目 录

第 1 章 相场理论 ····· 1
1.1 资料 ····· 1
1.2 介绍 ····· 1
1.3 相场范式 ····· 1
1.3.1 总自由能泛函 ····· 2
1.3.2 局部自由能密度函数 ····· 3
1.3.3 梯度自由能密度函数 ····· 8
1.3.4 弹性能 ····· 9
1.4 演化方程和数值方法 ····· 10
1.5 应用 ····· 11
1.5.1 凝固建模 ····· 11
1.5.2 固态相变 ····· 11
1.5.3 晶粒长大 ····· 12
1.5.4 位错微观结构 ····· 13
1.6 相场方法在多尺度建模中的作用 ····· 13
第 2 章 MOOSE ····· 15
2.1 介绍 ····· 15
2.1.1 资料 ····· 15
2.2 Examples and Tutorials ····· 15
2.2.1 As Simple As It Gets ····· 15
2.2.2 Adding a Custom Kernel ····· 19
2.2.3 Multiphysics Coupling ····· 22
2.2.4 Custom Boundary Conditions ····· 23
2.2.5 Automatic Mesh Adaptivity ····· 29

2.2.6 Transient Analysis ····· 32
2.2.7 Custom Initial Conditions ····· 32
2.2.8 Material Properties ····· 33
2.2.9 Stateful Materials Properties ····· 33
2.2.10 Auxiliary Variables ····· 34
2.2.11 Preconditioning ····· 34
2.2.12 Physics Based Preconditioning ····· 34
2.2.13 Custom Functions ····· 35
2.2.14 Postprocessors and Code Verification ····· 38
2.2.15 Custom Actions ····· 41
2.2.16 Creating a Custom Timestepper ····· 43
2.2.17 Adding a Dirac Kernel ····· 44
2.2.18 ODE Coupling ····· 44
2.2.19 UserObjects ····· 44
2.2.20 Debugging ····· 46
2.3 MultiApp Demonstration ····· 47
2.3.1 介绍 ····· 47
2.3.2 MultiApps ····· 47
2.3.3 Transfers ····· 56
2.3.4 MultiApp Coupling ····· 71
2.4 Phase Field ····· 76
2.4.1 基础相场方程 ····· 76
2.4.2 ICs:EBSD ····· 77
2.5 枝晶生长 ····· 83
2.5.1 雪花状枝晶生长 ····· 83
2.6 晶粒长大理论 ····· 83
2.6.1 多序参数理论模型 ····· 83
2.7 晶粒生长建模 ····· 89

2.7.1 Kernels ····· 89
2.7.2 Materials ····· 90
2.7.3 UserObjects ····· 91
2.8 3D 晶粒生长 ····· 92
附录 A Matlba 脚本 ····· 93
A.1 曲线图绘制脚本 ····· 93
附录 B moose 代码 ····· 95
B.1 Input 文件 ····· 95
B.1.1 晶粒长大 ····· 95
参考文献 ····· 103

第 1 章　相场理论

1.1　资料

1.2　介绍

微观结构是材料的非常小尺度的结构，定义为在 25 倍放大倍数以上的光学显微镜下显示的材料制备表面的结构。在许多领域，包括生物学，流体力学，化学反应和相变，微观结构的演化都很普遍。材料的微观结构可以由不同组分和/或晶体结构的空间分布相，不同取向的晶粒，不同结构变体的域，不同电极化或磁极化的域以及结构缺陷组成。这些结构特征通常具有在纳米至微米范围内的介观长度尺度。材料（例如金属、聚合物、陶瓷或复合材料）的微观结构会强烈影响物理特性，例如强度、韧性、延展性、硬度、耐腐蚀性、高/低温行为或耐磨性[1]。

发生微观结构演化以减少总自由能，其中可能包括大量化学自由能，界面能，弹性应变能，电磁能，或处于施加的外部场（例如施加的应力场，电场，温度场和磁场）。由于微观结构演化的复杂性和非线性性质，经常采用数值模拟方法来进行研究。在对微观结构演变建模的常规方法中，将组分或结构域分开的区域被视为数学上明锐界面，称为锐界面模型。然后将局部界面速度确定为边界条件的一部分，或者从界面运动的驱动力和界面迁移率中计算得出，这通常需要显示地跟踪界面。尽管这种界面跟踪方法在一维问题中可能是成功的，但对于复杂的三维微观结构来说却不切实际。

相场法是一种在介观尺度上模拟界面动力学的成熟技术，已广泛用于模拟凝固，固态相变，晶粒长大等领域。在相场法中，微观结构用一系列相场变量来表示，这些场变量在整个求解域上是连续变化。对于保守场变量，如浓度场，它的时间和空间演化由 Cahn-Hilliard 方程控制；对于非保守场变量，如在多晶材料中表示晶粒的序参数场，控制方程是 Allen-Cahn 方程。

1.3　相场范式

在过去的三十年中，相场方法已成为研究多种类型材料的形态和微观结构演化过程最强大的模拟方法之一。它基于 vander Walls 于一个多世纪以前以

及 Cahn & Hilliard 在 60 年前开发的弥散界面模型。用一对连续场方程描述了微观结构的演化，即 Cahn-Hilliard 非线性扩散方程(1.1) 和 Allen-Cahn（time-dependentGinzburg-Landau）方程(1.2)。

$$\frac{\partial c_i(r,t)}{\partial t} = \nabla M_{ij} \nabla \frac{\delta F}{\delta c_j(r,t)} \tag{1.1}$$

$$\frac{\partial \eta_p(r,t)}{\partial t} = -L_{pq} \nabla \frac{\delta F}{\delta \eta_q(r,t)} \tag{1.2}$$

其中 M_{ij} 和 L_{ij} 与原子或界面迁移率有关，$\{c_1, c_2, ..., c_N\}$ 是保守场变量，而 $\{\eta_1, \eta_2, ..., \eta_N\}$ 是非保守场变量。

相场模型的应用基本上可以分成两种类型。在第一种类型中，仅出于避免跟踪界面的目的而引入了相场变量（也称为相场）。基本上所有应用于凝固的相场模型都属于这种类型。实际上，术语“相场模型”是在对纯熔体的凝固进行建模时首次引入的。然后，通过渐近分析，选择相场模型中的热力学和动力学系数，以匹配常规锐界面方程中的相应参数。Karma[23] 和 Ode 等人最近综述了凝固相场模型的开发 [24]。

在第二种类型中，相场变量对应于定义明确的物理序参数，例如用于有序-无序转变的长程序参数场和用于相分离的组分场。这些类型的模型假定给定过程中的微观结构演化由相场方程控制，并且原则上所有热力学和动力学系数都可以与微观参数相关。它们已被广泛应用于模拟固态相变 [14,15]。

除了应用于凝固和固态相变外，还为许多其他重要的材料工艺提出了相场模型，包括晶粒生长和粗化 [14,25-27]，薄膜的微观结构演变 [29,30]，表面应力-诱导图案的形成 [30]，裂纹扩展 [31,32]，在应变存在下的晶体生长 [33、34]，位错-溶质相互作用 [35、36]，位错动力学 [37]，电迁移，以及多组分互扩散。

1.3.1 总自由能泛函

相场变量有两种类型，保守场变量和非保守场变量。在扩散界面描述中，由一组守恒 $\{c_1, c_2, \ldots, c_N\}$ 和非守恒 $\{\eta_1, \eta_2, \ldots, \eta_M\}$ 场变量描述的非均匀微结构系统的总自由能泛函如下：

$$F = \int_V \left[f_{loc}\left(c_1, c_2, \ldots, c_N;\ \eta_1, \eta_2, \ldots, \eta_M\right) + f_{gr}\left(c_1, c_2, \ldots, c_N;\ \eta_1, \eta_2, \ldots, \eta_M\right) + E_d \right] dV \tag{1.3}$$

其中，f_{loc} 是局部自由能密度，是场变量 c_i 和 η_j 的函数，并因模型而异。f_{gr} 是梯度自由能密度，

$$f_{gr} = \sum_{i}^{N} \frac{\kappa_i}{2} \left|\nabla c_i\right|^2 + \sum_{j}^{M} \frac{\kappa_j}{2} \left|\nabla \eta_j\right|^2 \tag{1.4}$$

其中 κ_i 和 κ_j 是梯度能量系数。最后，E_d 描述了系统中任何额外的能量源，例如形变能或静电能量。体积分中前两项表示短程化学相互作用对自由能的局部贡献，界面能仅仅来自于界面处和界面周围非零的梯度能量项。第三个项表示一个非局部项，它包含来自任何一种或多种远程相互作用（例如弹性相互作用，电偶极-偶极相互作用，静电相互作用等）的总自由能的贡献，而这些也依赖于相场变量。不同相场模型的主要区别在于自由能密度各个部分的不同。

1.3.2　局部自由能密度函数

局部自由能密度函数是相场模型的关键组成之一。许多相场模型，特别是在凝固建模中，对函数使用双阱形式，即

$$f(\eta) = 4\Delta f \left(-\frac{1}{2}\eta^2 + \frac{1}{4}\eta^4\right) \tag{1.5}$$

自由能函数具有由 $\eta = -1$ 和 $\eta = +1$ 表示的双简并最小值，它的曲线图如图 1.1所示。Δf 是具有最小自由能的两个状态之间的能垒。如果 η 表示组分场，两个最小值表示具有不同组分的两个平衡相，则在同构分解过程中，Δf 是将单个均相 $\eta = 0$ 转变为由 $\eta = -1$ 和 $\eta = +1$ 表示两相的非均相混合物的驱动力。如果 η 是长程序参数场，则 $\eta = -1$ 且 $\eta = +1$ 描述两个热力学退化的反相域状态。

对于某些过程，可能更希望自由能的两个最小值位于 $\eta = 0$ 和 $\eta = 1$ ，那么可以使用函数 $f(\eta) = 4\Delta f \eta^2(1-\eta)^2$, 其曲线图如图 1.2所示。

有时候在相场模型中使用另一种自由能函数，称为双障碍势，

$$f(\eta) = 4\Delta f \left(1 - \eta^2\right) + I(\eta) \tag{1.6}$$

其中，

$$I(\eta) = \begin{cases} \infty & \text{if} \quad |\eta| > 1 \\ 0 & \text{if} \quad |\eta| \leq 1 \end{cases} \tag{1.7}$$

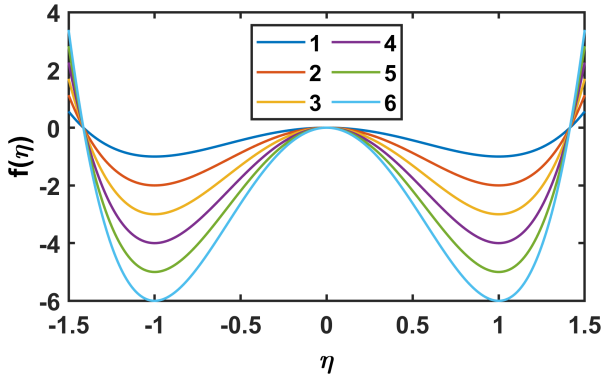

图 1.1 双井势函数 1

Figure 1.1 Double-well potential function 1

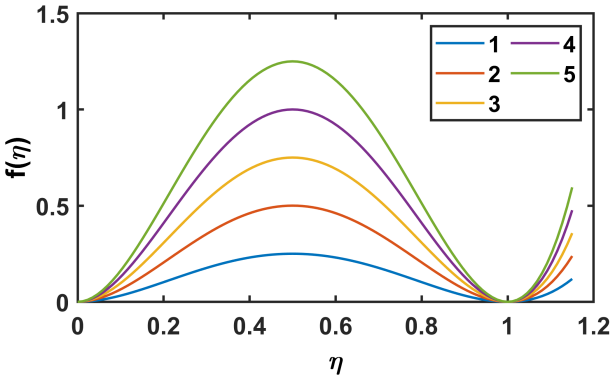

图 1.2 双井势函数 2

Figure 1.2 Double well potential function 2

其曲线图如图 1.3所示。该势具有计算优势，即场变量在界面区域外假定值为 −1 和 +1，而在双阱势方程 (1.5)的情况下，场变量的值缓慢变为 −1 和 +1 远离界面。双障碍势首先由 Oono 和 Puri 在淬火系统的有序化或相分离过程中模式演化的细胞动力学建模中引入，后来它被 Blowey&Elliot 用于调幅分解的建模。最近，这种势函数已被应用于模拟金属互连中空隙的形态演变。

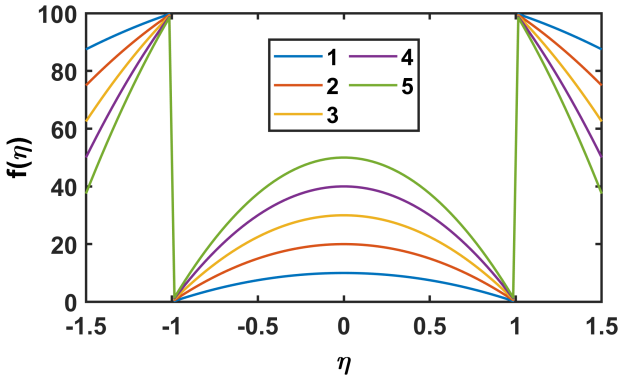

图 1.3 双障碍势函数

Figure 1.3 Double barrier potential

在他们的位错动力学相场模型中，Wang 等人引入了以下局部自由能函数，即所谓的结晶能，

$$f(\eta) = \Delta f \, \sin^2(\pi\eta) \tag{1.8}$$

其曲线图如图 1.4所示。它有无数个位于 $\eta = -\infty, \ldots, -1, 0, +1, \ldots, \infty$ 最小值，Δf 是最小值之间的能垒。在这种情况下，势函数最小值处的相场绝对值表示滑移面上方和下方两个晶格平面之间的不连续相对位移，以滑移系上给定的 Burgers 矢量为单位测量，相场的正负号描述了 Burgers 矢量的相反方向。

相场模型在实际材料过程中的许多应用中，通常需要引入多个场变量或将一种类型的场与另一种类型的场耦合。即使在纯液体凝固的相场模型中，相场也与温度场耦合，其中已经使用的势能之一是 [35],

$$f(\eta, T) = 4\Delta f \left(-\frac{1}{2}\eta^2 + \frac{1}{4}\eta^4\right) + \frac{15\alpha}{8}\left(\eta - \frac{2}{3}\eta^3 + \frac{1}{5}\eta^5\right)(T - T_m) \tag{1.9}$$

其中 α 是大于零的常数，T_m 是平衡熔化温度，其曲线图如图 1.5所示。由于方程 (1.9)中第二相对 η 的特定依赖性，两个平衡项处 η 的值于过冷度 $T - T_m$ 无关。

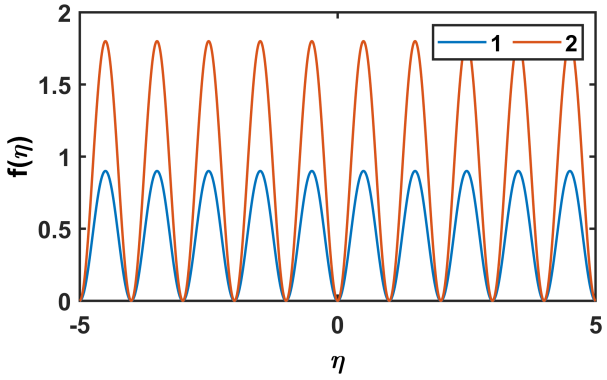

图 1.4 结晶势函数

Figure 1.4 crystalline energy potential

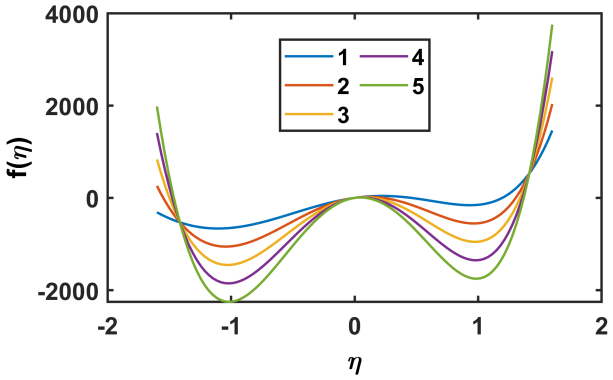

图 1.5 耦合势函数 1

Figure 1.5 Coupling potential function 1

另一个涉及场变量之间耦合的例子是晶粒生长的相场模型，其中场变量描述了具有不同取向的晶粒的空间分布。这涉及将双阱势（方程 (1.5)）简单地扩展为具有无限数量的最小值 [16],

$$f\left(\eta_1,\eta_2,\ldots\right)=4\Delta f\left(-\frac{1}{2}\sum_i\eta_i^2+\frac{1}{4}\sum_i\eta_i^4\right)+\alpha\sum_i\sum_{j>i}\eta_i^2\eta_j^2 \tag{1.10}$$

其中 α 为大于零的系数，Δf 为最小状态间的能垒, 其曲线图如图 1.6所示。当 $\alpha>2\Delta f$ 时，无穷多个最小值位于 $(1,0,\ldots),(0,1,...),(-1,0,...)$ 等处，代表多晶中晶粒可能的取向。

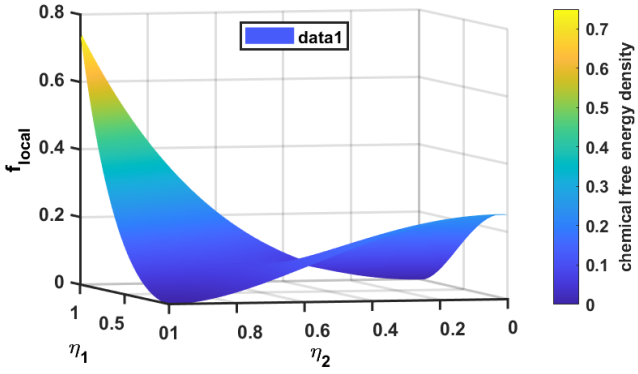

图 1.6 耦合势函数 2

Figure 1.6 Coupling potential function 2

对于许多固态相变，场变量对应于明确定义的物理序参数。在这种情况下，局部自由能函数通常表示为序参数多项式，使用传统的朗道式展开。要求多项式中的所有项对于高温相的对称操作是不变的。例如，对于从二元合金的面心立方 (FCC) 基体中析出有序相 ($L1_2$)，扩散项高达四阶，局部自由能函数由为，

$$\begin{aligned}f\left(c,\eta_1,\eta_2,\eta_3\right)=f_d(c,T)+\tfrac{1}{2}A_2(c,T)\left(\eta_1^2+\eta_2^2+\eta_3^2\right)+\tfrac{1}{3}A_3(c,T)\eta_1\eta_2\eta_3\\+\tfrac{1}{4}A_{41}(c,T)\left(\eta_1^4+\eta_2^4+\eta_3^4\right)+\tfrac{1}{4}A_{42}(c,T)\left(\eta_1^2\eta_2^2+\eta_2^2\eta_3^2+\eta_1^2\eta_3^2\right)\end{aligned} \tag{1.11}$$

