基于亚网格修正的 Level Set 方法
两相界面旋转剪切流动

姓名：李勇

学号：BA22005035

2022 年 12 月 10 日

1 Level set 方法简介

与 VOF 方法一样，Level set 方法也是目前主流的界面追踪方法之一，同样属于隐式界面方法，界面可以定义为某个标量函数的等值面（或等值线），在 Level set 方法中描述界面位置和形状的标量函数为符号距离函数。如图 1 所示，当符号距离函数$\phi < 0$时为第一相流体，当$\phi > 0$时为第二相流体，$\phi = 0$处为界面位置。

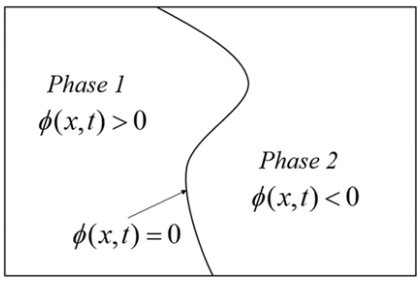


图 1 符号距离函数ϕ及其对应的空间区域

Level set 方法的优点主要体现在以下几点：在界面发生拓扑几何变化时（如液滴的破碎和融合），Level set 方法不需要做任何特殊处理就能直接计算；算法方便从低维模拟推广至高维；适合做局域化计算（MPI 并行计算）；Level set 方法计算得到的界面法向量和界面曲率更加准确。相比于 VOF 方法采用体积分数作为标量函数，Level set 方法标量函数为符号距离函数。VOF 方法中界面处的体积分数仅用几个网格从 0 跨越到 1，使得体积分数在界面处是不连续的，Level set 方法标量函数的采用符号距离函数，其跨越界面时标量场是光滑连续的，故计算得到的界面法向量和界面曲率更加准确。

若界面的位置为$\vec{x}_I$，则计算域内任意位置到界面的最短距离为：

$$d(\vec{x}) = \min\left(\left|\vec{x} - \vec{x}_I\right|\right) \tag{1}$$

$d(\vec{x})$称为欧几里得距离，显然$d(\vec{x}) = 0$为界面位置。距离函数有个很重要的特性，即在界面的一侧时，其梯度的模恒等于 1。

$$\left|\nabla d\right| = 1 \tag{2}$$

为了区分界面两边，且保障符号距离函数ϕ跨越界面时是连续的，可以定义界面一侧为Ω^+，另一侧为Ω^-，则符号距离函数可以写为:

$$\phi(\vec{x}) = \begin{cases} d(\vec{x}), \ \vec{x} \in \Omega^+ \\ -d(\vec{x}), \vec{x} \in \Omega^- \end{cases} \tag{3}$$

从而公式（2）在跨越界面时也成立$\left|\nabla\phi\right| = 1$，确保了符号距离函数$\phi$在界面附近光滑连续，方便计算界面法向和曲率。

$$\kappa = \nabla \cdot \vec{n} = \nabla \cdot \frac{\nabla\phi}{\left|\nabla\phi\right|} = \nabla^2\phi \tag{4}$$

2 数值方法

2.1 界面推进

符号距离函数ϕ的推进是通过求解 level-set 方程（方程 5）实现的，该方程可以直接用界面上的物质微团的物质导数推导得到。

$$\frac{\partial \phi}{\partial t}+\vec{u}\cdot\nabla\phi=0 \tag{5}$$

上式用于描述界面$\phi=0$随流体流动速度$\vec{u}$变化的运动过程。即该方程是界面的控制方程，界面$\phi=0$随$\vec{u}$是精确演化的，但$\phi\neq0$区域在演化后不再是符号距离函数，也就是说界面演化后界面外的ϕ值不再是到界面的最短距离。这是由于跨过界面，沿法线方向上流体速度存在$\frac{\partial u_n}{\partial n}\neq0$，即速度梯度不为 0，界面两侧的运动的速度不一样，会逐渐使得ϕ场失去符号距离函数的含义。如图 2 所示，红色$\phi=0$的等值线是界面位置，初始时刻等值线$\phi=1$和$\phi=0$之间的距离为 1，当界面内侧速度大于界面速度时，内侧的等值线$\phi=1$传播得比红色界面更快，经过一个 dt 时间后，等值线$\phi=1$和$\phi=0$之间的距离不再是 1，整个ϕ场也会变得扭曲，失去符号距离函数$|\nabla\phi|=1$的特性。

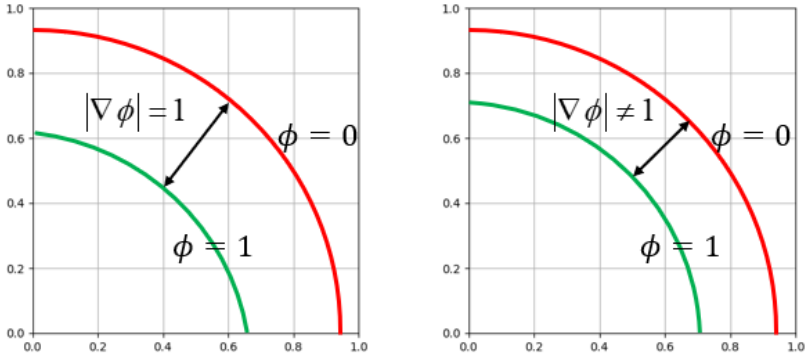


图 2 界面演化前后ϕ场变化

方程（5）的二维离散形式可以写成：

$$\frac{\phi^{n+1}-\phi^{n}}{\Delta t}+u_{i,j}^{n}\frac{\phi_{i+1/2,j}^{n}-\phi_{i-1/2,j}^{n}}{\Delta x}+v_{i,j}^{n}\frac{\phi_{i,j+1/2}^{n}-\phi_{i,j-1/2}^{n}}{\Delta y}=0 \tag{6}$$

其中非定常项采用一阶精度 Euler 前差格式，对流项采用 3 阶迎风 QUICK 格式：

$$\phi_{i+1/2}=\begin{cases}\dfrac{3\phi_{i+1}+6\phi_i-\phi_{i-1}}{8}, u_i>0\\ \dfrac{3\phi_i+6\phi_{i+1}-\phi_{i+2}}{8}, u_i<0\end{cases} \tag{7}$$

迎风格式表明空间导数离散时要体现对流项代表信息传递的各向异性，即信息从上游传向下游。此外对流项也可以采用其他高阶格式，如 ENO、WENO 格式等。

2.2 重新初始化

在界面演化后，为了使得ϕ场恢复成符号距离函数，同时又不改变界面的位置，需要对界面演化后的ϕ场进行重新初始化的操作（Reinitialization）。新求解的 level set 函数ϕ_d需要与刚刚演化后的ϕ场有着共同的界面$\phi=0$，且ϕ_d必须是符号距离函数，要满足符号距离函数的特性，即满足公式（2）。本文采用 Sussman 等人[1]介绍的迭代法来重新初始化ϕ场，在该方法中，若ϕ场已经接近是符号距离函数了，那仅用几步迭代就能将ϕ场转化为有效的符号距离函数，具体方式是求解以下的偏微分方程：

$$\frac{\partial \phi}{\partial \tau}=\operatorname{sgn}\left(\phi^0\right)\left(1-|\nabla \phi|\right) \tag{8}$$

