Thresholding Method for simulating the alternate KWC model Code User Manual

Jaekwang Kim, Matt Jacobs, Nikhil Chandra Admal

September 23, 2020

1 Introduction

This is the user manual for illustrating details of the numerical scheme suggested in the white paper. A `C++` based software is developed to simulate the kinetics of grain growth described by defined by KWC (Kobayashi–Warren–Carter) model [1, 2, 3]

```
Input  : System size $N$, parameter $\epsilon, \xi$ and an initial grain configuration $\theta^0$
Output: Time evolution of the alternate KWC model
Initialize Grains $\theta^0$ on domain $\Omega$
while  $t < T$ do
    Compute grain boundary energy $\mathcal{J}(\llbracket\theta\rrbracket)$

     Execute Primal-dual algorithm
    while $\max(\eta^{n+1} - \eta^n) < tol$ do
        Update $\eta^{n+1}$ using $\psi^n$ Update $\psi^{n+1}$ using $\eta^{n+1}$
    end

     Execute thresholding algorithm
    Identify interior regions of grains $I_p$
        Execute Fast Marching
        Allow each grain $I_p$ to grow with speed $1/(1-\eta)^2$
        Threshold $\theta(x), x \in \Omega - I$ to that of a interior grain arrives at earliest

     Compute free energy $\mathcal{W}_{\mathrm{kwc}}(\eta, \theta)$
end
```

Algorithm 1: Suggested algorithms for the alternative KWC simulation

In this model, the material free energy density $\mathcal{W}$ is defined by two field variable: the order-phase parameter η and the orientation-phase parameter θ. The free energy of the alternate KWC is

$$\mathcal{W}[\eta, \theta] = \int \left[\frac{(1-\eta)^2}{2\epsilon} + \frac{\epsilon}{2}|\nabla\phi|^2 + sg(\eta)\mathcal{J}\big(\llbracket\theta\rrbracket\big)\delta(x - x_0) \right] dV. \tag{1}$$

The code solves the KWC gradient flow by iteratively solving the following optimization problems:

$$\eta^{k+1} = \arg\min_{\eta} \mathcal{W}[\eta, \theta^k] \tag{2}$$

$$\theta^{k+1} = \arg\min_{\theta} \mathcal{W}[\eta^{k+1}, \theta] \tag{3}$$

To solve the η-sub problem, the code uses the Primal-dual algorithm [4, 5]. For second, the θ-sub problem, we use thresholding dynamics.

The developed code has a modularized structure. A user-program can be executed by through a driver file (we suggest a `.cpp` format) referring necessary C++ functions and classes from different header files, which are categorized by its task. Each function and class can be replaced by a user-defined form, as long as the input & output formatting is consistent. The list of header files is summarized in Table 1. and the flow chart of algorithm is attached in Figure 1

2 Environment

The current code is written by `C++` language and has been tested with g++ compiler. For Fast Fourier Transform (FFT) and inverse FFT, it borrows necessary operators from `FFTW3` library. Simple linear algebra is operated through a `Eigen` software. We suggest to use a 'makefile' tool to export these environment variables, an example format of which is as follow

```
IDIR   += -I../../include/KWC_Simulation
IDIR   += -I../../Eigen
CC= g++
CFLAGS += -Ofast
CFLAGS += -std=c++14
CFLAGS += -lm
CFLAGS += -lfftw3_threads
CFLAGS += -lfftw3
program:
        $(CC) KWC_driver.cpp -o main $(CFLAGS) $(IDIR)
```

For data visualization, we use `ffmpeg` libraries and `Paraview`. Yet, those two are not mandatory.