其中 $f_d(c,T)$ 是无序相的自由能，A_2、A_3、A_{41} 和 A_{42} 是扩散系数，它们是温度和成分的函数。自由能函数（方程 (1.11)）在阶次参数方面具有四重简并最小值。如果 $A_3(c,T)<0$，则自由能最小值位于

$$\left(\eta_0,\eta_0,\eta_0\right),\left(\eta_0,-\eta_0,-\eta_0\right),\left(-\eta_0,\eta_0,-\eta_0\right),\left(-\eta_0,-\eta_0,\eta_0\right) \tag{1.12}$$

其中 η_0 是给定成分和温度下的平衡长程序参数。方程 (1.12)中给出的四组长程序参数描述了与基体无序 FCC 相的原始晶格平移相关的 $L1_2$ 有序相的四个能量等效反相域。可以通过将方程 (1.12)中的序参数值代入自由能函数（方程 (1.11)）中来获得作为组成函数的有序相的自由能。

1.3.3 梯度自由能密度函数

微观结构的固有特性是界面的存在。由界面处的成分或结构不均匀性引起的额外自由能称为界面能。为了将界面能与相场模型中的梯度能量项联系起来，让我们考虑一个由单个场变量 η 描述的简单系统。这种系统的微观结构的总自由能 F 简化为，

$$F = F_{\text{bulk}} + F_{\text{int}} = \int_v \left[f(\eta) + \frac{1}{2}\kappa_\eta (\nabla\eta)^2 \right] dV \tag{1.13}$$

其中 F_{bulk} 和 F_{int} 分别是体能和界面能，κ_η 是梯度能量系数。当 eta 是组分或长程序参数场时，梯度能量系数可以表示为成对原子间相互作用能 [3,46]。使用双阱自由能函数（方程 (1.5)），很容易得出比界面能（单位面积的界面能）σ_{gr} 为，

$$\sigma_{gr} = \frac{4\sqrt{2}}{3}\sqrt{\kappa_\phi \Delta f} \tag{1.14}$$

方程 (1.14)如何计算得来的没搞懂!!! 对于具有多个场变量的系统，界面能通常必须以数值方式评估。

由于固体的晶体学性质，界面能通常是各向异性的。界面能各向异性的程度对颗粒的生长形态和平衡形状具有显着影响。在相场模型中已经提出了许多方法来引入界面能各向异性。因为 $\sigma_{gr} \propto \sqrt{\kappa_\eta}$，引入各向异性的最简单方法之一是使 σ_{gr} 和 $\sqrt{\kappa_\eta}$ 具有相同的方向 $(\Delta\eta/|\Delta\eta|)$ 相关性。例如，在凝固建模中，可以通过梯度能量系数 [47,48] 的方向依赖性引入界面能的三次各向异性，

$$\sqrt{\kappa(\boldsymbol{n})} = a\left[1 + b\left(n_x^4 + n_y^4 + n_x^4\right)\right] \tag{1.15}$$

这里 n_x, n_y, n_z 是单位方向向量 n 的 x、y 和 z 分量。需要指出的是，虽然这种方法可以方便地利用梯度能量系数的取向相关性来引入界面能各向异性，但很难在物理上证明其合理性。

在固态相变中，界面能各向异性可以通过梯度项自然地引入。例如，对于从 FCC 基体中析出有序 $L1_2$ 相，梯度能量项可以写为 [38,39],

$$\frac{1}{2}\kappa_c[\nabla c(\mathbf{r})]^2 + \frac{1}{2}\sum_{p=1}^{3}\kappa_{ij}^{\eta}(p)\nabla_i\eta_p(\mathbf{r})\nabla_j\eta_p(\mathbf{r}) \tag{1.16}$$

最后，为了引入所需的界面能各向异性，还可以添加高阶梯度能量项，尽管它们也很难在物理上证明是合理的。例如，有必要引入四阶项以产生立方各向异性，

$$\kappa_{ijkl}\left(\frac{\partial^2\phi}{\partial r_i\partial r_j}\right)\left(\frac{\partial^2\phi}{\partial r_k\partial r_l}\right) \tag{1.17}$$

其中梯度系数 κ_{ijkl} 是一个四阶张量，r_i 是位置向量的第 i 个分量。结果表明，该高阶梯度能量项不显示尖峰的界面能，因此平衡粒子形状不包含平面小面。然而，由此产生的界面能各向异性可能足以在 Wulff 形状中产生拐角。

1.3.4　弹性能

固体中的相变通常在其早期阶段产生相干微观结构。在相干微观结构中，晶格面和方向在界面上是连续的，并且相和域之间的晶格失配通过弹性位移来适应。相场模型中，弹性能对总自由能的贡献可以通过将弹性应变能表示为场变量的函数或通过包括场变量与局部自由位移梯度之间的耦合项来直接引入。最近讨论了这两种方法之间的关系 [53]。

考虑由保守组分场 $c(r)$ 和非保守序参数场 $\eta(r)$ 描述的广义微观结构。假设局部无应力的应变与成分场呈线性比例，并且与序参数场呈二次关系，即，

$$\varepsilon_{ij}^{o}(r) = \varepsilon_{ij}^{c}c(r) + \varepsilon_{ij}^{\eta}\eta^2(r) \tag{1.18}$$

应该强调的是，无应力的应变对成分的线性相关性只是为了方便而假设的。可以使用相同的方法来获得弹性能量，其中晶格参数与组分呈非线性关系。无应力的应变（方程 (1.18)）包含两条信息：一个是由场变量 $c(r)$ 和 $\eta(r)$ 描述的微观结构，另一个是微观结构中相或域之间的晶体学关系通过关于成分和序参数的晶格膨胀系数给出，即 ϵ_{ij}^{c} 和 ϵ_{ij}^{η} 。相干微观结构中的局部弹性应力由下式给出，

$$\sigma_{ij}(r) = \lambda_{ijkl}(r)\varepsilon_{kl}^{el}(r) = \lambda_{ijkl}(r)\left[\varepsilon_{kl}(r) - \varepsilon_{kl}^{c}c(r) - \varepsilon_{kl}^{\eta}\eta^2(r)\right] \tag{1.19}$$

其中 $\lambda_{ijkl}(r)$ 是弹性张量，通常是空间相关的或不均匀的，$\epsilon_{kl}(r)$ 是总局部应变。在适当的边界条件下，它是通过求解以下机械平衡方程获得的。

$$\frac{\partial \sigma_{ij}}{\partial r_j} = 0 \tag{1.20}$$

使用计算所得弹性形变 $\epsilon_{kl}(r)$，微观结构的总弹性能量可以通过一下表达式计算得出，

$$E = \frac{1}{2}\int_V \lambda_{ijkl}(r)\varepsilon_{ij}^{el}\varepsilon_{kl}^{el}dV \tag{1.21}$$

在存在外部应力的情况下，机械能对总自由能的贡献变为，

$$E = \frac{1}{2}\int_V \lambda_{ijkl}(r)\varepsilon_{ij}^{el}\varepsilon_{kl}^{el}dV - \int_V \sigma_{ij}^{a}(r)\varepsilon_{ij}(r)dV \tag{1.22}$$

其中 σ_{ij}^{a} 是施加的应力，可能是不均匀的。为了求解力学平衡方程（方程 (1.20)），大多数现有的相场模拟假设均匀弹性，其中忽略不同相之间的弹性模量差异，即 $\lambda_{ijkl}(r) = Const$ [14,15]。在这种情况下，可以对任何任意微观结构的弹性场进行分析评估，如 Khachaturyan 的研究所示 [46,54]。通过弹性模量张量对场变量的依赖性，在结合弹性不均匀性方面做出了重大努力。当弹性不均匀性很小时，可以使用一阶近似 [52,55,56]。最近，研究表明可以通过迭代方法结合高阶近似值来求解非均匀弹性方程，从而可以引入高弹性非均匀性 [57]。这种迭代方法的独特之处在于，与齐次近似相比，可以通过使用越来越多的高阶近似来连续提高精度，而不会显着增加计算时间。最后，人们可以使用直接数值方法来求解弹性方程，例如使用共轭梯度法 (CGM)[58,59] 或有限元法 [31]。然而，这种直接数值求解通常比使用更近似方法的求解更耗时。

1.4 演化方程和数值方法

有了上面讨论的微观结构的总自由能，相场模型中场变量的演变可以通过求解以下 Cahn-Hilliard 和 Allen-Cahn 方程获得。

使用相场方法对微观结构演化建模然后简化为寻找动力学方程 (1.1)和方程 (1.2)的解。大多数相场模拟在均匀空间网格和显式时间步长上采用简单的二阶有限差分方法。众所周知，在这种显式方案中，时间步长必须很小以保持数值解稳定。

对于周期性边界条件，经常采用的技术之一是快速傅立叶变换方法，它将积分微分方程转换为代数方程 [参见例如 (61–63)]。或者，可以先将积分微分方程转换为有限差分方程，然后将其转换为傅立叶空间。但是在这种情况下，空间离散化的精度只是二阶而不是光谱精度。并且，由于谱方法通常对空间变量使用统一网格，因此可能难以解析具有中等数量网格点的极其尖锐的界面。在这种情况下，自适应频谱方法可能更合适。

已经开发了实空间自适应网格算法来求解应用于凝固的相场方程 (68,69)。已经表明，与使用均匀网格的方法相比，自适应方法中的变量数量显着减少。这允许人们在更大的系统和更长的模拟时间中求解场模型。然而，这种自适应方法通常比统一网格实现起来要复杂得多。

1.5　应用

现有的相场应用主要集中在三种主要的材料过程：凝固、固态相变以及晶粒长大和粗化。最近，已经开发了许多新的相场模型，用于对薄膜和表面、位错动力学、裂纹扩展和电迁移进行建模。

1.5.1　凝固建模

凝固是当液体冷却至低于平衡熔化温度 (T_m) 时，液体转变为结晶固体。它的基本理解对许多实际过程来说非常重要，例如铸造、晶体生长和焊接。使用相场方法模拟凝固过程的想法早在大约 40 年前就引入了。其目的是希望在凝固过程中预测复杂的树枝状图案形成，而无需明确跟踪固液界面。Kobayashi[47] 首次证明了它的成功，他使用单组分熔体等温凝固的相场模型模拟了逼真的 3D 枝晶。从那时起，大量的研究工作一直致力于开发凝固的相场模型，以便更详细地总结凝固的相场建模。例如，已经开发出的相场模型可以通过包含热噪声自动生成枝晶的侧分支 (71)。已经为二元合金系统提出了许多相场模型，并已用于凝固微观结构建模（85-96）。

……

1.5.2　固态相变

相场模型已被开发用于各种扩散和无扩散固态相变 [简要回顾，见 (14,15)]。例子包括同构相分离 (52,55,56,61,120)，有序金属间相从无序基质中析出 (39,40,121–

124)，立方到四方的转变 (62,125–129)，六方到斜方晶的转变（130-133)，铁电转变（20,42-44,60,134)，单晶和多晶中适当和不适当的马氏体转变（135-137)，以及施加应力下的相变（40、136、138-140)。固态相变的相场建模背后的主要思想是假设系统的自由能可以表示为使用朗道型展开式的物理定义序参数的函数（参见方程 8 和 10）例子)，这些序参数的演化遵循 Allen-Cahn 和 Cahn-Hilliard 方程。事实上，对于扩散有序-无序和组分相分离过程，可以从相应的微观方程（39,141）推导出相场方程。固态相变的一个独特特征是，从转变的早期阶段产生的基本上所有微观结构都是相干的微观结构。相干微结构的形成涉及弹性应变能的产生，其大小取决于晶格失配程度、弹性特性以及相干粒子或域的形状和空间分布 (46)。虽然整体化学自由能仅取决于每个相的体积分数，但弹性能是共存相的体积分数和形态的函数。因此，弹性能通常在相干系统的微观结构演化中起主导作用。

1.5.3 晶粒长大

凝固和固-固相变都是由体积自由能的降低驱动的，而发生粗化和晶粒生长以降低微观结构的总界面自由能。已经提出了几种相场模型来描述晶粒生长。第一个模型是由 Chen 等 [2] 提出的，其中不同晶体取向的晶粒由一组非保守序参数场表示。阶参数场的演变由 Allen-Cahn 方程描述。该模型已广泛应用于模拟二维 [3–5] 和三维系统 [6] 中的晶粒生长动力学。Steinbach 提出了一种类似的多阶参数模型，即所谓的，用于晶粒生长。两种模型的主要区别在于多相场模型对序参数施加了以下约束，$\sum_i \eta_i = 1$，即给定点的所有序参数之和为 1.0。该约束的物理解释是序参数代表不同取向晶粒的体积分数。最近，提出了一个有趣的相场模型来研究晶粒 [7–9]。它不同于晶粒长大的多阶参数模型，它使用两个序参数来描述晶粒结构；一代表晶粒度；另一个反映了晶体的主要局部取向。虽然局部取向顺序参数的松弛模拟了多阶参数模型中不存在的晶粒旋转，但该顺序参数在无序液态中是未定义的。已经通过修改多阶参数模型研究了晶界能量和迁移率各向异性对晶粒生长的影响。除了单相材料中的晶粒生长外，还开发了相场模型来研究扩散诱导的晶界迁移、溶质阻力影响下的晶粒生长动力学、粗化动力学连续基体中分散相的高体积分数，以及两相系统中耦合晶粒生长和 Ostwald 熟化的动力学。

1.5.4　位错微观结构

……

1.6　相场方法在多尺度建模中的作用

众所周知，许多材料结构演化过程发生在很宽的长度和时间范围内。尽管最近提出了创新的多尺度建模方法来对许多复杂的多组分材料系统中的多尺度过程进行建模，但在可预见的未来，从一个尺度传递到另一个尺度的直接信息仍然是主要方法。在这里，相场方法可以作为原子级基本计算和宏观本构建模之间的桥梁。例如，第一性原理和相场相结合的方法已被应用于解决铝合金中 θ0 (Al2Cu) 析出的问题 (187, 188)。θ_0 析出物不仅出现在二元 Al-Cu 合金中，而且也是各种工业铝合金中的强化析出物。所有必要的热力学信息，包括基体相和沉淀相的体积自由能、界面能及其各向异性以及晶格失配，都是通过结合簇扩展的第一性原理计算获得的。因此，可以仅使用从第一性原理获得的热力学信息，使用相场模型来预测析出物的形状。从这些相场模拟中可以获得更多定量的微观结构信息，这有助于理解沉淀硬化行为。原子计算也被用来获得固液界面的基本属性，例如界面能量和迁移率，以及它们的各向异性 (189, 190)，作为相场模拟的输入 (72)。

对于复杂的多组分系统，已经报道了将相场模型与现有的热力学和动力学数据库连接起来的许多努力 (103, 106, 107)。可以使用 CALPHAD 方法（191）从现有数据库直接构建相场模型的自由能函数。还可以纳入来自数据库的原子迁移率的组分依赖性。然而，为了考虑固态过程中弹性能的影响，必须构建额外的数据库，例如晶格参数和弹性常数。通过独立的可靠数据库，可以使用相场法预测复杂多组分合金的微观结构演变。

第 2 章　MOOSE

2.1　介绍

2.1.1　资料

1. Youtube 教程：MOOSE finite element framework tutorial part-1 | Introduction and Installation
2. Moose 官网安装教程：Install MOOSE
3. MOOSE site:https://inl.gov/ filetype:pdf

2.2　Examples and Tutorials

Examples and Tutorials: 包括各种演示，旨在介绍多物理场面向对象仿真环境 (MOOSE) 的基础知识，用于创建自定义应用程序以解决独特且具有挑战性的多物理场问题。每个示案例都侧重于 MOOSE 的不同方面，主要是可用于解决多物理场问题的基本系统。

2.2.1　As Simple As It Gets

此示例简要描述了基本输入文件的创建以及使用 MOOSE 解决问题所需的六个部分。包括：Mesh，见节 **??**；Variables；Kernels；BCs；BCs；Executioner；Executioner；Outputs。

E:/GitHub/moose/examples/ex01_inputfile/ex01.i

```
[Mesh]
  # We use a pre-generated mesh file (in exodus format).
  # This mesh file has 'top' and 'bottom' named boundaries defined inside it.
  file = mug.e
[]

[Variables]
  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  [./diff]
```

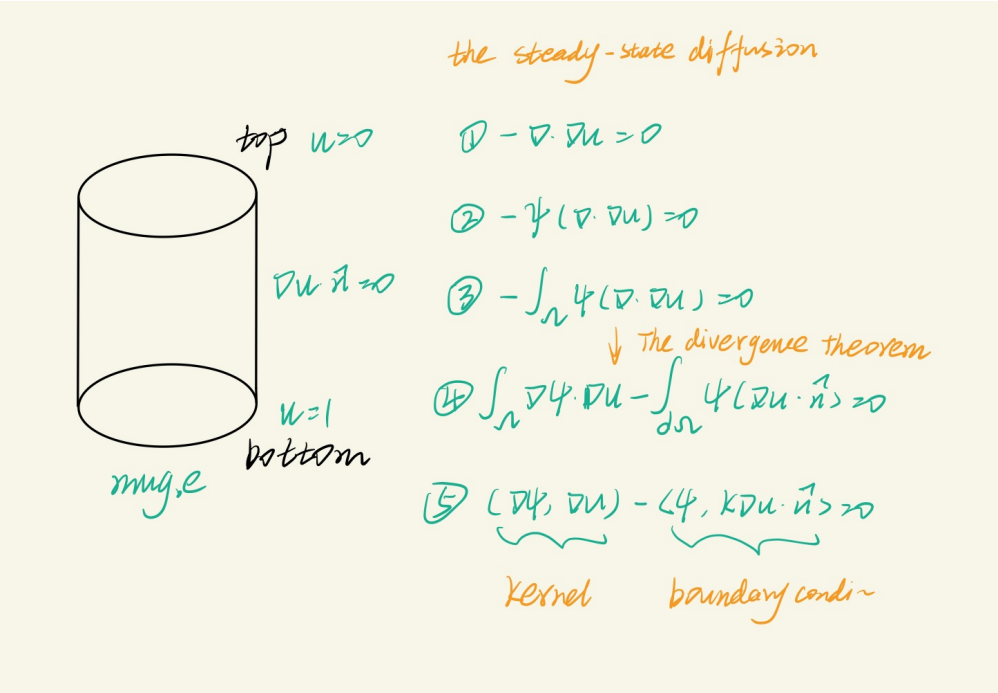

图 2.1 其中 u 表示变量，ψ 表示试函数

Figure 2.1 Among them, u means variable, and ψ means trial function.

```
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  [./bottom] # arbitrary user-chosen name
    type = DirichletBC
    variable = diffused
    boundary = 'bottom' # This must match a named boundary in the mesh file
    value = 1
  [../]

  [./top] # arbitrary user-chosen name
    type = DirichletBC
    variable = diffused
    boundary = 'top' # This must match a named boundary in the mesh file
    value = 0
  [../]
[]

[Executioner]
  type = Steady
  solve_type = 'PJFNK'
  file_base = 'out'
```

```
[]

[Outputs]
  execute_on = 'timestep_end'
  exodus = true
[]
```

2.2.1.1　Kernel: Diffusion

Recall the steady-state diffusion equation on the 3D domain Ω:

$$-\nabla \cdot \nabla u = 0 \in \Omega$$

The weak form of this equation includes a volume integral, which in inner-product notation, is given by:

$$\left(\nabla \psi_i, \nabla u_h\right) = 0 \quad \forall \psi_i$$

where ψ_i are the test functions and u_h is the finite element solution.

