基于 VOF-PLIC 方法的“回”字型两相流界面运动

姓名：李勇

学号：BA22005035

2022 年 11 月 29 日

1　两相流方法简介

两相流问题在自然界和工业应用中广泛存在，例如雨滴露珠等自然现象或是发动机燃油雾化等工程应用。在两相流中，不同的相由不同的组分构成，各相之间的相互作用复杂，且界面随时间不断变化，甚至可能发生界面合并与破碎等拓扑变形。为准确计算这些两相流动问题，要求我们研究高精度、低成本的两相流算法来捕捉界面流动，以便于我们将其耦合进动量方程中。

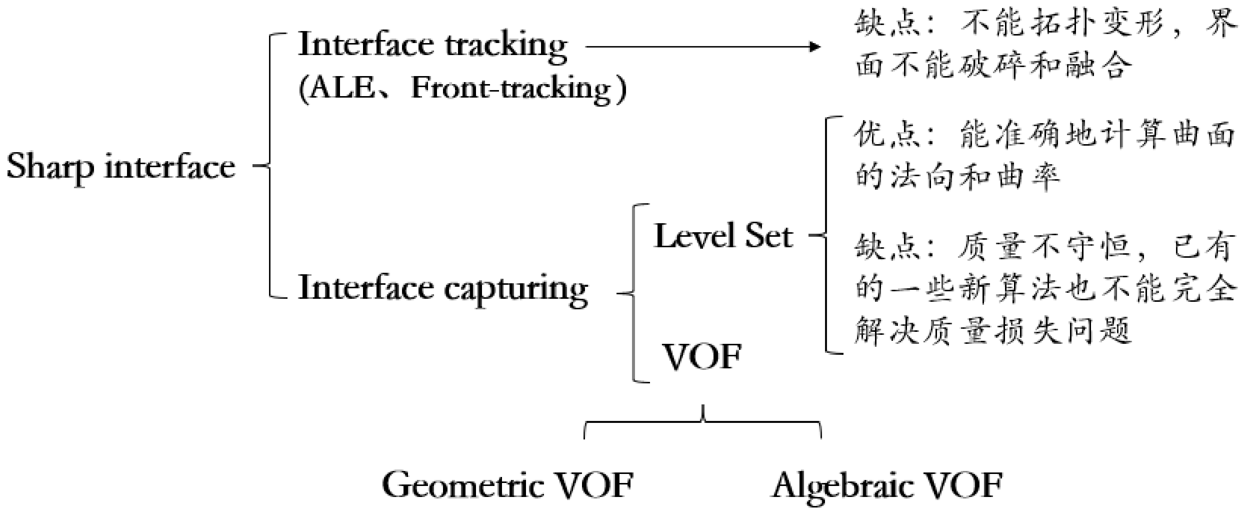


图 1 清晰界面方法分类

主流的清晰界面算法分为 Interface tracking 和 Interface capturing 两类（如图 1 所示）。其中 Interface tracking 方法（例如 Front Tracking method）因为不能拓扑变形，界面不能破碎和融合所以不适用于大变形的界面运动。Interface capturing 方法中最流行的是 VOF 方法和 Level Set 方法，Level Set 方法能准确地计算曲面的法向和曲率但难以保障质量守恒，而 VOF 方法可以自动保持质量守恒，在界面发生拓扑变化时也不要做特殊处理，便于捕捉流动界面。

VOF 方法考虑一个界面浸没在计算网格中，用体积分数来表征界面，C_{ij} 表示 I 相流体在网格内所占的流体体积。根据推进体积分数时采用的方法不同，可以将 VOF 方法分为 Geometric VOF 和 Algebraic VOF。因为 Algebraic VOF 方法采用欧拉方法推进界面，界面容易扩散导致界面变厚且不再清晰，一般需要加入人工压缩，影响计算的精度。Geometric VOF 方法在推进时则分为两步，先在界面所在网格内依据已知的体积分数重构界面，再通过几何方法计算每个

网格内的体积通量，从而推进上一步重构的界面。依据体积分数重新构建界面的方法有SLIC方法和PLIC方法（如图2），SLIC方法是早期的Geometric VOF版本，计算精度不够，且会造成很多背景噪音般的小液滴，PLIC方法的界面由一系列不相连的线段组成，每个线段切割网格后留下的阴影面积与组分值相等，具有较高的精度。本文也采用VOF-PLIC方法来研究两相流界面运动问题。

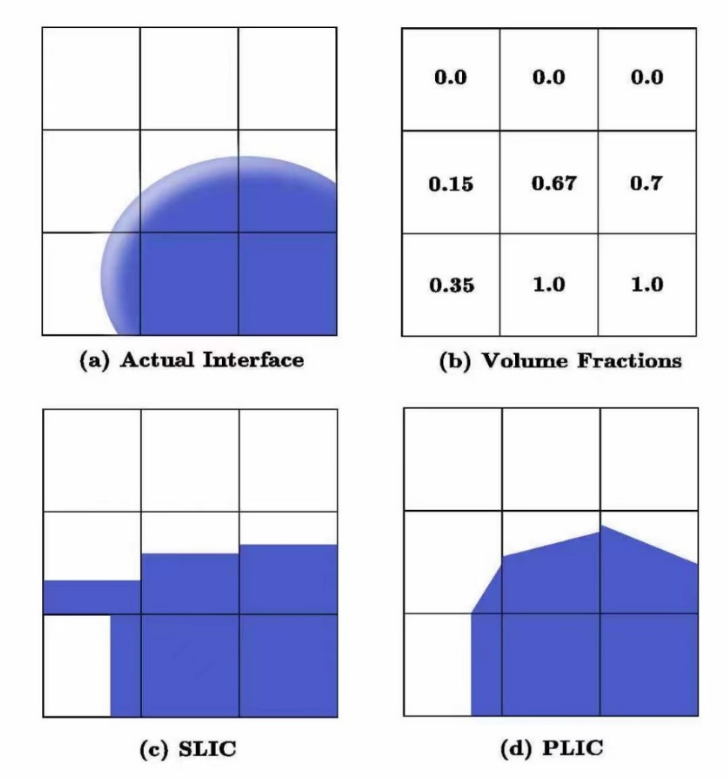


图 2 真实界面与 VOF 方法

2 数值方法

在VOF方法中相场的指标函数由体积分数C给出，表示Ⅰ相流体在网格内所占的流体体积，体积分数C其定义如下：

$$C_{ij} = \iint_{\Omega_{ij}} S(x,y)\,dxdy / V_{ij} \qquad (1)$$

其中 S 为界面位置函数，V_{ij} 是网格体积。在以上定义下可以得到在Ⅰ相流体的网格内体积分数的值全为1，在Ⅱ相流体网格内体积分数的值全为0，在两相界面处体积分数的值为0到1之间，其效果如图2 (b)所示。

$$C_{ij} = \begin{cases} 1 & \text{Ⅰ相流体} \\ 0 & \text{Ⅱ相流体} \end{cases} \qquad (2)$$

