Liquid Drop Breakup in Turbulent Flow

by

Cheng Zhong

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering
University of Alberta

Abstract

The focus of this study is to gain a fundamental understanding of liquid-liquid dispersion formation in homogeneous isotropic turbulence. This information is crucial to improve the reliability of existing models that describe drop breakup in turbulent flow. These models inherit numerous assumptions, simplifications, experimental constants and fitting parameters. Visualization and quantification of drop behavior in homogeneous turbulence will allow assessment of these models. Direct numerical simulations were used to investigate the dynamics of drop behavior. The free energy lattice Boltzmann method was used to perform simulations.

The homogeneous isotropic turbulence was generated in a three-dimensional fully-periodic domain of 300^3 lattice units in size using a forcing method. Three turbulent flow fields at different levels of energy input were investigated. Then, drops of different initial diameter were injected. The dispersed to continuous fluid viscosity ratios equal to 0.1, 1, and 10 were considered. The DNSs produce detailed description of the flow. The main goal of this study was to translate these data to the useful quantities that can be applied to assess the drop breakup models. This work specifically focused on understanding of drop interaction with turbulent structures. A normalized Q_n criterion was used to visualize the structures. Different combinations of a threshold value and a cutoff volume were studied to explore the effect of these two important parameters and to identify the best combination. The interaction between turbulent vortices and the drops was visualized by extracting coherent structures and tracking liquid-liquid interface in two phase turbulence. The three-dimensional

energy spectra of single phase and two-phase turbulence were also quantified. The statistical characteristics of liquid-liquid turbulence were investigated: the probability density function of vorticity, of normalized energy dissipation rate, and the eigenvalues of the strain tensor. By utilizing these tools, the guidelines are proposed for improvement of the breakup models.

Preface

A part of work in this study was presented in an extended abstract for the 26th Annual Conference of the CFD Society of Canada that was hosted at the University of Manitoba, June 10 - 12, 2018. Dr. Komrakova was the supervisory author of the abstract and was involved in the analysis of the results as well as the composition of the abstract.

Acknowledgements

I would like to appreciate my supervisor Dr. Alexandra Komrakova for her enthusiasm and professional guidance in my research. She encouraged me to jump out of the safe zone and explore more exciting things. Everytime I was struggling for a problem, she was always there with invaluable suggestion that helped me come up with new ideas. She also patiently gave me support on writing. Under her supervision, I can feel the improvement of my ability and extension of my knowledge.

I would also thanks to Dr. Carlos Lange who gave me invaluable advice not only on research but also on life.

My thanks to my friends: Feiyang Jiang, Muxi Li, Lei Li, Zhen Yang, Li Lei. I have a good time at University of Alberta with their help.

Financial support from the Natural Sciences and Engineering Research Council of Canada is greatly appreciated. I would also appreciate the computational resource provided by Compute Canada.

Finally, numerous thanks to my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Applications of liquid-liquid systems

Emulsions and dispersions are of great interest in chemical industry [1, 2, 3], pharmacy [4, 5, 6, 7], electrochemistry [8, 9, 10], biology [11, 12, 13], petroleum industry [14, 15, 16], food [17, 18, 19], cosmetics [20, 21, 22], and agriculture [23, 24, 25]. In these systems, at least two immiscible liquids are agitated to generate a dispersed flow that ensures good contact between the phases and allows for the control of the interfacial area. The additional gases, liquids, or solids may also exist in these systems. When the liquids are agitated, multiple events occur in the system simultaneously: the breakup of drops, known as 'dispersion'; the merging of drops, known as 'coalescence'; and the suspension of drops [26]. These events results in the variance of drop size distribution (DSD) in the liquid-liquid system, thus, affects the final state of the system.

In an agitated vessel, for different operating conditions such as various impeller speed and agitation time, the different state of dispersed phase and phase contact will form. The variance of liquid-liquid interface or drop size distribu-

tion in the system results in the variance of mass transfer and heat transfer rates between the phases [11]. Due to various goals of industrial applications, the liquid-liquid systems are controlled to reach the different requirements on properties of flows such as the interfacial area, and, thus, drop size distributions. For example, a specific phase contact has to be formed in nitration reactions, because the reaction rate and temperature are mainly governed by the interfacial area; in suspension polymerization process, the generation of uniform beads is necessary [26], thus, a specific drop size distribution should be reached.

The thorough understanding of liquid-liquid system formation leads to improvement of industrial process, quantity and quality of products. For example, improvement of speed of liquid-liquid extraction in chemical industry fulfilled by interface control was provided in [2], in that study, the target analyte that exist in homogeneous aqueous phase was extracted into a water-immiscible sedimented phase. The interference that would have resulted from organic-aqueous phase contact disappears.

In the chemical and biochemical industries, the exploration of immiscible liquid-liquid dispersion in stirred vessels is important because it commonly exists in industrial processes such as liquid-liquid extraction [27], and suspension polymerization [28]. In solvent extraction problem, a large interfacial area between two liquid phases is necessary to keep the mass transfer between two phases, which results in a requirement of continuous agitation [29]. In food and pharmaceutical industries, the emulsification often occurs when small drops (for example, $1\mu m$) are distributed in continuous phases [30, 31]. Usually, the emulsion is stable and viscous, the rheological characteristics are non-Newtonian.

The demand on environment protection boosts the development of renewable energy such as biofuels [32, 33]. The liquid-liquid equilibrium state is vital in the design of separation and purification process of crude bioethanol and biodiesel fuel [12]. However, it is also proposed in [12] that the equilibrium data that is used to predict liquid-liquid equilibrium state is not available from experiments. Thus, a UNIQUAC Functional-group Activity Coefficients is introduced to gain equilibrium data. To have a higher product yields and prevent unexpected repolymerization in biofuel upgrading reactions [30], the hydrophobic zeolites are introduced to stabilize water/oil emulsions and catalyse reactions at liquid-liquid interface.

In an optic study, liquid-liquid systems are used as the lens in sensing, medical diagnostics [34, 35]. For example, by electrowetting or giving pressure on a water-oil interface, the optic properties of a drop will change with its shape. Based on this control of liquid-liquid interface, an easy adjustable tunable liquid lens are created for optic experiments [35].

The water-oil emulsions are also common in dyes and pigments problem [31, 36]. Due to the carcinogenic and mutagenic effect of effluents from industries such as printing, dyestuff manufacturing, these pollutants have to be diminished. Thus, liquid surfactant membranes and emulsion liquid membranes are used to remove or recover dyes from effluents [31].

In chemical industry, antibiotics are usually produced in a liquid-liquid system, therefore, the separation of antibiotics is very important. The aqueous system often consists of water and a room temperature ionic liquid. Solvent extraction methods such as liquid-liquid partitioning are studied to fulfill this goal [37].

Liquid-liquid dispersions are common in industries, but at the same time

they are complex. The predictive tools that provides a guideline on how to agitate the liquids in a given vessel in order to produce the dispersion with known properties are needed [38, 39, 40]. The control of liquid-liquid dispersion can improve the products, suppress undesired by-products, and optimize industrial processes [41, 42, 43]. Therefore, the comprehension of behaviour of the system on a drop scale is of significant, as well as the quantification of the interaction of the drop with the surrounding flow. Experiments and simulations can be used for these purposes.

A large number of studies on liquid-liquid systems were carried out experimentally [44, 45, 46, 47]. Through these experiments, phenomena, such as drop breakup and coalescence, were observed. Various theories and assumptions were also introduced from the observation and analyses. Experiments are needed to get more data about drop breakup.

One of the recent studies of a single drop breakup in a rotor-stator mixer combines experiment and simulation [48]. Several findings were outlined: the breakup event mainly exist at downstream region of stator and the inside of jet (the dispersed phase was inject in the system by jet); a larger mother drop increases the time that drop interacts with vortex; the increase of Weber number results in the increase of breakup probability; the distance between current state to the equilibrium state drop size decides the number of fragments after breakup.

A summary of various experiments on a single drop breakup in turbulent flows was given in [49]. With the development of experimental system, facilities of high precision are utilized to capture drops behavior in liquid-liquid system at a smaller length and time scale [50, 51]. Particle image velocimetry (PIV) method with high precision charge-coupled device (CCD) camera is often used

in liquid-liquid systems study. For example, four Photron Fastcam Ultima APX cameras were set up to obtain velocity field at a frame rate of $1000Hz$ [50]. The total time of coalescence is around $600ms$. An experimental study on how dispersed phase volume fraction affects liquid-liquid velocity field was carried out in a baffled cylindrical tank stirred with a six-bladed Rushton turbine [51]. The diameter and height of tank are $0.14m$. The tip speed is $1.3m/s$. The PIV system contains a Kodak Megaplus ES:1.0 CCD camera. However, they found that under this experimental setup, the dispersed phase volume fraction can only up to 10%.

Hasan [52] summarized experiments at different length and times scales in turbulent liquid-liquid systems. The breakup time, which is defined as the time taken from the start of deformation of mother drop to the occurrence of breakage, is an important parameter to predict the population of drops in the binary system. The experiments in his review have drop breakup time that varies from $1ms$ to $100ms$, the diameter of mother drop varies from $0.1mm$ to $3mm$, the breakup time will increase with the increase of drop size and vice versa. However, he pointed out that wide discrepancy exists in the measurement of breakup time because the start of deformation is usually not accurately recorded in experiments. Similar errors also occurs at the observation of occurrence of breakage: the resolution and frame rate limitation may result in the ignorance of infinitesimal satellite that is generated at first moments of breakup process. For the same reason, the record of drop size distribution in liquid-liquid system may be inaccurate, it will result in the inaccurate prediction of mass transfer and heat transfer.

Therefore, drawbacks and limitations still exist in the experimental study of liquid-liquid systems. With the development of computational technology,

the numerical study becomes a powerful tool that interacts with experimental study in the investigation of liquid-liquid systems.

As is denoted in Hasan's review [52], the breakup time decreases with the decrease of drop size. If the size of drop is decreased to much smaller scale, such as Kolmogorov length scale, the experimental facilities will not be able to capture the mother drop and daughters, as well as the breakup event because it will finish in a very short time. Thus, the experimental investigation at a very small length scale and time scale is not a good choice.

Besides, for a given experimental setup, the operating conditions are usually limited. For example, the dispersed phase volume fraction in the experimental setup given in [51] can only up to 10%. The modification of experiment is necessary if the investigation goes to a larger volume fraction.

In chemical industry, the study of mixing time in a liquid-liquid system is significant because it decides the performance of chemical reaction [53]. This is also a vital parameter that is taken into account for mixing system design, optimization, and scale-up from the laboratory scale to the industrial scale. It directly reflects the effectiveness of mixing system. Usually, it is challenging to investigate the mixing time in an industrial scale facility, experiments such as oil-water mixing were carried out in a laboratory size tank. The results obtained from the experiment were usually combined with laboratory scale empirical correlation, thus, problems may exist when extrapolating results to industrial scale agitated vessels [54]. Meanwhile, the experiment study didn't provide detailed localized information as well as the homogeneity degree at different locations [55, 56]. Therefore, the investigation on detail location is limited in experiments.

The disposal of material used in experiments is also a problem. Some

liquids in experiments such as liquid-liquid extraction are pollutants for environment [31]. The oil-water binary system is the common among these liquid-liquid systems, to dispose pollutions and clean the experimental facilities, methods such as oil skimmers, centrifuges, and depth filters are adopted, materials such as polyvinylidene fluoride membranes, boron nitride nanotubes-coated stainless steel meshes, and marshmallow-like gels are utilized, thus, extra budget and time are needed [57].

The demand of investigation on the bridge of local hydrodynamics of the flow and the breakup events boosts the development of multi-scale numerical frameworks.

1.2 Introduction to CFD-PBE numerical framework

With the development of computational technology, the computational fluid dynamics (CFD) becomes a powerful tool. Attempts to better understand the dynamics of turbulent liquid-liquid systems resulted in the development of a numerical framework that involves CFD simulations of flow coupled with the solution of the population balance equation (PBE) that accounts for drop breakup and coalescence. Various modifications of this framework are implemented in commercial software [58, 59, 60]. The wide applications of CFD-PBE numerical framework are reported: investigate drop breakup and coalescence in a pulsed column [61]; simulate the fluidized bed spray granulation [62]; study local hydrodynamics for cell proliferation and protein synthesis in a stirred bioreactor [63].

The population balance plays a critical role in industrial applications [64, 65, 66]. For example, it can be used to predict the nanoparticle precipitation in production [67], forcast the granule growth behaviour [68], control crystal size [69], predict bubble/drop/aggregation breakup and coalescence [70, 38, 71] in agricultural chemicals, minerals, pharmaceuticals, etc. The application of population balance not only covers liquid-liquid systems [72] but also gas-liquid [73], solid-liquid [74] and gas-solid [75] systems. In the implementation of population balance, variables such as the number, mass or volume of particles of the dispersed phase are usually used to describe the population [76], the distributions of particles and the way it affects system behavior are of great interest in research.

The fundamental assumption of PBE is that there must be a number density of particles at each point, the equation is often coupled with conservation equations for entities in continuous phase [76]. In liquid-liquid systems, the breakup and coalescence of dispersed phases can be explained in terms of population balance because new particles can occur in breakup and coalescence processes.

The derivation process of PBE was provided in [76] in detail, it presented a general multi-dimensional form of the PBE:

$$\frac{\partial f}{\partial t} + \nabla_x \cdot \dot{X} f + \nabla_r \cdot \dot{R} f = h \tag{1.1}$$

where f denotes the number density function $f(x, r, t)$ with external (x) and internal (r) coordinates; $\dot{X}$ and $\dot{R}$ are velocities for external and internal coordinates; h represents the resultant net generation rate of particles from birth and death.

In the liquid-liquid system, the continuity equation for the number density function f is provided by [77]:

$$\frac{\partial f(V)}{\partial t} + \nabla \cdot (\vec{u} f(V)) = S(V) \tag{1.2}$$

where V is the drop volume that is regarded as an internal coordinate; $S(V)$ are source terms consist of breakage and coalescence of the drop [77]:

$$S(V) = B^c(V) - D^c(V) + B^b(V) - D^b(V) \tag{1.3}$$

where B^b and B^c are the birth rate due to breakage and coalescence; D^b, and D^c are the death rate due to breakage and coalescence.

In this study, the coalescence of drop is neglected. The source terms due to breakage are then replaced by [78]:

$$S(V) = \int_d^\infty \beta(V, V') g(V') f(V') dV' - g(V) f(V) \tag{1.4}$$

where $\beta(V, V')$ is the daughter distribution function that illustrates the size distribution of daughter drops split out from a mother drop of size V'; $g(V')$ denotes breakup kernel (or frequency of breakup) of mother drop of size V'.

The persistent challenge of such multi-scale simulations lies in the fact that the results are highly dependent on the choice of the breakup and coalescence sub-models (or kernels) that require specification of breakup time, breakup frequency, number of fragments after the breakup, daughter size distribution, etc [79, 80]. For example, even for a simplified PBE (Equation (1.4)) in which only the breakup event in liquid-liquid flow is considered, the different choice of breakup kernel significantly affects the results of PBE. In recent years, many

breakup kernels were provided [81, 80, 82]. However, some obvious discrepancies exist in these kernels, they may resulte in the non-accurate probability of equal breakup, the dependence on energy dissipation rate, the ignorance of small break fraction [70, 83]. A complete breakup kernel must contain the breakup rate and daughter size distribution [83]. A common breakup kernel (CT kernel) that takes the energy dissipation rate ε, dispersed phase density ρ_c, and interfacial energy σ into account was proposed in [84]:

$$g(V) = C_1 \frac{\varepsilon^{1/3}}{V^{2/9}} exp\left(-\frac{C_2\sigma}{\rho_d \varepsilon^{2/3} V^{5/9}}\right) \tag{1.5}$$

where C_1 and C_2 are two constants. As another example, a multifractal (MF) kernel is proposed in [85]:

$$g(V) = \int_{\alpha_{min}}^{\alpha^x} g(V,\alpha) P(\alpha) d\alpha \tag{1.6}$$

where α is singularity strength $P(\alpha)$ is the probability density. The detail of expansion of each components were given in [85, 86, 87]. The comparison of two breakup kernels were reported in [78]: the CT breakup kernel only depends on the energy dissipation rate, other turbulent properties as well as the effect of viscous stress are not involved; the MF breakup kernel is a viscosity dependent kernel, it covers the effect of internal intermittency on breakup such as the large fluctuations on turbulent dissipation rate). The preassumption of both two breakup kernels is that drops breakup because of the pressure fluctuation.

It was also demonstrated that the breakup rate can be over or under predicted by an order of magnitude depending on the underlying assumption of drop/vortex interactions [88].

Considering that the CFD-PBE is becoming an indispensable tool to analysis liquid-liquid systems, therefore, fundamentally well-justified sub-models that can be used for CFD-PBE multi-scale simulations are highly needed. Before the development of sub-models, the major assumptions made in breakup kernels are identified.

1.3 Assumptions made in breakup kernels

Several assumptions in breakup kernels are summarized:

- The large-scale deformation in which the unstable oscillations result in drop break up is more frequent than the tearing mechanism that generates a small drop [89]. The observation of experiment revealed that the unequal size breakup had higher probability compared to the equal size breakup.

- In turbulent flow, only vortices of size equal to or smaller than the drop size leads to the drop breakup [90]. Vortices larger than drop size merely transport drops.

- With the kinetic energy $E(\lambda)$ at vortex size λ, the daughter drop size is confined by the minimum and maximum breakup fraction $f_{v,min}$ and $f_{v,max}$ [83]. The dynamic pressure of vortex $0.5\rho_c u_\lambda^2$ must be larger than the capillary pressure σ/r, it results in the minimum breakup fraction. The vortex kinetic energy must be larger than increment of surface energy in breakup process, it results in the maximum breakup fraction.

- If a drop has volume v, the probability of breaking up into two daughter

drops with volumes vf_v and $(1-v)f_v$ also stay at the range between $f_{v,min}(v, E(\lambda))$ and $f_{v,max}(v, E(\lambda))$ [83].

- There is no mean relative velocity discrepancy between continuous phase and dispersed phase. Therefore, the breakup event is only resulted from turbulent velocity fluctuation [88].

- The relationship between daughter size distribution and the required energy for daughter drop formation is linear. [79]

Among these assumptions, the interaction between drop and vortices is of great interest. Attempts are made in finding relationship between breakup event and surrounding vortex.

1.4 Introduction to coherent structures

Drop/vortex interaction is important because it affects the breakup mechanism of assumption in breakup kernel, however, the data that are available for this analysis are limited. To understand the drop/vortex interaction, the visualization of coherent structures is used. The coherent structure is a region concentrated with high vorticity that makes fluid move around a core, it is coherent in space and temporally evolves vortical motions [91, 92]. Coherent structures are regarded as the elementary components where vortices cluster in turbulent flow [93].

Vortex is to be observed as an rotational region around a core. The concept of vorticity proposed to describe a vortex was defined as the curl of velocity ($\omega = \nabla \times v$). The identification of coherent structures commonly relies on properties of vortices [94]. The turbulent coherent structures were

investigated to reveal the information of turbulence such as pressure, velocity, energy et al. Accurate, convenient and low cost methods were developed and preferred by investigators. Those commonly used methods for the extraction of coherent structures are Reynolds decomposition [95], Galilean (constant convection velocity) decomposition [95], large eddy simulation (LES) decomposition (low-pass filtering) [95], Q criterion [96, 97, 98, 99], Coherent Vortex Simulation (CVS) [100, 101, 102], and Lagrangian method [103, 104].

Adrian et al [95] used Galilean decomposition, Reynolds decomposition, and LES decomposition to visualize vortices in turbulent pipe flow when Reynolds number equals to 50000. Two-dimensional vorticity contours were given. The Reynolds decomposition was not good at vortices visualization because it extracted small-scale vortices well but missed large-scale properties. Besides, the LES decomposition was found to be the best for vorticity visualization due to the good presentation of eddies in all three layers of flow in the pipe.

Q criterion is the widely adopted method for coherent structures extraction due to its low computational cost. It is classified as a Eulerian method based on the principle. In Q criterion, Q is defined as the balance between rotation and strain rates. Thus, the region with positive Q indicates higher rotation. Using this method, Dubief and Delcayre [96] obtained worms shape vortex structures in isotropic free decaying turbulence. Furthermore, in a turbulent mixing layer simulation, they realized that the large-scale span-wise vortices together with stream-wise vortices were forcefully affected by two-dimensional initial perturbations. As for the channel flow with $Re = 160$ and $Ma = 0.3$, they found that it is the coherent structures at the inner region that bolsters the turbulence flow. Besides, the second eigenvalues isosurfaces plots for these

simulations were given. Q isosurfaces and second eigenvalues isosurfaces plots were always near the same, but there was more noise in second eigenvalues isosurfaces plot.

Using Q criterion, Kareem et al. [97] extracted vortex structures from large, intermediate and fine scale homogeneous isotropic turbulent fields at Taylor microscale Reynolds number equal to 81. The threshold values for three flow fields are 5.406, 6.103 and 6.012 correspondingly. In order to generate clear plots, structures with the volume lower than 4% of the maximum structure volume were erased as noise. Vortex structures in fine scale flow were observed to be more sprier in stretching than those in larger scale flow. In large scale flow, structures were easier to stretch when the breakup of vortex occurs.

Ghasempour et al [98] studied structure properties in turbulent pipe flow using Q criterion, the Reynolds number was 20000. They introduced a vortex tracking method to capture single coherent vortices in turbulent flow through its lifetime. Repeating the vortex cross-section, boundary and volume tracking procedure, investigators can get lifetime tracking of a vortex, which helps to identify properties of coherent vortex such as TKE, volume and aspect ratio. They visualized the shape of a vortex from birth to break up. The variance of captured TKE with the variance of threshold values was also given.

Later, a normalized Q_n criterion was introduced [99]. The Q is normalized by rotation squared. The simulation was still performed in a pipe, the threshold value of Q_n was selected as 0.1 to capture as many structures as possible. Considering the fact that vortices can grow up after absorbing turbulent kinetic energy (TKE), they introduced morphological methods to simulate as well as track the growth process and Biot-Sawart law to avoid overgrowth. Properties of vortices and the probability distribution of TKE are found to be

insistent with the dynamic Smagorinsky-Lilly subgrid scale model.

The Lagrangian method is also a popular method which is used to extract coherent structures. Haller and Yuan [103] proposed that the vital difference between Lagrangian method and Eulerian method such as Q criterion is that the Lagrangian method focuses on the boundaries of structures instead of the transitory properties. They defined material lines to depict boundaries of structures. Moreover, a scrupulous structure extraction criterion was also provided. They performed this analytic criterion on a two-dimensional barotropic turbulence flow for validation. The structures gleaned by them were found to be in agreement with those derived from statistics.

Green et al [104] captured coherent structures using this Lagrangian method in an isolated hairpin vortex flow and a fully developed turbulent flow. The results were then compared to those obtained from Eulerian method - the Q criterion. The Reynolds number was 180. By calculating the direct Lyapunov exponent (DLE), they obtained Lagrangian coherent structures with better detail. Moreover, the extraction was found to be definitely independent by using Lagrangian method while the extraction procedure was determined by the threshold value using Eulerian method. However, the Lagrangian method needs more computational calculation due to the larger amount of data involved in.

Farge and Schneider [100] proposed Coherent Vortex Simulation (CVS) method based on Ha Minh's semi-deterministic model. The principle of CVS is computing the coherent structures, which is the organized part of a flow, and modeling the incoherent turbulent environment, which is the random part of a flow. The implementation of CVS method depends on the type of turbulent flow owing to the nonlinear filtering pass. They performed CVS method on

two-dimensional mixing layer flow and wavelet forced homogeneous isotropic turbulent flow, the results were in consonance with those obtained by DNS.

Farge et al [101] also extracted coherent structures from three dimensional homogeneous isotropic turbulent flow at $Re_\lambda = 150$ using CVS method, the computational domain consists of 256^3 grids. They realized that the classical Richardson's scenario that the transmission of energy is accompanied by the breakup of eddies is not able to demonstrate the energy cascade. Thus, they claimed that the transfer of energy is the result of nonlinear vortex interactions. The coherent energy transfers through the whole of inertial range. As for the incoherent energy, it exists at the whole of inertial scales but dissipation just occurs at the smallest scales.

Schneider et al [102] implement CVS method in three dimensional forced and unforced turbulent mixing layers. The Taylor microscale Reynolds number was 150. The coherent structures gained by orthogonal wavelet basis was represented by 3.8% of the wavelets but includes more than 99% of turbulent kinetic energy in forced mixing layer. And in the unforced mixing layer simulation, the obtained coherent structures was represented by 4.2% of the wavelet. Moreover, the comparison of CVS filtering and LES filtering such as the low-pass Fourier cut-off filtering was given.

Turiel et al [105] studied extraction of vortex structures using CVS in an oceanic flow. They realized that the loss of energy in oceanic coherent flows is relatively high. Furthermore, the incoherent part obtained was not in Gaussian distribution, which made the wavelet decomposition unreal.

Other efforts on turbulent coherent structures were also made in last decades. In 2001, Miyauchi and Tanahashi [106] studied the generalization of scaling law of structure by comparing vortices in homogeneous isotropic turbulence, turbu-

lence mixing layer, turbulent channel flows and MHD homogeneous turbulence at the fine scale. Velocities such as azimuthal, axial and advection velocity as well as length scales such as Kolmogorov microscale, Taylor microscale, and integral length scale were analyzed as the characteristic indicators. The probability density function of the diameter of eddy and circulation were also given to analyzing properties.

Samanta et al [107] performed DNS in both viscoelastic and Newtonian fluids turbulent flow in a parallel-plate channel. The Reynolds number was 180. They used Karhunen-Loeve (K-L) analysis to study the time evolution of coherent structures.

Experimental studies were also performed to analyze coherent structures. Grulke et al [108] did an experimental study of coherent structures in turbulent fluctuations of a simple magnetized torus (SMT) device. They found that the physical properties of plasma can decide the shape of coherent structures.

Staplehurst et al [109] did an experimental study for large-scale columnar structures based on the structure formation theory proposed by Davidson et al [110]. The experiment was performed in a homogeneous freely decaying rotating turbulent flow with $Ro \approx 1$. They introduced two-point correlations in the axial direction to study columnar structures. The structure formation was found to be validated.

These references reveal that coherent structures are important in the study of drop behavior in liquid-liquid systems. To visualize coherent structures, the normalized Q_n criterion is the best method for this study due to its relatively low computational cost, meanwhile, it removes the limitation resulted from the order effect that occurs in the widely used method - Q criterion.

1.5 Objectives and goal of the thesis

The goal of this study is to develop various tools for analysis of data obtained from the direct numerical simulations (DNSs) of liquid-liquid dispersions in homogeneous isotropic turbulent flow. The results of such analysis will then be used to develop well-grounded sub-models for PBEs. The DNSs of liquid-liquid dispersions in a homogeneous isotropic turbulent flow field were performed using the diffuse interface free energy lattice Boltzmann method (LBM). The multi-relaxation-time (MRT) collision operator was used for the collision step. First, the fully-developed turbulence was generated in a three-dimensional fully-periodic cubic domain using a forcing method. Then, a drop of different size and viscosity ratio was injected into the turbulent flow field. The evolution of liquid-liquid interface and coherent structures was visualized. The statistical characteristics of turbulence were also presented.

The underlying objectives of the study are to

- Identify the fundamental assumption used to derive the drop breakup kernels that significantly affect the drop size distribution. For instance, most of the models preassume that drop breaks after interaction with a *single* turbulent vortex. There is no either experimental or numerical evidence for this. Another basic question is whether vortices of the size greater than the drop diameter lead to drop breakup. Most of the existing models are based on the assumption that only vortices of the size equal to or smaller control drop breakup.

- Analyze the statistical characteristics of turbulence. For example, the probability density distribution of vorticity.

- Derive the calculation of three-dimensional spectrum of turbulence and develop the tool for calculation.
- Implement the multi-relaxation-time collision operator for the lattice Boltzmann method to improve the stability of simulations.
- Find an evidence for or disprove the outlined assumptions by visualization and quantification of the results of high-resolution three-dimensional DNSs of liquid-liquid dispersion in homogeneous isotropic turbulence.

Chapter 2

DNSs of liquid-liquid systems

In Section 2.1, a literature review focused on direct numerical simulations of turbulent liquid-liquid systems is presented. In Section 2.2, the free energy lattice Boltzmann method is introduced, two different collision operators are compared. In Section 2.3, the generation of single phase turbulent flow is given. In Section 2.4, the calculation of three-dimensional energy spectra is presented. In Section 2.5, the normalized Q_n criterion and newly designed boundary identification method are introduced for the visualization of coherent structures.

2.1 Literature review on DNSs of turbulent liquid-liquid systems

Only studies of turbulent liquid-liquid systems using DNSs are reviewed here. Due to the limited number of studies in this area, it is important to discuss them in order to verify and validate the numerical methods that are used for complex systems.

Derksen and Akker [111] simulated liquid-liquid dispersion in stirred three-dimensional periodic domain. Two fluids with the same viscosity and density were set in the simulation. The turbulent flow was generated by the lattice Boltzmann method (LBM) in a 256^3 cells domain. The initial droplet diameter was 20 lattice units. The dispersed phase volume fraction was 16%. The fluid was turbulent at the beginning, its turbulent kinetic energy (TKE) decayed quickly after stepping into a short steady balance. They provided coalescence and breakup process of droplets in cross-section view. The drop size distributions (DSD) was also given. The DSD became wider with time, and more small droplets were generated. It revealed that the breakup process was dominant. To contrast, the coalescence process dominates if there was no turbulent force injected into the fluid.