其中ϕ^0是初始化前的ϕ场，sgn 函数称为 sharp sign function，其表达式由公式（9）给出，它的作用是使ϕ_d与ϕ有共同的界面，确保初始化过程中界面位置$\phi=0$不被改变。τ 是伪时间，并没有物理意义，ϕ场以公式（8）进行伪时间推进，公式（8）的稳态解就是新求得的符号距离函数场ϕ，且它满足公式（2）。

$$\operatorname{sgn}\left(\phi^0\right)=\begin{cases}1 & \phi^0>0\\ 0 & \phi^0=0\\ -1 & \phi^0<0\end{cases} \tag{9}$$

重新初始化方程（8）改写成方程（10）的非线性双曲方程的形式后可以看到它有个很重要的数学特性，即它解的信息会从界面处沿法线方向以$\operatorname{sgn}\left(\phi^0\right)$为速度向全场发出，即它的特征速度在界面的两侧都是沿法向从界面指向外侧，ϕ场始终都是从界面朝两侧开始初始化。这个性质使得我们不必要求解方程（8）的稳态解，因为我们只需要界面附近的ϕ场保持符号距离函数的特性即可，只需要采用固定的迭代次数就能使界面附近的ϕ场满足这样的要求。

$$\frac{\partial \phi}{\partial \tau}+\left[\operatorname{sgn}\left(\phi^0\right)\frac{\nabla \phi}{|\nabla \phi|}\right]\cdot\nabla \phi=\operatorname{sgn}\left(\phi^0\right) \tag{10}$$

方程（8）的离散形式如下，时间项同样采用一阶精度 Euler 前差格式：

$$\phi_{ij}^{n+1}=\phi_{ij}^{n}-\Delta\tau(\operatorname{sgn}(\phi_{ij}^{0})G(\phi_{ij}^{n})) \tag{11}$$

上式中$G(\phi_{ij}^{n})$是$|\nabla\phi|-1$的离散形式，具体定义如下：

$$G(\phi_{i,j}^{n})=\begin{cases}\sqrt{\max(a_+^2,b_-^2)+\max(c_+^2,d_-^2)}-1 & \phi_{i,j}^{0}>0\\ \sqrt{\max(a_-^2,b_+^2)+\max(c_-^2,d_+^2)}-1 & \phi_{i,j}^{0}<0\\ 0 & otherwise\end{cases} \tag{12}$$

其中 $f_+ = \max(f,0)$， $f_- = \min(f,0)$，则 $\frac{\partial \phi}{\partial x}$， $\frac{\partial \phi}{\partial y}$ 采用一阶迎风格式离散：

$$\begin{aligned} a &= D_x^- \left(\phi_{i,j}\right) = (\phi_{i,j} - \phi_{i-1,j}) / \Delta x \\ b &= D_x^+ \left(\phi_{i,j}\right) = (\phi_{i+1,j} - \phi_{i,j}) / \Delta x \\ c &= D_y^- \left(\phi_{i,j}\right) = (\phi_{i,j} - \phi_{i,j-1}) / \Delta y \\ d &= D_y^+ \left(\phi_{i,j}\right) = (\phi_{i,j+1} - \phi_{i,j}) / \Delta y \end{aligned} \tag{13}$$

以上格式在界面附近时，迎风格式所用ϕ场信息会跨越界面，可能引起界面反向。为了使在界面附近的点也保持迎风格式，界面附近点的迎风格式采用网格中心到界面的距离代替一个网格步长，而远离界面的位置原格式保持不变。此时方程（11）可以修正为：

$$\phi_{i,j}^{n+1} = \begin{cases} \phi_{i,j}^n - \dfrac{\Delta \tau}{\Delta x}\left[\operatorname{sgn}(\phi_{i,j}^0)\left|\phi_{i,j}^n\right| - D_{i,j}\right] & if\ (i,j) \in \sum_{\Delta x} \\ \phi_{i,j}^n - \Delta \tau \operatorname{sgn}(\phi_{i,j}^0) G(\phi_{i,j}^n) & otherwise \end{cases} \tag{14}$$

其中 $\sum_{\Delta x}$ 指的是临近界面的网格，在此处ϕ符号变号，代表在紧邻界面的网格点上通过亚网格的几何途径来构建迎风格式[4]； $D_{i,j}$ 代表网格中心到界面的距离，是从 ϕ^0 中近似得到的符号距离函数：

$$D_{i,j} = \frac{\Delta x \phi_{i,j}^0}{\Delta \phi_{i,j}^0} \tag{15}$$

$\Delta \phi_{i,j}^0$ 取各种情况下斜率的最大值：

$$\Delta \phi_{i,j}^0 = \max \left\{ \begin{array}{l} \left[(\phi_{i+1,j}^0 - \phi_{i-1,j}^0)^2 + (\phi_{i,j+1}^0 - \phi_{i,j-1}^0)^2\right]^{1/2} / 2, \\ \left|\phi_{i+1,j}^0 - \phi_{i,j}^0\right|, \left|\phi_{i,j}^0 - \phi_{i-1,j}^0\right|, \left|\phi_{i,j+1}^0 - \phi_{i,j}^0\right|, \left|\phi_{i,j}^0 - \phi_{i,j-1}^0\right|, \gamma \end{array} \right\} \tag{16}$$

其中 γ 是相对于网格尺寸的常数小量，可以取 $\gamma = 0.1\Delta x$ 或 $\gamma = 0.10\Delta x$。

3 问题描述

本文对 1×1 计算域内的圆形两相界面在剪切流下的运动进行数值计算，计算域的尺寸和圆形的初始位置如图 3 所示。绕圆心旋转的定常剪切流速度分布如下式：

$$\mathbf{u} = \begin{cases} u = -2\pi \cos[\pi(x-0.5)] \sin[\pi(y-0.5)] \\ v = 2\pi \sin[\pi(x-0.5)] \cos[\pi(y-0.5)] \end{cases} \tag{17}$$

边界取 0 梯度边界条件 $\frac{\partial \phi}{\partial x} = 0$， $\frac{\partial \phi}{\partial y} = 0$。界面会被剪切流拉伸变形，并在尾部形成细丝。在计算 T = 2.0 s 后将流场翻转继续计算至 T = 4.0 s。计算预期是界面在运行完 4 s 后界面恢复成初始的形状。