2.1 界面重构

VOF-PLIC 方法分为界面重构和界面推进两个部分，界面重构时，用一系列不相连的线段组成界面位置，如图 2 (d)，每个线段切割网格后留下的阴影面积与网格面积的比值（二维问题）与组分体积分数相等。在重构界面时需要先利用周围 C 场的值计算法向方向。以下给出了用中心网格(i,j)周围 9 个网格的信息计算法向量的公式，以 x 方向的分量为例：

现在网格的四个顶点计算法向量：

$$\left.\frac{\partial C}{\partial x}\right|_{i+1/2,\, j+1/2} = \frac{1}{2}\left(C_{i+1,j} - C_{i,j} + C_{i+1,j+1} - C_{i,j+1}\right)/h \qquad (3)$$

中心网格(i,j)处的法向量可由四个顶点处法向量平均得到：

$$\left.\frac{\partial C}{\partial x}\right|_{i,j} = \frac{1}{4}\begin{pmatrix} \left.\frac{\partial C}{\partial x}\right|_{i+1/2,\, j+1/2} + \left.\frac{\partial C}{\partial x}\right|_{i-1/2,\, j+1/2} \\ + \left.\frac{\partial C}{\partial x}\right|_{i+1/2,\, j-1/2} + \left.\frac{\partial C}{\partial x}\right|_{i-1/2,\, j-1/2} \end{pmatrix} \qquad (4)$$

得到法线方向后，构建一定斜率的直线线段去分割网格，如图 3 所示，使得分隔后的体积分数符合 C_{ij} 的值。

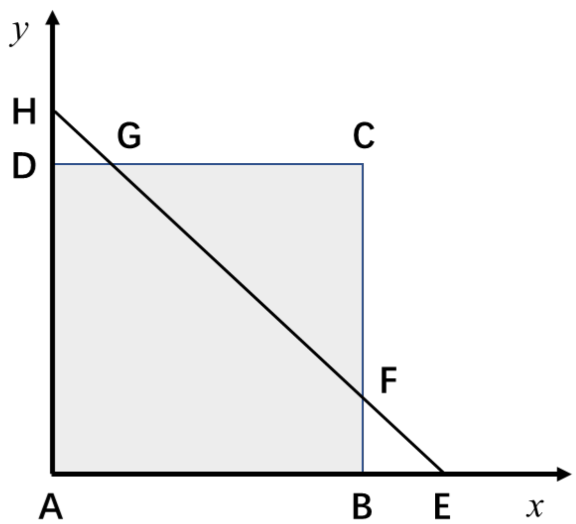


图 3 线段切割网格单位示意图

已知线段法向量后可以构建线段方程（5）：

$$n_x x + n_y y = \alpha\left(C\right) \tag{5}$$

其中 α 为常数，由体积分数确定，使得多边形 ABFGD 的面积为 $C_{ij} \bullet h^2$。由几何关系可得的面积为：

$$S_{ABFGD} = S_{\Delta HAE} - S_{\Delta HDG} - S_{\Delta BEF} \tag{6}$$

从而可以得出 C_{ij} 与 α 之间的一一对应关系：

$$C_{i,j} = \frac{\alpha^2}{2n_x n_y}\left[\begin{array}{l} 1 - H\left(\alpha - n_x dx\right)\left(\dfrac{\alpha - n_x dx}{\alpha}\right)^2 \\ -H\left(\alpha - n_y dy\right)\left(\dfrac{\alpha - n_y dy}{\alpha}\right)^2 \end{array}\right] / \left(dxdy\right) \tag{7}$$

其中 H 函数为 Heaviside Function，其定义如下：

$$H\left(z\right) = \begin{cases} 1 & z > 0 \\ 0 & z \le 0 \end{cases} \tag{8}$$

在已知 C_{ij} 的情况下可以根据方程（7）反求 α。通过平移线段经过几个关键位置来比较 C 值与目标 C_{ij} 的大小，从而确定方程中 H 的值然后解析出 α。

2.2 界面推进

在界面重构中求出线段的 α 后可以利用 Lagrangian 方法将得到的线段（5）分别沿 x 和 y 方向进行单独地推进（即时间分裂方法）。

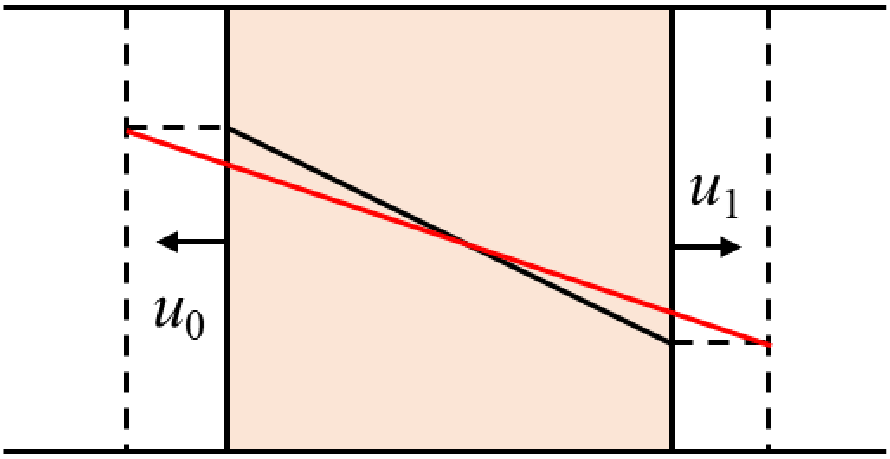


图 4 单个网格内的界面随速度的演化

以 x 方向推进为例，在网格内界面上每点在 x 方向的速度可以采用线性插值来近似，如方程（9）。沿 x 方向推进时 n_x 和 α 会发生变化但 n_y 不会。

$$u\left(x\right)=u_0\left(1-\frac{x}{dx}\right)+u_1\frac{x}{dx} \tag{9}$$

根据方程（9），每个网格内的界面随速度的演化后的效果如图 4 所示，得到的中间*时刻的线段方程：

$$n_x^*x+n_yy=\alpha^* \tag{10}$$

其中：

$$n_x^*=\frac{n_x^n}{1+\left(u_1-u_0\right)\Delta t/dx} \tag{11}$$

$$\alpha^*=\alpha^n+\frac{n_x^nu_0\Delta t}{1+\left(u_1-u_0\right)\Delta t/dx} \tag{12}$$