Komrakova et al [112] explored drop deformation and breakup in liquid-liquid simple shear flow using LBM. Five dimensionless numbers, Reynolds number Re, capillary number Ca, viscosity ratio λ, Peclet number Pe and Cahn number Ch, were introduced to fully describe the liquid-liquid system. The impact of Pe, Ch and mesh resolution on the mechanism of deformation was studied. In 2015, Komrakova et al [113] studied the effect of Kolmogorov scale resolution, energy input, viscosity ratio and dispersed phase volume frac-

tion on deformation of droplets in a three-dimensional periodic domain. The homogeneous isotropic turbulent flow was created by linear forcing. The comparison of kinetic energy spectrum in single-phase and two-phase was also given. The Introduction of the second phase changes the energy spectrum fiercely. They also highlighted the three drawbacks of the numerical method they used: the numerical dissolution of small drops, over-estimation of drop coalescence and the occurrence of spurious currents.

Hagiwara et al [114] investigated drop deformation in a turbulent channel flow domain which was meshed as $64 \times 44 \times 32$ by DNSs. Classification of five different types of turbulent structures were provided: shear, convergence, eddy, donor eddies and streaming. They were classified using the zone classification method given in [115]. They found that the droplet was sharpened in the shear-dominant region while squeezed in the convergence-dominant region. The deformation in the eddy-dominant region was not obvious.

Scarbolo and Soldati [116] investigated drop dynamic in turbulent channel flow using DNS combined with interface tracking. The shear Reynolds number was $Re_\tau = 100$ and Peclet number was $Pe = 2.56 \times 10^5$. The four different Weber numbers were $We = 0.0053, 0.0106, 0.0212, 0.0424$. The ratio between Kolmogorov length scale and drop diameter was from 0.06 to 0.13. The simulation domain was divided into $256 \times 128 \times 129$ meshes. They found the existence of interface generated discrepancy of surrounding velocity field, which led to the vorticity generation. The figures that drop surrounded by coherent structures were also given, the drop with low deformability was surrounded by more coherent structures.

Roccon et al [117] studied viscosity effect on the breakup and coalescence of drops in a wall-bounded turbulent flow. The $4\pi h \times 2\pi h \times 2h$ domain was

divided into $512 \times 256 \times 257$ meshes. Turbulent flows with Weber number $We = 0.75, 1.5, 3$ were generated, the viscosity ratio between dispersed phase and continuous phase were $\lambda = 0.01, 0.1, 1, 10, 100$ for each Weber number. 256 drops with diameter $d = 90$ meshes were injected into the turbulent flow at $Re_\tau = 150$. The two-phase flow was tracked by the phase field method (PFM). The average velocity of the mixture of two phases was observed always to be increased in time due to the absorption of turbulent kinetic energy by drops. Furthermore, the drop was not likely to deform at small Weber number $We < 1$ because the surface tension, which prevents the drop breakage, was dominant, and the change of viscosity ratio didn't affect drop breakup or coalescence rate in this situation. In the contrast, drop tends to deform at larger Weber number $We > 1$, and the increment of viscosity ratio drastically decreased the breakup rate and increased the coalescence rate.

The DNS of drop/near-wall turbulence interaction was reported by Iwasaki [118] et al. The simulation was resolved in a rectangular domain. A drop of diameter equals to one-fourth of the wall distance was initially placed in the range of 20 - 60 wall units from one moving wall. To investigate effects of viscosity μ_d/μ_c and interfacial tension $\sigma^* = \sigma/\rho U_W^2 h$ of drop (where U_W^2 was the speed of moving wall, h was half of distance between two moving walls), three different cases were implemented: the fluid element $\mu_d/\mu_c = 1$, $\sigma^* = 0$; the viscous drop without interfacial tension $\mu_d/\mu_c = 40$, $\sigma^* = 0$; the viscous drop with interfacial tension $\mu_d/\mu_c = 40$, $\sigma^* = 0.01$. An equal density was set for dispersed and continuous phase. The Reynolds numbers $Re^* = U_W h/\nu$ and $Re^+ = u_\tau h/\nu$ were 1300 and 82.6 respectively (where u_τ was the friction velocity). The liquid-liquid interface was tracked by VOF method. They found that the deformation of high-viscosity drop was smaller compared to

the deformation of fluid element; for the high-viscosity drop with interfacial tension, the deformation nearly did not occur. Thus, both the high-viscosity and interfacial tension suppress the deformation. They also extracted coherent structures at near-wall region, the existence of drop with interfacial tension was found to result in the attenuation of near-wall streamwise vortex. Besides, in the wake flow region of drop, the occurrence of small vortex was observed. Finally, in a wide region around the drop, the Reynolds-shear stress product was observed to be higher than that in other region. They attributed this phenomenon to the introduced translational flow by the drop.

Yuge and Hagiwara [119] studied non-isothermal turbulent upward channel flow by DNS. The hydrofluoroether-water binary system was used. The Reynolds number Re_τ was 180. Four identical drops were placed at near-wall region and tracked by VOF method. Some conclusions were provided: several types of secondary flows introduced by drops resulted in the increase of Reynolds shear stress product, they were mitigated by the adjacent drops in streamwise direction; The near-wall drops deformed large streamwise vortices and mitigated small vortices; The drops increased heat transfer in binary system.

Combining with various liquid-liquid interface tracking methods, the implementation of DNS simulations helps exploring and explaining numerous fundamental behavior and phenomena in turbulent binary systems that contain multi-scales.

2.2 Free energy lattice Boltzmann method with multiple-relaxation-time operator

The diffuse interface free energy lattice Boltzmann method (LBM) proposed by Swift et al. [120] is used here. In diffuse interface methods [121, 122, 123], the interface is defined as a finite-thickness transition area where the variance of physical quantities is continuous. To describe composition of system, the order parameter φ is introduced [124, 125, 126]. It is regarded as the relative concentration of two components. The solution of continuity and momentum equations in conjunction with the Cahn-Hilliard convection-diffusion equation for the order parameter is adopted to simulate behavior of binary mixture [127]. Therefore, the continuity, momentum, convection-diffusion equations decide density, velocity and order parameter respectively [128].

$$\partial_t \rho + \partial_\alpha(\rho u_\alpha) = 0 \tag{2.1a}$$

$$\partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta) = \tag{2.1b}$$

$$-\partial_\beta P^{th}_{\alpha\beta} + \partial_\beta \nu(\rho\partial_{\alpha u_\beta} + \rho\partial_\beta u_\alpha) + \rho F_{t\alpha}$$

$$\partial_t\varphi + \partial_\alpha(\varphi u_\alpha) = M\partial^2_{\beta\beta}\mu \tag{2.1c}$$

where u_α is the velocity; the index α stands for the Cartesian directions x, y, z; ρ and ν are the density and the kinematic viscosity of the mixture, respectively; M is the mobility; $F_{t\alpha}$ is the forcing term to generate turbulence; $P^{th}_{\alpha\beta}$ is the "thermodynamic" pressure tensor which includes an isotropic contribution $P_{id}\delta_{\alpha\beta}$ that represents the ideal gas pressure and the "chemical" pressure tensor

$P^{chem}_{\alpha\beta}$ [128]: $P^{th}_{\alpha\beta} = P_{id}\delta_{\alpha\beta} + P^{chem}_{\alpha\beta}$. The ideal gas pressure is $P_{id} = \frac{\rho}{c_s^2}$,where c_s^2 is sound speed. The $P^{chem}_{\alpha\beta} = \left[\frac{A}{2}\varphi^2 - \frac{3A}{4}\varphi^4 - K\varphi\partial^2_{\alpha\alpha}\varphi - \frac{1}{2} \mid \partial_\alpha\varphi \mid^2\right] + K\partial_\alpha\varphi\partial_\beta\varphi$ is an active scalar and the set of Equation (2.1) is intimately coupled, because it is a function of order parameter φ [113]. $\mu(\varphi) = A\varphi - A\varphi^3 - K\partial^2_{\alpha\alpha}\varphi$ is the chemical potential. Here, $A < 0$ and K are parameters of the free energy model that are related to the surface tension and interface thickness.

In LBM model, the macroscopic equations are solved in two steps: streaming and collision [129]. At each time step, particles stream to neighboring lattice points along fixed lattice links in streaming process, then the velocity distribution at each point relaxes towards equilibrium distribution in collision process. Two discrete single-particle density distribution functions $f(\mathbf{r}, t)$ and $g(\mathbf{r}, t)$ are used to describe the motion of fluid. The function $f(\mathbf{r}, t)$ solves the continuity and momentum equations which are the Equation (2.1a) and (2.1b) respectively, the function $g(\mathbf{r}, t)$ solves the convection-diffusion equation which is the Equation (2.1c).

Usually, the single-relaxation-time collision operator, also known as the Bhatnagar-Gross-Krook (BGK) operator [130], is adopted to solve the collision process. It is the simplest and widely used collision operator because distribution functions are relaxed by a single dimensionless relaxation parameter τ. Under this scenario, the discrete lattice Boltzmann equations are given as [131]:

$$\begin{aligned} f_q\left(\mathbf{r}_\alpha + \mathbf{c}_{\alpha q}\Delta t, t + \Delta t\right) - f_q\left(\mathbf{r}_\alpha, t\right) &= \frac{-1}{\tau_f}\left(f_q - f_q^{eq}\right) + \mathbf{F}_q \\ g_q\left(\mathbf{r}_\alpha + \mathbf{c}_{\alpha q}\Delta t, t + \Delta t\right) - g_q\left(\mathbf{r}_\alpha, t\right) &= \frac{-1}{\tau_g}\left(g_q - g_q^{eq}\right) \end{aligned} \tag{2.2}$$

where index q represents the number of discrete velocity directions; index α

represents the Cartesian directions x, y and z; f_q^{eq} and g_q^{eq} are the discretized Maxwell-Boltzmann distribution functions (equilibrium distribution function); τ_f and τ_g are dimensionless relaxation parameters; $\mathbf{c}_{\alpha q}$ is discrete velocity set and $\mathbf{F}_q$ is the forcing term. The forcing term is incorporated as follows:

$$\mathbf{F}_q = w_q(\mathbf{c}_q \cdot \mathbf{F}_{t\alpha}) \tag{2.3}$$

The density ρ, local fluid momentum ρu_α and order parameter φ are defined as following:

$$\begin{aligned} \sum_q f_q &= \rho \\ \sum_q c_{\alpha q} f_q &= \rho u_\alpha + \frac{F_{t\alpha}}{2} \\ \sum_q g_q &= \varphi \end{aligned} \tag{2.4}$$

The calculation of equilibrium equations is given by [132]:

$$\begin{aligned} f_q^{eq} =& \frac{w_q}{c^2}\left(p_0 - K(\partial_{xx}^2\varphi + \partial_{yy}^2\varphi + \partial_{zz}^2\varphi) + c_{\alpha q}\rho u_\alpha + \frac{3}{2c^2}\left[c_{\alpha q}c_{\beta q} - \frac{c^2}{3}\delta_{\alpha\beta}\right]\rho u_\alpha u_\beta\right) \\ &+ \frac{K}{c^2}(w_q^{xx}\partial_x\varphi\partial_x\varphi + w_q^{yy}\partial_y\varphi\partial_y\varphi + w_q^{zz}\partial_z\varphi\partial_z\varphi \\ &+ w_q^{xy}\partial_x\varphi\partial_y\varphi + w_q^{xz}\partial_x\varphi\partial_z\varphi + w_q^{yz}\partial_y\varphi\partial_z\varphi) \qquad (2.5a) \\ g_q^{eq} =& \frac{w_q}{c^2}\left(\Gamma\mu + c_{\alpha q}\varphi u_\alpha + \frac{3}{2c^2}\left[c_{\alpha q}c_{\beta q} - \frac{c^2}{3}\delta_{\alpha\beta}\right]\varphi u_\alpha u_\beta\right) \qquad (2.5b) \end{aligned}$$

where w_q is weight corresponding to the norm of c_q, c_s is the speed sound. p_0 is the bulk pressure $p_0 = c_s^2\rho + \frac{A}{2}\varphi^2 + \frac{3B}{4}\varphi^4$. The distributions for $q = 0$ are

given by:

$$f_0^{eq} = \rho - \sum_{q=1}^{18} f_q^{eq}$$
$$g_0^{eq} = \varphi - \sum_{q=1}^{18} g_q^{eq} \tag{2.6}$$

In this study, the D3Q19 lattice arrangement (three dimensions and nineteen discrete velocity directions) is adopted. As is shown in Figure 2.1, the central point is connected with 6 central points in each face and 12 central points in each edge of the cube. Only uniform cubic lattices are used here, the mesh step Δx and time step Δt are taken as unity. Lattice Boltzmann method operates in lattice space in so-called lattice units (lu).

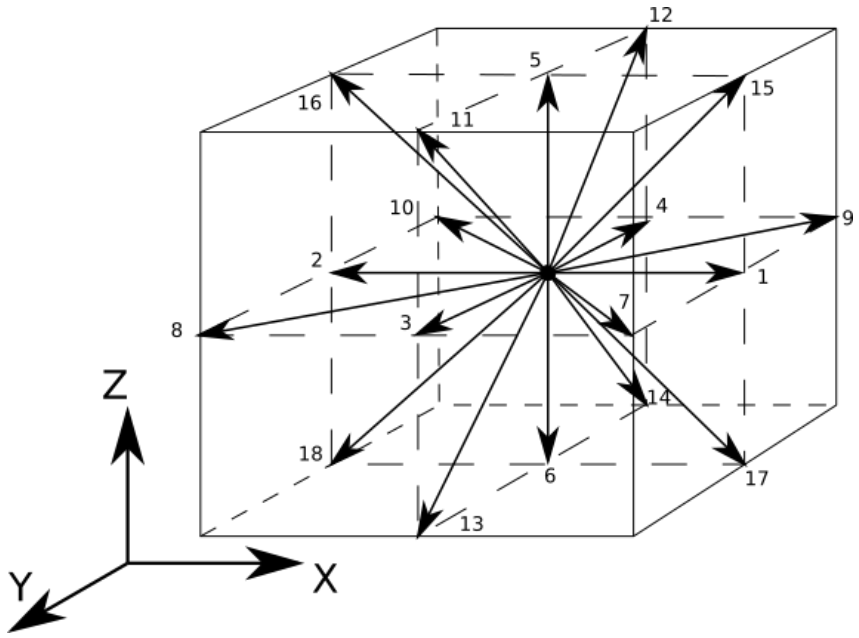


Figure 2.1: D3Q19 lattice arrangement

The discrete velocity sets and the weighting factors are defined as follows:

$$e_q = \begin{cases} (0,0,0) & q = 0 \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm 1) & q = 1,2,...,6 \\ (\pm 1,\pm 1,0),(0,\pm 1,\pm 1),(\pm 1,0,\pm 1) & q = 7,8,...,18 \end{cases} \tag{2.7}$$

and

$$w_q = \begin{cases} 1/3, & q = 0 \\ 1/18, & q = 1,2,...,6 \\ 1/36, & q = 7,8,...,18 \end{cases} \tag{2.8}$$

The multiple-relaxation-time (MRT) collision operator is implemented considering that multiple physical quantities in collision process may relax on different time scales [131]. It is used in this study to improve stability and accuracy of the simulations. Premnath et al. [133] analyzed advantageous of MRT operator by comparing with BGK operator. They proposed that numerical instability occurs in the relatively low viscosities fluids simulation. Due to computational constraints, the instability problems may be compounded in three-dimensional flows when physics may not be adequately resolved. The MRT operator is more stable than BGK operator because the different relaxation times can be individually relaxed to achieve stability. The MRT operator is also flexible enough to incorporate additional physics which cannot be naturally represented by BGK operator. In MRT operator, moments of distribution functions such as momentum and viscous stresses are solved directly, which provides a natural and convenient way to express various relaxation processes happened in various time scales due to collisions.

Therefore, the lattice Boltzmann equation which utilizes the MRT operator is given by:

$$f_q\left(\mathbf{r}_\alpha + \mathbf{c}_{\alpha q}\Delta t, t + \Delta t\right) - f_q\left(\mathbf{r}_\alpha, t\right) = -\boldsymbol{M}^{-1}\boldsymbol{S}(\boldsymbol{m}_q - \boldsymbol{m}_q^{eq}) + \mathbf{F}_q \tag{2.9}$$

where $\boldsymbol{M}$ is a 19×19 transformation matrix which transforms distribution functions f to velocity moments $\boldsymbol{m}$:

$$\boldsymbol{m} = \boldsymbol{M}f \tag{2.10}$$

Define collision matrix $\hat{\boldsymbol{S}}$ as follow:

$$\hat{\boldsymbol{S}} = \boldsymbol{M}^{-1}\boldsymbol{S}\boldsymbol{M} \tag{2.11}$$

Therefore, the Equation (2.9) can be replaced by:

$$f_q\left(\mathbf{r}_\alpha + \mathbf{c}_{\alpha q}\Delta t, t + \Delta t\right) - f_q\left(\mathbf{r}_\alpha, t\right) = -\hat{\boldsymbol{S}}(f_q - f_q^{eq}) + \mathbf{F}_q \tag{2.12}$$

The eigenvalues of S are all between 0 and 2 in order to maintain linear stability and the separation of scales [134]. It can be easily represented that BGK operator is a special case in which all relaxation times are equal, the collision matrix is $\hat{\boldsymbol{S}} = \boldsymbol{M}^{-1}\frac{1}{\tau}\boldsymbol{I}\boldsymbol{M} = \frac{1}{\tau}diag(\boldsymbol{I})^T$, where $\tau > \frac{1}{2}$ is the single relaxation time of BGK.

The transformation matrix represents components of the 19 orthogonal basis column vectors:

$$\boldsymbol{M} = \begin{bmatrix} \varsigma_0, \varsigma_1, \ldots, \varsigma_{18} \end{bmatrix}^T \tag{2.13}$$

The 19 orthogonal basis vectors $\varsigma_{q\beta}$ and derived transformation matrix $\boldsymbol{M}$ are listed in Appendix. The corresponding 19 velocity moments are given [135]:

$$\boldsymbol{m} = (\rho, e, e^2, j_x, q_x, j_y, q_y, j_z, q_z, 3p_{xx}, 3\pi_{xx}, p_{ww}, \pi_{ww}, p_{xy}, p_{yz}, p_{xz}, m_x, m_y, m_z)^T \tag{2.14}$$

where j_x, j_y, j_z are momentum or mass flux ($\boldsymbol{j} = \rho\boldsymbol{u}$); q_x, q_y, q_z are heat flux; p_{xx}, p_{ww}, p_{xy}, p_{yz}, and p_{xz} are components of the symmetric and traceless strain-rate tensor; π_{xx} and π_{ww} are fourth order moments which have the same symmetry as the diagonal part of the traceless tensor p_{ij}. The last three vectors are third order moments.

In MRT operator, the collision matrix (relaxation matrix) $\hat{\boldsymbol{S}}$ is diagonal in moment space $\mathbb{M}$:

$$\hat{\boldsymbol{S}} = diag(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}, s_{17}, s_{18})^T \tag{2.15}$$

where parameter s_q represent the inverse of the relaxation time $(1/\tau)$ of the various moments $\boldsymbol{m}$ in reaching their equilibrium values $\boldsymbol{m}^{eq}$. Because the collision process conserves particular moments (density and components of momentum) s_0, s_3, s_5 and s_7, the collision process values remain constant under variation of corresponding relaxation times. Thus, the relaxation parameters for these moments can be set to zero when there is no forcing term in the LBM. However, the collision matrix affects the forcing term in the effectively explicit in MRT LBM with forcing term, the relaxation times for these conserved moments should not be chosen as zero [133]. Therefore, they are set

as 1 for simplicity. The collision matrix then becomes:

$$\hat{\boldsymbol{S}} = diag(1, s_e, s_{e^2}, 1, s_q, 1, s_q, 1, s_q, s_\nu, s_\pi, s_\nu, s_\pi, s_\nu, s_\nu, s_\nu, s_m, s_m, s_m)^T \quad (2.16)$$

The equilibria of the non-conserved moments are given:

$$\begin{aligned}
m_1^{eq} &= -11\rho + \frac{19}{\rho}(j_x^2 + j_y^2 + j_z^2) \\
m_2^{eq} &= \omega_\varepsilon + \frac{\omega_{\varepsilon j}}{\rho}(j_x^2 + j_y^2 + j_z^2) \\
m_{4,6,8}^{eq} &= -\frac{2}{3} j_{x,y,z} \\
m_9^{eq} &= \frac{1}{\rho}(2j_x^2 - j_y^2 - j_z^2) \\
m_{11}^{eq} &= \frac{1}{\rho}(j_y^2 - j_z^2) \\
m_{10}^{eq} &= \omega_{xx} m_9^{eq} \\
m_{12}^{eq} &= \omega_{xx} m_{11}^{eq} \\
m_{13}^{eq} &= \frac{1}{\rho} j_x j_y \\
m_{14}^{eq} &= \frac{1}{\rho} j_y j_z \\
m_{15}^{eq} &= \frac{1}{\rho} j_z j_x \\
m_{16}^{eq} &= m_{17}^{eq} = m_{18}^{eq} = 0
\end{aligned} \quad (2.17)$$

In order to optimize the linear stability of the model, the parameters are chosen as [136]: $\omega_\varepsilon = \omega_{xx} = 0$, $\omega_{\varepsilon j} = -475/63$, $s_e = 1.19$, $s_e^2 = s_\pi = 1.4$, $s_q = 1.2$, $s_m = 1.98$. The speed sound is $c_s = 1/\sqrt{3}$ in lattice units of

$\Delta x = \Delta t = 1$. The kinematic viscosity ν and bulk viscosity ξ are:

$$\nu = \frac{1}{3}\left(\frac{1}{s_\nu} - \frac{1}{2}\right) \tag{2.18}$$

$$\xi = \frac{2}{9}\left(\frac{1}{s_e} - \frac{1}{2}\right) \tag{2.19}$$

The continuous phase and injected dispersed phase are of different kinematic viscosity. Thus, the kinematic viscosity of the mixture is a function of order parameter [132]:

$$\nu(\varphi) = \nu_c \frac{\varphi_0 - \varphi}{2\varphi_0} + \nu_d \frac{\varphi_0 + \varphi}{2\varphi_0} \tag{2.20}$$

where ν_c and ν_d are the kinematic viscosities of continuous phase and dispersed phase, respectively. φ_0 is set as 1.

2.3 Turbulence generation. Single-phase flow simulations

In this study, the statistically stationary homogeneous isotropic turbulence is generated in a three-dimension periodic domain. Due to the natural decay of turbulence, the energy input is the necessity for the maintenance of turbulence. Here, the energy input is organized by means of linear forcing that is proposed by Lundgren [137]. The local force imposed in the liquid is proportional to the local velocity

$$F_{t\alpha} = A_f u_\alpha \tag{2.21}$$

The parameter of force A_f is

$$A_f = \varepsilon/(3u_{rms}^2) \tag{2.22}$$

where ε is the volume-averaged energy dissipation rate per unit mass; u_{rms} is the volume-averaged root-mean-square fluid velocity. The dissipation rate ε is an input parameter in turbulence generation, if the resolution of the Kolmogorov length scale η_K is fixed, then these levels of dissipation can be set up by varying kinematic viscosity of the continuous phase ν_c due to the relationship $\eta_K = (\nu_c^3/\varepsilon)^{1/4}$. The turbulence generation method does not allow to have the root-mean-square velocity u_{rms} as an input parameter. therefore, as the output parameter, the u_{rms} values were calculated every time step after the simulation was performed. Derksen [138], Valino et al. [139] successfully implemented this type of linear forcing in the environment of the LBM.

A non-zero velocity field is used to start with the turbulence generation. It was initialized at $t = 0$ using the following relations

$$\begin{aligned} u_x &= u_{init} sin\left(\frac{2\pi j}{\lambda_0}\right) \\ u_y &= u_{init} sin\left(\frac{2\pi k}{\lambda_0}\right) \\ u_z &= u_{init} sin\left(\frac{2\pi i}{\lambda_0}\right) \end{aligned} \tag{2.23}$$

where $\lambda_0 = 1.01L_d/4$, $i = j = k = (1 : L_d)$ (corresponding to x, y, and z, respectively); $u_{init} = 5u_K$ is the maximum velocity in the initial distribution; L_d is the domain edge size; u_K is the Kolmogorov velocity scale. This flow field is divergence free.

To achieve statistical stationary quantities, it is necessary to run simula-

tions for a longer period of time. Therefore, before injecting the dispersed phase, all simulations have to be conducted long enough to ensure time-invariant statistics. It ensures the full development of single phase turbulent flow [113].

2.4 Three-dimensional energy spectrum

According to Kolmogorov's theory [140], the kinetic energy enters the turbulent flow at the largest scales of motion. With the breakup of eddies, the kinetic energy is continuously transferred to smaller scales until the Kolmogorov scale is reached. At this smallest length scale, the eddy motion is stable because the Reynolds number is sufficiently small, and the kinetic energy is dissipated due to viscous effect. This is also known as 'energy cascade'. The energy cascade consists of two subranges: inertial subrange and dissipation subrange. In inertial subrange, the viscous effects are negligible, the inertial effect determines the dominant energy transfer from large to small eddies. The energy cascade in this subrange is described by:

$$E(\kappa) = C\varepsilon^{2/3}\kappa^{-5/3} \tag{2.24}$$

where C is a universal Kolmogorov constant, ε is the energy dissipation rate, κ is the wave number. This spectrum is also referred to as the Kolmogorov spectrum. In the dissipation subrange, the viscous effect is dominant, the energy is dissipated.

Here gives the derivation of calculation of three-dimensional energy spectrum. Considering two points i and j with distance $\boldsymbol{r}$ in Cartesian coordinates,

where $\boldsymbol{r}$ is the space vector between two points. The derivation of three dimensional energy spectra calculation presented below is based on [141, 142, 143]. In homogeneous isotropic turbulent flow, the correlation function between velocities of two measured points is given by [141]:

$$R_{ij}(\boldsymbol{r}) = \langle u_i(\boldsymbol{x}) u_j(\boldsymbol{x} + \boldsymbol{r}) \rangle \tag{2.25}$$

where the angle brackets denote ensemble averaging, u_i and u_j are velocities at two points, $\boldsymbol{x}$ is the location in Cartesian coordinates.

A spectrum is defined as the Fourier transforms of a correlation function. Thus, a pair of forward Fourier transform and inverse Fourier transform is [141]:

$$\tilde{f}(\kappa) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(r) e^{-i\kappa r} dr \tag{2.26}$$

$$f(r) = \int_{-\infty}^{\infty} \tilde{f}(\kappa) e^{i\kappa r} d\kappa \tag{2.27}$$

A three-dimensional spectrum is $\Phi_{ij}(\boldsymbol{\kappa})$ calculated as [141]:

$$\Phi_{ij}(\boldsymbol{\kappa}) = \frac{1}{8\pi^3} \iiint_{-\infty}^{\infty} R_{ij}(\boldsymbol{r}) e^{-i\boldsymbol{\kappa r}} d\boldsymbol{r} \tag{2.28}$$

where $\boldsymbol{\kappa} = (\kappa_1, \kappa_2, \kappa_3)$, and $\kappa = |\boldsymbol{\kappa}|$.

Inversely:

$$R_{ij}(\boldsymbol{r}) = \iiint_{-\infty}^{\infty} e^{i\boldsymbol{\kappa r}} \Phi_{ij}(\boldsymbol{\kappa}) d\boldsymbol{\kappa} \tag{2.29}$$

When $i = j$, we get:

$$R_{ii}(0) = \langle u_i^2 \rangle = \iiint_{-\infty}^{\infty} \Phi_{ii}(\boldsymbol{\kappa}) d\boldsymbol{\kappa} \tag{2.30}$$

The total turbulent kinetic energy E_{tke} is an integrity of energy spectrum $E(\kappa)$. For simplicity, the direction of the Fourier modes is often removed because it has no effect on the calculation. Thus, the length of of the wave number vector instead of the vector itself is used: $\kappa = \mid \boldsymbol{\kappa} \mid = (\kappa_1^2 + \kappa_2^2 + \kappa_3^2)^{1/2}$. This is obtained by integrating over the spherical shell $S(\kappa)$, where κ denotes the radius of the sphere. Figure 2.2 represents the shell of radius κ, the surface denotes the energy spectrum (κ) of a specific radius κ. Therefore, the total turbulent kinetic energy E_{tke} is regarded as the sum of various (κ) which are denoted by diferent surfaces of different radius κ.