其头文件和源代码如下：

E:/GitHub/moose/framework/include/kernels/Diffusion.h

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#pragma once

#include "Kernel.h"

class Diffusion;

template <>
InputParameters validParams<Diffusion>();

/**
 * This kernel implements the Laplacian operator:
 * $\nabla u \cdot \nabla \phi_i$
 */
class Diffusion : public Kernel
```

```
{
public:
  static InputParameters validParams();

  Diffusion(const InputParameters & parameters);

protected:
  virtual Real computeQpResidual() override;

  virtual Real computeQpJacobian() override;
};
```

E:/GitHub/moose/framework/src/kernels/Diffusion.C

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#include "Diffusion.h"

registerMooseObject("MooseApp", Diffusion);

defineLegacyParams(Diffusion);

InputParameters
Diffusion::validParams()
{
  InputParameters params = Kernel::validParams();
  params.addClassDescription("The Laplacian operator ($-\\nabla \\cdot \\nabla u$), with the weak "
                             "form of $(\\nabla \\phi_i, \\nabla u_h)$.");
  return params;
}

Diffusion::Diffusion(const InputParameters & parameters) : Kernel(parameters) {}

Real
Diffusion::computeQpResidual()
{
  return _grad_u[_qp] * _grad_test[_i][_qp];
}

Real
```

```
Diffusion::computeQpJacobian()
{
  return _grad_phi[_j][_qp] * _grad_test[_i][_qp];
}
```

2.2.2　Adding a Custom Kernel

本示例建立在示例 1 的基础上，并介绍了如何创建自定义内核，这是将弱形式添加到 MOOSE 的机制。

问题声明：对于稳态对流传热方程：

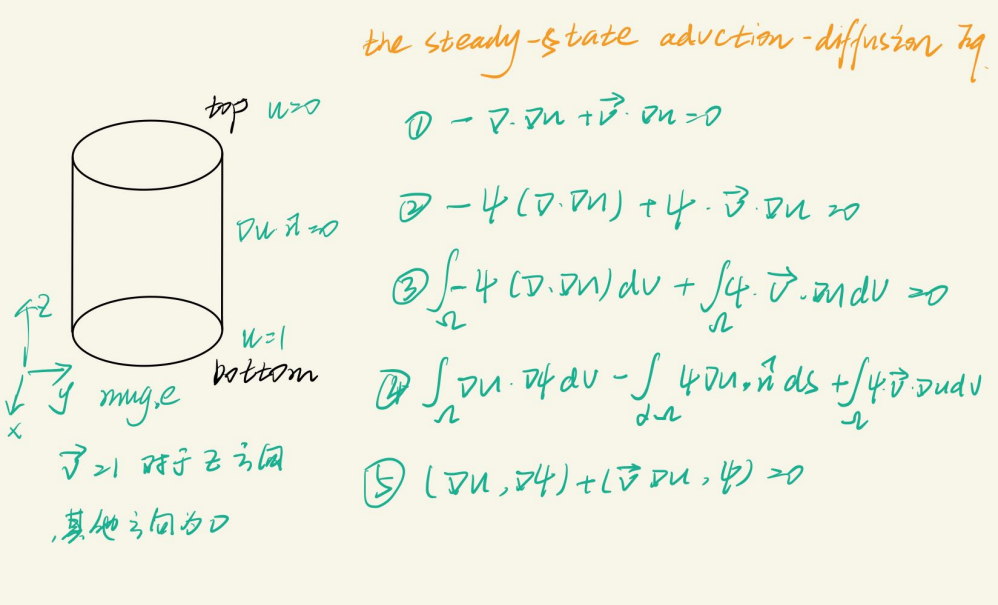

图 2.2 其中 u 表示有限元的解，ψ 表示试函数

Figure 2.2 Among them, u finite element solution, and ψ means trial function.

通过继承 MOOSE 中已经存在的项目来创建一个 C++ 对象，以此来创建问题中对流部分，其源代码如下：

E:/GitHub/moose/examples/ex02_Kernel/ex02.i

```
[Kernels]
  [diff]
    type = Diffusion  # 内置的扩散kernel， $(\nabla u,\nabla \psi)$
    variable = convected
  []

  [conv]
    type = ExampleConvection # 自定义的对流kernel， $V.grad(u), test$
    variable = convected
```

```
    velocity = '0.0 0.0 1.0'
  []
[]
```

E:/GitHub/moose/examples/ex02_kernel/include/kernels/ExampleConvection.h

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#pragma once

#include "Kernel.h"

/**
 * The forward declaration is so that we can declare the validParams() function
 * before we actually define the class... that way the definition isn't lost
 * at the bottom of the file.
 */

// Forward Declarations, 向前声明
class ExampleConvection;

/**
 * validParams returns the parameters that this Kernel accepts / needs
 * The actual body of the function MUST be in the .C file.
 */
template <>
InputParameters validParams<ExampleConvection>();

/**
 * Define the Kernel for a convection operator that looks like:
 *
 * (V . grad(u), test)
 *
 * where V is a given constant velocity field.
 */
class ExampleConvection : public Kernel
{
public:
  /**
   * This is the constructor declaration. This class takes a string and
   * a InputParameters object, just like other Kernel-derived classes.
```

```
43    * 构造函数声明，用于初始化。
      */
45   ExampleConvection(const InputParameters & parameters);

47 protected:
     /**
49    * Responsible for computing the residual at one quadrature point.
      * This function should always be defined in the .C file.
51    */
     virtual Real computeQpResidual() override;
53
     /**
55    * Responsible for computing the diagonal block of the preconditioning matrix.
      * This is essentially the partial derivative of the residual with respect to
57    * the variable this kernel operates on ("u").
      *
59    * Note that this can be an approximation or linearization.  In this case it's
      * not because the Jacobian of this operator is easy to calculate.
61    *
      * This function should always be defined in the .C file.
63    */
     virtual Real computeQpJacobian() override;
65
   private:
67   /**
      * A vector object for storing the velocity.  Convenient for
69    * computing dot products.
      */
71   RealVectorValue _velocity;
   };
```

E:/GitHub/moose/examples/ex02_kernel/src/kernels/ExampleConvection.C

```
1  //* This file is part of the MOOSE framework
   //* https://www.mooseframework.org
3  //*
   //* All rights reserved, see COPYRIGHT for full restrictions
5  //* https://github.com/idaholab/moose/blob/master/COPYRIGHT
   //*
7  //* Licensed under LGPL 2.1, please see LICENSE for details
   //* https://www.gnu.org/licenses/lgpl-2.1.html
9
   #include "ExampleConvection.h"
11
   /**
13  * All MOOSE based object classes you create must be registered using this macro.  The first
    * argument is the name of the App you entered in when running the stork.sh script with an "
      App"
```

```
 * suffix. If you ran "stork.sh Example", then the argument here becomes "ExampleApp". The
     second
 * argument is the name of the C++ class you created.
 */
registerMooseObject("ExampleApp", ExampleConvection);
// 如创建Panda的App，那么ExampleApp将变成PandaApp
/**
 * This function defines the valid parameters for
 * this Kernel and their default values
 */
template <>
InputParameters
validParams<ExampleConvection>()
{
  InputParameters params = validParams<Kernel>(); // 父对象
  params.addRequiredParam<RealVectorValue>("velocity", "Velocity Vector");
  return params;
}

ExampleConvection::ExampleConvection(const InputParameters & parameters)
  : // You must call the constructor of the base class first
    Kernel(parameters), // 基类初始化
    _velocity(getParam<RealVectorValue>("velocity")) // 最后没有逗号
{
}

Real
ExampleConvection::computeQpResidual()
{
  // velocity * _grad_u[_qp] is actually doing a dot product
  return _test[_i][_qp] * (_velocity * _grad_u[_qp]);
}

Real
ExampleConvection::computeQpJacobian()
{
  // the partial derivative of _grad_u is just _grad_phi[_j]
  return _test[_i][_qp] * (_velocity * _grad_phi[_j][_qp]);
}
```

2.2.3 Multiphysics Coupling

问题声明：考虑 3-D 域上的耦合方程组 Ω:

创建对流 Kernel, 只需要在算例 2 的基础上进行小的改动，将 v 设置为耦合变量而非已知常数，

E:/GitHub/moose/examples/ex03_coupling/ex03.i

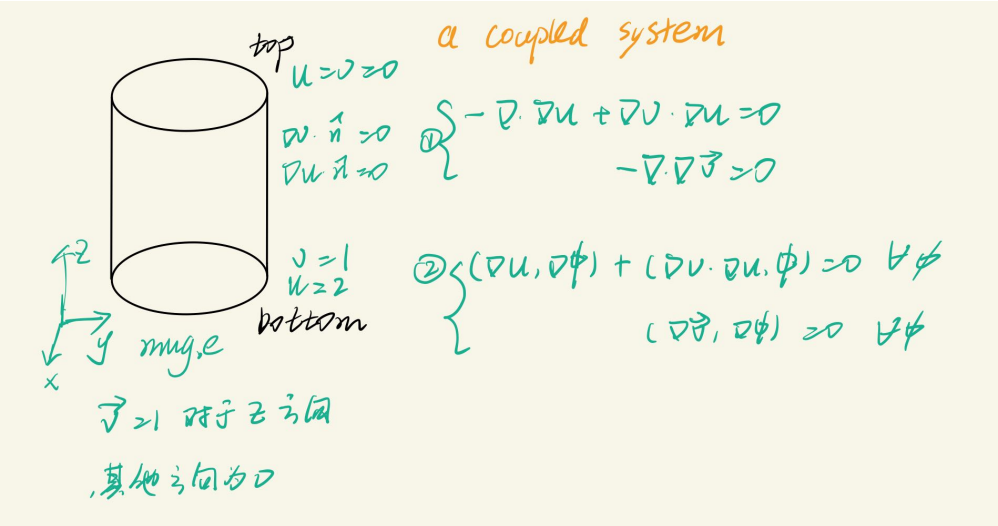

图 2.3 其中 u, v 表示有限元的解，ψ 表示试函数

Figure 2.3 Among them, u, v finite element solution, and ψ means trial function.

```
[Kernels]
  [./diff_convected] # ()
    type = Diffusion
    variable = convected
  [../]

  [./conv]
```

E:/GitHub/moose/examples/ex03_coupling/include/kernels/ExampleConvection.h

```
private:
  const VariableGradient & _grad_some_variable; // const 表示不可修改，可以用来区分~~
```

E:/GitHub/moose/examples/ex03_coupling/src/kernels/ExampleConvection.C

```
ExampleConvection::ExampleConvection(const InputParameters & parameters)
  : Kernel(parameters),
    // using the user-specified name for the coupled variable, retrieve and store a
    reference to the
    // coupled variable.
    _grad_some_variable(coupledGradient("some_variable"))
{
}
```

2.2.4　Custom Boundary Conditions

MOOSE 提供了几个基本的边界条件，可以直接在您的输入文件中使用，也可以扩展以提供自定义行为。本案例中，扩展了 MOOSE 的内置 Dirichlet 和 Neu-

mann 边界条件对象，允许它们与方程系统中的自变量进行耦合。并且提供了一个周期性边界条件的案例。

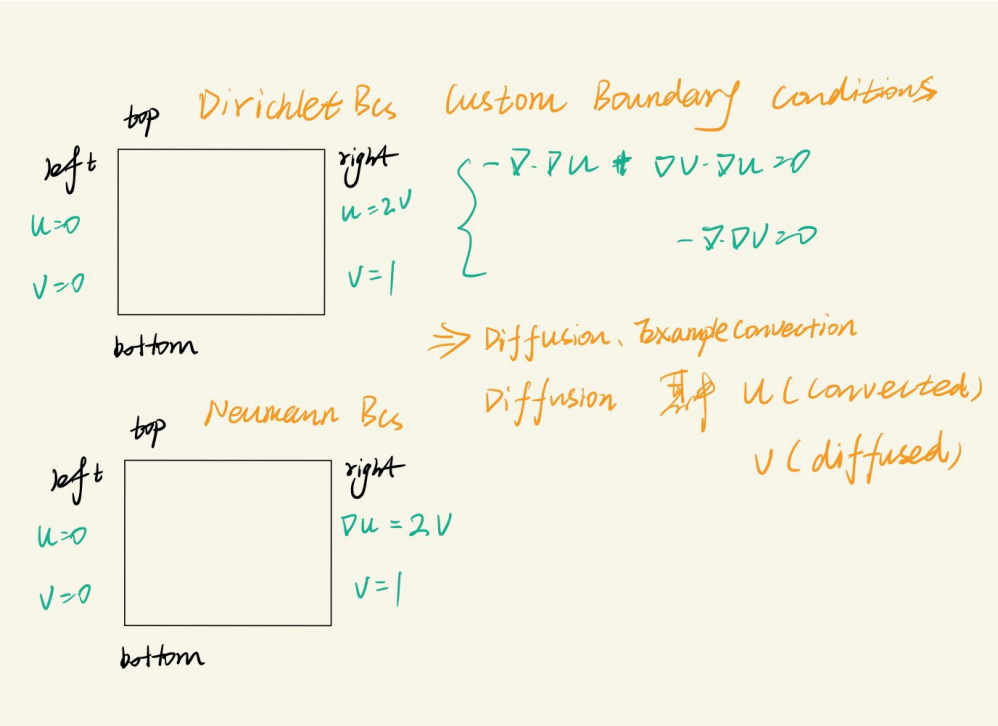

图 2.4 其中 u, v 表示有限元的解，ψ 表示试函数

Figure 2.4 Among them, u, v finite element solution, and ψ means trial function。

MOOSE 中的 Dirichlet BC 都继承自一个 NodalBC 基类，

E:/GitHub/moose/examples/ex04_bcs/include/bcs/CoupledDirichletBC.h

```
1  //* This file is part of the MOOSE framework
   //* https://www.mooseframework.org
3  //*
   //* All rights reserved, see COPYRIGHT for full restrictions
5  //* https://github.com/idaholab/moose/blob/master/COPYRIGHT
   //*
7  //* Licensed under LGPL 2.1, please see LICENSE for details
   //* https://www.gnu.org/licenses/lgpl-2.1.html
9
   #pragma once
11
   #include "NodalBC.h"
13
   class CoupledDirichletBC;
15
   template <>
17 InputParameters validParams<CoupledDirichletBC>();
```

```
19 /// Implements a coupled Dirichlet BC where u = alpha * some_var on the boundary.
   class CoupledDirichletBC : public NodalBC
21 {
   public:
23   CoupledDirichletBC(const InputParameters & parameters);

25 protected:
     virtual Real computeQpResidual() override;
27
   private:
29   /// Multiplier on the boundary.
     Real _alpha;
31   /// reference to a user-specifiable coupled (independent) variable
     const VariableValue & _some_var_val;
33 };
```

E:/GitHub/moose/examples/ex04_bcs/dirichlet_bc.i

```
   [BCs]
2    active = 'left_convected right_convected_dirichlet left_diffused right_diffused'

4    [./left_convected]
       type = DirichletBC
6      variable = convected
       boundary = 'left'
8      value = 0
     [../]
10
     [./right_convected_dirichlet]
12     type = CoupledDirichletBC
       variable = convected
14     boundary = 'right'
       alpha = 2
16
       some_var = diffused
18   [../]

20   # Note: This BC is not active in this input file
     [./right_convected_neumann]
22     type = CoupledNeumannBC
       variable = convected
24     boundary = 'right'
       alpha = 2
26
       some_var = diffused
28   [../]
```

```
  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]
```

MOOSE 中的 Neumann BC 都继承自一个 IntegratedBC 基类，

E:/GitHub/moose/examples/ex04_bcs/include/bcs/CoupledNeumannBC.h

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#pragma once

#include "IntegratedBC.h"

class CoupledNeumannBC;

template <>
InputParameters validParams<CoupledNeumannBC>();

/**
 * Implements a simple constant Neumann BC where grad(u)=alpha * v on the boundary.
 * Uses the term produced from integrating the diffusion operator by parts.
 */
class CoupledNeumannBC : public IntegratedBC
{
public:
  CoupledNeumannBC(const InputParameters & parameters);

protected:
  virtual Real computeQpResidual() override;
```

```
private:
  /// Multiplier on the boundary.
  Real _alpha;
  /// reference to a user-specifiable coupled (independent) variable
  const VariableValue & _some_var_val;
};
```

E:/GitHub/moose/examples/ex04_bcs/neumann_bc.i

```
[BCs]
  active = 'left_convected right_convected_neumann left_diffused right_diffused'

  [./left_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left'
    value = 0
  [../]

  # Note: This BC is not active in this input file
  [./right_convected_dirichlet]
    type = CoupledDirichletBC
    variable = convected
    boundary = 'right'
    alpha = 2

    some_var = diffused
  [../]

  [./right_convected_neumann]
    type = CoupledNeumannBC
    variable = convected
    boundary = 'right'
    alpha = 2

    some_var = diffused
  [../]

  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = 'left'
    value = 0
  [../]

  [./right_diffused]
    type = DirichletBC
```

```
    variable = diffused
    boundary = 'right'
    value = 1
  [../]

[]
```

