

##### **M**icrostructure **In**telligent **Des**ign Software

**快速指导手册**

微结构智能设计软件

**April, 2023**

##### **MInDes – a program for microstructure intelligent design software**

##### **Copyright (c) 2019-2023**

##### **Science center for phase diagram, phase transition, material intelligent design and manufacture, Central South University, China**

##### **This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.**

##### **This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.**

##### **You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.**

目录

[1. MInDes介绍 4](#_Toc134432920)

[2. 软件的配置安装、模拟及可视化 5](#_Toc134432921)

[2.1. 运行环境 5](#_Toc134432922)

[2.2. 软件安装 5](#_Toc134432923)

[2.3. 模拟流程及可视化 8](#_Toc134432924)

[2.4. 基本输入参数简介 14](#_Toc134432925)

[3. 模块介绍 18](#_Toc134432926)

[3.1. 微结构初始化 18](#_Toc134432927)

[3.2. 预处理 23](#_Toc134432928)

[3.3. 相界面 28](#_Toc134432929)

[3.4. 固体力学 33](#_Toc134432930)

[3.5. 流体力学 35](#_Toc134432931)

# MInDes介绍

**MInDes采用C++高级程序语言搭建了基本程序框架。程序底层设置了必要的数据（函数）类型、运算方法、数据网格结构、程序核心循环、各物理场求解器和并行框架。程序中层是对接各求解器的接口模块，由研究人员进行模型、程序功能模块、数据库的二次开发设计。程序表层将在可视化界面上处理MInDes的输入（.mindes）、输出（.log、.vts、.dat和.txt等）等。MInDes可在windows、linux双系统上运行，将使用OpenMP、CUDA等并行库加速计算，使用差分法、傅里叶谱方法、格子玻尔兹曼法等对各物理场进行求解。**

**开发者：**

**MInDes软件开发自2023年始，是中南大学相图、相变及材料智能设计与制备实验室（领导者：杜勇教授，长江学者、杰出青年）的在读博士黄奇（2018年-2024年）在其博士期间的主要成果MID-MESO程序库的框架基础上修改完善而来。多年后续开发期间，许多科研工作者对此进行了贡献：**

**黄奇、**

**手册更新历史：**

**2023年4月（第一版）、2023年11月（第二版）、…**

**合作联系：**

**黄奇博士:** [qihuang0908@163.com](mailto:qihuang0908@163.com)**;**

# 软件的配置安装、模拟及可视化

## 运行环境

Windows:

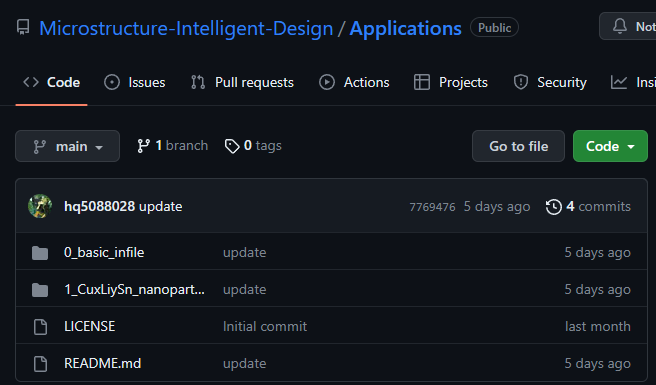
64位（x64）系统。

Linux:

暂无要求。

## 软件安装

从Github下载安装包（Applications-main.zip）：



解压安装包（Applications-main.zip）



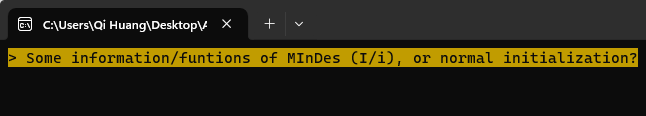
基本的输入文件都放在文件夹（0\_basic\_infile）中，其他应用案例放在后续文件夹中，以文件夹（1\_CuxLiySn\_nanoparticle\_lithiation）为例，打开该文件夹



从上至下：该案例的输入文件、动态库（.dll）、MInDes（linux系统执行文件）、exe运行文件（x64 windows系统执行文件）及其他文件。

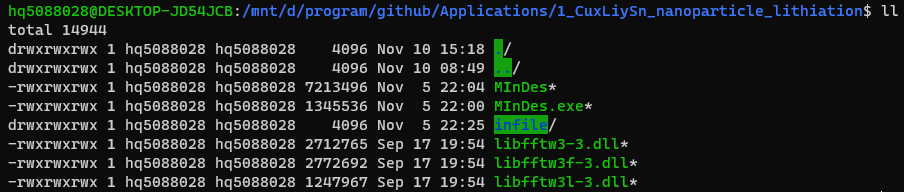
在对应系统中运行执行文件，进入软件配置界面

Windows系统下，双击打开MInDes.exe：

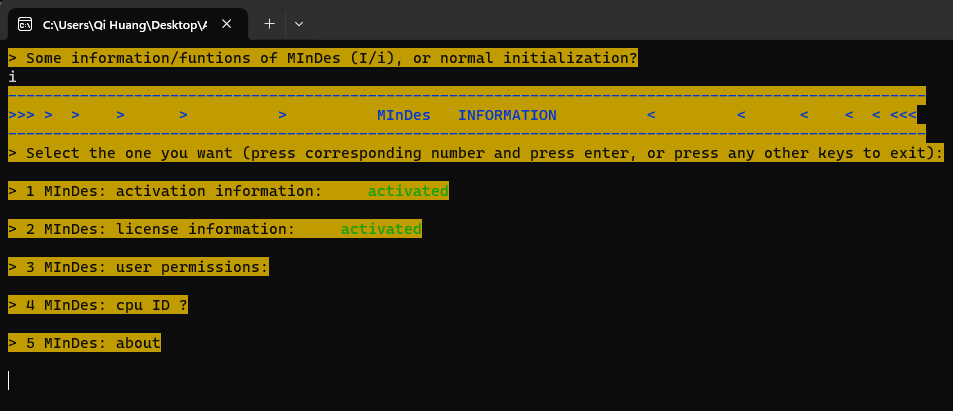


敲击键盘“i”并回车，进入信息界面。

Linux系统下，在该文件夹内执行MInDes，

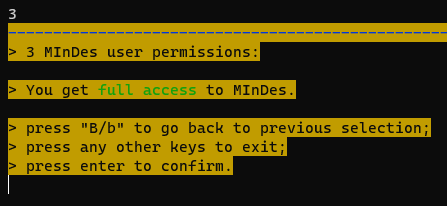


直接进入信息界面，总界面：

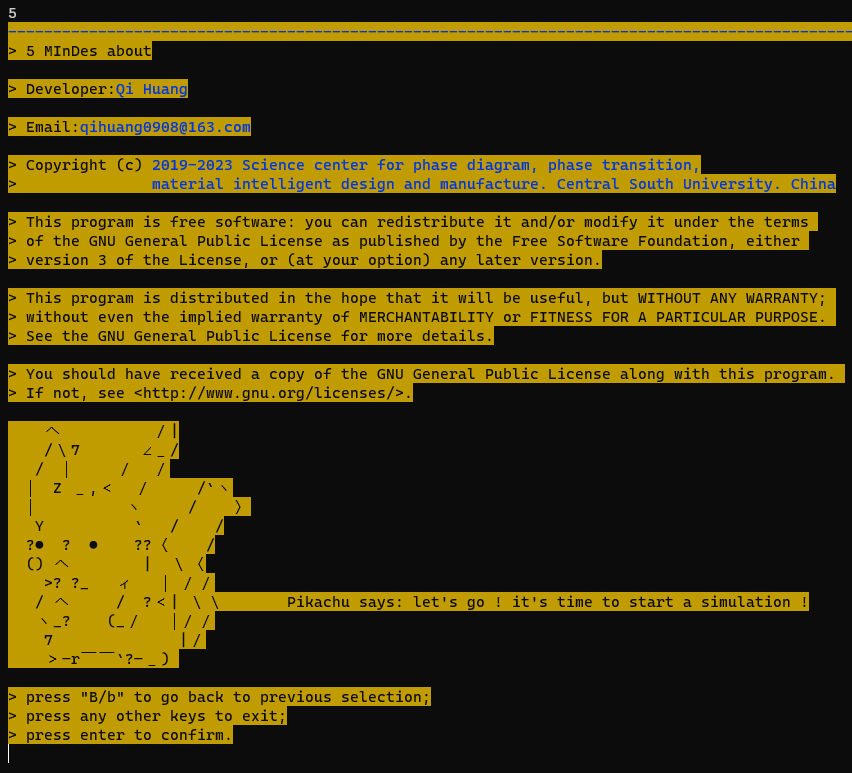


可在此界面查看（1）激活信息及（2）license信息。

在此界面可查看当前软件权限（3）：



最后，可查看软件概况：

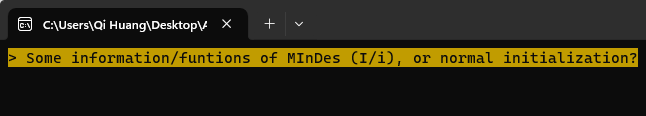


## 模拟流程及可视化

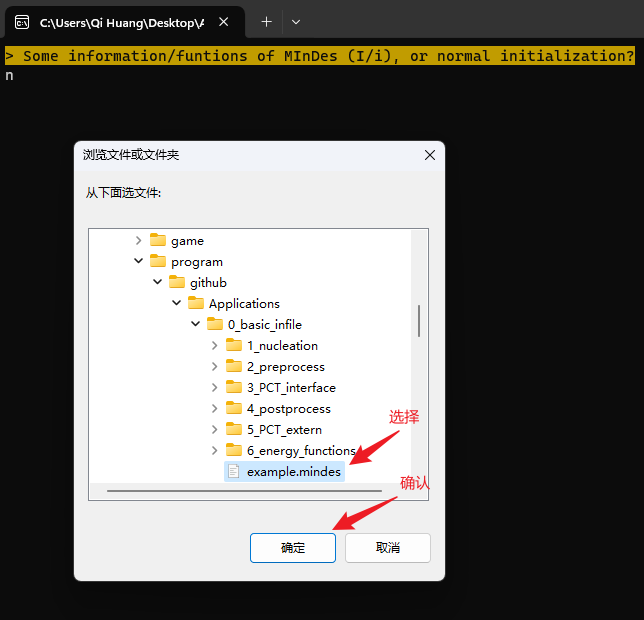
**通过MInDes执行文件读取.mindes输入文件**

Windows读取方法：

双击打开MInDes：



输入N/n选择normal initialization，enter确认，弹出windows文件浏览系统，可在该界面找到对应的输入文件的.mindes文件，比如0\_basic\_infile文件夹中的example.mindes文件（其他的案例可以读取对应文件夹中的.mindes文件）：



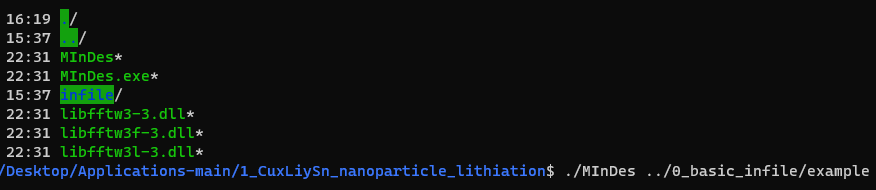
软件将自动读取该**输入文件**，并在**同目录下**生成**同名文件夹**用于保存**输出文件**：



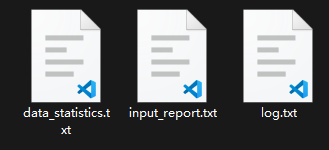
因此也需要注意，同目录下、同名文件夹内的同名输出文件文件会在输出过程中被覆盖，可通过修改该文件夹名称保留输出文件。

Linux读取方法：

定位到MInDes所在文件夹（必须在该文件夹内启动MInDes软件），在启动软件时，在后面输入读取.mindes文件的索引，同样可读取0\_basic\_infile文件夹中的example.mindes文件：



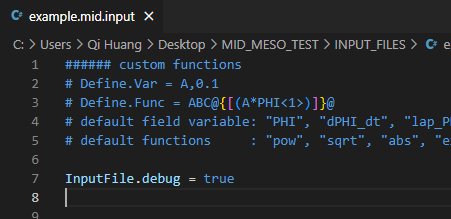
输出文件与Windows系统一致，可以看到example输出文件夹中默认存在三个文本文件：



统计模拟过程数据的data\_statistics.txt文件、对输入文件内所有keys的反馈文件input\_report.txt和日志文件log.txt文件。

