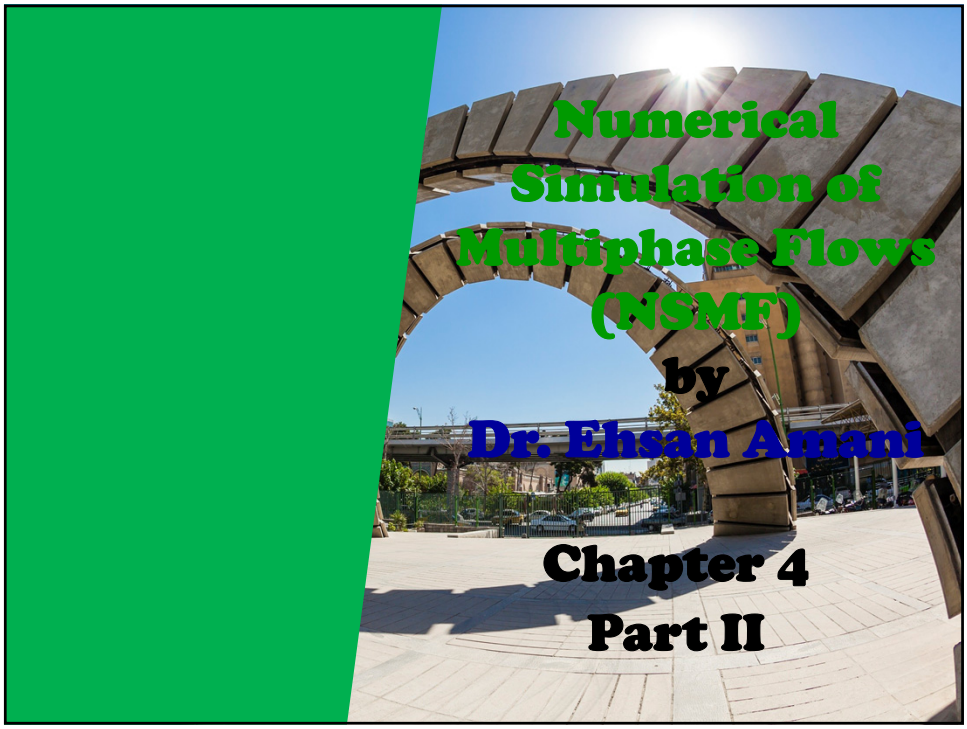


Challenge: Advecting a sharp interface

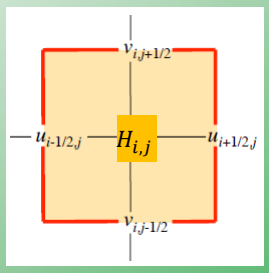
- **Step1: Advecting the indicator function**
  - **Second-order Central Difference (CD)**

$$\frac{\partial H}{\partial t} + \boldsymbol{\nabla}.(\boldsymbol{u}H) = 0$$

$$\frac{\partial}{\partial t}\int_V HdV = -\oint_S H(\boldsymbol{u}.\boldsymbol{n})dS$$

$$\frac{\partial}{\partial t}\left(H_{i,j}\Delta x\Delta y\right) = -\oint_S H(\boldsymbol{u}.\boldsymbol{n})dS$$

$$H_{i,j}^{n+1} = H_{i,j}^{n} - \frac{\Delta t}{\Delta x\Delta y}\oint_S H(\boldsymbol{u}.\boldsymbol{n})dS$$



More precisely, $\int_V HdV = C\Delta V$ and H should be replaced with C in FVM discretized equations, like in Ref. [1]. However, to maintain general FVM notation, we keep H in place of C.

Chap 4 By E. Amani

Challenge: Advecting a sharp interface

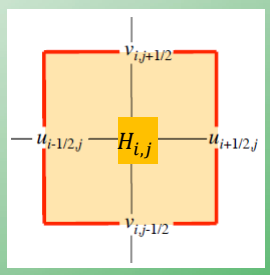
- **Step1: Advecting the indicator function**
  - **Second-order Central Difference (CD)**

$$\frac{\partial H}{\partial t} + \boldsymbol{\nabla}.(\boldsymbol{u}H) = 0$$

$$\frac{\partial}{\partial t}\int_V H dV = -\oint_S H(\boldsymbol{u}.\boldsymbol{n})dS$$

$$\frac{\partial}{\partial t}\left(H_{i,j}\Delta x\Delta y\right) = -\oint_S H(\boldsymbol{u}.\boldsymbol{n})dS$$

$$H_{i,j}^{n+1} = H_{i,j}^{n} - \frac{\Delta t}{\Delta x\Delta y}\sum u_{f,n}H_f S_f$$



$$H_{i,j}^{n+1} = H_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(u_{i+1/2,j}^{n}\frac{H_{i+1,j}^{n}+H_{i,j}^{n}}{2} - u_{i-1/2,j}^{n}\frac{H_{i-1,j}^{n}+H_{i,j}^{n}}{2}\right) - \frac{\Delta t}{\Delta y}\left(v_{i,j+1/2}^{n}\frac{H_{i,j+1}^{n}+H_{i,j}^{n}}{2} - v_{i,j-1/2}^{n}\frac{H_{i,j-1}^{n}+H_{i,j}^{n}}{2}\right)$$

Chap 4 By E. Amani

Coding: NSMF1.py

- **Step1: Advecting the indicator function**
  - **Second-order Central Difference (CD)**

$$H_{i,j}^{n+1} = H_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(u_{i+1/2,j}^{n}\frac{H_{i+1,j}^{n}+H_{i,j}^{n}}{2} - u_{i-1/2,j}^{n}\frac{H_{i-1,j}^{n}+H_{i,j}^{n}}{2}\right) - \frac{\Delta t}{\Delta y}\left(v_{i,j+1/2}^{n}\frac{H_{i,j+1}^{n}+H_{i,j}^{n}}{2} - v_{i,j-1/2}^{n}\frac{H_{i,j-1}^{n}+H_{i,j}^{n}}{2}\right)$$