MOOSE 为指定周期性边界条件提供内置支持。其中周期性边界条件的设置方法有自动和手动方法，

E:/GitHub/moose/examples/ex04_bcs/periodic_bc.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 50
  ny = 50
  nz = 0

  xmax = 40
  ymax = 40
  zmax = 0
  elem_type = QUAD4
[]

[Variables]
  [./u]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  [./diff]
    type = Diffusion
    variable = u
  [../]

  [./forcing]
    type = ExampleGaussContForcing
    variable = u
    x_center = 2
    y_center = 4
  [../]

  [./dot]
    type = TimeDerivative
    variable = u
```

```
  [../]
[]

[BCs]
  [./Periodic]
    #Note: Enable either "auto" or both "manual" conditions for this example
    active = 'manual_x manual_y'

    # Can use auto_direction with Generated Meshes
    [./auto]
      variable = u
      auto_direction = 'x y'
    [../]

    # Use Translation vectors for everything else
    [./manual_x]
      variable = u
      primary = 'left'
      secondary = 'right'
      translation = '40 0 0'
    [../]

    [./manual_y]
      variable = u
      primary = 'bottom'
      secondary = 'top'
      translation = '0 40 0'
    [../]
  [../]
[]

[Executioner]
  type = Transient
  dt = 1
  num_steps = 20
  nl_rel_tol = 1e-12
[]

[Outputs]
  execute_on = 'timestep_end'
  exodus = true
[]
```

2.2.5　Automatic Mesh Adaptivity

MOOSE 支持网格自适应，可以在解决问题时在较高/较低误差的区域自动细化和粗化网格。除了减少计算时间外，这还可以提高结果的质量。无需编写任何

C++ 代码即可使用网格自适应。相反，它可以通过填写输入文件中的 Adaptivity 部分轻松启用：

E:/GitHub/moose/examples/ex05_amr/ex05.i

```
[Mesh]
  file = cube-hole.e
[]

# This is where mesh adaptivity magic happens:
[Adaptivity]
  marker = errorfrac # this specifies which marker from 'Markers' subsection to use
  steps = 2 # run adaptivity 2 times, recomputing solution, indicators, and markers each
     time

  # Use an indicator to compute an error-estimate for each element:
  [./Indicators]
    # create an indicator computing an error metric for the convected variable
    [./error] # arbitrary, use-chosen name
      type = GradientJumpIndicator
      variable = convected
      outputs = none
    [../]
  [../]

  # Create a marker that determines which elements to refine/coarsen based on error
     estimates
  # from an indicator:
  [./Markers]
    [./errorfrac] # arbitrary, use-chosen name (must match 'marker=...' name above
      type = ErrorFractionMarker
      indicator = error # use the 'error' indicator specified above
      refine = 0.5 # split/refine elements in the upper half of the indicator error range
      coarsen = 0 # don't do any coarsening
      outputs = none
    [../]
  [../]
[]

[Variables]
  [./convected]
    order = FIRST
    family = LAGRANGE
  [../]
  [./diffused]
    order = FIRST
    family = LAGRANGE
  [../]
```

```
[]

[Kernels]
  [./example_diff]
    type = ExampleCoefDiffusion
    variable = convected
    coef = 0.125
  [../]
  [./conv]
    type = ExampleConvection
    variable = convected
    some_variable = diffused
  [../]
  [./diff]
    type = Diffusion
    variable = diffused
  [../]
[]

[BCs]
  # convected=0 on all vertical sides except the right (x-max)
  [./cylinder_convected]
    type = DirichletBC
    variable = convected
    boundary = inside
    value = 1
  [../]
  [./exterior_convected]
    type = DirichletBC
    variable = convected
    boundary = 'left top bottom'
    value = 0
  [../]
  [./left_diffused]
    type = DirichletBC
    variable = diffused
    boundary = left
    value = 0
  [../]
  [./right_diffused]
    type = DirichletBC
    variable = diffused
    boundary = right
    value = 10
  [../]
[]

[Executioner]
```

```
  type = Steady
  solve_type = 'PJFNK'

  l_tol = 1e-3
  nl_rel_tol = 1e-12
[]

[Outputs]
  execute_on = 'timestep_end'
  exodus = true
[]
```

关于网格自适应的理解需要进一步整理!!!

2.2.6 Transient Analysis

瞬态分析 vs 稳态分析：在稳态过程中，系统的响应，无论是应力、温度还是其他，都不会随时间变化。在瞬态分析中，此响应与时间有关。所有现实世界的过程都有稳态和瞬态阶段，但根 据所需的结果，分析其中一个更合适。稳态分析和瞬态分析之间的区别适用于分析类型，包括应力、传热、流体流动、静电和磁。

1. Input 文件：ex06.i
2. 头文件：kernels-ExampleTimeDerivative.C
3. 源代码：kernels-ExampleTimeDerivative.C

在扩散方程中加入瞬态项（时间导数项），即 $-\nabla \cdot \nabla u = 20\frac{\partial u}{\partial t}$。通过继承 TimeDerivative 来创建一个时间相关的残差。

2.2.7 Custom Initial Conditions

MOOSE 提供了多种方法来指定瞬态问题的初始条件 (IC)。相同的功能可用于指定稳态问题的初始猜测。除了使用许多用于指定初始条件的内置方法中的任何一种之外，还可以创建自定义方法。

1. Input 文件：transient.i
2. 头文件：ics-ExampleIC.h
3. 源代码：ics-ExampleIC.C

2.2.8　Material Properties

MOOSE 包括对在整个模拟计算过程中创建和共享材料属性的内置支持。MOOSE 为每个正交点自动运行/更新材料属性计算。内核、后处理器和其他对象都可以方便地访问这些属性。此示例演示了使用自定义非线性材料属性的 kernel 的对流扩散问题，其包括材料属性的产生和使用，

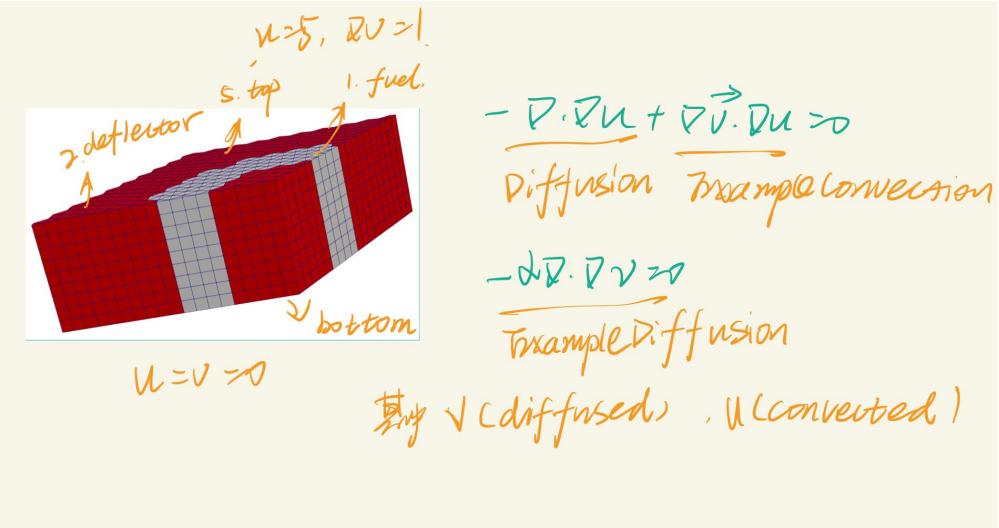

图 2.5 其中 u, v 表示有限元的解，ψ 表示试函数

Figure 2.5 Among them, u, v finite element solution, and ψ means trial function。

1. Input 文件：ex09.i
2. 头文件 1：materials-ExampleMaterial.h
3. 源代码 1：materials-ExampleMaterial.C
4. 头文件 2：kernels-ExampleConvection.h
5. 源代码 2：kernels-ExampleConvection.C

2.2.9　Stateful Materials Properties

本案例主要是用于将材料参数设置为有状态的形式，这样有利于在计算不正常中断时可以在上次计算的基础上再次计算。

1. Input 文件：ex09.i
2. 头文件：materials-ExampleMaterial.h
3. 源代码：materials-ExampleMaterial.C

2.2.10 Auxiliary Variables

辅助变量可以用于后处理等。

1. Input 文件：ex10.i
2. 头文件：auxkernels-ExampleAux.h
3. 源代码：auxkernels-ExampleAux.Cmaterials-ExampleMaterial.C

2.2.11 Preconditioning

准确且完整的预处理矩阵对于有效的预处理无雅可比牛顿 - 克里洛夫 (PJFNK) 求解非常重要。这对于 solve_type = NEWTON 是绝对必要的。MOOSE 有多种用于构建预处理矩阵的选项：

1. (Default) Block Diagonal Preconditioning：块对角预处理使用内核和集成边界条件的 computeQpJacobian 方法来构建块对角矩阵。它不会考虑可变耦合。如果用户未在其输入文件中指定 [Preconditiong] 块，则这是默认设置。
2. (Default) Block Diagonal Preconditifoning：单矩阵预处理器使用内核和集成 BC 的 computeQpJacobian 和 computeQpOffDiagJacobian 方法构建其矩阵，后者负责耦合变量的贡献。计算 QpOffDiagJacobian 方法的一个很好的简单示例是在 CoupledForce 中。用户也可以选择从他们的 SMP 矩阵中省略某些非对角线项；这在详细的预处理文章中进行了概述。
3. Finite Difference Preconditioner (FDP)：这将创建一个近乎完美的预处理矩阵；但是，它非常慢并且只能串行工作。FDP 应仅用于调试目的。
4. Physics Based Preconditioner (PBP)：
5. Input 文件 1：fdp.i
6. Input 文件 1：smp.i

关于预处理矩阵的理解需要进一步整理!!!

2.2.12 Physics Based Preconditioning

1. Input 文件：ex12.i
2. 头文件：kernels-CoupledForce.h
3. 源代码：kernels-CoupledForce.Cmaterials-ExampleMaterial.C

2.2.13　Custom Functions

自定义解析函数：$sin(\alpha\pi x)$

E:/GitHub/moose/examples/ex13_functions/ex13.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2

  nx = 100
  ny = 100

  xmin = 0.0
  xmax = 1.0

  ymin = 0.0
  ymax = 1.0
[]

[Variables]
  [./forced]
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Functions]
  # A ParsedFunction allows us to supply analytic expressions
  # directly in the input file
  [./bc_func]
    type = ParsedFunction
    value = sin(alpha*pi*x)
    vars = 'alpha'
    vals = '16'
  [../]

  # This function is an actual compiled function
  # We could have used ParsedFunction for this as well
  [./forcing_func]
    type = ExampleFunction
    alpha = 16
  [../]
[]

[Kernels]
  [./diff]
    type = Diffusion
    variable = forced
```

```
  [../]

  # This Kernel can take a function name to use
  [./forcing]
    type = BodyForce
    variable = forced
    function = forcing_func
  [../]
[]

[BCs]
  # The BC can take a function name to use
  [./all]
    type = FunctionDirichletBC
    variable = forced
    boundary = 'bottom right top left'
    function = bc_func
  [../]
[]

[Executioner]
  type = Steady
  solve_type = 'PJFNK'
[]

[Outputs]
  execute_on = 'timestep_end'
  exodus = true
[]
```

E:/GitHub/moose/examples/ex13_functions/include/functions/ExampleFunction.h

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#pragma once

#include "Function.h"

class ExampleFunction; // 提前声明

template <>
```

```
InputParameters validParams<ExampleFunction>(); //  参数输入接口

class ExampleFunction : public Function // 继承
{
public:
  ExampleFunction(const InputParameters & parameters); // 构造函数

  virtual Real value(Real t, const Point & p) const override;

protected:
  Real _alpha; // 常数
};
```

E:/GitHub/moose/examples/ex13_functions/src/functions/ExampleFunction.C

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#include "ExampleFunction.h"

registerMooseObject("ExampleApp", ExampleFunction);

template <>
InputParameters
validParams<ExampleFunction>()
{
  InputParameters params = validParams<Function>();
  params.addParam<Real>("alpha", 1.0, "The value of alpha");
  return params;
}

ExampleFunction::ExampleFunction(const InputParameters & parameters)
  : Function(parameters), _alpha(getParam<Real>("alpha"))
{
}

Real
ExampleFunction::value(Real /*t*/, const Point & p) const
{
  return _alpha * _alpha * libMesh::pi * libMesh::pi *
         std::sin(_alpha * libMesh::pi * p(0)); // p(0) == x
}
```

2.2.14 Postprocessors and Code Verification

后处理器和代码验证：与更加细小的网格进行比较，通过比较两个输入文件（ex14_solution_comparison_1.i 和 ex14_solution_comparison_1.i），演示了如何使用 SolutionUserObject 读取精细网格的解，然后将粗网格的解与使用 ElementL2Error 的解进行比较。

1. 第一个输入文件计算精细网格解决方案并输出一个 XDA 文件。

2. XDA 文件包含在以前的解决方案中完美读取所需的全套数据……即使是在自适应的网格上。

3. 接下来，SolutionFunction 利用 SolutionUserObject 将解呈现为 MOOSE 函数。

4. 最后，使用 ElementL2ErrorPostprocessor 来计算解的差异。

问题

1. xda 文件是什么?

2. 如何进行误差分析的?

E:/GitHub/moose/examples/ex14_pps/ex14.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 32
  ny = 32
  xmin = 0.0
  xmax = 1.0
  ymin = 0.0
  ymax = 1.0
[]

[Variables]
  [forced]
    order = FIRST
    family = LAGRANGE
  []
[]

[Functions]
  # A ParsedFunction allows us to supply analytic expressions directly in the input file
  [exact]
    type = ParsedFunction
    value = sin(alpha*pi*x)
```

```
    vars = alpha
    vals = 16
  []

  # This function is an actual compiled function
  [force]
    type = ExampleFunction
    alpha = 16
  []
[]

[Kernels]
  [diff]
    type = ADDiffusion
    variable = forced
  []

  # This Kernel can take a function name to use
  [forcing]
    type = ADBodyForce
    variable = forced
    function = force
  []
[]

[BCs]
  # The BC can take a function name to use
  [all]
    type = FunctionDirichletBC
    variable = forced
    boundary = 'bottom right top left'
    function = exact
  []
[]

[Executioner]
  type = Steady
  solve_type = NEWTON
  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Postprocessors]
  [h]
    type = AverageElementSize
  []
  [error]
    type = ElementL2Error
```

```
    variable = forced
    function = exact
  []
[]

[Outputs]
  execute_on = 'timestep_end'
  exodus = true
  csv = true
[]
```

E:/GitHub/moose/examples/ex14_pps/mms_spatial.py

```
#!/usr/bin/env python3
#* This file is part of the MOOSE framework
#* https://www.mooseframework.org
#*
#* All rights reserved, see COPYRIGHT for full restrictions
#* https://github.com/idaholab/moose/blob/master/COPYRIGHT
#*
#* Licensed under LGPL 2.1, please see LICENSE for details
#* https://www.gnu.org/licenses/lgpl-2.1.html

import mms
df1 = mms.run_spatial('ex14.i', 4, executable='./ex14-opt')
df2 = mms.run_spatial('ex14.i', 4, 'Mesh/second_order=true', 'Variables/forced/order=SECOND'
    , executable='./ex14-opt')

fig = mms.ConvergencePlot(xlabel='Element Size ($h$)', ylabel='$L_2$ Error')
fig.plot(df1, label='1st Order', marker='o', markersize=8)
fig.plot(df2, label='2nd Order', marker='o', markersize=8)
fig.save('ex14_mms.png')
```

E:/GitHub/moose/examples/ex14_pps/mms_exact.py

```
#!/usr/bin/env python3
#* This file is part of the MOOSE framework
#* https://www.mooseframework.org
#*
#* All rights reserved, see COPYRIGHT for full restrictions
#* https://github.com/idaholab/moose/blob/master/COPYRIGHT
#*
#* Licensed under LGPL 2.1, please see LICENSE for details
#* https://www.gnu.org/licenses/lgpl-2.1.html

import mms
fs,ss = mms.evaluate('-div(grad(u))', 'sin(a*pi*x)', scalars=['a'])
mms.print_fparser(fs)
mms.print_hit(fs, 'force')
```

```
mms.print_hit(ss, 'exact')
```

2.2.15　Custom Actions

自定义动作，本例创建动作 ConvectionDiffusionAction，用于创建 kernles-convected，kernles-diffused，

E:/GitHub/moose/examples/ex14_pps/ex14.i

```
[Functions]
  # A ParsedFunction allows us to supply analytic expressions directly in the input file
  [exact]
    type = ParsedFunction
    value = sin(alpha*pi*x)
    vars = alpha
    vals = 16
```

E:/GitHub/moose/examples/ex15_actions/include/actions/ConvectionDiffusionAction.h

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#pragma once

#include "Action.h"

class ConvectionDiffusionAction : public Action
{
public:
  ConvectionDiffusionAction(InputParameters params);

  virtual void act() override;
};

template <>
InputParameters validParams<ConvectionDiffusionAction>();
```

E:/GitHub/moose/examples/ex15_actions/src/actions/ConvectionDiffusionAction.C

```
//* This file is part of the MOOSE framework
//* https://www.mooseframework.org
```

```
//*
//* All rights reserved, see COPYRIGHT for full restrictions
//* https://github.com/idaholab/moose/blob/master/COPYRIGHT
//*
//* Licensed under LGPL 2.1, please see LICENSE for details
//* https://www.gnu.org/licenses/lgpl-2.1.html

#include "ConvectionDiffusionAction.h"
#include "Factory.h"
#include "Parser.h"
#include "FEProblem.h"

registerMooseAction("ExampleApp", ConvectionDiffusionAction, "add_kernel");

template <>
InputParameters
validParams<ConvectionDiffusionAction>()
{
  InputParameters params = validParams<Action>();
  params.addRequiredParam<std::vector<NonlinearVariableName>>(
      "variables", "The names of the convection and diffusion variables in the simulation");

  return params;
}

ConvectionDiffusionAction::ConvectionDiffusionAction(InputParameters params) : Action(params
    ) {}

void
ConvectionDiffusionAction::act()
{
  std::vector<NonlinearVariableName> variables =
      getParam<std::vector<NonlinearVariableName>>("variables");
  std::vector<VariableName> vel_vec_variable;

  /**
   * We need to manually setup our Convection-Diffusion and Diffusion variables on our two
   * variables we are expecting from the input file.  Much of the syntax below is hidden by
     the
   * parser system but we have to set things up ourselves this time.
   */

  // Do some error checking
  mooseAssert(variables.size() == 2, "Expected 2 variables, received " << variables.size());

  // Setup our Diffusion Kernel on the "u" variable
  {
    InputParameters params = _factory.getValidParams("Diffusion");
```

```
    params.set<NonlinearVariableName>("variable") = variables[0];
    _problem->addKernel("Diffusion", "diff_u", params);
  }

  // Setup our Convection Kernel on the "u" variable coupled to the diffusion variable "v"
  {
    InputParameters params = _factory.getValidParams("ExampleConvection");
    params.set<NonlinearVariableName>("variable") = variables[0];
    //    params.addCoupledVar("some_variable", "The gradient of this var");
    vel_vec_variable.push_back(variables[1]);
    params.set<std::vector<VariableName>>("some_variable") = vel_vec_variable;
    _problem->addKernel("ExampleConvection", "conv", params);
  }

  // Setup out Diffusion Kernel on the "v" variable
  {
    InputParameters params = _factory.getValidParams("Diffusion");
    params.set<NonlinearVariableName>("variable") = variables[1];
    _problem->addKernel("Diffusion", "diff_v", params);
  }
}
```

E:/GitHub/moose/examples/ex15_actions/src/base/ExampleApp.C

```
/**
 * An Action is a little different than registering the other MOOSE
 * objects.  First, you need to register your Action like normal in its file with
 * the registerMooseAction macro. - e.g.:
 *
 *     registerMooseAction("ExampleApp", ConvectionDiffusionAction, "add_kernel");
 *
 * Then we need to tell the parser what new section name to look for and what
 * Action object to build when it finds it.  This is done directly on the syntax
 * with the registerActionSyntax method.
 *
 * The section name should be the "full path" of the parsed section but should NOT
 * contain a leading slash.  Wildcard characters can be used to replace a piece of the
 * path.
 */
registerSyntax("ConvectionDiffusionAction", "ConvectionDiffusion");
```

2.2.16　Creating a Custom Timestepper

E:/GitHub/moose/examples/ex16_timestepper/ex16.i

```
[Kernels]
  [./example_diff]
    type = ExampleDiffusion
```

```
    variable = convected
[../]

[./conv]
  type = ExampleConvection
```

2.2.17 Adding a Dirac Kernel

Dirac Kernel 是啥!!!

