

THE FINITE VOLUME METHOD: A CRASH COURSE

www.tfd.chalmers.se

Lars Davidson, M2 Fluid Dynamics
Chalmers University of Technology
Gothenburg, Sweden

EQUATIONS

- Navier-Stokes for steady, incompressible, laminar flow (on *conservative form*)

$$\frac{\partial u_j}{\partial x_j} = 0 \quad (1)$$

$$\frac{\partial u_j u_j}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_j}{\partial x_j \partial x_j} \quad (2)$$

- The Poisson equation (the diffusion equation)

$$\nu \frac{\partial^2 u}{\partial x_j \partial x_j} = S \quad (3)$$

FINITE VOLUME SOLVERS

- **pyCALC-RANS** solves 2D Navier-Stokes equation. The grid may be curvi-linear. It is fully vectorized (no `for` loops). 1300 lines. [flow chart](#)
- **pyPoisson** solves the 2D Poisson equation. The grid may be curvi-linear. It is a stripped version of **pyCALC-RANS** . 450 lines.
- **pyPoisson-3D** is a simplified 3D Poisson solver on equidistant (i.e. constant Δx , Δy and Δz) Cartesian grids. It is fully vectorized (no `for` loops). 175 lines.
- **RANS0.py** is a 1D RANS solver for fully-developed turbulent channel flow (k-eps model)
 - DNS data for plotting can be downloaded [here](#)

NOMENCLATURE FOR A 2D GRID

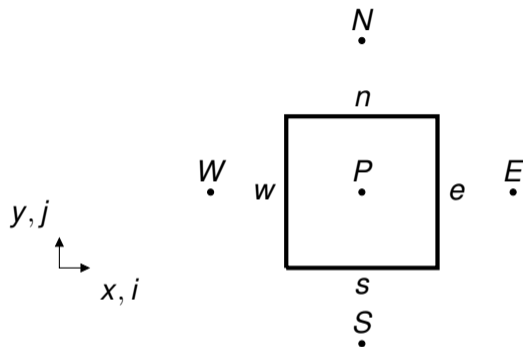


FIGURE: Control volume.

- A schematic 2D control volume grid is shown above
- Capital letters define nodes E(ast), W(est), N(orth) and S(outh), and small letters define faces of the control volumes.

1D POISSON EQUATION

$$\nu \frac{\partial^2 u}{\partial x^2} = S \quad (4)$$

1D GRID

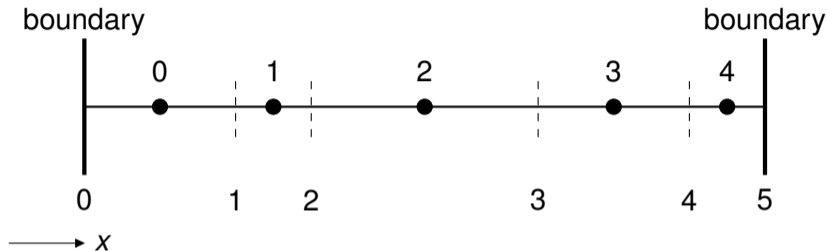


FIGURE: $n_i=5$. The bullets denote cell centers of the control volumes where the solution vector, u , is stored. They are labeled 0–4. Dashed lines denote control volume faces labeled 0–5.

- The size of the full coefficient matrix will be 5×5
- If I want to solve the matrix system with a Python sparse-matrix solver, the size of the solution vector must be 5×1
- That's the reason why the boundary values of u cannot be stored in the u array.
- The b.c. are implemented as source terms, see Slide 10.

1D POISSON EQUATION

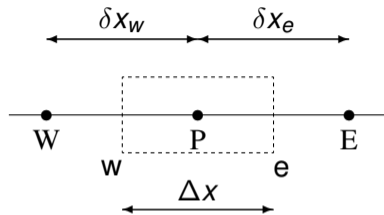


FIGURE: 1D control volume. Node P located in the middle of the control volume.

$$\int_w^e \left[\frac{d}{dx} \left(\Gamma \frac{du}{dx} \right) + S \right] dx = \left(\Gamma \frac{du}{dx} \right)_e - \left(\Gamma \frac{du}{dx} \right)_w + \bar{S} \Delta x = 0 \quad (5)$$

$$\left(\frac{du}{dx} \right)_e \simeq \frac{u_E - u_P}{\delta x_e}, \quad \left(\frac{du}{dx} \right)_w \simeq \frac{u_P - u_W}{\delta x_w} \quad (6)$$

$$\Gamma_w = f_x \Gamma_P + (1 - f_x) \Gamma_W \quad (7)$$

- $$f_x = \frac{|\overrightarrow{Ww}|}{|\overrightarrow{Pw}| + |\overrightarrow{Ww}|} \quad (8)$$

- $|\overrightarrow{Pw}|$ is the distance from P (the node) to w (the west face).
- In **pyCALC-RANS** and **pyPoisson** the interpolation factors (f_x, f_y) are stored in the Python arrays f_x and f_y .