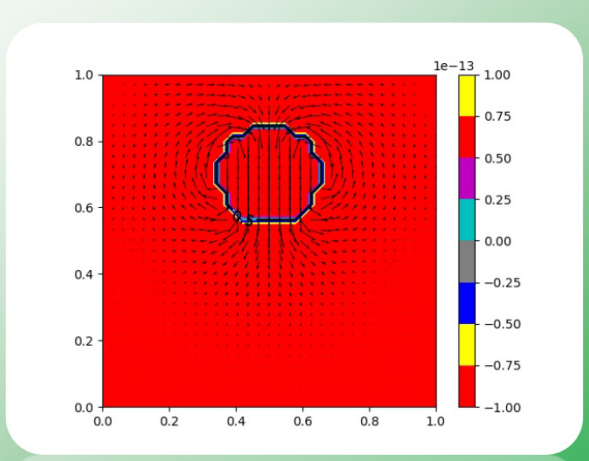
NSMF1.py × results.py ×

```
#---------------------- START TIME LOOP -----------------------
for istep in range(nstep):
    print(istep)

    #--- ADVECT marker using centered difference plus diffusion ----
    chio=chi.copy()

    if scalar_adv=='c':
        chi[2:nx+2,2:ny+2]=(chio[2:nx+2,2:ny+2]-(0.5*dt/dx)*(u[2:nx+2,2:ny+2]*(chio[3:nx+3,2:ny+2]
                    +chio[2:nx+2,2:ny+2]) - u[1:nx+1,2:ny+2]*(chio[1:nx+1,2:ny+2]+chio[2:nx+2,2:ny+2]))
                    -(0.5* dt/dy)*(v[2:nx+2,2:ny+2]*(chio[2:nx+2,3:ny+3]
                    +chio[2:nx+2,2:ny+2])-v[2:nx+2,1:ny+1]*(chio[2:nx+2,1:ny+1]+chio[2:nx+2,2:ny+2]) )
                    +(m0n*dt/dx/dx)*(chio[3:nx+3,2:ny+2]-2.0*chio[2:nx+2,2:ny+2]+chio[1:nx+1,2:ny+2])
                    +(m0n*dt/dy/dy)*(chio[2:nx+2,3:ny+3]-2.0*chio[2:nx+2,2:ny+2]+chio[2:nx+2,1:ny+1]))
```

Chap 4 By E. Amani

Coding: NSMF1.py

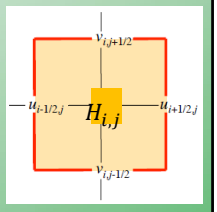
- **Step1: Advecting the indicator function**
  - **Second-order Central Difference (CD)**



Chap 4 **By E. Amani**

Coding: NSMF3.py

- **Step1: Advecting the indicator function**
  - **First-Order Upwind (FOU)**



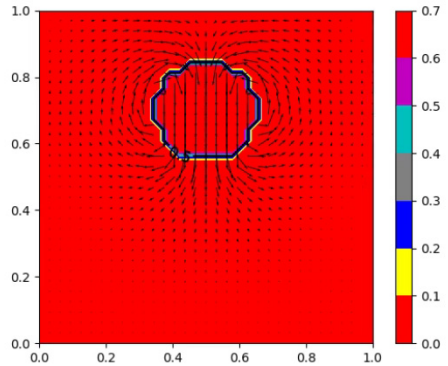
$$H_{i,j}^{n+1} = H_{i,j}^{n} - \frac{\Delta t}{\Delta x}\{[\max(u_{i+1/2,j}^{n}H_{i,j}^{n},0) - \max(-u_{i+1/2,j}^{n}H_{i+1,j}^{n},0)] - [\max(u_{i-1/2,j}^{n}H_{i-1,j}^{n},0) - \max(-u_{i-1/2,j}^{n}H_{i,j}^{n},0)]\} - \frac{\Delta t}{\Delta y}\{[\max(v_{i,j+1/2}^{n}H_{i,j}^{n},0) - \max(-v_{i,j+1/2}^{n}H_{i,j+1}^{n},0)] - [\max(v_{i,j-1/2}^{n}H_{i,j}^{n},0) - \max(-v_{i,j-1/2}^{n}H_{i,j}^{n},0)]\}$$

```
                    +(m0n*dt/dy/dy)*(chio[2:nx+2,3:ny+3]-2.0*chio[2:nx+2,2:ny+2]+chio[2:nx+2,1:ny+1]))
    elif scalar_adv=='u':
        chi[2:nx+2,2:ny+2]=(chio[2:nx+2,2:ny+2]
                    -(dt/dx)*(
                    (np.maximum(u[2:nx+2,2:ny+2]*chio[2:nx+2,2:ny+2],0)-np.maximum(-u[2:nx+2,2:ny+2]*chio[3:nx+3,2:ny+2],0))
                    -
                    (np.maximum(u[1:nx+1,2:ny+2]*chio[1:nx+1,2:ny+2],0)-np.maximum(-u[1:nx+1,2:ny+2]*chio[2:nx+2,2:ny+2],0)))
                    -(dt/dy)*(
                    (np.maximum(v[2:nx+2,2:ny+2]*chio[2:nx+2,2:ny+2],0)-np.maximum(-v[2:nx+2,2:ny+2]*chio[2:nx+2,3:ny+3],0))
                    -
                    (np.maximum(v[2:nx+2,1:ny+1]*chio[2:nx+2,1:ny+1],0)-np.maximum(-v[2:nx+2,1:ny+1]*chio[2:nx+2,2:ny+2],0))))
    else:
```

Chap 4 **By E. Amani**

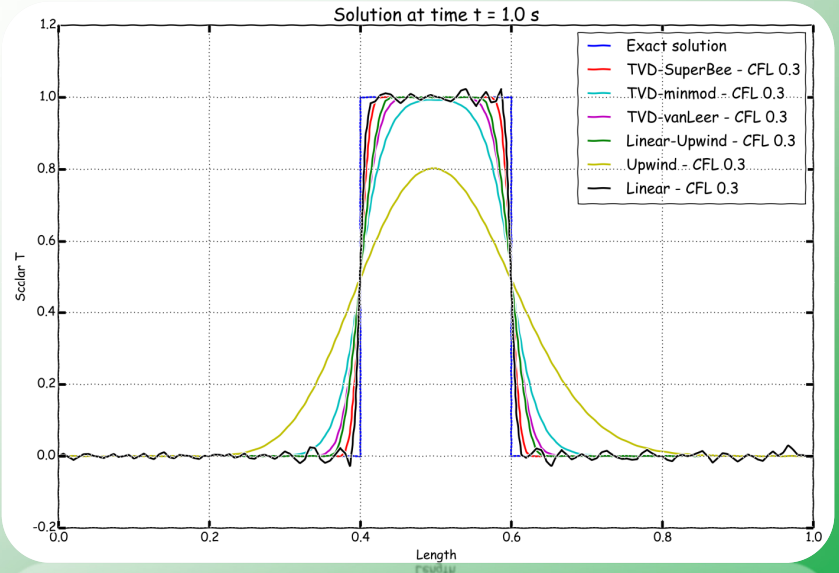
Coding: NSMF1.py

- **Step1: Advecting the indicator function**
  - **First-Order Upwind (FOU)**



Chap 4 By E. Amani

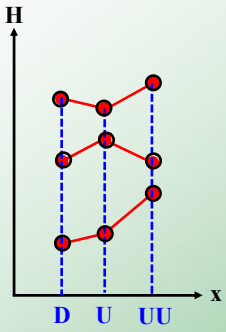
Coding: 1D advection test

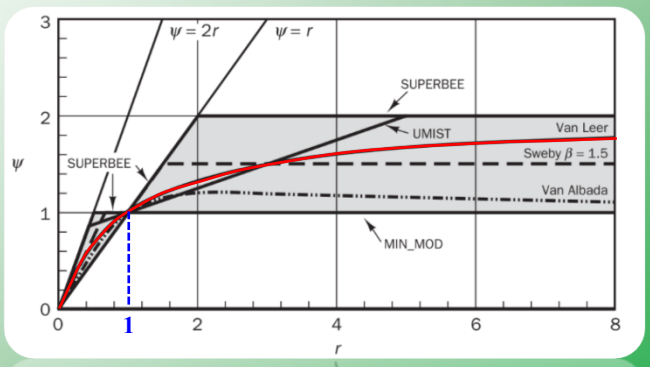


Chap 4 By E. Amani

Conventional flux limiters

- **Step1: Advecting the indicator function**
  - **Non-linear flux limiters**





Lecture Notes: IV.4

Chap 4 By E. Amani

Coding: NSMF1.py

- **Step1: Advecting the indicator function**
  - **Non-linear flux limiter: vanLeer**