1. Input 文件：ex17.i
2. 头文件：DiracKernel-CoupledForce.h
3. 源代码：DiracKernel-CoupledForce.C

2.2.18 ODE Coupling

ODE(ordinary differential equation) Coupling

2.2.19 UserObjects

问题是依赖于时间的扩散，Dirichlet 边界条件左侧为 0，右侧为 1。材料中计算的扩散系数取决于每个块上变量的平均值。因此，随着浓度扩散，扩散系数增加，但每个块的系数不同（基于该块上变量的平均值）。为此，我们需要 3 个对象协同工作：

1. UserObjects-BlockAverageValue：：一个 UserObject，它计算域的每个块上变量的平均值，并提供 averageValue() 以检索特定块上的平均值。userobjects-CoupledForce.h，userobjects-BlockAverageValue.C

2. Materials-BlockAverageDiffusionMaterial：一种基于由 BlockAverageValue UserObject 计算的变量平均值计算“扩散率”的 Material 对象。

3. Kernels-ExampleDiffusion：之前看到的相同内核使用了“diffusivity”材料属性。这个类的主要目的是提供一个 averageValue 方法，它接受一个 SubdomainID，它只是一个整数值，指定网格的哪个块来执行平均值计算。

E:/GitHub/moose/examples/ex20_user_objects/ex20.i

```
[Mesh]
  file = two_squares.e
  dim = 2
[]

```

```
[Variables]
  [./u]
    initial_condition = 0.01
  [../]
[]

[Kernels]
  [./diff]
    type = ExampleDiffusion
    variable = u
  [../]
  [./td]
    type = TimeDerivative
    variable = u
  [../]
[]

[BCs]
  [./left]
    type = DirichletBC
    variable = u
    boundary = leftleft
    value = 0
  [../]
  [./right]
    type = DirichletBC
    variable = u
    boundary = rightright
    value = 1
  [../]
[]

[Materials]
  [./badm]
    type = BlockAverageDiffusionMaterial
    block = 'left right'
    block_average_userobject = bav
  [../]
[]

[UserObjects]
  [./bav]
    type = BlockAverageValue
    variable = u
    execute_on = timestep_begin
    outputs = none
  [../]
[]
```

```
[Executioner]
  type = Transient
  num_steps = 10
  dt = 1

  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]
```

2.2.20 Debugging

调试:

1. 在 MOOSE 应用程序开发生涯的某个时刻，将产生一个错误这是不可避免的。

2. 有时，打印语句足以帮助确定错误的原因。

3. 对于更复杂的错误，调试器在帮助查明问题方面比打印语句更有效。

4. 存在许多调试器：LLDB、GDB、Totalview、ddd 等。

5. 通常最好使用与编译器相关联的调试器（如果有）。

6. 这里我们重点介绍 LLDB/GDB，因为它使用起来相对简单，并且包含在 MOOSE 二进制可再发行包中。

7. “Segmentation fault”、“Segfault”或“Signal 11”错误表示内存错误（通常是数组访问越界）。

要将调试器与基于 MOOSE 的应用程序一起使用，您必须以优化模式 (opt) 以外的其他方式编译您的应用程序。我们强烈推荐调试 (dbg) 模式，以便您在堆栈跟踪中获得完整的行号信息:

```
cd ~/projects/moose/examples/ex21_debugging
METHOD=dbg make -j8
gdb --args ./ex21-dbg -i ex21.i
```

2.3　MultiApp Demonstration

链接，使用 MOOSE MultiApps 和 Transfers 无缝耦合原生 (MOOSE) 应用程序；使用 MOOSE-Wrapped Apps 有效地耦合非本地代码；并且应用程序开发侧重于实现物理 (PDE) 而不是数值实现问题。

2.3.1　介绍

耦合范式分为三类，分别为：Loosely-Coupled；Tightly-Coupled / Picard；Fully-Coupled，其主要特点如下：

1. Loosely-Coupled

(a) Each physics solved with a separate linear/nonlinear solve.

(b) Data exchange once per timestep (typically)

2. Tightly-Coupled / Picard

(a) Each physics solved with a separate linear/nonlinear solve.

(b) Data is exchanged and physics re-solved until "convergence"

3. Fully-Coupled

(a) Each physics solved with a separate linear/nonlinear solve.

(b) All physics solved for in one linear/nonlinear solve

MOOSE 最初是为了解决偏微分方程组的完全耦合系统而创建的，但是并非所有系统都需要完全耦合。MultiApp 系统允许多个 MOOSE（或外部）应用程序同时并行运行。一个 MultiApp 可能代表数千个单独的解决方案。MOOSE 中的 Transfer 系统旨在向 MultiApps 推送和拉取场和数据。

2.3.2　MultiApps

MultiApp System,

1. 基于 MOOSE 的求解可以嵌套以实现多尺度多物理场仿真，如宏观模拟可以与嵌入式微观结构模拟相结合。

2. 任意级别的求解器。

3. 每个求解都并行展开，以最有效地利用计算资源。

4. 有效地将多个团队的代码联系在一起。

5. MOOSE 封装的外部应用程序可以存在于该层次结构中的任何位置。

6. 应用程序不需要知道它们在 MultiApp 层次结构中运行！

本节中的案例只是在 MasterApp 连接 subApp，但是之间并没有进行数据之间传输，这个将在下一节中完成。主和次应用之间的连接是通过对象 Transient-MultiApp

2.3.2.1 01_master

在相同的 dt 下, 先计算 subApp, 之后计算 MasterApp。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/01_master.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [u]
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = u
  []
  [force]
    type = BodyForce
    variable = u
    value = 1.
  []
  [td]
    type = TimeDerivative
    variable = u
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = u
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
```

```
    variable = u
    boundary = right
    value = 0
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 1.

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]

[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions = '0 0 0' # MasterApp中计算subApp的位置
    input_files = '01_sub.i'
  []
[]

# 在相同的dt下,先计算subApp,之后计算MasterApp
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/01_sub.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [./v]
  [../]
[]

[Kernels]
  [./diff]
    type = Diffusion
```

```
    variable = v
  [../]
  [./td]
    type = TimeDerivative
    variable = v
  [../]
[]

[BCs]
  [./left]
    type = DirichletBC
    variable = v
    boundary = left
    value = 0
  [../]
  [./right]
    type = DirichletBC
    variable = v
    boundary = right
    value = 1
  [../]
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 1

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/01_master.log

```

Creating MultiApp sub_app of type pandaTestApp of level 1 and number 0:
sub_app0:

Running App: main

Framework Information:
MOOSE Version:           git commit b9ee36a60b on 2021-07-01
```

```
LibMesh Version:
PETSc Version:            3.15.1
SLEPc Version:            3.15.1
Current Time:             Mon Sep 13 21:51:15 2021
Executable Timestamp:     Sun Aug  1 19:47:41 2021

Parallelism:
  Num Processors:          1
  Num Threads:             1

Mesh:
  Parallel Type:           replicated
  Mesh Dimension:          2
  Spatial Dimension:       2
  Nodes:
    Total:                 121  # 节点的数目:(10+1)^2
    Local:                 121
  Elems:
    Total:                 100 # 单元数目: 10^2
    Local:                 100
  Num Subdomains:          1
  Num Partitions:          1

Nonlinear System:
  Num DOFs:                121 # 自由度 = 变量数目*节点数目
  Num Local DOFs:          121
  Variables:               "u"
  Finite Element Types:    "LAGRANGE"
  Approximation Orders:    "FIRST"

Execution Information:
  Executioner:             Transient
  TimeStepper:             ConstantDT
  Solver Mode:             Preconditioned JFNK
  PETSc Preconditioner:    hypre boomeramg

Initializing All MultiApps
sub_app0:
sub_app0: Running App: sub_app0
sub_app0: Parallelism:
sub_app0:   Num Processors:          1
sub_app0:   Num Threads:             1
sub_app0:
sub_app0: Mesh:
sub_app0:   Parallel Type:           replicated
sub_app0:   Mesh Dimension:          2
sub_app0:   Spatial Dimension:       2
sub_app0:   Nodes:
```

```
sub_app0:      Total:                  121
sub_app0:      Local:                  121
sub_app0:    Elems:
sub_app0:      Total:                  100
sub_app0:      Local:                  100
sub_app0:    Num Subdomains:           1
sub_app0:    Num Partitions:           1
sub_app0:
sub_app0: Nonlinear System:
sub_app0:    Num DOFs:                 121
sub_app0:    Num Local DOFs:           121
sub_app0:    Variables:                "v"
sub_app0:    Finite Element Types:     "LAGRANGE"
sub_app0:    Approximation Orders:     "FIRST"
sub_app0:
sub_app0: Execution Information:
sub_app0:    Executioner:              Transient
sub_app0:    TimeStepper:              ConstantDT
sub_app0:    Solver Mode:              Preconditioned JFNK
sub_app0:    PETSc Preconditioner:     hypre boomeramg
sub_app0:
sub_app0:
sub_app0: Time Step 0, time = 0
Finished Initializing All MultiApps

Time Step 0, time = 0

Time Step 1, time = 1, dt = 1

Backing Up MultiApps on TIMESTEP_BEGIN
Beginning backing up MultiApp sub_app
Finished backing up MultiApp sub_app
Finished Backing Up MultiApps on TIMESTEP_BEGIN


No Transfers on TIMESTEP_BEGIN To MultiApps


Executing MultiApps on TIMESTEP_BEGIN
Solving MultiApp 'sub_app' with target time 1 and dt 1 with auto-advance on
sub_app0:
sub_app0: Time Step 1, time = 1, dt = 1
sub_app0:  0 Nonlinear |R| = 3.077070e+00
sub_app0:       0 Linear |R| = 3.077070e+00
sub_app0:       1 Linear |R| = 1.002240e-01
sub_app0:       2 Linear |R| = 2.200562e-03
sub_app0:       3 Linear |R| = 7.233961e-05
sub_app0:       4 Linear |R| = 1.140542e-06
```

```
sub_app0:  1 Nonlinear |R| = 1.147170e-06
sub_app0:       0 Linear |R| = 1.147170e-06
sub_app0:       1 Linear |R| = 2.902123e-08
sub_app0:       2 Linear |R| = 3.840552e-10
sub_app0:       3 Linear |R| = 1.271138e-11
sub_app0:       4 Linear |R| = 3.724333e-13
sub_app0:  2 Nonlinear |R| = 3.767918e-13
sub_app0:  Solve Converged!
Successfully Solved MultiApp sub_app.
Finished Executing MultiApps on TIMESTEP_BEGIN


No Transfers on TIMESTEP_BEGIN To MultiApps

 0 Nonlinear |R| = 9.246621e-02
      0 Linear |R| = 9.246621e-02
      1 Linear |R| = 1.018933e-02
      2 Linear |R| = 2.162902e-04
      3 Linear |R| = 2.932690e-06
      4 Linear |R| = 4.947314e-08
 1 Nonlinear |R| = 4.950753e-08
      0 Linear |R| = 4.950753e-08
      1 Linear |R| = 1.255670e-09
      2 Linear |R| = 2.917129e-11
      3 Linear |R| = 1.068090e-12
      4 Linear |R| = 1.515029e-14
 2 Nonlinear |R| = 1.518402e-14
 Solve Converged!
```

2.3.2.2　02_master_sublimit

```
[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2 # 将subApp的dt减小，小于masterApp dt = 1

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]
```

若 subApp 的 dt = 0.2 小于 MasterApp 的 dt = 1；则在一个 dt=1 中，先使用 dt = 0.2 五次计算 subApp，之后计算 MasterApp，这为一个周期。

2.3.2.3 03_master_subcycle

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/03_master_subcycle.i

```
[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions = '0 0 0'
    input_files = '03_sub_subcycle.i'
    sub_cycling = true
    # 默认情况下，不输出中间步骤 - 只有子应用程序到达主程序时间后才输出最终解。
    # 可以使用“sub_cycling”参数启用执行子循环的能力，这允许子应用程序在每次执行时执行多个时间步骤。
    # output_sub_cycles = true
    # To enable outputting all steps solved by the sub-app turn on output_subcycles in the MultiApp block.
  []
[]
```

2.3.2.4 04_master_multiple

在相同的 MultiApp 中具有多个 sub-apps; 其中由两种方法提供耦合位置：1. 直接使用 MultiApp 中的坐标点，2. 使用 inputFiles 输入文件（txt 文件，每列 3 个数据点）；对于位置，有两个途径去指定输入文件的位置：1. 单个输入文件，即每一个 sub-app 使用相同的输入文件，2. 为每个位置指定输入文件，每一个 sub-app 使用指定的输入位置文件。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/04_master_multiple.i

```
[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions   = '0 0 0  1 0 0  2 0 0'
    # positions_file = 04_positions.txt
    # input_files = '04_sub1_multiple.i'
    input_files = '04_sub1_multiple.i  04_sub2_multiple.i 04_sub3_multiple.i'
    output_in_position = true
    # 每一个subApp中的*.e文件左下角反应的是positions中的坐标点
  []
[]
```

三个位置，三个 subApp(基于 04_sub1_multiple.i)，产生 04_master_multiple_out_sub_app0-2.e；

当并行运行 MultiApps and sub-apps 时，主应用程序始终在全部处理器上运

行。出于这个原因，使 Master 成为最大、最困难的解决方案通常是有利的。每个 MultiApp 一次执行一个（暂时会清楚）。MultiApp 中的子应用程序全部同时执行（如果可能）。运行 05_master_parallel.i。结果：1 个核心，需要 37.348s；3 个核心，需要 15.941s；6 个核心，需要 9.848s。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/05_master_parallel.i

```
[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions   = '0 0 0  1 0 0  2 0 0'
    input_files = '05_sub_parallel.i'
  []
[]
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/06_master_twoapps.i

```
[MultiApps]
  [app1]
    type = TransientMultiApp
    positions   = '0 0 0  1 0 0  2 0 0'
    input_files = '06_sub_twoapps.i'
  []
```

2.3.2.5　07_master_multilevel

这允许任意深度的多尺度（空间-时间）模拟。考虑具有地震分析的核反应堆模拟：千米级地震模拟；米级安全壳模拟；米级二次模拟；米级压力容器模拟；厘米级中子学模拟；厘米级流体模拟；毫米级 CFD 计算；厘米级燃料模拟；微米级材料模拟。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/07_master_multilevel.i

```
[MultiApps]
  [uno]
    type = TransientMultiApp
    positions   = '0 0 0  1 0 0'
    input_files = '07_sub_multilevel.i'
  []
[]
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step01_multiapps/07_sub_multilevel.i

```
[MultiApps]
  [dos]
```

```
    type = TransientMultiApp
    positions   = '0 0 0  1 0 0'
    input_files = '07_sub_sub_multilevel.i'
  []
[]
```

2.3.3 Transfers

Transfers System，Transfers 是在 MultiApp 层次结构中上下移动信息的方式。Transfers 可以在三个不同的地方读取信息和存款信息：AuxiliaryVariable fields；Postprocessors；UserObjects。一些最常用的传输：

1. MeshFunction: Interpolate a field from one domain to another；
2. NearestNode: Move field data by matching nodes/centroids；
3. Postprocessor: Move PP data from one app to another
4. UserObject: Evaluate a ”spatial” UO in one app at the other app’s nodes/centroids and deposit the information in an AuxiliaryVariable field

最重要的是：通过让 MOOSE 移动数据和填充场..... 应用程序不需要知道或关心数据来自哪里或如何到达那里！之所以称为“MeshFunction”，是因为它将一个域（Solution 或 Aux）变成了一个空间“函数”，可以在域内的任何地方进行评估。然后用于在接收应用程序网格的节点/质心处对该字段进行采样，以填充辅助变量字段。所需参数：

1. direction: Like all Transfers, this is either to_multiapp or from_multiapp
2. multi_app: Which MultiApp to interact with
3. source_variable: The variable to read from
4. variable: The Auxiliary variable to write to

所有传输都与维度无关（如果有意义的话！）。在与 MultiApp 之间传输时，单个传输实际上将同时传输到该 Multi-App 中的所有子应用程序或从该

1. Field Mapping: L2 Projection, Interpolation, Evaluation
2. “Postprocessed” Spatial Data: Layered Integrals and Averages, Assembly Averaged Data, etc.
3. Scalar Transfer: Postprocessor values (Integrals, Averages, Point Evaluations, etc.); Can be transferred as a scalar or interpolated into a field; Useful for multi-scale.