3 Driver File

We recommend to execute the program using a driver file written in `.cpp` format. We will example basic elements of a driver file with an example, which runs the bicrystal simulation. Driver files for

Header file	Contents
`InitCrystal.h`	Functions that set the initial condition of simulation, i.e. initialize $\theta(x)$
`DataOut.h`	Functions related to output solution for visualization
`Material.h`	A class that defines GB energy $\mathcal{J}\left([\![\theta]\!]\right)$ of different types of material
`PostProcessor`	Functions that compute the KWC free energy of polycrystal
`PrimalDual.h`	A class that solves η-sub problem at a given $\theta(x)$ configuration
`KWCThresholding.h`	A class that solves θ-sub problem at a given $\eta(x)$ configuration
`KWCJumpFunction.h`	A class that design $\mathcal{J}$ from a provided external GB data
`Metrics.h`	A list of simple mathematical functions

Table 1: Table of header files

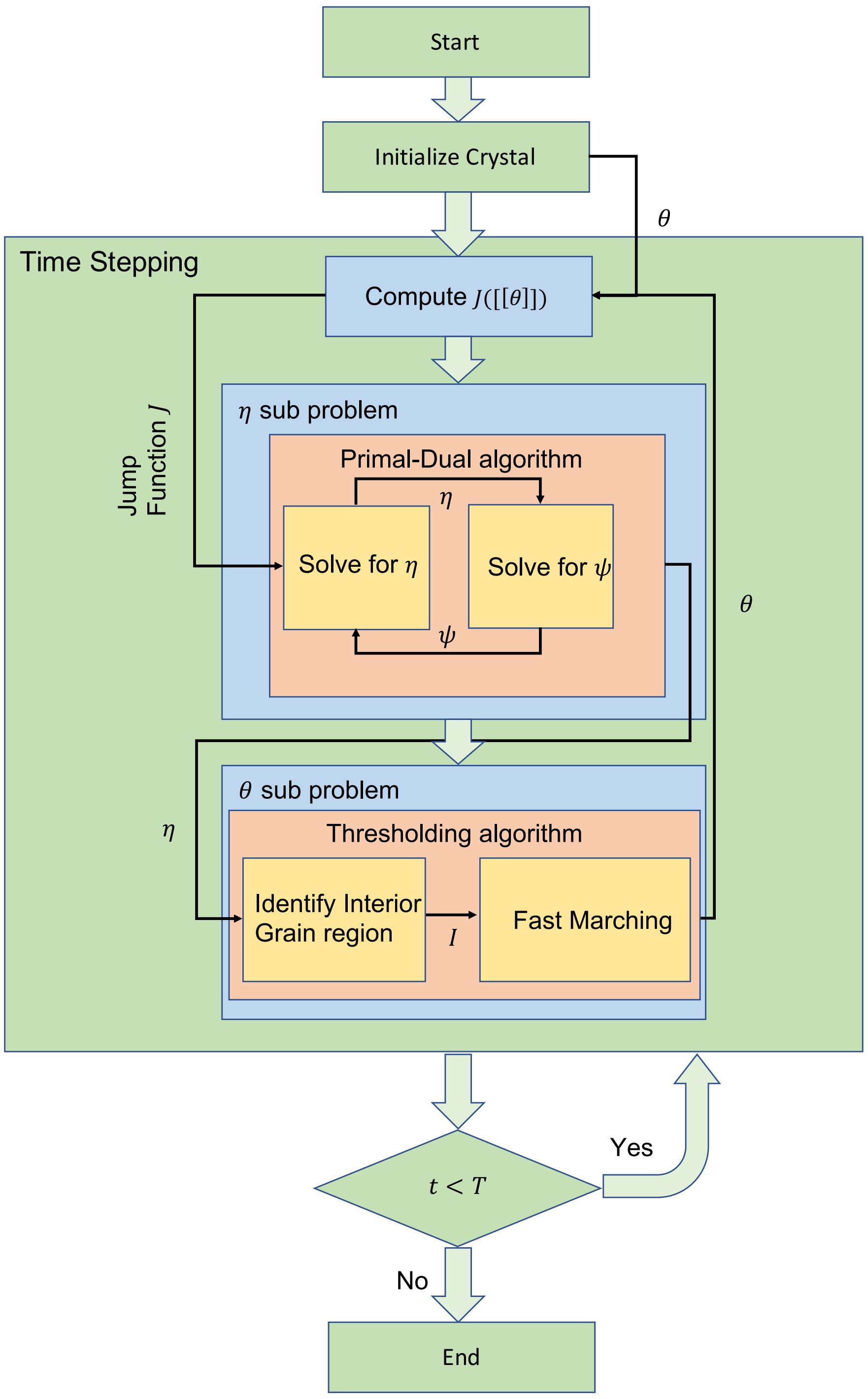

Figure 1: Algorithm Flow chart

other grain configuration can build on this basic structure.

3.1 Execution

Once the driver file is successfully compiled, it will create an executable, Say that it is ‘main’. The executable can be run with following arguments, which stands for size of computational domain N_x, N_y, N_z, and ϵ value.

```
./main 512 512 1 0.05
```

3.2 Driver file strcuture

- List of required header files

```
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <float.h>
#include <stdio.h>
#include <assert.h>
#include <iostream>
#include <vector>
#include "DataOut.h"
#include "InitCrystal.h"
#include "PostProcessor.h"
#include "PrimalDual_Neumann.h"
```

- Define domain size and declare global variables

```
int main(int argc, char *argv[]){

  //Read global variables from bash
  //Define grid size and set model parameter epsilon
  int n1=atoi(argv[1]);
  int n2=atoi(argv[2]);
  int n3=atoi(argv[3]);
  double epsilon=atof(argv[4]);

  /* Construct Classes */
  int const DIM=3;
  const unsigned int lcount=1;
  int pcount = n1*n2*n3;
  double dt= epsilon * epsilon; //initial choice of dt

  double *Xangles = new double[lcount]();
  double *Yangles = new double[lcount]();
  double *Zangles = new double[lcount]();