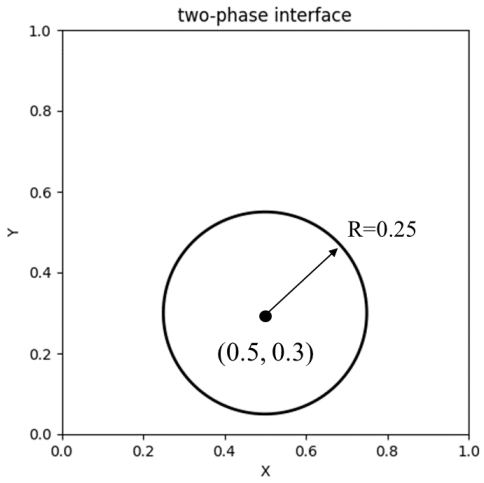
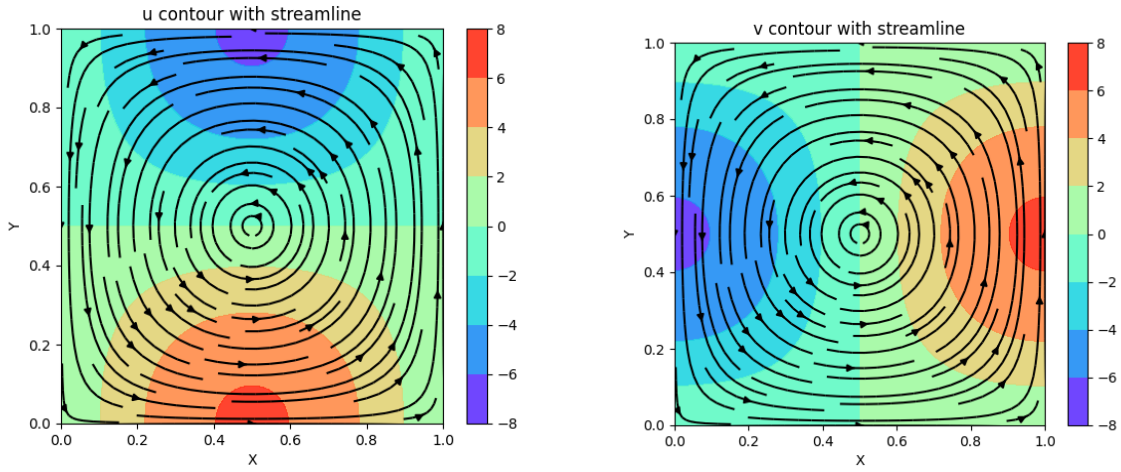


