```
import numpy as np
from scipy.fft import fftn, ifftn
import matplotlib.pyplot as plt

def ParentGrains(nx, ny, grainBs):
    dt = 0.1
    nstep = int(1e4)
    M = 1
    Kappa = 1
    nxy = nx * ny

    # Random seed for initialization
    seed = np.random.randint(10, nx-10, size=(grainBs, 2))
    R = 2
    phiB = np.zeros((nx, ny, grainBs))

    # Initialize grains
    for g in range(grainBs):
        x_start = max(0, seed[g, 0]-R)
        x_end = min(nx, seed[g, 0]+R+1)
        y_start = max(0, seed[g, 1]-R)
        y_end = min(ny, seed[g, 1]+R+1)
        phiB[x_start:x_end, y_start:y_end, g] = 1

    kx = 2 * np.pi * np.fft.fftfreq(nx)
    ky = 2 * np.pi * np.fft.fftfreq(ny)
    kx, ky = np.meshgrid(kx, ky, indexing='ij')
    k2 = kx**2 + ky**2

    phiBk = np.zeros_like(phiB, dtype=np.complex128)
    phiplot = np.zeros((nx, ny))
    dfdphiB = np.zeros_like(phiB)
    dfdphiBk = np.zeros_like(phiB, dtype=np.complex128)

    # Color map
    cc = plt.cm.hsv(np.linspace(0, 1, grainBs + 1))
    cc[0, :3] = [0, 0, 0]

    # Evolution steps
    for istep in range(nstep):
        phiB2 = phiB**2
        sumphiB2 = np.sum(phiB2, axis=2)

        for g in range(grainBs):
            phiBk[:, :, g] = fftn(phiB[:, :, g])
            dfdphiB[:, :, g] = -phiB[:, :, g] + phiB[:, :, g]**3 + 3 * phiB[:, :, g] * (sumphiB2 - phiB2[:, :, g
            dfdphiBk[:, :, g] = fftn(dfdphiB[:, :, g])

            phiBk[:, :, g] = (phiBk[:, :, g] - dt * M * dfdphiBk[:, :, g]) / (1 + dt * M * Kappa * k2)
            phiB[:, :, g] = np.real(ifftn(phiBk[:, :, g]))

            # Threshold values
            phiB[:, :, g] = np.clip(phiB[:, :, g], 0, 1)

        totalphiB = np.sum(phiB, axis=2)
        if np.min(totalphiB) >= 0.9995:
            break

    phiBlong = phiB.reshape(-1, grainBs)
    for gp in range(grainBs):
        inrange = (phiBlong[:, gp] == 1)
        iflag = np.arange(grainBs)
        iflag = np.delete(iflag, gp)
        phiBlong[inrange, iflag] = 0

    totalphiBlong = np.sum(phiBlong, axis=1)
    diffphiBlong = np.column_stack((np.arange(nxy), 1 - totalphiBlong))
    diffphiBlong = diffphiBlong[(diffphiBlong[:, 1] != 0) & (diffphiBlong[:, 1] != 1)]

    for r in range(diffphiBlong.shape[0]):
```

```
            loc = int(diffphiBlong[r, 0])
            c = np.argmax(phiBlong[loc, :])
            phiBlong[loc, c] += diffphiBlong[r, 1]

        phiBlong = np.clip(phiBlong, 0, 1)
        phiB = phiBlong.reshape(nx, ny, grainBs)

        for g in range(grainBs):
            phiplot[phiB[:, :, g] > 0.5] = g + 1

        plt.figure(figsize=(8, 8))
        plt.imshow(phiplot, cmap='hsv', origin='lower')
        plt.colorbar()
        plt.axis('equal')
        plt.axis([0, nx, 0, ny])
        plt.savefig('parent_grains.png')

        # Save result
        np.save('BetaMap_16p1028grid.npy', phiB)
