

Challenge: Advectiong a sharp interface

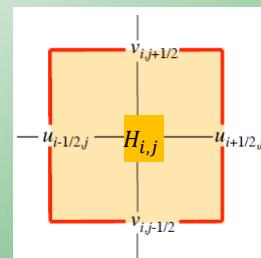
- Step1: Advectiong the indicator function
 - Second-order Central Difference (CD)

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

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

$$\frac{\partial}{\partial t} (H_{i,j} \Delta x \Delta y) = - \oint_S H(\mathbf{u} \cdot \mathbf{n}) dS$$

$$H_{i,j}^{n+1} = H_{i,j}^n - \frac{\Delta t}{\Delta x \Delta y} \nabla_h \cdot (\mathbf{u}H)$$



More precisely, $\int_V H dV = C \Delta V$ 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 .

Challenge: Advectiong a sharp interface

- Step1: Advectiong the indicator function

 - Second-order Central Difference (CD)

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

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

$$\frac{\partial}{\partial t} (H_{i,j} \Delta x \Delta y) = - \oint_S H(\mathbf{u} \cdot \mathbf{n}) dS$$

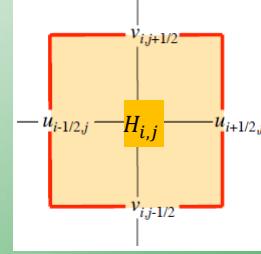
$$H_{i,j}^{n+1} = H_{i,j}^n - \frac{\Delta t}{\Delta x \Delta y} \nabla_h \cdot (\mathbf{u}H)$$

$$H_{i,j}^{n+1}$$

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

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

Chap 4



By E. Amani

Coding: NSMF1.py

- Step1: Advectiong the indicator function

 - Second-order Central Difference (CD)

$$H_{i,j}^{n+1}$$

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

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

```

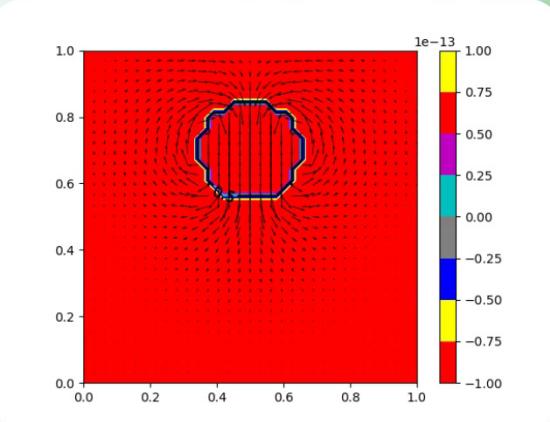
NSMF1.py x results.py x
82 #----- START TIME LOOP -----
83 for istep in range(nstep):
84     print(istep)
85
86     #--- ADVECT marker using centered difference plus diffusion ---
87     chio=chi.copy()
88
89     if scalar.adve='c':
90         chi[2:nx+2,2:ny+2]=(chi[2:nx+2,2:ny+2]-0.5*dt/dx)*(u[2:nx+2,2:ny+2]*(chi[3:nx+3,2:ny+2]
91                     +chi[2:nx+2,2:ny+2])-u[1:nx+1,2:ny+2]*(chi[1:nx+1,2:ny+2]+chi[2:nx+2,2:ny+2]))
92         +0.5*(dt/dy)*(v[2:nx+2,2:ny+2]*(chi[2:nx+2,1:ny+3]
93                     +chi[2:nx+2,2:ny+2])-v[2:nx+2,1:ny+1]*(chi[2:nx+2,1:ny+1]+chi[2:nx+2,2:ny+2])) )
94         +(mon*dt/dx)*(chi[3:nx+3,2:ny+2]-2.0*chi[2:nx+2,2:ny+2]+chi[1:nx+1,2:ny+2])
95         +(mon*dt/dy)*(chi[2:nx+2,3:ny+3]-2.0*chi[2:nx+2,2:ny+2]+chi[2:nx+2,1:ny+1]))
```