界面沿 x 方向推进后可能有部分界面进入相邻网格，即产生体积通量。需要根据线段方程（10）通过几何运算计算出流出该网格和留在该网格的体积通量。取向左的体积通量为 $C_L(i, j)$，向右的体积通量为 $C_R(i, j)$，留在网格的体积分数为 $C_S(i, j)$，则可以得出中间*时刻全程的体积分数：

$$C_{ij}^*=C_S\left(i,j\right)+C_L\left(i,j\right)+C_R\left(i,j\right) \tag{13}$$

在沿 y 方向进行推进时，需要根据 C_{ij}^* 重新计算 n_x、n_y 和 α，再依据上面相同的步骤更新 C_{ij}^{n+1}。即从 n 时刻推进到 n+1 时刻需要两次界面重构和两次界面推进（二维问题，三维是三次重构三次推进）。

对于一些特定的构型（如网格不在界面上），我们可以直接通过下式给出流向左边和流向右边的体积通量。当网格处于 II 相时有：

$$C_S\left(i,j\right)=C_L\left(i,j\right)=C_R\left(i,j\right)=0 \tag{14}$$

当网格处于 I 相时有：

$$C_R(i,j)=\max\left(\frac{u_1\Delta t}{dx},0\right) \qquad (15)$$

$$C_L(i,j)=\max\left(-\frac{u_0\Delta t}{dx},0\right) \qquad (16)$$

$$C_S(i,j)=1-\max\left(\frac{u_0\Delta t}{dx},0\right)+\min\left(\frac{u_1\Delta t}{dx},0\right) \qquad (16)$$

以上方法是处理法向量在第一象限时的方法，当法向量在其他象限时，需要根据几何的对称性将法向量投影到第一象限。以落在第三象限的法向量为例，将法向量翻转到第一象限时，n_x 和 n_y 都要变号，相应的 u 和 v 也要变号，此外 CL 与 CR 也要进行交换。

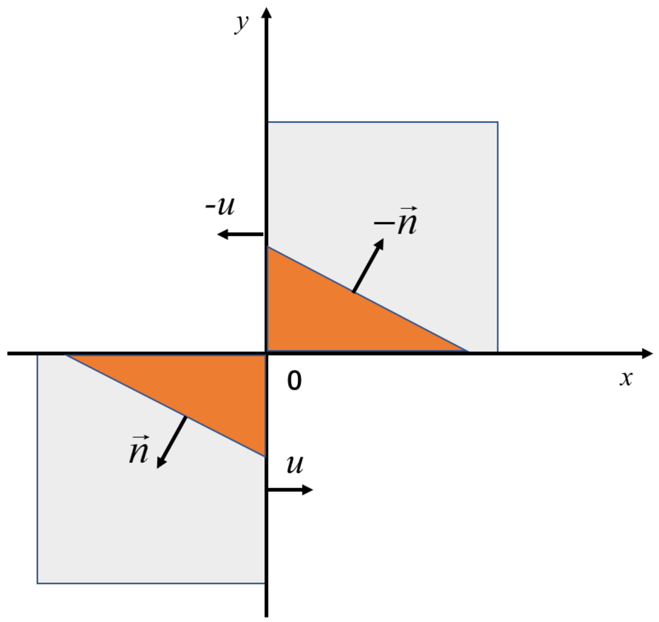


图 5 翻转法向量

在极特殊情况中会发生 C < 0 或 C > 1，这显然是非物理的，故在 n 时刻推进到 n+1 时刻出现这种情况时，需要以下处理方法避免这样情况。

$$C_{i,j}=\min\left[1,\max\left(0,C_{i,j}\right)\right] \qquad (15)$$

3　问题描述

本文对“回”字型的两相界面运动问题进行数值计算，取 1×1 计算域内左上角的“回”字型的界面（如图 6 所示），在均匀来流(u = 1, v = -0.5)下的运动

情况，计算域的上下左右边界为周期性边界条件。计算预期是在一个周期下界面与初始时刻的界面重合。

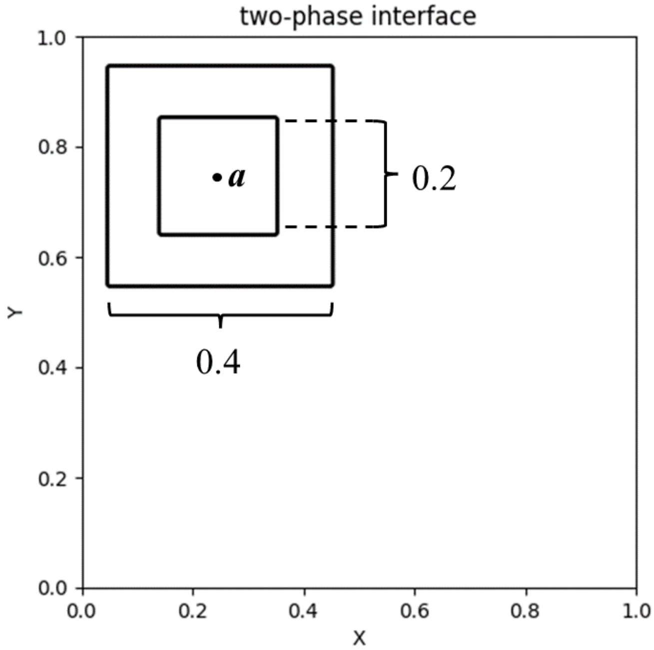


图 6 界面运动初始条件

界面初始位置如上图所示，相关参数如下：

计算域：1×1

正方形中点 a 坐标：(0.25, 0.75)