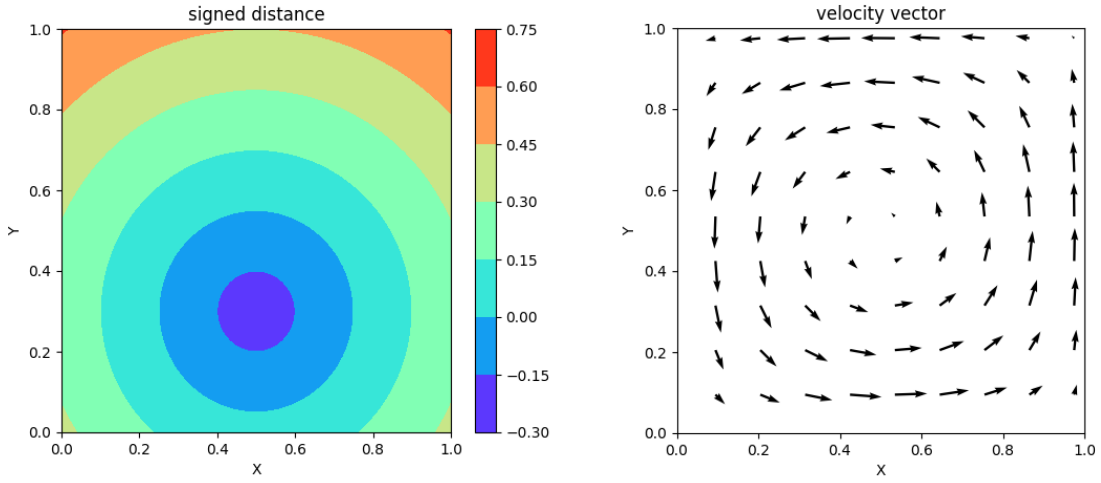
图 3 界面运动初始条件

前 2 s 内的绕圆心旋转剪切流速度分布图如下：



（左：u 速度分布及流线，右：v 速度分布及流线）

图 4 初始时刻速度分布及流线



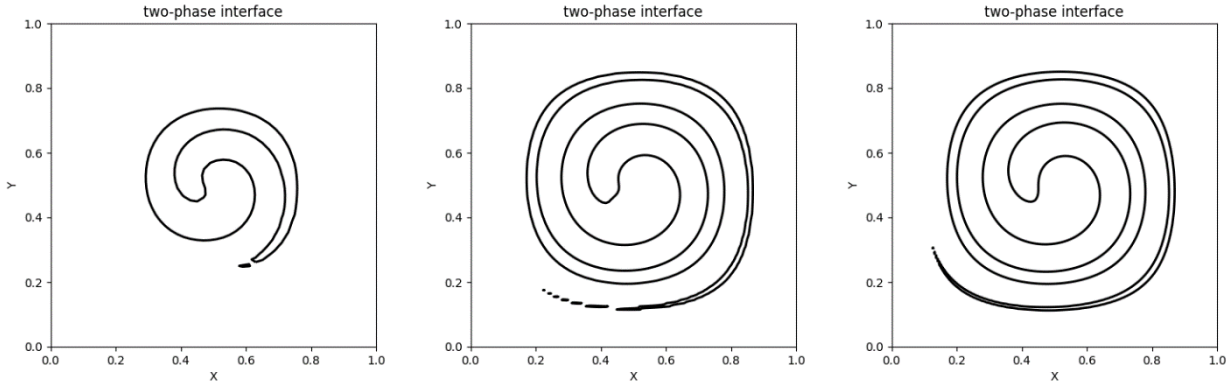
（左：初始时刻ϕ场，右：速度矢量）

图 5 初始时刻ϕ场分布及速度矢量图

4 结果与讨论

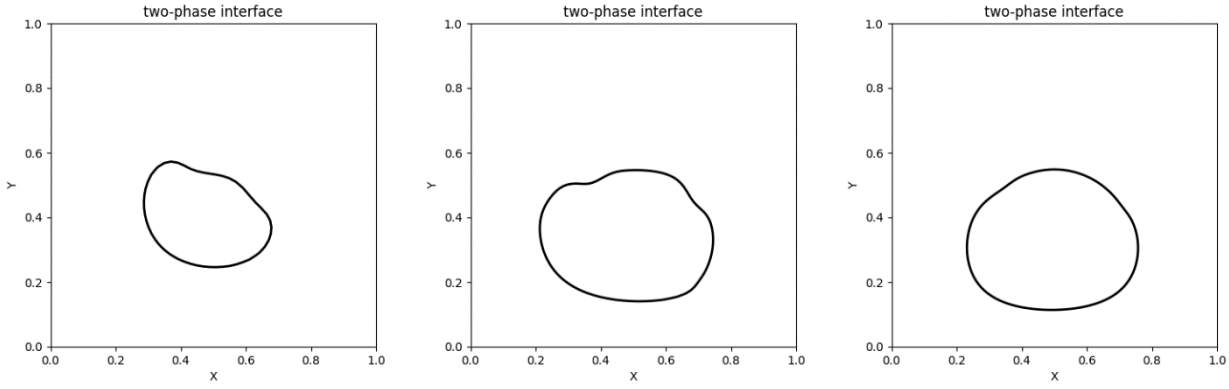
4.1 网格无关性验证

为了验证计算结果独立于网格，图 6 给出了在 50×50、100×100 和 200×200 网格下的计算结果，对比 T = 2.0 s 时刻和速度翻转后运行到 T = 4.0 s 时刻的界面变化。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，后面小结的讨论都在 100×100 网格下进行。界面尾部的液丝厚度小于临界厚度后会发生断裂，这是由于网格解析度不够造成的，当网格解析度更小时，会连这部分断裂的液丝也捕捉不到（如图 6 左上图）。



（左：50×50，中：100×100，右：200×200）

T = 2.0 s



（左：50×50，中：100×100，右：200×200）

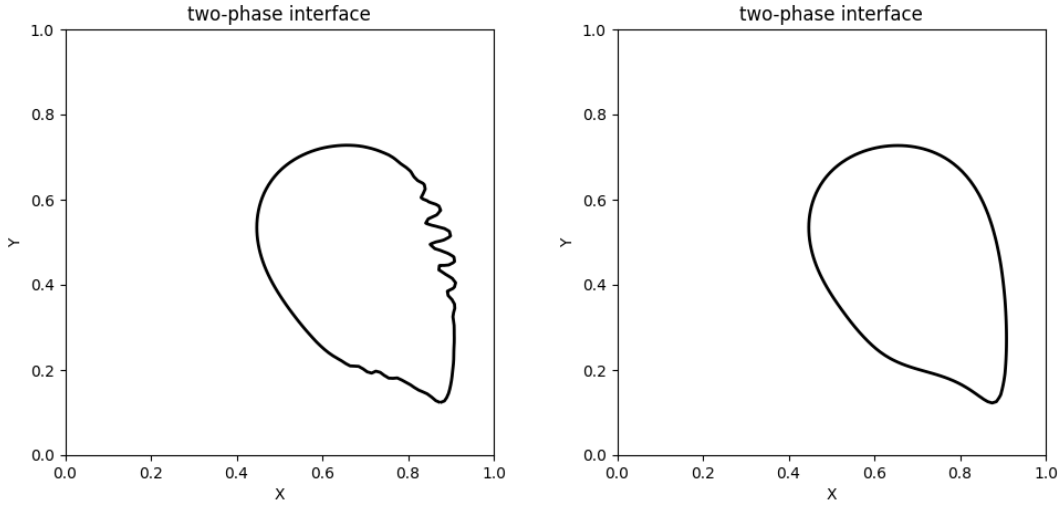
T = 4.0 s

图 6 网格无关性验证

4.2 亚网格界面修正

在界面初始化过程中，信息是从界面出发，沿垂直于界面的特征线向外传递，我们在离散空间导数时希望能保持这一特性。但是对 $\frac{\partial \phi}{\partial x}$，$\frac{\partial \phi}{\partial y}$ 采用一阶迎风格式离散（公式 13）时，

在界面附近所用ϕ场信息已经跨越界面，格式并非真正意义上的迎风，会导致界面修正失败，即界面修正可能发生反向，引起界面出现非物理的褶皱，如图 7（左图）所示。通过 Russo 等人[4]的亚网格修正方法可以有效避免迎风格式引起的非物理的褶皱。本文还发现该亚网格修正的效果也会受到 CFL 数的影响，建议在小 CFL 数下进行亚网格修正。