$$H_{i,j}^{n+1} = H_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left\{u_{i+\frac{1}{2},j}^{n}\left[H_{i,j}^{n} + \frac{\left(H_{i+1,j}^{n} - H_{i,j}^{n}\right)}{2}\left\{1 - \xi_{i+\frac{1}{2},j}\left[1 - \psi\left(r_{i+\frac{1}{2}}\right)\right]\right\}\right] - u_{i-\frac{1}{2},j}^{n}\left[H_{i-1,j}^{n} + \frac{\left(H_{i,j}^{n} - H_{i-1,j}^{n}\right)}{2}\left\{1 - \xi_{i-\frac{1}{2},j}\left[1 - \psi\left(r_{i-\frac{1}{2}}\right)\right]\right\}\right]\right\} - \cdots$$

```
106         else:
107             kesiu=2*(u>=0)-1
108             kesiv=2*(v>=0)-1
109             chi[2:nx+2,2:ny+2]=(chio[2:nx+2,2:ny+2]
110                                 -(dt/dx)*(
111                                 u[2:nx+2,2:ny+2]*(chio[2:nx+2,2:ny+2]+(chio[3:nx+3,2:ny+2]-chio[2:nx+2,2:ny+2])*
112                                                 (1-kesiu[2:nx+2,2:ny+2]*(1-limiterf(rx(kesiu,chio,2,nx+2,2,ny+2))))/2)
113                                 -
114                                 u[1:nx+1,2:ny+2]*(chio[1:nx+1,2:ny+2]+(chio[2:nx+2,2:ny+2]-chio[1:nx+1,2:ny+2])*
```

Chap 4 By E. Amani

Coding: NSMF1.py

- **Step1: Advecting the indicator function**
  - **Non-linear flux limiter: vanLeer**

$$\psi(r) = \frac{r + |r|}{1 + |r|}$$

$$r^n_{x,i+1/2} = \frac{\frac{\left(1+\xi_{i+\frac{1}{2},j}\right)}{2}\left(H^n_{i,j} - H^n_{i-1,j}\right) + \frac{\left(1-\xi_{i+\frac{1}{2},j}\right)}{2}\left(H^n_{i+2,j} - H^n_{i+1,j}\right)}{\left(H^n_{i+1,j} - H^n_{i,j}\right) + \varepsilon}$$

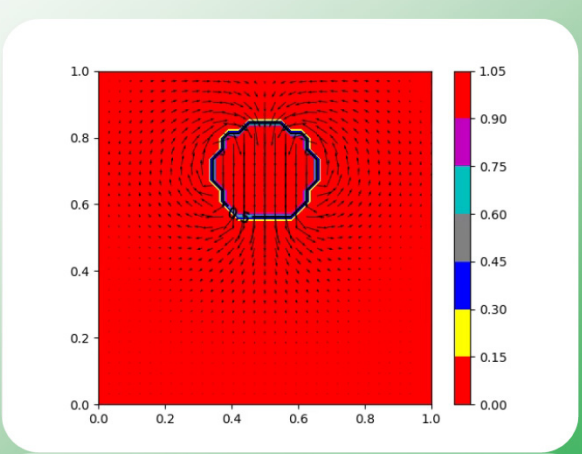
E:\doc-desktop\Codes\NSMF\final\NSMF1.py

NSMF1.py × results.py ×

```
37
38      #-------------------- Limiter definition ----------------------
39      def limiterf(rf):
40          return (rf+abs(rf))/(1+abs(rf)) #van Leer limiter
41          #return 1 #central
42          #return 0 #upwinf
43
44      def rx(kesiu,phi,i0,ie,j0,je):
45          return (((1+kesiu[i0:ie,j0:je])/2*(phi[i0:ie,j0:je]-phi[i0-1:ie-1,j0:je])+
46                   (1-kesiu[i0:ie,j0:je])/2*(phi[i0+2:ie+2,j0:je]-phi[i0+1:ie+1,j0:je]))/
47                   (phi[i0+1:ie+1,j0:je]-phi[i0:ie,j0:je]+smallVal))
```


Coding: NSMF1.py

- **Step1: Advecting the indicator function**
  - **Non-linear flux limiter: vanLeer**



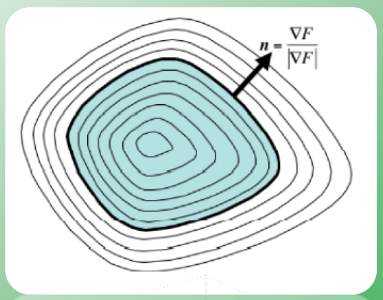
Sharp interface advection methods

- **Specific advection methods for sharp interfaces**
  - **Volume Of Fluid (VOF)**

$$C = \langle \chi \rangle = \langle H \rangle = \alpha$$ → *Volume fraction or color function*

  - **Level Set (LS)**

$$H = H(\phi), \qquad \phi(\vec{x})$$ → *Level-set function*

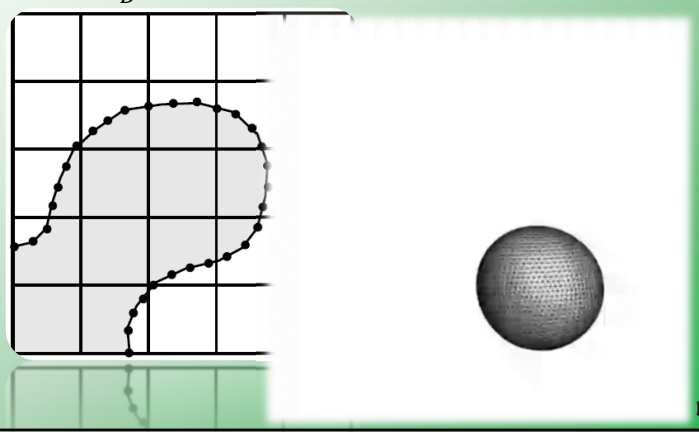


Chap 4 By E. Amani

Sharp interface advection methods

- **Specific advection methods for sharp interfaces**
  - **Front Tracking (FT)**

$$I(\vec{x}) = \int_D G(\vec{x} - \vec{y})\, H(\vec{y}) \mathrm{d}\vec{y}$$ → *Indicator function*