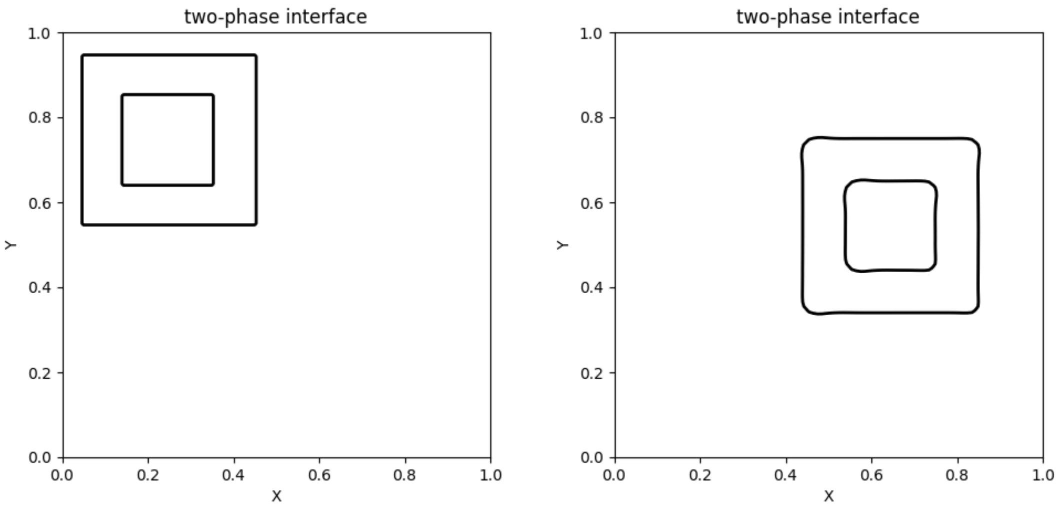
均匀来流：u = 1.0, v = -0.5

周期 T = 2.0

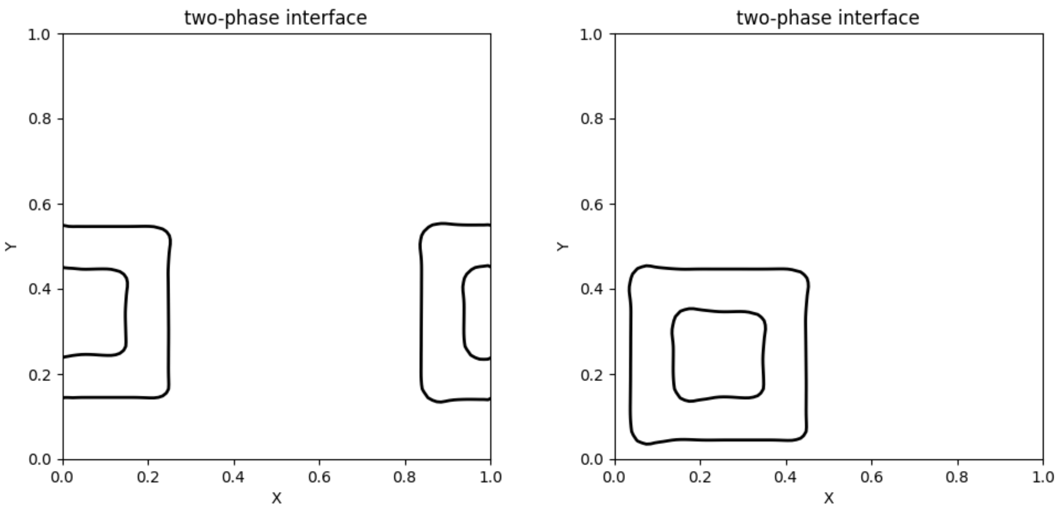
4　结果与讨论

4.1　不同时刻界面演化

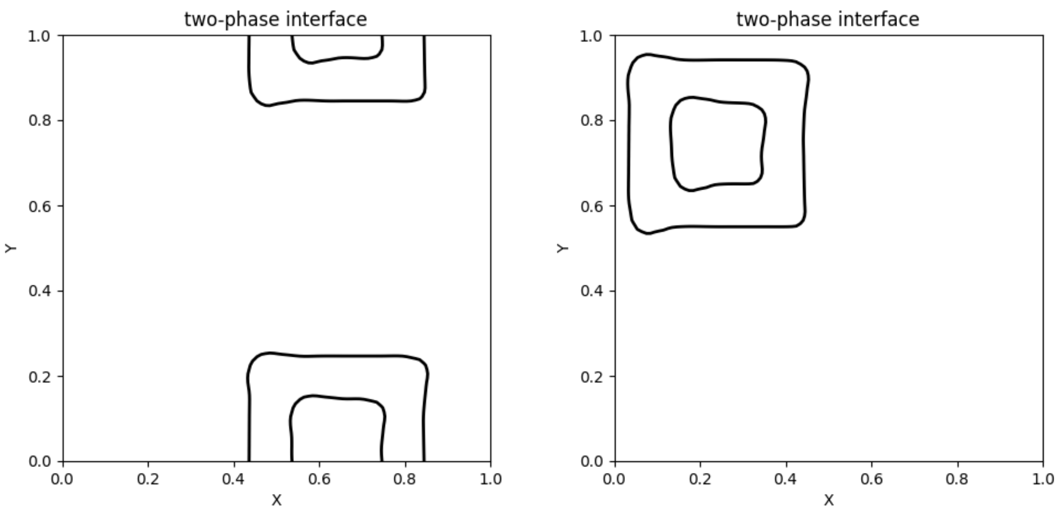
图 7 分别给出了 100×100 网格下 t = 0、0.2T、0.4T、0.5T、0.7T、T 时刻下的界面形状。可以发现界面在均匀来流(u = 1, v = -0.5)的作用下不断向右下角运动，在遇到周期性边界时从左边界回到计算域。随着时间的推进，回字型界面的形状不再维持原状，尤其是四个顶角不再变得尖锐。可以从各个时刻的界面形状看出四个顶角有不同程度地突出，这可能与程序中修改的界面推进算法有关，使得体积通量总比预期的要大一点。



