1,j

# Steady Solutions & Time-Marching Methods



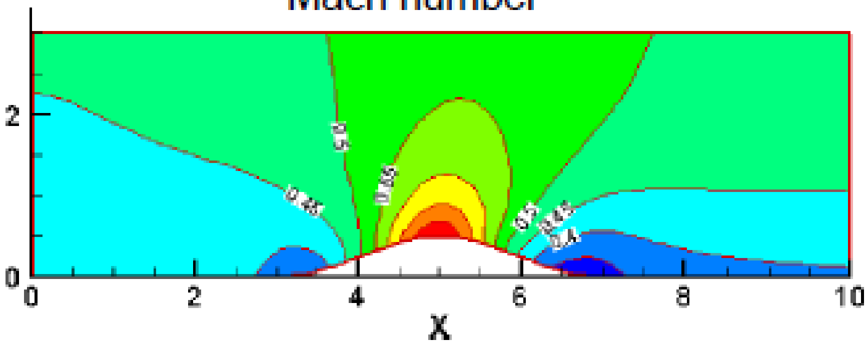

Sketch of the flow field over a supersonic blunt-nose body.

(Courtesy of Anderson, Computational Fluid Dynamics)

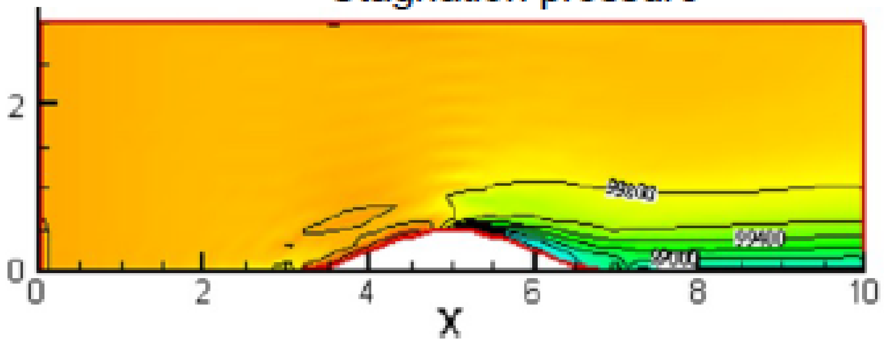
- First order in time
- Second order in space
- Numerical viscosity (smoothing)

# BASIC SOLVER

Mach number




Stagnation pressure




**Scope for enhancing:**

Converges in ~ 2600 iterations — Speed

Not symmetric, Stagnation pressure loss — Accuracy (due to smoothing)

Maximum stable time step ~ 1.6Δt (CFL = 0.5, SF = 0.5) — Stability

# Extensions: Enhanced Solver

**Time Accuracy (Class A)**
(enhances stability)

- Adams-Bashforth
- Runge-Kutta

**Space Accuracy (Class B)**

- Deferred Corrections
- High order smoothing
- High order differencing

**Speed (Class C)**
(easiest)

- Constant stagnation enthalpy
- Spatially varying time steps
- Residual averaging

Pick "*at least* " one of each class

Propose the *best combination* you have found

Use enhanced solver for *all test cases*

Previous students liked *Deferred Corrections + Runge-Kutta*

# ENHANCE STABILITY

**Adams-Bashforth** (Multi-Step method) – uses n, n-1 time levels

Taylor series expansion for $[?]_i^{n+1}$ in time:

$$[\;]_i^{n+1} = [\;]_i^n + \left.\frac{\partial[\;]}{\partial t}\right|_i^n \Delta t + \frac{1}{2}\left.\frac{\partial^2[\;]}{\partial t^2}\right|_i^n \Delta t^2 + O(\Delta t^3)$$