THE DISCRETIZED EQUATION

- Insertion of Eq. 6 into Eq. 5 gives

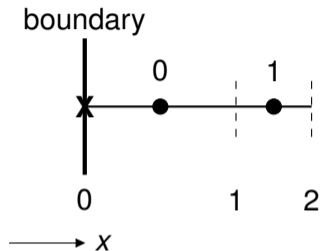
$$a_P u_P = a_E u_E + a_W u_W + S_U$$

$$a_E = \frac{\Gamma_e}{\delta x_e}, a_W = \frac{\Gamma_w}{\delta x_w}, S_U = \bar{S} \Delta x, a_P = a_E + a_W - S_P, S = S_U + S_P u_P \quad (9)$$

- The resulting sparse-matrix reads

$$\begin{bmatrix} & C0 & C1 & C2 & C3 & C4 \\ L0 : & a_{P,0} & -a_{E,0} & 0 & 0 & 0 \\ L1 : & -a_{W,1} & a_{P,1} & -a_{E,1} & 0 & 0 \\ L2 : & 0 & -a_{W,2} & a_{P,2} & -a_{E,2} & 0 \\ L3 : & 0 & 0 & -a_{W,3} & a_{P,3} & -a_{E,3} \\ L4 : & 0 & 0 & 0 & a_{W,4} & a_{P,4} \end{bmatrix}$$

BOUNDARY CONDITIONS AT $x = 0$



- For the u equation at cell $i = 0$ the discretized equation reads

$$a_{P,0}u_P = a_{W,0}^X u_W + a_{E,0}u_E + S_{U,0}, \quad a_{P,0} = a_{W,0}^X + a_{E,0} - S_{P,0}$$

- $u_W = u_{bc}$ is the b.c. (X above)
- But $a_{W,0}^X$ does not exist in the coefficient matrix (see previous slide)
- I set b.c. via $S_{U,0}$ and $S_{P,0}$ as $S_{U,0} = a_{W,0}^X u_{bc}$ and $S_{P,0} = -a_{W,0}^X$
- Note that if I don't set $S_{U,0}$ and $S_{P,0}$ then the b.c. is $du/dx = 0$ (hom. Neumann)

2D POISSON (DIFFUSION) EQUATION

$$\nu \frac{\partial^2 u}{\partial x^2} + \nu \frac{\partial^2 u}{\partial y^2} = S \quad (10)$$

2D POISSON (DIFFUSION) EQUATION

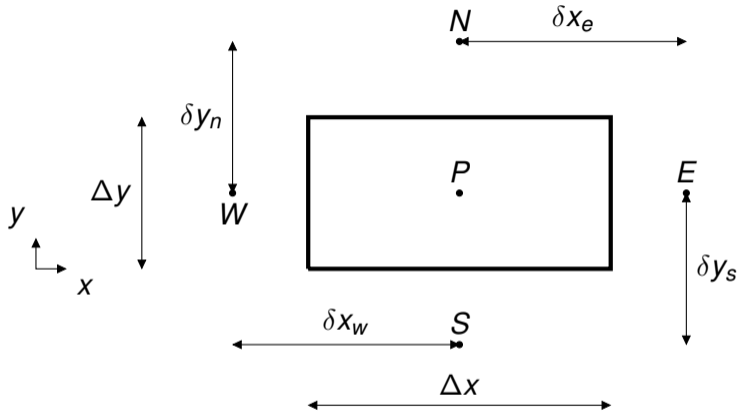


FIGURE: 2D control volume.