Combining with the Equation (2.30), we get:

$$\begin{aligned} E_{tke} &= \int_0^{\infty} E(\kappa) d\kappa \\ &= \frac{1}{2} \langle u_i^2 \rangle \\ &= \iiint_{-\infty}^{\infty} \frac{1}{2} \Phi_{ii}(\boldsymbol{\kappa}) d\boldsymbol{\kappa} \\ &= \int_0^{\infty} \oint \frac{1}{2} \Phi_{ii}(\boldsymbol{\kappa}) dS(\kappa) d\kappa \end{aligned} \tag{2.31}$$

From Equation (2.31), the relationship is obtained:

$$\begin{aligned} E(\kappa) &= \oint \frac{1}{2} \Phi_{ii}(\boldsymbol{\kappa}) dS(\kappa) \\ &= \frac{1}{2} \Phi_{ii}(\boldsymbol{\kappa}) 4\pi\kappa^2 \\ &= 2\pi\kappa^2 \Phi_{ii}(\boldsymbol{\kappa}) \end{aligned} \tag{2.32}$$

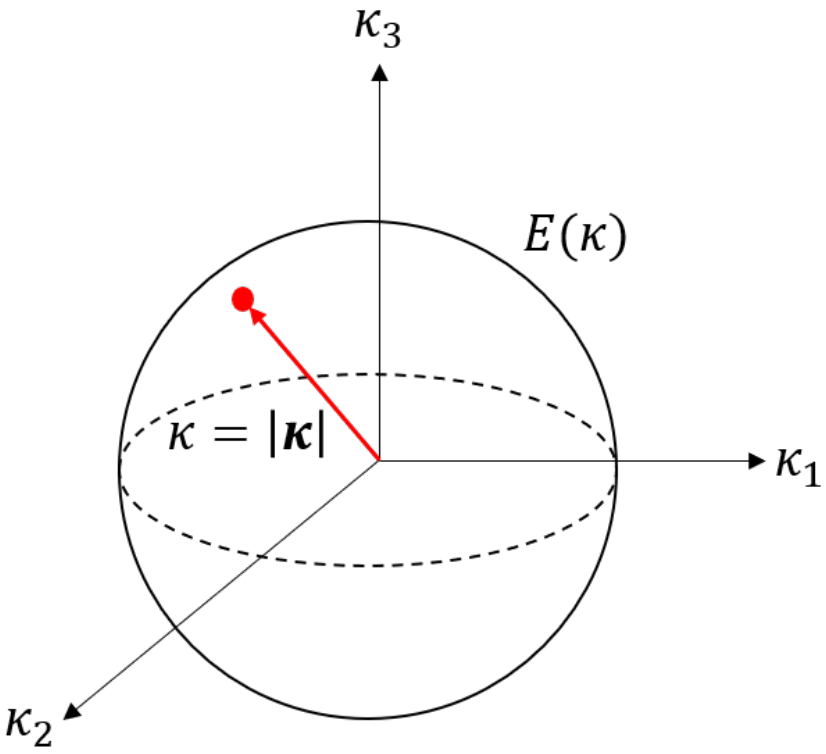


Figure 2.2: Three-dimensional shell ingeration

In Equation (2.31), we can also get:

$$\begin{aligned} E_{tke} &= \iiint_{-\infty}^{\infty} \frac{1}{2}\Phi_{ii}(\boldsymbol{\kappa})d\boldsymbol{\kappa} \\ &= \frac{1}{2}\sum_{\boldsymbol{\kappa}} \Phi_{ii}(\boldsymbol{\kappa})\Delta\kappa_1\Delta\kappa_2\Delta\kappa_3 \end{aligned} \tag{2.33}$$

The total turbulent kinetic energy can be also calculated by:

$$\begin{aligned} E_{tke} &= \sum_{\kappa} E(\kappa) \\ &= \frac{1}{2}\langle\tilde{u}^{\star}(\boldsymbol{\kappa})\tilde{u}(\boldsymbol{\kappa})\rangle \end{aligned} \tag{2.34}$$

Combining with the Equation (2.33) and (2.34), the calculation of Φ_{ii} is given (in this study, the interval of wave number is the same):

$$\Phi_{ii} = \frac{\langle\tilde{u}^{\star}(\boldsymbol{\kappa})\tilde{u}(\boldsymbol{\kappa})\rangle}{(\Delta\kappa)^3} \tag{2.35}$$

Substitute the Equation (2.35) into Equation (2.32):

$$\begin{aligned} E(\kappa) &= 2\pi\kappa^2\Phi_{ii}(\boldsymbol{\kappa}) \\ &= 2\pi\kappa^2\frac{\langle\tilde{u}^{\star}(\boldsymbol{\kappa})\tilde{u}(\boldsymbol{\kappa})\rangle}{(\Delta\kappa)^3} \end{aligned} \tag{2.36}$$

where $\tilde{u}^{\star}(\boldsymbol{\kappa})$ is the conjugate of $\tilde{u}(\boldsymbol{\kappa})$.

To implement the Equation (2.36) in computer code, a process of three-dimensional energy spectrum calculation is proposed:

1. Transfer the three physical space (in Cartesian coordinates) velocity fields $u(x, y, z)$, $v(x, y, z)$, $w(x, y, z)$ to Fourier space velocity fields $\tilde{u}_1(\kappa_1, \kappa_2, \kappa_3)$, $\tilde{u}_2(\kappa_1, \kappa_2, \kappa_3)$, $\tilde{u}_3(\kappa_1, \kappa_2, \kappa_3)$ using Fast Fourier Transfer (FFT), such as the MATLAB function 'fftn'.

2. Calculate $\langle \tilde{u}_i(\kappa_1, \kappa_2, \kappa_3)\tilde{u}_i^\star(\kappa_1, \kappa_2, \kappa_3)\rangle$, the angle brackets denote the ensemble average. The emsemble average of a random variable x is given as $\langle x\rangle = \dfrac{1}{V}\sum_{n=1}^{V} x_n$. Thus, the $\langle \tilde{u}_i(\kappa_1, \kappa_2, \kappa_3)\tilde{u}_i^\star(\kappa_1, \kappa_2, \kappa_3)\rangle$ of different components are:

$$\frac{\sum^{N}\sum^{N}\sum^{N} \tilde{u}_1(\kappa_1, \kappa_2, \kappa_3)\tilde{u}_1^\star(\kappa_1, \kappa_2, \kappa_3)}{(N \times N \times N)^2}$$
$$\frac{\sum^{N}\sum^{N}\sum^{N} \tilde{u}_2(\kappa_1, \kappa_2, \kappa_3)\tilde{u}_2^\star(\kappa_1, \kappa_2, \kappa_3)}{(N \times N \times N)^2}$$
$$\frac{\sum^{N}\sum^{N}\sum^{N} \tilde{u}_3(\kappa_1, \kappa_2, \kappa_3)\tilde{u}_3^\star(\kappa_1, \kappa_2, \kappa_3)}{(N \times N \times N)^2}$$

where N is the number of points along the edge of the domain ($N = 300$ in our case).

3. Establish a Fourier space cube domain, the number allocated to each point along the edge is order in this array: $R = [0, 1, \ldots, N/2, -N/2 + 1, -N/2+2, \ldots, -1]$. This order is implemented to generate three dimensional coordinates defined by vectors X, Y, Z using MATLAB intrinsic function 'meshgrid': $[X, Y, Z] = meshgrid(R, R, R)$. Thus a Fourier space length tensor $r(\kappa_1, \kappa_2, \kappa_3)$ of size N^3 can be obtained in MATLAB, the code is: $r = \sqrt{(X)^2 + (Y)^2 + (Z)^2}$.

4. The length of wavenumber is divided into $N/2$ bins from 1 to $N/2$. Therefore, the number of bins used in calculation is $N/2$.

5. The wavenumber is $k_w = [1, N/2]\triangle x$, where $\triangle x = 2\pi/L$

6. For the pth bin, record all locations $\mathcal{L}(\kappa_1, \kappa_2, \kappa_3)$ of points that satisfy the relationship:

$r(\kappa_1, \kappa_2, \kappa_3)\triangle x \in \left(\frac{k_w(p-1)+k_w(p)}{2}, \frac{k_w(p)+k_w(p+1)}{2}\right]$.

The number of recorded points is $\mathcal{N}$.

7. The energy in each bin p is

$$E(p) = 2\pi {k_w}^2 \left(\frac{\sum \tilde{u}_1(\mathcal{L})\tilde{u}_1^\star(\mathcal{L})}{N^2} + \frac{\sum \tilde{u}_2(\mathcal{L})\tilde{u}_2^\star(\mathcal{L})}{N^2} + \frac{\sum \tilde{u}_3(\mathcal{L})\tilde{u}_3^\star(\mathcal{L})}{N^2}\right) / \left(\mathcal{N}\triangle x^3\right),$$

where $p \in [1, N/2]$

8. Plot energy spectrum $E(p)$ as the function of k_w.

There are two ways to calculate the turbulent kinetic energy, the first one is calculated from Fourier space velocity field:

$$E_v = \frac{1}{2}\langle \tilde{u}_i(\kappa_1, \kappa_2, \kappa_3)\tilde{u}_i^\star(\kappa_1, \kappa_2, \kappa_3)\rangle \tag{2.37}$$

The second one is calculated from the energy spectrum:

$$\begin{aligned} E_e &= \int_0^\infty E(\kappa)d\kappa \\ &= \sum E(\kappa)\Delta\kappa \end{aligned} \tag{2.38}$$

The energy dissipation rate is also calculated in two ways, the first one is the average of local energy dissipation rate of all points:

$$\varepsilon_v = \sum \varepsilon_p / L^3 \tag{2.39}$$

where ε_p is the local energy dissipation rate of each point, it is given by [144]:

$$\varepsilon_p = 2\nu\langle S_{ij}S_{ij}\rangle \tag{2.40}$$

where ν is the kinematic viscosity of fluid, S_{ij} is the strain rate[144]:

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.41}$$

where the velocity gradient tensor is[144]:

$$\frac{\partial u_i}{\partial x_j} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} & \frac{\partial u_1}{\partial x_3} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} & \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_3}{\partial x_1} & \frac{\partial u_3}{\partial x_2} & \frac{\partial u_3}{\partial x_3} \end{bmatrix} \tag{2.42}$$

The second way is based on the energy spectrum [140]:

$$\varepsilon_e = \int_0^\infty 2\nu\kappa^2 E(\kappa) d\kappa \tag{2.43}$$

The compensated energy spectrum is [140]:

$$E_c(\kappa) = E(\kappa)\kappa^{5/3}\varepsilon^{-2/3} \tag{2.44}$$

The Kolmogorov hypothesis predicts the $-5/3$ spectrum in inertial subrange. This power-law behavior is clearly visualized in the compensated energy spectrum because it is flattened.

The dissipated energy spectrum is [140]:

$$E_d(\kappa) = E(\kappa)\kappa^2\nu \tag{2.45}$$

2.5 Visualization method of coherent structures

The visualization and quantification of drop/vortices interaction is a primary source of information for developing the sub-models. The original Q criterion method transforms a velocity field (vector) to a Q field (scalar), which is then used to identify and visualize coherent structures. In this method, Q is defined as the balance between rotation and strain rate [98].

$$Q = \frac{1}{2}\left(\Omega^2 - S^2\right) \tag{2.46}$$

where $\Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right)$ is the rotation rate, $S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$ is the strain rate. These are the antisymmetric and symmetric parts of the velocity gradient tensor correspondingly. Q is defined as the balance between rotation and strain rates [98]. According to Equation (2.46), those areas with positive Q values have higher rotation rate.

However, sometimes the absolute maximum Q value is several orders of magnitude larger than the absolute minimum value. The wide range of Q values decreases the accuracy of visualization. The normalized Q_n criterion is an advanced method which solves this problem since it removes the effect of order resulted from the wide range of value Q. In this method, Q_n is obtained by normalizing Q with the rotation squared [99].

$$Q_n = \frac{\Omega^2 - S^2}{\Omega^2} \tag{2.47}$$

For a given time instant, the process of extraction of coherent structures

included following steps [97]:

1. Calculate Q_n values for the entire domain.

2. Set up a threshold value Q_{th} for the Q_n field. The selection of the threshold value affects the visualization results. The extracted coherent structures are small if the Q_{th} is high (close to the maximum Q_n value). However, a low threshold may make the coherent structures too dense to visualize.

3. Find the maximum Q_n value and its location. The point with the maximum Q_n value is regarded as the 'starting point' in a structure only if this maximum Q_n value is larger than the threshold Q_{th}.

4. Identify all neighboring points from the 'starting point', points whose Q_n values higher than the Q_{th} are sorted out as in the same structure.

5. Record all points in this structure in another group, then reset all Q_n values of these points to zero.

6. Repeat steps 3 to 5 until no more structure is extracted. Record the maximum volume of the structure.

7. Set up a volume criterion V_{cr}. To make the visualization clear, structures smaller than the volume criterion are regards as noise and are removed. If the structure is larger than or equal to the V_{cr}, all Q_n values of the structure will be recorded. Usually, the volume criterion V_{cr} is set as less than the 4% of the maximum volume of coherent structures $V_{cr} \leq 4\%V_{max}$ [97].

In Fig. 3.13, the coherent structures of single phase homogeneous isotropic turbulent flow are presented, they are colored by the vorticity magnitude. The maximum Q_n value of the entire domain is 1 while the maximum unnormalized Q value calculated by Equation (2.46) is 1.745×10^{-3}. The minimum meaningful Q_n and Q values are 0, because only areas with positive values have higher rotation rate.

In this study, a practical identification method is proposed to fulfill the step 4 of the process. The method is explained based on a two-dimensional structure. The three-dimensional structure identification can be achieved by adding one more dimension in the identification process. Before introducing the method, three operations have to be defined. Assume the Q_n value of point $\mathcal{P}$ is higher than the threshold value, only the nearest four points surrounding $\mathcal{P}$ are checked. If a point in the domain has Q_n value is lower than the threshold value, the point $\mathcal{P}$ is then recorded as the 'boundary point'. Assume a 'moving block' that only departures from the 'starting point' and moves on points one by one along with x and y directions; Assume a 'flag' on each point, set all 'flags' to zero at the beginning.

Fig. 2.3 covers a random complex two-dimensional coherent structure. The 'starting point' is marked with a star and with coordinate $(0, 0)$. Assume that this is the first structure. The process of two-dimensional structure identification includes the following steps:

1. Beginning from the 'starting point', the 'moving block' moves to each point one by one along with x and y directions. If the Q_n value on current 'moving block' is higher than the threshold value, then judge the value of flag, change the flag of this point to $\mathcal{I}$ if the flag is zero,

where $\mathcal{I}$ is the $\mathcal{I}$th structure that is under identification; If the flag is not zero, then jump it and moves on the next point. Keep moving until four 'boundary points' are recorded.

2. When 'moving block' moves on a point that is higher than the threshold and has 'zero' flag, check all points along with the direction which is perpendicular to the direction that the 'moving block' moves along currently. If the flags of points are zero, then change them to $\mathcal{I}$; if they are not zero, then jump to the next point. This is called 'perpendicular check'. Finish the 'perpendicular check' when two 'boundary points' are recorded. Only new 'boundary points' are recorded as the new generation 'boundary points'. This step is regarded as the 'second layer check'.

   In Fig. 2.3, the 'moving point' is now at $(-3, 0)$. Check all points along y direction (marked by arrows) until two 'boundary points' $(-4, \pm 2)$ are recorded. Now all points between $(-4, 2)$ and $(-4, -2)$, $(0, 0)$ and $(-4, 0)$ have flag '1' because this is the first structure. By the end of step 1 and 2, all 'boundary points' that can be recorded are marked as circles in Fig. 2.3, these points are classified as the first generation 'boundary points'.

3. All this generation 'boundary points' are regarded as 'starting point', repeat these steps to get further generations of 'boundary points' until no more new 'boundary points' are found.

   In Fig. 2.3, the second generation 'boundary points' are marked as triangles. All 'boundary points' are found and all flags in the structure are changed to '1' till now. The structure is successfully identified.

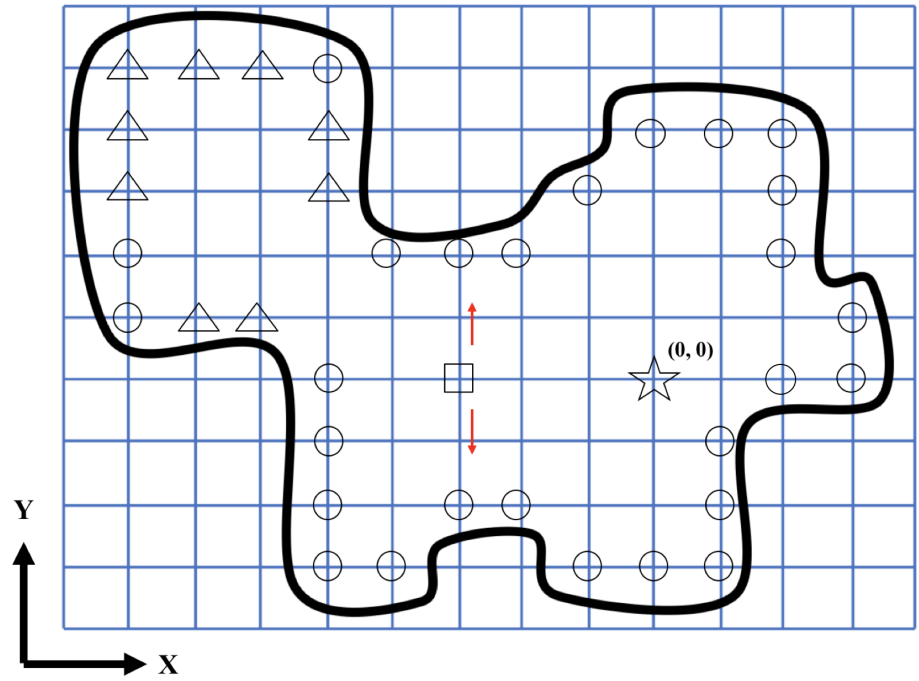


Figure 2.3: A random two-dimensional structure

To extend to the three-dimensional structures extraction, several modifications need to be implemented in the operation definition and identification process that were given above. The number of points which have to be checked to ensure a 'boundary point' becomes six; One more direction is added in the movement of 'moving block' and 'perpendicular check'; The number of 'boundary points' in steps 1 and 2 are six and four; A further layer check - 'third layer check', which is similar to the 'second layer check' has to be added following the 'second layer check'.

The normalized Q_n criterion extraction method coupled with the identification method described above can obtain coherent structures accurately despite the complexity of the vortices shape. With the unique 'flag' on each point, the raw data of exact positions and number of coherent structures are also obtained.

Chapter 3

Results and Discussion

In this chapter, results of multiple cases of single-phase and two-phase turbulent flow simulations are presented and discussed. The simulation cases and parameters are summarized in Section 3.1. To investigate statistical characteristics of liquid-liquid turbulent flow, the energy spectra, the probability distribution function (PDF) of vorticity, energy dissipation rate, and eigenvalues of strain tensor of the single-phase flow is discussed in Section 3.2. In Section 3.3, the effect of dispersed phase volume fraction on the energy spectra, PDF of vorticity, energy dissipation rate, and eigenvalues of strain tensor are presented. In Section 3.4, the effect of dispersed phase effect on the energy spectra, PDF of vorticity, energy dissipation rate, and eigenvalues of strain rate are shown. In Section 3.5, coherent structures of single-phase flow, and coherent structures together with the liquid-liquid interface of two-phase flow are given.

3.1 Simulation parameters

Three single-phase homogeneous isotropic turbulent flow fields were generated in a fully-periodic cubic domain of size $L \times L \times L = 300 \times 300 \times 300$ lattice units [lu]. The Kolmogorov length scale η_K is set equal to 1 [lu] for all simulations.

The flow field in conventional mixing devices is very inhomogeneous. This means depending on location in the vessel, the drop will be exposed to different flow conditions. To investigate the behaviour of drop under different turbulent conditions, three cases characterized by different intensity of turbulent flow field were considered. To replicate these flow conditions, three values of relaxation time related to fluid viscosity were selected: $\tau_c = 0.54,\ 0.525,\ 0.5$. These relaxation times result in different values of viscosity of continuous phase $\nu_c = c_s^2(\tau_c - 0.5)$, they were then used in the Equation 2.20. The energy dissipation rate was calculated according to the Kolmogorov correlation [140] $\varepsilon = \nu_c^3/\eta_K^4$, where $\eta_K = 1$ [lu]. The simulation parameters that characterize three turbulent flow fields are shown in Table 3.1.

The turbulent generation approach did not allow to set the volume-averaged root-mean-squared velocity u_{rms} a priori. For that reason, it was determined as a result of simulation. Based on obtained value, four parameters were calculated: Taylor micro-scale $\lambda = (15\nu_c u_{rms}^2/\varepsilon)^{1/2}$ [140]; Taylor micro-scale

Table 3.1: Simulation parameters. Lattice units: [lu] - lattice units for length scale; [ts] - time step.

Case #	τ_c	ν_c	ε	u_{rms}	t_K, [ts]	t_{eddy}, [ts]	λ, [lu]	Re_λ
Case 1	0.54	$13.3 \cdot 10^{-3}$	$2.4 \cdot 10^{-6}$	$5.1 \cdot 10^{-2}$	75	1102	14.85	56.91
Case 2	0.525	$8.3 \cdot 10^{-3}$	$5.8 \cdot 10^{-7}$	$3.2 \cdot 10^{-2}$	120	1770	14.87	57.07
Case 3	0.51	$3.3 \cdot 10^{-3}$	$3.7 \cdot 10^{-8}$	$1.3 \cdot 10^{-2}$	300	4424	14.91	57.41

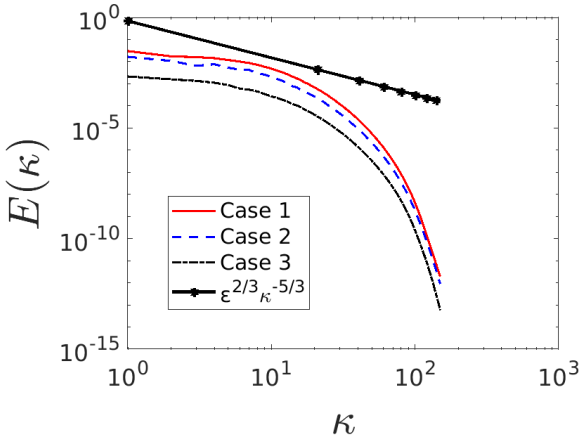
Reynolds number $Re_\lambda = u_{rms}/\nu_c$; Kolmogorov time scale $t_K = (\nu_c/\varepsilon)^{1/2}$; Eddy turnover time $t_{eddy} = u_{rms}^2/\varepsilon$. Therefore, for all considered cases, $\lambda \approx 15$ and $Re \approx 57$. Thus parameters are also presented in Table 3.1.

First, a fully-developed turbulent flow field was generated. Then a spherical single mother drop was injected into the flow field of condition Case 2. The size of drop varies to make the dispersed phase volume fraction to be $\phi = 0.02\%,\ 0.05\%,\ 3.35\%$. Further, 382 spherical mother drops were injected into the same flow field, the diameter of each drop was 30 [lu], that makes the dispersed phase volume fraction to be 20%. The effect of dispersed phase viscosity was also investigated by setting the dispersed to continuous phase viscosity ratio to $\mu_d/\mu_c = 0.1,\ 1,\ 10$.

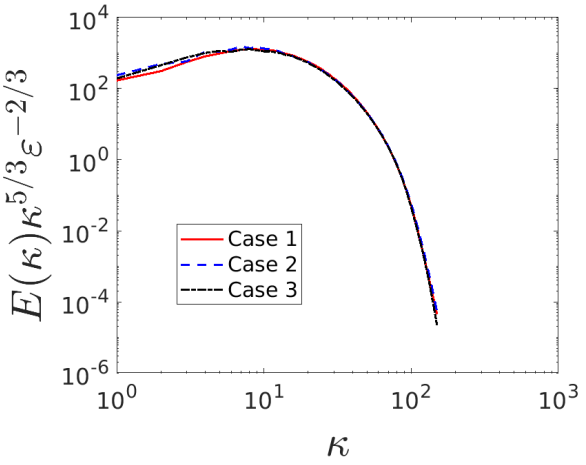
3.2 Single phase flow

3.2.1 Energy spectra

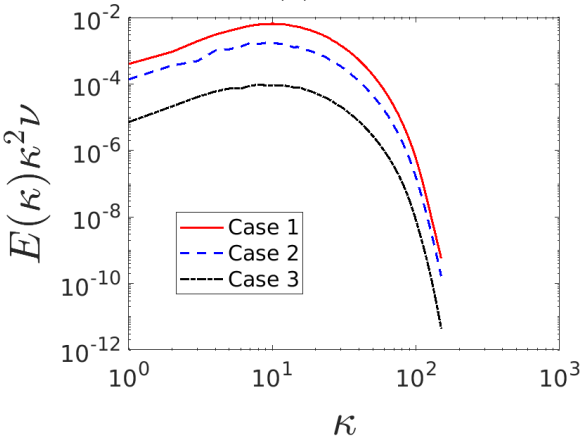
Fig. 3.1 presents the comparison of energy spectra, compensated energy spectra and dissipated energy spectra for Case 1, Case 2 and Case 3. As can be seen from Fig. 3.1(a), the simulation domain of edge length $L = 300$ [lu] resolves the inertial sub-range in three cases. However, the resolved inertial sub-range in three cases are narrow. The Fig. 3.1(b) highlights this conclusion: by multiplying the energy spectrum by $\kappa^{5/3}\varepsilon^{-2/3}$, the curve that denotes inertial sub-range becomes a horizontal line in compensated energy spectra plot. The horizontal region only exists in a range $8 < \kappa < 10$. To resolve a wider inertial sub-range, it is necessary to extend the size of the simulation domain. In turn, this will increase the computational cost of DNSs.



(a)



(b)



(c)

Figure 3.1: Energy spectra of single phase turbulent flows at different energy inputs. (a) Energy spectra; (b) Compensated energy spectra; (c) Dissipated energy spectra.

In Fig. 3.1(a) the energy at low wave number (large length scale) is different in three cases. The energy entered turbulence is maximum in Case 1 while is minimum in Case 3. The discrepancy of energy between Case 1 and Case 2 is much smaller than that between Case 2 and Case 3. The energy entered in Case 2 is nearly the same as the energy entered in Case 1. This proves the parameter set in Table 3.1: the energy dissipation rate of Case 1 is 4.1 times larger than that of Case 2 while the energy dissipation rate of Case 2 is 15.7 times larger than that of Case 3. In Fig. 3.1(b), the dissipation ε and wave number κ have no effect on the plot, because by scaling with $\kappa^{5/3}\varepsilon^{-2/3}$, the relationship is obtained: $E(\kappa)\kappa^{5/3}\varepsilon^{-2/3}\ \varepsilon^{2/3}\kappa^{-5/3} = \varepsilon^0\kappa^0$. Thus, the three curves nearly overlap in the compensated energy spectrum.

In Fig. 3.1(c), the energy of Case 1 dissipated fast at high wave number, meaning that the higher energy input results in higher energy dissipation. By scaled with $\kappa^2\nu$, the relationship is obtained: $E(\kappa)\kappa^2\nu \sim \varepsilon\kappa^{1/3} \sim \nu^3\kappa^{1/3}$, the Kolmogorov $-5/3$ law becomes $1/3$, the curve that represents the inertial subrange in the log-log is parallel to the straight line of $1/3$ slope. The viscosity in Case 1 is 1.6 times larger than that in Case 2, the viscosity in Case 2 is 2.5 times larger than that in Case 3. Due to the relationship of ν^3 in dissipated energy spectra, the discrepancy between Case 2 and Case 3 is far larger than that between Case 1 and Case 2. As can be seen, at large wave number, the higher energy input flow condition has higher energy dissipation. This supports the conclusion from Fig. 3.1.

It is also noticed that there is minor difference in these plots for a given time instant and the time-averaged data. Thus the energy spectra presented here are based on one time instant.

The turbulent kinetic energy calculated by Equation (2.37) and (2.38), as

Table 3.2: Turbulent kinetic energy and average energy dissipation rate calculated by different methods.

Case #	E_v	E_e	ε_v	ε_e	ε
Case 1	$3.5 \cdot 10^{-3}$	$3.1 \cdot 10^{-3}$	$1.6 \cdot 10^{-6}$	$2.3 \cdot 10^{-6}$	$2.4 \cdot 10^{-6}$
Case 2	$1.7 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	$4.0 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$
Case 3	$2.5 \cdot 10^{-4}$	$2.2 \cdot 10^{-4}$	$2.4 \cdot 10^{-8}$	$3.5 \cdot 10^{-8}$	$3.7 \cdot 10^{-8}$

well as the average energy dissipation rate calculated by Equation (2.39) and (2.43) in three cases are given in Table. 3.2. E_v and E_e are the velocity-based and energy spectrum-based turbulent kinetic energy; ε_v and ε_e are the velocity-based and energy spectrum-based average energy dissipation rate.

As can be seen, the turbulent kinetic energy obtained from the velocity field and energy spectrum are not exactly the same, the one from velocity field is larger. For the average energy dissipation rate, the one obtained from energy spectrum is larger than that from velocity field. Compared to the energy dissipation rate ε, the results from energy spectrum are closer to these values.

3.2.2 Vorticity

Vortex is to be observed as an rotational region around a core. The concept of vorticity proposed to describe a vortex was defined as the curl of velocity ($\omega = \nabla \times v$). The vorticity was calculated using the MATLAB intrinsic function 'curl'. Fig. 3.2 represents the probability density function (PDF) of vorticity in single phase turbulent flows of three energy inputs. The calculation of vorticity was implemented for each lattice cube, thus, a three-dimensional vorticity magnitude field of 300^3 values was obtained. For a continuous variable X, the probability density function $f(x)$ satisfies the relationship $\int f(x)dx = 1$.

Therefore, the probability density function for discrete random variable is $p(x_i) = \frac{n_i}{N\delta x}$, where n_i is the number of variable at x_i, N is the total number of variable, δx is the step between x_i and x_{i+1}. The probability density function reveals the distribution of the variable.

As can be seen from the Fig. 3.2, the maximum density of vorticity in Case 1, Case 2, and Case 3 are 74, 120, and 305, respectively. Therefore, with the increase of energy input, the maximum density of vorticity decreases. The maximum density of vorticity in Case 3 is 2.5 times of that in Case 2, and the maximum density of vorticity in Case 2 is 1.6 times of that in Case 1.