Chap 4

By E. Amani

Coding: NSMF1.py

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



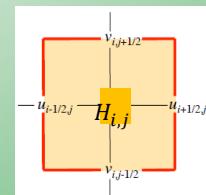
Chap 4

By E. Amani

Coding: NSMF3.py

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

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



```

95         + (m0*dt/dy/dy)*(chio[2:nx+2,3:ny+3]-2.*chio[2:nx+2,2:ny+2]+chio[2:nx+2,1:ny+1])) \\
96     elif scalar_adve== 'u' :
97         chio[2:nx+2,2:ny+2]=(chio[2:nx+2,2:ny+2] \\
98             -(dt/dx)*' \\
99                 (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)) \\
100            - \\
101            (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))) \\
102            -(dt/dy)*' \\
103            (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)) \\
104            - \\
105            (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)))) \\
106     else:

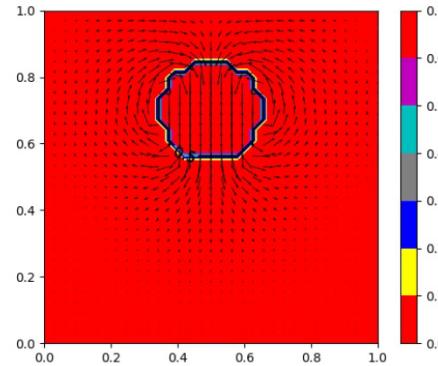
```

Chap 4

By E. Amani

Coding: NSMF1.py

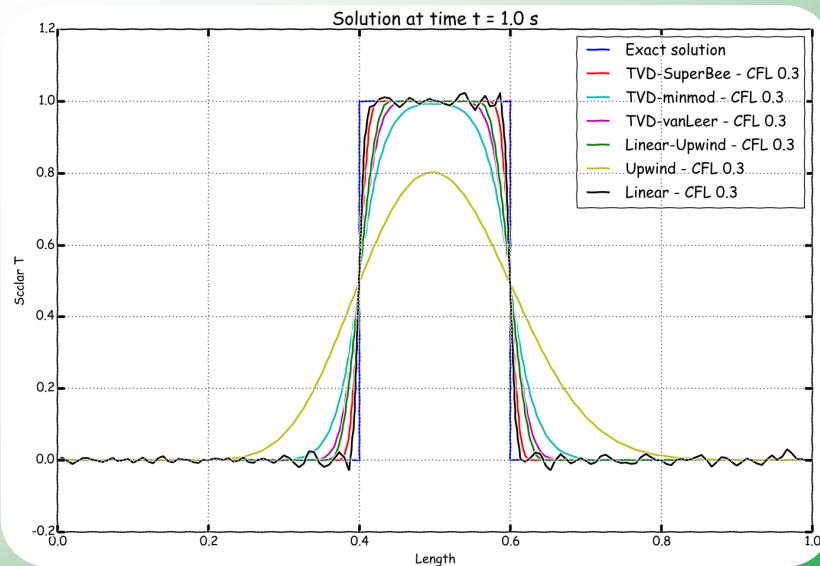
- Step1: Advectiong the indicator function
 - First-Order Upwind (FOU)



Chap 4

By E. Amani

Coding: 1D advection test



Chap 4

By E. Amani

Conventional flux limiters

- Step1: Advectiong the indicator function
 - Non-linear flux limiters

 Lecture Notes: IV.4

Chap 4

By E. Amani

Coding: NSMF1.py

- Step1: Advectiong the indicator function

- Non-linear flux limiter: vanLeer

$$\begin{aligned}
 H_{i,j}^{n+1} &= H_{i,j}^n \\
 &- \frac{\Delta t}{\Delta x} \left\{ u_{i+\frac{1}{2},j}^n \left[H_{i,j}^n + \frac{(H_{i+1,j}^n - H_{i,j}^n)}{2} \left\{ 1 - \xi_{i+\frac{1}{2},j} [1 - \psi(r_{i+\frac{1}{2}})] \right\} \right] \right. \\
 &\quad \left. - u_{i-\frac{1}{2},j}^n \left[H_{i-1,j}^n + \frac{(H_{i,j}^n - H_{i-1,j}^n)}{2} \left\{ 1 - \xi_{i-\frac{1}{2},j} [1 - \psi(r_{i-\frac{1}{2}})] \right\} \right] \right\} \\
 &\quad \dots
 \end{aligned}$$

```

186     else:
187         kesiu=2*(j>0)-1
188         kesiv=2*(j>0)-1
189         chi[2:nx+2,2:ny+2]=(chio[2:nx+2,2:ny+2]
190                     -(dt/dx)*(
191                         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])*(
192                             (1-kesiu[2:nx+2,2:ny+2])*(1-limiterf(r*(kesiu,chio,2,nx+2,ny+2))))/2)
193
194         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: Advectiong the indicator function

- Non-linear flux limiter: vanLeer

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

$$r_{x,i+1/2}^n = \frac{\left(1 + \xi_{i+\frac{1}{2},j}\right)(H_{i,j}^n - H_{i-1,j}^n) + \left(1 - \xi_{i+\frac{1}{2},j}\right)(H_{i+2,j}^n - H_{i+1,j}^n)}{(H_{i+1,j}^n - H_{i,j}^n) + \varepsilon}$$