2.3.3.1　01_master_meshfunction

在相同的 dt 下, 先计算 subApp, 之后计算 MasterApp。使用到对象 MultiAppMeshFunctionTransfer 来进行数据之间的传输。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/01_master_meshfunction.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [u]
  []
[]

[AuxVariables]
  [tv]
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = u
  []
  [force]
    type = BodyForce
    variable = u
    value = 1.
  []
  [td]
    type = TimeDerivative
    variable = u
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = u
    boundary = left
    value = 0
  []
  [right]
```

```
    type = DirichletBC
    variable = u
    boundary = right
    value = 0
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]

[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions = '0.5 0 0'
    input_files = '01_sub_meshfunction.i'
  []
[]

[Transfers]
  [pull_v]
    type = MultiAppMeshFunctionTransfer
    # The name of the MultiApp to use.
    multi_app = sub_app

    # Transfer from the sub-app to this app
    # Whether this Transfer will be 'to' or 'from' a MultiApp, or bidirectional
    direction = from_multiapp

    # The name of the variable in the sub-app
    # The variable to transfer from.
    source_variable = v

    # The name of the auxiliary variable in this app
    # The auxiliary variable to store the transferred values in.
    variable = tv
  []
```

```
  [push_u]
    type = MultiAppMeshFunctionTransfer

    multi_app = sub_app

    # Transfer to the sub-app from this app
    direction = to_multiapp

    # The name of the variable in this app
    source_variable = u

    # The name of the auxiliary variable in the sub-app
    variable = tu
  []
[]

# 在主程序中出现tv的辅助变量，用于传输由01_sub_meshfunction.i计算的有限元解
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/01_sub_meshfunction.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 9
  ny = 9
[]

[Variables]
  [v]
  []
[]

[AuxVariables]
  [tu]
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = v
  []
  [td]
    type = TimeDerivative
    variable = v
  []
[]
```

```
[BCs]
  [left]
    type = DirichletBC
    variable = v
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
    variable = v
    boundary = right
    value = 1
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]
```

2.3.3.2 02_master_nearestnode

有时插值成本太高，或者没有意义。例如，代表 z 方向“无限”体积的 x,y 平面中的“2D”计算可能需要耦合到覆盖相同空间的 3D 计算。在这种情况下，当从 2D 计算转换到 3D 计算时，同一 x,y 列中的所有节点/元素都应接收相同的值。可以使用 hrefhttps://mooseframework.inl.gov/source/transfers/MultiAppNearestNodeTransfer.htmlMultiAppNea来完成。这个想法是接收网格中的每个节点（或 CONSTANT/MONOMIAL 字段的元素质心）与发送网格中的最近匹配配对。然后可以轻松地使用这些将数据从发送网格移动到接收网格。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/02_master_nearestnode.i

```
[Mesh]
  type = GeneratedMesh
```

```
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [u]
  []
[]

[AuxVariables]
  [tv]
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = u
  []
  [force]
    type = BodyForce
    variable = u
    value = 1.
  []
  [td]
    type = TimeDerivative
    variable = u
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = u
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
    variable = u
    boundary = right
    value = 1
  []
[]

[Executioner]
  type = Transient
```

```
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]

[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions = '0.1 0.1 0  0.4 0.4 0  0.7 0.7 0'
    input_files = '02_sub_nearestnode.i'
    execute_on = timestep_end
    output_in_position = true
  []
[]

[Transfers]
  [push_u]
    type = MultiAppNearestNodeTransfer

    multi_app = sub_app

    # Transfer to the sub-app from this app
    direction = to_multiapp

    # The name of the variable in this app
    source_variable = u

    # The name of the auxiliary variable in the sub-app
    variable = tu
  []
[]
```

2.3.3.3 03_master_uot

通常需要移动的不是“场”——而是空间上的均匀化之后的值。例如，如果表示流体流经管道的 1D 模拟正在通过产生热量的 3D 域，那么将管道周围产生的热量整合到 3D 域内并将其传递到管道模拟中可能是有利的。这种情况由 MultiAppUserObjectTransfer 直接处理。请记住，后处理器是 UserObjects 的

特例：它们只计算单个值。UserObjects 本身可以被认为是后处理器，可以计算比单个值多得多的值。在 MOOSE 中，对于保存具有某种空间相关性的数据的用户对象有一个特殊的名称：它们是“Spatial UserObjects”……并且它们能够在空间的任何点（有时是时间！）进行评估。一些可能对传输有用的 Spatial UserObjects：1.(NearestPoint)LayeredAverage：计算“层”中一个方向的场的平均值。“NearestPoint”版本可以从给定点周围的元素创建多个 LayeredAverages。2.(NearestPoint)LayeredIntegral: 相似；(NearestPoint)LayeredSideAverage：相似；SolutionUserObject：将文件中的 solution 呈现为空间用户对象。

这个例子与上一个相似——除了我们还制作了主域 3D。这个想法是在子应用程序附近整合来自主应用程序的场，并将其传输到每个子应用程序。相反，子应用程序字段在列上的层中进行平均，并将这些平均值传输回主应用程序。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/03_master_uot.i

```
[Mesh]
  type = GeneratedMesh
  dim = 3

  nx = 10
  ny = 10
  nz = 10

  zmax = 3
[]

[Variables]
  [u]
  []
[]

[AuxVariables]
  [v_average]
    order = CONSTANT
    family = MONOMIAL
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = u
  []
  [force]
```

```
    type = BodyForce
    variable = u
    value = 1.
  []
  [td]
    type = TimeDerivative
    variable = u
  []
[]

[BCs]
  [front]
    type = DirichletBC
    variable = u
    boundary = front
    value = 0
  []
  [back]
    type = DirichletBC
    variable = u
    boundary = back
    value = 1
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]

[UserObjects]
  [layered_integral]
    type = NearestPointLayeredIntegral
    points = '0.15 0.15 0  0.45 0.45 0  0.75 0.75 0'
    direction = z
    num_layers = 4
    variable = u
  []
```

```
[]

[MultiApps]
  [sub_app]
    type = TransientMultiApp
    positions = '0.15 0.15 0  0.45 0.45 0  0.75 0.75 0'
    input_files = '03_sub_uot.i'
    execute_on = timestep_end
    output_in_position = true
  []
[]

[Transfers]
  [push_u]
    type = MultiAppUserObjectTransfer
    multi_app = sub_app
    direction = to_multiapp
    variable = u_integral
    user_object = layered_integral
  []

  [pull_v]
    type = MultiAppUserObjectTransfer
    multi_app = sub_app
    direction = from_multiapp
    variable = v_average
    user_object = layered_average
  []
[]
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/03_sub_uot.i

```
[Mesh]
  type = GeneratedMesh
  dim = 3
  nx = 1
  ny = 1
  nz = 10

  xmin = -0.05
  xmax = 0.05
  ymin = -0.05
  ymax = 0.05
  zmax = 3
[]

[Variables]
  [v]
```

```
  []
[]

[AuxVariables]
  [u_integral]
    order = CONSTANT
    family = MONOMIAL
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = v
  []
  [td]
    type = TimeDerivative
    variable = v
  []
[]

[BCs]
  [front]
    type = DirichletBC
    variable = v
    boundary = front
    value = 0
  []
  [back]
    type = DirichletBC
    variable = v
    boundary = back
    value = 1
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
```

```
  exodus = true
[]

[UserObjects]
  [layered_average]
    type = NearestPointLayeredAverage
    points = '0 0 0'
    direction = z
    num_layers = 4
    variable = v
  []
[]
```

2.3.3.4　04_master_multiscale

当子应用域代表主域的极小部分时 - 需要不同类型的传输。例如，如果主域是 1m x 1m，子应用域是 1nm x 1nm，那么尝试从主域“插入”一个场是没有意义的。实际上，子应用程序位于主域内的点上。最后一类传输，那些移动的标量值，在这里发挥作用。VariableValueSampleTansfer 等“采样”传输将自动评估来自子应用程序位置的主域的单个值并将其移动到子应用程序。在相反的方向上，后处理器传输将同质化的值从子应用程序移动到主应用程序。

这里使用 VariableValueSampleTransfer 将值从主域获取到子应用程序。相反，PostprocessorInterpolationTransfer 用于获取在微观模拟中计算的值，然后在它们之间进行插值以创建平滑场。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/04_master_multiscale.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [u]
  []
[]

[AuxVariables]
  [vt]
  []
[]
```

```
[Kernels]
  [diff]
    type = Diffusion
    variable = u
  []
  [force]
    type = BodyForce
    variable = u
    value = 1.
  []
  [td]
    type = TimeDerivative
    variable = u
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = u
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
    variable = u
    boundary = right
    value = 1
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]

[MultiApps]
```

```
  [micro]
66   type = TransientMultiApp
     positions = '0.15 0.15 0  0.45 0.45 0  0.75 0.75 0'
68   input_files = '04_sub_multiscale.i'
     cli_args = 'BCs/right/value=1 BCs/right/value=2 BCs/right/value=3'
70   # 传递给子应用程序的其他命令行参数。 如果提供了一个集合，则参数将应用于所有，否则每个子
     应用程序都必须有一个集合。
     execute_on = timestep_end
72   output_in_position = true
  []
74 []

76 [Transfers]
  [push_u]
78   type = MultiAppVariableValueSampleTransfer
     multi_app = micro
80   direction = to_multiapp
     source_variable = u
82   variable = ut
  []
84
  [pull_v]
86   type = MultiAppPostprocessorInterpolationTransfer
     multi_app = micro
88   direction = from_multiapp
     variable = vt
90   postprocessor = average_v
  []
92 []
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step02_transfers/04_sub_multiscale.i

```
[Mesh]
2  type = GeneratedMesh
   dim = 2
4  nx = 10
   ny = 10
6 []

8 [Variables]
  [v]
10 []
[]
12
[AuxVariables]
14 [ut]
  []
16 []
```

```
[Kernels]
  [diff]
    type = Diffusion
    variable = v
  []
  [td]
    type = TimeDerivative
    variable = v
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = v
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
    variable = v
    boundary = right
    value = 1
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
[]

[Postprocessors]
  [average_v]
    type = ElementAverageValue
    variable = v
  []
[]
```

2.3.4　MultiApp Coupling

现在我们需要结合这些功能来执行耦合计算。知道基于 MOOSE 的计算很容易耦合到辅助变量或后处理器值 - 很明显这是如何工作的：开发在耦合到的辅助场中具有替代值的独立计算。将它们放在一个 MultiApp hierarchy 中; 利用 Transfers 将值从一个应用程序移动到另一个应用程序，以用“真实”值填充 Auxiliary fields。

在这里，我们将在每个时间步简单地交换一次值并继续。这可以通过 MultiApps 和 Transfers 轻松实现。默认情况下，MultiApp 将在时间步长的正确时刻执行（如 $execute_on$ 所指示），并且与该 MultiApp 关联的传输也在该时间执行。然后模拟将进入下一个时间步。

这里我们继续进行微观结构计算。但是现在我们将添加耦合到传输场的 Kernels 和 MaterialProperties。主应用程序可以被认为是计算一个“源”项——然后在子应用程序中用作强制函数。主应用程序可以被认为是计算一个“源”项——然后在子应用程序中用作强制函数。我们最终得到的是整个主域上的平滑场，它代表“扩散率”- 然后将其作为“扩散率”输入到由主应用程序求解的扩散方程中。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step03_coupling/01_master.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [u]
  []
[]

[AuxVariables]
  [vt]
  []
[]

[Kernels]
  [diff]
    type = MatDiffusion
```

```
    variable = u
    # the diffusion coefficient $D$ (‘diffusivity‘) is provided by a ‘FunctionMaterial‘
     function material
  []
  [force]
    type = BodyForce
    variable = u
    value = 1.
  []
  [td]
    type = TimeDerivative
    variable = u
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = u
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
    variable = u
    boundary = right
    value = 1
  []
[]

[Materials]
  [diff]
    type = ParsedMaterial
    f_name = D
    args = 'vt'
    function = 'vt'
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
```

```
[]

[Outputs]
  exodus = true
[]

[MultiApps]
  [micro]
    type = TransientMultiApp
    positions = '0.15 0.15 0  0.45 0.45 0  0.75 0.75 0'
    input_files = '01_sub.i'
    execute_on = timestep_end
    output_in_position = true
  []
[]

[Transfers]
  [push_u]
    type = MultiAppVariableValueSampleTransfer
    multi_app = micro
    direction = to_multiapp
    source_variable = u
    variable = ut
  []

  [pull_v]
    type = MultiAppPostprocessorInterpolationTransfer
    multi_app = micro
    direction = from_multiapp
    variable = vt
    postprocessor = average_v
  []
[]
```

E:/GitHub/moose/tutorials/tutorial02_multiapps/step03_coupling/01_sub.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [v]
  []
[]

```

```
[AuxVariables]
  [ut]
  []
[]

[Kernels]
  [diff]
    type = Diffusion
    variable = v
  []
  [force]
    type = CoupledForce
    variable = v
    v = ut
    coef = 100
  []
  [td]
    type = TimeDerivative
    variable = v
  []
[]

[BCs]
  [left]
    type = DirichletBC
    variable = v
    boundary = left
    value = 0
  []
  [right]
    type = DirichletBC
    variable = v
    boundary = right
    value = 1
  []
[]

[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

```

```
[Outputs]
  exodus = true
[]

[Postprocessors]
  [average_v]
    type = ElementAverageValue
    variable = v
  []
[]
```

2.3.4.1　Picard

松耦合虽然简单且稳定，但可能并不准确。由于每个时间步仅交换一次数据，因此两个应用程序中的 solution 很可能彼此不处于平衡状态。随着时间步长变大，情况尤其如此。要解决此问题，您可以在应用程序之间来回迭代，直到达到“静止点”。在 MOOSE 中，我们将此称为 Picard 迭代，尽管它还有许多其他名称，包括“紧密耦合”。要在 MultiApps 中获得这种行为，所需要做的就是将 Master 的 Executioner 块中的 $picard_max_its$ 设置为大于 1 的值。请注意，您可以在大型 MultiApp 层次结构中的任何时候执行此操作！一个警告：为了使其工作，两个应用程序都需要具有备份/恢复功能。所有基于 MOOSE 的应用程序都已具备此功能，但 MOOSE 封装的应用程序需要做一些工作。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step03_coupling/02_master_picard.i

```
[Executioner]
  type = Transient
  end_time = 2
  dt = 0.2

  picard_max_its = 10

  nl_abs_tol = 1e-10
  picard_rel_tol = 1e-6
  picard_abs_tol = 1e-10

  solve_type = 'PJFNK'

  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]
```

2.3.4.2 Picard With Subcycling

由于 MOOSE 中的高级备份/恢复功能，即使在两个（或更多!）应用程序不使用相同时间步长的情况下，我们实际上也有能力进行 Picard 迭代。这很有用的一种情况是子循环。这可以让您对一种物理（例如固体力学）采用更大的时间步长，而对另一种物理（CFD）采用更小的时间步长，同时仍能在两者之间找到一个静止点。这里的重要一点是，应用程序需要将其状态恢复到主应用程序试图采取的当前时间步长的开始时间。

同样的问题再次解决了 - 但现在子应用程序正在采取更小的时间步长并且它 sub_cycling。

E:/GitHub/moose/tutorials/tutorial02_multiapps/step03_coupling/03_master_subcycling_picard.i

```
[MultiApps]
  [micro]
    type = TransientMultiApp
    positions = '0.15 0.15 0  0.45 0.45 0  0.75 0.75 0'
    input_files = '03_sub_subcycling_picard.i'
    execute_on = timestep_end
    output_in_position = true
    sub_cycling = true
  []
[]
```

由于 MOOSE 中的高级备份/恢复功能，即使在两个（或更多!）应用程序不使用相同时间步长的情况下，我们实际上也有能力进行 Picard 迭代。这很有用的一种情况是子循环。

2.4 Phase Field

2.4.1 基础相场方程

2.4.1.1 ACInterface

若界面迁移率 L 是任何变量的函数，如温度 T, 那么 $L(\eta, arg)$,

$$\left(\kappa_i \nabla \eta_i, \nabla\left(L_i \psi_m\right)\right)=\nabla \eta_i \cdot \kappa_i \nabla\left(L_i \psi_m\right)=\nabla \eta_i \cdot \kappa_i\left(\nabla L_i \cdot \psi_m+L_i \cdot \nabla \psi_m\right) \tag{2.1}$$

$$\nabla L(\eta, args)=\nabla \eta \cdot \frac{\partial L}{\partial \eta}+\nabla args \cdot \frac{\partial L}{\partial args} \tag{2.2}$$

2.4.2　ICs:EBSD

要读取实验电子背散射衍射 (EBSD) 数据，需要三个组件：EBSDMesh Mesh object；EBSDReader UserObjcet；Initial conditions (such as the ReconVarIC action provides)