**设置.mindes文件**

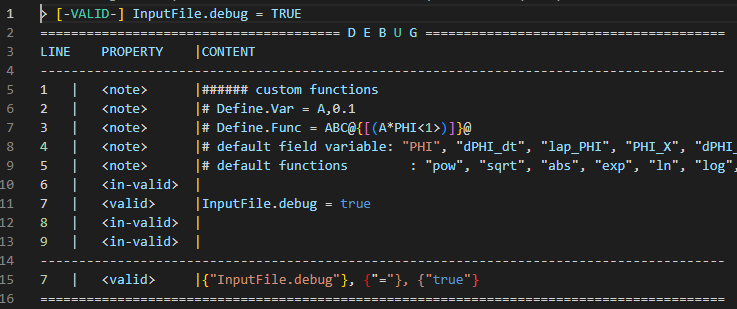
打开example.mindes文件，观察到除了上方被注释的5行，有一行key值：InputFile.debug = true



为与注释行做区分，我们称之为命令行。命令行开头、末尾都不能空格，中间两个空格将命令行分为3部分，第一部分为命令行的钥匙（key）：“InputFile.debug”，第二部分统一都为等号：“=”，第三部分为命令行的值（value）：“true”，为确保命令行不会失效，修改输入文件时不应再引入更多空格。

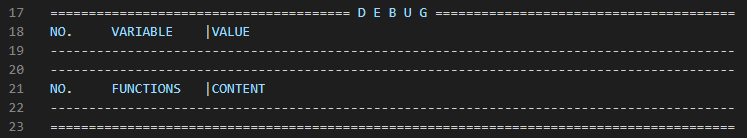
该命令行的功能是对该输入文件进行测试，并在输出在文件input\_report.txt中，然后使用者就可以通过input\_report.txt文件来测试、检查所有命令行的的书写情况。

如，输出案例输入文件的input\_report.txt是一个多行的文本文件，其中第一部分如下图，是对所有输入文本有效性的识别

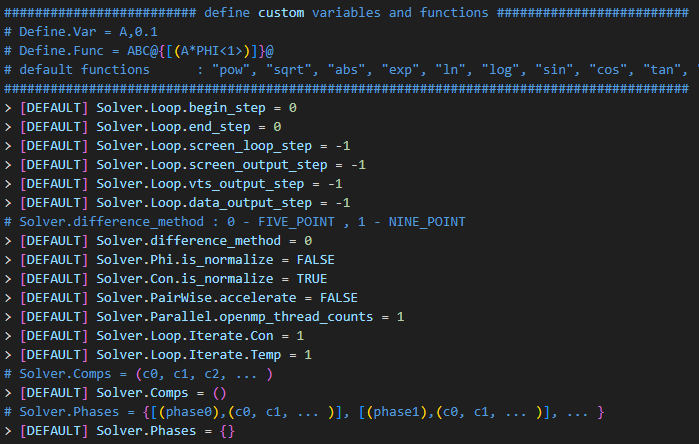


如图，第5-13行是读取自example.mindes输入文件中所有文本。这些文本会被分为3类：注释行<note>、有效行<valid>、无效行<in-valid>，注释及无效行会在进一步识别过程中被忽略，而有效行会被软件管理（如第15行）。

同样我们可以在输入文件中定义变量和函数，以辅助进行一些复杂的输入，这些定义在如下图的容器中被测试（本输入文本未定义变量和函数）：



然后，后面所有的是对输入命令行的识别，我们可以看到有关于命令行的解释，和对应命令的读取情况：



命令前方的[DEFAULT]代表没有读取到这个命令，因此采用默认值。而读取到的命令，前方为[-VALID-]：

> [-VALID-] InputFile.debug = TRUE

部分命令行的读取意味着开启某个模块，从而会解锁更多的可识别的命令行。因此使用者需要通过input\_report.txt及已有的案例来反复测试和修改输入文件，来达到想要的模拟效果。

**解读日志信息（log.txt）**

日志信息会直接被输出在控制台上（不论是Windows还是Linux），同时也会以文件的形式（log.txt）存储在输出文件夹中。例如输出文件夹example中的log.txt：

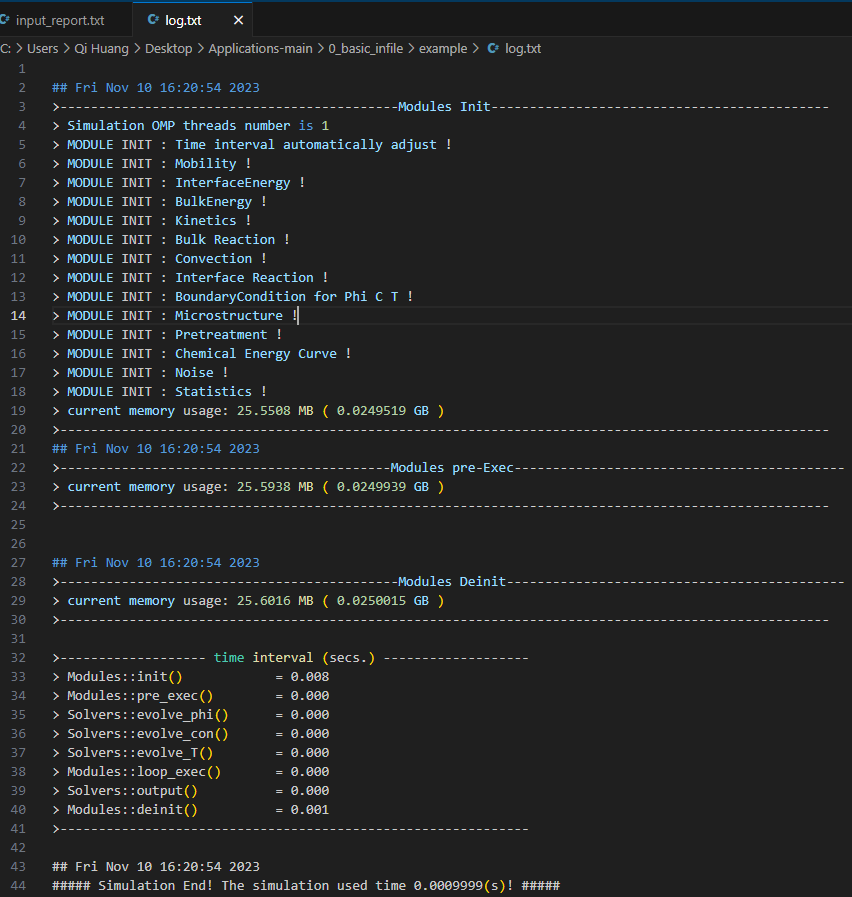
其中，Modules Init为模块初始化区域，这里将提示OpenMP使用的线程数，初始化过程中尝试启动的模块及内存的消耗。

然后是Modules pre-Exec区域，这里将执行预处理功能，执行完毕后同样会统计内存的消耗。

接着在Modules pre-Exec和Modules Deinit之间是程序的循环主体，用于输出模拟过程中的相、组分、温度的演化信息。

最后，执行Modules Deinit来释放模拟网格空间。

在程序结束前，会统计模拟各阶段的耗时，以辅助开发者进行程序高效运行设计。



**可视化（.vts）**

需要安装paraview软件辅助MInDes进行可视化。

在循环过程中，可以通过一些命令来控制输出：

> [DEFAULT] Solver.Loop.screen\_loop\_step = -1

> [DEFAULT] Solver.Loop.screen\_output\_step = -1

> [DEFAULT] Solver.Loop.vts\_output\_step = -1

> [DEFAULT] Solver.Loop.data\_output\_step = -1

其中screen\_loop\_step控制一些无运算消耗的输出，用于提示循环执行到第几步，以及phi、c、T的收敛情况；screen\_output\_step控制一些相平均分数、浓度平均分数的统计情况，需要消耗少量运算进行统计；vts\_output\_step控制vts展示文件的输出；data\_output\_step控制网格数据备份输出。在这四个命令默认的情况下，都不会进行输出。

对于展示文件vts，在主求解器和各个模块中，有相应的命令来控制一些数据的输出：

> [DEFAULT] Postprocess.PCT.VTS.phi = FALSE

> [DEFAULT] Postprocess.PCT.VTS.con = FALSE

> [DEFAULT] Postprocess.PCT.VTS.potential = FALSE

> [DEFAULT] Postprocess.PCT.VTS.energy\_density = FALSE

> [DEFAULT] Postprocess.PCT.VTS.temperature = FALSE

> [DEFAULT] Postprocess.PCT.VTS.phase\_con = FALSE

> [DEFAULT] Postprocess.PCT.VTS.phase\_potential = FALSE

> [DEFAULT] Postprocess.PCT.VTS.grains\_rev = FALSE

> [DEFAULT] Postprocess.PCT.VTS.phi\_index = FALSE

> [DEFAULT] Postprocess.PCT.VTS.phi\_gradient = FALSE

> [DEFAULT] Postprocess.PCT.VTS.phi\_name = FALSE

> [DEFAULT] Postprocess.PCT.VTS.phi\_summary = FALSE

> [DEFAULT] Postprocess.PCT.VTS.interface\_flag = FALSE

在输出的vts中，scalar代表输出数据为标量，vec3代表输出数据为向量



打开vts文件后，会自动来到paraview界面，或者可以在paraview软件中批量打开vts文件，以形成动图。paraview的操作需自行学习。

**再（套嵌）模拟（.dat）**

输出的.dat文件包含模拟网格中相分数、浓度、温度及自定义变量信息，使用者可以通过命令来控制是否通过.dat文件来初始化一个模拟网格

> [DEFAULT] Preprocess.Microstructure.is\_datafile\_init = FALSE

在初始化过程中，需要注意模拟网格、相和组分等信息与.dat文件存储的是否匹配，否则将导致难以预料的后果。

在xxxx\2\_preprocess文件夹中的3\_merge\_phis\_auto.mindes案例就用到了该方法，可用作参考。在Windows下也可通过文件浏览系统索引需要的.dat文件。

## 基本输入参数简介

1. > [DEFAULT] Solver.Loop.begin\_step = 0

模拟开始步

1. > [DEFAULT] Solver.Loop.end\_step = 0

模拟结束步

1. > [DEFAULT] Solver.Loop.screen\_loop\_step = -1

模拟步 及 phi、c、T收敛情况提示

1. > [DEFAULT] Solver.Loop.screen\_output\_step = -1

综合屏幕输出步

1. > [DEFAULT] Solver.Loop.vts\_output\_step = -1

vts文件输出步

1. > [DEFAULT] Solver.Loop.data\_output\_step = -1

网格数据文件输出步

1. # Solver.difference\_method : 0 - FIVE\_POINT , 1 - NINE\_POINT
2. > [DEFAULT] Solver.difference\_method = 0

差分方法

1. > [DEFAULT] Solver.Phi.is\_normalize = FALSE
2. > [DEFAULT] Solver.Con.is\_normalize = TRUE

相及组分是否归一

1. > [DEFAULT] Solver.PairWise.accelerate = FALSE

反对称相场模型加速

1. > [DEFAULT] Solver.Parallel.openmp\_thread\_counts = 1

并行线程数

1. > [DEFAULT] Solver.Loop.Iterate.Con = 1
2. > [DEFAULT] Solver.Loop.Iterate.Temp = 1

多步迭代，一个相场步对应的多个浓度场步及温度场步（暂未调试）

1. # Solver.Comps = (c0, c1, c2, ... )
2. > [DEFAULT] Solver.Comps = ()

体系内的组分定义

1. # Solver.Phases = {[(phase0),(c0, c1, ... )], [(phase1),(c0, c1, ... )], ... }
2. > [DEFAULT] Solver.Phases = {}

体系内的相定义

1. # Solver.GrainsOrientations = {[(phi\_index\_0, phi\_index\_2, ... ),(rotation\_angle\_1, rotation\_angle\_2, rotation\_angle\_3)],  ... }
2. > [DEFAULT] Solver.GrainsOrientations = {}

定义晶粒的取向（未调试）

1. # Solver.GrainsOrientations.rotation\_gauge = 0 - XYX, 1 - XZX, 2 - YXY, 3 - YZY, 4  - ZXZ, 5  - ZYZ
2. #                                            6 - XYZ, 7 - XZY, 8 - YXZ, 9 - YZX, 10 - ZXY, 11 - ZYX
3. > [DEFAULT] Solver.GrainsOrientations.rotation\_gauge = 4

晶粒取向的旋转准则

1. > [DEFAULT] Solver.PCT.RealTime.init = 0.000000

起始的真实时间

1. > [DEFAULT] Solver.PCT.TimeInterval.dt = 1.000000

模拟的时间步长

1. # Solver.PCT.TimeInterval.auto\_adjust = (delt\_step,max\_scale,is\_reduce\_output)
2. > [DEFAULT] Solver.PCT.TimeInterval.auto\_adjust = (100,1e3,true)

自动调整时间步长，以使phi、c、T演化稳定

1. > [DEFAULT] Solver.PCT.phi\_increment\_limit = 0.001000
2. > [DEFAULT] Solver.PCT.con\_increment\_limit = 0.001000
3. > [DEFAULT] Solver.PCT.temp\_increment\_limit = 0.001000