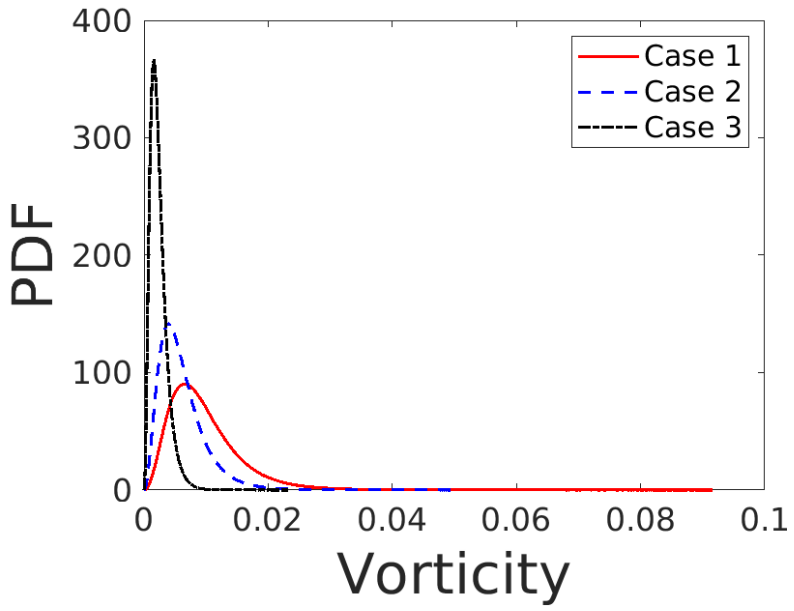


Figure 3.2: PDF of vorticity in single phase turbulent flows of different energy inputs

The increase of energy input also results in the increase of distribution of vorticity, i.e. the distribution of vorticity is wider. It validates the energy spectra in Fig. 3.1: for a given wave number (vortex of a given length scale), the vortex in larger energy input flow condition contains more energy because

the vorticity is higher.

3.2.3 Local energy dissipation rate

The energy dissipation rate reveals the rate at which the turbulent kinetic energy is transferred into thermal energy with the dissipation of small eddies [140]. The calculation is given from Equation (2.40) to (2.42). Fig. 3.3 represents the normalized PDF of energy dissipation rate in single phase turbulent flows of different energy inputs at one time instant. The energy dissipation rate of each lattice cube is calculated and normalized by the average dissipation rate of the corresponding case listed in Table. 3.1. For example, the curve that denotes flow condition of Case 1 is obtained by scaling with the average dissipation rate of Case 1 in Table. 3.1.

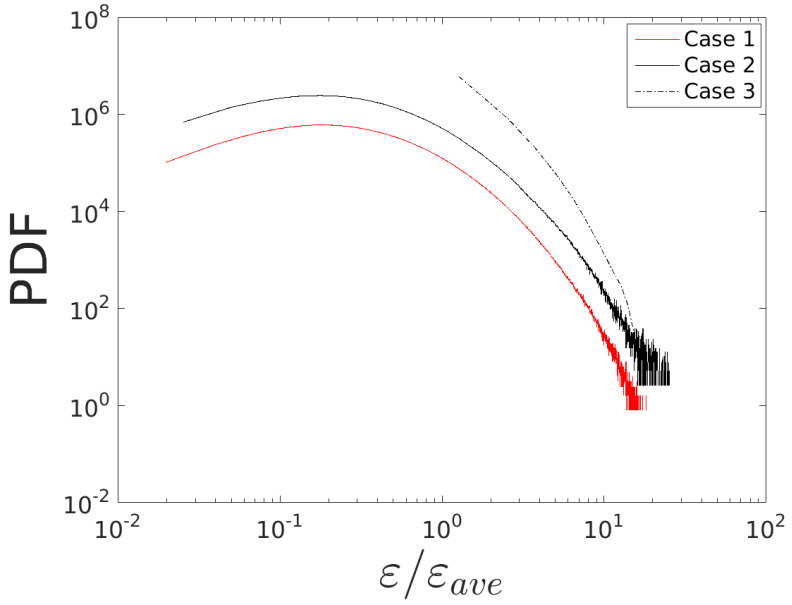


Figure 3.3: PDF of normalized energy dissipation rate in single phase turbulent flows of different energy inputs

As can be seen in the Fig. 3.3, in the Case 1 and Case 2, most of the

normalized energy dissipation rates in the region are smaller than the average energy dissipate rate ε_{ave}. The increase of energy input results in the decrease of PDF of normalized energy dissipation rate. Further, both the maximum and minimum values of normalized energy dissipation rate are smaller in the higher energy input case. The PDF of $\varepsilon/\varepsilon_{ave}$ decreases at higher normalized dissipation rate, finally, the PDF oscillates at a specific state. For example, in Case 1, the PDF reaches the maximum value at $\varepsilon/\varepsilon_{ave} = 0.18$, then it decreases from 5.98×10^5 to 1, finally, the PDF oscillates around 1. The distribution of $\varepsilon/\varepsilon_{ave}$ is more discrete at higher value region.

3.2.4 Eigenvalues of strain tensor

The velocity gradient tensor given in the Equation (2.42) can be decomposed into a symmetric strain tensor and an antisymmetric rotation tensor. The vorticity in Section 3.2.2 describes the rotation rate of particles related to the rotation tensor. In this section, the eigenvalues of strain tensor are studied to reveal the compression and stretching. The strain tensor s_{ij} is given by:

$$s_{ij} = \frac{1}{2}\begin{bmatrix} \frac{\partial u_1}{\partial x_1} + \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} & \frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} \\ \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} + \frac{\partial u_2}{\partial x_2} & \frac{\partial u_2}{\partial x_3} + \frac{\partial u_3}{\partial x_2} \\ \frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1} & \frac{\partial u_2}{\partial x_3} + \frac{\partial u_3}{\partial x_2} & \frac{\partial u_3}{\partial x_3} + \frac{\partial u_3}{\partial x_3} \end{bmatrix}$$

The eigenvalues of each strain tensor are calculated. The negative eigenvalue λ_1 is most compressive while the positive eigenvalue λ_3 is the most stretching, the eigenvalue λ_2 between λ_1 and λ_3 is either compressive or stretching [145].

The eigenvalues of strain tensor are calculated using MATLAB 2017 in-

trinsic function 'eig' for each lattice cube. Fig. 3.4 represents the PDF of eigenvalues of strain tensor of single phase turbulent flows of different energy inputs, the red curves denote flow condition Case 1, blue curves denote flow condition Case 2, and black curves denote flow condition Case 3; the dashed curves represent eigenvalue λ_1 (negative eigenvalue), solid curves represent eigenvalues λ_2, and dash-dot curves represent eigenvalues λ_3 (positive eigenvalue), $\lambda_1 < \lambda_2 < \lambda_3$.

As can be seen, in higher energy input flow condition, the distribution of eigenvalues is wider. The increase of energy input results in the density decrease of λ_1, increase of λ_2 and λ_3. Therefore, in higher energy input flow, the flow field is more stretching than compressive.

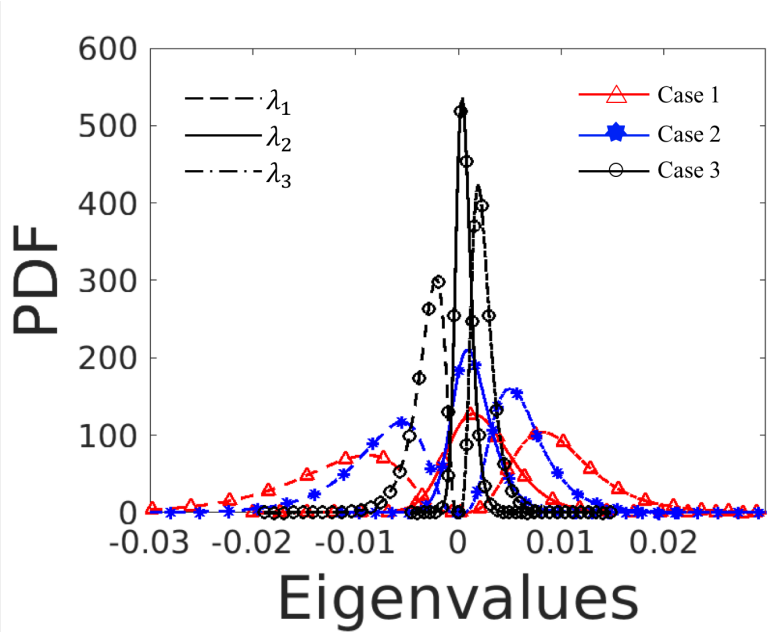


Figure 3.4: PDF of eigenvalues of strain tensor of single phase turbulent flows of different energy inputs

3.3 Effect of dispersed phase volume fraction

3.3.1 Energy spectra

In Fig. 3.5, the energy spectra for a single phase flow as well as for the cases with different values of dispersed phase volume fraction ϕ in the flow condition Case 2 are shown. The spherical drop of different sizes are injected into the flow Case 2 to reach the different dispersed phase volume fraction $\phi = 0.02\%$, 0.05%, 3.35% and 20%. For the larger dispersed phase volume fraction, the energy at high wave number (small length scale) is larger. In Fig. 3.5(a), the energy of $\phi = 20\%$ at high wave number is around 10^5 larger than the energy of single-phase flow. Therefore, the presence of deformable liquid-liquid interface promotes the energy dissipation. This is more clear in Fig. 3.5(b): the increase of dispersed phase volume fraction results in larger energy dissipation rate. The similar conclusion was found in [146], which was a study of solid-liquid dispersion: the larger dispersed phase volume fraction led to a higher turbulent energy at high wave number, it is accompanied by an increase of energy dissipation rate.

However, the slight decrease of energy in the inertial sub-range resulted by the increase of dispersed phase volume fraction is noticed. The energy of the condition $\phi = 20\%$ around low wave number $\kappa = 10$ is the minimum among different dispersed phase volume fraction cases. Therefore, in higher dispersed volume fraction case, the energy covered by different size eddies is distributed more uniformly. This conclusion is also supported by Fig. 3.5(c).

Another important finding of this work is that the implementation of MRT mitigated the artificial significant $(10^{10} - 10^{12})$ energy gain at high wave numbers reported earlier [113]. This energy gain is obvious when the velocity at

the diffuse interface is of order equals to the spurious velocity that represents a numerical peculiarity of diffuse interface methods including LBM. Therefore, by implementing the MRT collision operator, the magnitude of physical velocity becomes larger than the spurious velocity.

3.3.2 Vorticity

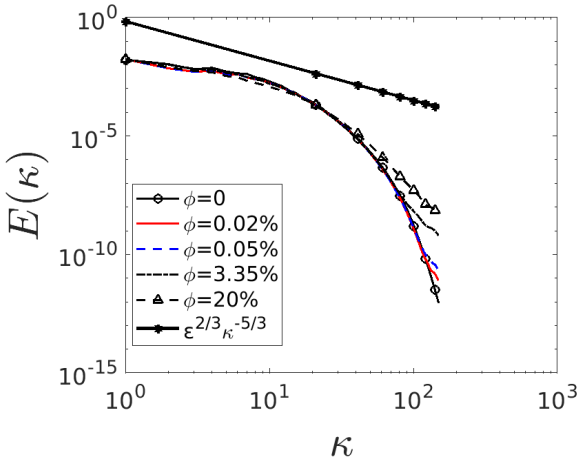
Fig. 3.6 represents the PDF of vorticity in two phase turbulent flows of different dispersed phase volume fraction. The increase of dispersed phase volume fraction results in the increase of density of distribution at lower vorticity (around 0) and decrease of density of distribution at higher vorticity (larger than 3×10^{-3}, smaller than -3×10^{-3}). Therefore, in high dispersed phase volume fraction case, the average vorticity magnitude of flow field is smaller.

3.3.3 Local energy dissipation rate

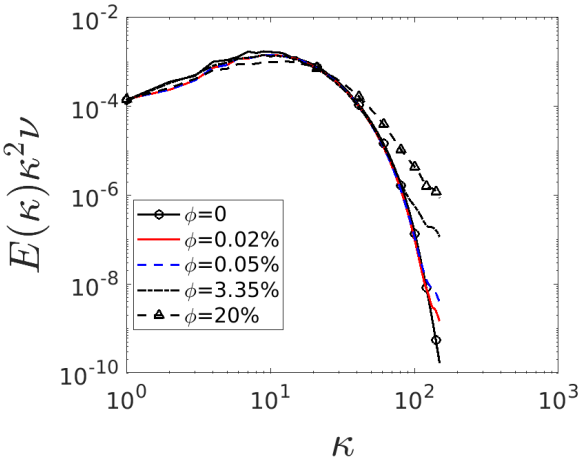
Fig. 3.7 gives the PDF of normalized energy dissipation rate in two phase turbulent flows of different dispersed phase volume, the energy dissipation rate for each lattice cube in domain is calculated and normalized by the average dissipation rate of Case 2 ($\varepsilon_{ave} = 5.8 \times 10^{-7}$).

In the Fig. 3.7, the maximum PDF of $\varepsilon/\varepsilon_{ave}$ is 2.4×10^6 around $\varepsilon/\varepsilon_{ave} = 0.18$ in drop-free turbulence, the maximum PDF of $\varepsilon/\varepsilon_{ave}$ is 3.5×10^6 around $\varepsilon/\varepsilon_{ave} = 0.11$ in turbulence with $\phi = 20\%$. Therefore, the increase of liquid-liquid interface results in the increase of the maximum PDF of $\varepsilon/\varepsilon_{ave}$, and the decrease of value of $\varepsilon/\varepsilon_{ave}$ that has the maximum PDF.

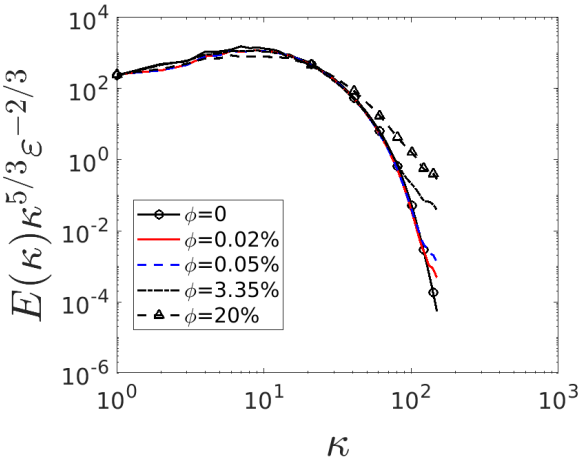
However, the increase of dispersed phase volume fraction results in the decrease of PDF at higher value of the normalized energy dissipation rate



(a)



(b)



(c)

Figure 3.5: Energy spectra of two-phase turbulent flows of different dispersed phase volume fraction in Case 2. (a) Energy spectra; (b) Dissipated energy spectra; (c) Compensated energy spectra.

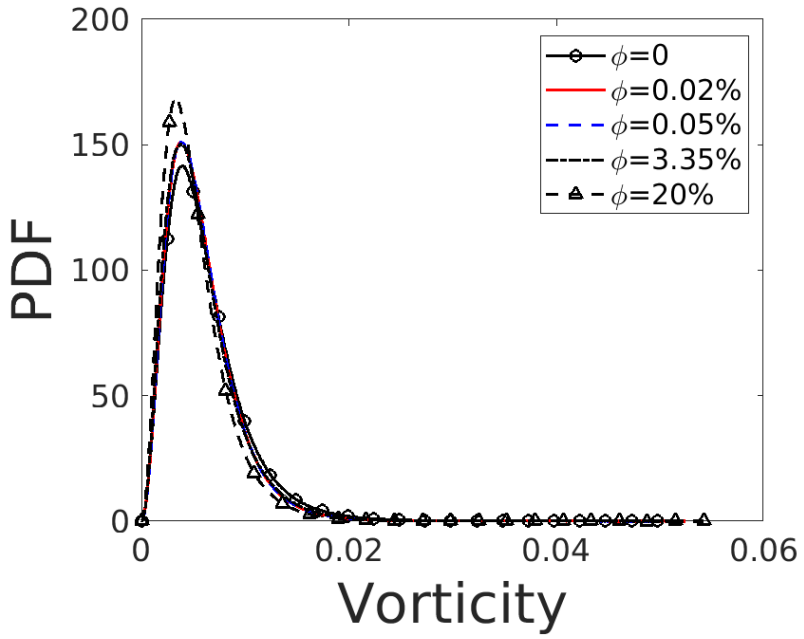


Figure 3.6: PDF of vorticity in two phase turbulent flows of different dispersed phase volume fraction in Case 2

($\varepsilon/\varepsilon_{ave} > 0.38$). For example, at $\varepsilon/\varepsilon_{ave} = 2$, the PDF of $\varepsilon/\varepsilon_{ave}$ equals to 1.05×10^5 in $\phi = 0$ while equals to 5.73×10^4 in $\phi = 0.20\%$.

3.3.4 Eigenvalues of strain tensor

Fig. 3.8 represents the PDF of eigenvalues of strain tensor of single phase and two phase turbulent flows of different dispersed phase volume fraction, the black curves denote the case $\phi = 0$ (single phase), red curves denote the case $\phi = 0.02\%$, blue curves denote the case $\phi = 0.05\%$, green curves denote the case $\phi = 3.35\%$, magenta curves denote the case $\phi = 20\%$; the dashed curves represent eigenvalue λ_1, solid curves represent eigenvalues λ_2, and dash-dot curves represent eigenvalues λ_3, $\lambda_1 < \lambda_2 < \lambda_3$.

As can be seen, the increase of dispersed phase volume fraction results

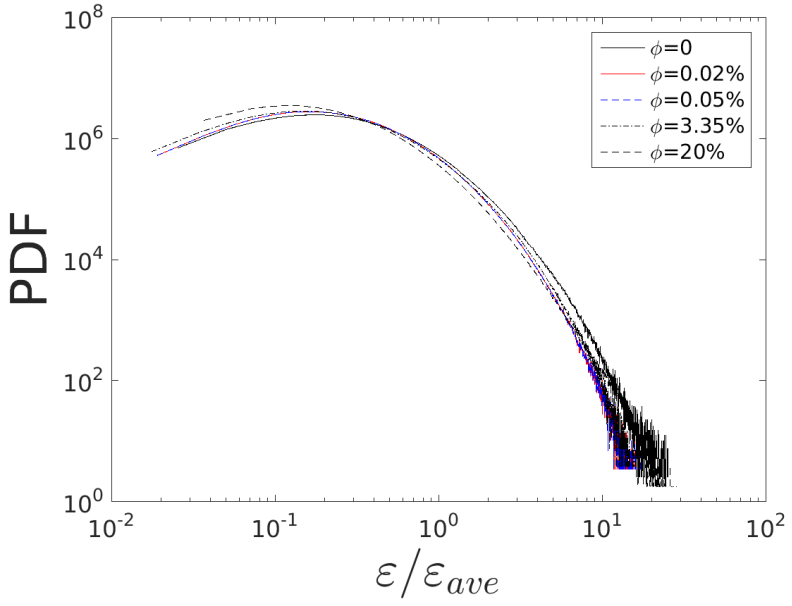


Figure 3.7: PDF of normalized energy dissipation rate in two phase turbulent flows of different dispersed phase volume fraction in Case 2

in the increase of the maximum probability density, and the increase of λ_1, decrease of λ_2 as well as λ_3. This reveals that the increase of liquid-liquid interface boosts the compression.

3.4 Effect of dispersed phase viscosity

3.4.1 Energy spectra

The energy spectra in the flow condition Case 2 with different viscosity ratios $\mu_d/\mu_c = 0.1$, 1, 10 at dispersed phase volume fraction $\phi = 0.05\%$ are presented in Fig. 3.9. As is shown in Fig. 3.9(a) and (c), the change of viscosity ratio has no effect on the inertial sub-range. However, at high wave number, the small scale eddies in large viscosity ratio case contain less energy, the increase of dispersed phase viscosity suppresses dissipation rate.

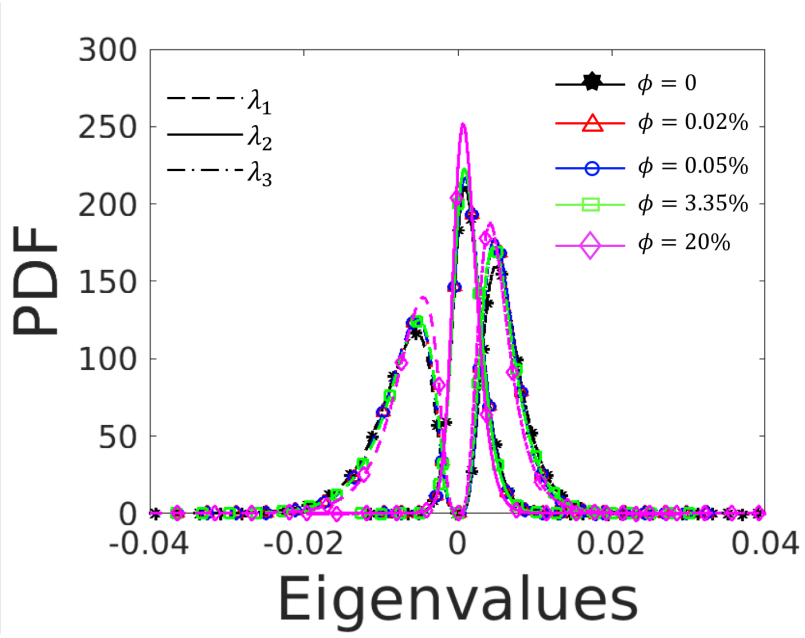


Figure 3.8: PDF of eigenvalues of strain tensor of single phase and two phase turbulent flows of different dispersed phase volume fraction in Case 2

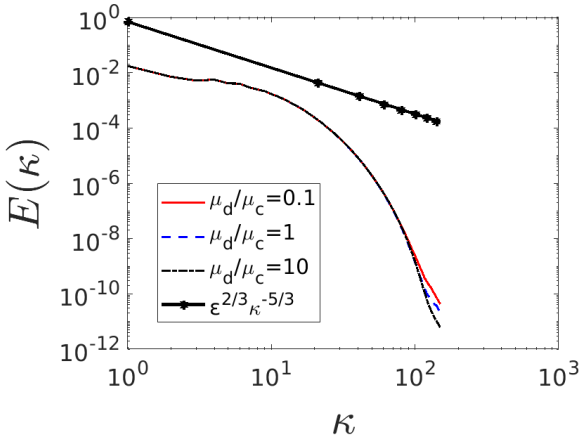
3.4.2 Vorticity

Fig. 3.10 represents the PDF of vorticity in two phase turbulent flows of different viscosity ratio, the viscosity ratio μ_d/μ_c is 0.1, 1, and 10. As can be seen, the change of viscosity nearly has no effect on the PDF of vorticity: the maximum is 125.5 in the case $\mu_d/\mu_c = 1$, 122.3 in the cases $\mu_d/\mu_c = 0.1$ and $\mu_d/\mu_c = 10$.

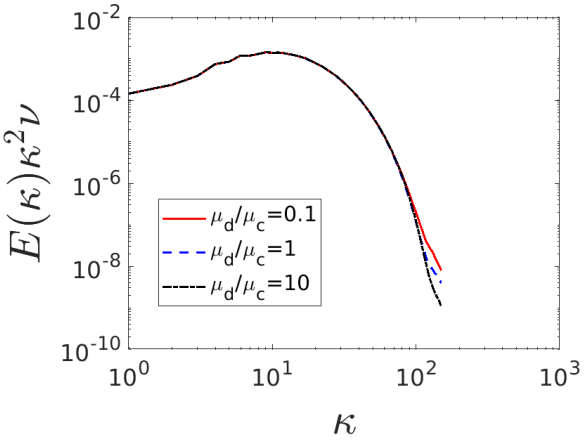
3.4.3 Local energy dissipation rate

Fig. 3.11 gives the PDF of normalized energy dissipation rate in two phase turbulent flows of different viscosity ratio, the energy dissipation rate or each lattice cube in domain is calculated and normalized by the average dissipation rate of Case 2 ($\varepsilon_{ave} = 5.8 \times 10^{-7}$).

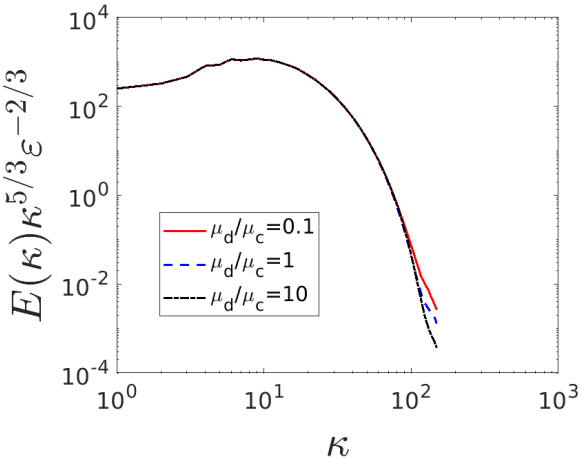
As can be seen, the maximum PDF is 2.63×10^6, it occurs around $\varepsilon/\varepsilon_{ave} =$



(a)



(b)



(c)

Figure 3.9: Energy spectra of two-phase turbulent flows of different viscosity ratio at $\phi = 0.05\%$ in Case 2. (a) Energy spectra; (b) Dissipated energy spectra; (c) Compensated energy spectra.

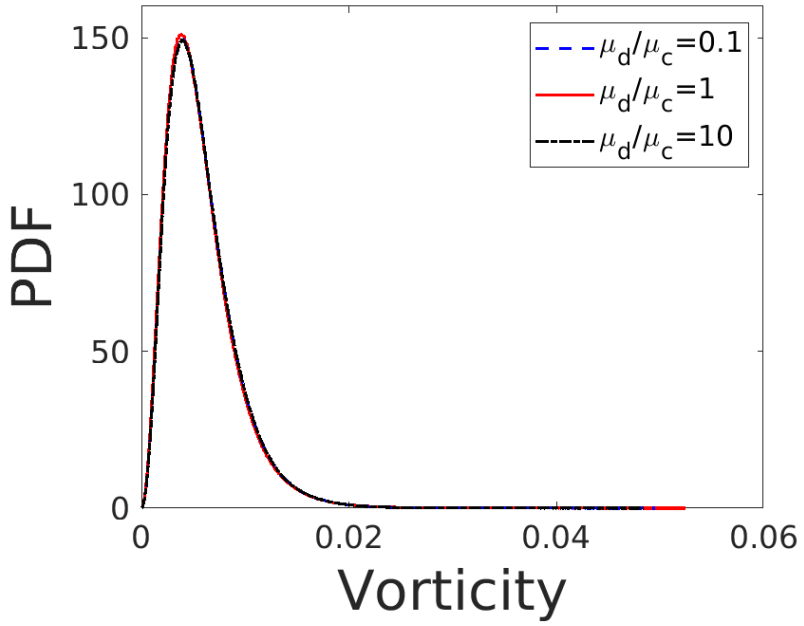


Figure 3.10: PDF of vorticity in two phase turbulent flows of different viscosity ratio at $\phi = 0.05\%$ in Case 2

0.18 in three viscosity ratios cases. It is noticed that the variance of viscosity ratio μ_d/μ_c has no obviously influence on the PDF of normalized energy dissipation rate.

3.4.4 Eigenvalues of strain tensor

Fig. 3.12 represents the PDF of eigenvalues of strain tensor of two phase turbulent flows of different viscosity ratio μ_d/μ_c, the red curves denote the case $\mu_d/\mu_c = 0.1$, the blue curves denote the case $\mu_d/\mu_c = 1$, the black curves denote the case $\mu_d/\mu_c = 10$; the dashed curves represent eigenvalue λ_1, solid curves represent eigenvalues λ_2, and dash-dot curves represent eigenvalues λ_3, $\lambda_1 < \lambda_2 < \lambda_3$. As can be seen, the variance of viscosity ratio nearly results in no variance of PDF of eigenvalues. Thus, both the compression and streching

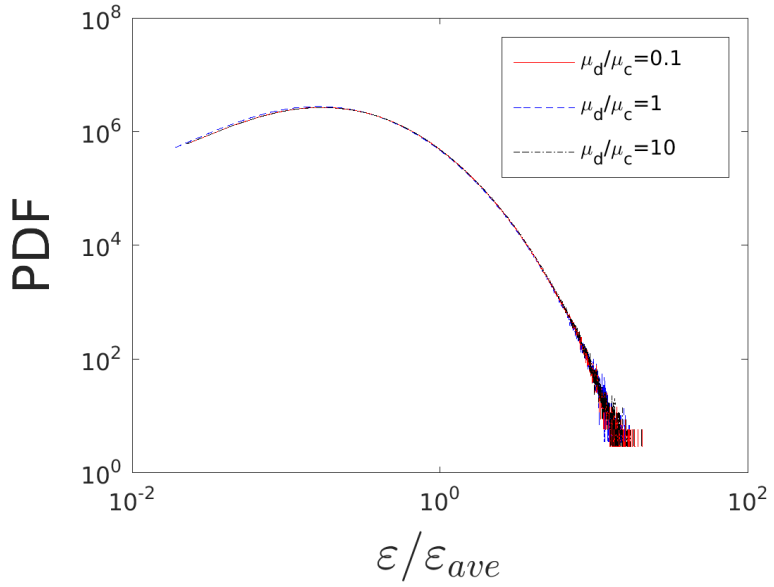


Figure 3.11: PDF of normalized energy dissipation rate in two phase turbulent flows of different viscosity ratio at $\phi = 0.05\%$ in Case 2

are impervious from the variance of viscosity.