2D POISSON (DIFFUSION) EQUATION

$$\int_w^e \int_s^n \left[\frac{\partial}{\partial x} \left(\Gamma \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\Gamma \frac{\partial u}{\partial y} \right) + S \right] dx dy = 0.$$

- I start by the first term. The integration in x direction is carried out in exactly the same way as in 1D, i.e.

$$\begin{aligned} \int_w^e \int_s^n \left[\frac{\partial}{\partial x} \left(\Gamma \frac{\partial u}{\partial x} \right) \right] dx dy &= \int_s^n \left[\left(\Gamma \frac{\partial u}{\partial x} \right)_e - \left(\Gamma \frac{\partial u}{\partial x} \right)_w \right] dy \\ &= \int_s^n \left(\Gamma_e \frac{u_E - u_P}{\delta x_e} - \Gamma_w \frac{u_P - u_W}{\delta x_w} \right) dy \end{aligned}$$

2D POISSON (DIFFUSION) EQUATION

- Now integrate in the y direction. I do this by estimating the integral

$$\int_s^n f(y) dy = f_P \Delta y + \mathcal{O}((\Delta y)^2)$$

(i.e. f is taken at the mid-point P) which is second order accurate, since it is exact if f is a linear function. For our equation I get

$$\begin{aligned} & \int_s^n \left(\Gamma_e \frac{u_E - u_P}{\delta x_e} - \Gamma_w \frac{u_P - u_W}{\delta x_w} \right) dy \\ &= \left(\Gamma_e \frac{u_E - u_P}{\delta x_e} - \Gamma_w \frac{u_P - u_W}{\delta x_w} \right) \Delta y \end{aligned}$$

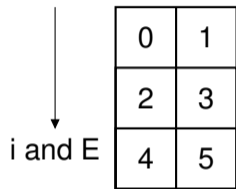
2D POISSON (DIFFUSION) EQUATION

- Rewriting it as an algebraic (i.e. the discretized) equation for u_P , I get

$$\begin{aligned} a_P u_P &= a_E u_E + a_W u_W + a_N u_N + a_S u_S + S_U & (11) \\ a_E &= \frac{\Gamma_e \Delta y}{\delta x_e}, \quad a_W = \frac{\Gamma_w \Delta y}{\delta x_w}, \quad a_N = \frac{\Gamma_n \Delta x}{\delta y_n}, \quad a_S = \frac{\Gamma_s \Delta x}{\delta y_s} \\ S_U &= \bar{S} \Delta x \Delta y, \quad a_P = a_E + a_W + a_N + a_S - S_P. \end{aligned}$$

MATRIX FOR 2D FLOW. $n_i \times n_j = (3, 2)$.

—————→ j and N



$$\begin{bmatrix}
 & C0 & C1 & C2 & C3 & C4 & C5 \\
 L0 : & a_{P,0} & -a_{N,0} & -a_{E,0} & 0 & 0 & 0 \\
 L1 : & -a_{S,1} & a_{P,1} & 0 & -a_{E,1} & 0 & 0 \\
 L2 : & -a_{W,2} & -a_{S,2} & a_{P,2} & -a_{N,2} & -a_{E,2} & 0 \\
 L3 : & 0 & -a_{W,3} & -a_{S,3} & a_{P,3} & 0 & a_{E,3} \\
 L4 : & 0 & 0 & -a_{W,4} & 0 & a_{P,4} & -a_{N,4} \\
 L5 : & 0 & 0 & 0 & -a_{W,5} & 0 & a_{P,5}
 \end{bmatrix}$$

MATRIX FOR 2D FLOW. $n_i \times n_j = (3, 4)$.

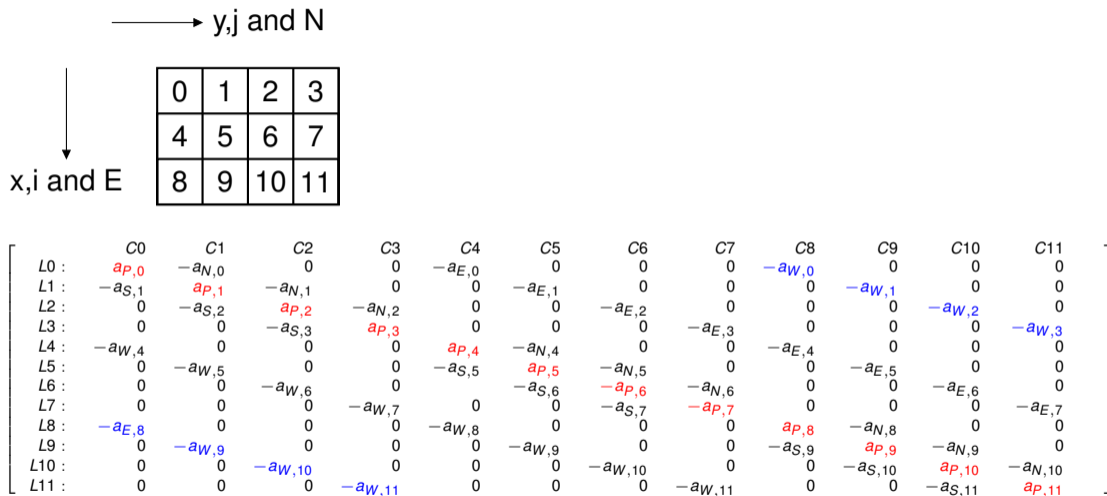


FIGURE: Cyclic in x, i . The coefficients due to cyclic boundary conditions are blue.