2nd derivative evaluated as one-sided difference in time

$$\left.\frac{\partial^2[\;]}{\partial t^2}\right|_i^n = \frac{\partial}{\partial t}\left(\frac{\partial[\;]}{\partial t}\right)_i^n = \frac{\left.\frac{\partial[\;]}{\partial t}\right|_i^n - \left.\frac{\partial[\;]}{\partial t}\right|_i^{n-1}}{\Delta t} + O(\Delta t)$$

Combine:

$$[\;]_i^{n+1} = [\;]_i^n + \left.\frac{\partial[\;]}{\partial t}\right|_i^n \Delta t + \frac{1}{2}\left(\frac{\left.\frac{\partial[\;]}{\partial t}\right|_i^n - \left.\frac{\partial[\;]}{\partial t}\right|_i^{n-1}}{\Delta t} + O(\Delta t)\right)\Delta t^2 + \boxed{O(\Delta t^3)}$$

**FACSEC**

0 → 1st order

0.5 → 2st order

> 0.5 → Further increases stability

Truncation error

# ENHANCE STABILITY

## **Adams-Bashforth** (Multi-Step method)

- Can be simplified to

$$[\;]_i^{n+1} = [\;]_i^n + \left( \underbrace{\boxed{\frac{3}{2}\frac{\partial[\;]}{\partial t}\bigg|_i^n}}_{1.5*\Sigma \mathrm{Flux}^n} - \underbrace{\boxed{\frac{1}{2}\frac{\partial[\;]}{\partial t}\bigg|_i^{n-1}}}_{-0.5*\Sigma \mathrm{Flux}^{n-1}} \right)\Delta t + O(\Delta t^3)$$

- Since not solving unsteady problems the 2nd order temporal accuracy will not give an accuracy benefit

- Modified Eqn analysis shows the method adds a diffusive error term in time (Vanishes at convergence)
  - This improves stability and much lower values of smoothing coefficients can be used

# ENHANCE STABILITY

**Runge-Kutta** (Multi-stage method) – uses only $n^{th}$ time level

Unlike multi-step, progress from time level 'n' to 'n+1' using 'm' sub-stages

$[?]^n$ = starting value of variable at time level 'n' i.e.

$$\boxed{[?]^0} = [?]^n$$

$$\boxed{[?]^1} = [?]^0 + \left.\frac{\partial[?]}{\partial t}\right|^0 \frac{1}{4}\Delta t \rightarrow \sum flux_0/4$$

$$\boxed{[?]^2} = [?]^0 + \left.\frac{\partial[?]}{\partial t}\right|^1 \frac{1}{3}\Delta t \rightarrow \sum flux_1/3$$

$$\boxed{[?]^3} = [?]^0 + \left.\frac{\partial[?]}{\partial t}\right|^2 \frac{1}{2}\Delta t \rightarrow \sum flux_2/2$$

$$[?]^4 = [?]^0 + \left.\frac{\partial[?]}{\partial t}\right|^3 \frac{1}{1}\Delta t \rightarrow \sum flux_3/1$$

$$[?]^{n+1} = [?]^4$$

Always add to starting value

Use new values to find changes in next sub-stage

- Significantly larger time steps
- More CPU time per iteration!
- Low values of artificial viscosity!
- Minimum complexity & memory

UNIVERSITY OF CAMBRIDGE

# ENHANCE ACCURACY

## Deferred Corrections

- In classic CFD form would involve say making a stable 1st order estimate of the solution and then adding on a correction based on a high order spatial discretization to yield Accuracy & Stability

- Estimate in basic code:

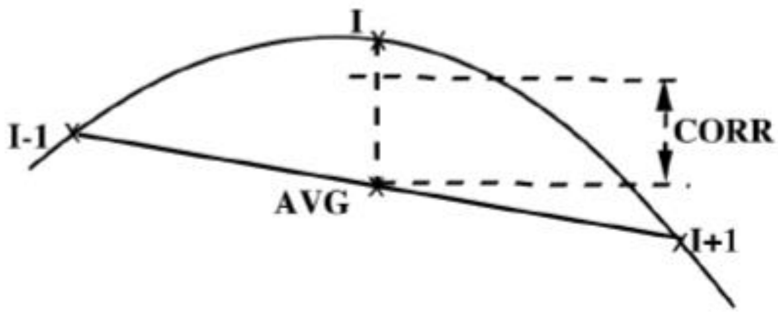
  $[?]_I = (1-\varepsilon)[?]_{unsmoothed} + \varepsilon[?]_{smooth}$ ← Too much smoothing