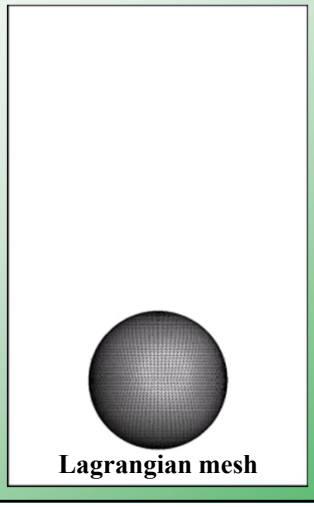


Chap 4 By E. Amani

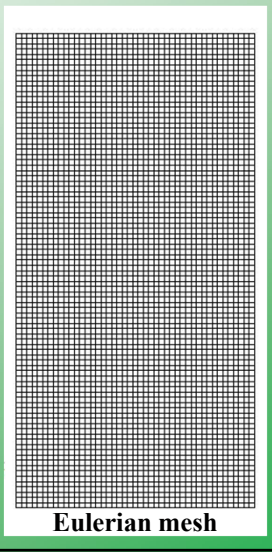
Sharp interface advection methods

- Specific advection methods for sharp interfaces
  - Front Tracking (FT)

DNS of droplets using Front Tracking Method (FTM) ►



Lagrangian mesh



Eulerian mesh

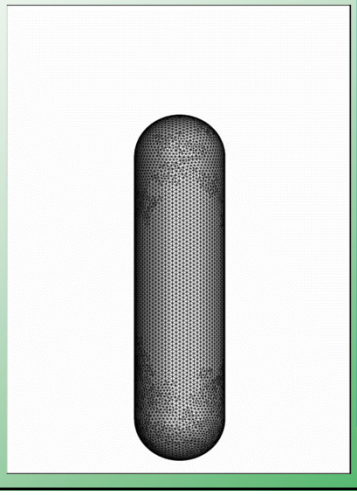
Chap 4

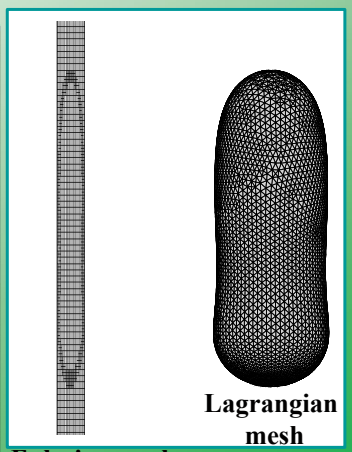
By E. Amani

Sharp interface advection methods

- Specific advection methods for sharp interfaces
  - Front Tracking (FT)

DNS of droplets using Front Tracking Method (FTM) ►





Lagrangian mesh

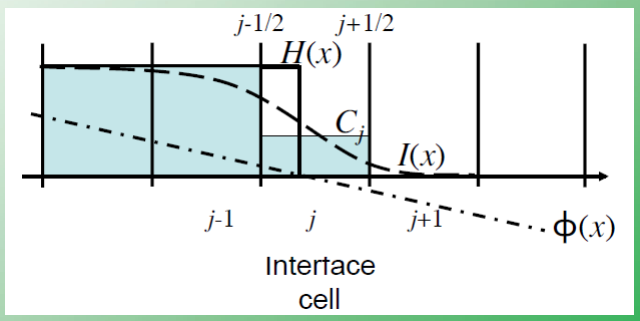
Eulerian mesh

Chap 4

By E. Amani

Sharp interface advection methods

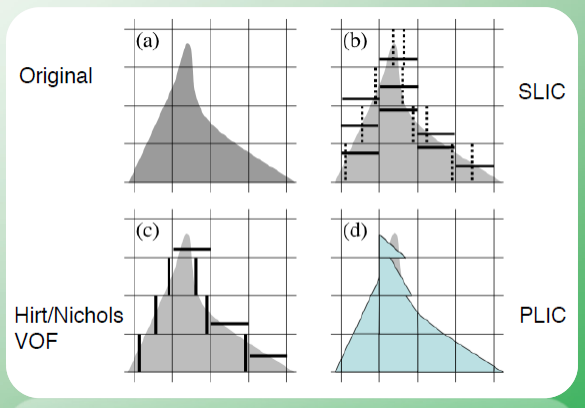
- Specific advection methods for sharp interfaces
  - Volume Of Fluid (VOF)
  - Level Set (LS)
  - Front Tracking (FT)
  - …



Chap 4 By E. Amani

VOF

- Geometric methods
  - Young's Piecewise Linear Interface Calculation (PLIC) method (FLUENT, OpenFOAM)
  - …



Chap 4 By E. Amani

VOF

- Geometric methods → *Generally: More accurate*
  - Young's Piecewise Linear Interface Calculation (PLIC) method (FLUENT, OpenFOAM)
  - …
- Algebraic (fluxing) methods → *Generally: More robust, less computational costs*
  - Multidimensional Universal Limiter with Explicit Solution (MULES) (OpenFOAM)
  - …

Chap 4 By E. Amani

Algebraic VOF

- MULES algorithm
- Flux-Corrected Transport (FCT)
- Compressive Interface Capturing Scheme for Arbitrary Meshes (CICSAM)
- Inter-gamma compressive
- …

Chap 4 By E. Amani

Algebraic VOF

- **MULES algorithm based on OpenFOAM:**
  - **Adding a compression term**

$$\frac{\partial H}{\partial t} + \nabla.(\boldsymbol{u}H) + \underbrace{\nabla.[\boldsymbol{u}_r H(1-H)]}_{\text{Artificial compression term}} = 0$$

  - **Using MULES limiter: Instead of Eq. (2.4)**

$$F_f^n = F_{fU}^n + \lambda_{\text{MULES}} F_{fC}^n$$

Upwind approximation — *MULES limiter* — *High-order Flux correction*

$$F_f^n = \psi F_{fH}^n + (1-\psi)F_{fU}^n = F_{fU}^n + \psi(\underbrace{F_{fH}^n - F_{fU}^n}_{F_{fC}^n})$$

High-order approximation

Algebraic VOF

- **MULES algorithm based on OpenFOAM:**

$$F_{fC} = F_{fH}^n - F_{fU} + \phi_{rf} H_{rf}(1 - H_{rf})$$

$F_{fU}^n = \phi_f H_f$: H_f from Eqs. (2.4)-(5.4), $\psi_f = 0$

$F_{fH}^n = \phi_f H_f$: H_f from Eqs. (2.4)-(5.4), $\psi_f \rightarrow$ vanLeer limiter

H_{rf} from Eqs. (2.4)-(5.4), $\psi_{rf} \rightarrow$

$$\psi_{rf} = \min\left\{\max\left[1 - \max\left[\left(1-(4H_P(1-H_P))\right)^2, \left(1-(4H_N(1-H_N))\right)^2\right], 0\right], 1\right\}$$

$$\phi_{rf} = \boldsymbol{U}_{rf}.\boldsymbol{S}_f = \min_{f \in V_{ij}}\left[c_\gamma \frac{|\phi_f|}{\|\boldsymbol{S}_f\|}, \max_{f \in D}\left(\frac{|\phi_f|}{\|\boldsymbol{S}_f\|}\right)\right](\boldsymbol{n}_f.\boldsymbol{S}_f)$$