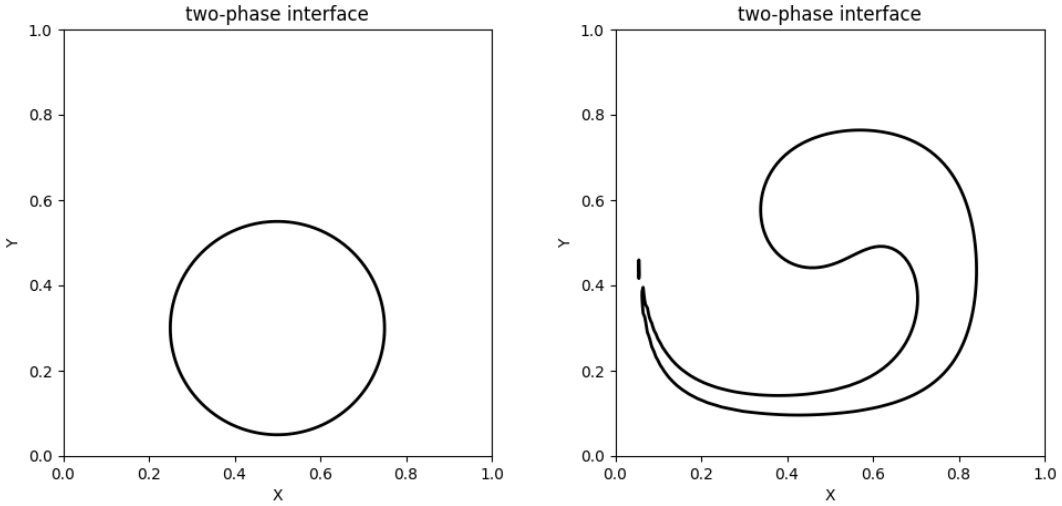


（左：迎风格式引起界面褶皱，右：亚网格修正后的效果）

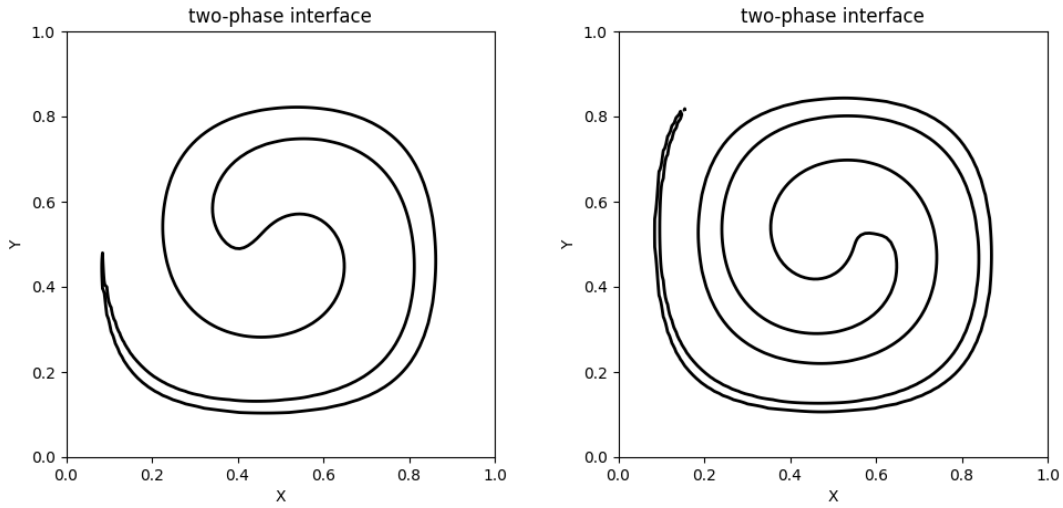
图 7 T= 0.09 s 时刻界面位置

4.3 不同时刻界面演化

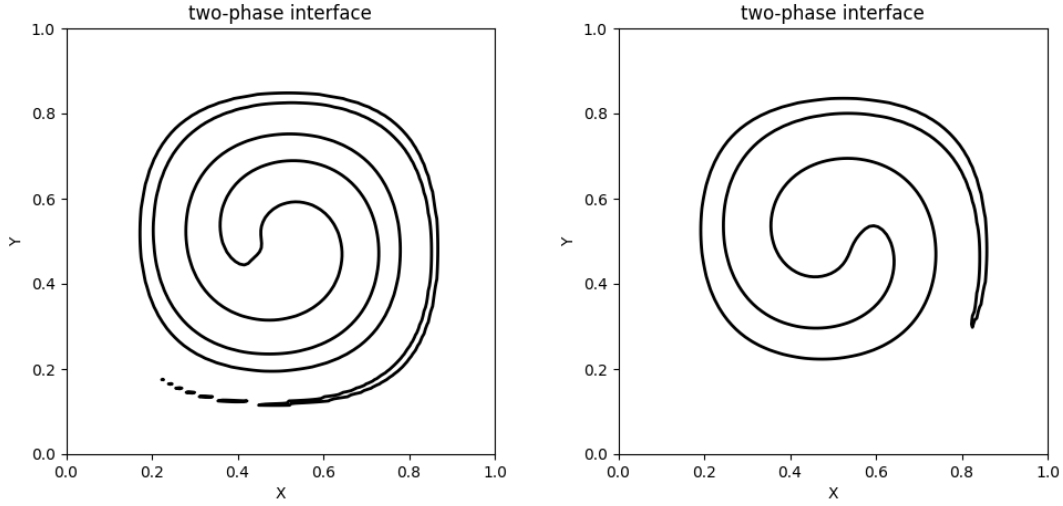
图 8 分别给出了 100×100 网格下 T = 0.0、0.5、1.0、1.5、2.0、2.5、3.0、4.0 s 时刻的界面形状。可以发现界面在旋转剪切流的作用下不断拉伸变形，在 T = 2.0 s 之后开始恢复成原来的形状，但由于数值误差等因素，T = 2.0 s 时刻的界面并不能完全恢复成圆形。界面在拉伸的过程后尾部的液丝会因为网格解析度不够而断裂，增大网格量可以部分解决这一问题。