  $\varepsilon$ :smooth factor between 0 and 1

- Improve accuracy by adding the correction:

  $[?]_I = (1-\varepsilon)[?]_{unsmoothed} + \varepsilon([?]_{smooth} + \text{correction})$ ← Unstable

  $\text{correction} = [?]_{unsmoothed} - [?]_{smooth}$



# ENHANCE ACCURACY

## Deferred Corrections

- In classic CFD form would involve say making a stable 1st order estimate of the solution and then adding on a correction based on a high order spatial discretization to yield **Accuracy & Stability**

- Estimate in basic code:

  $[?]_l = (1-\varepsilon)[?]_{unsmoothed} + \varepsilon[?]_{smooth}$ ← Too much smoothing

  $\varepsilon$ :smooth factor between 0 and 1

- Improve accuracy by adding the correction:

  $[?]_l = (1-\varepsilon)[?]_{unsmoothed} + \varepsilon([?]_{smooth} + \text{correction})$ ← Unstable

  $\text{correction} = [?]_{unsmoothed} - [?]_{smooth}$

- Introduce correction gradually: (Initial 'grad_correction ' = 0)

  grad_correction = 0.99*grad_correction + 0.01*fcorr*correction

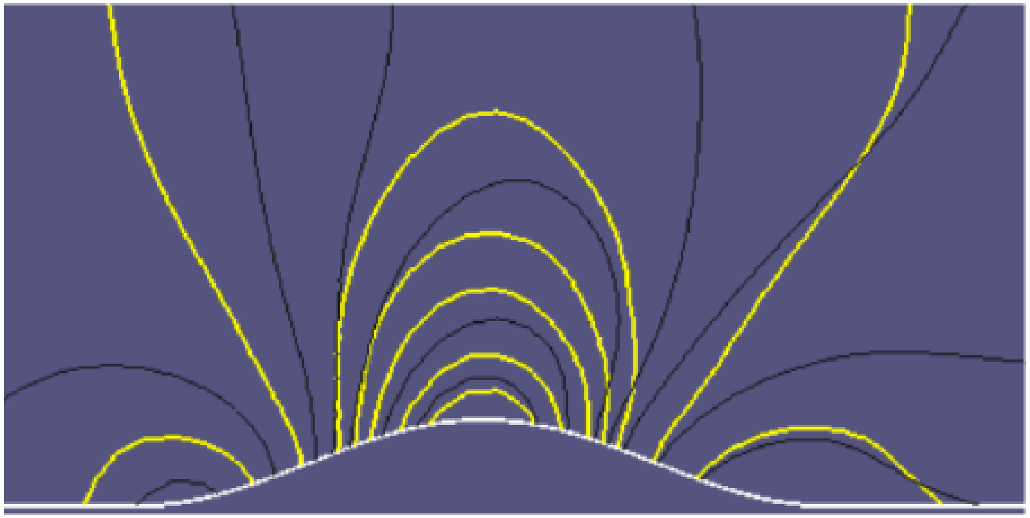
  fcorr: typical value 0.9

  $[?]_l = (1-\varepsilon)[?]_{unsmoothed} + \varepsilon([?]_{smooth} + \text{grad\_correction})$

# ENHANCE ACCURACY

## Deferred Corrections

Contours of M. Black = Basic LAX, yellow = DC.