采用耦合模型来创建具有应力的初始微观结构，

Listing 1 EBSD_reconstruction_grain_growth_mech.i

```
# This example reconstructs the grain structure from an EBSD data file
# Then, an isotropic grain model is run with linear elasticity and an anisotropic
# elasticity tensor that uses the measured EBSD angles.
[Mesh]
  [ebsd_mesh]
    type = EBSDMeshGenerator
    uniform_refine = 2 #Mesh can go two levels coarser than the EBSD grid
    filename = IN100_128x128.txt
  []
[]

[GlobalParams]
  op_num = 8
  var_name_base = gr
  displacements = 'disp_x disp_y'
[]

[Variables]
  [PolycrystalVariables] #Polycrystal variable generation (30 order parameters)
  []
  [disp_x]
  []
  [disp_y]
  []
[]

[AuxVariables]
  [bnds]
  []
  [gt_indices]
    order = CONSTANT
    family = MONOMIAL
  []
  [unique_grains]
    order = CONSTANT
    family = MONOMIAL
  []
  [vonmises_stress]
```

```
    order = CONSTANT
    family = MONOMIAL
  []
  [C1111]
    order = CONSTANT
    family = MONOMIAL
  []
  [phi1]
    order = CONSTANT
    family = MONOMIAL
  []
  [Phi]
    order = CONSTANT
    family = MONOMIAL
  []
  [phi2]
    order = CONSTANT
    family = MONOMIAL
  []
  [EBSD_grain]
    family = MONOMIAL
    order = CONSTANT
  []
[]

[ICs]
  [PolycrystalICs]
    [ReconVarIC]
      ebsd_reader = ebsd
      coloring_algorithm = bt
    []
  []
[]

[Kernels]
  [PolycrystalKernel]
  []
  [PolycrystalElasticDrivingForce]
  []
  [TensorMechanics]
  []
[]

[AuxKernels]
  [BndsCalc]
    type = BndsCalcAux
    variable = bnds
    execute_on = 'initial timestep_end'
```

```
[]
[gt_indices]
  type = FeatureFloodCountAux
  variable = gt_indices
  execute_on = 'initial timestep_end'
  flood_counter = grain_tracker
  field_display = VARIABLE_COLORING
[]
[unique_grains]
  type = FeatureFloodCountAux
  variable = unique_grains
  execute_on = 'initial timestep_end'
  flood_counter = grain_tracker
  field_display = UNIQUE_REGION
[]
[C1111]
  type = RankFourAux
  variable = C1111
  rank_four_tensor = elasticity_tensor
  index_l = 0
  index_j = 0
  index_k = 0
  index_i = 0
  execute_on = timestep_end
[]
[vonmises_stress]
  type = RankTwoScalarAux
  variable = vonmises_stress
  rank_two_tensor = stress
  scalar_type = VonMisesStress
  execute_on = timestep_end
[]
[phi1]
  type = OutputEulerAngles
  variable = phi1
  euler_angle_provider = ebsd
  grain_tracker = grain_tracker
  output_euler_angle = 'phi1'
  execute_on = 'initial'
[]
[Phi]
  type = OutputEulerAngles
  variable = Phi
  euler_angle_provider = ebsd
  grain_tracker = grain_tracker
  output_euler_angle = 'Phi'
  execute_on = 'initial'
[]
```

```
  [phi2]
    type = OutputEulerAngles
    variable = phi2
    euler_angle_provider = ebsd
    grain_tracker = grain_tracker
    output_euler_angle = 'phi2'
    execute_on = 'initial'
  []
  [grain_aux]
    type = EBSDReaderPointDataAux
    variable = EBSD_grain
    ebsd_reader = ebsd
    data_name = 'feature_id'
    execute_on = 'initial'
  []
[]

[BCs]
  [top_displacement]
    type = DirichletBC
    variable = disp_y
    boundary = top
    value = -2.0
  []
  [x_anchor]
    type = DirichletBC
    variable = disp_x
    boundary = 'left right'
    value = 0.0
  []
  [y_anchor]
    type = DirichletBC
    variable = disp_y
    boundary = bottom
    value = 0.0
  []
[]

[Modules]
  [PhaseField]
    [EulerAngles2RGB]
      crystal_structure = cubic
      euler_angle_provider = ebsd
      grain_tracker = grain_tracker
    []
  []
[]
```

```
[Materials]
  [Copper]
    # T = 500 # K
    type = GBEvolution
    block = 0
    T = 500
    wGB = 0.6 # um
    GBmob0 = 2.5e-6 # m^4/(Js) from Schoenfelder 1997
    Q = 0.23 # Migration energy in eV
    GBenergy = 0.708 # GB energy in J/m^2
    molar_volume = 7.11e-6; # Molar volume in m^3/mol
    length_scale = 1.0e-6
    time_scale = 1.0e-6
  []
  [ElasticityTensor]
    type = ComputePolycrystalElasticityTensor
    grain_tracker = grain_tracker
  []
  [strain]
    type = ComputeSmallStrain
    block = 0
    displacements = 'disp_x disp_y'
  []
  [stress]
    type = ComputeLinearElasticStress
    block = 0
  []
[]

[Postprocessors]
  [dt]
    type = TimestepSize
  []
  [n_elements]
    type = NumElems
    execute_on = 'initial timestep_end'
  []
  [n_nodes]
    type = NumNodes
    execute_on = 'initial timestep_end'
  []
  [DOFs]
    type = NumDOFs
  []
[]

[UserObjects]
  [ebsd]
```

```
    type = EBSDReader
  []
  [grain_tracker]
    type = GrainTrackerElasticity
    compute_var_to_feature_map = true
    ebsd_reader = ebsd

    fill_method = symmetric9
    C_ijkl = '1.27e5 0.708e5 0.708e5 1.27e5 0.708e5 1.27e5 0.7355e5 0.7355e5 0.7355e5'
    euler_angle_provider = ebsd
  []
[]

[Executioner]
  type = Transient
  scheme = bdf2
  solve_type = PJFNK
  petsc_options_iname = '-pc_type -pc_hypre_type -pc_hypre_boomeramg_strong_threshold'
  petsc_options_value = '  hypre    boomeramg                      0.7'
  l_tol = 1.0e-4
  l_max_its = 20
  nl_max_its = 20
  nl_rel_tol = 1.0e-8
  start_time = 0.0
  num_steps = 30
  dt = 10
  [Adaptivity]
    initial_adaptivity = 0
    refine_fraction = 0.7
    coarsen_fraction = 0.1
    max_h_level = 2
  []
  [TimeStepper]
    type = IterationAdaptiveDT
    cutback_factor = 0.9
    dt = 10.0
    growth_factor = 1.1
    optimal_iterations = 7
  []
[]

[Outputs]
  csv = true
  exodus = true
[]
```

运行出错，报错如下:

Listing 2 EBSD_reconstruction_grain_growth_mech.log

```
*** ERROR ***
/home/pw-moose/projects/panda/grain_growth/ebsd/EBSD_reconstruction_grain_growth_mech.i:66:
    section 'ICs/PolycrystalICs/ReconVarIC' does not have an associated "Action".
Did you misspell it?
```

使用 EBSDReader 重建微结构，使用 phase_field 模块示例文件 IN100-111grn.i 创建，

2.5　枝晶生长

2.5.1　雪花状枝晶生长

1. 输入文件：snow.i
2. 参考文献：https://www.sci-hub.ren/10.1016/0167-2789(93)90120-p
3. 代码未学习

2.6　晶粒长大理论

提出了几种相场模型来模拟多晶材料的晶粒生长动力学。Chen 和 Yang 的模型是最早和应用最广泛的模型之一，其中不同晶向的晶粒由一组非守恒序参数场表示 [2]。该模型已用于二维 [2] 和三维 [6] 相场模拟晶粒生长。Darvishi Kamachali 等 [10] 提出了另一类多相场模型，对序参数进行了约束，使得给定点的所有序参数之和趋于统一。对这一约束的物理解释是，序参量表示不同取向晶粒的体积分数。Warren 及其同事 [11，12] 提出了一个二阶参数模型，其中表示晶体的有序性和前主导的局部取向。

2.6.1　多序参数理论模型

本节采用 Chen 等 [2], Moelans 等 [11]提出的相场模型进行晶粒生长，并利用 Tonks 等 [12]的耦合方法来将弹性能嵌入相场范式中。在该模型中，多晶材料的每个晶粒由一个非守恒场变量，即序参数 η_i 表示，该参数在晶粒内部取值为 1，在其他晶粒中取值为 0。晶界由它们的中间值 $0 < \eta_i < 1$ 表示。相场模型包含两个基本要素：适当的自由能函数和场变量的时间演化函数。当同时考虑曲率驱动和弹性应变驱动的晶粒生长时，包含 N 晶粒的多晶材料的整个系统的自由能

可以写为，

$$F = \int_V \left[f_{loc}\left(\eta_1, \ldots, \eta_N\right) + f_{gr}\left(\eta_1, \ldots, \eta_N\right) + E_d \right] dV \tag{2.3}$$

其中 V 代表系统的体积，f_{loc}, f_{gr}, E_d 分别代表整个系统的局部自由能密度、梯度能量密度和外加能源。每个能量密度项将在下面进一步详述。根据基于自由能泛函的演化方程，微观结构演化由序参数演化表示。

Eqs.(2.13) 中的局部能量密度 f_{loc} 为晶粒填充模拟域提供了驱动力，同时惩罚了同一位置多个序参数的非零值，其具体表达式为，

$$f_{loc} = \mu \left(\sum_{i=1}^{N} \left(\frac{\eta_i^4}{4} - \frac{\eta_i^2}{2} \right) + \gamma \sum_{i=1}^{N} \sum_{j>i}^{N} \eta_i^2 \eta_j^2 \right) + \frac{1}{4} \tag{2.4}$$

其中 μ 和 γ 是模型参数，它们通过使用 Moelans 等 [11] 中提到的表达式与晶界属性相关联。梯度能量密度 $f_{gr}(\eta_1, \ldots, \eta_N)$ 定义如下，

$$f_{gr} = \sum_{i}^{N} \frac{\kappa_i}{2} \left| \nabla \eta_i \right|^2 \tag{2.5}$$

其中 κ_i 是梯度能量系数。不同域之间的梯度直接揭示了晶界的位置，这也是相场法与锐界面法相比的一大优势。E_d 为系统提供额外的能量来源，如变形能、静电能、磁能等。为了考虑由弹性变形引起的异常晶粒长大，通过以下方程将弹性能量密度视为附加自由能函数 E_d，

$$\begin{aligned} E_d &= \frac{1}{2} \boldsymbol{\sigma} : \boldsymbol{\epsilon} \\ &= \frac{1}{2} \mathcal{C} : \boldsymbol{\epsilon} \cdot \boldsymbol{\epsilon} \end{aligned} \tag{2.6}$$

使用 Tonks 等 [12] 中的插值方法，弹性张量 $\mathcal{C}$ 是序参数的函数，

$$\mathcal{C}\left(\eta_1, \ldots, \eta_N\right) = \frac{\sum_i^N h\left(\eta_i\right) C_i}{\sum_i^N h\left(\eta_i\right)} \tag{2.7}$$

其中，C_i 是晶体取向矩阵 $\mathbf{R}_{gr}$ 旋转后得到的每个晶粒的弹性张量，其中旋转矩阵由三个欧拉定义角度 $[\theta, \varphi, \psi]_i$ 最初赋予每个颗粒，它们之间的关系将在后面讨论。此外，插值函数 $h\left(\eta_i\right)$ 对于阶参数 $\eta_i = 0$ 等于 0 和 $\eta_i = 1$ 的 1，表达式为

$$h\left(\eta_i\right) = \frac{1}{2}\left(1 + \sin\left(\pi\left(\left(\eta_i - 0.5\right)\right)\right)\right) \tag{2.8}$$

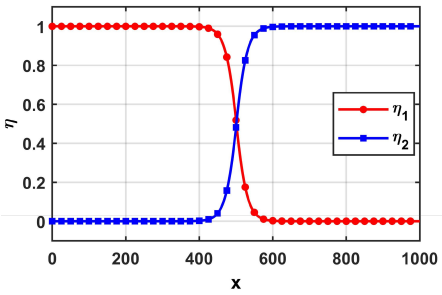

图 2.6 插值函数

Figure 2.6 Interpolation function

序参数遵循 Allen-Cahn 方程，也称为时间相关的 Ginzburg-Landau 方程 [13]，

$$\frac{\partial \eta_i}{\partial t} = -L\frac{\delta F}{\delta \eta_i} \tag{2.9}$$

其中 t 是时间，L 是序参数的迁移率。通过结合 Eqs.(2.3)、Eqs.(2.13)、Eqs.(2.5) 和 Eqs.(2.9)，序参数 η_i 的演化方程为，

$$\frac{\partial \eta_i}{\partial t} = -L\left(\mu\left(\eta_i^3 - \eta_i + 2\gamma\eta_i\sum_{j\neq i}^{N}\eta_j^2\right) - \kappa_i\nabla^2\eta_i + \frac{\partial E_d}{\partial \eta_i}\right) \tag{2.10}$$

最后，选择模型参数 μ、γ、κ_i 和 L 以重现所需的晶界能量和晶界迁移率。根据 Moelans 等 [11]，这些参数通过以下等式由边界能量 σ_{GB}、晶界迁移率 m_{GB} 和扩散晶界宽度 ω_{GB} 决定，

$$\begin{aligned} \gamma &= 1.5 \\ \kappa_i &\approx \frac{3}{4}\sigma_{\mathrm{GB}}\omega_{\mathrm{GB}} \\ L &\approx \frac{4}{3}\frac{m_{\mathrm{GB}}}{\omega_{\mathrm{GB}}} \\ \mu &\approx \frac{3}{4}\frac{1}{f_{0,\,\mathrm{saddle}}(\gamma)}\frac{\sigma_{\mathrm{GB}}}{\omega_{\mathrm{GB}}} \end{aligned} \quad \ldots (2.11)$$

其中 $f_{0,\mathrm{saddle}}(\gamma)$ 是系统在鞍点处的局部自由能。参考文献 [11] 中描述了参数化方法的详细信息。

注意，本节采用纯铜作为模拟材料，其中晶界能量和晶粒迁移率由 Schönfelder 等 [14]使用分子动力学方法确定，即 $\sigma_{\mathrm{GB}} = 0.708\ J/m^2$ 和，

$$m = m_0 \exp\left(\frac{-Q}{k_{\mathrm{B}}T}\right) \tag{2.12}$$

其中 k_{B} 是 Boltzmann 常数，本次模拟中的绝对温度 T 为 $500K$，晶界迁移率前置因子 $m_0 = 2.5\times10^{-6}\ \mathrm{m}^4/(\mathrm{J}\cdot\mathrm{s})$ 和晶界迁移激活能 $Q = 0.23\ \mathrm{eV}$。

力学参数，即未旋转系统中的弹性模量 C_0 是模型参数的另一部分。由于纯铜具有面心立方 (FCC) 结构，弹性张量是各向异性的 (C_{12}, C_{44}) [14]，表示为：

它是相场变量及其梯度的函数。假设摩尔体积为常数，系统处于热平衡状态。能量梯度系数严格为正，因此相场变量的梯度对系统自由能的贡献为正。其中 m 是模型参数。κ 是能量梯度系数。提出了一个形式为 mf_0 的均匀自由能，

$$f_{loc} = \mu \left(\sum_{i}^{N} \left(\frac{\eta_i^4}{4} - \frac{\eta_i^2}{2} \right) + \gamma \sum_{i=1}^{N} \sum_{j>i}^{N} \eta_i^2 \eta_j^2 \right) + \frac{1}{4} \tag{2.13}$$

$$\begin{aligned} C_{11} &= 181.50 - 0.109T \text{ GPa} \\ C_{12} &= 101.80 - 0.062T \text{ GPa} \\ C_{44} &= 101.80 - 0.057T \text{ GPa} \end{aligned} \quad \ldots (2.14)$$

$$C_0 = \begin{pmatrix} C_{11} & C_{12} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{11} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{12} & C_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{44} \end{pmatrix} \tag{2.15}$$

接下来，由三个欧拉角 $[\theta, \varphi, \psi]_i$ 定义的旋转矩阵 $\mathbf{R}_{gr}$，用于描述从局部晶格配置 C_0 到当前配置的刚性晶格旋转，并表示为，

$$\mathbf{R}_{gr} = \mathbf{R}_\theta \mathbf{R}_\varphi \mathbf{R}_\psi \tag{2.16}$$

正交向量空间中的旋转张量 $\mathbf{R}_\theta$、$\mathbf{R}_\varphi$ 和 $\mathbf{R}_\psi$ 的矩阵表达式为，

$$
\begin{aligned}
\left(\mathbf{R}_\theta\right)_{ij} &= \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \\
\left(\mathbf{R}_\varphi\right)_{ij} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix}, \\
\left(\mathbf{R}_\psi\right)_{ij} &= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned}
\tag{2.17}
$$

每个晶粒的弹性张量 C_i s 定义为，

$$
C_i = \left(\mathbf{R}_{gr} \otimes \mathbf{R}_{gr}\right) C_0 \left(\mathbf{R}_{gr} \otimes \mathbf{R}_{gr}\right)^T \tag{2.18}
$$

在二维铜多晶中假设了各向同性晶界特性。因此，每个晶界的行为由等式 (2.11) 中的晶界能量 σ_{GB} 和晶界迁移率 m_{GB} 决定。此模拟中使用的所有参数都显示在 Table 2.1 中。

表 2.1 纯铜晶粒生长模拟的材料参数。

Table 2.1 Material parameters used in the grain growth simulation of pure copper.