自动调整时间步长过程中，期望phi、c、T演化的精度

1. > [DEFAULT] Solver.Mesh.Nx = 1
2. > [DEFAULT] Solver.Mesh.Ny = 1
3. > [DEFAULT] Solver.Mesh.Nz = 1

模拟空间的大小，为使并行有效需以定义x方向网格优先

1. > [DEFAULT] Solver.Mesh.dr = 1.000000

模拟网格的大小

1. # Solver.Mesh.BoundaryCondition : 0 - FIXED , 1 - PERIODIC , 2 - ADIABATIC
2. > [DEFAULT] Solver.Mesh.BoundaryCondition.x\_up = 1
3. > [DEFAULT] Solver.Mesh.BoundaryCondition.x\_down = 1
4. > [DEFAULT] Solver.Mesh.BoundaryCondition.y\_up = 1
5. > [DEFAULT] Solver.Mesh.BoundaryCondition.y\_down = 1
6. > [DEFAULT] Solver.Mesh.BoundaryCondition.z\_up = 1
7. > [DEFAULT] Solver.Mesh.BoundaryCondition.z\_dowm = 1

边界条件

1. # ModelsManager.Phi.equation : 0 - Const, 1 - AllenCahn Standard, 2 - AllenCahn Pairwise, 3 - CahnHilliard Standard
2. > [DEFAULT] ModelsManager.Phi.equation = 0

相场方程

1. # ModelsManager.Con.equation : 0 - Const, 1 - TotalConcentration, 2 - PhaseConcentration, 3 - GrandPotential
2. > [DEFAULT] ModelsManager.Con.equation = 0

浓度场方程

1. # ModelsManager.Con.valid\_domain : 0 - Standard, 1 - Reverse
2. > [DEFAULT] ModelsManager.Con.valid\_domain = 0

浓度场有效区域

1. # ModelsManager.Temp.equation : 0 - Const, 1 - Standard
2. > [DEFAULT] ModelsManager.Temp.equation = 0

温度场方程

1. > [DEFAULT] Preprocess.Microstructure.is\_datafile\_init = FALSE

是否以数据网格文件初始化一个模拟

1. # .matrix = {[(phi\_index),(phi\_name),(phi\_comp\_0\_value, phi\_comp\_1\_value, ... )],[(total\_comp\_0\_value, total\_comp\_1\_value, ... )],[(temp\_value)]}
2. > [DEFAULT] Preprocess.Microstructure.matrix = {[()]}

模拟基体的定义

1. # .property = [(phi\_index\_begin, phi\_index\_end), (phi\_name, ... ),(phi\_weight, ...)]
2. > [DEFAULT] Preprocess.Microstructure.voronoi.property = [()]

生成多晶结构

1. # .property = (bmp\_layer, file\_name)
2. > [DEFAULT] Preprocess.Microstructure.bmp24.property = ()

从图片读取二维结构

1. > [DEFAULT] Preprocess.Microstructure.geometry\_layer\_number = 0

在几何区域内拓展网格结构和生成网格数据

1. # .property = [(phi\_index\_begin, phi\_index\_end), (phi\_name, ... ),(phi\_weight, ...),(in phi\_index, ... )]
2. > [DEFAULT] Preprocess.Microstructure.voronoi\_inPhis.property = [()]

定义位于某个/些相区内的多晶结构

1. # Preprocess.reconstruct\_phis = {[(phi\_index\_0, phi\_index\_1, ... ), (phi\_name)], .... }
2. > [DEFAULT] Preprocess.reconstruct\_phis = {[()]}

重构某些晶粒的性质

1. # Preprocess.merge\_phis = {[(phi0,phi1,phi2, ... ), is\_phi\_c\_merge], .... }
2. > [DEFAULT] Preprocess.merge\_phis = {[()]}

混合某些相形成一个相

1. # Preprocess.auto\_merge\_phis = (phis\_index\_0, phis\_index\_1, ... )
2. > [DEFAULT] Preprocess.auto\_merge\_phis = ()

自动混合互相不接触的多个相

1. # Preprocess.relax\_interface = (relax\_steps, output\_steps, fix\_phi\_after\_relax)
2. > [DEFAULT] Preprocess.relax\_interface = ()

弛豫相界面

1. > [DEFAULT] Preprocess.remove\_inexistent\_phis = FALSE

剔除不存在的相，减少内存空间消耗

1. > [DEFAULT] Preprocess.re\_ordering\_phis\_indexs\_from = 0

对各个相的索引再排序

1. # Preprocess.fill\_phis = {[(phi\_index\_0, phi\_index\_1, ... ), (phi\_con\_1, phi\_con\_2, ... ), (total\_con\_1, total\_con\_2, ... ), (temperature)], .... }
2. > [DEFAULT] Preprocess.fill\_phis = {[()]}

填充相区内的组分、温度数据

1. # Postprocess.physical\_fields = (mechanic, fluid dynamic, electric)
2. > [DEFAULT] Postprocess.physical\_fields = (false,false,false)
3. > [-VALID-] Postprocess.physical\_fields(0) = FALSE
4. > [-VALID-] Postprocess.physical\_fields(1) = FALSE
5. > [-VALID-] Postprocess.physical\_fields(2) = FALSE

开启外场：机械场、流场、电场

1. > [DEFAULT] Postprocess.Statistics.file\_name = data\_statistics
2. > [DEFAULT] Postprocess.Statistics.is\_phi\_c\_t = FALSE
3. > [DEFAULT] Postprocess.Statistics.is\_electricity = FALSE
4. # Postprocess.Statistics.datafiles = (datafile1, ... )
5. > [DEFAULT] Postprocess.Statistics.datafiles = ()

统计网格内的数据并进行输出

1. > [DEFAULT] Postprocess.PCT.VTS.phi = FALSE
2. > [DEFAULT] Postprocess.PCT.VTS.con = FALSE
3. > [DEFAULT] Postprocess.PCT.VTS.potential = FALSE
4. > [DEFAULT] Postprocess.PCT.VTS.energy\_density = FALSE
5. > [DEFAULT] Postprocess.PCT.VTS.temperature = FALSE
6. > [DEFAULT] Postprocess.PCT.VTS.phase\_con = FALSE
7. > [DEFAULT] Postprocess.PCT.VTS.phase\_potential = FALSE
8. > [DEFAULT] Postprocess.PCT.VTS.grains\_rev = FALSE
9. > [DEFAULT] Postprocess.PCT.VTS.phi\_index = FALSE
10. > [DEFAULT] Postprocess.PCT.VTS.phi\_gradient = FALSE
11. > [DEFAULT] Postprocess.PCT.VTS.phi\_name = FALSE
12. > [DEFAULT] Postprocess.PCT.VTS.phi\_summary = FALSE
13. > [DEFAULT] Postprocess.PCT.VTS.interface\_flag = FALSE

一些可以在vts文件中输出的数据，是否输出的开关

1. > [DEFAULT] <AUTO> Solver.Loop.stop = FALSE

正常停止一个运行程序的开关，程序会跳出未完成的主循环并进行结束输出和内存释放操作。

# 模块介绍

## 微结构初始化

在第二节的介绍中，结构初始化由以下命令行控制：

# .matrix = {[(phi\_index),(phi\_name),(phi\_comp\_0\_value, phi\_comp\_1\_value, ... )],[(total\_comp\_0\_value, total\_comp\_1\_value, ... )],[(temp\_value)]}

> [DEFAULT] Preprocess.Microstructure.matrix = {[()]}

# .property = [(phi\_index\_begin, phi\_index\_end), (phi\_name, ... ),(phi\_weight, ...)]

> [DEFAULT] Preprocess.Microstructure.voronoi.property = [()]

# .property = (bmp\_layer, file\_name)

> [DEFAULT] Preprocess.Microstructure.bmp24.property = ()

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_number = 0

# .property = [(phi\_index\_begin, phi\_index\_end), (phi\_name, ... ),(phi\_weight, ...),(in phi\_index, ... )]

> [DEFAULT] Preprocess.Microstructure.voronoi\_inPhis.property = [()]

**首先使用者需要声明微结构的基体，所有的结构都会被初始化在这个基体上：**

Preprocess.Microstructure.matrix

**接着可以在基体上生成几何体，每个几何体位于一个几何层中，几何层代表了生成几何体的先后顺序，后生成的几何体会覆盖先生成的几何体，因此可以通过设计几何层形成层次感：**

Preprocess.Microstructure.geometry\_layer\_number = 1

然后，使用者需要声明几何层的一些属性。如仅有一层几何层，只需要声明第一层的属性（按0开始排序）

# .property = (phi\_index, phi\_name, geometry\_type, rotation\_gauge, reverse\_region)

#              geometry\_type  : 0 - None, 1 - Ellipsoid, 2 - Polyhedron

#              rotation\_gauge : 0 - XYX, 1 - XZX, 2 - YXY, 3 - YZY, 4  - ZXZ, 5  - ZYZ

#                               6 - XYZ, 7 - XZY, 8 - YXZ, 9 - YZX, 10 - ZXY, 11 - ZYX

> [-VALID-] Preprocess.Microstructure.geometry\_layer\_0.property = (0,G0,0,1,false)

相的索引（phi\_index）用于区分每个晶粒，对应、、等，而相的属性phi\_name可以认为是具有相同物理性质的晶粒。如具有相同的热力学吉布斯自由能函数，具有相同弹性模量等，用于进一步分类。几何类型（geometry\_type）目前支持椭球、多面体的生成。生成步与无量纲模拟步对应，一般情况下设置为第0步，即在模块预处理过程中就会生成结构。旋转规则（rotation\_gauge）可用于二维、三维结构的旋转，一般取几何体的相对坐标轴作为旋转参照物。反向填充（reverse\_region）即为填充几何体外的部分。

* 初始化一个椭球/椭圆

如果在设置几何层时，定义几何类型为椭球，使用者就需要在该几何层上进一步定义椭球的参数，及几何区域内的数据，如下

# .ellipsoid = [(core\_x,core\_y,core\_z),(radius\_x,radius\_y,radius\_z),(rotation\_angle\_1,rotation\_angle\_2,rotation\_angle\_3)]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.ellipsoid = [(0,0,0),(0,0,0),(0,0,0)]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.T = 0.000000

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.phi = 1.000000

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.is\_normalized = TRUE

# .x = [(comp\_0\_name,comp\_0\_value),(comp\_1\_name,comp\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.x = [()]

# .custom\_int = [(custom\_0\_index, custom\_0\_value),(custom\_1\_index, custom\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_int = [()]

# .custom\_double = [(custom\_0\_index, custom\_0\_value),(custom\_1\_index, custom\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_double = [()]

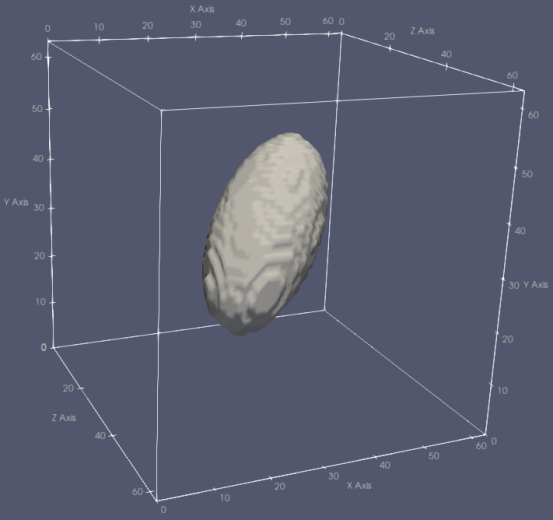
# .custom\_vec3 = [(custom\_0\_index, custom\_0\_value\_0, custom\_0\_value\_1, custom\_0\_value\_2), (custom\_1\_index, custom\_1\_value\_0, custom\_1\_value\_1, custom\_1\_value\_2), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_vec3 = [()]

# .custom\_vec6 = [(custom\_vec6\_index, custom\_vec6\_value\_0, custom\_vec6\_value\_1, custom\_vec6\_value\_2, custom\_vec6\_value\_3, custom\_vec6\_value\_4, custom\_vec6\_value\_5), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_vec6 = [()]

案例详见1\_nucleation/0\_ellipse.mindes 及 1\_nucleation/1\_ellipsoid.mindes



* 初始化一个多边形/多面体

如果在设置几何层时，定义几何类型为多面体，使用者就需要在该几何层上进一步定义多面体的参数

# .polyhedron = {[inside\_point],[surf\_point,surf\_point,surf\_point], .... ,[(rotation\_angle\_1,rotation\_angle\_2,rotation\_angle\_3)]}

#                point = (position\_x,position\_y,position\_z)

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.polyhedron = {[(-1,0,0)],[(0,0,0),(0,1,0),(0,0,1)],[(0,0,0)]}

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.T = 0.000000

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.phi = 1.000000

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.is\_normalized = TRUE

# .x = [(comp\_0\_name,comp\_0\_value),(comp\_1\_name,comp\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.x = [()]

# .custom\_int = [(custom\_0\_index, custom\_0\_value),(custom\_1\_index, custom\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_int = [()]

# .custom\_double = [(custom\_0\_index, custom\_0\_value),(custom\_1\_index, custom\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_double = [()]

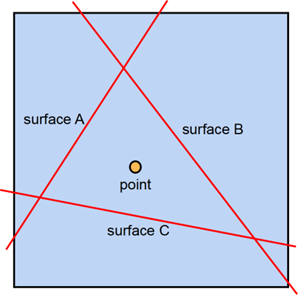
# .custom\_vec3 = [(custom\_0\_index, custom\_0\_value\_0, custom\_0\_value\_1, custom\_0\_value\_2), (custom\_1\_index, custom\_1\_value\_0, custom\_1\_value\_1, custom\_1\_value\_2), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_vec3 = [()]

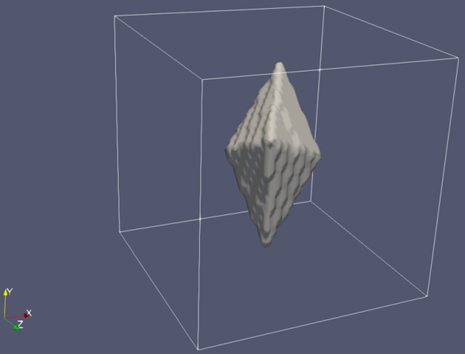
# .custom\_vec6 = [(custom\_vec6\_index, custom\_vec6\_value\_0, custom\_vec6\_value\_1, custom\_vec6\_value\_2, custom\_vec6\_value\_3, custom\_vec6\_value\_4, custom\_vec6\_value\_5), ...]