        return phiB



    import numpy as np
    from scipy.fftpack import fftn, ifftn

    def solve_elasticity(phiA, StrucphiB, eigen, e1_pl, e2_pl, e3_pl, grainBs, variants,
                         tot_Cpq, hom_Cpq, het_Cpq, gomega, tot_Cpqrs, Spq, Sigmay2, nx, ny, kx, ky, dt, delta_Gm, \

        niter = int(1e2)
        tolerance = 1e-4

        # Eigenstrains
        ei1 = np.zeros((nx, ny))
        ei2 = np.zeros((nx, ny))
        ei3 = np.zeros((nx, ny))

        for g in range(grainBs):
            for v in range(variants):
                ei1 += StrucphiB[:, :, v, g] * eigen[:, :, 0, 0, v, g] * phiA[:, :, v, g]
                ei2 += StrucphiB[:, :, v, g] * eigen[:, :, 1, 1, v, g] * phiA[:, :, v, g]
                ei3 += StrucphiB[:, :, v, g] * eigen[:, :, 0, 1, v, g] * phiA[:, :, v, g]

        ei1 += e1_pl
        ei2 += e2_pl
        ei3 += e3_pl
        ei_devia = (ei1 + ei2) / 2

        # Elastic strain and stress field
        u = np.zeros((nx, ny, 2, niter + 1))
        uk = np.zeros((nx, ny, 2))
        T0 = np.zeros((nx, ny, 3))
        T = np.zeros((nx, ny, 3))
        Tk = np.zeros((nx, ny, 3))
        hom_e = np.zeros(3)
        het_e = np.zeros((nx, ny, 3, niter + 1))
        het_ek = np.zeros((nx, ny, 3))
        s = np.zeros((nx, ny, 3))
        sk = np.zeros((nx, ny, 3))

        # Zeroth-order iteration
        for i in range(3):
            s[:, :, i] = (hom_Cpq[i, 0] * ei1 + hom_Cpq[i, 1] * ei2 + 2 * hom_Cpq[i, 2] * ei3)
            sk[:, :, i] = fftn(s[:, :, i])

        for i in range(2):
            uk[:, :, i] = -1j * (
                gomega[:, :, i] * (sk[:, :, 0] * kx + sk[:, :, 2] * ky) +
                gomega[:, :, 2] * (sk[:, :, 2] * kx + sk[:, :, 1] * ky))
            u[:, :, i, 0] = np.real(ifftn(uk[:, :, i]))
```

```
        het_ek[:, :, 0] = 1j * kx * uk[:, :, 0]
        het_ek[:, :, 1] = 1j * ky * uk[:, :, 1]
        het_ek[:, :, 2] = 0.5 * 1j * (kx * uk[:, :, 1] + ky * uk[:, :, 0])

        for i in range(3):
            het_e[:, :, i, 0] = np.real(ifftn(het_ek[:, :, i]))
            het_e[:, :, i, 0] -= np.trapz(np.trapz(het_e[:, :, i, 0])) / (nx * ny)

        for iter in range(1, niter + 1):
            for i in range(3):
                T[:, :, i] = (
                    T0[:, :, i] - het_Cpq[:, :, i, 0] * het_e[:, :, 0, iter - 1] -
                    het_Cpq[:, :, i, 1] * het_e[:, :, 1, iter - 1] -
                    2 * het_Cpq[:, :, i, 2] * het_e[:, :, 2, iter - 1])
                Tk[:, :, i] = fftn(T[:, :, i])

            for i in range(2):
                uk[:, :, i] = -1j * (
                    gomega[:, :, i] * (Tk[:, :, 0] * kx + Tk[:, :, 2] * ky) +
                    gomega[:, :, 2] * (Tk[:, :, 2] * kx + Tk[:, :, 1] * ky))
                u[:, :, i, iter] = np.real(ifftn(uk[:, :, i]))