（左：t = 0T，右：t = 0.2T）



（左：t = 0.4T，右：t = 0.5T）

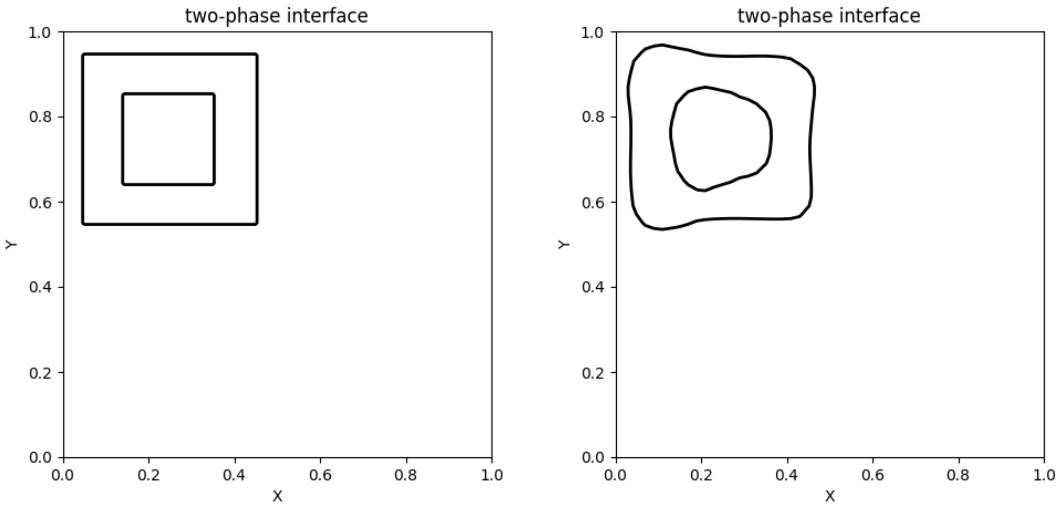


（左：t = 0.7T，右：t = 1T）

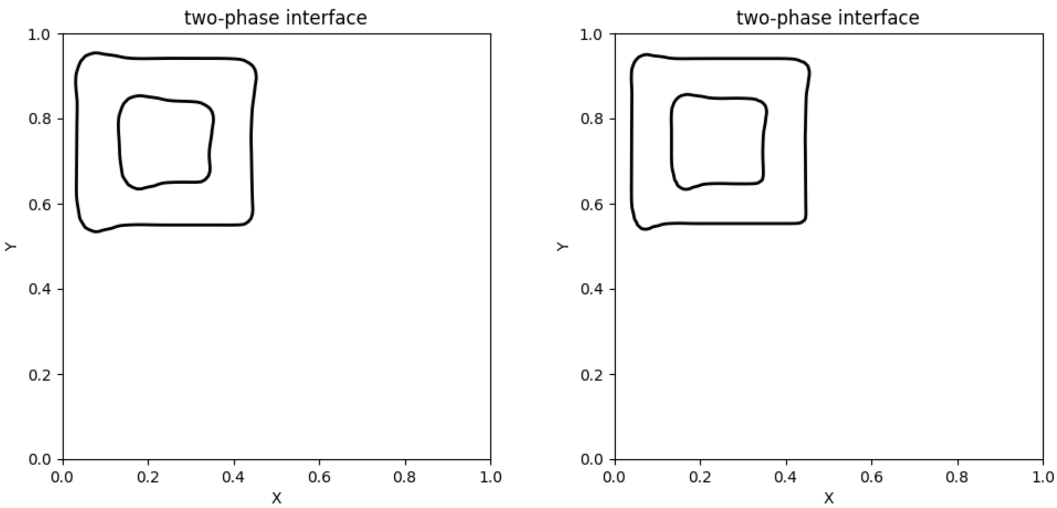
图 7 不同时刻界面演化

4.2　网格无关性验证

为了验证计算结果独立于网格，图 8 给出了 50×50、100×100 网格和 150×150 网格的计算结果，对比一个周期 T = 2.0 后的界面变化。从图中可以看出 100×100 网格下的计算结果已经可以较好地独立于网格，更细的网格会在左侧的顶角处处理得更好。



（左：初始条件，右：50×50）



（左 100×100，右 150×150）

图 8 网格无关性验证

5 结论

本文采用 Python 编程语言实现了课堂上丁老师讲授的 VOP-PLIC 算法，对“回”字型两相流界面运动进行了数值计算，通过分析不同时刻两相界面运动的情况判定了算法实现的有效性，检验了计算结果对网格的依赖性。

通过本次大作业的练习，本人学习到了很多复现数值算法的方法，包括在算法上如何在计算域中实施周期性边界条件、如何采用时间分裂法推进界面运动等，包括在编写程序时如何调用 debug 工具调试代码，检查不同变量或是函数存在的问题等等。最重要的是通过本次大作业对算法和程序的练习与思考，建立起了对复现论文算法的信心。

6 参考文献

[1] Ashgriz N , Poo J Y . FLAIR: Flux line-segment model for advection and interface reconstruction[J]. Journal of Computational Physics, 1991, 93(2):449-468.

[2] Parker B , Youngs D . Two and three dimensional Eulerian simulation of fluid flow with material interfaces[J]. 1992.

[3] Gueyffier D , Jie L , Nadim A , et al. Volume-of-Fluid Interface Tracking with Smoothed Surface Stress Methods for Three-Dimensional Flows[J]. Journal of Computational Physics, 1999, 152(2):423-456.

[4] Kawano, Akio. A simple volume-of-fluid reconstruction method for three-dimensional two-phase flows[J]. Computers & Fluids, 2016.

7 代码附件

本文的 VOF-PLIC 算法由 Python 语言实现，不同功能分文件进行编写，各文件功能如下：

文件名	文件功能
main.py	主文件，程序入口
calc_normal_vecter.py	根据体积分数 C 计算出界面的法向量 nx 和 ny
interface_reconstruction.py	重构界面，利用 C 与 nx, ny 计算出线段 nx*x+ny*y=alpha 的 alpha 值
interface_propagation.py	界面推进，先计算体积通量 CL、CR 和 CS，再沿 x 方向和 y 方向推进 C
myplot.py	自定义绘图

main.py 文件：

```
import numpy as np
# from matplotlib import pyplot as plt
import myplot
import calc_normal_vecter as nxy
import interface_reconstruction as reconstruction
import interface_propagation as propagation

# Control parameters
xmin = 0
xmax = 1
ymin = 0
ymax = 1
m = 100  # number of cells in every direction
dx = (xmax - xmin) / m  # Size of 1 grid cell
dy = (ymax - ymin) / m
t = 0.  # Initial time
T = 0.4  # Final time
CFL = 0.01  # CFL 数
dt = CFL * dx  # Time step
print("dt=", dt)

# Assign initial conditions
x = np.arange(xmin - dx / 2, xmax + dx / 2 * 3, dx)  # 边界外 1 个 ghost cells
y = np.arange(ymin - dy / 2, ymax + dy / 2 * 3, dy)
# x = np.arange(xmin-3*dx/2, xmax+5*dx/2, dx)  # 边界外 2 个 ghost cells
# y = np.arange(ymin-3*dy/2, ymax+5*dy/2, dy)
print('包含 ghost cells 的网格数量: ', len(x))  # 统计包含 ghost cells 的网格数量
# u = np.zeros((len(y), len(x)))  # len(y)是行数，len(x)列数 在速度发生变化的流场
中采用
# v = np.zeros((len(y), len(x)))
u = 1.0
v = -0.5
C = np.zeros((len(y), len(x)))
# 数组切片后没有生成新的数组，因为切片是右开区间，所以要加上 1， # index = (x+dx/2)/dx
C[int((0.55 + dy / 2) / dy):int((0.95 + dy / 2) / dy + 1), int((0.05 + dx /
2) / dx):int((0.45 + dx / 2) / dx + 1)] = 1
C[int((0.65 + dy / 2) / dy):int((0.85 + dy / 2) / dy + 1), int((0.15 + dx /
2) / dx):int((0.35 + dx / 2) / dx + 1)] = 0

# Plot initial conditions
myplot.plotC(x, y, C)
myplot.plotContour(x, y, C)

while t < T:
    # Modify the code so that the last step is adjusted to exactly reach T
    if t + dt > T:
        dt = T - t

    # 计算界面的法向量 nx 与 ny
    nx, ny = nxy.calcnxy(dx, dy, C)
    # myplot.plotValue(x,y,nx)

    # interface reconstruction 求出 alhpa
    alpha = reconstruction.calcAlpha(x, y, dx, dy, C, nx, ny)
    # myplot.plotValue(x,y,alpha)

    # interface propagation 求出下一时刻的 C
    # 沿 x 方向求出 Cstar
```

```python
    nx, ny = nxy.calcnxy(dx, dy, C)   # 深浅拷贝问题导致 nx 与 ny 需要重新指向
    Cstar = propagation.calcCstar(nx, ny, u, v, dt, dx, dy, alpha, x, y, C)
    # myplot.plotValue(x,y,Cstar)
    # myplot.plotContour(x,y,Cstar)

    # 沿 y 方向求出下一时刻 C
    C = Cstar.copy()
    nx, ny = nxy.calcnxy(dx, dy, C)
    alpha = reconstruction.calcAlpha(x, y, dx, dy, C, nx, ny)
    nx, ny = nxy.calcnxy(dx, dy, C)
    C_nplus1 = propagation.calcC_nplus1(nx, ny, u, v, dt, dx, dy, alpha, x, y, C)

    # C = C_nplus1.copy()
    C = C_nplus1.copy()
    t = t + dt
    print("总时间{0}，目前时间{1}".format(T, t))

# 后处理
myplot.plotContour(x, y, C)
myplot.plotC(x, y, C)
```