Cell i.j — *Model constant* — *Entire domain*

$0 < c_\gamma < 2$, *default* $c_\gamma = 1$

Algebraic VOF

- **MULES algorithm based on OpenFOAM:**

$$\boldsymbol{n}_f = \frac{(\nabla H)_f}{\|(\nabla H)_f\| + \delta}, \delta = \frac{\varepsilon}{\Delta}, \varepsilon = 10^{-8}$$

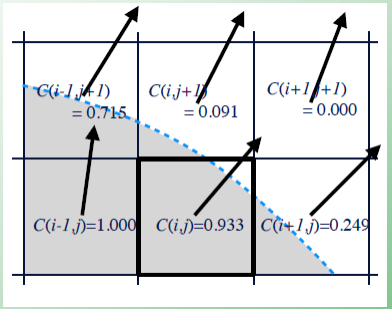
$$(\nabla H)_f = \frac{(\nabla H)_P + (\nabla H)_N}{2}$$

Δ: $\sqrt[3]{V_{ij}}$

  - **The main step: Determining λ_{MULES}**
    - **See the sources in file "Chap4-MULES.rar"**

Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
     a. **Computing interface normals**

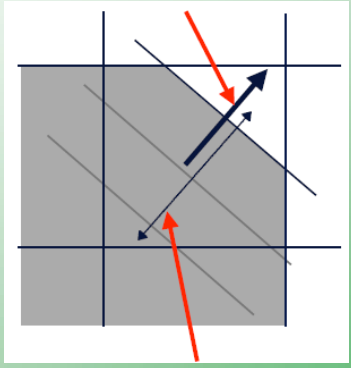


$$\vec{m} = -\vec{\nabla}_h C$$

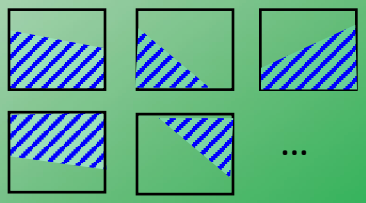
Discrete del operator

Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
     a. **Computing interface normals**
     b. **Computing interface locations**



$\vec{m}.\vec{x} = m_x x + m_y y = \mathrm{a}$



Chap 4 By E. Amani

Coding: NSMF2.py

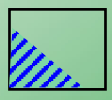
- **Notes on Young's GVOF algorithm:**
  - ✓ **Area vs. a: Fast explicit formulation (Appendix C of Ref. [1])** $\vec{m}.\vec{x} = m_x x + m_y y = \mathrm{a}$

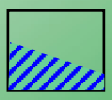
$m_1 = \min(m_x, m_y) \quad m_2 = \max(m_x, m_y)$

$A_1 = m_1/(2m_2),$

$$A = \frac{\mathrm{a}^2}{2m_1 m_2}, \quad \text{for: } 0 \le \mathrm{a} < m_1,$$

$$A = \frac{\mathrm{a}}{m_2} - A_1, \quad \text{for: } m_1 \le \mathrm{a} \le 1/2,$$





$$\mathrm{a} = \sqrt{2m_1 m_2 A}, \quad \text{for: } 0 \le A < A_1,$$

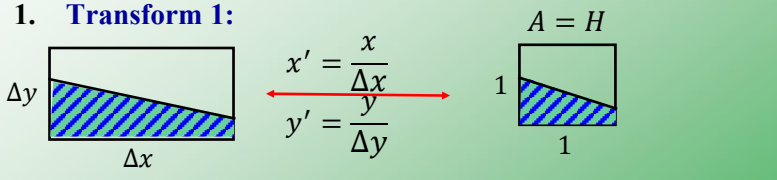
$$\mathrm{a} = m_2 A + \frac{m_1}{2}, \quad \text{for: } A_1 \le A \le 1/2.$$

  - ▪ **Limitations:** $\Delta x = \Delta y = 1$
    $A, \mathrm{a} < 0.5$
    $m_x, m_y > 0$
    $m_x + m_y = 1$

Chap 4 By E. Amani

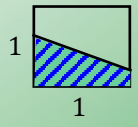
Coding: NSMF2.py

- **Notes on Young's GVOF algorithm:**
  - ✓ **Transformations:**
    1. **Transform 1:**



$$m_x x + m_y y = \mathrm{a} \longleftrightarrow \underbrace{(m_x \Delta x)}_{m'_x} x' + \underbrace{(m_y \Delta y)}_{m'_y} y' = \underbrace{\mathrm{a}}_{\mathrm{a}'}$$

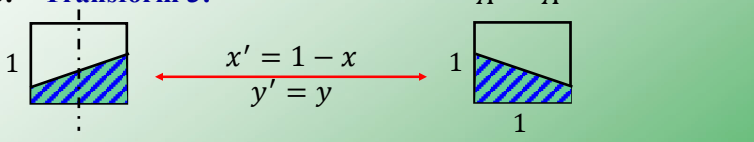
    2. **Transform 2:**



$$m_x x + m_y y = \mathrm{a} \quad (m_x + m_y \neq 1) \longleftrightarrow \underbrace{\frac{m_x}{c}}_{m'_x} x + \underbrace{\frac{m_y}{c}}_{m'_y} y = \underbrace{\frac{\mathrm{a}}{c}}_{\mathrm{a}'}$$

Coding: NSMF2.py

- **Notes on Young's GVOF algorithm:**
  - ✓ **Transformations:**
    3. **Transform 3:**

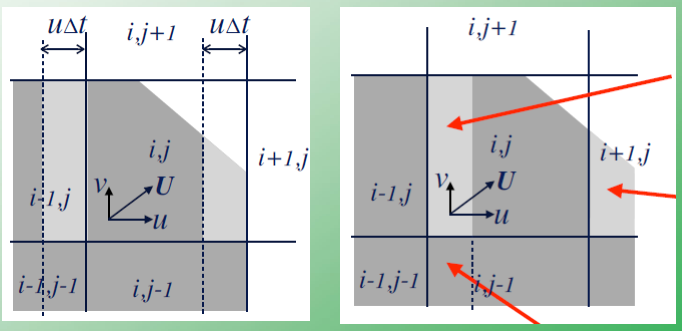


$$m_x x + m_y y = \mathrm{a} \quad (m_x < 0) \longleftrightarrow \underbrace{(-m_x)}_{m'_x} x' + \underbrace{m_y}_{m'_y} y' = \underbrace{\mathrm{a} - m_x}_{\mathrm{a}'}$$

    4. **…**

Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
  2. **Advection of front:**
     - ❑ **Splitting methods (Combined linear mapping, …)**