> [DEFAULT] Preprocess.Microstructure.geometry\_layer\_0.custom\_vec6 = [()]

采用面切割+体内点锚定的方法确定一个多面体/多边形，该方法也可轻易拓展到Voronoi结构生成逻辑中。

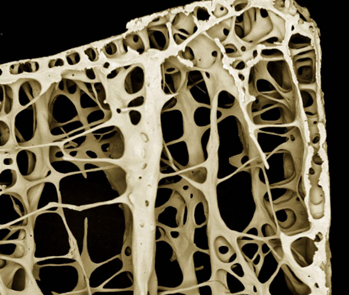


案例见1\_nucleation/2\_polygon.mindes 及 1\_nucleation/3\_polyhedron.mindes



**除了几何体形核之外，还可以根据图片（bmp24格式）来初始化一个更加复杂/自然的结构：**

比如面对下图的复杂结构，



调用bmp24方法，

# .property = (bmp\_layer, file\_name)

> [DEFAULT] Preprocess.Microstructure.bmp24.property = ()

# .phi = (phi\_index, phi\_name, phi\_fraction)

> [DEFAULT] Preprocess.Microstructure.bmp24\_0.phi = ()

# .x = [(comp\_0\_name,comp\_0\_value), ...]

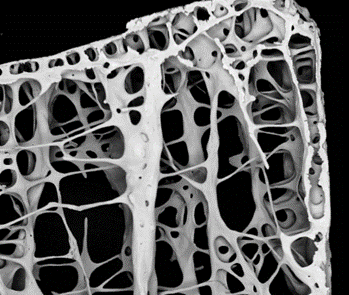
> [DEFAULT] Preprocess.Microstructure.bmp24\_0.x = [()]

> [DEFAULT] Preprocess.Microstructure.bmp24\_0.gray\_threshold = (0.0,1.0)

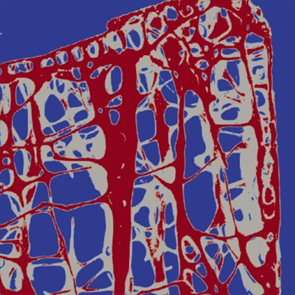
> [DEFAULT] Preprocess.Microstructure.bmp24\_0.temperature = 0.000000

> [DEFAULT] Preprocess.Microstructure.bmp24\_0.is\_normalized = TRUE

该方法会读取bmp24格式的图片，并将其转化为灰度图片，如图



bmp24方法自带图层（bmp\_layer），可以分步从图片冲读取不同灰度的区域，然后对于该区域可以设置想生成相的索引（phi\_index），相的性质（phi\_name），及相的值（phi\_fraction）。对于该区域也可以生成对应组分值（x）。可以设置该区域的灰度值范围（gray\_threshold，0~1：黑~白）。对于该区域也可以生成对应组分值（temperature）。也可以设置相分数是否有归一的需求。



具体案例可见1\_nucleation/ 4\_read\_bmp24.mindes，通过假设两个相，在背景相（蓝色）中生成第一个相（红色）及第二个相（白色）。

**Voronoi结构是相场模拟中较为常见的结构，可通过如下命令快速生成基本的voronoi结构：**

# .property = [(phi\_index\_begin, phi\_index\_end), (phi\_name, ... ),(phi\_weight, ...)]

> [DEFAULT] Preprocess.Microstructure.voronoi.property = [()]

# .box = [(box\_origin\_point),(box\_end\_point)]

> [DEFAULT] Preprocess.Microstructure.voronoi.box = [(0,0,0),(0,0,0)]

# .x = [(comp\_0\_name,comp\_0\_value), (comp\_1\_name,comp\_1\_value), ...]

> [DEFAULT] Preprocess.Microstructure.voronoi.x = [()]

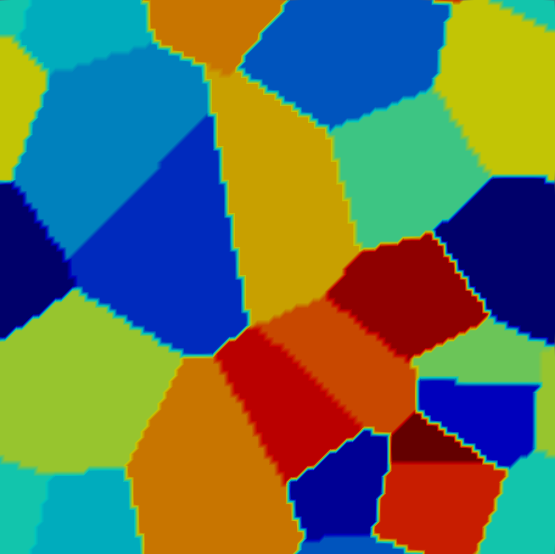
> [DEFAULT] Preprocess.Microstructure.voronoi.temperature = 0.000000

> [DEFAULT] Preprocess.Microstructure.voronoi.rand\_seed = 0

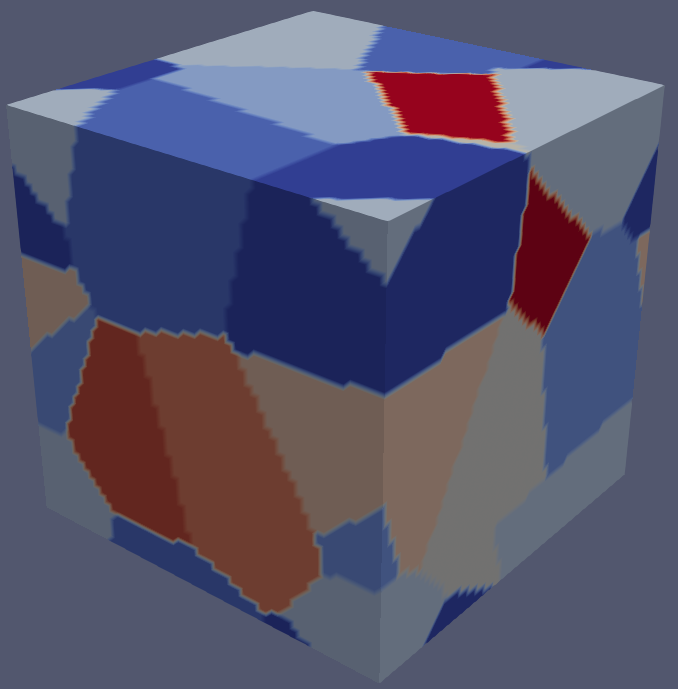
> [DEFAULT] Preprocess.Microstructure.voronoi.point\_distance = -1.000000

首先定义想生成的晶粒索引的范围，随机生成的晶粒的性质和权重。其中盒子为生成voronoi结构的区域（box）。在定义的区域内会随机撒点生成晶粒，晶粒数量由相索引起始和终止决定。各个晶粒具体分别属于什么相，由相名称及权重来随机分配。可以设置区域内的组分（x）和温度（temperature）。同时可以设置撒点时点的间距（point\_distance）。

具体二维案例见1\_nucleation/ 5\_voronoi2d.mindes



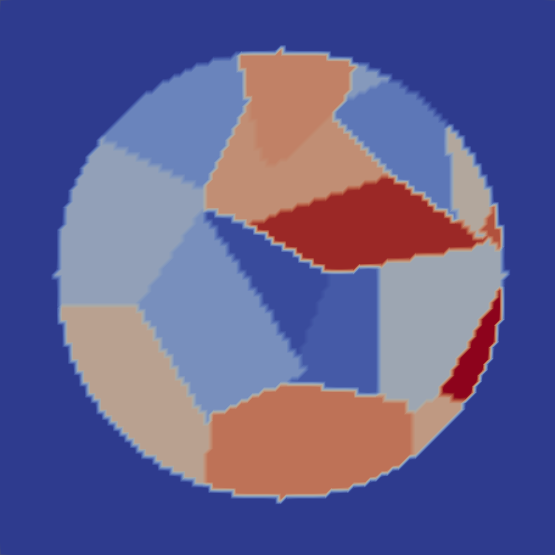
三维案例见1\_nucleation/ 8\_voronoi3d.mindes



**形核功能的叠加：例生成椭圆中的Voronoi**

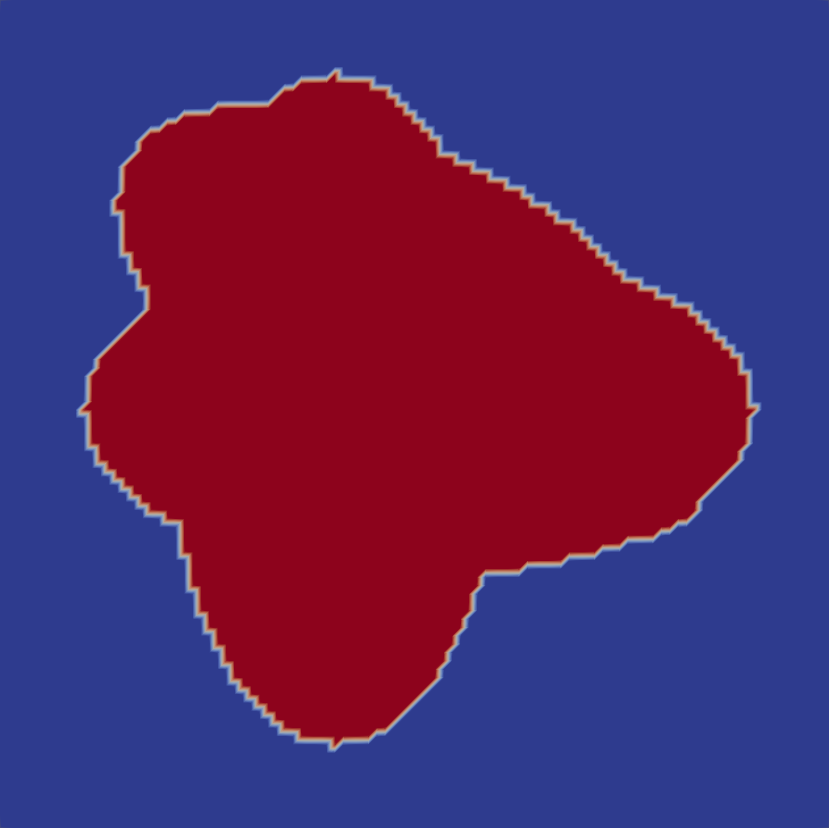
先生成voronoi结构，以及使用ellips反填充。

见1\_nucleation/ 6\_voronoi2d\_ellips.mindes



**形核功能的叠加2：例生成已有区域中的Voronoi**

假设的任意几何区域：



在该区域内生成voronoi结构：

# .property = [(phi\_index\_begin, phi\_index\_end), (phi\_name, ... ),(phi\_weight, ...),(in phi\_index, ... )]

> [-VALID-] Preprocess.Microstructure.voronoi\_inPhis.property = [(1,60),(Grain0),(1.0),(0)]