            for i in range(3):
                het_ek[:, :, i] = (1j * kx * uk[:, :, 0] if i == 0 else
                                   1j * ky * uk[:, :, 1] if i == 1 else
                                   0.5 * 1j * (kx * uk[:, :, 1] + ky * uk[:, :, 0]))
                het_e[:, :, i, iter] = np.real(ifftn(het_ek[:, :, i]))
                het_e[:, :, i, iter] -= np.trapz(np.trapz(het_e[:, :, i, iter])) / (nx * ny)

            # Convergence check
            diffu2 = (u[:, :, 0, iter] - u[:, :, 0, iter - 1]) ** 2 + (u[:, :, 1, iter] - u[:, :, 1, iter - 1]) ** 2
            conver = np.sqrt(np.trapz(np.trapz(diffu2)))
            if conver < tolerance:
                break

        # Strain and stress calculations
        e11 = hom_e[0] + het_e[:, :, 0, iter] - ei1
        e22 = hom_e[1] + het_e[:, :, 1, iter] - ei2
        e12 = hom_e[2] + het_e[:, :, 2, iter] - ei3

        s11 = tot_Cpq[:, :, 0, 0] * e11 + tot_Cpq[:, :, 0, 1] * e22 + 2 * tot_Cpq[:, :, 0, 2] * e12
        s22 = tot_Cpq[:, :, 1, 0] * e11 + tot_Cpq[:, :, 1, 1] * e22 + 2 * tot_Cpq[:, :, 1, 2] * e12
        s12 = tot_Cpq[:, :, 2, 0] * e11 + tot_Cpq[:, :, 2, 1] * e22 + 2 * tot_Cpq[:, :, 2, 2] * e12

        E_elastic = (s11 * e11 + s22 * e22 + 2 * s12 * e12) * delta_Gm / Vmol

        return E_elastic



    import numpy as np

    def yieldstress(sumphiA2, sigmaA_pl, sigmaB_pl, kappaHP_pl, D_alpha_pl, G_pl, delta_Gm, Vmol,
                    JCa_pl, JCb_pl, JCn_pl, JCm1_pl, JCm2_pl, T, T0, Tr, e1_pl, e2_pl, e3_pl):
        """
        Calculate yield stress and equivalent plastic strain based on modified Hall-Petch and MJC models.
        """
        # Yield stress from modified Hall-Petch function with volume fraction of alpha variant
        sigmay_HP = (sigmaA_pl * sumphiA2 + sigmaB_pl * (1 - sumphiA2) + kappaHP_pl / D_alpha_pl**0.5) * G_pl

        # Deviatoric components of plastic strain
        Dev_e1_pl = e1_pl - (e1_pl + e2_pl) / 2
        Dev_e2_pl = e2_pl - (e1_pl + e2_pl) / 2
        Dev_e3_pl = e3_pl

        # Equivalent plastic strain
        y2 = (Dev_e1_pl**2 + Dev_e2_pl**2 + 2 * Dev_e3_pl**2) / 2
        e0_pl = np.sqrt(4 / 3 * y2)

        # Yield stress from plastic strain hardening based on MJC model
        sigmay_MJC = (JCa_pl + JCb_pl * (1 + JCm1_pl * np.log(T0 / Tr)) * e0_pl**JCn_pl) * (1 - T**JCm2_pl)
```

```
    # Total yield stress
    Sigmay = sigmay_HP + sigmay_MJC
    Sigmay *= 1e06  # Convert to Pa

    Sigmay = Sigmay / (delta_Gm / Vmol) / 1.5
    Sigmay2 = Sigmay**2

    return Sigmay2, e0_pl



import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from PIL import Image

# Initialize
nx = 1024
ny = nx
No = [16, 18, 20, 24, 30, 40]  # Number of selected frames
len_No = len(No)

# Load data
print("Loading BetaMap_16p1028grid.npy...")