Symbol	Description	Units	Value
C_{11}	elastic stiffness constant	GPa	127.0
C_{12}	elastic stiffness constant	GPa	70.8
C_{44}	elastic stiffness constant	GPa	73.55
σ_{GB}	GB energy	J/m^2	0.708
m_0	GB mobility prefactor	$\mathrm{m}^4/(\mathrm{J}\cdot\mathrm{s})$	2.5×10^{-6}
Q	GB migration activation energy	eV	0.23
k_{B}	Boltzmann constant	eV/K	8.617×10^{-5}
T	the absolute temperature	K	500
m_{GB}	GB mobility	$\mathrm{m}^4/(\mathrm{J}\cdot\mathrm{s})$	1.2×10^{-8}
ω_{GB}	diffusion GB width	m	1.5×10^{-8}
γ	model parameter	–	1.5
κ_i	gradient energy coefficient	J/m	7.965×10^{-9}
L	the order parameter mobility	$\mathrm{m}^4/(\mathrm{J}\cdot\mathrm{s})$	1.069
$f_{0,\,\mathrm{saddle}}(\gamma)$	GB energy constant	–	0.125
μ	model coefficient	J/m^3	9.2×10^7

2.7　晶粒生长建模

本节基于 poly_grain_growth_2D_eldrforce.i 来解构界面能-弹性能耦合的晶粒长大相场模型。

2.7.1　Kernels

Variables 模块设置了三类变量，分别为 $\{\eta_i\}, x, y$，具体代码如下：

Listing 3 poly_grain_growth_2D_eldrforce.i

```
[Variables]
  [./PolycrystalVariables] # ${\eta_i}$
    # PolycrystalVariablesAction
      # var_name_base = gr
      # op_num = 8
  [../]
  [./disp_x] # $x$
    order = FIRST
    family = LAGRANGE
  [../]
  [./disp_y] # $y$
    order = FIRST
    family = LAGRANGE
  [../]
[]
```

Kernels 主要分三块：第一块是局部自由能和梯度能的弱形式-PolycrystalKernelAction；第二块是弹性能的弱形式-PolycrystalElasticDrivingForceAtion；第三块是张量力学部分，即力学平衡方程-TensorMechanics。

Listing 4 poly_grain_growth_2D_eldrforce.i

```
[Kernels]
  [./PolycrystalKernel]
    # PolycrystalKernelAction -- Set up ACGrGrPoly, ACInterface, TimeDerivative, and
    ACGBPoly kernels
    # ACGBPoly' is used only when there are bubbles in the grain-growth simulations.
      # var_name_base = gr
      # op_num = 8
    # ndef = 0 # specifies the number of deformed grains to create
    # use_displaced_mesh = false # Whether to use displaced mesh in the kernels
  [../]
  [./PolycrystalElasticDrivingForce]
    # PolycrystalElasticDrivingForceAction
    # var_name_base = gr
    # op_num = 8
```

```
    # use_displaced_mesh = false
  [../]
  [./TensorMechanics]
    # registerSyntax("TensorMechanicsAction", "Modules/TensorMechanics/Master/*");
    use_displaced_mesh = true
    displacements = 'disp_x disp_y'
  [../]
[]
```

首先 PolycrystalKernelAction 和 PolycrystalElasticDrivingForceAction 批量产生 op_num = 8 个的 ACGrGrPoly, ACInterface, TimeDerivative 和 ACGrGrElastic-DrivingForce kernels，其适用于 MOOSE 开发的内积形式的有限元弱形式如下图所示，

$$R_{\eta_i} = \underbrace{\left(\frac{d\eta_i}{\partial t}, \psi_m\right)}_{\substack{\text{TimeDerivative}\\ \text{IE\_gr0,}\\ \text{IE\_gr1,}\\ \text{IE\_gr\#}}} + \underbrace{(\nabla(k_i\eta_i), \nabla(L\psi_m))}_{\substack{\text{ACInterface}\\ \text{ACInt\_gr0,}\\ \text{ACInt\_gr1,}\\ \text{ACInt\_gr\#}}} + \underbrace{L\left(\frac{\partial f_{loc}}{\partial \eta_i}, \psi_m\right)}_{\substack{\text{ACGrGrPoly}\\ \text{ACBulk\_gr0,}\\ \text{ACBulk\_gr1,}\\ \text{ACBulk\_gr\#}}} + \underbrace{\quad ? \quad}_{\text{ACGBPoly}}$$

$$+ \underbrace{L\left(\frac{\partial E_d}{\partial \eta_i}, \psi_m\right)}_{\substack{\text{ACGrGrElasticDrivingForce}\\ \text{AC\_ElasticDrivingForce\_gr0,}\\ \text{AC\_ElasticDrivingForce \_gr1,}\\ \text{AC\_ElasticDrivingForce \_gr\#}}}$$

图 2.7 总自由能泛函的弱形式

Figure 2.7 The weak form of the total free energy functional

至于 TensorMechanics 主要用于求解给定本构模型的情况下，求解准固体力学的平衡方程。

2.7.2 Materials

本节中材料模块主要为控制偏微分方程组提供必要的模型参数，主要分两块：第一，晶粒长大模型的模型参数；第二，线弹性模型相关的模型参数。

首先是负责纯曲率驱动晶粒长大的参数部分，这些参数被用来复现真实的晶界情况，输入的参数有温度，晶界宽度，参考晶界迁移率，迁移能，晶界能；输出的参数有该温度下晶界迁移率，序参数的迁移率，梯度能量系数，μ 等 。

Listing 5 poly_grain_growth_2D_eldrforce.i

```
[Materials]
```

```
  [./Copper]
    type = GBEvolution
    block = 0
    T = 500 # Kelvin
    wGB = 15 # nm
    GBmob0 = 2.5e-6 # m^4/(Js) from Schoenfelder 1997
    Q = 0.23 # Migration energy in eV
    GBenergy = 0.708 # GB energy in J/m^2
    # length_scale = 1.0e-9 # m
    # time_scale = 1.0e-9 # s
    # molar_volume = 24.62e-6 # m^3/mol, needed for temperature gradient driving force
    # output: _M_GB[_qp], _L[_qp], _kappa[_qp], _gamma[_qp], _mu[_qp], _act_wGB[_qp]
  [../]
```

之后是线弹性本构方程一块的材料参数，主要是输出梯度化之后的弹性张量,以及获取每个正交点的弹性形变之后,通过本构方程计算所得的弹性应力。其中 ComputePolycrystalElasticityTensor 负责将旋转之后的弹性张量（grain_tracker）梯度化处理，并求解弹性张量关于序参量的导数，用于输入到 PolycrystalElastic-Driving 中用于耦合弹性形变能驱动晶粒长大。

Listing 6 poly_grain_growth_2D_eldrforce.i

```
  [./ElasticityTensor]
    type = ComputePolycrystalElasticityTensor
    grain_tracker = grain_tracker # 输入旋转之后，但没有梯度化处理的弹性张量，四阶张量
    # length_scale = 1.0e-9 # s
    # pressure_scale = 1.0e6 # Pa
  [../]
  [./strain]
    type = ComputeSmallStrain
    block = 0
    displacements = 'disp_x disp_y'
  [../]
  [./stress]
    type = ComputeLinearElasticStress
    block = 0
  [../]
[]
```

2.7.3　UserObjects

……

2.8 3D 晶粒生长

问题

1. 运行 3D_6000_gr.i，晶粒数目 6000，111 服务器直接死机；

2. 运行 3D_6000_gr01.i，缩小原始模型为原来的 1(8)2，关闭网格自适应，出现 3D_6000_gr01.log 显示的错误；

3. 运行 3D_6000_gr02.i，缩小原始模型为原来的 1(8)2，打开网格自适应，出现 3D_6000_gr01.log 显示的错误；

参考资料

1. Grain growth simulation stuck at Nonlinear |R| step

附录 A　Matlba 脚本

A.1　曲线图绘制脚本

Listing 7 Matlab example

```
%% DESCRIPTION OF THE CODE
% Title: 凝固函数可视化
% Author: 土豆三撇
% Data: 2021.09.07
% Email: 1248110286@qq.com

n = 2;
phi = linspace(-1*n,n,1000);

num_FontSize_label = 15;
num_FontSize_legend = 14

for i = 1:6
    deltaF = i;
    fLocal = 4*deltaF.*(-1/2.*phi.^2+1/4.*phi.^4);
    plot(phi,fLocal,...
        'LineWidth',1.5,...
        'DisplayName',num2str(i));
    hold on
end
hold off
xlabel('\eta',...
  'FontSize',num_FontSize_label,...
  'FontWeight','bold',...
  'Color','k');
ylabel('f(\eta)',...
      'FontSize',num_FontSize_label,...
      'FontWeight','bold',...
      'Color','k');
set(gca,'FontSize',num_FontSize_label,'Fontwei','Bold','Linewidth',1.5);
lgd = legend('FontSize',num_FontSize_legend,'TextColor','black','Location','north','
    NumColumns',2);

% ylim([-6 6]);
xlim([-1.5,1.5]);

hfig = figure(1);
figWidth = 14.5;
figHight = 8.5;

```

```
set(hfig,'PaperUnits','centimeters');
set(hfig,'PaperPosition',[0 0 figWidth figHight])

print(hfig,'double_well_potential1','-r400','-dpng')
```

附录 B　moose 代码

B.1　Input 文件

B.1.1　晶粒长大

Listing 8 poly_grain_growth_2D_eldrforce.i

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 20
  ny = 20
  nz = 0
  xmax = 1000
  ymax = 1000
  zmax = 0
  elem_type = QUAD4
  uniform_refine = 2
[]

[GlobalParams]
  op_num = 8
  var_name_base = gr
  grain_num = 36
[]

[Variables]
  [./PolycrystalVariables] # ${\eta_i}$
    # PolycrystalVariablesAction
      # var_name_base = gr
      # op_num = 8
  [../]
  [./disp_x] # $x$
    order = FIRST
    family = LAGRANGE
  [../]
  [./disp_y] # $y$
    order = FIRST
    family = LAGRANGE
  [../]
[]

[Kernels]
  [./PolycrystalKernel]
    # PolycrystalKernelAction -- Set up ACGrGrPoly, ACInterface, TimeDerivative, and
    ACGBPoly kernels
```

```
    # ACGBPoly' is used only when there are bubbles in the grain-growth simulations.
      # var_name_base = gr
      # op_num = 8
    # ndef = 0 # specifies the number of deformed grains to create
    # use_displaced_mesh = false # Whether to use displaced mesh in the kernels
  [../]
  [./PolycrystalElasticDrivingForce]
    # PolycrystalElasticDrivingForceAction
    # var_name_base = gr
    # op_num = 8
    # use_displaced_mesh = false
  [../]
  [./TensorMechanics]
    # registerSyntax("TensorMechanicsAction", "Modules/TensorMechanics/Master/*");
    use_displaced_mesh = true
    displacements = 'disp_x disp_y'
  [../]
[]

[Materials]
  [./Copper]
    type = GBEvolution
    block = 0
    T = 500 # Kelvin
    wGB = 15 # nm
    GBmob0 = 2.5e-6 # m^4/(Js) from Schoenfelder 1997
    Q = 0.23 # Migration energy in eV
    GBenergy = 0.708 # GB energy in J/m^2
    # length_scale = 1.0e-9 # m
    # time_scale = 1.0e-9 # s
    # molar_volume = 24.62e-6 # m^3/mol, needed for temperature gradient driving force
    # output: _M_GB[_qp], _L[_qp], _kappa[_qp], _gamma[_qp], _mu[_qp], _act_wGB[_qp]
  [../]
  [./ElasticityTensor]
    type = ComputePolycrystalElasticityTensor
    grain_tracker = grain_tracker # 输入旋转之后，但没有梯度化处理的弹性张量，四阶张量
    # length_scale = 1.0e-9 # s
    # pressure_scale = 1.0e6 # Pa
  [../]
  [./strain]
    type = ComputeSmallStrain
    block = 0
    displacements = 'disp_x disp_y'
  [../]
  [./stress]
    type = ComputeLinearElasticStress
    block = 0
  [../]
```

```
[]

[UserObjects]
  [./euler_angle_file]
    type = EulerAngleFileReader
    file_name = grn_36_rand_2D.tex
  [../]
  [./voronoi]
    type = PolycrystalVoronoi
    coloring_algorithm = bt
  [../]
  [./grain_tracker]
    type = GrainTrackerElasticity
    threshold = 0.2
    compute_var_to_feature_map = true
    execute_on = 'initial timestep_begin'
    flood_entity_type = ELEMENTAL

    C_ijkl = '1.27e5 0.708e5 0.708e5 1.27e5 0.708e5 1.27e5 0.7355e5 0.7355e5 0.7355e5'
    fill_method = symmetric9
    euler_angle_provider = euler_angle_file
  [../]
[]

[ICs]
  [./PolycrystalICs]
    [./PolycrystalColoringIC]
      polycrystal_ic_uo = voronoi
    [../]
  [../]
[]

[AuxVariables]
  [./bnds]
    order = FIRST
    family = LAGRANGE
  [../]
  [./elastic_strain11]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./elastic_strain22]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./elastic_strain12]
    order = CONSTANT
    family = MONOMIAL
```

```
  [../]
  [./unique_grains]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./var_indices]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./vonmises_stress]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./C1111]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./euler_angle]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]

[AuxKernels]
  [./BndsCalc]
    type = BndsCalcAux
    variable = bnds
    execute_on = timestep_end
  [../]
  [./elastic_strain11]
    type = RankTwoAux
    variable = elastic_strain11
    rank_two_tensor = elastic_strain
    index_i = 0
    index_j = 0
    execute_on = timestep_end
  [../]
  [./elastic_strain22]
    type = RankTwoAux
    variable = elastic_strain22
    rank_two_tensor = elastic_strain
    index_i = 1
    index_j = 1
    execute_on = timestep_end
  [../]
  [./elastic_strain12]
    type = RankTwoAux
    variable = elastic_strain12
```

```
    rank_two_tensor = elastic_strain
    index_i = 0
    index_j = 1
    execute_on = timestep_end
  [../]
  [./unique_grains]
    type = FeatureFloodCountAux
    variable = unique_grains
    execute_on = timestep_end
    flood_counter = grain_tracker
    field_display = UNIQUE_REGION
  [../]
  [./var_indices]
    type = FeatureFloodCountAux
    variable = var_indices
    execute_on = timestep_end
    flood_counter = grain_tracker
    field_display = VARIABLE_COLORING
  [../]
  [./C1111]
    type = RankFourAux
    variable = C1111
    rank_four_tensor = elasticity_tensor
    index_l = 0
    index_j = 0
    index_k = 0
    index_i = 0
    execute_on = timestep_end
  [../]
  [./vonmises_stress]
    type = RankTwoScalarAux
    variable = vonmises_stress
    rank_two_tensor = stress
    scalar_type = VonMisesStress
    execute_on = timestep_end
  [../]
  [./euler_angle]
    type = OutputEulerAngles
    variable = euler_angle
    euler_angle_provider = euler_angle_file
    grain_tracker = grain_tracker
    output_euler_angle = 'phi1'
    execute_on = 'initial timestep_end'
  [../]
[]

[BCs]
  [./Periodic]
```

```
    [./All]
      auto_direction = 'x'
      variable = 'gr0 gr1 gr2 gr3 gr4 gr5 gr6 gr7'
    [../]
  [../]
  [./top_displacement]
    type = DirichletBC
    variable = disp_y
    boundary = top
    value = -50.0
  [../]
  [./x_anchor]
    type = DirichletBC
    variable = disp_x
    boundary = 'left right'
    value = 0.0
  [../]
  [./y_anchor]
    type = DirichletBC
    variable = disp_y
    boundary = bottom
    value = 0.0
  [../]
[]

[Postprocessors]
  [./ngrains]
    type = FeatureFloodCount
    variable = bnds
    threshold = 0.7
  [../]
  [./dofs]
    type = NumDOFs
  [../]
  [./dt]
    type = TimestepSize
  [../]
  [./run_time]
    type = PerfGraphData
    section_name = "Root"
    data_type = total
  [../]
[]

[Preconditioning]
  [./SMP]
    type = SMP
    coupled_groups = 'disp_x,disp_y'
```

```
  [../]
[]

[Executioner]
  type = Transient
  scheme = bdf2
  solve_type = PJFNK
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart -
     pc_hypre_boomeramg_strong_threshold'
  petsc_options_value = 'hypre boomeramg 31 0.7'
  l_tol = 1.0e-4
  l_max_its = 30
  nl_max_its = 25
  nl_rel_tol = 1.0e-7
  start_time = 0.0
  num_steps = 50
  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 1.5
    growth_factor = 1.2
    cutback_factor = 0.8
    optimal_iterations = 8
  [../]
  [./Adaptivity]
    initial_adaptivity = 2
    refine_fraction = 0.8
    coarsen_fraction = 0.05
    max_h_level = 3
  [../]
[]

[Outputs]
  file_base = poly36_grtracker
  exodus = true
[]
```

参考文献

[1] Chen L Q. Phase-Field Models for Microstructure Evolution [J/OL]. Annual Review of Materials Research, 2002, 32: 113-140. DOI: 10.1146/annurev.matsci.32.112001.132041.

[2] Chen L Q, Yang W. Computer simulation of the domain dynamics of a quenched system with a large number of nonconserved order parameters: The grain-growth kinetics [J/OL]. Physical Review B, 1994, 50: 15752-15756. DOI: 10.1103/PhysRevB.50.15752.

[3] Fan D, Chen L Q. Diffuse-interface description of grain boundary motion [J/OL]. Philosophical Magazine Letters, 1997, 75: 187-196. DOI: 10.1080/095008397179615.

[4] Fan D, Geng C, Chen L Q. Computer simulation of topological evolution in 2-D grain growth using a continuum diffuse-interface field model [J/OL]. Acta Materialia, 1997, 45: 1115-1126. DOI: 10.1016/S1359-6454(96)00221-2.

[5] Fan D, Chen L Q. Computer simulation of grain growth using a continuum field model [J/OL]. Acta Materialia, 1997, 45: 611-622. DOI: 10.1016/S1359-6454(96)00200-5.

[6] Krill Iii C E, Chen L Q. Computer simulation of 3-D grain growth using a phase-field model [J/OL]. Acta Materialia, 2002/07/17/, 50: 3059-3075. DOI: 10.1016/S1359-6454(02)00084-8.

[7] Warren J A, Carter W C, Kobayashi R. A phase field model of the impingement of solidifying particles [J/OL]. Physica A: Statistical Mechanics and its Applications, 1998, 261: 159-166. DOI: 10.1016/S0378-4371(98)00381-1.

[8] Warren J A, Kobayashi R, Craig Carter W. Modeling grain boundaries using a phase-field technique [J/OL]. Journal of Crystal Growth, 2000, 211: 18-20. DOI: 10.1016/S0022-0248(99)00856-8.

[9] Lobkovsky A E, Warren J A. Phase-field model of crystal grains [J/OL]. Journal of Crystal Growth, 2001, 225: 282-288. DOI: 10.1016/S0022-0248(01)00867-3.

[10] Darvishi Kamachali R, Steinbach I. 3-D phase-field simulation of grain growth: Topological analysis versus mean-field approximations [J/OL]. Acta Materialia, 2012, 60: 2719-2728. DOI: 10.1016/j.actamat.2012.01.037.

[11] Moelans N, Blanpain B, Wollants P. Quantitative analysis of grain boundary properties in a generalized phase field model for grain growth in anisotropic systems [J/OL]. Physical Review B, 07/16/ 2008, 78: 024113. DOI: 10.1103/PhysRevB.78.024113.

[12] Tonks M, Millett P, Cai W, et al. Analysis of the elastic strain energy driving force for grain boundary migration using phase field simulation [J/OL]. Scripta Materialia, 2010, 63: 1049-1052. DOI: 10.1016/j.scriptamat.2010.07.034.

[13] Cahn J W, Allen S M. A MICROSCOPIC THEORY FOR DOMAIN WALL MOTION AND

ITS EXPERIMENTAL VERIFICATION IN Fe-Al ALLOY DOMAIN GROWTH KINETICS [J/OL]. Le Journal de Physique Colloques, 1977, 38: C7-C7-54. DOI: 10.1051/jphyscol:1977709.

[14] Schönfelder B, Wolf D, Phillpot S R, et al. Molecular-Dynamics Method for the Simulation of Grain-Boundary Migration [J/OL]. Interface Science, 1997, 5: 245-262. DOI: 10.1023/A:1008663804495.