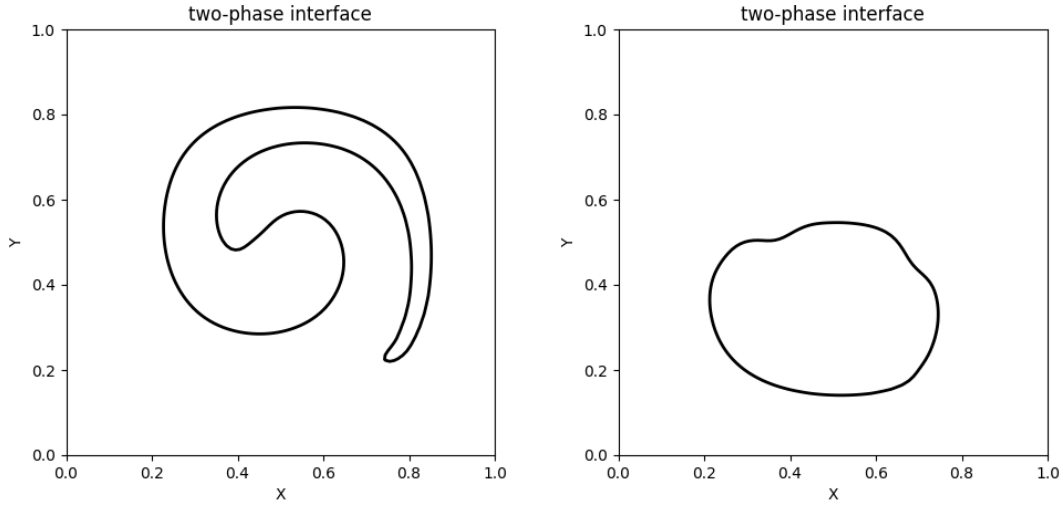
（左：T = 0.0 s，右：T = 0.5 s）



（左：T = 1.0 s，右：T = 1.5 s）



（左：T = 2.0 s，右：T = 2.5 s）

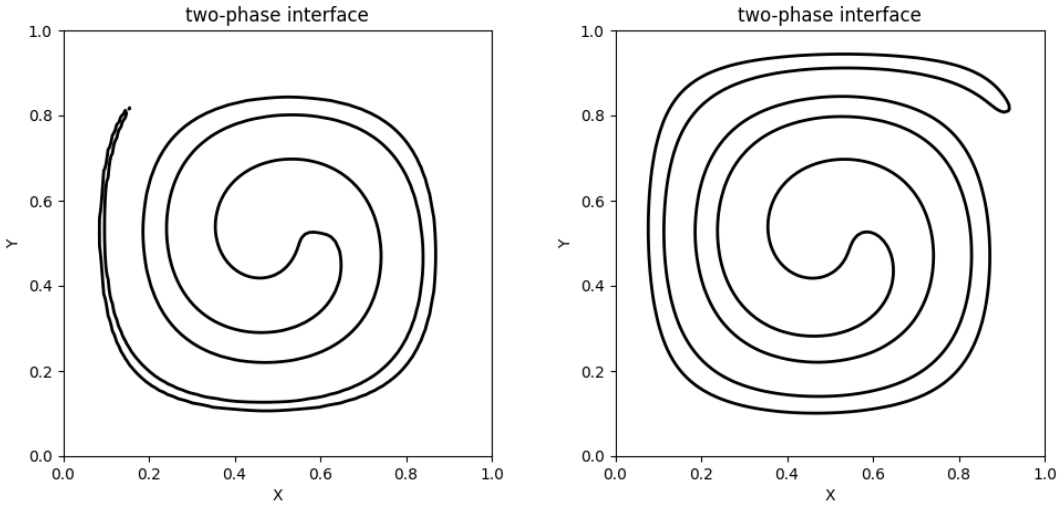


（左：T = 3.0 s，右：T = 4.0 s）

图 8 不同时刻界面演化

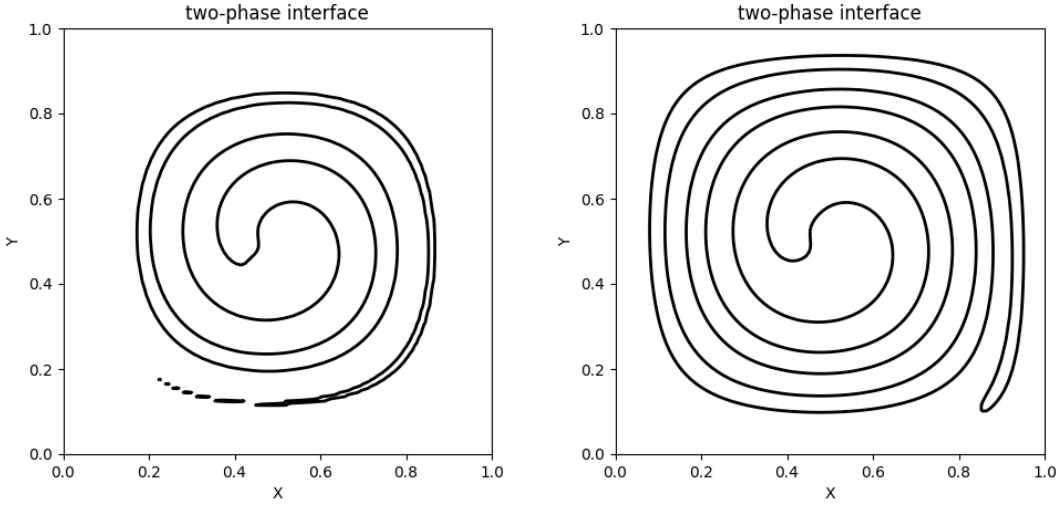
4.4 初始化过程对质量守恒的影响

本文的初始化过程采用一阶迎风格式对ϕ场重新计算符号距离函数，精度肯定不会很高。通过对比在不同时间步间隔下进行 Reinitialization 可以判断该过程对质量守恒性的影响。



（左：间隔 200，右：间隔 20）

T = 1.5 s



（左：间隔 200，右：间隔 20）

T = 2.0 s

图 9 不同时间步间隔下进行初始化效果

图 9 给出的是 100×100 网格下不同时间步间隔进行 Reinitialization 的结果，对比 T = 1.5 s 时刻采用不同时间步间隔下进行初始化后演化的界面，可以看出间隔 20 比间隔 200 有更长更宽的尾部，这冗长的尾部质量是不守恒的，T = 2.0 s 时刻有着相同的结论。Reinitialization 过程次数进行得越少，重新初始化过程对界面位置的影响越小。本文发现该方法实现的

Reinitialization 过程会让小尺度界面变得很饱满，让相互靠近的界面有着比实际情况下更宽的厚度，从而变相拉宽拉长了原本细长的尾部，使得界面尾部的质量不守恒。

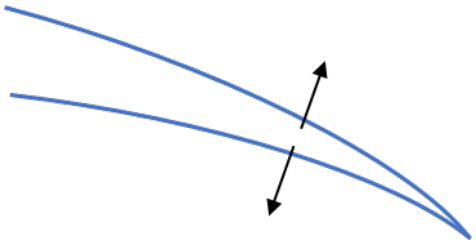


图 10 重新初始化会导致两个界面靠近时界面拉宽

5 结论

本文采用基于亚网格修正的 Level Set 方法研究了两相界面在旋转剪切流下的运动情况，研究表明 100×100 网格下的计算结果已经可以较好地独立于网格。当网格解析度不够时，界面尾部的液丝厚度在小于临界厚度后会发生断裂，提高网格数量可以有效解决这一问题。本文采用的亚网格修正方法可以有效避免迎风格式引起的非物理的褶皱，且该亚网格修正的效果也会受到 CFL 数的影响，小 CFL 数下亚网格修正的效果更好。此外，本文还发现频繁的初始化过程会严重影响两相流运动的质量守恒性，频繁的初始化过程会让小尺度界面变得很饱满，让相互靠近的界面有着比实际情况下更宽的厚度，从而变相拉宽拉长原本就细长的尾部，使得界面尾部的质量不守恒。

6 参考文献

[1] Sussman M， Smereka P， Osher S . A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow[J]. J.comput.phys, 1994, 114(1):146-159.

[2] Fedkiw S， Osher S . Level set methods and dynamic implicit surfaces. 2002.

[3] Sethian J A， Smereka P . LEVEL SET METHODS FOR FLUID INTERFACES[J]. Annual Review of Fluid Mechanics, 2003, 35(1):341-372.

[4] Russo G， Smereka P . A Remark on Computing Distance Functions[J]. Journal of Computational Physics, 2000, 163(1):51-67.

[5] Subai M . Level Set Methods and Fast Marching Methods[C]// Computer and Information Technology. SRCE - Sveučilišni računski centar, 2003.

7 代码附件

本文的 level set 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，定义初始条件，实现迭代求解
interface_propagation.py	3 阶 QUICK 格式推进 phi
re_init.py	对ϕ场进行重新初始化恢复成符号距离函数，包括亚网格修正
myplot.py	自定义绘图