data = np.load('BetaMap_16p1028grid.npy', allow_pickle=True)  # Adjust based on file format
phiB = data['phiB']

phiB2 = phiB ** 2
sumphiB2 = np.sum(phiB2, axis=2)
maxsumphiB2 = np.max(sumphiB2)
minsumphiB2 = np.min(sumphiB2)

rho = 0.25
psi = (rho * (maxsumphiB2 - sumphiB2) + (sumphiB2 - minsumphiB2)) / (maxsumphiB2 - minsumphiB2)
inrange = (psi < 0.85)

E_elastic0 = np.zeros((nx, ny))
sumphiAplot20 = np.zeros((nx, ny))
output_Vons0 = np.zeros((nx, ny))

for t in No:
    time_str = f"{t * 0.1004:6.4f}"

    # Load specific files
    E_elastic = np.load(f"01_ElasticEnergy_ttime_{time_str}s.npy")
    sumphiAplot2 = np.load(f"01_sumphiAplot2_ttime_{time_str}s.npy")
    output_Vons = np.load(f"01_VonMisesStress_ttime_{time_str}s.npy")

    # Filenames
    filename1 = f"ElasticEnergy_ttime_{time_str}_GBs.tiff"
    filename2 = f"sumphiAplot2_ttime_{time_str}_GBs.tiff"
    filename3 = f"VonMisesStress_ttime_{time_str}_GBs.tiff"

    # Process Elastic Energy
    maxE = np.max(E_elastic)
    E_elastic0[inrange] = maxE
    E_elastic[inrange] = 0

    plt.figure()
    plt.imshow(E_elastic0, cmap="white", interpolation="none")
    plt.imshow(E_elastic, cmap="jet", alpha=0.75, interpolation="none")
    plt.colorbar()
    plt.title(f"time: {time_str}s")
    plt.axis("equal")
    plt.axis([0, nx, 0, ny])
    plt.savefig(filename1, format='tiff')
    plt.close()

    # Process SumphiAplot2
    maxphiAplot2 = np.max(sumphiAplot2)
```

```
    sumphiAplot20[inrange] = maxphiAplot2
    sumphiAplot2[inrange] = 0

    plt.figure()
    plt.imshow(sumphiAplot20, cmap="white", interpolation="none")
    plt.imshow(sumphiAplot2, cmap="cool", alpha=0.75, interpolation="none")
    plt.colorbar(norm=Normalize(vmin=0, vmax=2))
    plt.title(f"time: {time_str}s")
    plt.axis("equal")
    plt.axis([0, nx, 0, ny])
    plt.savefig(filename2, format='tiff')
    plt.close()

    # Process Von Mises Stress
    maxVons = np.max(output_Vons)
    output_Vons0[inrange] = maxVons
    output_Vons[inrange] = 0

    plt.figure()
    plt.imshow(output_Vons0, cmap="white", interpolation="none")
    plt.imshow(output_Vons, cmap="jet", alpha=0.75, interpolation="none")
    plt.colorbar()
    plt.title(f"time: {time_str}s")
    plt.axis("equal")
    plt.axis([0, nx, 0, ny])
    plt.savefig(filename3, format='tiff')
    plt.close()
```

```
Loading BetaMap_16p1028grid.npy...
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-9-c10123f34e55> in <cell line: 14>()
     12 # Load data
     13 print("Loading BetaMap_16p1028grid.npy...")