# .box = [(box\_origin\_point),(box\_end\_point)]

> [-VALID-] Preprocess.Microstructure.voronoi\_inPhis.box = [(0,0,0),(100,100,0)]

# .x = [(comp\_0\_name,comp\_0\_value), (comp\_1\_name,comp\_1\_value), ...]

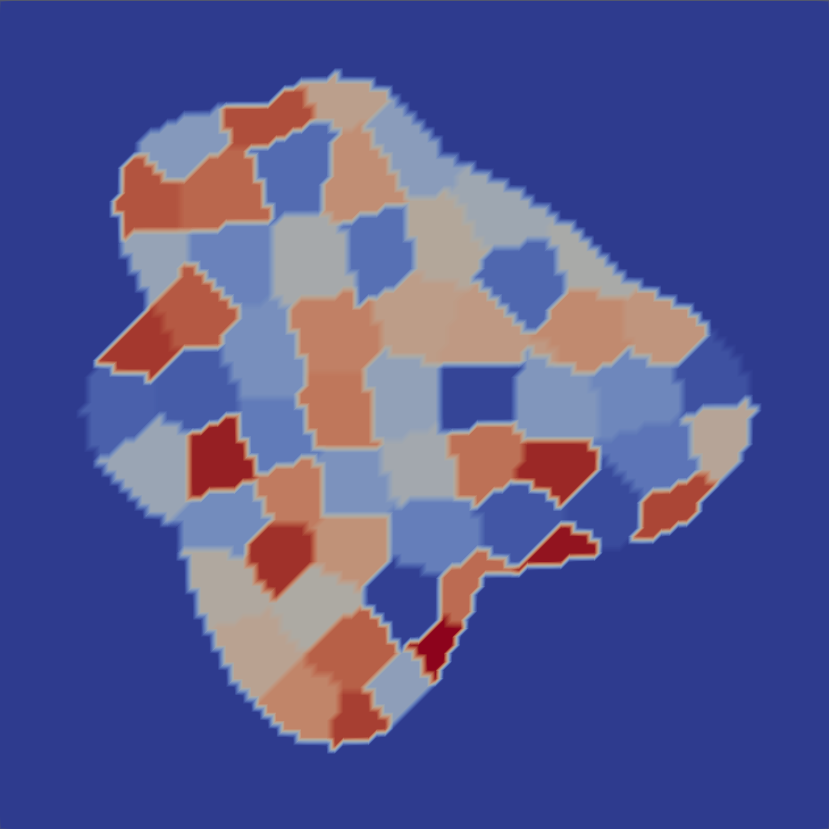
> [DEFAULT] Preprocess.Microstructure.voronoi\_inPhis.x = [()]

> [DEFAULT] Preprocess.Microstructure.voronoi\_inPhis.temperature = 0.000000

> [-VALID-] Preprocess.Microstructure.voronoi\_inPhis.rand\_seed = 0

> [-VALID-] Preprocess.Microstructure.voronoi.point\_distance = 7

与一般情况生成多晶结构类似，不过需要在property的设置末尾注明会在哪些相区内生成多晶结构。



## 预处理

在预处理模块中集成了一些辅助处理功能。

**相重构**

# Preprocess.reconstruct\_phis = {[(phi\_index\_0, phi\_index\_1, ... ), (phi\_name)], .... }

> [DEFAULT] Preprocess.reconstruct\_phis = {[()]}

重构调整读取已有网格结构（.dat）后，相的性质。

**弛豫界面**

案例见2\_pretreatment/1\_relax\_interface.mindes

相场模型计算得到的是扩散界面，在相场法中，部分数值问题源自于还没有形成一个扩散的界面，该功能可在模拟前迭代弛豫出一个扩散的界面，以满足部分需求，比如对于一个具有尖锐界面的多边形结构



通过命令：

# Preprocess.relax\_interface = (relax\_steps, output\_steps, fix\_phi\_after\_relax)

> [DEFAULT] Preprocess.relax\_interface = ()

弛豫结果为：



**相混合**

案例见2\_preprocess/2\_merge\_phis.mindes 和 2\_preprocess/3\_merge\_phis\_auto.mindes。

# Preprocess.merge\_phis = {[(phi0,phi1,phi2, ... ), is\_phi\_c\_merge], .... }

> [DEFAULT] Preprocess.merge\_phis = {[()]}

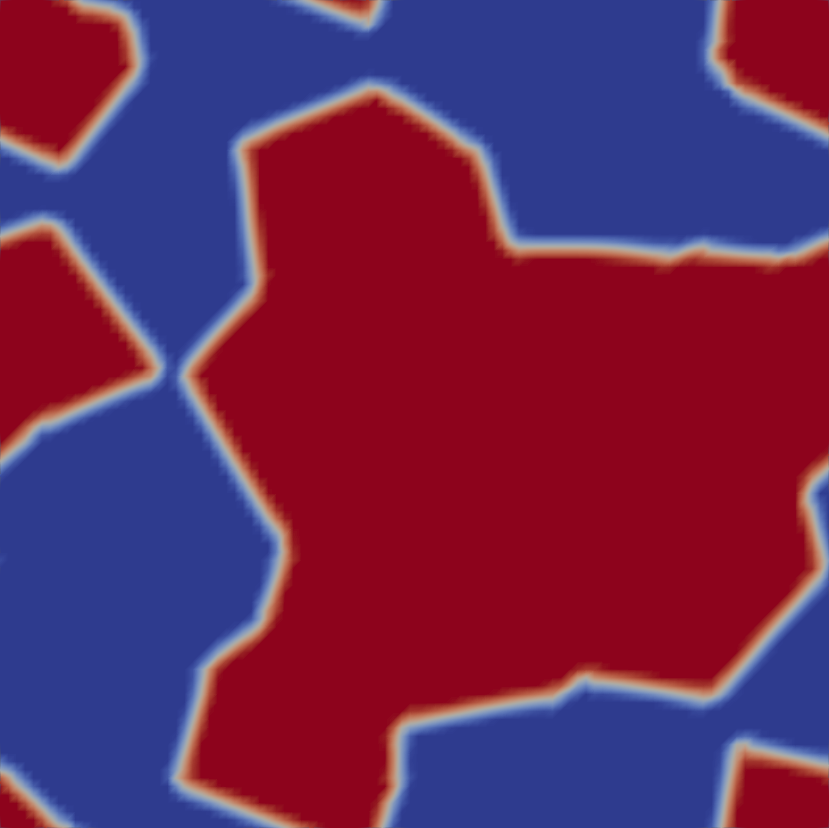
# Preprocess.auto\_merge\_phis = (phis\_index\_0, phis\_index\_1, ... )

> [DEFAULT] Preprocess.auto\_merge\_phis = ()

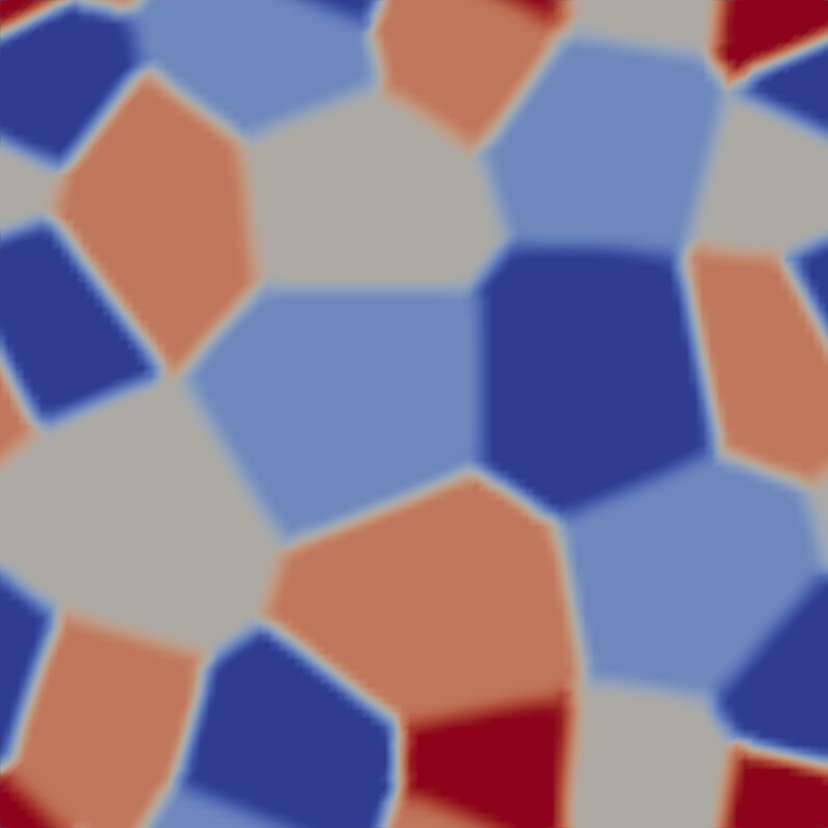
针对如下图所示随机生成的20个晶粒，可以将其中部分晶粒合并（在不影响模拟结果的情况下，可以节约计算内存，提高计算效率）



可将随机生成的20个晶粒混合为两个晶粒：



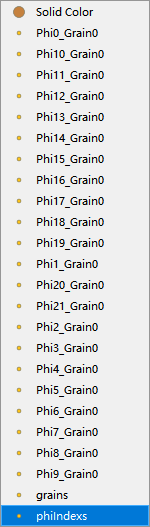
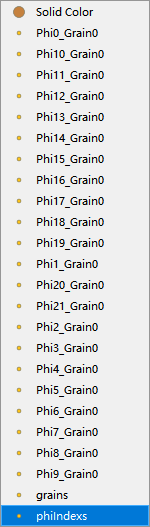
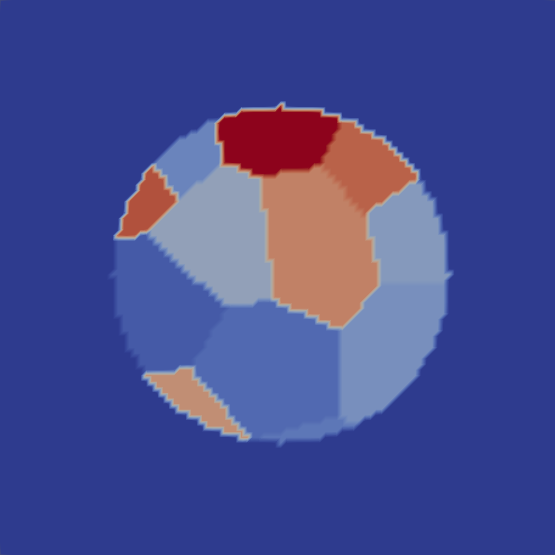
可将随机生成的20个晶粒中不接触的晶粒自动混合



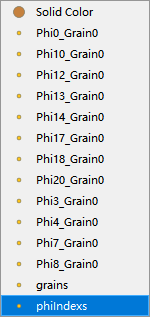
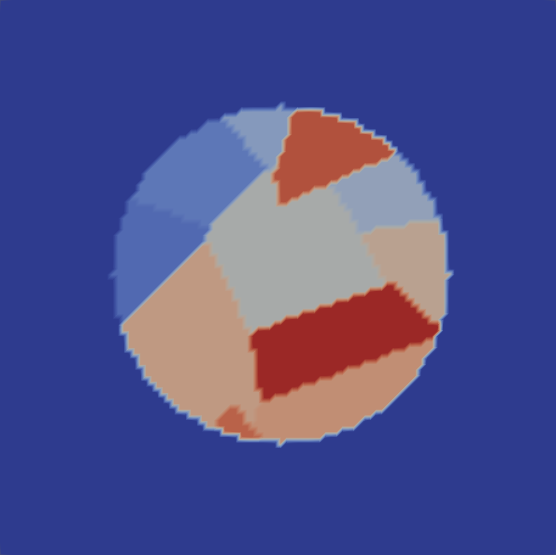
**剔除不存在的相**

> [DEFAULT] Preprocess.remove\_inexistent\_phis = FALSE

在构造复杂结构时难免会有相被覆盖或者消失，为节约内存、提高计算效率，剔除不存在的相是有必要的。以在椭圆中生成voronoi结构为例：



可见在该结构中存在22个晶粒，然而其中大部分在椭圆形核过程中被覆盖了，是不存在的，因此可使用该功能。相体积分数为0.0的晶粒会被自动剔除，结果只剩下12个晶粒：



**相索引重排序：**

> [DEFAULT] Preprocess.re\_ordering\_phis\_indexs\_from = 0

从索引0开始重新给晶粒/序参量排序，使其更加有序。

**相填充**

# Preprocess.fill\_phis = {[(phi\_index\_0, phi\_index\_1, ... ), (phi\_con\_1, phi\_con\_2, ... ), (total\_con\_1, total\_con\_2, ... ), (temperature)], .... }