main.py 文件：

```
import math
import numpy as np
import myplot
import interface_propagation as propagation
import re_init as re

# 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 = 4.0 # Final time
CFL = 0.01  # CFL 数
dt = CFL * dx  # Time step
print("dt=", dt)
count = 0  # 统计迭代次数
interval = 200  # re-init 间隔的时间步
nt = 5  # re-init 时迭代步数
t1 = 2.0  # 速度反号时间 2.0
# savet = [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]  # 指定需要保存图片的时间
# savet = [0.5, 1.0, 1.5, 2.0]

# 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)))
phi = np.zeros((len(y), len(x)))
# define the circle
R = 0.25
x_center, y_center = 0.5, 0.3
for j in range(0, len(y)):
    for i in range(0, len(x)):
        # 初始化符号距离函数 phi
        phi[j, i] = math.sqrt((x[i] - x_center) ** 2 + (y[j] - y_center) ** 2) - R
        # 初始化速度
    u[j, i] = -2 * math.pi * math.cos(math.pi * (x[i] - 0.5)) * math.sin(math.pi * (y[j] - 0.5))
        # u[j, i] = 1.0
        v[j, i] = 2 * math.pi * math.sin(math.pi * (x[i] - 0.5)) * math.cos(math.pi * (y[j] - 0.5))
        # v[j, i] = 1.0

# Plot initial conditions
# myplot.plotC(x, y, phi)
# myplot.plotContour(x, y, phi)
# myplot.plotU(x, y, u, v)
# myplot.plotu(x, y, u, v)
# myplot.plotv(x, y, u, v)

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

    if count == int(t1/dt+1):
        print("速度反号，反号时间：{0}，反号前迭代了：{1}".format(t,count))
        u = -1*u
        v = -1*v
```

```
    # 界面推进
  phi = propagation.calcPhi(x,y,dx,dy,dt,u,v,phi)

    # Re-init
    if (count+1) % interval == 0:
        print("开始 Re-init，此时迭代步为：{0}".format(count))
        phi = re.reinit(x, y, dx, dy, dt, nt, phi)

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

    # 选择合适的图片保存方式，方式一：指定时间保存
  # for i in range(len(savet)):  # 在 savet 指定的时间下保存图片
  #     if count == int(savet[i]/dt):
    #         myplot.saveC(x, y, t, phi)

    # 选择合适的图片保存方式，方式二：间隔保存图片，方便制成动图
  if count % 100 == 0:
        myplot.saveC(x, y, t, phi)
        # myplot.saveContour(x, y, t, phi)

print("总共迭代了：{0}次".format(count))
# 后处理
myplot.plotC(x, y, phi)
myplot.plotContour(x, y, phi)
#myplot.plotU(x, y, u, v)  # 查看反向后的速度
#myplot.plotu(x, y, u, v)
#myplot.plotv(x, y, u, v)
```

interface_propagation.py 文件：

```
#  3 阶 QUICK 格式推进 phi

import numpy as np

def calcPhi(x,y,dx,dy,dt,u,v,phi):
    #  定义中间变量
    phiR = np.zeros_like(phi)
    phiL = np.zeros_like(phi)
    phiU = np.zeros_like(phi)
    phiD = np.zeros_like(phi)

    phin = phi.copy()

    #  计算网格面通量
    for j in range(2, len(y)-2):
        for i in range(2, len(x)-2):
            if u[j, i] >= 0.0:
                phiR[j, i] =(3*phin[j, i+1]+6*phin[j, i]-phin[j, i-1])/8
                phiL[j, i] =(3*phin[j, i]+6*phin[j, i-1]-phin[j, i-2])/8
            else:
                phiR[j, i] =(3*phin[j, i]+6*phin[j, i+1]-phin[j, i+2])/8
                phiL[j, i] =(3*phin[j, i-1]+6*phin[j, i]-phin[j, i+1])/8

            if v[j, i] >= 0.0:
                phiU[j, i] =(3*phin[j+1, i]+6*phin[j, i]-phin[j-1, i])/8
                phiD[j, i] =(3*phin[j, i]+6*phin[j-1, i]-phin[j-2, i])/8
            else:
                phiU[j, i] =(3*phin[j, i]+6*phin[j+1, i]-phin[j+2, i])/8
                phiD[j, i] =(3*phin[j-1, i]+6*phin[j, i]-phin[j+1, i])/8

    #  时间推进 phi
    phi[2:-2, 2:-2] = phin[2:-2, 2:-2] - u[2:-2, 2:-2] * dt / dx * (phiR[2:-2, 2:-2] - phiL[2:-2, 2:-2]) \
                                       - v[2:-2, 2:-2] * dt / dy * (phiU[2:-2, 2:-2] - phiD[2:-2, 2:-2])

    # 更新边界条件
    # phi[1, :] = 2 * phi[2, :] - phi[3, :]
    # phi[0, :] = 2 * phi[1, :] - phi[2, :]  # 第 0 行，列是[:]
    # phi[-2, :] = 2 * phi[-3, :] - phi[-4, :]
    # phi[-1, :] = 2 * phi[-2, :] - phi[-3, :]
    # phi[:, 1] = 2 * phi[:, 2] - phi[:, 3]
    # phi[:, 0] = 2 * phi[:, 1] - phi[:, 2]
    # phi[:, -2] = 2 * phi[:, -3] - phi[:, -4]
    # phi[:, -1] = 2 * phi[:, -2] - phi[:, -3]

    phi[0, :] = phi[2, :]  # 第 0 行，列是[:]
    phi[1, :] = phi[2, :]
    phi[-1, :] = phi[-3, :]
    phi[-2, :] = phi[-3, :]
    phi[:, 0] = phi[:, 2]
    phi[:, 1] = phi[:, 2]
    phi[:, -1] = phi[:, -3]
```

```
    phi[:, -2] = phi[:, -3]

    # 以下更新边界条件的方式有可能在边界处形成非物理的界面
    # phi[0, :] = 2 * phi[2, :] - phi[4, :]  # 第 0 行，列是[:]
    # phi[1, :] = 2 * phi[2, :] - phi[3, :]
    # phi[-1, :] = 2 * phi[-3, :] - phi[-5, :]
    # phi[-2, :] = 2 * phi[-3, :] - phi[-4, :]
    # phi[:, 0] = 2 * phi[:, 2] - phi[:, 4]
    # phi[:, 1] = 2 * phi[:, 2] - phi[:, 3]
    # phi[:, -1] = 2 * phi[:, -3] - phi[:, -5]
    # phi[:, -2] = 2 * phi[:, -3] - phi[:, -4]

    return  phi
```