```
E:\doc-desktop\Codes\NSMF\final\NSMF1.py
  NSMF1.py x results.py x
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 #upwind
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))
```

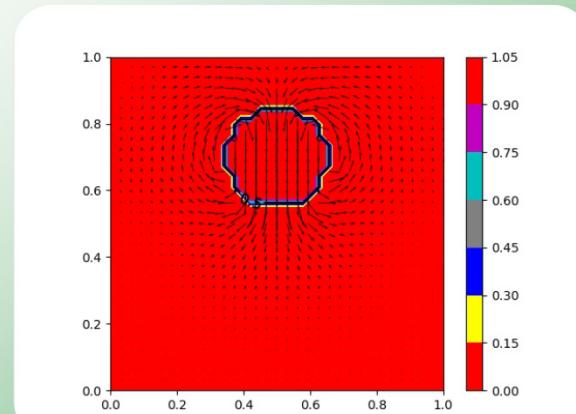
Chap 4

By E. Amani

Coding: NSMF1.py

- Step1: Advectiong the indicator function

- Non-linear flux limiter: vanLeer



Chap 4

By E. Amani

Sharp interface advection methods

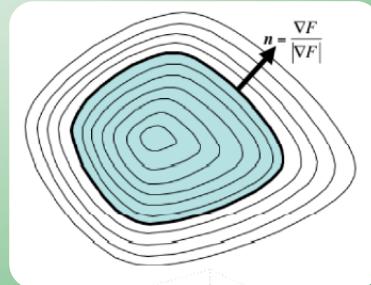
- Specific advection methods for sharp interfaces

- Volume Of Fluid (VOF)

$$C = \langle \chi \rangle = \langle H \rangle = \alpha \quad \rightarrow \text{Volume fraction or color function}$$

- Level Set (LS)

$$H = H(\phi), \quad \phi(\vec{x}) \quad \rightarrow \text{Level-set function}$$



Chap 4

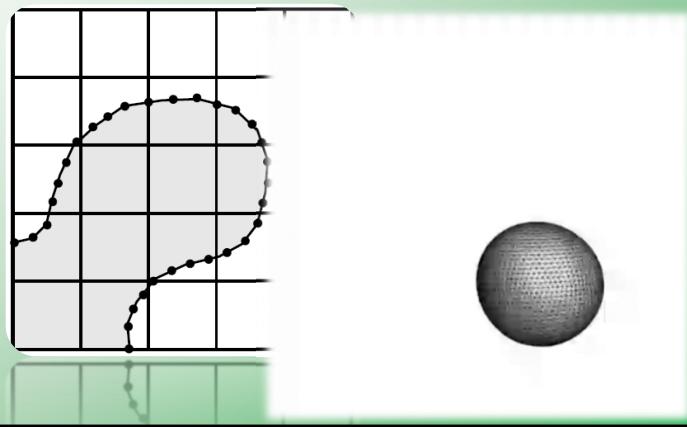
By E. Amani

Sharp interface advection methods

- Specific advection methods for sharp interfaces

- Front Tracking (FT)

$$I(\vec{x}) = \int_D G(\vec{x} - \vec{y}) H(\vec{y}) d\vec{y} \quad \rightarrow \text{Indicator function}$$



Chap 4

By E. Amani

Sharp interface advection methods

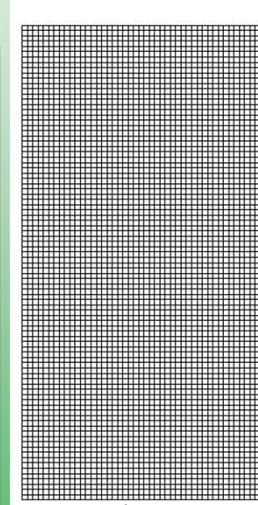
- Specific advection methods for sharp interfaces
 - Front Tracking (FT)

DNS of droplets
using Front
Tracking Method
(FTM) ►

Chap 4



Lagrangian mesh



Eulerian mesh

By E. Amani

Sharp interface advection methods

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

DNS of droplets
using Front
Tracking Method
(FTM) ►

Chap 4



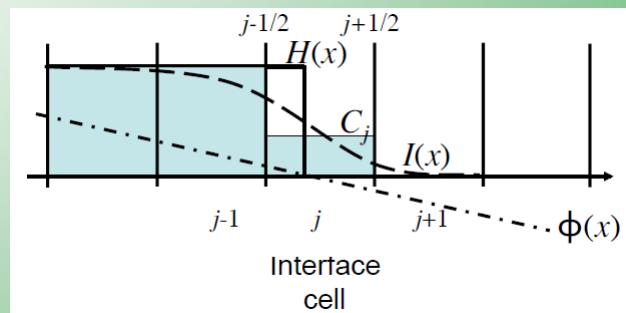
Lagrangian
mesh

Eulerian mesh

By E. Amani

Sharp interface advection methods

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

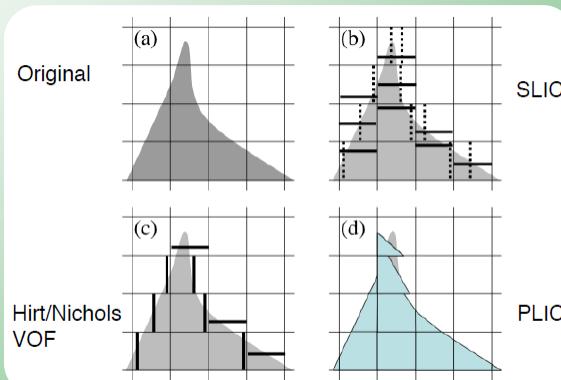


Chap 4

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

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

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

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

Upwind MULES High-order
approximation limiter Flux correction

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

High-order approximation F_{fC}ⁿ

Chap 4

By E. Amani

Algebraic VOF

- MULES algorithm based on OpenFOAM:

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