CONVECTION – DIFFUSION

- The 1D convection-diffusion equation reads (ϕ is, e.g., u , v , T , ...)

$$\frac{d}{dx}(u\phi) = \frac{d}{dx}\left(\Gamma \frac{d\phi}{dx}\right) + S$$

- I discretize this equation in the same way as the diffusion equation.

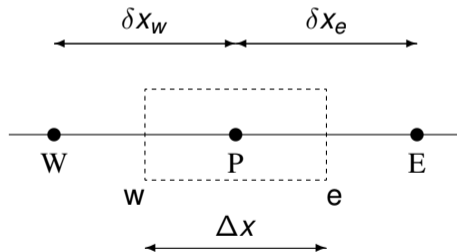


FIGURE: 1D control volume. Node P located in the middle of the control volume.

CONVECTION – DIFFUSION

- I start by integrating over the control volume

$$\int_w^e \frac{d}{dx} (u\phi) dx = \int_w^e \left[\frac{d}{dx} \left(\Gamma \frac{d\phi}{dx} \right) + S \right] dx. \quad (12)$$

- The convective term (the left-hand side)

$$\int_w^e \frac{d}{dx} (u\phi) dx = (u\phi)_e - (u\phi)_w = u_e\phi_e - u_w\phi_w$$

- I assume the velocity u to be known, or, rather, I take u from the previous iteration
- How to estimate ϕ_e and ϕ_w ? Linear interpolation (central differencing) gives

CONVECTION – DIFFUSION

$$\phi_w = f_x \phi_P + (1 - f_x) \phi_W, \quad u_w = f_x u_P + (1 - f_x) u_W$$

where f_x is the interpolation function (see Eq. 7, p. 8). Assuming $f_x = 0.5$, inserting the discretized diffusion and the convection terms into Eq. 12 I obtain

$$(u)_e \frac{\phi_E + \phi_P}{2} - (u)_w \frac{\phi_P + \phi_W}{2} = \frac{\Gamma_e(\phi_E - \phi_P)}{\delta x_e} - \frac{\Gamma_w(\phi_P - \phi_W)}{\delta x_w} + \bar{S} \Delta x$$

which can be rearranged as

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + S_U, \quad a_E = \frac{\Gamma_e}{\delta x_e} - \frac{1}{2}(u)_e, \quad a_W = \frac{\Gamma_w}{\delta x_w} + \frac{1}{2}(u)_w \quad (13)$$
$$S_U = \bar{S} \Delta x, \quad a_P = \frac{\Gamma_e}{\delta x_e} + \frac{1}{2}(u)_e + \frac{\Gamma_w}{\delta x_w} - \frac{1}{2}(u)_w$$

CONVECTION – DIFFUSION

- $a_P \geq a_E + a_W$ is the requirement to make sure that the iterative solver converges. Hence, I add the continuity equation

$$(u)_w - (u)_e = 0$$

to a_P so that (see Eq. 13)

$$a_P = a_E + a_W.$$

- Now you have learnt how to solve the Poission (diffusion) equation and the convection-diffussion equation. Next step is the Navier-Stokes equation, see Eq. 2.
- Navier-Stokes includes the pressure and hence we must find an equation for it.
- The pressure is obtained from the pressure-correction equation, p'

THE PRESSURE-CORRECTION (p') EQUATION

- The p' equation is obtained by applying the SIMPLEC algorithm [2]
- The mass flux \dot{m} is divided into an old value, \dot{m}^* , and a correction value, \dot{m}' as (in 1D)

$$\dot{m}_e = \dot{m}_e^* + \dot{m}'_e, \quad \dot{m}'_e = \left(\vec{A} \cdot \vec{u}' \right)_e = A_e u'_e \quad (14)$$

where u' is the correction velocity.