> [DEFAULT] Preprocess.fill\_phis = {[()]}

该功能可在对应相区中填充组分及温度。

## 相界面

需要模拟相界面，首先需要选择相场动力学方程：

# ModelsManager.Phi.equation : 0 - Const, 1 - AllenCahn Standard, 2 - AllenCahn Pairwise, 3 - CahnHilliard Standard

> [-VALID-] ModelsManager.Phi.equation = 2

**一维、二维相界面**

首先，可以选择界面能的模型：

# ModelsManager.Phi.InterfaceEnergy.int\_gradient : 0 - Steinbach\_1996 , 1 - Steinbach\_1999 , 2 - Steinbach\_G2009

> [-VALID-] ModelsManager.Phi.InterfaceEnergy.int\_gradient = 0

# ModelsManager.Phi.InterfaceEnergy.int\_potential : 0 - Nestler\_Well , 1 - Nestler\_Obstacle , 2 - Steinbach\_P2009

> [-VALID-] ModelsManager.Phi.InterfaceEnergy.int\_potential = 1

梯度能：0 - Steinbach\_1996, 1 - Steinbach\_1999, 2 - Steinbach\_G2009

势能：0 - Nestler\_Well, 1 - Nestler\_Obstacle, 2 - Steinbach\_P2009

# ModelsManager.Phi.Lij.const  = Lij\_value

#                      .matrix = [(phi\_i, phi\_j, Lij\_value), ... ]

#                      .block = [(phi\_begin, phi\_end, Lij\_value), ... ]

> [-VALID-] ModelsManager.Phi.Lij.const = 1

# ModelsManager.Phi.Lij.Const.block = [(phi\_index1, phi\_index2, ... ), ... ]

> [DEFAULT] ModelsManager.Phi.Lij.Const.block = [()]

设置界面迁移率

# ModelsManager.Phi.xi\_ab.const  = xi\_ab

#                        .matrix = [(phi\_a, phi\_b, xi\_ab\_value), ...]

> [-VALID-] ModelsManager.Phi.xi\_ab.const = 1

# ModelsManager.Phi.xi\_abc.const  = xi\_ab

#                         .matrix = [(phi\_a, phi\_b, phi\_c, xi\_abc\_value), ...]

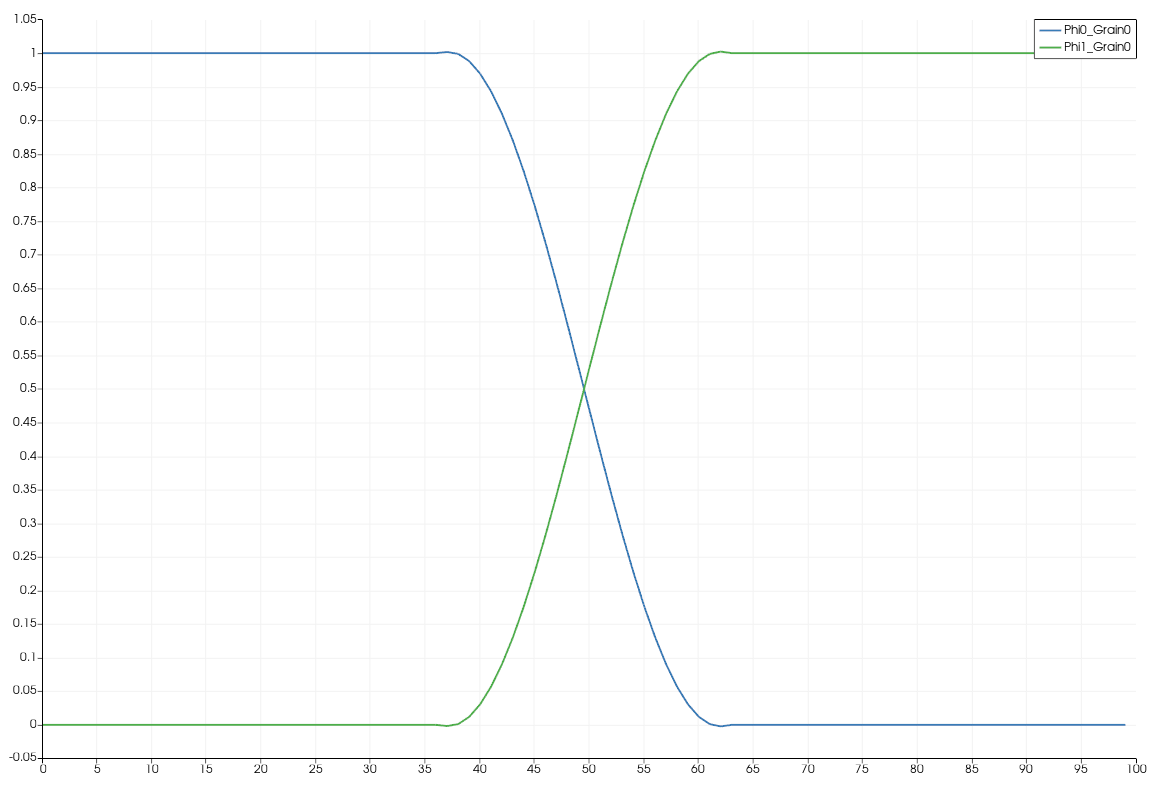
> [-VALID-] ModelsManager.Phi.xi\_abc.const = 0

设置界面能

> [-VALID-] ModelsManager.Phi.InterfaceEnergy.int\_width = 10

设置界面宽度

案例见3\_PCT\_interface \1\_two\_phis\_diffuse\_interface\ allen\_cahn\_pairwise\_1d.mindes

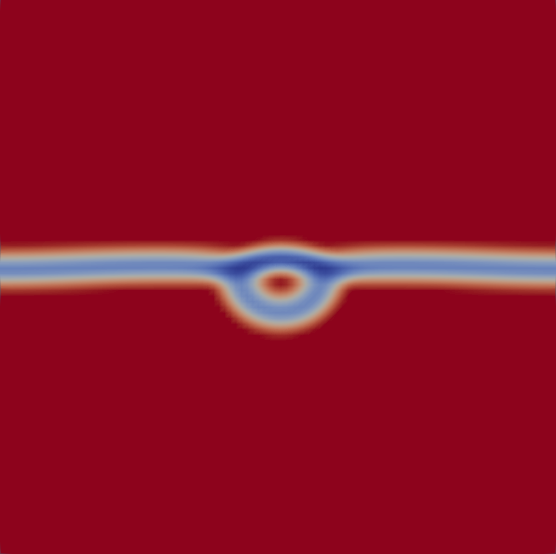


案例见3\_PCT\_interface\2\_three\_phis\_junction\allen\_cahn\_pairwise\_2d.mindes



**多相结**

案例见3\_PCT\_interface\2\_three\_phis\_junction\allen\_cahn\_pairwise\_curvature.mindes

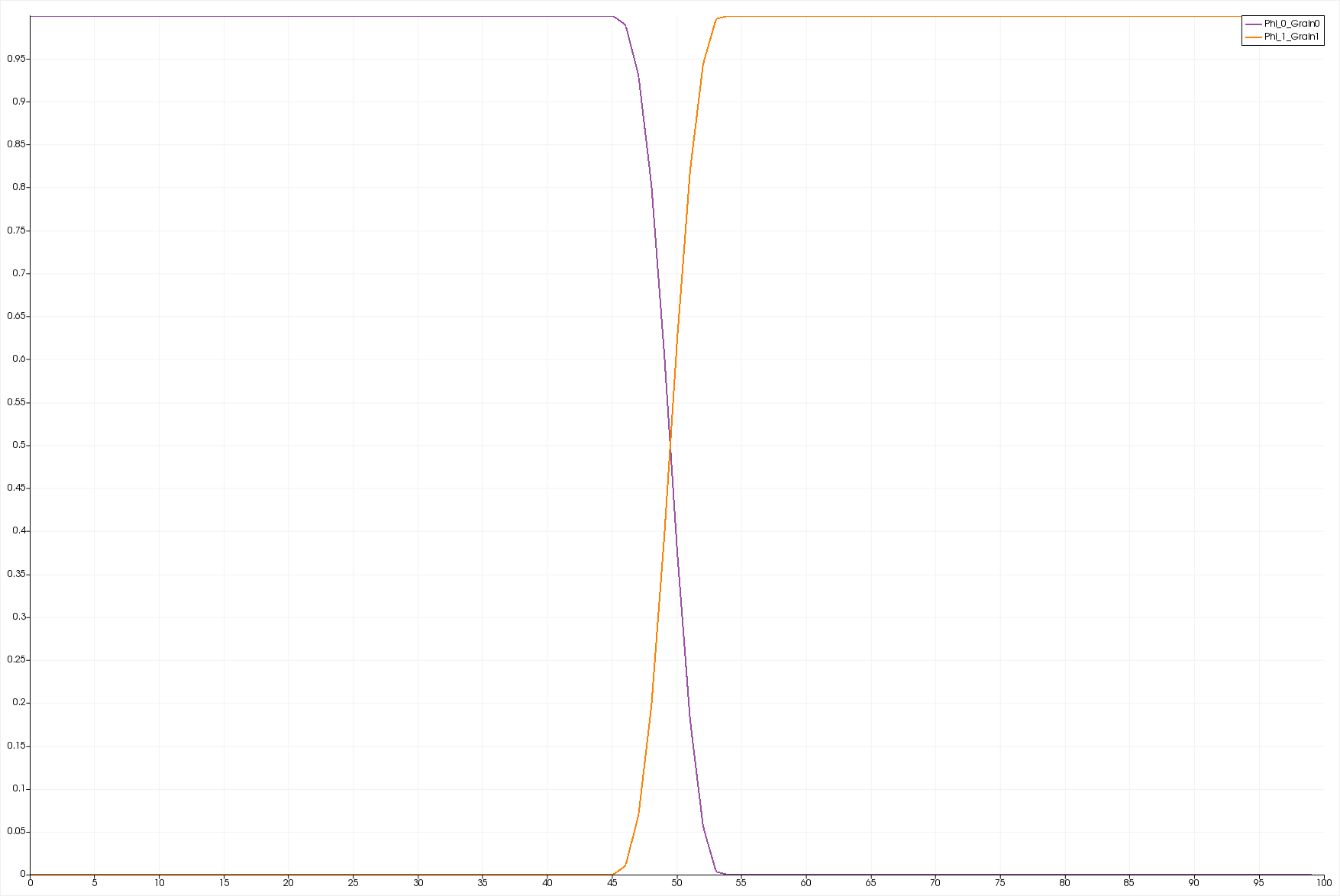
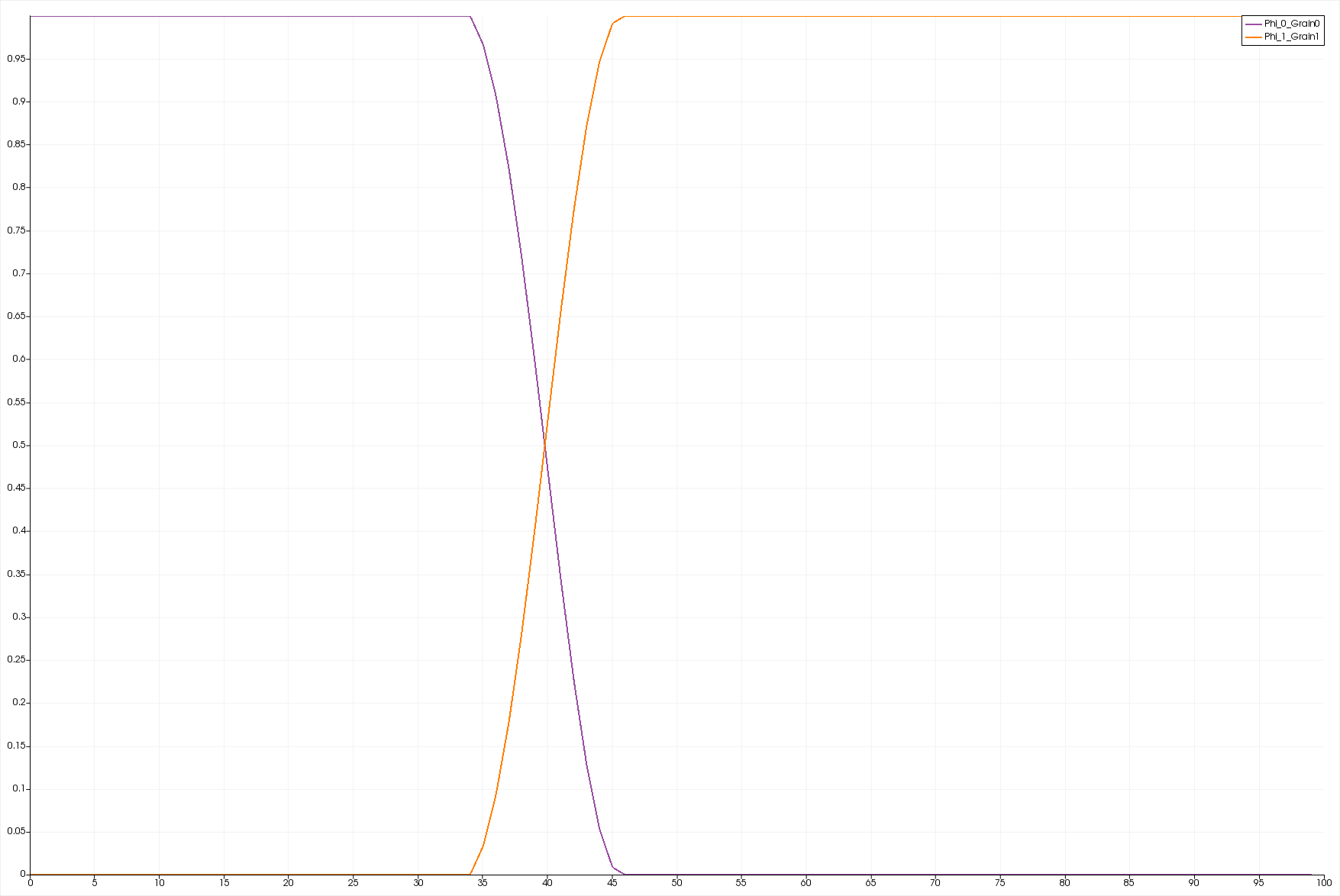