re_init.py 文件：

```
import math
import numpy as np

# reinitialization 会导致质量不守恒（尤其是该文件用的格式），要尽量少用

# def sgn(phi,dx):  # epsilon 由网格尺寸的 2-3 倍给定，此处仅给了 1 倍
#     return phi/math.sqrt(phi**2+dx**2)

def reinit(x,y,dx,dy,dt,nt,phi):
    # 迭代前将 phi 赋值给 phid，后面求出来的 phid 是 reinit 后的符号距离函数
    phid = phi.copy()

    # 定义中间变量
    a = np.zeros_like(phi)
    b = np.zeros_like(phi)
    c = np.zeros_like(phi)
    d = np.zeros_like(phi)
    G = np.zeros_like(phi)
    sgn = np.ones_like(phi)
    # subcell 临时变量
    deltaphi = np.ones_like(phi)
    temp1 = np.zeros_like(phi)
    temp2 = np.zeros_like(phi)
    temp3 = np.zeros_like(phi)
    temp4 = np.zeros_like(phi)
    temp5 = np.zeros_like(phi)
    gamma = 0.1*dx
    D = np.zeros_like(phi)

    # reinitialization
    for n in range(nt):
        phin = phid.copy()

        # 一阶迎风格式
        a[2:-2, 2:-2] = (phin[2:-2, 2:-2]-phin[2:-2, 1:-3])/dx
        b[2:-2, 2:-2] = (phin[2:-2, 3:-1]-phin[2:-2, 2:-2])/dx
        c[2:-2, 2:-2] = (phin[2:-2, 2:-2]-phin[1:-3, 2:-2])/dy
        d[2:-2, 2:-2] = (phin[3:-1, 2:-2]-phin[2:-2, 2:-2])/dy

        for j in range(2, len(y)-2):
            for i in range(2, len(x) - 2):
                if phi[j, i] > 0.0:  # 必须以原始的 phi 进行判断
                    sgn[j, i] = 1.0
                    G[j, i] = math.sqrt(max((max(a[j, i], 0.0))**2, (min(b[j, i],0.0))**2) \
                                        +max((max(c[j, i], 0.0))**2, (min(d[j, i],0.0))**2))-1
                elif phi[j, i] < 0.0:
                    sgn[j, i] = -1.0
                    G[j, i] = math.sqrt(max((min(a[j, i], 0.0))**2, (max(b[j, i], 0.0))**2) \
                                        +max((min(c[j, i], 0.0))**2,(max(d[j, i], 0.0))**2))-1
                else:
                    sgn[j, i] = 0.0
                    G[j, i] = 0.0

        # 没有进行亚网格修正
        phid[2:-2, 2:-2] = phin[2:-2, 2:-2] - dt*sgn[2:-2, 2:-2]*G[2:-2, 2:-2]

        # 进行亚网格修正
        temp1[2:-2, 2:-2] = np.sqrt((phi[2:-2, 3:-1]-phi[2:-2, 1:-3])**2+(phi[3:-1, 2:-2]-phi[1:-3, 2:-
2])**2)/2.0
        temp2[2:-2, 2:-2] = np.abs(phi[2:-2, 3:-1]-phi[2:-2, 2:-2])
        temp3[2:-2, 2:-2] = np.abs(phi[2:-2, 2:-2]-phi[2:-2, 1:-3])
        temp4[2:-2, 2:-2] = np.abs(phi[3:-1, 2:-2]-phi[2:-2, 2:-2])
        temp5[2:-2, 2:-2] = np.abs(phi[2:-2, 2:-2]-phi[1:-3, 2:-2])

        deltaphi =
np.maximum(temp1,np.maximum(temp2,np.maximum(temp3,np.maximum(temp4,np.maximum(temp5,gamma)))))
        D = dx*phi/deltaphi
```

```
        for j in range(2, len(y)-2):
            for i in range(2, len(x) - 2):
                if phi[j, i]*phi[j, i-1]<0.0 or phi[j, i]*phi[j, i+1]<0.0 or \
                        phi[j, i]*phi[j-1, i]<0.0 or phi[j, i]*phi[j+1, i]<0.0:
                    phid[j, i] = phin[j, i] - dt/dx*(sgn[j, i]*math.fabs(phin[j, i])-D[j, i])
                # else:
                #     phid[j, i] = phin[j, i] - dt * sgn[j, i] * G[j, i]

        # 更新边界条件
       # phid[1, :] = 2*phid[2, :]-phid[3, :]
        # phid[0, :] = 2*phid[1, :]-phid[2, :]  # 第0行，列是[:]
        # phid[-2, :] = 2*phid[-3, :]-phid[-4, :]
        # phid[-1, :] = 2*phid[-2, :]-phid[-3, :]
        # phid[:, 1] = 2*phid[:, 2]-phid[:, 3]
        # phid[:, 0] = 2*phid[:, 1]-phid[:, 2]
        # phid[:, -2] = 2*phid[:, -3]-phid[:, -4]
        # phid[:, -1] = 2*phid[:, -2]-phid[:, -3]

        phid[0, :] = phid[2, :]  # 第0行，列是[:]
        phid[1, :] = phid[2, :]
        phid[-1, :] = phid[-3, :]
        phid[-2, :] = phid[-3, :]
        phid[:, 0] = phid[:, 2]
        phid[:, 1] = phid[:, 2]
        phid[:, -1] = phid[:, -3]
        phid[:, -2] = phid[:, -3]

        # phid[0, :] = 2*phid[2, :]-phid[4, :]  # 第0行，列是[:]
        # phid[1, :] = 2*phid[2, :]-phid[3, :]
        # phid[-1, :] = 2*phid[-3, :]-phid[-5, :]
        # phid[-2, :] = 2*phid[-3, :]-phid[-4, :]
        # phid[:, 0] = 2*phid[:, 2]-phid[:, 4]
        # phid[:, 1] = 2*phid[:, 2]-phid[:, 3]
        # phid[:, -1] = 2*phid[:, -3]-phid[:, -5]
        # phid[:, -2] = 2*phid[:, -3]-phid[:, -4]

    return phid
```

myplot.py 文件：

```
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=[0.0])
    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.title("signed distance")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()

def saveContour(x, y, t, C):
    plt.figure(figsize=(5, 5), dpi=100)
    X, Y = np.meshgrid(x, y)
    plt.contourf(X, Y, C[:], cmap="rainbow_r", levels=[-100.0, 0.0, 100.0])
    # plt.grid()
```

```
    # plt.colorbar()
    # plt.title("volume fraction contour")
    plt.title("signed distance")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.savefig('C%.2fs.png' % t)
    plt.close()


# 以下代码用于二维流场中，速度 u 和 v 变化时
def plotU(x, y, u, v):
    plt.figure(figsize=(5, 5), dpi=100)
    X, Y = np.meshgrid(x, y)
    # plt.contourf(X, Y, u, alpha=0.9, cmap="rainbow")
    # plt.colorbar()
    # plt.streamplot(X, Y, u, v, color="k", linewidth=1.5)  # 流线
   # plt.title("u contour with streamline")
    interval = 11
    plt.quiver(X[::interval, ::interval], Y[::interval, ::interval], u[::interval, ::interval],
               v[::interval, ::interval])
    plt.title("velocity vector")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()


def plotu(x, y, u, v):
    plt.figure(figsize=(6, 5), dpi=100)
    X, Y = np.meshgrid(x, y)
    plt.contourf(X, Y, u, alpha=0.9, cmap="rainbow")
    plt.colorbar()
    plt.streamplot(X, Y, u, v, color="k", linewidth=1.5)  # 流线
   plt.title("u contour with streamline")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()


def plotv(x, y, u, v):
    plt.figure(figsize=(6, 5), dpi=100)
    X, Y = np.meshgrid(x, y)
    plt.contourf(X, Y, v, alpha=0.9, cmap="rainbow")
    plt.colorbar()
    plt.streamplot(X, Y, u, v, color="k", linewidth=1.5)  # 流线
   plt.title("v contour with streamline")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.show()


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


def saveC(x, y, t, 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=[0.0])
    plt.title("two-phase interface")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.savefig('%.2fs.png' % t)
    plt.close()
```