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

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

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

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

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

$$\text{Cell } i,j \quad \text{Model constant} \quad \text{Entire domain}$$

$$0 < c_\gamma < 2, \text{ default } c_\gamma = 1$$

Chap 4

By E. Amani

Algebraic VOF

- MULES algorithm based on OpenFOAM:

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

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

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

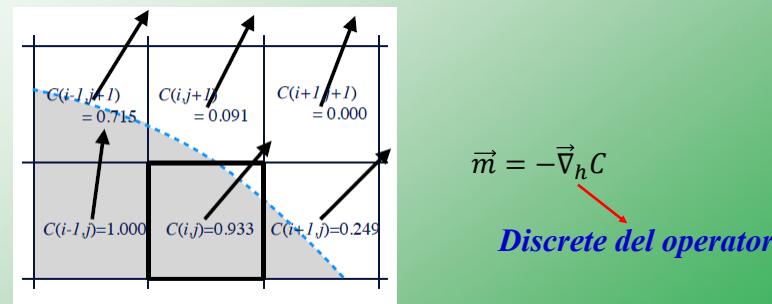
Chap 4

By E. Amani

Geometric VOF

- Algorithm:

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



Chap 4

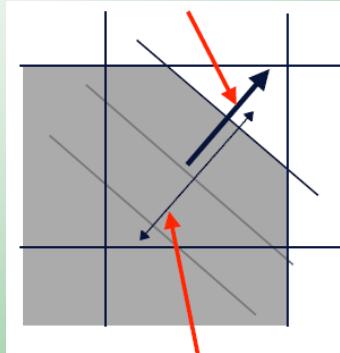
By E. Amani

Geometric VOF

- Algorithm:

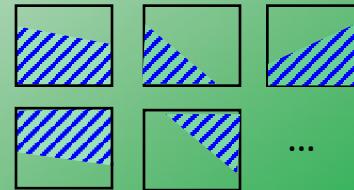
- Reconstruction of interface:

- Computing interface normals
- Computing interface locations



Chap 4

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



By E. Amani

Coding: NSMF2.py

- Notes on Young's GVOF algorithm:

- Area vs. a : Fast explicit formulation (Appendix C of Ref. [1])

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

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

$$A_1 = m_1 / (2m_2),$$

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

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



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

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

- Limitations:

$$\Delta x = \Delta y = 1$$

$$A, a < 0.5$$

$$m_x, m_y > 0$$

$$m_x + m_y = 1$$

Chap 4

By E. Amani

Coding: NSMF2.py

- Notes on Young's GVOF algorithm:

- Transformations:

- Transform 1:

$$x' = \frac{x}{\Delta x} \quad y' = \frac{y}{\Delta y}$$

$$m_x x + m_y y = a \quad (m_x \Delta x)x' + (m_y \Delta y)y' = \underbrace{a}_{a'}$$

$$\underbrace{m'_x}_{m_x} \quad \underbrace{m'_y}_{m_y}$$

- Transform 2:

$$m_x x + m_y y = a \quad \frac{m_x}{c}x + \frac{m_y}{c}y = \frac{a}{c}$$

$$m_x + m_y \neq 1 \quad \underbrace{m'_x}_{m_x} \quad \underbrace{m'_y}_{m_y} \quad \underbrace{a'}_{a'}$$