calc_normal_vecter.py 文件：

```python
# 根据体积分数 C 计算出界面的法向量 nx 和 ny

import numpy as np

"""
对网格作如下定义
up, down, left and right
        lu ——— ru
        |       |
        |       |
        ld ——— rd
"""


def calcnxy(dx, dy, C):
    # 定义中间变量
  lu = np.empty_like(C)
    ru = np.empty_like(C)
    ld = np.empty_like(C)
    rd = np.empty_like(C)
    nx = np.zeros_like(C)
    ny = np.zeros_like(C)

    # 计算 x 方向的法向量 nx = dC/dx
    ru[1:-1, 1:-1] = 0.5 * (C[1:-1, 2:] - C[1:-1, 1:-1] + C[2:, 2:] - C[2:, 1:-1]) / dx
    lu[1:-1, 1:-1] = 0.5 * (C[1:-1, 1:-1] - C[1:-1, 0:-2] + C[2:, 1:-1] - C[2:, 0:-2]) / dx
    ld[1:-1, 1:-1] = 0.5 * (C[0:-2, 1:-1] - C[0:-2, 0:-2] + C[1:-1, 1:-1] - C[1:-1, 0:-2]) / dx
    rd[1:-1, 1:-1] = 0.5 * (C[0:-2, 2:] - C[0:-2, 1:-1] + C[1:-1, 2:] - C[1:-1, 1:-1]) / dx
    nx[1:-1, 1:-1] = (ru[1:-1, 1:-1] + lu[1:-1, 1:-1] + ld[1:-1, 1:-1] +
```

```
rd[1:-1, 1:-1]) / 4

    # 计算 y 方向的法向量 ny = dC/dy
    ru[1:-1, 1:-1] = 0.5 * (-C[1:-1, 2:] - C[1:-1, 1:-1] + C[2:, 2:] + C[2:, 
1:-1]) / dy
    lu[1:-1, 1:-1] = 0.5 * (-C[1:-1, 1:-1] - C[1:-1, 0:-2] + C[2:, 1:-1] + 
C[2:, 0:-2]) / dy
    ld[1:-1, 1:-1] = 0.5 * (-C[0:-2, 1:-1] - C[0:-2, 0:-2] + C[1:-1, 1:-1] + 
C[1:-1, 0:-2]) / dy
    rd[1:-1, 1:-1] = 0.5 * (-C[0:-2, 2:] - C[0:-2, 1:-1] + C[1:-1, 2:] + 
C[1:-1, 1:-1]) / dy
    ny[1:-1, 1:-1] = (ru[1:-1, 1:-1] + lu[1:-1, 1:-1] + ld[1:-1, 1:-1] + 
rd[1:-1, 1:-1]) / 4

    # 更新边界条件，对于 nx 与 ny 的计算是否省略更新边界条件取决于下一步计算是否用到
nx[0, :]的值
  nx[0, :] = nx[-2, :]   # 第 0 行，列是[:]
    nx[-1, :] = nx[1, :]
    nx[:, 0] = nx[:, -2]
    nx[:, -1] = nx[:, 1]

    ny[0, :] = ny[-2, :]   # 第 0 行，列是[:]
    ny[-1, :] = ny[1, :]
    ny[:, 0] = ny[:, -1]
    ny[:, -1] = ny[:, 1]

    return -nx, -ny

"""
调用该模块的方式：
import calc_normal_vecter as nxy
nx, ny = nxy.calcnxy(dx, dy, C)
"""
```