---> 14 data = np.load('BetaMap_16p1028grid.npy', allow_pickle=True)  # Adjust based on file format
     15 phiB = data['phiB']
     16

/usr/local/lib/python3.10/dist-packages/numpy/lib/npyio.py in load(file, mmap_mode, allow_pickle, fix_imports, encoding,
max_header_size)
    425             own_fid = False
    426         else:
--> 427             fid = stack.enter_context(open(os_fspath(file), "rb"))
    428             own_fid = True
    429

FileNotFoundError: [Errno 2] No such file or directory: 'BetaMap_16p1028grid.npy'
```

Next steps: Explain error

```
import numpy as np
import matplotlib.pyplot as plt

# == Parent beta grains and alpha variants
grainBs = 16
variants = 2

# == Simulation system
nx = 1024
ny = nx
nxy = nx * ny
lengthx = nx * 2.5e-8  # length in x direction of simulation, unit: m
dx = lengthx / nx  # mesh size, unit: m

# == Simulation parameters
Kb = 1.3806e-23  # Boltzmann constant
T0 = 1075  # temperature
gamma = 50 * 1e-3  # interfacial energy, unit: J/m2
Vmol = 1e-5  # molar volume, unit: m3/mol
thick = 5 * dx  # thickness of interface, unit: m
ttime = 0  # initial time
dt0 = 2e-03  # delta time
M0 = 1.6e-07  # interfacial mobility, unit: J/(m3*s)
nstep = int(1e04)
```

```
nprint1 = int(1e02)  # for loop step and output step

nucleationstep = 450  # nucleation step
ampnoise = 3e-3  # amplitude of noise for nucleation

rho = 0.25  # misfit strain relaxation parameter at grain boundary

kappa0 = 3 * gamma * thick / np.sqrt(8)  # gradient coefficient, unit: J/m

# == Gibbs free energy constant
diffGibbs = -3.1375e02  # driving force, unit: J/mol
g_barrier = 3 * Vmol * gamma / (4 * np.sqrt(2) * thick)  # Gibbs energy barrier, unit: J/mol

a0 = 32 * g_barrier  # Gibbs free energy coefficient
b0 = 3 * a0 - 12 * diffGibbs
c0 = 2 * a0 - 12 * diffGibbs

# == Dimensionless form
delta_Gm = -diffGibbs
delta_M = 1e-18

a = a0 / delta_Gm
b = b0 / delta_Gm
c = c0 / delta_Gm

kappa = kappa0 * Vmol / (delta_Gm * dx**2)
dt = dt0 * delta_Gm * delta_M / dx**2
M = M0 * dx**2 / (delta_M * Vmol)

# == Elastic strain parameters
ang = 2 * np.pi * np.array([0.6978, 0.3171, 0.9502, 0.0344,
                            0.4387, 0.3816, 0.7655, 0.7952,
                            0.1869, 0.4898, 0.4456, 0.6463,
                            0.7094, 0.7547, 0.2760, 0.6797])

e0 = np.zeros((2, 2, 2))
e0[:, :, 0] = [[-0.0490, 0], [0, 0.0670]]  # Transformation strain for 2 variants
e0[:, :, 1] = [[0.0670, 0], [0, -0.0490]]

c11 = 97.7e9 / (delta_Gm / Vmol)  # Elastic constants in dimensionless form
c12 = 82.7e9 / (delta_Gm / Vmol)
c44 = 37.5e9 / (delta_Gm / Vmol)

e11 = np.zeros((nx, ny))
e22 = np.zeros((nx, ny))
e12 = np.zeros((nx, ny))
e21 = np.zeros((nx, ny))
s11 = np.zeros((nx, ny))
s22 = np.zeros((nx, ny))
s12 = np.zeros((nx, ny))
s21 = np.zeros((nx, ny))

# == Plastic strain/stress parameters
D_alpha_pl = 0.3  # Average thickness of alpha lath from SLM, unit: mm
sigmaA_pl = 550
sigmaB_pl = 1350  # Lattice friction stress of alpha and beta phase, unit: MPa
kappaHP_pl = 300  # Hall-Petch coefficient, unit: MPa
kappa_pl = 0.23
n_pl = 0.4  # Fitting parameters from experimental data
epsilon_pl = 1  # Strain rate
mu_pl = (54 - 0.03 * T0) * 1e9  # Shear modulus, unit: Pa
b_pl = 2.9e-10  # Unit: m

k_pl = 1e-03  # Constant for fourth-order plastic kinetic coefficient tensor