Chap 4

By E. Amani

Coding: NSMF2.py

- Notes on Young's GVOF algorithm:

- Transformations:

- Transform 3:

$$x' = 1 - x \quad y' = y$$

$$m_x x + m_y y = a \quad (-m_x)x' + m_y y' = \underbrace{a - m_x}_{a'}$$

$$m_x < 0 \quad \underbrace{m'_x}_{m_x} \quad \underbrace{m'_y}_{m_y}$$

- ...

Chap 4

By E. Amani

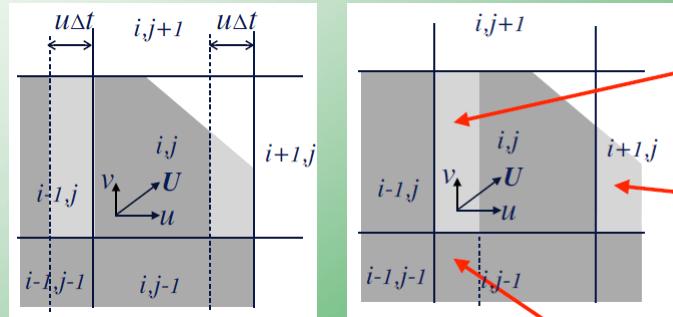
Geometric VOF

- Algorithm:

- Reconstruction of interface:

- Advection of front:

- Splitting methods (Combined linear mapping, ...)



Chap 4

By E. Amani

Geometric VOF

- Algorithm:

- Reconstruction of interface:

- Advection of front:

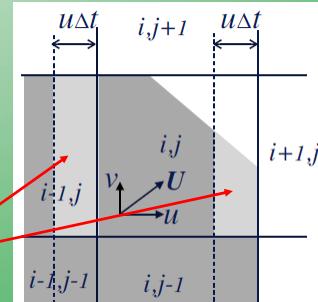
- Splitting methods: Simple

$$\frac{\partial}{\partial t} \int_V H dV + \oint_S H(\mathbf{u} \cdot \mathbf{n}) dS = 0$$

$x\text{-dir}$

$$H_{i,j}^{n+1} = H_{i,j}^n - \underbrace{\frac{\Delta t}{\Delta V} \int_{i+\frac{1}{2}} H u dS}_{\tilde{\Phi}_{i+1/2}} + \underbrace{\frac{\Delta t}{\Delta V} \int_{i-\frac{1}{2}} H u dS}_{\tilde{\Phi}_{i-1/2}}$$

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



Chap 4

By E. Amani

Geometric VOF

- Algorithm:

- Reconstruction of interface:

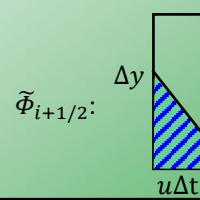
- Advection of front:

- Splitting methods: Combined linear mapping

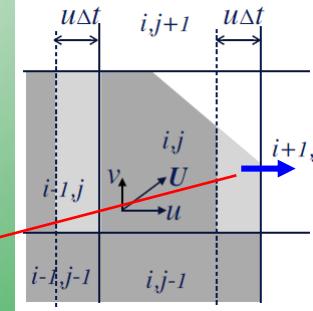
$$\frac{\partial}{\partial t} \int_V H dV + \oint_S H(\mathbf{u} \cdot \mathbf{n}) dS = \int_V H(\nabla \cdot \mathbf{u}) dV$$

$\xrightarrow{x\text{-dir}}$

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



Chap 4



By E. Amani

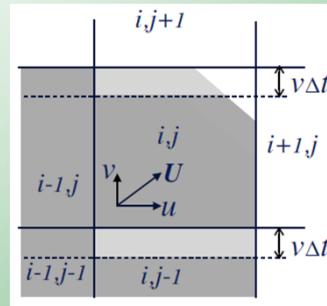
Geometric VOF

- Algorithm:

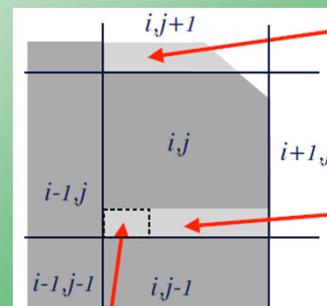
- Reconstruction of interface:

- Advection of front:

- Splitting methods (simple, Combined linear mapping, ...): y-direction



Chap 4



By E. Amani

Geometric VOF

- **Algorithm:**

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

Chap 4

By E. Amani

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”

Chap 4

By E. Amani

Benchmark #1: 2D vortex

- Assessing interface advection methods

Given velocity field

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

$$E = \sum_{i,j} dx dy (C_{i,j}^T - C_{i,j}^0)$$