Chap 4 By E. Amani

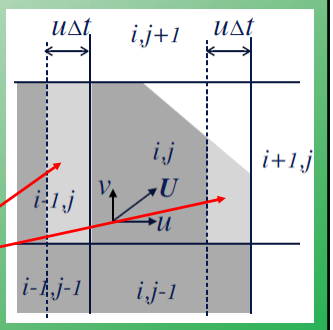
Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
  2. **Advection of front:**
     - ❑ **Splitting methods: Simple**

$$\frac{\partial}{\partial t}\int_V HdV + \oint_S H(\boldsymbol{u}.\boldsymbol{n})dS = 0$$

x-dir $$H_{i,j}^{n+1} = H_{i,j}^{n} - \underbrace{\frac{\Delta t}{\Delta V}\oint_{i+\frac{1}{2}} HudS}_{\widetilde{\Phi}_{i+1/2}} + \underbrace{\frac{\Delta t}{\Delta V}\oint_{i-1/2} HudS}_{\widetilde{\Phi}_{i-1/2}}$$

$$H_{i,j}^{n+1} = H_{i,j}^{n} - \widetilde{\Phi}_{i+1/2} + \widetilde{\Phi}_{i-1/2}$$



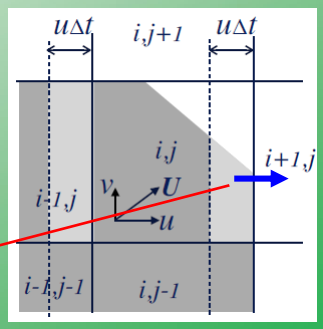
Chap 4 By E. Amani

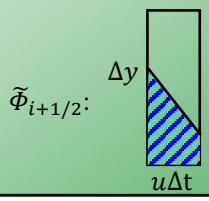
Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
  2. **Advection of front:**
     - ❑ **Splitting methods: Combined linear mapping**

$$\frac{\partial}{\partial t}\int_V H dV + \oint_S H(\boldsymbol{u}.\boldsymbol{n})dS = \int_V H(\nabla.\boldsymbol{u})dV$$

x-dir $\quad H_{i,j}^{n+1} = H_{i,j}^{n} - \tilde{\Phi}_{i+1/2} + \tilde{\Phi}_{i-1/2} + H_{i,j}^{n}(u_{i+1/2} - u_{i-1/2})$

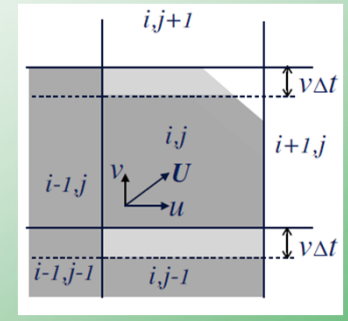


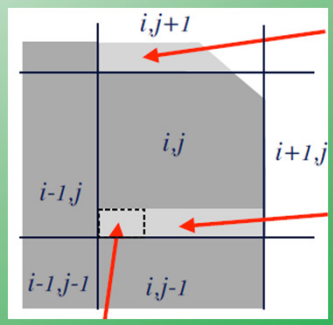


$\tilde{\Phi}_{i+1/2}$:

Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
  2. **Advection of front:**
     - ❑ **Splitting methods (simple, Combined linear mapping, …): y-direction**





Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
  2. **Advection of front:**
     - ❑ **Splitting methods**
     - ❑ **Unspilt methods**

Geometric VOF

- **Algorithm:**
  1. **Reconstruction of interface:**
  2. **Advection of front:**
  3. **Redistribution:**
     - ✓ **To satisfy ($0 < C < 1$)**

  - ➢ **For more details of GVOF algorithms, see Ref. [1]**
  - ➢ **For more details of Young's GVOF algorithm, see Ref [1] and the sources in file "Chap4-YoungGVOF.rar"**
  - ➢ **I implemented the complete algorithm in "NSMF2.py" code.**

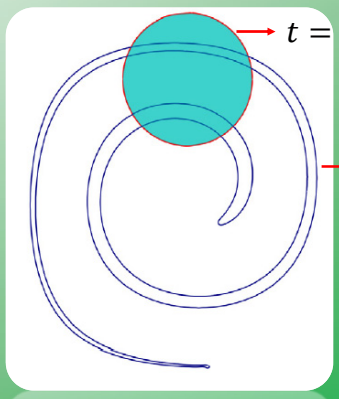
Benchmark #1: 2D vortex

- **Assessing interface advection methods**
  - **Given velocity field**

$$\psi(x,y,t) = \frac{1}{\pi}\cos\left(\frac{\pi t}{T}\right)\sin^2\left(\frac{\pi x}{L_x}\right)\sin^2\left(\frac{\pi y}{L_y}\right);\ u = \frac{\partial\psi}{\partial y}, v = -\frac{\partial\psi}{\partial x}$$

$$E = \sum_{i,j} dxdy\left|C_{i,j}^{T} - C_{i,j}^{0}\right|$$

$$E_M = \frac{|V_T - V_0|}{V_0} = \frac{\left|\sum_{i,j} C_{i,j}^{T} dxdy - \sum_{i,j} C_{i,j}^{0} dxdy\right|}{\sum_{i,j} C_{i,j}^{0} dxdy}$$



$L_x = L_y = 1, T = 2$

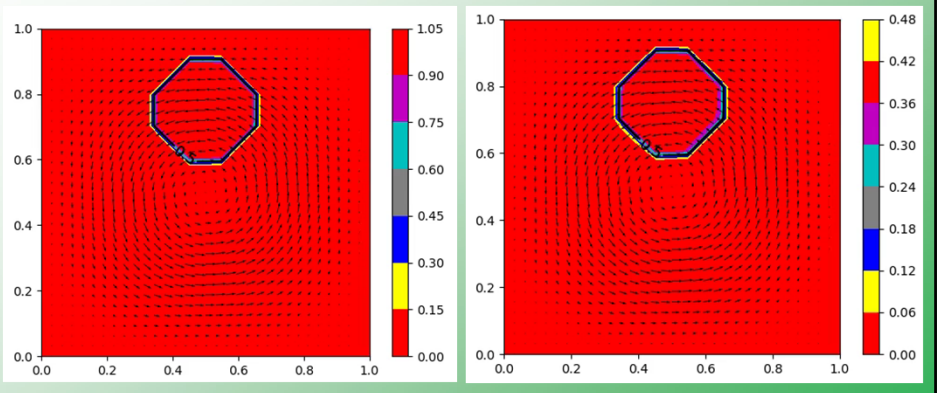
Grid: 32×32

$\Delta t = 0.005$

Chap 4 By E. Amani

Benchmark #1: 2D vortex

CD ▼

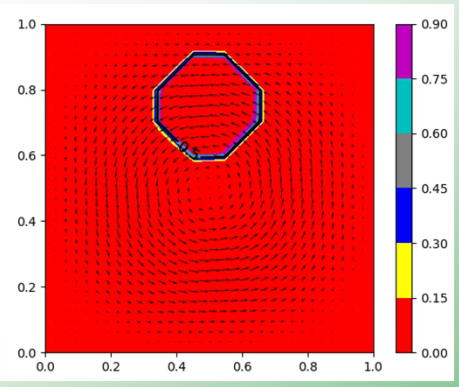


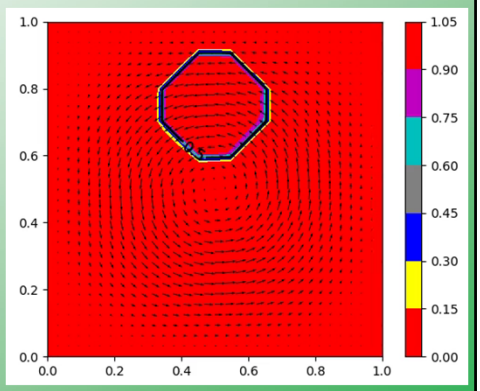
▲ FOU

Chap 4 By E. Amani

Benchmark #1: 2D vortex

vanLeer ▼





▲ Young's GVOF (Combined linear mapping)