Much more symmetric !

fcorr : proportion of artificial viscosity we want to cancel, typically about 0.9
Effective smooth_fac = (1-fcorr)*smooth_fac = 0.1 * smooth_fac

# ENHANCE ACCURACY

## **High-Order Smoothing** (Further minimize smoothing!)

- We have been explicitly adding as smoothing

$$\text{Smoothing} = \nabla^2[?] = \frac{\partial^2[?]}{\partial x^2} + \frac{\partial^2[?]}{\partial y^2} \quad \longleftarrow \quad 2^{nd} \text{ order smoothing}$$

- For greater accuracy we can use

$$\text{Smoothing} = \nabla^4[?] = \frac{\partial^4[?]}{\partial x^4} + \frac{\partial^4[?]}{\partial y^4} \quad \longleftarrow \quad 4^{th} \text{ order smoothing}$$

- Commercial CFD codes generally have

$$\text{Smoothing} = \varepsilon_1 f_1[?]\nabla^2[?] \quad + \quad \varepsilon_2 f_2[?]\nabla^4[?] \quad \longleftarrow \quad \text{Blend !}$$

Tricky at boundaries, Leave it $2^{nd}$ order !

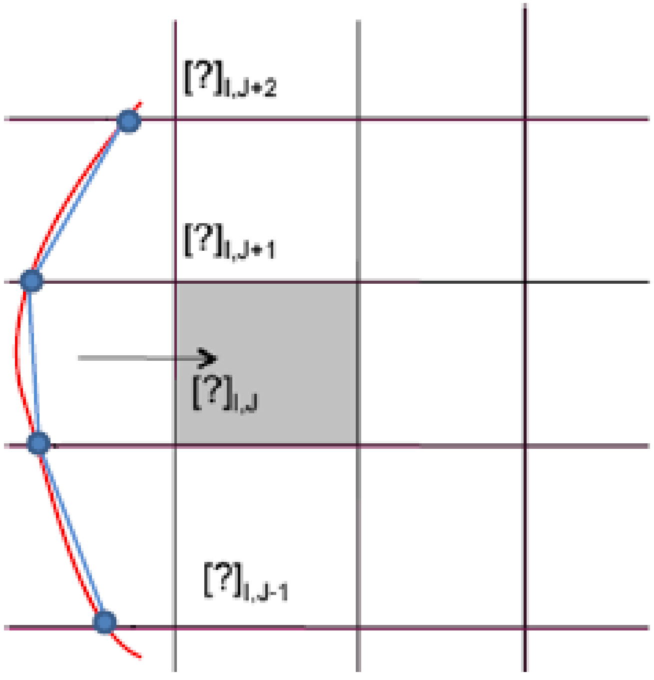
# ENHANCE ACCURACY

## High-Order Differencing

In basic code: Assumed properties to vary linearly between grid points

Construct high-order fluxes using:

- Finite difference formula
- Compact schemes
- Simple high-order interpolations




Eg: Cubic variation in interior domain
Parabolic variation at boundaries
Uniform grid spacing to simplify algebra

# ENHANCE SPEED

## Constant stagnation enthalpy

Assumptions: Flow is adiabatic, Not time accurate

Get rid of Energy Equation, set $h_o = c_p\, T_{o,in}$

## Spatially adaptive time-step

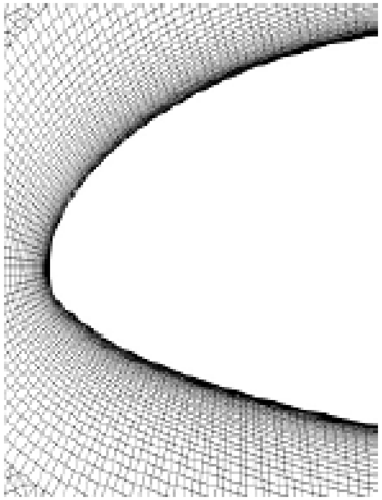
Different time steps for different elements