**给一个界面恒定驱动**

相场框架下，界面的驱动力由体相能及组分势来驱动，假设相具有一个恒定的体能

# ModelsManager.Phi.BulkEnergy.const = [(phi\_name, bulk\_energy), ... ]

> [-VALID-] ModelsManager.Phi.BulkEnergy.const = [(Grain0,0.0),(Grain1,-0.01)]

>>>>

其中紫色（Grain0）、黄色（Grain1）实线为两相体积分数分布，设定中Grain1体能量密度更低，所以黄色相长大。

**界面上的相浓度**

浓度场的演化，需先选择浓度场动力学方程：

# ModelsManager.Con.equation : 0 - Const, 1 - TotalConcentration, 2 - PhaseConcentration, 3 - GrandPotential

> [DEFAULT] ModelsManager.Con.equation = 0

然后需要定义参数

# ModelsManager.Con.valid\_domain : 0 - Standard, 1 - Reverse

> [DEFAULT] ModelsManager.Con.valid\_domain = 0

> [-VALID-] ModelsManager.Con.ValidDomain.phase\_indexes = (0)

> [-VALID-] ModelsManager.Con.ValidDomain.threshold = 0.5

> [DEFAULT] ModelsManager.Con.GrandPotential.range = (-100000.0,100000.0)

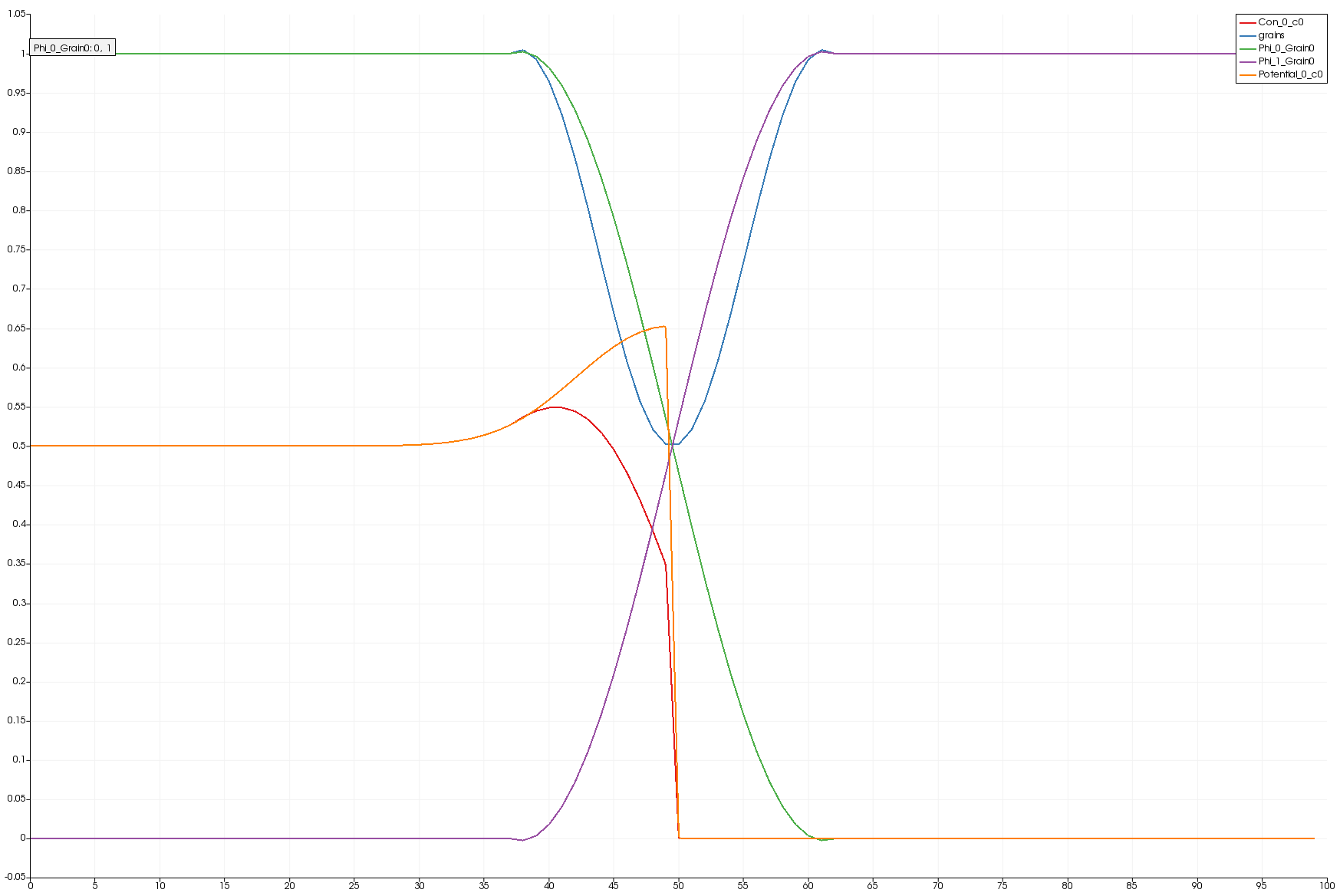
其中，valid\_domain是设置标准有效区域还是反转的有效区域。phase\_indexes对应的是有效的相，反转则是排除这个相是有效区域。Threshold是判断有效相相分数的判据，range是对大势方程值的截断。

# ModelsManager.Con.Mii = [(phase\_0\_M\_00 , phase\_0\_M\_11, ...) , ... ]

> [DEFAULT] ModelsManager.Con.Mii = [()]

为每个相每个组分定义化学扩散速率。

案例见3\_PCT\_interface\5\_concentration\_on\_interface\3\_smooth\_grand\_potential\ allen\_cahn\_pair\_wise\_1d.mindes 大势方程的演化：



其中绿色、紫色实线为两相体积分数分布，红色线为总组分的分布，黄色线为绿色相区的扩散势分布。其中绿色相区被设定为有效区域。

## 固体力学

打开物理场中的机械场开关

# Postprocess.physical\_fields = (mechanic, fluid dynamic, electric)

> [-VALID-] Postprocess.physical\_fields = (true,false,false)

然后设置机械场-弹性求解参数

# Postprocess.SolidMechanics.momentum\_balance = 0 - None , 1 - Explicit , 2 - Implicit (Ingo Steinbach) , 3 - Implicit (Armen G. Khachaturyan)

> [-VALID-] Postprocess.SolidMechanics.momentum\_balance = 2

> [DEFAULT] Postprocess.SolidMechanics.restart\_iterator\_in\_loop = FALSE

> [DEFAULT] Postprocess.SolidMechanics.Implicit.bc\_ite\_rate = 1.000000

# Postprocess.SolidMechanics.fix\_boundary.type = (BC\_X, BC\_Y, BC\_Z) , 0 - Average , 1 - Strain , 2 - Stress

> [-VALID-] Postprocess.SolidMechanics.fix\_boundary.type = (0,0,0)

> [-VALID-] Postprocess.SolidMechanics.write\_displacement\_field = TRUE

> [-VALID-] Postprocess.SolidMechanics.max\_iteration\_steps = 100

> [-VALID-] Postprocess.SolidMechanics.debug = TRUE

> [-VALID-] Postprocess.SolidMechanics.strain\_accuracy = 1e-07

> [-VALID-] Postprocess.SolidMechanics.solid\_phases = (Grain0,Grain1)

> [-VALID-] Postprocess.SolidMechanics.plasticity = FALSE

> [-VALID-] Postprocess.SolidMechanics.Stiffness.Grain0 = [(C11,C12,C12,0,0,0),(C12,C11,C12,0,0,0),(C12,C12,C11,0,0,0),(0,0,0,Gm,0,0),(0,0,0,0,Gm,0),(0,0,0,0,0,Gm)]

> [-VALID-] Postprocess.SolidMechanics.EigenStrain.Grain0 = (0,0,0,0,0,0)

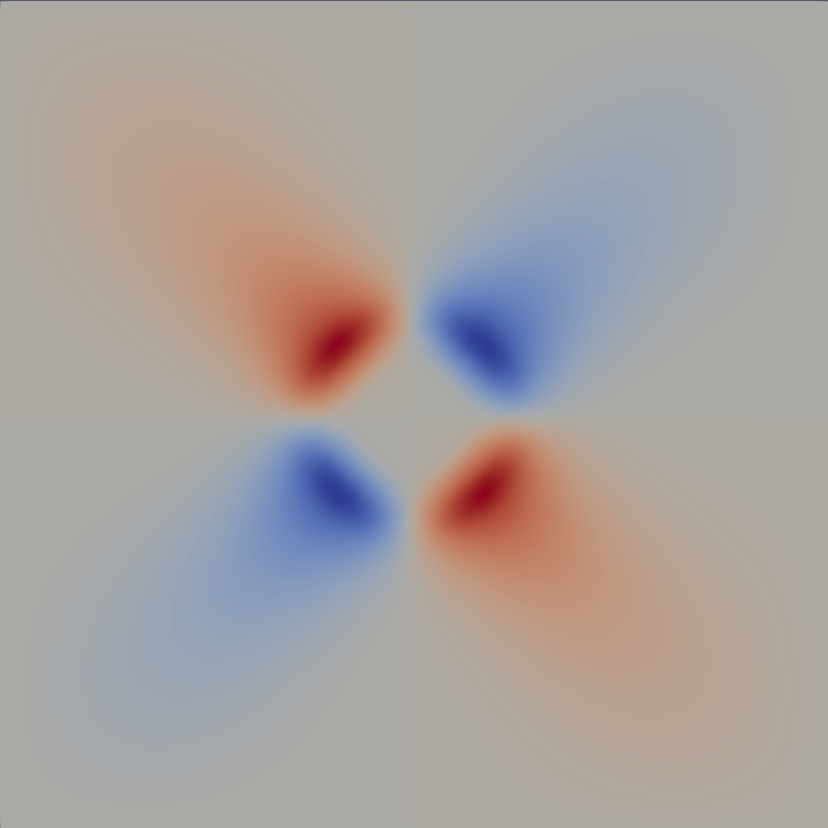
> [-VALID-] Postprocess.SolidMechanics.Stiffness.Grain1 = [(C11,C12,C12,0,0,0),(C12,C11,C12,0,0,0),(C12,C12,C11,0,0,0),(0,0,0,Gm,0,0),(0,0,0,0,Gm,0),(0,0,0,0,0,Gm)]

> [-VALID-] Postprocess.SolidMechanics.EigenStrain.Grain1 = (0.01,0.01,0,0,0,0)

# Postprocess.SolidMechanics.StiffnessEigenStrain.model = (model\_1, model\_2, ...)   0 - Normal, 1 - PhaseDependent\_MolarVolume , 2 - RegionDependent

> [DEFAULT] Postprocess.SolidMechanics.StiffnessEigenStrain.model = (0)

得如下结果



打开塑性求解开关

> [-VALID-] Postprocess.SolidMechanics.plasticity = TRUE

然后设置机械场-塑性求解参数

> [DEFAULT] Postprocess.SolidMechanics.Elastoplasticity.mapping\_steps = 1

> [-VALID-] Postprocess.SolidMechanics.Plasticity.max\_iteration\_steps = 1

# Postprocess.SolidMechanics.Plasticity.yield\_stress = ( phase\_0\_stress, ... )

#                                      .hardening\_modulus = (phase\_0\_modulus, ...)

#                                      .shear\_modulus = (phase\_0\_modulus, ...)

> [-VALID-] Postprocess.SolidMechanics.Plasticity.yield\_stress = (m\_yeild\_stress,i\_yeild\_stress)