Benchmark #1: 2D vortex

- Performance comparison

	CD	FOU	vanLeer	GVOF
Accuracy error (%)	4.20	9.61	4.82	0.63
Mass conservation error (%)	10.68	0.00	0.00	0.00

Project#1: DNS of multiphase flows

Part I

- MULES algorithm (Fall 2022): Code development
- Flux-Corrected Transport (FCT) (Fall 2023): Code development
- Young's GVOF + Combined linear mapping (Fall 2024): My code comprehension
- CICSAM (Fall 2025): Code development
- …

Surface tension computation

- Continuous Surface Force (CSF)
- Continuous Surface Stress (CSS)
- Kernel method
- Proper Representation of Surface Tension (PROST)
- Height Function (HF)
- Pressure Correction Marker Method (PCMM)
- …

Surface tension computation

- **Continuous Surface Force (CSF)**
  - **Delta function approximation**

$$\vec{F}_\sigma = \vec{f}_\sigma \delta_s = \sigma\kappa\underbrace{\delta_s \hat{n}}_{-\vec{\nabla}H \sim -\vec{\nabla}C} \sim -\sigma\kappa\vec{\nabla}C \sim -\sigma\kappa\vec{\nabla}_h C$$

$$\vec{F}_\sigma = -\sigma\kappa\vec{\nabla}_h C$$

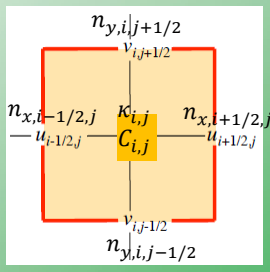
$$\kappa = -\vec{\nabla}.\hat{n}$$

$$\hat{n} = -\frac{\vec{\nabla}_h C}{|\vec{\nabla}_h C|} = \frac{\vec{m}}{|\vec{m}|};\ \vec{m} = -\vec{\nabla}_h C$$

Surface tension computation

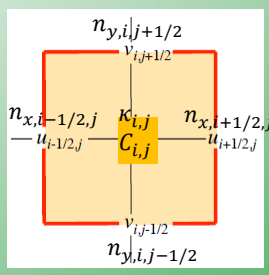
- **CSF: Algorithm**
  - **Smoothing volume fraction**

$$\tilde{C}_{i,j} = \frac{1}{2}C_{i,j} + \frac{1}{8}\left(C_{i+1,j} + C_{i,j+1} + C_{i-1,j} + C_{i,j-1}\right)$$



Surface tension computation

- **CSF: Algorithm**
  - **Smoothing volume fraction**
  - **Standard CSF**



$$F_{x,i+1/2,j} = -\sigma\kappa_{i+1/2,j}\left(\frac{\partial C}{\partial x}\right)_{i+1/2,j}$$

$$= -\sigma\frac{(\kappa_{i,j}+\kappa_{i+1,j})}{2}\frac{(C_{i+1,j}-C_{i,j})}{dx}$$

$$\kappa_{i,j} = -\left[\frac{1}{dx}(n_{x,i+1/2,j}-n_{x,i-1/2,j})+\frac{1}{dy}(n_{y,i,j+1/2}-n_{y,i,j-1/2})\right]$$

$$n_{x,i+1/2,j} = \frac{m_{x,i+1/2,j}}{(m^2_{x,i+1/2,j}+m^2_{y,i+1/2,j})^{1/2}+\varepsilon}$$

$$m_{x,i+1/2,j} = -\frac{C_{i+1,j}-C_{i,j}}{dx} \quad m_{y,i,j+1/2} = -\frac{C_{i,j+1}-C_{i,j}}{dy}$$

$$m_{y,i+1/2,j} = \frac{1}{4}(m_{y,i,j+1/2}+m_{y,i+1,j+1/2}+m_{y,i,j-1/2}+m_{y,i+1,j-1/2})$$

Chap 4 **By E. Amani**

Coding: VOF.py

- **CSF: Algorithm**
  - **Smoothing volume fraction**

$$\tilde{C}_{i,j} = \frac{1}{2}C_{i,j}+\frac{1}{8}(C_{i+1,j}+C_{i,j+1}+C_{i-1,j}+C_{i,j-1})$$

NSMF1.py × results.py × NSMF2.py × NSMF3.py × VOF3.py ×

```
41      elif (surfaceTen=='2'): #standard CSF algorithm
42          chiF=np.zeros((nx+3,ny+3))
43          #filter chi
44          chiF[2:nx+2,2:ny+2]=(0.5*chi[2:nx+2,2:ny+2]+0.125*(chi[2:nx+2,1:ny+1]+chi[2:nx+2,3:ny+3]
45                               +chi[1:nx+1,2:ny+2]+chi[3:nx+3,2:ny+2]))
```

Chap 4 **By E. Amani**

Coding: VOF.py

- **CSF: Algorithm**
  - **Standard CSF**

$$m_{x,i+1/2,j} = -\frac{C_{i+1,j} - C_{i,j}}{dx} \quad m_{y,i,j+1/2} = -\frac{C_{i,j+1} - C_{i,j}}{dy}$$

```
#standard CSF
mxu[2:nx+1,2:ny+2]=-(chiF[3:nx+2,2:ny+2]-chiF[2:nx+1,2:ny+2])/dx
myv[2:nx+2,2:ny+1]=-(chiF[2:nx+2,3:ny+2]-chiF[2:nx+2,2:ny+1])/dy
```

$$m_{y,i+1/2,j} = \frac{1}{4}\left(m_{y,i,j+1/2} + m_{y,i+1,j+1/2} + m_{y,i,j-1/2} + m_{y,i+1,j-1/2}\right)$$

```
mxv[2:nx+2,2:ny+1]=interpolate_ufTOvf(mxu, 2, nx+2, 2, ny+1)
myu[2:nx+1,2:ny+2]=interpolate_vfTOuf(myv, 2, nx+1, 2, ny+2)
```

$$n_{x,i+1/2,j} = \frac{m_{x,i+1/2,j}}{\left(m_{x,i+1/2,j}^2 + m_{y,i+1/2,j}^2\right)^{1/2} + \varepsilon}$$

$$\kappa_{i,j} = -\left[\frac{1}{dx}\left(n_{x,i+1/2,j} - n_{x,i-1/2,j}\right) + \frac{1}{dy}\left(n_{y,i,j+1/2} - n_{y,i,j-1/2}\right)\right]$$

```
mxu/=(np.sqrt(mxu*mxu+myu*myu)+1.e-16)
myv/=(np.sqrt(mxv*mxv+myv*myv)+1.e-16)
k[2:nx+2,2:ny+2]=-((mxu[2:nx+2,2:ny+2]-mxu[1:nx+1,2:ny+2])/dx+
                   (myv[2:nx+2,2:ny+2]-myv[2:nx+2,1:ny+1])/dy)
```