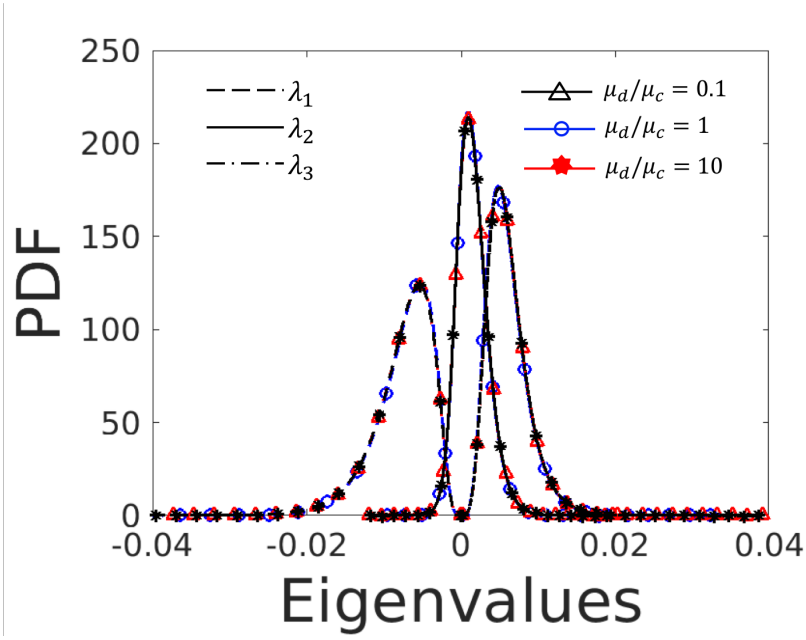


Figure 3.12: PDF of eigenvalues in two phase turbulent flows of different viscosity ratio at $\phi = 0.05\%$ in Case 2

3.5 Coherent structures

3.5.1 Single-phase flow

In normalized Q_n criterion, the selection of threshold value Q_{th} and cutoff volume V_{cr} is important because it directly affects the extraction of coherent structures. Therefore, the effects of Q_{th} and V_{cr} are investigated on the flow condition of Case 2.

Fig. 3.13 presents coherent structures extracted under different Q_{th} in Case 2, the structures are colored by vorticity magnitude ($\| \nabla \times u \|$). This figure reveals the effect of Q_{th}. In Table 3.3, the parameters and results of coherent structures under effect of Q_{th} were listed.

V_{cr} is the cutoff volume, V_{max} is the volume of the maximum structure. N is the number of coherent structures in the domain. As can be seen, the increase

Table 3.3: Parameters and results of coherent structures under effect of Q_{th}. N is the number of structures; V_{max} is the volume of the maximum structure.

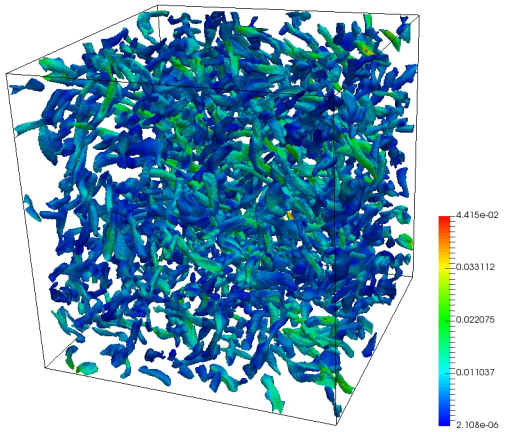
Figure #	Q_{th}	V_{cr}/V_{max}	N	V_{max}
Figure 3.13(a)	0.8		688	10162
Figure 3.13(b)	0.85	4%	778	5784
Figure 3.13(c)	0.9		879	2470

of Q_{th} results in the increase of the number of structures and decrease of the size of structures. The color remains the same in three sub-figures because the change of Q_{th} has no influence on the vorticity field.

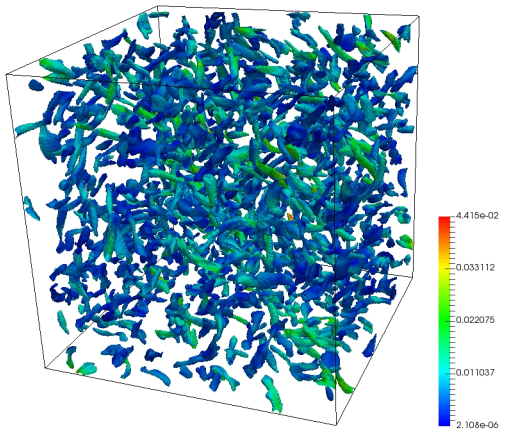
Therefore, Q_{th} should be carefully selected. If Q_{th} is large (close to 1), the size of structures will be small. It results in the lack of information in visualization, for example, part of important vortices maybe neglected; if Q_{th} is too small, the structures will be too dense to visualize. When $Q_{th} = 0.8$, the total volume of extracted coherent structures is 4.1% of the volume of the domain, it matches the published result [97]. Thus, in this study, $Q_{th} = 0.8$ is regarded as a suitable threshold value for visualization.

Fig. 3.14 gives coherent structures extracted under the same Q_{th}, but the noises are removed under different V_{cr}. The structures are colored by vorticity magnitude ($\| \nabla \times u \|$). In Table 3.4, the parameters and results of coherent structures under effect of V_{cr} were listed.

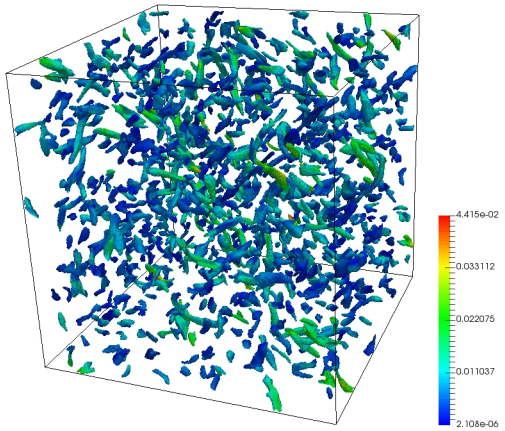
As can be seen, the increases of V_{cr} results in the decrease of the number of structures. Only 20% of number of structures remain when the V_{cr} is increased from 0 to 2% of the largest structure, but 62% of number of structures are left when the V_{cr} is increased from 2% to 4% of the largest structure. Therefore, the deduction of noise is more obvious at the beginning of increase of V_{cr}. With the increase of V_{cr}, the deduction of noise becomes weak. It is because the



(a)



(b)



(c)

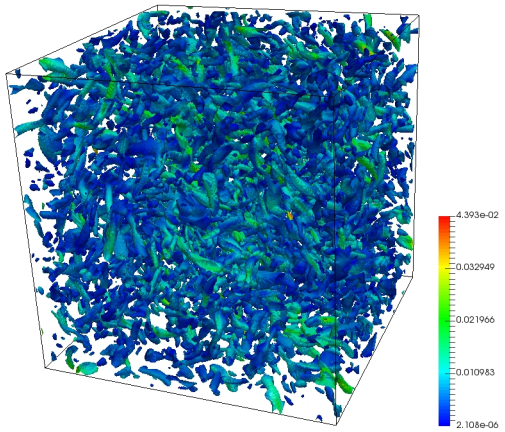
Figure 3.13: Coherent structures of different threshold Q_{th} extracted in Case 2. (a) $Q_{th} = 0.8, V_{cr}/V_{max} = 4\%$; (b) $Q_{th} = 0.85, V_{cr}/V_{max} = 4\%$; (c) $Q_{th} = 0.9, V_{cr}/V_{max} = 4\%$.

Table 3.4: Parameters and results of coherent structures under effect of V_{cr}. N is the number of structures; V_{max} is the volume of the maximum structure.

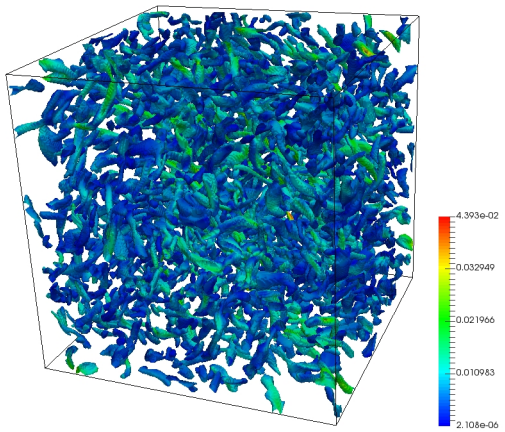
Figure #	Q_{th}	V_{cr}/V_{max}	N	V_{max}
Figure 3.14(a)		0	5521	
Figure 3.14(b)	0.8	2%	1114	10162
Figure 3.14(c)		4%	688	

noises are successfully removed. The cutoff volume $V_{cr} = 4\%V_{max}$ is regarded as a suitable value.

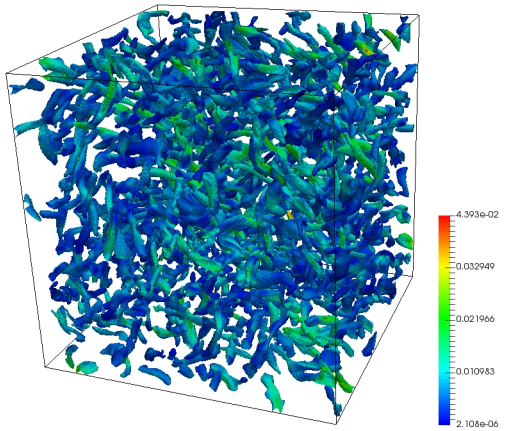
In Fig. 3.15, coherent structures of different cases and the corresponding probability distribution function (PDF) of volume-equivalent diameters are presented. In Table 3.5, the parameters and results of coherent structures of different flow conditions were listed. V_{cst} is the total volume of all coherent structures, V_t is the volume of domain. The volume-equivalent diameter of each structure is calculated by $d_e = \left(\frac{6V_{cs}}{\pi}\right)^{1/3}$, where V_{cs} is the volume of the structure. In Fig. 3.15(a), (c), (e), the coherent structures are colored by the vorticity magnitude ($\| \nabla \times u \|$). As can be seen, the decrease of energy input results in the decrease of vorticity magnitude. It can be also concluded that in Case 1 and Case 2, the number of structures and the volume of the largest structure are respectively the same. It is attributed to the little difference in energy input in Case 1 and Case 2 represented in Fig. 3.1(a). In Fig. 3.15(b), (d), (f), all bins are of the same width. The volume-equivalent diameter of structures in Case 3 is larger than that in Case 1 and 2. Therefore, the lower energy input results in the decrease of the number structures and increase of the size of structures. The assumption is that in a lower energy input flow condition, the flow is less turbulent, from Fig. 3.1(c) the dissipation rate is less, the vortices remain large size, thus the number of structures is small. To



(a)



(b)



(c)

Figure 3.14: Coherent structures of different volume criterion V_{cr}. (a) $Q_{th} = 0.8, V_{cr}/V_{max} = 0\%$; (b) $Q_{th} = 0.8, V_{cr}/V_{max} = 2\%$; (c) $Q_{th} = 0.8, V_{cr}/V_{max} = 4\%$.

Table 3.5: Parameters and results of coherent structures of different flow conditions. N is the number of structures; V_{max} is the volume of the maximum structure; V_{cst} is the total volume of all structures; V_t is the volume of the domain.

Figure #	Q_{th}	V_{cr}/V_{max}	N	V_{max}	V_{cst}/V_t
Figure 3.15(a)	0.8	4%	686	13882	4.1%
Figure 3.15(c)			688	10162	4.3%
Figure 3.15(e)			363	25937	3.4%

the contrary, a flow field with larger energy input is more turbulent, from Fig. 3.1(c) we know the dissipation rate is larger, the vortices in flow field tend to break up into smaller vortices, it leads to the large number and small size of structures.

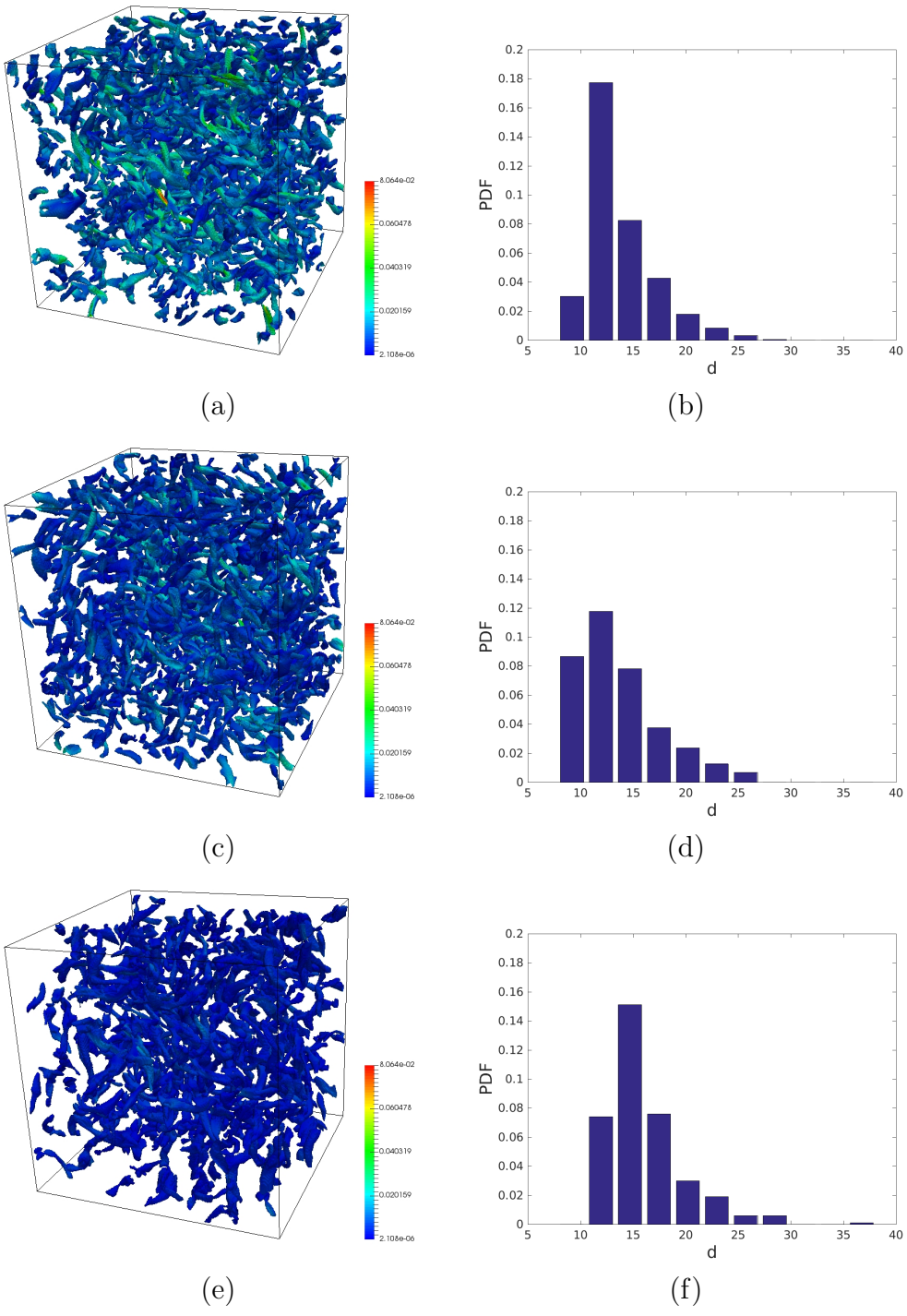


Figure 3.15: Coherent structures and corresponding PDF of volume-equivalent diameter. $Q_{th} = 0.8, V_{cr}/V_{max} = 4\%$. (a) Coherent structures of Case 1; (b) PDF of volume-equivalent diameter of Case 1; (c) Coherent structures of Case 2; (d) PDF of volume-equivalent diameter of Case 2; (e) Coherent structures of Case 3; (f) PDF of volume-equivalent diameter of Case 3.

3.5.2 Two-phase flow

Visualization of turbulent flow field together with the liquid-liquid interface provides valuable information on how drop/vortices interaction occurs. The most commonly used assumption in breakup models is that only vortices of the size equal to or less than the drop size are responsible for drop breakup.

In Fig. 3.16, drop/coherent structures and the corresponding PDF of the volume-equivalent diameter of structures are presented. The results are presented for two time instances: the first one $t = 10t_K$ stands for the state just before the breakup, and the second state $t = 15t_k$ is after drop disintegration, where t_K is the Kolmogorov time scale. The liquid-liquid interface is colored by a solid color (gray), and coherent structures are colored by vorticity magnitude. The threshold value $Q_{th} = 0.8$ together with the cutoff volume $V_{cr} = 4\%V_{max}$ makes the total volume of extracted coherent structures to be 4.3% of the simulation domain. This volume of structures ensures the clearance of visualization.

The number of coherent structures in Fig. 3.16(a) and (c) are 714 and 700, respectively. Thus, there is no obvious difference in numbers of structures. The PDF of volume-equivalent diameter of structures is calculated by $d_e = \left(\frac{6V_{cs}}{\pi}\right)^{1/3}$, where V_{cs} is the volume of the structure. As can be seen from Fig. 3.16(b) and (d), the size of structures slightly decreases after the breakup of a drop. The turbulent kinetic energy of extracted coherent structures is 4.07% and 4.05% of the total kinetic energy in Fig. 3.16(a) and (c). This phenomenon is explained by the Kolmogorov hypothesis that energy transforms to a smaller scale with the breakup of vortices. The most volume-equivalent diameter of coherent structures is equal to or smaller than 15 [lu], which is also close to

the Taylor micro-scale given in Table 3.1. The injected mother drop had a diameter of 30 [lu], which was larger than the vortices. After the breakup, the daughter drops that did not drop again were smaller than the vortices. Therefore, this result validates the general assumption that only vortices of size equal to or smaller than drop size are responsible for the breakup event.

Our results also indicate that not a single but multiple vortices interact with the drop at the same time. This interaction was revealed when a mother drop was injected into turbulent flows: a breakup of the drop took place.

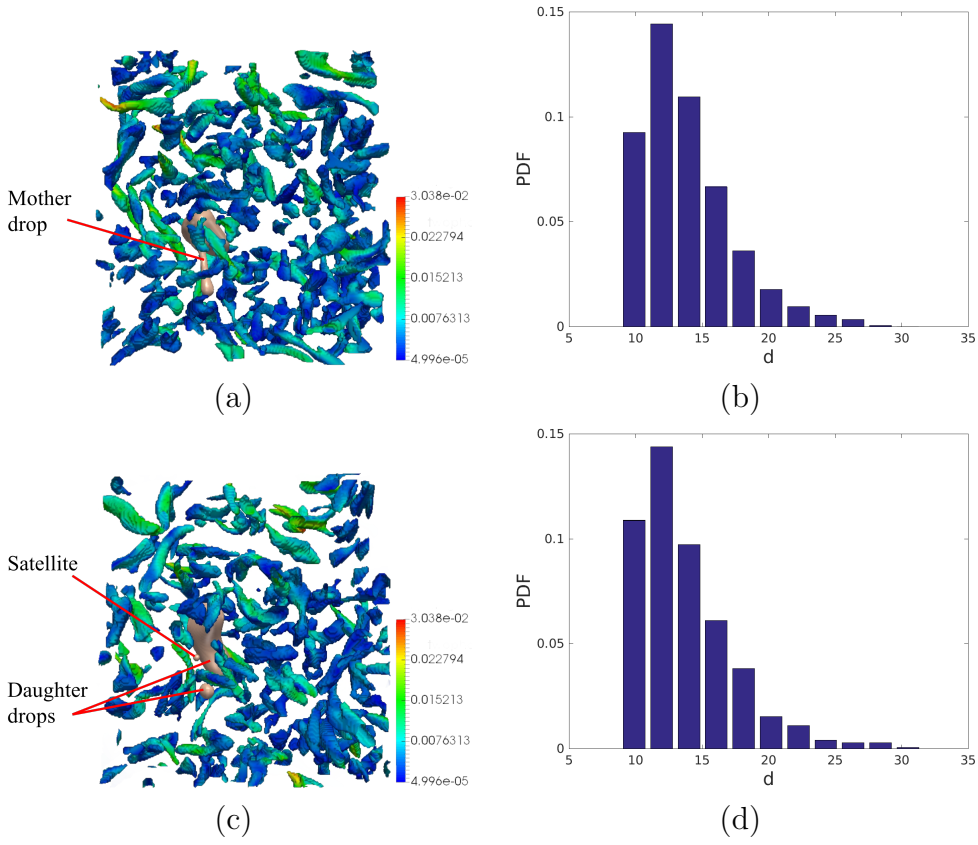


Figure 3.16: Drop/coherent structures and corresponding PDF of volume-equivalent diameter in Case 2 at different time instants. $Q_{th} = 0.8, V_{cr}/V_{max} = 4\%$. (a) Drop/coherent structures at $t/t_K = 10$; (b) PDF of volume-equivalent diameter at $t/t_K = 10$; (c) Drop/coherent structures at $t/t_K = 15$; (d) PDF of volume-equivalent diameter at $t/t_K = 15$.

Chapter 4

Conclusions

Liquid-liquid dispersions generated by the agitation of two immiscible liquids are of great importance in chemical and petroleum industries, pharmacy, biology, food, and cosmetics. In these multi-phase systems, the drop size distribution (DSD) of the dispersed phase defines dispersion properties. For that reason, the ability to predict and control the DSD is crucial. One of the modeling approaches used to estimate the DSD is based on the solution of population balance equations (PBEs). The persistent limitation of this approach is that the results heavily rely on the choice of breakup and coalescence models (kernels). In this study, direct numerical simulations (DNSs) are used to study a single drop behavior in homogeneous isotropic turbulence to understand the mechanisms of drop breakup. The results are necessary to develop well-grounded sub-models that can be incorporated into PBE modeling and improve its reliability.

In this study, three single phase flow conditions of different energy input were investigated in a three-dimensional cubic domain of size 300^3 [lu]. Then a spherical drop of different sizes was injected into the flow of con-

dition Case 2 to reach the different dispersed phase volume fraction ϕ = 0, 0.02%, 0.05%, 3.35%, 20%. This is fulfilled to study the effect of dispersed phase volume fraction. the density and viscosity of drop are set to be the same with density and viscosity of continuous phase. Further, the dispersed phase to continuous phase viscosity ratio μ_d/μ_c is set as 0.1, 1, and 10 to explore the effect of viscosity ratio under condition $\phi = 0.05\%$ in Case 2.

In the energy spectra of these single phase and two phase turbulent flow conditions, the inertial sub-range is observed to be resolved in the simulation domain. The increase of energy input, the increase dispersed phase volume fraction, as well as the decrease of viscosity ratio can promote the energy dissipation rate at high wave number. The increase of dispersed phase volume fraction can also makes the distribution of energy contained by eddies more uniform.

In the PDF of vorticity, as can be seen, the increase of energy input results in the increase of vorticity magnitude in flow field, the increase of dispersed phase volume fraction results in the decrease of vorticity magnitude, the variance of viscosity does not change the PDF of vorticity. Combined with the conclusion about energy dissipation rate at high wave number, it is assumed that the variance of ability at which the turbulent kinetic energy is transferred into thermal energy has no relevance on the variance of PDF of vorticity.

In the PDF of normalized energy dissipation rate, the increase of energy input results in the decrease of the probability density of normalized dissipation rate. The introduction of drop increases the PDF of normalized dissipation rate at low rate region: the maximum probability density increases from 2.4×10^6 to 3.5×10^6 with the increase of ϕ from 0 to 20%. At high rate region, the probability density in high dispersed phase volume fraction is smaller. As for

the viscosity ratio, the effect on PDF of normalized energy dissipation rate is not obvious.

In the PDF of eigenvalues of strain tensor, both the increase of energy input, and the decrease of dispersed phase volume fraction can result in the decrease of the negative eigenvalue λ_1, as well as the increase of middle eigenvalue λ_2 and positive eigenvalue λ_3, they promote the streching. The variance of viscosity ratio does not change the PDF of eigenvalues.

Coherent structures are extracted by the normalized Q_n criterion. The effect of threshold value Q_{th} and the cutoff volume V_{cr} were investigated. It is found that the increase of Q_{th} results in the increase of number of structures and decrease of size of structures, the increase of V_{cr} results in the decrease of the small structures, the deduction of noise is obvious at the beginning of the increase of V_{cr}, the deduction effect is recessionary with the continuous increase of V_{cr}. The combination of $Q_{th} = 0.8$ and $V_{cr}/V_{max} = 4\%$ is found to be a suitable group for extraction of all cases in this study. This combination of parameter ensures that the extracted volume of structures is enough for visualization, and prevents a overcrowd visualization.

The coherent structures in three cases of different energy input are extracted, the corresponding PDF of volume-equivalent diameter reveals that the lower energy input results in the larger size of structures.

A general assumption in breakup models is that only vortices of size equal to or smaller than the drop size are responsible for breakup event. To verify the assumption, the liquid-liquid interface together with coherent structures at the time instants right before and after breakup are tracked and extracted, the corresponding PDF of volume-equivalent diameter is also given. The simulation results reveal that the coherent structures are all of size equal to or

smaller than the drop size, the assumption is validated in this study.

The development of tools for this study is based on Fortran 90 and MATLAB 2017. In Fortran 90, the MRT collision operator is implemented for the lattice Boltzmann method. In MATLAB 2017, the program of the normalized Q_n criterion coupled with the newly designed boundary identification method is created; the calculation of three-dimensional energy spectra is also fulfilled in MATLAB 2017. Besides, other analysis tools of statistical characteristics of liquid-liquid turbulent flow are developed relying on the interaction of Fortran 90 and MATLAB 2017. The visualization is finished in Paraview. These tools improve the stability of generation of turbulence, provide analysis and visualization of drop/vortex interaction as well as the statistical characteristics of turbulence. They are significant for the later use in the breakup kernel development for PBE.

Chapter 5

Future work

In this study, several numerical tools to analyze large-scale sets of data were developed. Together with the DNSs approach, this framework is a strong foundation to study fundamentals of drop breakup in turbulent flow and extraction of information relevant for breakup models.

Further comprehension of interaction between vortices and drop is needed. In this study, coherent structures of the entire flow field and liquid-liquid interface are presented for the preliminary understanding of interaction between vortices. In the future, more attention can be paid to coherent structures that close to the drop. Further, the time evolution of important coherent structures close to the drop also deserves an observation.

Deeper investigation of statistical characteristics of liquid-liquid turbulent flow is necessary. Besides the visualization of breakup event and vortices/drop interaction, the analysis of statistical characteristics such as the correlation of transversal and longitudinal velocity provides a quantitative description and comprehension of the binary system.

Finally, the development of more efficient analysis tools is recommended.

For example, it takes hours to extract coherent structures for only one case in this study. Due the large amount of data generated by DNS, the program for analysis and visualization has to work faster in order to investigate more cases from different aspects.

Bibliography

[1] K Bäumler, M Wegener, AR Paschedag, and E Bänsch. Drop rise velocities and fluid dynamic behavior in standard test systems for liquid/liquid extraction—experimental and numerical investigations. *Chem. Eng. Sci.*, 66(3):426–439, 2011.

[2] MA Farajzadeh, M Bahram, S Zorita, and BG Mehr. Optimization and application of homogeneous liquid–liquid extraction in preconcentration of copper (II) in a ternary solvent system. *J. Hazard. Mater.*, 161(2-3): 1535–1543, 2009.

[3] F Pena-Pereira, I Lavilla, and C Bendicho. Miniaturized preconcentration methods based on liquid–liquid extraction and their application in inorganic ultratrace analysis and speciation: A review. *Spectrochim. Acta, Part B*, 64(1):1–15, 2009.

[4] D Attwood. *Surfactant systems: their chemistry, pharmacy and biology.* Springer Science & Business Media, 2012.

[5] PG Mazzola, AM Lopes, FA Hasmann, AF Jozala, TCV Penna, PO Magalhaes, CO Rangel-Yagui, and JA Pessoa. Liquid–liquid extraction of

biomolecules: an overview and update of the main techniques. *J. Chem. Technol. Biotechnol.*, 83(2):143–157, 2008.

[6] SC Cunha and JO Fernandes. Quantification of free and total bisphenol A and bisphenol B in human urine by dispersive liquid–liquid microextraction (DLLME) and heart-cutting multidimensional gas chromatography–mass spectrometry (MD–GC/MS). *Talanta*, 83(1): 117–125, 2010.

[7] Gh Bahrami, SH Mirzaeei, and A Kiani. Determination of acyclovir in human serum by high-performance liquid chromatography using liquid–liquid extraction and its application in pharmacokinetic studies. *J. Chromatogr. B*, 816(1-2):327–331, 2005.

[8] S Liu, Q Li, and Y Shao. Electrochemistry at micro-and nanoscopic liquid/liquid interfaces. *Chem. Soc. Rev.*, 40(5):2236–2253, 2011.

[9] C Bora and SK Dolui. Fabrication of polypyrrole/graphene oxide nanocomposites by liquid/liquid interfacial polymerization and evaluation of their optical, electrical and electrochemical properties. *Comput. Theor. Polym. Sci.*, 53(4):923–932, 2012.

[10] G Herzog, V Kam, and DWM Arrigan. Electrochemical behaviour of haemoglobin at the liquid/liquid interface. *Electrochim. Acta*, 53(24): 7204–7209, 2008.

[11] AG Volkov. *Liquid interfaces in chemical, biological and pharmaceutical applications*. CRC Press, 2001.

[12] H Kuramochi, K Maeda, S Kato, M Osako, K Nakamura, and S Sakai.

Application of UNIFAC models for prediction of vapor-liquid and liquid-liquid equilibria relevant to separation and purification processes of crude biodiesel fuel. *Fuel*, 88(8):1472–1477, 2009.

[13] AA Hyman, CA Weber, and F Jülicher. Liquid-liquid phase separation in biology. *Annu. Rev. Cell Dev. Biol.*, 30:39–58, 2014.

[14] KAP Colati, GP Dalmaschio, EVR de Castro, AO Gomes, BG Vaz, and W Romão. Monitoring the liquid/liquid extraction of naphthenic acids in brazilian crude oil using electrospray ionization FT-ICR mass spectrometry (ESI FT-ICR MS). *Fuel*, 108:647–655, 2013.

[15] L Yang, BJ Azzopardi, A Belghazi, and S Nakanishi. Phase separation of liquid-liquid two-phase flow at a T-junction. *AlChE J.*, 52(1):141–149, 2006.

[16] AZ Hezave, S Dorostkar, S Ayatollahi, M Nabipour, and B Hemmateenejad. Investigating the effect of ionic liquid (1-dodecyl-3-methylimidazolium chloride ([C12mim][Cl])) on the water/oil interfacial tension as a novel surfactant. *Colloids Surf., A*, 421:63–71, 2013.