G_pl = (kappa_pl * mu_pl * b_pl**3 / (Kb * T0 * np.log(1e7)))**n_pl

# == Modified Johnson-Cook (MJC) model for plastic strain hardening
JCa_pl = 0.92e03
JCb_pl = 0.4e03  # Fitting parameters for MJC model, unit: MPa
JCn_pl = 0.578
JCm1_pl = 0.1578
```

```
    JCm2_pl = 0.633
    Tr = 298  # Room temperature, unit: K
    Tm = 1878  # Melt temperature, unit: K

    T = (T0 - Tr) / (Tm - Tr)

    e1_pl = np.zeros((nx, ny))
    e2_pl = np.zeros((nx, ny))
    e3_pl = np.zeros((nx, ny))

    # == Pre-set matrix
    tmpphiAplot = np.zeros((nx, ny))
    phiAplot = np.zeros((nx, ny, variants, grainBs))
    sumphiAplot1 = np.zeros((nx, ny))
    sumphiAplot2 = np.zeros((nx, ny))

    sym_x, sym_y = np.meshgrid(np.arange(1, nx+1), np.arange(1, ny+1), indexing='ij')  # 2D spatial coordinates
    tmp_x = sym_x.flatten()
    tmp_y = sym_y.flatten()
    sym_cor_mat = np.column_stack((tmp_x, tmp_y, np.zeros(nxy)))

    vflag = nstep // nprint1
    VolF = np.zeros((vflag + 1, variants + 1))
    VolF[1:vflag + 1, 0] = np.arange(1, vflag + 1) * nprint1

    cc = plt.cm.hsv(np.linspace(0, 1, variants * grainBs))



    import numpy as np

    def greenmatrix_stiffnessmatrix_eigenstrain(grainBs, variants, phiB, psi, ang, e0, nx, ny, kx, ky, c11, c12, c44
        # Position-dependent elastic constant
        Cijkl = np.zeros((2, 2, 2, 2))
        Cijkl[0, 0, 0, 0] = c11
        Cijkl[1, 1, 1, 1] = c11
        Cijkl[0, 0, 1, 1] = c12
        Cijkl[1, 1, 0, 0] = c12
        Cijkl[0, 1, 0, 1] = c44
        Cijkl[0, 1, 1, 0] = c44
        Cijkl[1, 0, 0, 1] = c44
        Cijkl[1, 0, 1, 0] = c44

        rot = np.zeros((2, 2, grainBs))
        tot_Cpqrs = np.zeros((nx, ny, 2, 2, 2, 2))
        hom_Cpqrs = np.zeros((2, 2, 2, 2))
        het_Cpqrs = np.zeros((nx, ny, 2, 2, 2, 2))

        for p in range(2):
            for q in range(2):
                for r in range(2):
                    for s in range(2):
                        for g in range(grainBs):
                            rot[:, :, g] = [[np.cos(ang[g]), np.sin(ang[g])],
                                            [-np.sin(ang[g]), np.cos(ang[g])]]
                            for i in range(2):
                                for j in range(2):
                                    for k in range(2):
                                        for l in range(2):
                                            tot_Cpqrs[:, :, p, q, r, s] += (
                                                phiB[:, :, g] *
                                                rot[p, i, g] * rot[q, j, g] *
                                                rot[r, k, g] * rot[s, l, g] *
                                                Cijkl[i, j, k, l]
                                            )
                        hom_Cpqrs[p, q, r, s] = (
                            np.max(tot_Cpqrs[:, :, p, q, r, s]) +
                            np.min(tot_Cpqrs[:, :, p, q, r, s])
                        ) / 2
                        het_Cpqrs[:, :, p, q, r, s] = tot_Cpqrs[:, :, p, q, r, s] - hom_Cpqrs[p, q, r, s]

        tot_Cpq = np.zeros((nx, ny, 3, 3))
```

```
        hom_Cpq = np.zeros((3, 3))
        het_Cpq = np.zeros((nx, ny, 3, 3))

        # Mapping indices
        map_indices = [
            ((0, 0), (0, 0, 0, 0)),
            ((0, 1), (0, 0, 1, 1)),
            ((0, 2), (0, 0, 0, 1)),
            ((1, 0), (1, 1, 0, 0)),