  double *eta = new double[pcount]();
  int *labels = new int[pcount]();
  double *energyField = new double[pcount]();
  int Nthread = 1; //The number of threads to be used for FFTW
  ...
```

- Set material types

Next, we construct a material type for $\mathcal{J}$. You may need to set material parameters if necessary.

```
  //Here, 's' stands for the original KWC model of J
  char materialType='s';
  Material material;
  //set material constant value s
  material.s=1.0;
```

Here, the material type 's' stands for simple material, which is $\mathcal{J}$ of original KWC grain model.

- Initialize crystal

Then, we construct initialize a bicrystal.

```
  //Initialize crystal: set initial condition of $\theta(x)$
  InitializeCrystal::oneD_Bicrystal_configuration(n3,n2,n1,labels);
```

Other crystal configuration can be found in `InitCrystal.h` header file. A user can also define new crystal configuration independently for one's own needs.

- Construct algorithm `C++` class and link global variables

```
//Construct PD Algorithm class
double PDerror=1e-6; // tolerance of Primal-dual algorithm
int PDmaxIters=10000; // allowable iteration number of the algorithm

PrimalDual<DIM> EtaSubProblem(n3, n2, n1, PDerror, PDmaxIters,
                                lcount,epsilon, Nthread);

//Link pointers of Global variables to the Primal-Dual algorithm class
EtaSubProblem.setUpClass(eta, Xangles, Yangles, Zangles,
labels, energyField,materialType);
```

- Run simulation

```
    //Run Primal-dual algorithm
    EtaSubProblem.run(material,epsilon);

    //output int type data
    DataOut::Output1Dsolution(n1, n2, 0, eta , "eta1D", n1);

    //Calculate Grain boundary energy and print out
    double energy = computeKWCEnergy(material.s,n1,n2,epsilon,eta,Zangles,labels);
    std::cout <<"  GB Energy:  " << energy << std::endl;
```

- Release memory space

Once simulation is finished, we release memory spaces.

```
    EtaSubProblem.freeMemory();
    delete[] eta; eta=NULL;
    delete[] labels; labels=NULL;
    delete[] Xangles; Xangles=NULL;
    delete[] Yangles; Yangles=NULL;
    delete[] Zangles; Zangles=NULL;
```

These are the basic structure of example files.