[17] X Wen, Q Yang, Z Yan, and Q Deng. Determination of cadmium and copper in water and food samples by dispersive liquid-liquid microextraction combined with UV-vis spectrophotometry. *Microchem. J.*, 97 (2):249–254, 2011.

[18] H Abdolmohammad-Zadeh and GH Sadeghi. Combination of ionic liquid-based dispersive liquid–liquid micro-extraction with stopped-flow spectrofluorometry for the pre-concentration and determination of alu-

minum in natural waters, fruit juice and food samples. *Talanta*, 81(3): 778–785, 2010.

[19] P Viñas, N Campillo, I López-García, and M Hernández-Córdoba. Dispersive liquid–liquid microextraction in food analysis. a critical review. *Anal. Bioanal.Chem.*, 406(8):2067–2099, 2014.

[20] X Jia, Y Han, C Wei, T Duan, and H Chen. Speciation of mercury in liquid cosmetic samples by ionic liquid based dispersive liquid–liquid microextraction combined with high-performance liquid chromatography-inductively coupled plasma mass spectrometry. *J. Anal. At. Spectrom.*, 26(7):1380–1386, 2011.

[21] N Cabaleiro, I De La Calle, C Bendicho, and I Lavilla. Current trends in liquid–liquid and solid–liquid extraction for cosmetic analysis: a review. *Anal. Methods*, 5(2):323–340, 2013.

[22] N Memon, MI Bhanger, and MY Khuhawer. Determination of preservatives in cosmetics and food samples by micellar liquid chromatography. *J. Sep. Sci.*, 28(7):635–638, 2005.

[23] B Khan, X Qiao, and LS Lee. Stereoselective sorption by agricultural soils and liquid–liquid partitioning of trenbolone (17α and 17β) and trendione. *Environ. Sci. Technol.*, 43(23):8827–8833, 2009.

[24] A Bidari, MR Ganjali, P Norouzi, MRM Hosseini, and Y Assadi. Sample preparation method for the analysis of some organophosphorus pesticides residues in tomato by ultrasound-assisted solvent extraction followed by dispersive liquid–liquid microextraction. *Food Chem.*, 126(4):1840–1844, 2011.

[25] Q Wu, X Zhou, Y Li, X Zang, C Wang, and Z Wang. Application of dispersive liquid–liquid microextraction combined with high-performance liquid chromatography to the determination of carbamate pesticides in water samples. *Anal. Bioanal.Chem.*, 393(6-7):1755–1761, 2009.

[26] VA Atiemo-Obeng, SM Kresta, and EL Paul. *Handbook of Industrial Mixing*. John Wiley & Sons, Incorporated, 2004.

[27] F Valenzuela, C Fonseca, C Basualto, O Correa, C Tapia, and J Sapag. Removal of copper ions from a waste mine water by a liquid emulsion membrane method. *Miner. Eng.*, 18(1):33–40, 2005.

[28] D Maggioris, A Goulas, AH Alexopoulos, EG Chatzi, and C Kiparissides. Prediction of particle size distribution in suspension polymerization reactors: effect of turbulence nonhomogeneity. *Chem. Eng. Sci.*, 55(20): 4611–4627, 2000.

[29] AW Nienow, MF Edwards, and N Harnby. *Mixing in the process industries*. Butterworth-Heinemann, 1997.

[30] PA Zapata, J Faria, MP Ruiz, RE Jentoft, and DE Resasco. Hydrophobic zeolites for biofuel upgrading reactions at the liquid–liquid interface in water/oil emulsions. *JACS*, 134(20):8570–8578, 2012.

[31] M Djenouhat, O Hamdaoui, M Chiha, and MH Samar. Ultrasonication-assisted preparation of water-in-oil emulsions and application to the removal of cationic dyes from water by emulsion liquid membrane: Part 1: Membrane stability. *Sep. Purif. Technol.*, 62(3):636–641, 2008.

[32] S Crossley, J Faria, M Shen, and DE Resasco. Solid nanoparticles that

catalyze biofuel upgrade reactions at the water/oil interface. *Science*, 327(5961):68–72, 2010.

[33] PS Nigam and A Singh. Production of liquid biofuels from renewable resources. *Prog. Energy Combust. Sci.*, 37(1):52–68, 2011.

[34] B Berge. Liquid lens technology: principle of electrowetting based lenses and applications to imaging. In *Micro Electro Mechanical Systems, 2005. MEMS 2005. 18th IEEE International Conference on*, pages 227–230. IEEE, 2005.

[35] L Dong, AK Agarwal, DJ Beebe, and H Jiang. Adaptive liquid microlenses activated by stimuli-responsive hydrogels. *Nature*, 442(7102): 551, 2006.

[36] M Gharehbaghi and F Shemirani. A novel method for dye removal: ionic liquid-based dispersive liquid-liquid extraction (IL-DLLE). *CLEAN–Soil, Air, Water*, 40(3):290–297, 2012.

[37] A Soto, A Arce, and MK Khoshkbarchi. Partitioning of antibiotics in a two-liquid phase system formed by water and a room temperature ionic liquid. *Sep. Purif. Technol.*, 44(3):242–246, 2005.

[38] NB Raikar, SR Bhatia, MF Malone, and MA Henson. Experimental studies and population balance equation models for breakage prediction of emulsion drop size distributions. *Chem. Eng. Sci.*, 64(10):2433–2447, 2009.

[39] J Lovick, AA Mouza, SV Paras, GJ Lye, and P Angeli. Drop size distribution in highly concentrated liquid–liquid dispersions using a light

back scattering method. *J. Chem. Technol. Biotechnol.*, 80(5):483–601, 2005.

[40] S Maaß, F Metz, T Rehm, and M Kraume. Prediction of drop sizes for liquid-liquid systems in stirred slim reactors-Part I: Single stage impellers. *Open Chem. Eng. J. Open*, 162(2):792–801, 2010.

[41] CW Angle and HA Hamza. Predicting the sizes of toluene-diluted heavy oil emulsions in turbulent flow Part 2: Hinze-Kolmogorov based model adapted for increased oil fractions and energy dissipation in a stirred tank. *Chem. Eng. Sci.*, 61(22):7325–7335, 2006.

[42] SN Maindarkar, H Hoogland, and MA Henson. Predicting the combined effects of oil and surfactant concentrations on the drop size distributions of homogenized emulsions. *Colloids Surf., A*, 467:18–30, 2015.

[43] S Maaß, N Paul, and M Kraume. Influence of the dispersed phase fraction on experimental and predicted drop size distributions in breakage dominated stirred systems. *Chem. Eng. Sci.*, 76:140–153, 2012.

[44] JO Hinze. Fundamentals of the hydrodynamic mechanism of splitting in dispersion processes. *AlChE J.*, 1(3):289–295, 1955.

[45] GI Taylor. The formation of emulsions in definable fields of flow. *Proc. R. Soc. London, Ser. A, Containing Papers of a Mathematical and Physical Character*, 146(858):501–523, 1934.

[46] GE Charles and SG Mason. The coalescence of liquid drops with flat liquid/liquid interfaces. *Journal of Colloid Science*, 15(3):236–267, 1960.

[47] AE Handlos and T Baron. Mass and heat transfer from drops in liquid-liquid extraction. *AlChE J.*, 3(1):127–136, 1957.

[48] M Ashar, D Arlov, F Carlsson, F Innings, and R Andersson. Single droplet breakup in a rotor-stator mixer. *Chem. Eng. Sci.*, 181:186–198, 2018.

[49] S Maaß and M Kraume. Determination of breakage rates using single drop experiments. *Chem. Eng. Sci.*, 70:146–164, 2012.

[50] C Ortiz-Dueñas, J Kim, and EK Longmire. Investigation of liquid-liquid drop coalescence using tomographic PIV. *Exp. Fluids*, 49(1):111–129, 2010.

[51] FJE Svensson and A Rasmuson. PIV measurements in a liquid-liquid system at volume percentages up to 10% dispersed phase. *Exp. Fluids*, 41(6):917–931, 2006.

[52] BO Hasan. Breakage of drops and bubbles in a stirred tank: A review of experimental studies. *Chin. J. Chem. Eng.*, 25(6):698–711, 2017.

[53] D Cheng, X Feng, J Cheng, and C Yang. Numerical simulation of macromixing in liquid–liquid stirred tanks. *Chem. Eng. Sci.*, 101:272–282, 2013.

[54] Y Zhao, X Li, J Cheng, C Yang, and Z Mao. Experimental study on liquid–liquid macromixing in a stirred tank. *Ind. Eng. Chem. Res.*, 50 (10):5952–5958, 2011.

[55] M Jahoda, M Moštěk, A Kukuková, and V Machoň. CFD modelling of

liquid homogenization in stirred tanks with one and two impellers using large eddy simulation. *Chem. Eng. Res. Des.*, 85(5):616–625, 2007.

[56] Feng Wang and Zai-Sha Mao. Numerical and experimental investigation of liquid- liquid two-phase flow in stirred tanks. *Industrial & engineering chemistry research*, 44(15):5776–5787, 2005.

[57] Y Yu, H Chen, Y Liu, VSJ Craig, C Wang, LH Li, and Y Chen. Superhydrophobic and superoleophilic porous boron nitride nanosheet/polyvinylidene fluoride composite material for oil-polluted water cleanup. *Adv. Mater. Interfaces*, 2(1):1400267, 2015.

[58] BV Balakin, AC Hoffmann, and P Kosinski. Coupling STAR-CD with a population-balance technique based on the classes method. *Powder Technol.*, 257:47–54, 2014.

[59] A Vikhansky and A Splawski. Adaptive multiply size group method for CFD-population balance modelling of polydisperse flows. *Can. J. Chem. Eng.*, 93(8):1327–1334, 2015.

[60] A Buffo, M Vanni, DL Marchisio, and RO Fox. Multivariate quadrature-based moments methods for turbulent polydisperse gas–liquid systems. *Int. J. Multiphase Flow*, 50:41–57, 2013.

[61] A Amokrane, S Charton, N Sheibat-Othman, J Becker, JP Klein, and F Puel. Development of a CFD-PBE coupled model for the simulation of the drops behaviour in a pulsed column. *Can. J. Chem. Eng.*, 92(2): 220–233, 2014.

[62] Z Li, J Kessel, G Grünewald, and M Kind. Coupled CFD-PBE simulation of nucleation in fluidized bed spray granulation. *Drying Technol.*, 31(15):1888–1896, 2013.

[63] H Zhang, K Zhang, and S Fan. CFD simulation coupled with population balance equations for aerated stirred bioreactors. *Eng. Life Sci.*, 9(6): 421–430, 2009.

[64] DL Wright, R McGraw, and DE Rosner. Bivariate extension of the quadrature method of moments for modeling simultaneous coagulation and sintering of particle populations. *J. Colloid Interface Sci.*, 236(2): 242–251, 2001.

[65] IT Cameron, FY Wang, CD Immanuel, and F Stepanek. Process systems modelling and applications in granulation: A review. *Chem. Eng. Sci.*, 60(14):3723–3750, 2005.

[66] DD Eberl, DE Kile, and VA Drits. On geological interpretations of crystal size distributions: Constant vs. proportionate growth. *Am. Mineral.*, 87(8-9):1235–1241, 2002.

[67] HC Schwarzer, F Schwertfirm, M Manhart, HJ Schmid, and W Peukert. Predictive simulation of nanoparticle precipitation based on the population balance equation. *Chem. Eng. Sci.*, 61(1):167–181, 2006.

[68] SM Iveson. Limitations of one-dimensional population balance models of wet granulation processes. *Powder Technol.*, 124(3):219–229, 2002.

[69] H Briesen. Simulation of crystal size and shape by means of a reduced

two-dimensional population balance model. *Chem. Eng. Sci.*, 61(1):104–112, 2006.

[70] T Wang, J Wang, and Y Jin. Population balance model for gas- liquid flows: Influence of bubble coalescence and breakup models. *Ind. Eng. Chem. Res.*, 44(19):7540–7549, 2005.

[71] M Vanni. Approximate population balance equations for aggregation–breakage processes. *J. Colloid Interface Sci.*, 221(2):143–160, 2000.

[72] MM Attarakih, HJ Bart, and NM Faqir. Numerical solution of the bivariate population balance equation for the interacting hydrodynamics and mass transfer in liquid–liquid extraction columns. *Chem. Eng. Sci.*, 61(1):113–123, 2006.

[73] T Wang, J Wang, and Y Jin. A CFD-PBM coupled model for gas-liquid flows. *AlChE J.*, 52(1):125–140, 2006.

[74] CD Immanuel and FJ Doyle III. Solution technique for a multi-dimensional population balance model describing granulation processes. *Powder Technol.*, 156(2-3):213–225, 2005.

[75] Cs Mihálykó, BG Lakatos, A Matejdesz, and T Blickle. Population balance model for particle-to-particle heat transfer in gas–solid systems. *Int. J. Heat Mass Transfer*, 47(6-7):1325–1334, 2004.

[76] D Ramkrishna. *Population balances: Theory and applications to particulate systems in engineering*. Elsevier, 2000.

[77] C Drumm. Coupling of computational fluid dynamics and population balance modelling for liquid-liquid extraction. 2010.

[78] Z Gao, D Li, A Buffo, W Podgórska, and DL Marchisio. Simulation of droplet breakage in turbulent liquid-liquid dispersions with CFD-PBM: comparison of breakage kernels. *Chem. Eng. Sci.*, 142:277–288, 2016.

[79] C Tsouris and LL Tavlarides. Breakage and coalescence models for drops in turbulent dispersions. *AlChE J.*, 40(3):395–406, 1994.

[80] H Luo and HF Svendsen. Theoretical model for drop and bubble breakup in turbulent dispersions. *AlChE J.*, 42(5):1225–1233, 1996.

[81] CH Lee, LE Erickson, and LA Glasgow. Bubble breakup and coalescence in turbulent gas-liquid dispersions. *Chem. Eng. Commun.*, 59(1-6):65–84, 1987.

[82] L Hagesaether, HA Jakobsen, and HF Svendsen. A model for turbulent binary breakup of dispersed fluid particles. *Chem. Eng. Sci.*, 57(16): 3251–3267, 2002.

[83] T Wang, J Wang, and Y Jin. A novel theoretical breakup kernel function for bubbles/droplets in a turbulent flow. *Chem. Eng. Sci.*, 58(20):4629–4637, 2003.

[84] CA Coulaloglou and LL Tavlarides. Description of interaction processes in agitated liquid-liquid dispersions. *Chem. Eng. Sci.*, 32(11):1289–1297, 1977.

[85] J Baldyga and W Podgórska. Drop break-up in intermittent turbulence: Maximum stable and transient sizes of drops. *Can. J. Chem. Eng.*, 76 (3):456–470, 1998.

[86] W Podgórska and J Bałdyga. Drop break-up and coalescence in a stirred tank. *Task Quarterly*, 7(3):409–424, 2003.

[87] C Meneveau and KR Sreenivasan. The multifractal nature of turbulent energy dissipation. *J. Fluid Mech.*, 224:429–484, 1991.

[88] R Andersson and B Andersson. Modeling the breakup of fluid particles in turbulent flows. *AlChE J.*, 52(6):2031–2038, 2006.

[89] RP Hesketh, AW Etchells, and TWF Russell. Experimental observations of bubble breakage in turbulent flow. *Ind. Eng. Chem. Res.*, 30(5):835–841, 1991.

[90] F Lehr and D Mewes. A transport equation for the interfacial area density applied to bubble columns. *Chem. Eng. Sci.*, 56(3):1159–1166, 2001.

[91] M Lesieur. *Turbulence in fluids*, volume 40. Springer Science & Business Media, 2012.

[92] J Jeong and F Hussain. On the identification of a vortex. *J. Fluid Mech.*, 285:69–94, 1995.

[93] HE Fiedler. Coherent structures in turbulent flows. *Prog. Aerosp. Sci.*, 25(3):231–269, 1988.

[94] AKMF Hussain. Coherent structures—reality and myth. *Phys. Fluids*, 26(10):2816–2850, 1983.

[95] RJ Adrian, KT Christensen, and Z-C Liu. Analysis and interpretation of instantaneous turbulent velocity fields. *Exp. Fluids*, 29(3):275–290, 2000.

[96] Y Dubief and F Delcayre. On coherent-vortex identification in turbulence. *J. Turbul.*, 1(1):011–011, 2000.

[97] WA Kareem, S Izawa, AK Xiong, and Y Fukunishi. Extraction and tracking of multi-scaled vortices from a homogeneous isotropic turbulent field. *J. Turbul.*, (8):N12, 2007.

[98] F Ghasempour, R Andersson, N Kevlahan, and B Andersson. Multidimensional turbulence spectra–identifying properties of turbulent structures. In *J. Phys. Conf. Ser.*, volume 318, page 042022. IOP Publishing, 2011.

[99] F Ghasempour, R Andersson, and B Andersson. Identification and characterization of three-dimensional turbulent flow structures. *AlChE J.*, 62(4):1265–1277, 2016.

[100] M Farge and K Schneider. Coherent vortex simulation (CVS), a semi-deterministic turbulence model using wavelets. *Flow, Turbulence and Combustion*, 66(4):393–426, 2001.

[101] M Farge, G Pellegrino, and K Schneider. Coherent vortex extraction in 3D turbulent flows using orthogonal wavelets. *Phys. Rev. Lett.*, 87(5): 054501, 2001.

[102] K Schneider, M Farge, G Pellegrino, and MM Rogers. Coherent vortex simulation of three-dimensional turbulent mixing layers using orthogonal wavelets. *J. Fluid Mech.*, 534:39–66, 2005.

[103] G Haller and G Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D*, 147(3-4):352–370, 2000.

[104] MA Green, CW Rowley, and G Haller. Detection of Lagrangian coherent structures in three-dimensional turbulence. *J. Fluid Mech.*, 572:111–120, 2007.

[105] A Turiel, J Isern-Fontanet, and E García-Ladona. Wavelet filtering to extract coherent vortices from altimetric data. *J. Atmos. Oceanic Technol.*, 24(12):2103–2119, 2007.

[106] T Miyauchi and M Tanahashi. Coherent fine scale structure in turbulence. In *IUTAM Symposium on Geometry and Statistics of Turbulence*, pages 67–76. Springer, 2001.

[107] G Samanta, GM Oxberry, AN Beris, RA Handler, and KD Housiadas. Time-evolution K-L analysis of coherent structures based on DNS of turbulent Newtonian and viscoelastic flows. *J. Turbul.*, (9):N41, 2008.

[108] O Grulke, F Greiner, T Klinger, and A Piel. Comparative experimental study of coherent structures in a simple magnetized torus. *Plasma Phys. Controlled Fusion*, 43(4):525, 2001.

[109] PJ Staplehurst, PA Davidson, and SB Dalziel. Structure formation in homogeneous freely decaying rotating turbulence. *J. Fluid Mech.*, 598: 81–105, 2008.

[110] PA Davidson, PJ Staplehurst, and SB Dalziel. On the evolution of eddies in a rapidly rotating system. *J. Fluid Mech.*, 557:135–144, 2006.

[111] JJ Derksen and HEA Van Den Akker. Multi-scale simulations of stirred liquid–liquid dispersions. *Chem. Eng. Res. Des.*, 85(5):697–702, 2007.

[112] AE Komrakova, Orest Shardt, D Eskin, and JJ Derksen. Lattice Boltzmann simulations of drop deformation and breakup in shear flow. *International Journal of Multiphase Flow*, 59:24–43, 2014.

[113] AE Komrakova, D Eskin, and JJ Derksen. Numerical study of turbulent liquid-liquid dispersions. *AlChE J.*, 61(8):2618–2633, 2015.

[114] Y Hagiwara, Y Takashina, and M Tanaka. Direct numerical simulation of the basic phase-interactions in liquid turbulent channel flow with immiscible droplets. *Nucl. Eng. Des.*, 175(1-2):49–57, 1997.

[115] NK-R Kevlahan. Rapid distortion of turbulent structures. In *Advances in Turbulence IV*, pages 411–415. Springer, 1993.

[116] L Scarbolo and A Soldati. Turbulence modulation across the interface of a large deformable drop. *J. Turbul.*, 14(11):27–43, 2013.

[117] A Roccon, M De Paoli, F Zonta, and A Soldati. Viscosity-modulated breakup and coalescence of large drops in bounded turbulence. *Phys. Rev. Fluids*, 2(8):083603, 2017.

[118] T Iwasaki, K Nishimura, M Tanaka, and Y Hagiwara. Direct numerical simulation of turbulent couette flow with immiscible droplets. *Int. J. Heat Fluid Flow*, 22(3):332–342, 2001.

[119] T Yuge and Y Hagiwara. The modifications of near-wall turbulence structure and heat transfer by immiscible droplets in turbulent liquid–liquid two-phase flow. *Int. J. Heat Fluid Flow*, 25(3):471–480, 2004.

[120] MR Swift, E Orlandini, WR Osborn, and JM Yeomans. Lattice Boltz-

mann simulations of liquid-gas and binary fluid systems. *Phys. Rev. E*, 54(5):5041, 1996.

[121] D Jacqmin. Calculation of two-phase Navier-Stokes flows using phase-field modeling. *J. Comput. Phys.*, 155(1):96–127, 1999.

[122] F Magaletti, F Picano, M Chinappi, L Marino, and CM Casciola. The sharp-interface limit of the Cahn-Hilliard/Navier-Stokes model for binary fluids. *J. Fluid Mech.*, 714:95–126, 2013.

[123] P Yue, JJ Feng, C Liu, and J Shen. A diffuse-interface method for simulating two-phase flows of complex fluids. *J. Fluid Mech.*, 515:293–317, 2004.

[124] VE Badalassi, HD Ceniceros, and S Banerjee. Computation of multiphase systems with phase field models. *J. Comput. Phys.*, 190(2): 371–397, 2003.

[125] JW Cahn and JE Hilliard. Free energy of a nonuniform system. I. Interfacial free energy. *J. Chem. Phys.*, 28(2):258–267, 1958.

[126] O Penrose and PC Fife. Thermodynamically consistent models of phase-field type for the kinetic of phase transitions. *Physica D*, 43(1):44–62, 1990.

[127] AJ Bray. Theory of phase-ordering kinetics. *Adv. Phys.*, 51(2):481–587, 2002.

[128] VM Kendon, ME Cates, I Pagonabarraga, J-C Desplat, and P Bladon. Inertial effects in three-dimensional spinodal decomposition of a sym-

metric binary fluid mixture: a lattice Boltzmann study. *J. Fluid Mech.*, 440:147–203, 2001.

[129] *Lattice Boltzmann method: fundamentals and engineering applications with computer codes, author=Mohamad, AA*. Springer Science & Business Media, 2011.

[130] PL Bhatnagar, EP Gross, and M Krook. A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. *Phys. Rev.*, 94(3):511, 1954.

[131] D Zhang, K Papadikis, and S Gu. Three-dimensional multi-relaxation time lattice-Boltzmann model for the drop impact on a dry surface at large density ratio. *Int. J. Multiphase Flow*, 64:11–18, 2014.

[132] AE Komrakova, D Eskin, and JJ Derksen. Lattice Boltzmann simulations of a single n-butanol drop rising in water. *Phys. Fluids*, 25(4): 042102, 2013.

[133] KN Premnath and J Abraham. Three-dimensional multi-relaxation time (MRT) lattice-Boltzmann models for multiphase flow. *J. Comput. Phys.*, 224(2):539–559, 2007.

[134] D d'Humières. Multiple-relaxation-time lattice Boltzmann models in three dimensions. *Philos. Trans. R. Soc. London, Ser. A*, 360(1792): 437–451, 2002.

[135] H Yu, LS Luo, and SS Girimaji. LES of turbulent square jet flow using an MRT lattice Boltzmann model. *Comput. Fluids*, 35(8-9):957–965, 2006.

[136] P Lallemand and LS Luo. Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. *Phys. Rev. E*, 61(6):6546, 2000.

[137] TS Lundgren. Linearly forces isotropic turbulence. Technical report, Minnesota Univ Minneapolis, 2003.

[138] JJ Derksen. Flow-induced forces in sphere doublets. *J. Fluid Mech.*, 608:337–356, 2008.

[139] L Valiño, J Martín, and G Házi. Dynamics of isotropic homogeneous turbulence with linear forcing using a lattice Boltzmann method. *Flow, turbulence and combustion*, 84(2):219, 2010.

[140] SB Pope. Turbulent flows, 2001.

[141] DK Wilson. Three-dimensional correlation and spectral functions for turbulent velocities in homogeneous and surface-blocked boundary layers. Technical report, Army Research Lab Adelphi MD, 1997.

[142] PG Saffman. The large-scale structure of homogeneous turbulence. *J. Fluid Mech.*, 27(3):581–593, 1967.

[143] T Tatsumi. The theory of decay process of incompressible isotropic turbulence. *Proc. R. Soc. Lond. A*, 239(1216):16–45, 1957.

[144] D Xu and J Chen. Accurate estimate of turbulent dissipation rate using PIV data. *Exp. Therm Fluid Sci.*, 44:662–672, 2013.

[145] D Saha, M Soos, B Luthi, M Holzner, A Liberzon, MU Babler, and W Kinzelbach. Experimental characterization of breakage rate of col-

loidal aggregates in axisymmetric extensional flow. *Langmuir*, 30(48): 14385–14395, 2014.

[146] S Elghobashi and GC Truesdell. On the two-way interaction between homogeneous turbulence and dispersed solid particles. I: Turbulence modification. *Phys. Fluids A*, 5(7):1790–1801, 1993.

Appendix A

MRT implementation

The transformation matrix $\boldsymbol{M}$ is derived as:

$$\begin{vmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
-30 & -11 & -11 & -11 & -11 & -11 & -11 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
12 & -4 & -4 & -4 & -4 & -4 & -4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -4 & 4 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 2 & 2 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 \\
0 & -4 & -4 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & -2 & -2 & 2 & 2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1
\end{vmatrix}$$

The 19 orthogonal basis vectors $\varsigma_{q\beta}$ are given:

$$
\begin{aligned}
\varsigma_{0\beta} &= \| e_\beta \| \\
\varsigma_{1\beta} &= 19 \| e_\beta \|^2 - 30 \\
\varsigma_{2\beta} &= (21 \| e_\beta \|^4 - 53 \| e_\beta \|^2 + 24)/2 \\
\varsigma_{3\beta} &= e_{\beta x} \\
\varsigma_{5\beta} &= e_{\beta y} \\
\varsigma_{7\beta} &= e_{\beta z} \\
\varsigma_{4\beta} &= (5 \| e_\beta \|^2 - 9) e_{\beta x} \\
\varsigma_{6\beta} &= (5 \| e_\beta \|^2 - 9) e_{\beta y} \\
\varsigma_{8\beta} &= (5 \| e_\beta \|^2 - 9) e_{\beta z} \\
\varsigma_{9\beta} &= 3e_{\beta x}^2 - \| e_\beta \|^2 \\
\varsigma_{11\beta} &= e_{\beta y}^2 - e_{\beta z}^2[0] \\
\varsigma_{13\beta} &= e_{\beta x} e_{\beta y} \\
\varsigma_{14\beta} &= e_{\beta y} e_{\beta z} \\
\varsigma_{15\beta} &= e_{\beta z} e_{\beta x} \\
\varsigma_{10\beta} &= (3 \| e_\beta \|^2 - 5)(3e_{\beta x}^2 - \| e_\beta \|^2) \\
\varsigma_{12\beta} &= (3 \| e_\beta \|^2 - 5)(e_{\beta y}^2 - e_{\beta z}^2) \\
\varsigma_{16\beta} &= (e_{\beta y}^2 - e_{\beta z}^2) e_{\beta x} \\
\varsigma_{17\beta} &= (e_{\beta z}^2 - e_{\beta x}^2) e_{\beta y} \\
\varsigma_{18\beta} &= (e_{\beta x}^2 - e_{\beta y}^2) e_{\beta z}
\end{aligned}
\tag{1.1}
$$

where $\beta \in \{0, 1, \ldots, 18\}$.