            ((1, 1), (1, 1, 1, 1)),
            ((1, 2), (1, 1, 0, 1)),
            ((2, 0), (0, 1, 0, 0)),
            ((2, 1), (0, 1, 1, 1)),
            ((2, 2), (0, 1, 0, 1))
        ]
        for idx, Cpqrs_idx in map_indices:
            tot_Cpq[:, :, idx[0], idx[1]] = tot_Cpqrs[:, :, Cpqrs_idx[0], Cpqrs_idx[1], Cpqrs_idx[2], Cpqrs_idx[3]]
            hom_Cpq[idx[0], idx[1]] = hom_Cpqrs[Cpqrs_idx[0], Cpqrs_idx[1], Cpqrs_idx[2], Cpqrs_idx[3]]
            het_Cpq[:, :, idx[0], idx[1]] = het_Cpqrs[:, :, Cpqrs_idx[0], Cpqrs_idx[1], Cpqrs_idx[2], Cpqrs_idx[3]]

        # Eigenstrain for different variants in different parent grains
        eigen = np.zeros((nx, ny, 2, 2, variants, grainBs))
        for g in range(grainBs):
            for v in range(variants):
                for ii in range(2):
                    for jj in range(2):
                        for kk in range(2):
                            for ll in range(2):
                                eigen[:, :, ii, jj, v, g] += (
                                    psi * rot[ii, kk, g] * rot[jj, ll, g] * e0[kk, ll, v]
                                )

        # Green's tensor
        gomega = np.zeros((nx, ny, 3))
        Spq = np.zeros((nx, ny, 3, 3))
        for ix in range(nx):
            for iy in range(ny):
                Spq[ix, iy, :, :] = np.linalg.inv(tot_Cpq[ix, iy, :, :])
                n = np.array([kx[ix, iy], ky[ix, iy]])
                iomega = np.zeros((2, 2))
                for ii in range(2):
                    for jj in range(2):
                        iomega[ii, jj] = (
                            hom_Cpqrs[0, ii, jj, 0] * n[0] * n[0] +
                            hom_Cpqrs[0, ii, jj, 1] * n[0] * n[1] +
                            hom_Cpqrs[1, ii, jj, 0] * n[1] * n[0] +
                            hom_Cpqrs[1, ii, jj, 1] * n[1] * n[1]
                        )
                omega = np.linalg.inv(iomega)
                gomega[ix, iy, 0] = omega[0, 0]
                gomega[ix, iy, 1] = omega[1, 1]
                gomega[ix, iy, 2] = omega[0, 1]

        gomega[0, 0, :] = 0
        Spq /= k_pl

        return tot_Cpq, hom_Cpq, het_Cpq, eigen, gomega, tot_Cpqrs, Spq


    import numpy as np

    # Clear and initialize
    print("Clearing workspace and initializing parameters...")

    # Simulation parameters
    #from simulation_parameters import *  # Assuming parameters are in a separate Python module

    # Load data
    print("Loading BetaMap_16p1028grid.npy...")
    data = np.load('BetaMap_16p1028grid.npy')  # Adjust based on actual file format
    phiB = data['phiB']
```

```
# Interpolation function for relaxation of misfit strain
phiB2 = phiB ** 2
sumphiB2 = np.sum(phiB2, axis=2)
maxsumphiB2 = np.max(sumphiB2)
minsumphiB2 = np.min(sumphiB2)

psi = (rho * (maxsumphiB2 - sumphiB2) + (sumphiB2 - minsumphiB2)) / (maxsumphiB2 - minsumphiB2)

StrucphiB0 = np.ones((nx, ny, grainBs))
StrucphiB0[phiB == 0] = 0
StrucphiB = np.zeros((nx, ny, variants, grainBs))

for g in range(grainBs):
    for v in range(variants):
        StrucphiB[:, :, v, g] = StrucphiB0[:, :, g]

phiA = np.zeros((nx, ny, variants, grainBs))

# Pre-FFT parameters
tmpkx = 2 * np.pi * np.fft.fftfreq(nx)
tmpky = tmpkx
kx, ky = np.meshgrid(tmpkx, tmpky)
k2 = kx**2 + ky**2
kx = np.divide(kx, np.sqrt(k2), out=np.zeros_like(kx), where=k2!=0)