$$E_M = \frac{|V_T - V_0|}{V_0}$$

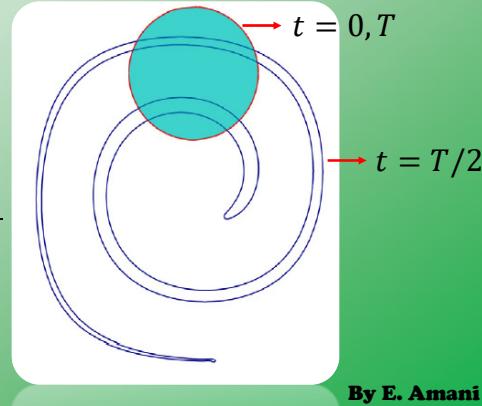
$$= \frac{|\sum_{i,j} C_{i,j}^T dx dy - \sum_{i,j} C_{i,j}^0 dx dy|}{\sum_{i,j} C_{i,j}^0 dx dy}$$

$$L_x = L_y = 1, T = 2$$

Grid: 32×32

Chap 4

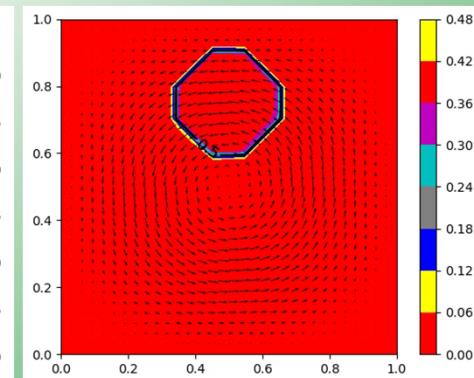
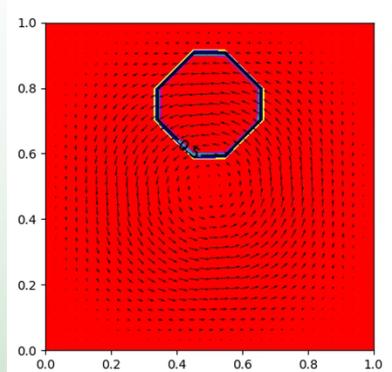
$\Delta t = 0.005$



By E. Amani

Benchmark #1: 2D vortex

CD ▼



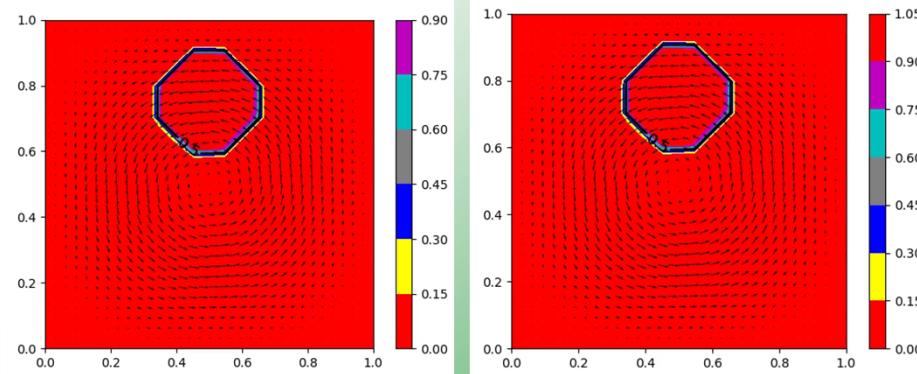
▲ FOU

Chap 4

By E. Amani

Benchmark #1: 2D vortex

vanLeer ▼



Chap 4

By E. Amani

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

Chap 4

By E. Amani

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
- ...

Chap 4

By E. Amani

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)
- ...

Chap 4

By E. Amani

Surface tension computation

- **Continuous Surface Force (CSF)**

- **Delta function approximation**

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

$\underbrace{}$
 $-\vec{\nabla} H \sim -\vec{\nabla} C$

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

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

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

Chap 4

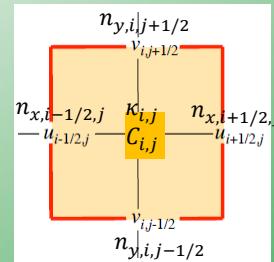
By E. Amani

Surface tension computation

- **CSF: Algorithm**

- **Smoothing volume fraction**

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



Chap 4

By E. Amani

Surface tension computation

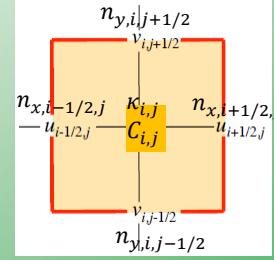
- Continuous Surface Force (CSF)

 - Smoothing volume fraction

 - Standard CSF

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

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



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

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

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

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

Chap 4 **By E. Amani**

Coding: VOF.py

- Continuous Surface Force (CSF)

 - 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 x results.py x NSMF2.py x NSMF3.py x VOF3.py x
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

● Continuous Surface Force (CSF)

➤ Standard CSF

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