Not time accurate

Use local dmin, cell velocity and sound speed:

$$\Delta t = dmin_{loc}/(U+c)_{loc}$$

Any benefit? Limited benefit on uniform grids

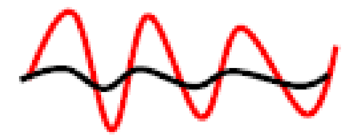


stretched grids

# ENHANCE SPEED

## Residual Averaging

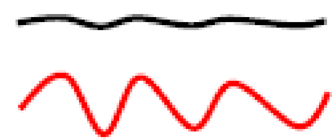
- Have smoothed solved for variables, Φ, for stability



$(n+1)^{th}$ time level

- Essentially solving  ΔΦ/Δt=Residual(=Σflux)
- At convergence Residual = 0. Hence, can smooth R – like did for Φ and not change answer.

$n^{th}$ time level



$(n+1)^{th}$ time level

- More stability allowing bigger Δt
- In practice, for convenience, we smooth ΔΦ (ΔΦ α R)

Any Benefit with default schemes? – Try with Runge Kutta !

# Flow Guess

The `FLOW_GUESS` subroutine improves on `CRUDE_GUESS` as an initial estimate of the flow

Calculate area of duct at each i-line

$$A(i) = \sqrt{\Delta y^2 + \Delta x^2}$$

Assuming isentropic flow, compute speed and density at outflow

$$T_{out} = T_{0,in} \left( {}^{p_{out}}/_{p_{0,in}} \right)^{\gamma - 1/\gamma}$$

$$\rho_{out} = {}^{p_{out}}/_{RT_{out}}$$

$$V_{out} = \sqrt{2c_p \left( T_{0,in} - T_{out} \right)}$$

Compute mass flow at outlet

$$m = \rho_{out} A_{out} V_{out}$$

Compute speed at each i-line based on outlet density

$$V(i) = {}^{m}/_{\rho_{out} A(i)}$$

Use speed to get a local (realisable) temperature

$$T(i) = T_{0,in} - {}^{V^2(i)}/_{2c_p}$$

$$T(i) \geq T_{lim}$$

# Flow Guess

The `FLOW_GUESS` subroutine improves on `CRUDE_GUESS` as an initial estimate of the flow