Chap 4 **By E. Amani**

Coding: VOF.py

- **CSF: Algorithm**
  - **Standard CSF**

$$F_{x,i+1/2,j} = -\sigma\kappa_{i+1/2,j}\left(\frac{\partial C}{\partial x}\right)_{i+1/2,j}$$

$$= -\frac{\sigma}{2dx}\left(C_{i+1,j} - C_{i,j}\right)\left(\kappa_{i,j} + \kappa_{i+1,j}\right)$$

```
return (-sigma/(2*dx)*(chi[3:nx+2,2:ny+2]-chi[2:nx+1,2:ny+2])*(k[3:nx+2,2:ny+2]+k[2:nx+1,2:ny+2]),
        -sigma/(2*dy)*(chi[2:nx+2,3:ny+2]-chi[2:nx+2,2:ny+1])*(k[2:nx+2,3:ny+2]+k[2:nx+2,2:ny+1]))
```

Chap 4 **By E. Amani**

Benchmark #2: static droplet

- **Assessing surface tension computation**
  - **Analytical solution for pressure difference**

$$\Delta p = p_{in} - p_{out} = \sigma\left(\frac{1}{R_1} + \frac{1}{R_2}\right) \xrightarrow{2D} \Delta p = \frac{\sigma}{R} \longrightarrow \frac{R\Delta p}{\sigma} = 1$$

  - **Parasitic current**

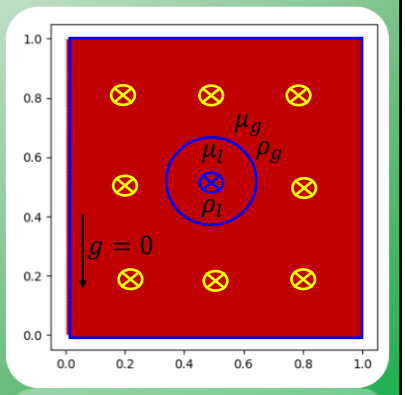
$$Ca = \frac{\mu_l U_{\max}}{\sigma}$$

$\rho_g = 1\ kg/m^3$ $\rho_l = 20\ kg/m^3$

$\mu_l = \mu_g = 0.01\ Pa.s$ **Grid: 32×32**

$\sigma = 0.1\ \mathrm{N/m}$ $\Delta t = 0.00125\ s$

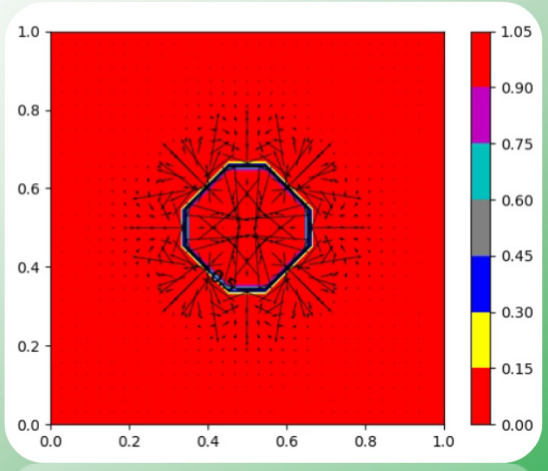
$d = 0.3\ m$ **Young's geometric VOF + CSF**



Chap 4 By E. Amani

Benchmark #2: static droplet

- **Results**

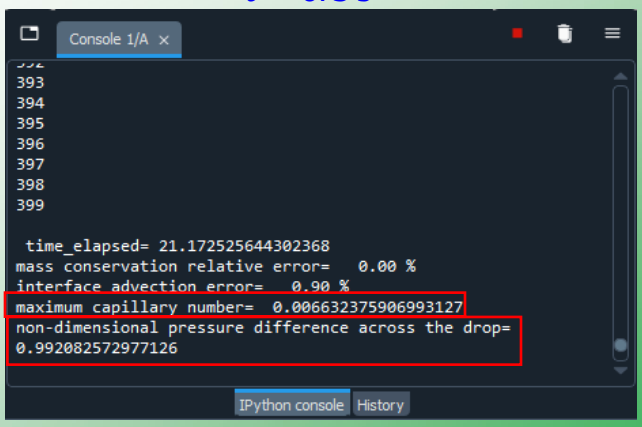


Chap 4 By E. Amani

Benchmark #2: static droplet

- **Performance**

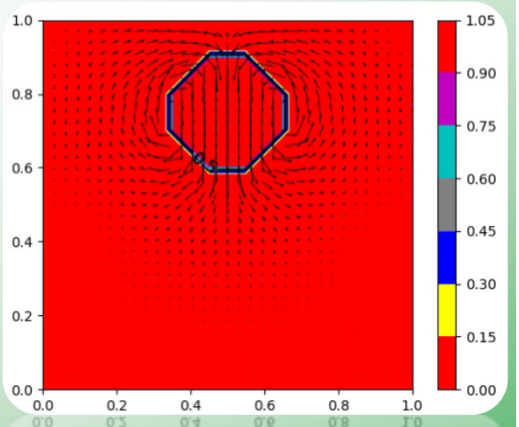
$t = 0.5\ s$

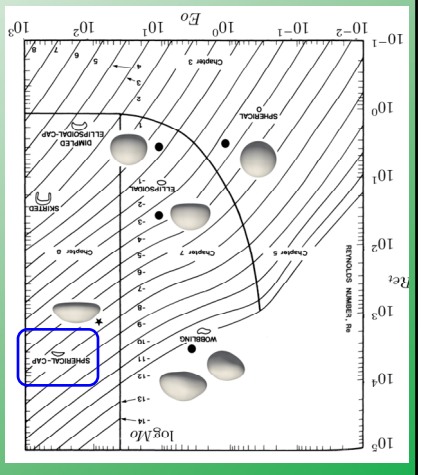


Chap 4 By E. Amani

Coding: NSMF3.py

- **Free-falling drop**
  - Young's geometric VOF + CSF

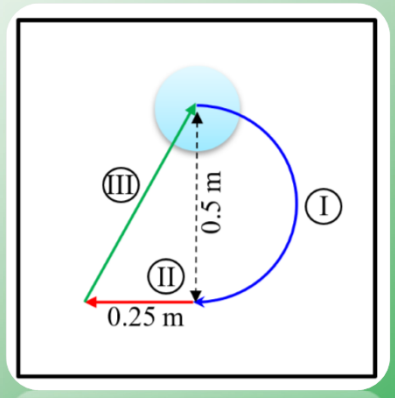




Chap 4 By E. Amani

Hands-on practice

- **HW#4:**
  - Checking the FLUENT standard tutorial on VOF
  - Test of FLUENT VOF-based interface advection strategies



Chap 4 By E. Amani

Project#1: DNS of multiphase flows

Part II

- **CSF + filtering (Fall 2022, 2023): Code development**
- **PROST (Fall 2024): Code development**
- **Kernel smoothing method (Fall 2025): Code development**
- **…**

Chap 4 By E. Amani