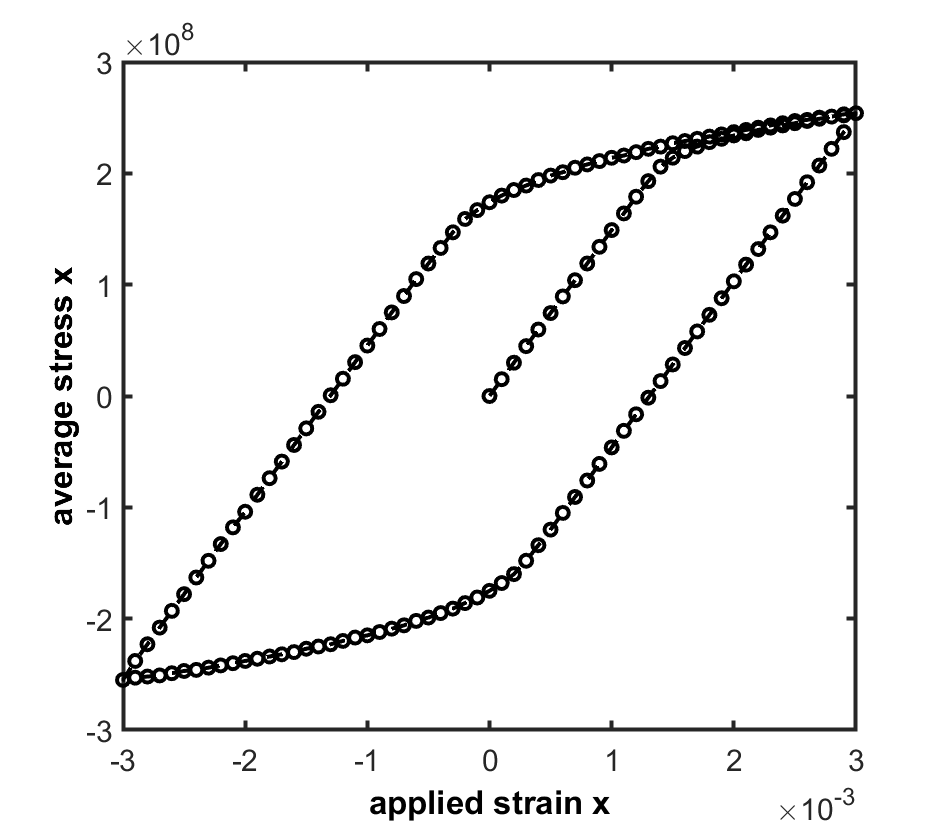
> [-VALID-] Postprocess.SolidMechanics.Plasticity.yield\_stress(0) = 2.75e+08

> [-VALID-] Postprocess.SolidMechanics.Plasticity.yield\_stress(1) = 2.75e+11

> [-VALID-] Postprocess.SolidMechanics.Plasticity.hardening\_modulus = (m\_hardening,i\_hardening)

> [-VALID-] Postprocess.SolidMechanics.Plasticity.shear\_modulus = (mG,iG)

在对材料体拉伸挤压时得到塑性求解得平均应力结果



也有对复杂颗粒得应力求解结果



案例详见：

0\_basic\_infile\4\_postprocess\1\_solid\_mechanics\_solvers\1\_elastic\_field\_implicit.mindes

0\_basic\_infile\4\_postprocess\1\_solid\_mechanics\_solvers\2\_elastic\_complex\_structure\_implicit.mindes

0\_basic\_infile\4\_postprocess\1\_solid\_mechanics\_solvers\3\_elastic\_plastic\_flow\_implicit.mindes

等

## 流体力学

打开流体力学开关：

# Postprocess.physical\_fields = (mechanic, fluid dynamic, electric)

> [-VALID-] Postprocess.physical\_fields = (false,true,false)

设置流体力学求解参数：

# Postprocess.FluidDynamics.solver = 0 - None , 1 - Pressure\_Correction , 2 - Lattice\_Boltzmann

> [-VALID-] Postprocess.FluidDynamics.solver = 2

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.max\_iterate\_steps = 10000

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.debug\_solver = TRUE

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.debug\_output\_step = 100

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.momentum\_accuracy = 1e-06

# Postprocess.FluidDynamics.LatticeBoltzmann.solid\_phases = (phase\_name, ... )

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.solid\_phases = ()

# tau = viscosity / fluid\_dt / Cs2 + 0.5

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.liquid\_viscosity = 0.1

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.liquid\_density = 1

# .LatticeBoltzmann.boundary\_condition = (down\_x,up\_x,down\_y,up\_y)

#                                        0 - Wall, 1 - Period, 2 - Free, 3 - Pressure, 4 - Normal\_Flow

#                            .pressure = p0 , density0 = p0 / Cs^2 , Cs = 1 / sqrt(3)

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.boundary\_condition = (1,1,0,0)

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC\_Down\_Y.wall\_roughness = 1

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC\_Down\_Y.wall\_speed = 0

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC\_Up\_Y.wall\_roughness = 1

> [-VALID-] Postprocess.FluidDynamics.LatticeBoltzmann.BC\_Up\_Y.wall\_speed = 0.1

# Postprocess.FluidDynamics.LatticeBoltzmann.source = ()

#             0 - Forces

> [DEFAULT] Postprocess.FluidDynamics.LatticeBoltzmann.source = ()

> [DEFAULT] Postprocess.FluidDynamics.LatticeBoltzmann.two\_phase\_flow = FALSE

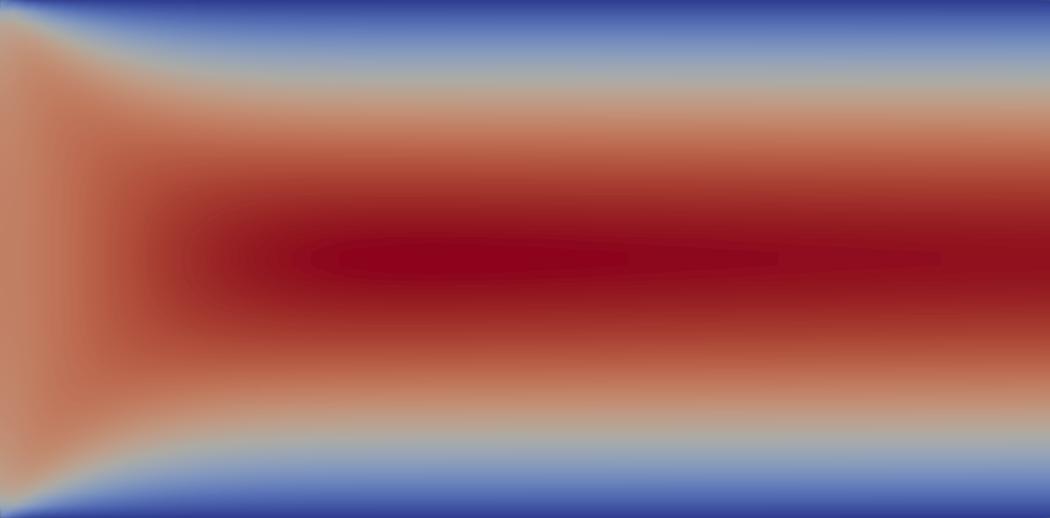
案例详见：

0\_basic\_infile\4\_postprocess\2\_fluid\_dynamics\_solvers

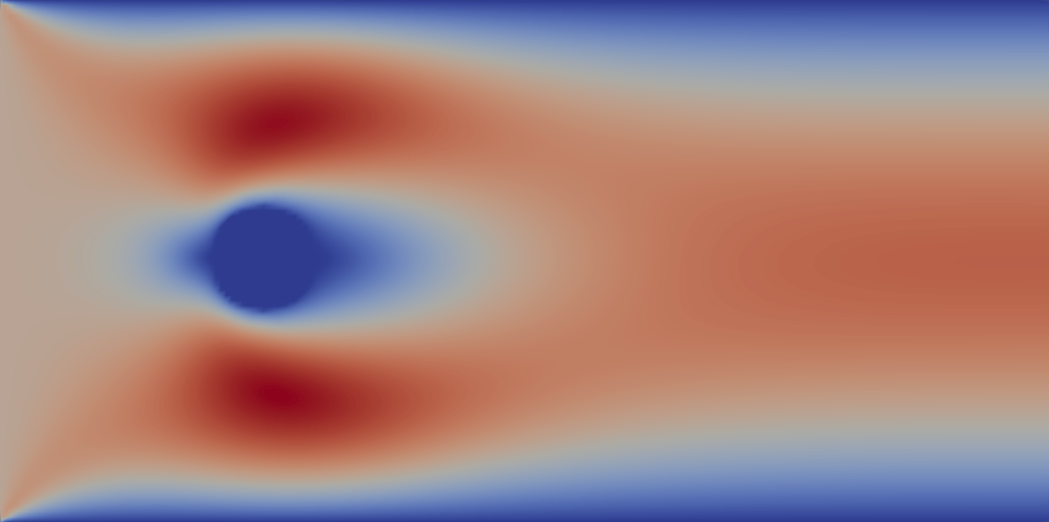
求解Couette流：



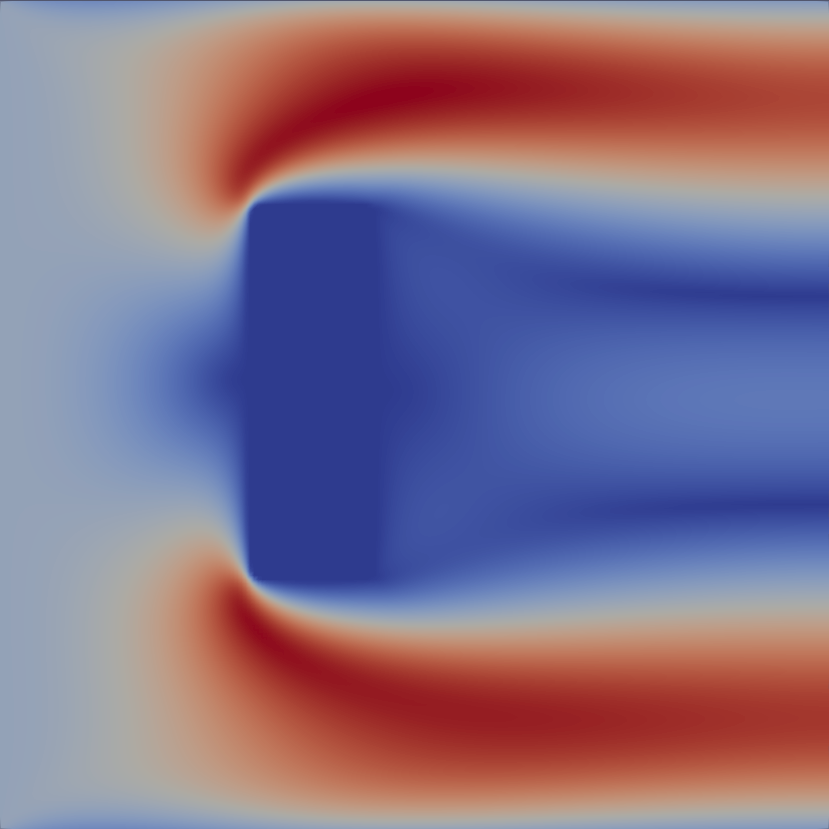
求解二维管道流：



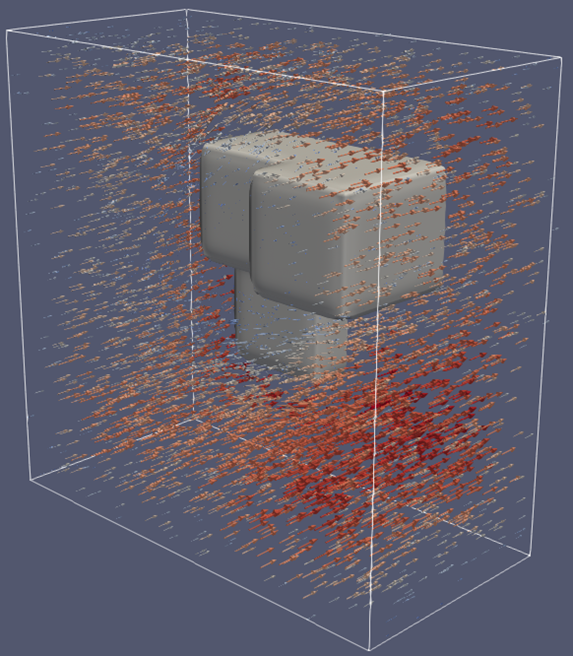
求解圆固体绕流（二维）：



求解方固体绕流（二维）：



求解方固体绕流（三维）：



叠加温度场，自然对流：