```

46      #standard CSF
47      mxu[2:nx+1,2:ny+2]=(chiF[3:nx+2,2:ny+2]-chiF[2:nx+1,2:ny+2])/dx
48      myv[2:nx+2,2:ny+1]=(chiF[2:nx+2,3:ny+2]-chiF[2:nx+2,2:ny+1])/dy
49      mxv[2:nx+2,2:ny+1]=interpolate_uftOvf(mxu, 2, nx+2, 2, ny+1)
50      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]
51      mxu=(np.sqrt(mxu*mxu+myu*myu)+1.e-16)
52      myv=(np.sqrt(mxv*mxv+myv*myv)+1.e-16)
53      k[2:nx+2,2:ny+2]=-(mxu[2:nx+2,2:ny+2]-mxu[1:nx+1,2:ny+2])/dx+
54      (myv[2:nx+2,2:ny+2]-myv[2:nx+2,1:ny+1])/dy

```

Chap 4

By E. Amani

Coding: VOF.py

● Continuous Surface Force (CSF)

➤ Standard CSF

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

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

```

54      myv[2:nx+2,2:ny+2]=(chiF[3:nx+2,2:ny+2]-chiF[2:nx+1,2:ny+2])*(k[3:nx+2,2:ny+2]+k[2:nx+1,2:ny+2]),
55      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]),
56      -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]))
57

```

Chap 4

By E. Amani

Benchmark #2: static droplet

- Assessing surface tension computation

 - Analytical solution for pressure difference

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

 - Parasitic current

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