In the following, we will consider other cases to introduce other features of the code, e.g. how to execute the thresholding algorithm with different material type

4 Example codes

4.1 KWC polycrystal simulation

The example code `KWC_Polycrystal.cpp` will introduce

1. how to set a thresholding algorithm class
2. how to generate grain evolution animation

The code runs the polycrystal simulation with the original KWC model.

- Header files

To simulate grain growth, following header files should be included

```
#include "DataOut.h"
#include "InitCrystal.h"
#include "PostProcessor.h"
#include "PrimalDual.h"
#include "KWCThresholding.h"
```

- Defining domain size and global variables and selecting the simple KWC material type
  We skip these trivial parts, because they are same with the previous bicrystal simulation

- Initialize crystal

Now, we construct a random 2D-polycrystal using another function define in `InitCrystal.h`

```
//The possible maximum Z-orientation
double maxZangle = 70.0* M_PI/180.0;

InitializeCrystal::RandomCrystalConfiguration2D(n3,n2,n1,lcount,labels,
                            maxZangle, Xangles, Yangles,Zangles);
```

- Construct algorithm `C++` class and link global variables

The thresholding algorithm class needs an additional parameter ξ which will be used as an initial criteria to identify interior regions of grain, i.e. if $\mathcal{J}\left([\![\theta]\!] < \xi\right.$ which we will consider as grain interiors.

```
double PDerror=1e-6; // tolerance of Primal-dual algorithm
int PDmaxIters=10000; // allowable iteration number of the algorithm
PrimalDual<DIM> EtaSubProblem(n3, n2, n1, PDerror, PDmaxIters,lcount,
                                                    epsilon, Nthread);

// Criteria for Identifying interior regions of grains
double initThresCriteria = 2*epsilon ;

KWCThreshold<DIM> FastMarching(n3,n2,n1,lcount,initThresCriteria);

EtaSubProblem.setUpClass(eta, Xangles, Yangles, Zangles,
                         labels, energyField,materialType);

FastMarching.setUpProblem(eta, Xangles, Yangles, Zangles,
                         labels, energyField,'P');
```

- (Optional) Setting FFMEPG for grain animation

The developed code can generate grain evolution movie while simulation is running. This will necessitate that `ffmpeg` is installed on the machine. We will convert the Z-orientation value of grains into a number between [0,255]. Then, we will use black-and-white images to collect pictures of grains at each time step.

```
  /* movie data */
  unsigned char *pixels=new unsigned char[pcount];

  unsigned char *colors=new unsigned char[lcount];
  for(int l=0;l<lcount;l++){
      // Distribute colors to angles
        colors[l]=255*Zangles[l]/maxZangle;
  }
```

```
char *string;
asprintf(&string,"ffmpeg -y -f rawvideo -vcodec rawvideo -pix_fmt gray -s
 %dx%d -r 30 -i - -f mp4 -q:v 5 -an -vcodec mpeg4 out.mp4", n1,n2);

//open an output pipe
FILE *pipeout = popen(string, "w");
```

- Run simulation

We take 200 time steps, and the pixel grain data will be reformatted to black-and-white image data.

```
//Start Dynamics
for (int i =0 ; i<200 ; i++)
{
  PrepareFFMPEG2DPixels(n1,n2,0,pixels, labels, colors);
  fwrite(pixels, 1, pcount, pipeout);

  EtaSubProblem.run(material, epsilon);
  FastMarching.run(epsilon);
}
```

- Release memory space

We release memory space. In this case, we also need to take care of the memory space used for the thresholding class. If one used `ffmpeg`, ones need to flush out the command to the shell.

```
fflush(pipeout);
pclose(pipeout);

EtaSubProblem.freeMemory();
FastMarching.freeMemory();

delete[] eta; eta=NULL;
delete[] labels; labels=NULL;
delete[] Xangles; Xangles=NULL;
delete[] Yangles; Yangles=NULL;
delete[] Zangles; Zangles=NULL;
```

4.2 Simulation of the alternate KWC model

The main advantage of using the alternate KWC model is that one can freely construct the function $\mathcal{J}$ in Eq (1). In this example, we will how to use alternate simulation.