interface_reconstruction.py 文件：

```
# 重构界面，利用 C 与 nx, ny 计算出线段 nx*x+ny*y=alpha 的 alpha 值

import numpy as np
import math

# 定义一个依据 alpha 求 C 的函数，其中 np.heaviside(alpha-nx*dx,0.0)第二个参数的意思是
当第一个参数等于 0 时，取第二参数的值
def calcC(alpha, dx, dy, nx, ny):   # 传参时传入 nx[j, i], alpha[j, i]
    alpha += 0.000001
    nx += 0.000001
    ny += 0.000001
    C = alpha**2/(2*nx*ny)*(1-np.heaviside(alpha-nx*dx,0.0)*((alpha-
nx*dx)/alpha)**2 \
                            -np.heaviside(alpha-ny*dy,0.0)*((alpha-
ny*dy)/alpha)**2)/(dx*dy)
    return C

def calcAlpha(x, y, dx, dy, C, nx, ny):
    # 定义中间变量
  alpha = np.zeros_like(C)
    minima = 0.000001
```

```
    # 计算每个网格的 alpha 值
    for j in range(0, len(y)):
        for i in range(0, len(x)):
            # 当法向量不是在第一象限时，将向量投影在第一象限
  # 为了保证方程不变，当存在 nx 或 ny 变号时，更改相应的 u 或 v，即 u*nx=(-u)*(-nx)

            if nx[j, i] < 0:
                nx[j, i] = -nx[j, i]
                # u = -u
            if ny[j, i] < 0:
                ny[j, i] = -ny[j, i]
                # v = -v

            if minima< C[j, i] < 1.0-minima:
                nx[j, i] += minima
                ny[j, i] += minima
                alpha[j, i] += minima
                #C[j, i] += minima
                case = 0

                # alpha1 线段过 D 点
        alpha1 = ny[j, i] * dy
                C1 = calcC(alpha1, dx, dy, nx[j, i], ny[j, i])
                # alpha2 线段过 B 点
        alpha2 = nx[j, i] * dx
                C2 = calcC(alpha2, dx, dy, nx[j, i], ny[j, i])

                # 判断两个 heaviside function 的值并计算 alpha 的值
        if C[j, i] <= min(C1, C2):
                    # H1, H2 = 0.0, 0.0
                    # alpha[j, i] = math.sqrt(2 * nx[j, i] * ny[j, i] * dx *
dy * math.fabs(C[j, i]))
                    alpha[j, i] = math.sqrt(2 * nx[j, i] * ny[j, i] * dx *
dy * C[j, i])
                    case = 1
                elif C[j, i] >= max(C1, C2):
                    # H1, H2 = 1.0, 1.0
                    case = 2
                    #求根公式法
         a = -1.0
                    b = 2 * (nx[j, i] * dx + ny[j, i] * dy)
                    c = -(2 * nx[j, i] * ny[j, i] * dx * dy * C[j, i] +
(nx[j, i] * dx) ** 2 + (ny[j, i] * dy) ** 2)
                    # alpha[j, i] = (-b + math.sqrt(b ** 2 - 4 * a * c)) /
(2 * a)
                    if (b**2-4*a*c) > 0.0:
                        alpha[j, i] = (-b+math.sqrt(b**2-4*a*c))/(2*a)   # 洛
通说+改为-更合适
         #else:
                        #alpha[j, i] = minima
                    """ # 迭代法
                    alpha11 = max(alpha1, alpha2)
                    alpha22 = nx[j, i]*dx + ny[j, i]*dy
                    for i in range(2000000):
                        alphat = 0.5*(alpha11+alpha22)
                        #alphat += minima
                        Ct = calcC(alphat, dx, dy, nx[j, i], ny[j, i])
                        if(math.fabs((Ct - C[j, i])<minima)):
                            alpha[j, i] = alphat
```

```
                            break
                        if(Ct>C[j, i]):
                            alpha22 = alphat
                        else:
                            alpha11 = alphat
                    """

                elif C2 < C[j, i] < C1:
                    # H1, H2 = 1.0, 0.0
                    # alpha[j, i] = (2*nx[j, i]*ny[j, i]*dx*dy*C[j, i]+(nx[j, i]*dx)**2)/(2*nx[j, i]*dx)
                    alpha[j, i] = (2 * ny[j, i] * dy * C[j, i] + nx[j, i] * dx) / 2
                    case = 3
                elif C1 < C[j, i] < C2:
                    # H1, H2 = 0.0, 1.0
                    # alpha[j, i] = (2*nx[j, i]*ny[j, i]*dx*dy*C[j, i]+(ny[j, i]*dy)**2)/(2*ny[j, i]*dy)
                    alpha[j, i] = (2 * nx[j, i] * dx * C[j, i] + ny[j, i] * dy) / 2
                    case = 4

                # 以下代码用于验证界面重构代码的准确性
       #"""
                #print("-----判断alpha---------")
                if math.fabs(C[j, i] - calcC(alpha[j, i], dx, dy, nx[j, i], ny[j, i])) > 0.01:
                    print("请注意：重构的alpha[]可能计算有误")
                    print(j, i, C[j, i],alpha[j,i])
                    print(calcC(alpha[j, i], dx, dy, nx[j, i], ny[j, i]))
                    print("case=",case)
                    #print("C1, C2",C1, C2)
                #"""
            else:
                alpha[j, i] = 0.0

    return alpha
```