- The relation between u' and p' is obtained from the discretized Navier-Stokes

$$u' = -\frac{\Delta V_P}{a_p^u} \frac{\partial p'}{\partial x} \quad (15)$$

THE PRESSURE-CORRECTION (p') EQUATION

where ΔV_P = volume of the control volume. By introducing Eq. 14 into Eq. 15 we obtain

$$\dot{m}'_e = - \left[\frac{\Delta V_P}{a_p^u} \vec{A} \cdot \nabla p' \right]_e = - \frac{\Delta V_P}{a_p^u} A_e \left(\frac{\partial p'}{\partial x} \right)_e \quad (16)$$

THE PRESSURE-CORRECTION (p') EQUATION

- Consider, for simplicity, the continuity equation in one dimension

$$\dot{m}_e - \dot{m}_w = 0 \quad (17)$$

- If $\dot{m} = \dot{m}^* + \dot{m}'$ and Eq. 16 are substituted into eq. 17 we obtain

$$\left[\frac{\Delta V_P A_x}{a_p^u} \frac{\partial p'}{\partial x} \right]_w - \left[\frac{\Delta V_P A_x}{a_p^u} \frac{\partial p'}{\partial x} \right]_e + \dot{m}_e^* - \dot{m}_w^* = 0 \quad (18)$$

- This is a Poisson (diffusion) equation for the pressure correction p' which is discretized as Eq. (11) by replacing u by p' and setting $\Gamma = \Delta V_P / a_p$.
- The boundary conditions at all boundaries is homogeneous Neumann, i.e. $\partial p' / \partial x = 0$ at west and east boundaries and $\partial p' / \partial y = 0$ at south and north boundaries.

THE SOLUTION PROCEEDURE

- 1 Assign initial values (usually 10^{-10}) to the variable fields u^* , v^* and p^*
- 2 Solve the u -Navier-Stokes equation by first calculating the coefficients and sources, then setting b.c. followed by application of the Python solver.
- 3 Point 2 is repeated for v
- 4 Solve the p' equation by calculating the coefficients and sources, then setting b.c. followed by application of the Python solver.
- 5 Correct the velocity fields u^* , v^* and mass fluxes (see Eq. 14) \dot{m}_e^* and \dot{m}_n^* with u' , v' .
- 6 Correct the pressure field p^* with p' to give the correct pressure field p .
- 7 Go to step 2 and repeat step 2 to 7 until convergence.

You can find more details about discretization and the pressure correction method in [lecture notes](#) (Chapter 2-9).

PYCALC-RANS

There are four main case-specific routines in each subdirectory

- `generate*.py` (`generate-channel-grid.py`,
`generate-bound-layer-grid.py`, ...)
- `setup_case.py`
- `modify_case.py`
- `pl*.py` (`pl-uvw.py`, `plot_inlet.py`, ...)

The main program

- `pyCALC-RANS.py`

- Generate simple Cartesian meshes

- SECTION 1 choice of differencing scheme
- SECTION 2 turbulence models
- SECTION 3 restart/save
- SECTION 4 fluid properties
- SECTION 5 relaxation factors
- SECTION 6 number of iteration and convergence criteria
- SECTION 7 all variables are printed during the iteration at node
- SECTION 8 save data for post-processing
- SECTION 9 residual scaling parameters
- SECTION 10 boundary conditions

MODIFY_CASE.PY

```
def modify_init (u2d, v2d, w2d, k2d, om2d, eps2d, vis2d, dist) :  
def modify_inlet () :  
def modify_conv (convw, convs, convl) :  
def modify_u (su2d, sp2d) :  
def modify_v (su2d, sp2d) :  
def modify_k (su2d, sp2d) :  
def modify_om (su2d, sp2d) :  
def modify_outlet (convw) :  
def modify_vis (vis2d) :  
def def fix_omega () :
```

GLOBAL DECLARATIONS

They are made in the file `global` which reads

```
global c_omega_1, c_omega_2, cmu, ...  
x2d, xp2d, y2d, yp2d
```


CREATE EXECUTABLE

The bash script `run-python` is used which reads

```
#!/bin/bash # delete first line sed '/setup_case()/d'  
setup_case.py > temp_file # add new first line plus global  
declarations cat ../global temp_file modify_case.py  
../pyCALC-RANS.py > exec-pyCALC-RANS.py; python -u  
exec-pyCALC-RANS.py > out
```

- To run **pyCALC-RANS** (and **pyPoisson**), look at Section 10.3 in [1].

REFERENCES

- [1] L. Davidson. pyCALC-RANS: a 2D Python code for RANS . Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg
Download the code [here](#), 2021.
- [2] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill, New York, 1980.