4.2.1 Design the jump function

The example code `Design_J.cpp` uses a Newton's iteration to fit $\mathcal{J}(\llbracket\theta\rrbracket)$ to an external grain boundary energy data. The code takes an input file `STGB_cu_110.txt` which is the covariance model prediction of FCC [110] Symmetric-tilt-grain-boundary energy [6, 7]. The format of the input file is as follow

```
 # tiltAngle(deg)          # W_data
 # -------------            -------------
                    0                0.0
                  0.5            0.10458
                    1           0.203463
                  1.5           0.296963
                    2           0.385376
                  2.5           0.468977
                    3           0.548028
                  3.5           0.598243
... (continues)
```

To construct $\mathcal{J}$, we can call `KWCDesignJ C++ class`. We only need to provide the number of data and the name of input&output file as follow.

```
int main(int argc, const char * argv[]) {
     int nData=361;
     KWCDesignJ optimizer(nData,"STGB_cu_110.txt","J_cu_110.txt");
     optimizer.run();
     return 0;
}
```

Then, the output file format, `J_cu_110.txt` can be later used for grain growth simulation.

```
    # tiltAngle        # Jtheta  # Total Energy
 # -------------   -------------   -------------
             0               0               0
           0.5       0.0432744         0.10458
             1        0.102464        0.203463
           1.5        0.171975        0.296963
             2        0.250429        0.385376
           2.5        0.337459        0.468977
             3        0.433323        0.548028
           3.5        0.502413        0.598243
... (continues)
```

4.3 Running KWC polycrystal simulation with the designed J

Now, we are ready to simulate grain evolution with crystal symmetry-invariance grain boundary energy. Most of program structure is same with the previous polycrystal simulation code, but we only need to simply replace the material type as ‘c’ and designate the $\mathcal{J}$ data.

- Set material types

```
char materialType='c';
  Material material;

  if(materialType =='c')  {
    material.setCovarianceModel(361, "inputs/jfun_cu_110.txt");
  }
```

References

[1] R. Kobayashi, J. A. Warren, and W. C. Carter. Vector-valued phase field model for crystallization and grain boundary formation. *Physica D: Nonlinear Phenomena*, 119:415–423, 1998.

[2] A. E. Lobkovsky and J. A. Warren. Sharp interface limit of a phase-field model of crystal grains. *Physical Review E*, 63:051605, 2001.

[3] J. A. Warren, R. Kobayashi, A. E. Lobkovsky, and W. C. Carter. Extending phase field models of solidification to polycrystalline materials. *Acta Materialla*, 51:6035–6058, 2003.

[4] A. Chambolle and T. Pock. A first-order Primal-Dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40:120–145, 2011.

[5] M. Jacobs, F. Leger, W. Li, and S. Osher. Solving large-scale optimization problems with a convergence rate independent of grid size. *SIAM Journal on Numerical Analysis*, 57:1100–1123, 2019.

[6] B. Runnels, I. J. Beyerlein, S. Conti, and M. Ortiz. An analytical model of interfacial energy based on a lattice-matching interatomic energy. *Journal of Mechanics and Physics of Solids*, 89:174–193, 2016.

[7] B. Runnels, I. J. Beyerlein, S. Conti, and M. Ortiz. A relaxation method for the energy and morphology of grain boundaries and interfaces. *Journal of Mechanics and Physics of Solids*, 94:388–408, 2016.