ky = np.divide(ky, np.sqrt(k2), out=np.zeros_like(ky), where=k2!=0)

# Green's operator and eigenstrain matrix (assumed function)
tot_Cpq, hom_Cpq, het_Cpq, eigen, gomega, tot_Cpqrs, Spq = greenmatrix_stiffnessmatrix_eigenstrain(
    grainBs, variants, phiB, psi, ang, e0, nx, ny, kx, ky, c11, c12, c44, k_pl
)

# Denominator in Allen-Cahn function
denom = 1 + dt * M * kappa * k2

# Microstructure evolution
for istep in range(1, nstep + 1):
    ttime += dt

    phiA2 = phiA ** 2
    sumphiA2 = np.sum(phiA2, axis=(2, 3))
    phiAk = np.fft.fft2(phiA)

    if np.max(sumphiA2) > 1:
        sumphiA2 /= np.max(sumphiA2)

    dfdphiA = a * phiA - b * phiA2 + c * phiA * sumphiA2
    dfdphiAk = np.fft.fft2(dfdphiA)

    Sigmay2 = 1e100 * np.ones((nx, ny))  # Extremely large yield stress

    deldphiAk, E_elastic, e1_pl, e2_pl, e3_pl, output_s11, output_s22, output_s12 = solve_elasticity(
        phiA, StrucphiB, eigen, e1_pl, e2_pl, e3_pl, grainBs, variants,
        tot_Cpq, hom_Cpq, het_Cpq, gomega, tot_Cpqrs, Spq, Sigmay2, nx, ny, kx, ky, dt, delta_Gm, Vmol
    )

    output_Vons = np.sqrt(output_s11**2 + output_s22**2 - output_s11 * output_s22 + 3 * output_s12**2)

    if istep <= nucleationstep:
        r1 = np.random.rand(nx, ny, variants, grainBs)
        r2 = np.random.rand(nx, ny, variants, grainBs)
        noise = -2 * np.log(r1) * np.sin(2 * np.pi * r2) * np.cos(2 * np.pi * r2) * ampnoise
        noisek = np.fft.fft2(noise)

        phiAk = (phiAk - dt * M * (dfdphiAk + deldphiAk)) / denom + noisek
    else:
        phiAk = (phiAk - dt * M * (dfdphiAk + deldphiAk)) / denom

    phiA = StrucphiB * np.real(np.fft.ifft2(phiAk))

    phiA[phiA > 1] = 1
```

```
        phiA[phiA < 0] = 0

        if istep % nprint1 == 0:
            filename1 = f"phiA2_ttime_{ttime:.4f}s.npy"
            filename2 = f"sumphiAplot2_ttime_{ttime:.4f}s.npy"
            filename3 = f"VonMisesStress_ttime_{ttime:.4f}s.npy"
            filename4 = f"ElasticEnergy_ttime_{ttime:.4f}s.npy"

            np.save(filename1, phiA2)
            np.save(filename2, sumphiA2)
            np.save(filename3, output_Vons)
            np.save(filename4, E_elastic)

            if np.max(sumphiA2) > 1 or np.max(phiA) < 1e-2 or np.isinf(deldphiAk).any():
                with open('breakpoint.txt', 'w') as f:
                    f.write(f"maxsumphiAplot: {np.max(sumphiA2)}\n")
                    f.write(f"maxphiA: {np.max(phiA):.4f}\n")
                break

filename = 'VolumnFraction.npy'
np.save(filename, VolF)
```

```
Clearing workspace and initializing parameters...
Loading BetaMap_16p1028grid.npy...
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-11-40e5d267720b> in <cell line: 11>()
      9 # Load data
     10 print("Loading BetaMap_16p1028grid.npy...")
---> 11 data = np.load('BetaMap_16p1028grid.npy')  # Adjust based on actual file format
     12 phiB = data['phiB']
     13

/usr/local/lib/python3.10/dist-packages/numpy/lib/npyio.py in load(file, mmap_mode, allow_pickle, fix_imports, encoding,
max_header_size)
    425             own_fid = False
    426         else:
--> 427             fid = stack.enter_context(open(os_fspath(file), "rb"))
    428             own_fid = True
    429

FileNotFoundError: [Errno 2] No such file or directory: 'BetaMap_16p1028grid.npy'
```

Next steps: Explain error

Start coding or generate with AI.