interface_propagation.py 文件:

```
# 界面推进，先计算体积通量 CL、CR 和 CS，再沿 x 方向和 y 方向推进 C

# import myplot
import numpy as np
import math
# from pynverse import inversefunc  # 计算反函数的库

def calcy(alpha, nx, ny, x):  # 传参时传入 nx[j, i]，alpha[j, i]
    ny += 0.000001
    y = (alpha - nx*x)/ny
    return y

def calcx(alpha, nx, ny, y):
    nx += 0.000001
    x = (alpha-ny*y)/nx
    return x

def calcCstar(nx,ny,u,v,dt,dx,dy,alpha,x,y,C):
```

```
    # 定义中间变量
  minima = 0.00001
    CL = np.zeros_like(C)
    CS = np.zeros_like(C)
    CR = np.zeros_like(C)
    Cstar = np.zeros_like(C)

    for j in range(0, len(y)):
        for i in range(0, len(x)):
            u0 = u
            u1 = u

            if nx[j, i] < 0.0:
                flag = 1   # 计算 CL 和 CR 后根据 flag 判断是否交换
      u0, u1 = -u1, -u0   # 当 nx<0 时反向
     else:
                flag = 0

            if nx[j, i] < 0:   # nx 与 ny 取正值
      nx[j, i] = -nx[j, i]
            if ny[j, i] < 0:
                ny[j, i] = -ny[j, i]

            if C[j, i] > 1.0 - minima:   # C=1 的情况
      CL[j, i] = max(-u * dt / dx, 0)
                # CL[j, i] = 0.0
                CS[j, i] = 1.0 - max(u * dt / dx, 0) + min(u * dt / dx, 0)
                CR[j, i] = max(u * dt / dx, 0)
            elif C[j, i] < minima:   # C=0 的情况
      CL[j, i] = 0.0
                CS[j, i] = 0.0
                CR[j, i] = 0.0
            else:   # 0<C<1 的情况
      if math.fabs(ny[j, i]) < minima:   # ny=0 的情况
        if u1 > 0.0:
                        CL[j, i] = max(-u0 * dt / dx, 0)
                        x1 = C[j, i] * dx * dy / dy
                        if (x1 + u1 * dt >= dx):
                            CS[j, i] = 1.0 - max(u0 * dt / dx, 0)
                            CR[j, i] = (x1 + u1 * dt - dx) / dx
                        else:
                            CS[j, i] = 1.0 - max(u0 * dt / dx, 0) - (dx -
(x1 + u1 * dt)) / dx
                            CR[j, i] = 0.0
                    else:
                        CR[j, i] = 0.0
                        x1 = C[j, i] * dx * dy / dy
                        x2 = x1 + u1*dt
                        if x2 < 0.0:
                            CL[j, i] = C[j, i]
                            CS[j, i] = 0.0
                        else:
                            s = x2*dy
                            CS[j, i] = s/(dx*dy)
                            CL[j, i] = C[j, i] - CS[j, i]
                            #"""
                elif math.fabs(nx[j, i]) < minima:   # nx=0 的情况
        CR[j, i] = max(C[j, i] * (dx * dy) / dx * u1 * dt / (dx * dy), 0)
                    CL[j, i] = max(-C[j, i] * (dx * dy) / dx * u0 * dt / (dx
```

```
* dy), 0)
                    CS[j, i] = C[j, i] - CR[j, i] - CL[j, i]
                else:  # nx>0 且 ny>0 的情况
        """
                    # 计算出 t=star 时刻的线段
                    nx_star = nx[j, i] / (1.0 + (u1 - u0) * dt / dx)
                    alpha_star = alpha[j, i] + (nx[j, i] * u0 * dt) / (1.0 +
(u1 - u0) * dt / dx)

                    OH = calcy(alpha_star, nx_star, ny[j, i], 0.0)
                    CL[j, i] = max(-min(OH, dy)*u0*dt/(dx*dy), 0.0)
                    BF = calcy(alpha_star, nx_star, ny[j, i], dx)
                    CR[j, i] = max(max(BF,0.0)*u1*dt/(dx*dy),0.0)
                    CS[j, i] = C[j, i] - CR[j, i] - CL[j, i]
                    """
                    OH = calcy(alpha[j, i], nx[j, i], ny[j, i], 0.0)
                    CL[j, i] = max(-min(OH, dy) * u0 * dt / (dx * dy), 0.0)
                    BF = calcy(alpha[j, i], nx[j, i], ny[j, i], dx)
                    CR[j, i] = max(max(BF, 0.0) * u1 * dt / (dx * dy), 0.0)
                    CS[j, i] = C[j, i] - CR[j, i] - CL[j, i]

                if flag:
                    CL[j, i], CR[j, i] = CR[j, i], CL[j, i]  # 当(nx, ny)不在
第一象限时进行交换 CL 和 CR


    #myplot.plotContour(x, y, CS)
    #myplot.plotContour(x,y,CR)
    Cstar[1:-1, 1:-1] = CS[1:-1, 1:-1] + CR[1:-1, 0:-2] + CL[1:-1, 2:]  # 更
新 C
    Cstar = np.minimum(1.0, np.maximum(0.0, Cstar))  # 清理大于 1 或小于 0 的 C

    # 周期性边界条件
  Cstar[0, :] = Cstar[-2, :]  # 第 0 行, 列是[:]
    Cstar[-1, :] = Cstar[1, :]
    Cstar[:, 0] = Cstar[:, -2]
    Cstar[:, -1] = Cstar[:, 1]

    return Cstar


def calcC_nplus1(nx,ny,u,v,dt,dx,dy,alpha,x,y,C):  # 与上面的函数同理
  # 定义中间变量
  minima = 0.001
    CD = np.zeros_like(C)
    CS = np.zeros_like(C)
    CU = np.zeros_like(C)
    C_nplus1 = np.zeros_like(C)

    for j in range(0, len(y)):
        for i in range(0, len(x)):
            v0 = v
            v1 = v

            if ny[j, i] < 0.0:
                flag = 1
                v0, v1 = -v1, -v0
                # print("反向")
            else:
```

```
                flag = 0

            if nx[j, i] < 0:
                nx[j, i] = -nx[j, i]
                #u = -u
            if ny[j, i] < 0:
                ny[j, i] = -ny[j, i]
                #v = -v

            if C[j, i] > 1.0 - minima:
                CD[j, i] = max(-v * dt / dy, 0)
                # CS[j, i] = 0.0
                CS[j, i] = 1.0 - max(v * dt / dy, 0) + min(v * dt / dy, 0)
                CU[j, i] = max(v * dt / dx, 0)
            elif C[j, i] < minima:
                CD[j, i] = 0.0
                CS[j, i] = 0.0
                CU[j, i] = 0.0
            else:
                if math.fabs(nx[j, i]) < minima:
                    if v1 > 0.0:
                        CD[j, i] = max(-v0 * dt / dy, 0)
                        y1 = C[j, i] * dx * dy / dx
                        if (y1 + v1 * dt >= dy):
                            CS[j, i] = 1.0 - max(v0 * dt / dy, 0)
                            CU[j, i] = (y1 + v1 * dt - dy) / dy
                        else:
                            CS[j, i] = 1.0 - max(v0 * dt / dy, 0) - (dy -
(y1 + v1 * dt)) / dy
                            CU[j, i] = 0.0
                    else:
                        CU[j, i] = 0.0
                        y1 = C[j, i] * dx * dy / dx
                        y2 = y1 + v1*dt
                        if y2 < 0.0:
                            CD[j, i] = C[j, i]
                            CS[j, i] = 0.0
                        else:
                            s = y2*dx
                            CS[j, i] = s/(dx*dy)
                            CD[j, i] = C[j, i] - CS[j, i]
                elif math.fabs(ny[j, i]) < minima:
                    CU[j, i] = max(C[j, i] * (dx * dy) / dy * v1 * dt / (dx
* dy), 0)
                    CD[j, i] = max(-C[j, i] * (dx * dy) / dy * v0 * dt / (dx
* dy), 0)
                    CS[j, i] = C[j, i] - CU[j, i] - CD[j, i]
                else:
                    #OH = calcy(alpha[j, i], nx[j, i], ny[j, i], 0.0)
                    AB = calcx(alpha[j, i], nx[j, i], ny[j, i], 0.0)
                    #CL[j, i] = max(-min(OH, dy) * u0 * dt / (dx * dy), 0.0)
                    CD[j, i] = max(-min(AB, dx)*v0*dt/(dx*dy),0.0)
                    DG = calcx(alpha[j, i], nx[j, i], ny[j, i], dy)
                    CU[j, i] = max(max(DG,0.0)*v1*dt/(dx*dy),0.0)
                    CS[j, i] = C[j, i] - CD[j, i] - CU[j, i]

                if flag:
                    CD[j, i], CU[j, i] = CU[j, i], CD[j, i]

    #myplot.plotContour(x, y, CS)
    #myplot.plotContour(x,y,CD)
```

```python
    C_nplus1[1:-1, 1:-1] = CS[1:-1, 1:-1] + CU[0:-2, 1:-1] + CD[2:, 1:-1]
    C_nplus1 = np.minimum(1.0, np.maximum(0.0, C_nplus1))

    # 周期性边界条件
  C_nplus1[0, :] = C_nplus1[-2, :]  # 第 0 行，列是[:]
    C_nplus1[-1, :] = C_nplus1[1, :]
    C_nplus1[:, 0] = C_nplus1[:, -2]
    C_nplus1[:, -1] = C_nplus1[:, 1]

    return C_nplus1
```

myplot.py 文件：

```python
import numpy as np
from matplotlib import pyplot as plt

"""
画等高线参考代码
plt.contour(x1, x2, z, colors=list('kbrbk'), linestyles=['--', '--', '-', '-
-', '--'],
                        linewidths=[1, 0.5, 1.5, 0.5, 1], levels=[-1, -0.5, 0,
0.5, 1])
以上是画 5 条等值线，并指定每条轮廓线的颜色、线型、线宽和值，当只需要一个值时(例如画 Levl
set 的 0 等值线)，[]只需要取一个值即可

"""

def plotC(x, y, C):
    plt.figure(figsize=(5, 5), dpi=100)
    X, Y = np.meshgrid(x, y)
    # plt.contourf(X, Y, C[:], cmap="rainbow")
    plt.contour(X, Y, C[:], colors='k', linewidths=[2.0], levels=[0.5])
    plt.title("two-phase interface")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()

def plotContour(x, y, C):
    plt.figure(figsize=(6, 5), dpi=100)
    X, Y = np.meshgrid(x, y)
    plt.contourf(X, Y, C[:], cmap="rainbow")
    # plt.grid()
    plt.colorbar()
    plt.title("volume fraction contour")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()
```