Update local density based on new temperature

$$p(i) = p_{0,in} \left( {T(i)}/{T_{0,in}} \right)^{\gamma/\gamma-1}$$

$$\rho(i) = {p(i)}/{RT(i)}$$

Use new density to update velocities

$$V(i) = {m}/{\rho(i) A(i)}$$

Resolve velocity into x and y

$$v_x = {V(i)\Delta x}/{\sqrt{\Delta x^2 + \Delta y^2}}$$

$$v_y = {V(i)\Delta y}/{\sqrt{\Delta x^2 + \Delta y^2}}$$

Form conserved variables from primitives

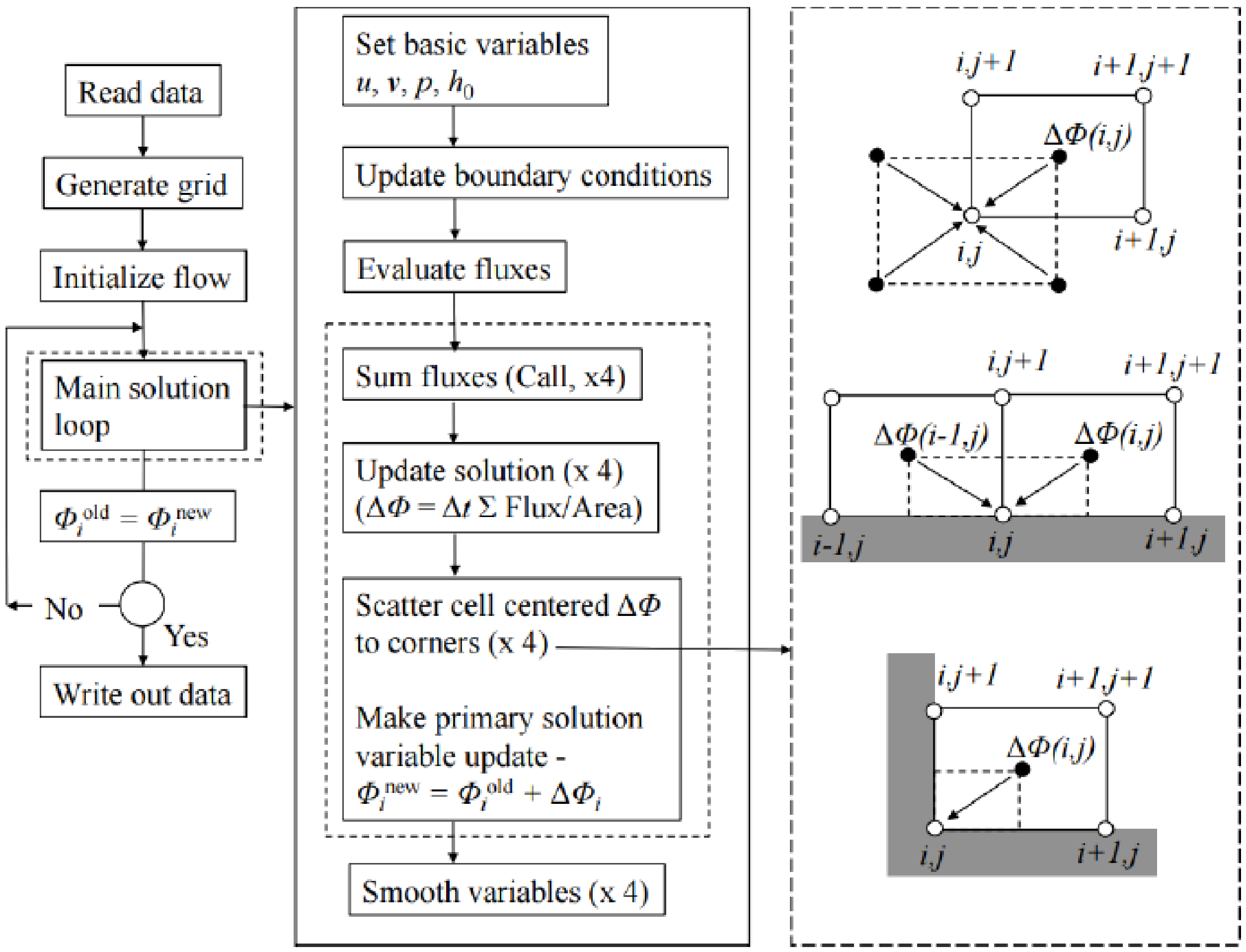
$$\rho$$

$$\rho v_x$$

$$\rho v_y$$

$$\rho E$$

# Summary


Read data
Generate grid
Initialize flow
Main solution loop
$\Phi_i^{old} = \Phi_i^{new}$
No
Yes
Write out data
Set basic variables $u, v, p, h_0$
Update boundary conditions
Evaluate fluxes
Sum fluxes (Call, x4)
Update solution (x 4) ($\Delta\Phi = \Delta t\ \Sigma$ Flux/Area)
Scatter cell centered $\Delta\Phi$ to corners (x 4)
Make primary solution variable update - $\Phi_i^{new} = \Phi_i^{old} + \Delta\Phi_i$
Smooth variables (x 4)
i,j+1
i+1,j+1
$\Delta\Phi(i,j)$
i,j
i+1,j
i,j+1
i+1,j+1
$\Delta\Phi(i-1,j)$
$\Delta\Phi(i,j)$
i-1,j
i,j
i+1,j
i,j+1
i+1,j+1
$\Delta\Phi(i,j)$
i,j
i+1,j