Appendix B

Procedure and script for 3D fft and energy spectrum

The three dimensional velocity fields u, v, w are inputs, the calculation is in MATLAB. Part of code is referred from the GitHub user 'fdietzsc' , the link is: https://github.com/fdietzsc/hita.

```
clc
clear
filepath1='/home/grad1/cheng/CZ Jan ...
    16/twophase/d300c6_case3/velx/';
filepath2='/home/grad1/cheng/CZ Jan ...
    16/twophase/d300c6_case3/vely/';
filepath3='/home/grad1/cheng/CZ Jan ...
    16/twophase/d300c6_case3/velz/';
filepath4='/home/grad1/cheng/CZ Jan ...
    16/twophase/d300c6_case3/energy spectrum/';
etak=1; %Kolmogrov scale
```

```
nu=(0.0005*0.1+1-0.0005)*8.3E-003;
epsilon=5.8E-007;                    %energy dissipation rate
dim=300;
avespectrum=0;
counter=0;
for t=12:1:12
u=reshape(load([filepath1,'velx',num2str(t,'%04i'),'.dat']),&...
[dim,dim,dim]);
v=reshape(load([filepath2,'vely',num2str(t,'%04i'),'.dat']),&...
[dim,dim,dim]);
w=reshape(load([filepath3,'velz',num2str(t,'%04i'),'.dat']),&...
[dim,dim,dim]);
    uu_fft=fftn(u);
    vv_fft=fftn(v);
    ww_fft=fftn(w);

        muu = abs(uu_fft)/length(u)^3;
        mvv = abs(vv_fft)/length(v)^3;
        mww = abs(ww_fft)/length(w)^3;

        muu = muu.^2;
        mvv = mvv.^2;
        mww = mww.^2;

        k_end = (dim)/2;

    rx=[0:1:dim-1] - (dim)/2+1;
    ry=[0:1:dim-1] - (dim)/2+1;
    rz=[0:1:dim-1] - (dim)/2+1;

    R_x=circshift(rx',[(dim)/2+1 1]);
```

```
39      R_y=circshift(ry',[(dim)/2+1 1]);
40      R_z=circshift(rz',[(dim)/2+1 1]);
41
42      [X,Y,Z]= meshgrid(R_x,R_y,R_z);
43      r=(sqrt(X.^2+Y.^2+Z.^2));
44
45      dx=2*pi/dim;
46      k=[1:k_end].*dx;
47      kll=[1:k_end];
48
49      spectrum=zeros(size(k,2),1);
50      bin_counter=zeros(size(k,2),1);
51      for N=2:k_end-1
52          picker = (r(:,:,:)*dx <= (k(N+1) + k(N))/2) & ...
53                    (r(:,:,:)*dx > (k(N) + k(N-1))/2);
54          spectrum(N) = sum(muu(picker))+...
55                        sum(mvv(picker))+...
56                        sum(mww(picker));
57          bin_counter(N) = size(find(picker==1),1);
58      end
59
60      picker = (r(:,:,:)*dx <= (k(2) + k(1))/2);
61      spectrum(1) = sum(muu(picker))+...
62                    sum(mvv(picker))+...
63                    sum(mww(picker));
64      bin_counter(1) = size(find(picker==1),1);
65
66      picker = (r(:,:,:)*dx > (k(end) + k(end-1))/2 & ...
67                r(:,:,:)*dx <= k(end));
68      spectrum(end) = sum(muu(picker))+...
69                      sum(mvv(picker))+...
```

```
70                      sum(mww(picker));
71      bin_counter(end) = size(find(picker==1),1);
72
73      spectrum = spectrum*2*pi.*k'.^2./(bin_counter.*dx.^3);
74
75      avespectrum=avespectrum+spectrum;
76      counter=counter+1;
77  end
78  avespectrum=avespectrum/counter;
79
80      y = [kll; avespectrum'];
81      fid = fopen([filepath4,'average_spectrum.dat'], 'w');
82      fprintf(fid, '%15.8E %15.8E\n', y);
83      fclose(fid);
84
85      compk=kll;
86      kll=kll/(2*pi/etak);  %scale kll
87  energy=avespectrum/(epsilon^(2/3)*etak^(5/3));
88
89  slope1=epsilon^(2/3)*kll.^(-5/3);
90
91  scaled=[kll;energy';slope1];
92  fid = fopen([filepath4,'average_spectrum_scaled.dat'], 'w');
93      fprintf(fid, '%15.8E %15.8E %15.8E\n', scaled);
94      fclose(fid);
95
96  %-----------------compensated energy spectrum------------
97  compspectrum = avespectrum.*compk'.^(5/3)*epsilon^(-2/3);
98  compk=compk*etak;
99  compensated = [compk;compspectrum'];
100 fid = fopen([filepath4,'average_spectrum_compensated.dat'], ...
```

```
        'w');
101      fprintf(fid, '%15.8E %15.8E\n', compensated);
102      fclose(fid);
103
104  %------------------Dissipation energy spectrum-----------
105  dissspectrum = avespectrum.*compk'.^2*nu;
106  dissk=compk*etak;
107  dissipated = [dissk;dissspectrum'];
108  fid = fopen([filepath4,'average_spectrum_dissipated.dat'], ...
        'w');
109      fprintf(fid, '%15.8E %15.8E\n', dissipated);
110      fclose(fid);
```

Appendix C

Procedure and script for coherent structures

The simulation parameters are set in the 'simparam' module in FORTRAN 90. In the main program, two subroutines are called: getvelocity.f90 and getq.f90. The getvelocity.f90 extract velocity data from original data file, the getq.f90 generated Q_n field from velocity. The main program is:

```
1  !-----------------------------------------
2  !-----------------------------------------
3  !----------------  Simulation parameters
4  !-----------------------------------------
5  !-----------------------------------------
6
7   module simparam
8  !
9  ! Simulation domain
10 !
```

```
11   integer,parameter::ni=300,nj=300,nk=300
12   real(kind=8),parameter::dx=1.0d0,dy=1.0d0,dz=1.0d0
13  !
14  ! Liquid parameter
15   real(kind=8),parameter::nu=8.296E-003 !viscosity
16  !
17  end module simparam
18  !----------------------------------------
19  !----------------------------------------
20  !----------------  Main program
21  !----------------------------------------
22  !----------------------------------------
23  program main
24  use simparam
25  implicit none
26  !-----
27  integer::ifld
28  real(kind=8),dimension(:,:,:),allocatable::vx,vy,vz
29  allocate( vx(ni,nj,nk) )
30  allocate( vy(ni,nj,nk) )
31  allocate( vz(ni,nj,nk) )
32  do ifld=12,12,1
33  !----------------------------------
34  !  Get velocity from VTK file
35  !----------------------------------
36   call getvelocity(ifld,vx,vy,vz)
37  !----------------------------------
38  !  Calculate q values
39  !----------------------------------
40   call getq(ifld,vx,vy,vz)
41  enddo
```

```
42  deallocate(vx)
43  deallocate(vy)
44  deallocate(vz)
45  end program main
```

The getvelocity.f90 subroutine is:

```
1  subroutine getvelocity(ifld,vx,vy,vz)
2  use simparam
3  implicit none
4   integer,intent(inout)::ifld
5   real(kind=8),intent(inout)::vx(ni,nj,nk),vy(ni,nj,nk)
6   real(kind=8),intent(inout)::vz(ni,nj,nk)
7   integer ::jump,i,j,k,p,temp,d0,d1,d2,d3
8   character ::store
9   real(kind=8),dimension(1:ni*nj*nk)::q
10  character(len=71) datfile1
11  character(len=67) datfile2,datfile3,datfile4
12  datfile1(1:63)='/home/grad1/cheng/CZ Jan ...
       16/twophase/d300c6_case3/paraviewv/vlc'    !50+
13  datfile1(68:71)='.vtk'
14  datfile2(1:59)='/home/grad1/cheng/CZ Jan ...
       16/twophase/d300c6_case3/velx/velx'
15  datfile2(64:67)='.dat'
16  datfile3(1:59)='/home/grad1/cheng/CZ Jan ...
       16/twophase/d300c6_case3/vely/vely'
17  datfile3(64:67)='.dat'
18  datfile4(1:59)='/home/grad1/cheng/CZ Jan ...
       16/twophase/d300c6_case3/velz/velz'
19  datfile4(64:67)='.dat'
20
21  102 format (e15.8,1X,e15.8,1X,e15.8)
22  103 format (A)
23  104 format (A,1X,I4,1X,I4,1X,I4)
24  105 format (A,1X,I1,1X,I1,1X,I1)
```

```
25   106 format (A,1X,I8)
26   107 format (f6.3)
27   108 format (f6.3,1X,f6.3,1X,f6.3)
28
29   if (ifld .lt. 10) then
30     d0=0
31     d1=0
32     d2=0
33     d3=ifld
34   endif
35  !-------
36   if ((ifld .ge. 10) .and. (ifld .lt. 100)) then
37     d0=0
38     d1=0
39     d2=inT(ifld/10)
40     d3=ifld-d2*10
41   endif
42  !-------
43   if ((ifld .ge. 100) .and. (ifld .lt. 1000)) then
44     d0=0
45     d1=inT(ifld/100)
46     d2=ifld-d1*100
47     d2=inT(d2/10)
48     d3=ifld-d1*100-d2*10
49   endif
50  !-------
51   if (ifld .ge. 1000) then
52     d0=inT(ifld/1000)
53     d1=ifld-d0*1000
54     d1=inT(d1/100)
55     d2=ifld-d0*1000-d1*100
```

```
56      d2=inT(d2/10)
57      d3=ifld-d0*1000-d1*100-d2*10
58    endif
59    datfile1(64:64)=char(d0+48)
60    datfile1(65:65)=char(d1+48)
61    datfile1(66:66)=char(d2+48)
62    datfile1(67:67)=char(d3+48)
63   !-------
64    datfile2(60:60)=char(d0+48)
65    datfile2(61:61)=char(d1+48)
66    datfile2(62:62)=char(d2+48)
67    datfile2(63:63)=char(d3+48)
68   !-------
69    datfile3(60:60)=char(d0+48)
70    datfile3(61:61)=char(d1+48)
71    datfile3(62:62)=char(d2+48)
72    datfile3(63:63)=char(d3+48)
73   !-------
74    datfile4(60:60)=char(d0+48)
75    datfile4(61:61)=char(d1+48)
76    datfile4(62:62)=char(d2+48)
77    datfile4(63:63)=char(d3+48)
78
79   open(unit=133,file=datfile1,form='formatted',status='unknown')
80   open(unit=212,file=datfile2,form='formatted',status='unknown')
81   open(unit=213,file=datfile3,form='formatted',status='unknown')
82   open(unit=214,file=datfile4,form='formatted',status='unknown')
83
84   do jump=1,9
85      read(133,'(A13)') store
86   enddo
```

```
87
88  do k=1,nk
89  do j=1,nj
90  do i=1,ni
91   read(133,102)vx(i,j,k),vy(i,j,k),vz(i,j,k)
92   write(212,102)vx(i,j,k)
93   write(213,102)vy(i,j,k)
94   write(214,102)vz(i,j,k)
95  enddo
96  enddo
97  enddo
98   close(unit=133)
99   close(unit=212)
100  close(unit=213)
101  close(unit=214)
102
103 end subroutine getvelocity
```

The getq.f90 subroutine is:

```
1  subroutine getq(ifld,vx,vy,vz)
2   use simparam
3   IMPLICIT NONE
4   integer,intent(inout)::ifld
5   real(kind=8),intent(in)::vx(ni,nj,nk),vy(ni,nj,nk),vz(ni,nj,nk)
6   integer ::jump,i,j,k,p,temp,ip,jp,kp,d0,d1,d2,d3
7   character ::store
8   real(kind=8),dimension(1:ni*nj*nk)::q,qq
9   real(kind=8)::omega2,s2,numerator,denominator
10  character(len=67) datfile10,datfile11
11
12  datfile10(1:59)='/home/grad1/cheng/CZ Jan ...
        16/onephase/d300b1_case3/qval/qval'
13  datfile10(64:67)='.dat'
14  datfile11(1:59)='/home/grad1/cheng/CZ Jan ...
        16/onephase/d300b1_case3/qval/qori'
15  datfile11(64:67)='.dat'
16
17  101 format (e15.6,1X,e15.6,1X,e15.6)
18  102 format (E11.3)
19  103 format (A)
20  104 format (A,1X,I4,1X,I4,1X,I4)
21  105 format (A,1X,I1,1X,I1,1X,I1)
22  106 format (A,1X,I8)
23  107 format (f6.3)
24  108 format (f6.3,1X,f6.3,1X,f6.3)
25  109 format (f12.3)
26
```

```
27   if (ifld .lt. 10) then
28     d0=0
29     d1=0
30     d2=0
31     d3=ifld
32   endif
33  !-------
34   if ((ifld .ge. 10) .and. (ifld .lt. 100)) then
35     d0=0
36     d1=0
37     d2=inT(ifld/10)
38     d3=ifld-d2*10
39   endif
40  !-------
41   if ((ifld .ge. 100) .and. (ifld .lt. 1000)) then
42     d0=0
43     d1=inT(ifld/100)
44     d2=ifld-d1*100
45     d2=inT(d2/10)
46     d3=ifld-d1*100-d2*10
47   endif
48  !-------
49   if (ifld .ge. 1000) then
50     d0=inT(ifld/1000)
51     d1=ifld-d0*1000
52     d1=inT(d1/100)
53     d2=ifld-d0*1000-d1*100
54     d2=inT(d2/10)
55     d3=ifld-d0*1000-d1*100-d2*10
56   endif
57  !-------
```

```
 datfile10(60:60)=char(d0+48)
 datfile10(61:61)=char(d1+48)
 datfile10(62:62)=char(d2+48)
 datfile10(63:63)=char(d3+48)
!-------
 datfile11(60:60)=char(d0+48)
 datfile11(61:61)=char(d1+48)
 datfile11(62:62)=char(d2+48)
 datfile11(63:63)=char(d3+48)
!-------
open(unit=216,file=datfile10,form='formatted',status='unknown')
open(unit=217,file=datfile11,form='formatted',status='unknown')

p=0
do k=1,nk
 kp=k+1
 if (kp.gt.nk) kp=kp-nk
do j=1,nj
 jp=j+1
 if (jp.gt.nj) jp=jp-nj
do i=1,ni
 ip=i+1
 if (ip.gt.ni) ip=ip-ni

p=p+1
numerator=-2.0d0*(vx(ip,j,k)-vx(i,j,k))*(vx(ip,j,k)-vx(i,j,k))&
          &/(dx*dx)&
          &-4.0d0*(vx(i,jp,k)-vx(i,j,k))*(vy(ip,j,k)-vy(i,j,k))&
          &/(dy*dx)&
          &-2.0d0*(vy(i,jp,k)-vy(i,j,k))*(vy(i,jp,k)-vy(i,j,k))&
          &/(dy*dy)&
```

```
89            &-4.0d0*(vx(i,j,kp)-vx(i,j,k))*(vz(ip,j,k)-vz(i,j,k))&
90            &/(dz*dx)&
91            &-2.0d0*(vz(i,j,kp)-vz(i,j,k))*(vz(i,j,kp)-vz(i,j,k))&
92            &/(dz*dz)&
93            &-4.0d0*(vy(i,j,kp)-vy(i,j,k))*(vz(i,jp,k)-vz(i,j,k))&
94            &/(dz*dy)
95
96  denominator= ...
        1.0d0*(vx(i,jp,k)-vx(i,j,k))*(vx(i,jp,k)-vx(i,j,k))&
97            &/(dy*dy)&
98            &+1.0d0*(vy(ip,j,k)-vy(i,j,k))*(vy(ip,j,k)-vy(i,j,k))&
99            &/(dx*dx)&
100           &-2.0d0*(vx(i,jp,k)-vx(i,j,k))*(vy(ip,j,k)-vy(i,j,k))&
101           &/(dy*dx)&
102           &+1.0d0*(vx(i,j,kp)-vx(i,j,k))*(vx(i,j,kp)-vx(i,j,k))&
103           &/(dz*dz)&
104           &+1.0d0*(vz(ip,j,k)-vz(i,j,k))*(vz(ip,j,k)-vz(i,j,k))&
105           &/(dx*dx)&
106           &-2.0d0*(vx(i,j,kp)-vx(i,j,k))*(vz(ip,j,k)-vz(i,j,k))&
107           &/(dz*dx)&
108           &+1.0d0*(vy(i,j,kp)-vy(i,j,k))*(vy(i,j,kp)-vy(i,j,k))&
109           &/(dz*dz)&
110           &+1.0d0*(vz(i,jp,k)-vz(i,j,k))*(vz(i,jp,k)-vz(i,j,k))&
111           &/(dz*dz)&
112           &-2.0d0*(vy(i,j,kp)-vy(i,j,k))*(vz(i,jp,k)-vz(i,j,k))&
113           &/(dz*dy)
114
115 q(p)=numerator/denominator
116 qq(p)=0.5d0*numerator
117 enddo
118 enddo
```

```
119  enddo
120
121  do p=1,ni*nj*nk
122  write(216,*)q(p)
123  write(217,*)qq(p)
124  enddo
125
126   close(unit=216)
127   close(unit=217)
128
129  end subroutine getq
```