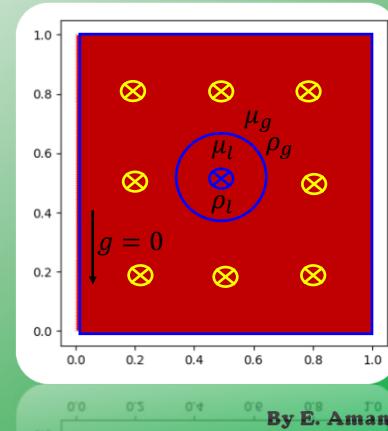
$$\rho_g = 1 \text{ kg/m}^3 \quad \rho_l = 20 \text{ kg/m}^3$$

$$\mu_l = \mu_g = 0.01 \text{ Pa.s} \quad \text{Grid: } 32 \times 32$$

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

$$d = 0.3 \text{ m} \quad \begin{matrix} \text{Young's} \\ \text{geometric} \\ \text{VOF + CSF} \end{matrix}$$

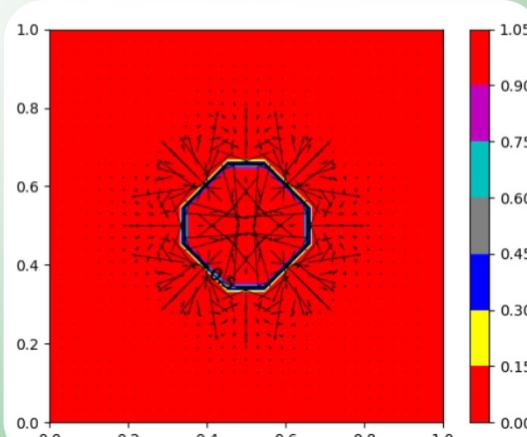
Chap 4



By E. Amani

Benchmark #2: static droplet

- Results



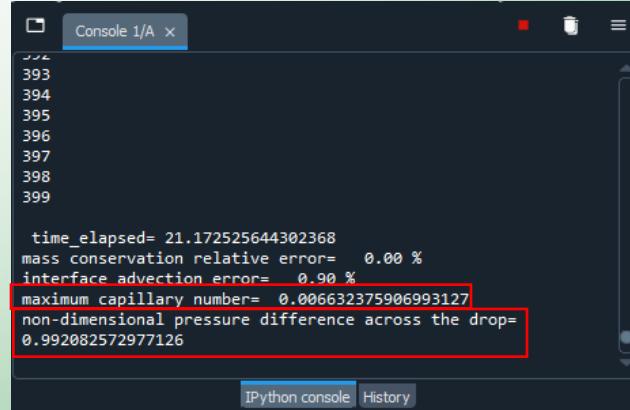
Chap 4

By E. Amani

Benchmark #2: static droplet

- Performance

$t = 0.5 \text{ s}$



```

392
393
394
395
396
397
398
399

time_elapsed= 21.172525644302368
mass conservation relative error=  0.00 %
interface advection error=  0.90 %
maximum capillary number=  0.006632375906993127
non-dimensional pressure difference across the drop=
0.992082572977126

```

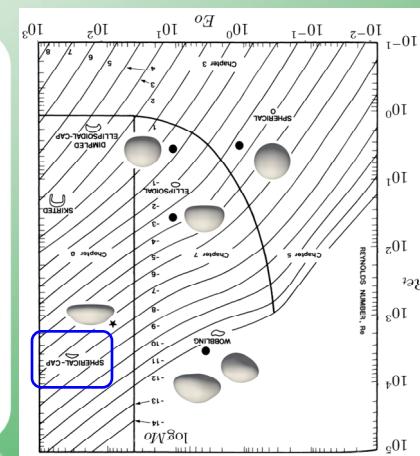
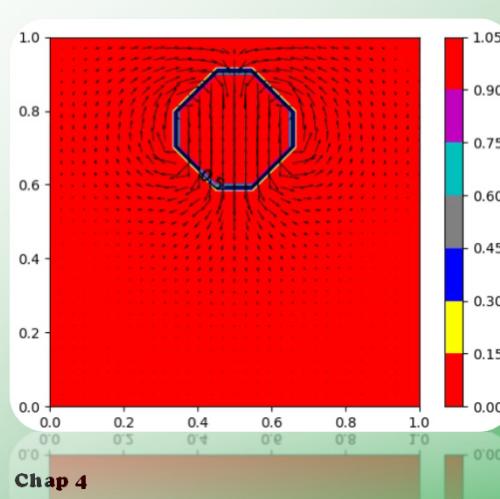
Chap 4

By E. Amani

Coding: NSMF3.py

- Free-falling drop

➤ Young's geometric VOF + CSF



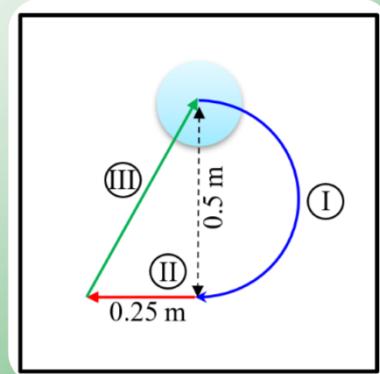
Chap 4

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

0152_III

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