With the Q_n field, the MATLAB code is then used to extract coherent structures:

```
clc
clear
tic
filepath1='/home/grad1/cheng/CZ Jan ...
    16/twophase/d300a6_case3/qval/';
filepath2='/home/grad1/cheng/CZ Jan ...
    16/twophase/d300a6_case3/qnum/';
dim=300;
hh=waitbar(0,'calculating...');
aa1=22;
aa2=22;
interv=1;
for t=aa1:interv:aa2
q=reshape(load([filepath1,'qval',num2str(t,'%04i'),'.dat'])...
    ,[dim,dim,dim]);    %oringinal q 3d matrix
qcheck=q;
qcheck(qcheck>100)=NaN;
qqq=max(qcheck(:))
% threshold=0.8*qqq;
threshold=0.8;
qboundx(1)=0;
qboundy(1)=0;
qboundz(1)=0;

qboundx(:)=[];
qboundy(:)=[];
qboundz(:)=[];
```

```
26 qnum=zeros(dim,dim,dim);
27 qfinal=zeros(dim,dim,dim);
28 n=1;
29 cutoffvolume=507;
30 q(q<threshold)=0;
31 qchange=q;                                    %changeable q 3d ...
       matrix
32
33 qmax=max(max(max(qchange)));
34 while qmax >= threshold
35     [r,c,v]=ind2sub(size(qchange),find(qchange==qmax));
36     for i=1:length(r)
37     %---------the max q is at point (x,y,z)--------
38     x=r(i);
39     y=c(i);
40     z=v(i);
41     %-----------------------------------------------
42     xloc=x;
43     yloc=y;
44     zloc=z;
45     if qchange(xloc,yloc,zloc)>= threshold
46         thismaxpoint=true;
47         %--------------------large ...
               loop------------------------------
48         while qchange(xloc,yloc,zloc) >= threshold     ...
               %move to +x direction
49             qnum(xloc,yloc,zloc)=n;              %marked ...
                   this point in "nth" vortex
50             xx=xloc;                     %point ...
                   (xloc,yloc,zloc) is regarded as a temporary ...
                   origin
```

```
51              yy=yloc;
52              zz=zloc;
53              %-----litte loop----------------------
54              while qchange(xx,yy,zz) >= threshold
55                  qnum(xx,yy,zz)=n;
56                  xxx=xx;
57                  yyy=yy;
58                  zzz=zz;
59                  %----------very little loope-----
60                  while qchange(xxx,yyy,zzz) >= threshold
61                      qnum(xxx,yyy,zzz)=n;
62                      zzz=zzz+1;  %move to +zzz direction
63                      if zzz>dim
64                          zzz=zzz-dim;
65                      end
66                  end
67                  qboundx(end+1)=xxx;
68                  qboundy(end+1)=yyy;
69                  if zzz==1
70                      qboundz(end+1)=dim;
71                  else
72                  qboundz(end+1)=zzz-1;
73                  end
74                  zzz=zz;
75
76                  while qchange(xxx,yyy,zzz) >= threshold
77                      qnum(xxx,yyy,zzz)=n;
78                      zzz=zzz-1;  %move to -zzz
79                      if zzz<1
80                          zzz=zzz+dim;
81                      end
```

```
82                  end
83                  qboundx(end+1)=xxx;
84                  qboundy(end+1)=yyy;
85                  if zzz==dim;
86                      qboundz(end+1)=1;
87                  else
88                  qboundz(end+1)=zzz+1;
89                  end
90                  zzz=zz;
91                  %-----------end very little loop-----
92
93                  yy=yy+1;            %move to +yy direction
94                  if yy>dim
95                      yy=yy-dim;
96                  end
97              end
98              qboundx(end+1)=xx;
99              if yy==1
100                 qboundy(end+1)=dim;
101             else
102             qboundy(end+1)=yy-1;
103             end
104             qboundz(end+1)=zz;
105             yy=yloc;
106
107             while qchange(xx,yy,zz) >= threshold
108                 qnum(xx,yy,zz)=n;
109                 %----------very little loope-----
110                 while qchange(xxx,yyy,zzz) >= threshold
111                     qnum(xxx,yyy,zzz)=n;
112                     zzz=zzz+1;  %move to +zzz direction
```

```
113                     if zzz>dim
114                         zzz=zzz-dim;
115                     end
116                 end
117                 qboundx(end+1)=xxx;
118                 qboundy(end+1)=yyy;
119                 if zzz==1
120                     qboundz(end+1)=dim;
121                 else
122                 qboundz(end+1)=zzz-1;
123                 end
124                 zzz=zz;
125
126                 while qchange(xxx,yyy,zzz) >= threshold
127                     qnum(xxx,yyy,zzz)=n;
128                     zzz=zzz-1;  %move to -zzz
129                     if zzz<1
130                         zzz=zzz+dim;
131                     end
132                 end
133                 qboundx(end+1)=xxx;
134                 qboundy(end+1)=yyy;
135                 if zzz==dim;
136                     qboundz(end+1)=1;
137                 else
138                 qboundz(end+1)=zzz+1;
139                 end
140                 zzz=zz;
141                 %-----------end very little loop-----
142                 yy=yy-1;           %move to -yy direction
143                 if yy<1
```

```
144                     yy=yy+dim;
145                 end
146             end
147             qboundx(end+1)=xx;
148             if yy==dim
149                 qboundy(end+1)=1;
150             else
151             qboundy(end+1)=yy+1;
152             end
153             qboundz(end+1)=zz;
154             yy=yloc;
155
156             while qchange(xx,yy,zz) >= threshold
157                 qnum(xx,yy,zz)=n;
158                 %----------very little loope-----
159                 while qchange(xxx,yyy,zzz) >= threshold
160                     qnum(xxx,yyy,zzz)=n;
161                     yyy=yyy+1;  %move to +yyy direction
162                     if yyy>dim
163                         yyy=yyy-dim;
164                     end
165                 end
166                 qboundx(end+1)=xxx;
167                 if yyy==1
168                     qboundy(end+1)=dim;
169                 else
170                 qboundy(end+1)=yyy-1;
171                 end
172                 qboundz(end+1)=zzz;
173                 yyy=yy;
174
```

```
175                 while qchange(xxx,yyy,zzz) >= threshold
176                     qnum(xxx,yyy,zzz)=n;
177                     yyy=yyy-1;  %move to -yyy
178                     if yyy<1
179                         yyy=yyy+dim;
180                     end
181                 end
182                 qboundx(end+1)=xxx;
183                 if yyy==dim
184                     qboundy(end+1)=1;
185                 else
186                 qboundy(end+1)=yyy+1;
187                 end
188                 qboundz(end+1)=zzz;
189                 yyy=yy;
190                 %-----------end very little loop-----
191                 zz=zz+1;           %move to +zz direction
192                 if zz>dim
193                     zz=zz-dim;
194                 end
195             end
196             qboundx(end+1)=xx;
197             qboundy(end+1)=yy;
198             if zz==1
199                 qboundz(end+1)=dim;
200             else
201             qboundz(end+1)=zz-1;
202             end
203             zz=zloc;
204
205             while qchange(xx,yy,zz) >= threshold
```

```
206                 qnum(xx,yy,zz)=n;
207                 %----------very little loope-----
208                 while qchange(xxx,yyy,zzz) >= threshold
209                     qnum(xxx,yyy,zzz)=n;
210                     yyy=yyy+1;  %move to +yyy direction
211                     if yyy>dim
212                         yyy=yyy-dim;
213                     end
214                 end
215                 qboundx(end+1)=xxx;
216                 if yyy==1
217                     qboundy(end+1)=dim;
218                 else
219                 qboundy(end+1)=yyy-1;
220                 end
221                 qboundz(end+1)=zzz;
222                 yyy=yy;
223
224                 while qchange(xxx,yyy,zzz) >= threshold
225                     qnum(xxx,yyy,zzz)=n;
226                     yyy=yyy-1;  %move to -yyy
227                     if yyy<1
228                         yyy=yyy+dim;
229                     end
230                 end
231                 qboundx(end+1)=xxx;
232                 if yyy==dim
233                     qboundy(end+1)=1;
234                 else
235                 qboundy(end+1)=yyy+1;
236                 end
```

```
237                 qboundz(end+1)=zzz;
238                 yyy=yy;
239                 %-----------end very little loop-----
240                 zz=zz-1;            %move to -zz direction
241                 if zz<1
242                     zz=zz+dim;
243                 end
244             end
245             qboundx(end+1)=xx;
246             qboundy(end+1)=yy;
247             if zz==dim;
248                 qboundz(end+1)=1;
249             else
250             qboundz(end+1)=zz+1;
251             end
252             zz=zloc;
253             %--------------------end little loop------
254             xloc=xloc+1;
255             if xloc>dim
256                 xloc=xloc-dim;
257             end
258         end
259         if xloc==1
260             qboundx(end+1)=dim;
261         else
262         qboundx(end+1)=xloc-1;
263         end
264         qboundy(end+1)=yloc;
265         qboundz(end+1)=zloc;
266         xloc=x;                                              ...
            %reset xloc to original x (where max q exists)
```

```
267         %------------------------------end large ...
                loop----------------
268
269         %--------------------large ...
                loop-----------------------------
270         while qchange(xloc,yloc,zloc) >= threshold     ...
                %move to -x direction
271             qnum(xloc,yloc,zloc)=n;
272             %-----litte loop----------------------
273             while qchange(xx,yy,zz) >= threshold
274                 qnum(xx,yy,zz)=n;
275                 %----------very little loope-----
276                 while qchange(xxx,yyy,zzz) >= threshold
277                     qnum(xxx,yyy,zzz)=n;
278                     zzz=zzz+1;  %move to +zzz direction
279                     if zzz>dim
280                         zzz=zzz-dim;
281                     end
282                 end
283                 qboundx(end+1)=xxx;
284                 qboundy(end+1)=yyy;
285                 if zzz==1
286                     qboundz(end+1)=dim;
287                 else
288                 qboundz(end+1)=zzz-1;
289                 end
290                 zzz=zz;
291
292                 while qchange(xxx,yyy,zzz) >= threshold
293                     qnum(xxx,yyy,zzz)=n;
294                     zzz=zzz-1;  %move to -zzz
```

```
295                     if zzz<1
296                         zzz=zzz+dim;
297                     end
298                 end
299                 qboundx(end+1)=xxx;
300                 qboundy(end+1)=yyy;
301                 if zzz==dim
302                     qboundz(end+1)=1;
303                 else
304                 qboundz(end+1)=zzz+1;
305                 end
306                 zzz=zz;
307                 %-----------end very little loop-----
308                 yy=yy+1;            %move to +yy direction
309                 if yy>dim
310                     yy=yy-dim;
311                 end
312             end
313             qboundx(end+1)=xx;
314             if yy==1
315                 qboundy(end+1)=dim;
316             else
317             qboundy(end+1)=yy-1;
318             end
319             qboundz(end+1)=zz;
320             yy=yloc;
321
322             while qchange(xx,yy,zz) >= threshold
323                 qnum(xx,yy,zz)=n;
324                 %----------very little loope-----
325                 while qchange(xxx,yyy,zzz) >= threshold
```

```
326                     qnum(xxx,yyy,zzz)=n;
327                     zzz=zzz+1;  %move to +zzz direction
328                     if zzz>dim
329                         zzz=zzz-dim;
330                     end
331                 end
332                 qboundx(end+1)=xxx;
333                 qboundy(end+1)=yyy;
334                 if zzz==1
335                     qboundz(end+1)=dim;
336                 else
337                 qboundz(end+1)=zzz-1;
338                 end
339                 zzz=zz;
340
341                 while qchange(xxx,yyy,zzz) >= threshold
342                     qnum(xxx,yyy,zzz)=n;
343                     zzz=zzz-1;  %move to -zzz
344                     if zzz<1
345                         zzz=zzz+dim;
346                     end
347                 end
348                 qboundx(end+1)=xxx;
349                 qboundy(end+1)=yyy;
350                 if zzz==dim
351                     qboundz(end+1)=1;
352                 else
353                 qboundz(end+1)=zzz+1;
354                 end
355                 zzz=zz;
356                 %----------end very little loop-----
```

```
357                 yy=yy-1;            %move to -yy direction
358                 if yy<1
359                     yy=yy+dim;
360                 end
361             end
362             qboundx(end+1)=xx;
363             if yy==dim
364                 qboundy(end+1)=1;
365             else
366             qboundy(end+1)=yy+1;
367             end
368             qboundz(end+1)=zz;
369             yy=yloc;
370
371             while qchange(xx,yy,zz) >= threshold
372                 qnum(xx,yy,zz)=n;
373                 %----------very little loope-----
374                 while qchange(xxx,yyy,zzz) >= threshold
375                     qnum(xxx,yyy,zzz)=n;
376                     yyy=yyy+1;  %move to +yyy direction
377                     if yyy>dim
378                         yyy=yyy-dim;
379                     end
380                 end
381                 qboundx(end+1)=xxx;
382                 if yyy==1;
383                     qboundy(end+1)=dim;
384                 else
385                 qboundy(end+1)=yyy-1;
386                 end
387                 qboundz(end+1)=zzz;
```

```
388                 yyy=yy;
389
390                 while qchange(xxx,yyy,zzz) >= threshold
391                     qnum(xxx,yyy,zzz)=n;
392                     yyy=yyy-1;  %move to -yyy
393                     if yyy<1
394                         yyy=yyy+dim;
395                     end
396                 end
397                 qboundx(end+1)=xxx;
398                 if yyy==dim
399                     qboundy(end+1)=1;
400                 else
401                 qboundy(end+1)=yyy+1;
402                 end
403                 qboundz(end+1)=zzz;
404                 yyy=yy;
405                 %-----------end very little loop-----
406                 zz=zz+1;            %move to +zz direction
407                 if zz>dim
408                     zz=zz-dim;
409                 end
410             end
411             qboundx(end+1)=xx;
412             qboundy(end+1)=yy;
413             if zz==1
414                 qboundz(end+1)=dim;
415             else
416             qboundz(end+1)=zz-1;
417             end
418             zz=zloc;
```

```
419
420             while qchange(xx,yy,zz) >= threshold
421                 qnum(xx,yy,zz)=n;
422                 %----------very little loope-----
423                 while qchange(xxx,yyy,zzz) >= threshold
424                     qnum(xxx,yyy,zzz)=n;
425                     yyy=yyy+1;  %move to +yyy direction
426                     if yyy>dim
427                         yyy=yyy-dim;
428                     end
429                 end
430                 qboundx(end+1)=xxx;
431                 if yyy==1
432                     qboundy(end+1)=dim;
433                 else
434                 qboundy(end+1)=yyy-1;
435                 end
436                 qboundz(end+1)=zzz;
437                 yyy=yy;
438
439                 while qchange(xxx,yyy,zzz) >= threshold
440                     qnum(xxx,yyy,zzz)=n;
441                     yyy=yyy-1;  %move to -yyy
442                     if yyy<1
443                         yyy=yyy+dim;
444                     end
445                 end
446                 qboundx(end+1)=xxx;
447                 if yyy==dim
448                     qboundy(end+1)=1;
449                 else
```

```
450                 qboundy(end+1)=yyy+1;
451                 end
452                 qboundz(end+1)=zzz;
453                 yyy=yy;
454                 %-----------end very little loop-----
455                 zz=zz-1;            %move to -zz direction
456                 if zz<1
457                     zz=zz+dim;
458                 end
459             end
460             qboundx(end+1)=xx;
461             qboundy(end+1)=yy;
462             if zz==dim
463                 qboundz(end+1)=1;
464             else
465             qboundz(end+1)=zz+1;
466             end
467             zz=zloc;
468             %--------------------end little loop------
469             xloc=xloc-1;
470             if xloc<1
471                 xloc=xloc+dim;
472             end
473         end
474         if xloc==dim;
475             qboundx(end+1)=1;
476         else
477         qboundx(end+1)=xloc+1;
478         end
479         qboundy(end+1)=yloc;
480         qboundz(end+1)=zloc;
```

```
481         xloc=x;                                                    ...
               %reset xloc to original x (where max q exists)
482         %------------------------------end large ...
               loop----------------
483
484         %--------------------large ...
               loop------------------------------
485
486         while qchange(xloc,yloc,zloc) >= threshold      ...
               %move to +y direction
487             qnum(xloc,yloc,zloc)=n;
488             %-----litte loop----------------------
489             while qchange(xx,yy,zz) >= threshold
490                 qnum(xx,yy,zz)=n;
491                 %----------very little loope-----
492                 while qchange(xxx,yyy,zzz) >= threshold
493                     qnum(xxx,yyy,zzz)=n;
494                     zzz=zzz+1;  %move to +zzz direction
495                     if zzz>dim
496                         zzz=zzz-dim;
497                     end
498                 end
499                 qboundx(end+1)=xxx;
500                 qboundy(end+1)=yyy;
501                 if zzz==1
502                     qboundz(end+1)=dim;
503                 else
504                 qboundz(end+1)=zzz-1;
505                 end
506                 zzz=zz;
507
```

```
508                 while qchange(xxx,yyy,zzz) >= threshold
509                     qnum(xxx,yyy,zzz)=n;
510                     zzz=zzz-1;  %move to -zzz
511                     if zzz<1
512                         zzz=zzz+dim;
513                     end
514                 end
515                 qboundx(end+1)=xxx;
516                 qboundy(end+1)=yyy;
517                 if zzz==dim
518                     qboundz(end+1)=1;
519                 else
520                 qboundz(end+1)=zzz+1;
521                 end
522                 zzz=zz;
523                 %-----------end very little loop-----
524                 xx=xx+1;           %move to +xx direction
525                 if xx>dim
526                     xx=xx-dim;
527                 end
528             end
529             if xx==1
530                 qboundx(end+1)=dim;
531             else
532             qboundx(end+1)=xx-1;
533             end
534             qboundy(end+1)=yy;
535             qboundz(end+1)=zz;
536             xx=xloc;
537
538             while qchange(xx,yy,zz) >= threshold
```

```
539                 qnum(xx,yy,zz)=n;
540                 %----------very little loope-----
541                 while qchange(xxx,yyy,zzz) >= threshold
542                     qnum(xxx,yyy,zzz)=n;
543                     zzz=zzz+1;  %move to +zzz direction
544                     if zzz>dim
545                         zzz=zzz-dim;
546                     end
547                 end
548                 qboundx(end+1)=xxx;
549                 qboundy(end+1)=yyy;
550                 if zzz==1
551                     qboundz(end+1)=dim;
552                 else
553                 qboundz(end+1)=zzz-1;
554                 end
555                 zzz=zz;
556
557                 while qchange(xxx,yyy,zzz) >= threshold
558                     qnum(xxx,yyy,zzz)=n;
559                     zzz=zzz-1;  %move to -zzz
560                     if zzz<1
561                         zzz=zzz+dim;
562                     end
563                 end
564                 qboundx(end+1)=xxx;
565                 qboundy(end+1)=yyy;
566                 if zzz==dim
567                     qboundz(end+1)=1;
568                 else
569                 qboundz(end+1)=zzz+1;
```

```
570                 end
571                 zzz=zz;
572                 %-----------end very little loop-----
573                 xx=xx-1;            %move to -xx direction
574                 if xx<1
575                     xx=xx+dim;
576                 end
577             end
578             if xx==dim
579                 qboundx(end+1)=1;
580             else
581             qboundx(end+1)=xx+1;
582             end
583             qboundy(end+1)=yy;
584             qboundz(end+1)=zz;
585             xx=xloc;
586
587             while qchange(xx,yy,zz) >= threshold
588                 qnum(xx,yy,zz)=n;
589                 %----------very little loope-----
590                 while qchange(xxx,yyy,zzz) >= threshold
591                     qnum(xxx,yyy,zzz)=n;
592                     xxx=xxx+1;  %move to +xxx direction
593                     if xxx>dim
594                         xxx=xxx-dim;
595                     end
596                 end
597                 if xxx==1
598                     qboundx(end+1)=dim;
599                 else
600                 qboundx(end+1)=xxx-1;
```

```
601                 end
602                 qboundy(end+1)=yyy;
603                 qboundz(end+1)=zzz;
604                 xxx=xx;
605
606                 while qchange(xxx,yyy,zzz) >= threshold
607                     qnum(xxx,yyy,zzz)=n;
608                     xxx=xxx-1;  %move to -xxx
609                     if xxx<1
610                         xxx=xxx+dim;
611                     end
612                 end
613                 if xxx==dim
614                     qboundx(end+1)=1;
615                 else
616                 qboundx(end+1)=xxx+1;
617                 end
618                 qboundy(end+1)=yyy;
619                 qboundz(end+1)=zzz;
620                 xxx=xx;
621                 %-----------end very little loop-----
622                 zz=zz+1;            %move to +zz direction
623                 if zz>dim
624                     zz=zz-dim;
625                 end
626             end
627             qboundx(end+1)=xx;
628             qboundy(end+1)=yy;
629             if zz==1
630                 qboundz(end+1)=dim;
631             else
```

```
632             qboundz(end+1)=zz-1;
633             end
634             zz=zloc;
635
636             while qchange(xx,yy,zz) >= threshold
637                 qnum(xx,yy,zz)=n;
638                 %----------very little loope-----
639                 while qchange(xxx,yyy,zzz) >= threshold
640                     qnum(xxx,yyy,zzz)=n;
641                     xxx=xxx+1;  %move to +xxx direction
642                     if xxx>dim
643                         xxx=xxx-dim;
644                     end
645                 end
646                 if xxx==1
647                     qboundx(end+1)=dim;
648                 else
649                 qboundx(end+1)=xxx-1;
650                 end
651                 qboundy(end+1)=yyy;
652                 qboundz(end+1)=zzz;
653                 xxx=xx;
654
655                 while qchange(xxx,yyy,zzz) >= threshold
656                     qnum(xxx,yyy,zzz)=n;
657                     xxx=xxx-1;  %move to -xxx
658                     if xxx<1
659                         xxx=xxx+dim;
660                     end
661                 end
662                 if xxx==dim
```

```
663                     qboundx(end+1)=1;
664                 else
665                 qboundx(end+1)=xxx+1;
666                 end
667                 qboundy(end+1)=yyy;
668                 qboundz(end+1)=zzz;
669                 xxx=xx;
670                 %-----------end very little loop-----
671                 zz=zz-1;            %move to -zz direction
672                 if zz<1
673                     zz=zz+dim;
674                 end
675             end
676             qboundx(end+1)=xx;
677             qboundy(end+1)=yy;
678             if zz==dim
679                 qboundz(end+1)=1;
680             else
681             qboundz(end+1)=zz+1;
682             end
683             zz=zloc;
684             %-------------------end little loop------
685             yloc=yloc+1;
686             if yloc>dim
687                 yloc=yloc-dim;
688             end
689         end
690         qboundx(end+1)=xloc;
691         if yloc==1
692             qboundy(end+1)=dim;
693         else
```

```
694         qboundy(end+1)=yloc-1;
695         end
696         qboundz(end+1)=zloc;
697         yloc=y;                                                 ...
                %reset yloc to original y (where max q exists)
698         %------------------------------end large ...
                loop----------------
699
700         %--------------------large ...
                loop------------------------------
701
702         while qchange(xloc,yloc,zloc) >= threshold      ...
                %move to -y direction
703              qnum(xloc,yloc,zloc)=n;
704              %-----litte loop----------------------
705              while qchange(xx,yy,zz) >= threshold
706                   qnum(xx,yy,zz)=n;
707                   %----------very little loope-----
708                   while qchange(xxx,yyy,zzz) >= threshold
709                        qnum(xxx,yyy,zzz)=n;
710                        zzz=zzz+1;  %move to +zzz direction
711                        if zzz>dim
712                             zzz=zzz-dim;
713                        end
714                   end
715                   qboundx(end+1)=xxx;
716                   qboundy(end+1)=yyy;
717                   if zzz==1
718                        qboundz(end+1)=dim;
719                   else
720                   qboundz(end+1)=zzz-1;
```

```
721                 end
722                 zzz=zz;
723
724                 while qchange(xxx,yyy,zzz) >= threshold
725                     qnum(xxx,yyy,zzz)=n;
726                     zzz=zzz-1;  %move to -zzz
727                     if zzz<1
728                         zzz=zzz+dim;
729                     end
730                 end
731                 qboundx(end+1)=xxx;
732                 qboundy(end+1)=yyy;
733                 if zzz==dim
734                     qboundz(end+1)=1;
735                 else
736                 qboundz(end+1)=zzz+1;
737                 end
738                 zzz=zz;
739                 %-----------end very little loop-----
740                 xx=xx+1;            %move to +xx direction
741                 if xx>dim
742                     xx=xx-dim;
743                 end
744             end
745             if xx==1
746                 qboundx(end+1)=dim;
747             else
748             qboundx(end+1)=xx-1;
749             end
750             qboundy(end+1)=yy;
751             qboundz(end+1)=zz;
```

```
752             xx=xloc;
753
754             while qchange(xx,yy,zz) >= threshold
755                 qnum(xx,yy,zz)=n;
756                 %----------very little loope-----
757                 while qchange(xxx,yyy,zzz) >= threshold
758                     qnum(xxx,yyy,zzz)=n;
759                     zzz=zzz+1;  %move to +zzz direction
760                     if zzz>dim
761                         zzz=zzz-dim;
762                     end
763                 end
764                 qboundx(end+1)=xxx;
765                 qboundy(end+1)=yyy;
766                 if zzz==1
767                     qboundz(end+1)=dim;
768                 else
769                 qboundz(end+1)=zzz-1;
770                 end
771                 zzz=zz;
772
773                 while qchange(xxx,yyy,zzz) >= threshold
774                     qnum(xxx,yyy,zzz)=n;
775                     zzz=zzz-1;  %move to -zzz
776                     if zzz<1
777                         zzz=zzz+dim;
778                     end
779                 end
780                 qboundx(end+1)=xxx;
781                 qboundy(end+1)=yyy;
782                 if zzz==dim
```

```
783                     qboundz(end+1)=1;
784                 else
785                 qboundz(end+1)=zzz+1;
786                 end
787                 zzz=zz;
788                 %-----------end very little loop-----
789                 xx=xx-1;           %move to -xx direction
790                 if xx<1
791                     xx=xx+dim;
792                 end
793             end
794             if xx==dim
795                 qboundx(end+1)=1;
796             else
797             qboundx(end+1)=xx+1;
798             end
799             qboundy(end+1)=yy;
800             qboundz(end+1)=zz;
801             xx=xloc;
802
803             while qchange(xx,yy,zz) >= threshold
804                 qnum(xx,yy,zz)=n;
805                 %----------very little loope-----
806                 while qchange(xxx,yyy,zzz) >= threshold
807                     qnum(xxx,yyy,zzz)=n;
808                     xxx=xxx+1;  %move to +xxx direction
809                     if xxx>dim
810                         xxx=xxx-dim;
811                     end
812                 end
813                 if xxx==1
```

```
814                     qboundx(end+1)=dim;
815                 else
816                 qboundx(end+1)=xxx-1;
817                 end
818                 qboundy(end+1)=yyy;
819                 qboundz(end+1)=zzz;
820                 xxx=xx;
821
822                 while qchange(xxx,yyy,zzz) >= threshold
823                     qnum(xxx,yyy,zzz)=n;
824                     xxx=xxx-1;  %move to -xxx
825                     if xxx<1
826                         xxx=xxx+dim;
827                     end
828                 end
829                 if xxx==dim
830                     qboundx(end+1)=1;
831                 else
832                 qboundx(end+1)=xxx+1;
833                 end
834                 qboundy(end+1)=yyy;
835                 qboundz(end+1)=zzz;
836                 xxx=xx;
837                 %-----------end very little loop-----
838                 zz=zz+1;           %move to +zz direction
839                 if zz>dim
840                     zz=zz-dim;
841                 end
842             end
843             qboundx(end+1)=xx;
844             qboundy(end+1)=yy;
```

```
845             if zz==1
846                 qboundz(end+1)=dim;
847             else
848             qboundz(end+1)=zz-1;
849             end
850             zz=zloc;
851
852             while qchange(xx,yy,zz) >= threshold
853                 qnum(xx,yy,zz)=n;
854                 %----------very little loope-----
855                 while qchange(xxx,yyy,zzz) >= threshold
856                     qnum(xxx,yyy,zzz)=n;
857                     xxx=xxx+1;  %move to +xxx direction
858                     if xxx>dim
859                         xxx=xxx-dim;
860                     end
861                 end
862                 if xxx==1
863                     qboundx(end+1)=dim;
864                 else
865                 qboundx(end+1)=xxx-1;
866                 end
867                 qboundy(end+1)=yyy;
868                 qboundz(end+1)=zzz;
869                 xxx=xx;
870
871                 while qchange(xxx,yyy,zzz) >= threshold
872                     qnum(xxx,yyy,zzz)=n;
873                     xxx=xxx-1;  %move to -xxx
874                     if xxx<1
875                         xxx=xxx+dim;
```

```
876                     end
877                 end
878                 if xxx==dim
879                     qboundx(end+1)=1;
880                 else
881                 qboundx(end+1)=xxx+1;
882                 end
883                 qboundy(end+1)=yyy;
884                 qboundz(end+1)=zzz;
885                 xxx=xx;
886                 %-----------end very little loop-----
887                 zz=zz-1;            %move to -zz direction
888                 if zz<1
889                     zz=zz+dim;
890                 end
891             end
892             qboundx(end+1)=xx;
893             qboundy(end+1)=yy;
894             if zz==dim
895                 qboundz(end+1)=1;
896             else
897             qboundz(end+1)=zz+1;
898             end
899             zz=zloc;
900             %--------------------end little loop------
901             yloc=yloc-1;
902             if yloc<1
903                 yloc=yloc+dim;
904             end
905         end
906         qboundx(end+1)=xloc;
```

```
907         if yloc==dim
908             qboundy(end+1)=1;
909         else
910         qboundy(end+1)=yloc+1;
911         end
912         qboundz(end+1)=zloc;
913         yloc=y;                                                    ...
               %reset yloc to original  (where max q exists)
914         %-----------------------------end large ...
               loop----------------
915
916         %--------------------large ...
               loop-----------------------------
917
918         while qchange(xloc,yloc,zloc) >= threshold     ...
               %move to +z direction
919             qnum(xloc,yloc,zloc)=n;
920             %-----litte loop----------------------
921             while qchange(xx,yy,zz) >= threshold
922                 qnum(xx,yy,zz)=n;
923                 %----------very little loope-----
924                 while qchange(xxx,yyy,zzz) >= threshold
925                     qnum(xxx,yyy,zzz)=n;
926                     yyy=yyy+1;  %move to +yyy direction
927                     if yyy>dim
928                         yyy=yyy-dim;
929                     end
930                 end
931                 qboundx(end+1)=xxx;
932                 if yyy==1
933                     qboundy(end+1)=dim;
```

```
934                 else
935                 qboundy(end+1)=yyy-1;
936                 end
937                 qboundz(end+1)=zzz;
938                 yyy=yy;
939
940                 while qchange(xxx,yyy,zzz) >= threshold
941                     qnum(xxx,yyy,zzz)=n;
942                     yyy=yyy-1;  %move to -yyy
943                     if yyy<1
944                         yyy=yyy+dim;
945                     end
946                 end
947                 qboundx(end+1)=xxx;
948                 if yyy==dim
949                     qboundy(end+1)=1;
950                 else
951                 qboundy(end+1)=yyy+1;
952                 end
953                 qboundz(end+1)=zzz;
954                 yyy=yy;
955                 %-----------end very little loop-----
956                 xx=xx+1;           %move to +xx direction
957                 if xx>dim
958                     xx=xx-dim;
959                 end
960             end
961             if xx==1
962                 qboundx(end+1)=dim;
963             else
964             qboundx(end+1)=xx-1;
```

```
965             end
966             qboundy(end+1)=yy;
967             qboundz(end+1)=zz;
968             xx=xloc;
969
970             while qchange(xx,yy,zz) >= threshold
971                 qnum(xx,yy,zz)=n;
972                 %----------very little loope-----
973                 while qchange(xxx,yyy,zzz) >= threshold
974                     qnum(xxx,yyy,zzz)=n;
975                     yyy=yyy+1;  %move to +yyy direction
976                     if yyy>dim
977                         yyy=yyy-dim;
978                     end
979                 end
980                 qboundx(end+1)=xxx;
981                 if yyy==1
982                     qboundy(end+1)=dim;
983                 else
984                 qboundy(end+1)=yyy-1;
985                 end
986                 qboundz(end+1)=zzz;
987                 yyy=yy;
988
989                 while qchange(xxx,yyy,zzz) >= threshold
990                     qnum(xxx,yyy,zzz)=n;
991                     yyy=yyy-1;  %move to -yyy
992                     if yyy<1
993                         yyy=yyy+dim;
994                     end
995                 end
```

```
996                 qboundx(end+1)=xxx;
997                 if yyy==dim;
998                     qboundy(end+1)=1;
999                 else
1000                qboundy(end+1)=yyy+1;
1001                end
1002                qboundz(end+1)=zzz;
1003                yyy=yy;
1004                %-----------end very little loop-----
1005                xx=xx-1;            %move to -xx direction
1006                if xx<1
1007                    xx=xx+dim;
1008                end
1009            end
1010            if xx==dim
1011                qboundx(end+1)=1;
1012            else
1013            qboundx(end+1)=xx+1;
1014            end
1015            qboundy(end+1)=yy;
1016            qboundz(end+1)=zz;
1017            xx=xloc;
1018
1019            while qchange(xx,yy,zz) >= threshold
1020                qnum(xx,yy,zz)=n;
1021                %----------very little loope-----
1022                while qchange(xxx,yyy,zzz) >= threshold
1023                    qnum(xxx,yyy,zzz)=n;
1024                    xxx=xxx+1;  %move to +xxx direction
1025                    if xxx>dim
1026                        xxx=xxx-dim;
```

```
1027                    end
1028                end
1029                if xxx==1
1030                    qboundx(end+1)=dim;
1031                else
1032                qboundx(end+1)=xxx-1;
1033                end
1034                qboundy(end+1)=yyy;
1035                qboundz(end+1)=zzz;
1036                xxx=xx;
1037
1038                while qchange(xxx,yyy,zzz) >= threshold
1039                    qnum(xxx,yyy,zzz)=n;
1040                    xxx=xxx-1;  %move to -xxx
1041                    if xxx<1
1042                        xxx=xxx+dim;
1043                    end
1044                end
1045                if xxx==dim
1046                    qboundx(end+1)=1;
1047                else
1048                qboundx(end+1)=xxx+1;
1049                end
1050                qboundy(end+1)=yyy;
1051                qboundz(end+1)=zzz;
1052                xxx=xx;
1053                %-----------end very little loop-----
1054                yy=yy+1;           %move to +yy direction
1055                if yy>dim
1056                    yy=yy-dim;
1057                end
```

```
1058            end
1059            qboundx(end+1)=xx;
1060            if yy==1
1061                 qboundy(end+1)=dim;
1062            else
1063            qboundy(end+1)=yy-1;
1064            end
1065            qboundz(end+1)=zz;
1066            yy=yloc;
1067
1068            while qchange(xx,yy,zz) >= threshold
1069                qnum(xx,yy,zz)=n;
1070                %----------very little loope-----
1071                while qchange(xxx,yyy,zzz) >= threshold
1072                    qnum(xxx,yyy,zzz)=n;
1073                    xxx=xxx+1;  %move to +xxx direction
1074                    if xxx>dim
1075                        xxx=xxx-dim;
1076                    end
1077                end
1078                if xxx==1
1079                    qboundx(end+1)=dim;
1080                else
1081                qboundx(end+1)=xxx-1;
1082                end
1083                qboundy(end+1)=yyy;
1084                qboundz(end+1)=zzz;
1085                xxx=xx;
1086
1087                while qchange(xxx,yyy,zzz) >= threshold
1088                    qnum(xxx,yyy,zzz)=n;
```

```
1089                    xxx=xxx-1;  %move to -xxx
1090                    if xxx<1
1091                        xxx=xxx+dim;
1092                    end
1093                end
1094                if xxx==dim
1095                    qboundx(end+1)=1;
1096                else
1097                qboundx(end+1)=xxx+1;
1098                end
1099                qboundy(end+1)=yyy;
1100                qboundz(end+1)=zzz;
1101                xxx=xx;
1102                %-----------end very little loop-----
1103                yy=yy-1;           %move to -yy direction
1104                if yy<1
1105                    yy=yy+dim;
1106                end
1107            end
1108            qboundx(end+1)=xx;
1109            if yy==dim
1110                qboundy(end+1)=1;
1111            else
1112            qboundy(end+1)=yy+1;
1113            end
1114            qboundz(end+1)=zz;
1115            yy=yloc;
1116            %--------------------end little loop------
1117            zloc=zloc+1;
1118            if zloc>dim
1119                zloc=zloc-dim;
```

```
1120            end
1121        end
1122        qboundx(end+1)=xloc;
1123        qboundy(end+1)=yloc;
1124        if zloc==1
1125            qboundz(end+1)=dim;
1126        else
1127        qboundz(end+1)=zloc-1;
1128        end
1129        zloc=z;                                                    ...
               %reset zloc to original z (where max q exists)
1130        %------------------------------end large ...
               loop----------------
1131
1132        %--------------------large ...
               loop------------------------------
1133
1134        while qchange(xloc,yloc,zloc) >= threshold      ...
               %move to -z direction
1135            qnum(xloc,yloc,zloc)=n;
1136            %-----litte loop----------------------
1137            while qchange(xx,yy,zz) >= threshold
1138                qnum(xx,yy,zz)=n;
1139                %----------very little loope-----
1140                while qchange(xxx,yyy,zzz) >= threshold
1141                    qnum(xxx,yyy,zzz)=n;
1142                    yyy=yyy+1;  %move to +yyy direction
1143                    if yyy>dim
1144                        yyy=yyy-dim;
1145                    end
1146                end
```

```
1147                qboundx(end+1)=xxx;
1148                if yyy==1
1149                    qboundy(end+1)=dim;
1150                else
1151                qboundy(end+1)=yyy-1;
1152                end
1153                qboundz(end+1)=zzz;
1154                yyy=yy;
1155
1156                while qchange(xxx,yyy,zzz) >= threshold
1157                    qnum(xxx,yyy,zzz)=n;
1158                    yyy=yyy-1;  %move to -yyy
1159                    if yyy<1
1160                        yyy=yyy+dim;
1161                    end
1162                end
1163                qboundx(end+1)=xxx;
1164                if yyy==dim
1165                    qboundy(end+1)=1;
1166                else
1167                qboundy(end+1)=yyy+1;
1168                end
1169                qboundz(end+1)=zzz;
1170                yyy=yy;
1171                %-----------end very little loop-----
1172                xx=xx+1;           %move to +xx direction
1173                if xx>dim
1174                    xx=xx-dim;
1175                end
1176            end
1177            if xx==1
```

```
1178                qboundx(end+1)=dim;
1179            else
1180            qboundx(end+1)=xx-1;
1181            end
1182            qboundy(end+1)=yy;
1183            qboundz(end+1)=zz;
1184            xx=xloc;
1185
1186            while qchange(xx,yy,zz) >= threshold
1187                qnum(xx,yy,zz)=n;
1188                %----------very little loope-----
1189                while qchange(xxx,yyy,zzz) >= threshold
1190                    qnum(xxx,yyy,zzz)=n;
1191                    yyy=yyy+1;  %move to +yyy direction
1192                    if yyy>dim
1193                        yyy=yyy-dim;
1194                    end
1195                end
1196                qboundx(end+1)=xxx;
1197                if yyy==1
1198                    qboundy(end+1)=dim;
1199                else
1200                qboundy(end+1)=yyy-1;
1201                end
1202                qboundz(end+1)=zzz;
1203                yyy=yy;
1204
1205                while qchange(xxx,yyy,zzz) >= threshold
1206                    qnum(xxx,yyy,zzz)=n;
1207                    yyy=yyy-1;  %move to -yyy
1208                    if yyy<1
```

```
1209                        yyy=yyy+dim;
1210                    end
1211                end
1212
1213                qboundx(end+1)=xxx;
1214                if yyy==dim
1215                    qboundy(end+1)=1;
1216                else
1217                qboundy(end+1)=yyy+1;
1218                end
1219                qboundz(end+1)=zzz;
1220                yyy=yy;
1221                %-----------end very little loop-----
1222                xx=xx-1;           %move to -xx direction
1223                if xx<1
1224                    xx=xx+dim;
1225                end
1226            end
1227            if xx==dim
1228                qboundx(end+1)=1;
1229            else
1230            qboundx(end+1)=xx+1;
1231            end
1232            qboundy(end+1)=yy;
1233            qboundz(end+1)=zz;
1234            xx=xloc;
1235
1236            while qchange(xx,yy,zz) >= threshold
1237                qnum(xx,yy,zz)=n;
1238                %----------very little loope-----
1239                while qchange(xxx,yyy,zzz) >= threshold
```

```
1240                     qnum(xxx,yyy,zzz)=n;
1241                     xxx=xxx+1;  %move to +xxx direction
1242                     if xxx>dim
1243                         xxx=xxx-dim;
1244                     end
1245                 end
1246                 if xxx==1
1247                     qboundx(end+1)=dim;
1248                 else
1249                 qboundx(end+1)=xxx-1;
1250                 end
1251                 qboundy(end+1)=yyy;
1252                 qboundz(end+1)=zzz;
1253                 xxx=xx;
1254
1255                 while qchange(xxx,yyy,zzz) >= threshold
1256                     qnum(xxx,yyy,zzz)=n;
1257                     xxx=xxx-1;  %move to -xxx
1258                     if xxx<1
1259                         xxx=xxx+dim;
1260                     end
1261                 end
1262                 if xxx==dim
1263                     qboundx(end+1)=1;
1264                 else
1265                 qboundx(end+1)=xxx+1;
1266                 end
1267                 qboundy(end+1)=yyy;
1268                 qboundz(end+1)=zzz;
1269                 xxx=xx;
1270                 %----------end very little loop-----
```

```
1271                yy=yy+1;            %move to +yy direction
1272                if yy>dim
1273                    yy=yy-dim;
1274                end
1275            end
1276            qboundx(end+1)=xx;
1277            if yy==1
1278                qboundy(end+1)=dim;
1279            else
1280            qboundy(end+1)=yy-1;
1281            end
1282            qboundz(end+1)=zz;
1283            yy=yloc;
1284
1285            while qchange(xx,yy,zz) >= threshold
1286                qnum(xx,yy,zz)=n;
1287                %----------very little loope-----
1288                while qchange(xxx,yyy,zzz) >= threshold
1289                    qnum(xxx,yyy,zzz)=n;
1290                    xxx=xxx+1;  %move to +xxx direction
1291                    if xxx>dim
1292                        xxx=xxx-dim;
1293                    end
1294                end
1295                if xxx==1
1296                    qboundx(end+1)=dim;
1297                else
1298                qboundx(end+1)=xxx-1;
1299                end
1300                qboundy(end+1)=yyy;
1301                qboundz(end+1)=zzz;
```

```
1302                xxx=xx;
1303
1304                while qchange(xxx,yyy,zzz) >= threshold
1305                    qnum(xxx,yyy,zzz)=n;
1306                    xxx=xxx-1;  %move to -xxx
1307                    if xxx<1
1308                        xxx=xxx+dim;
1309                    end
1310                end
1311                if xxx==dim
1312                    qboundx(end+1)=1;
1313                else
1314                qboundx(end+1)=xxx+1;
1315                end
1316                qboundy(end+1)=yyy;
1317                qboundz(end+1)=zzz;
1318                xxx=xx;
1319                %-----------end very little loop-----
1320                yy=yy-1;            %move to -yy direction
1321                if yy<1
1322                    yy=yy+dim;
1323                end
1324            end
1325            qboundx(end+1)=xx;
1326            if yy==dim
1327                qboundy(end+1)=1;
1328            else
1329            qboundy(end+1)=yy+1;
1330            end
1331            qboundz(end+1)=zz;
1332            yy=yloc;
```

```
1333            %--------------------end little loop------
1334            zloc=zloc-1;
1335            if zloc<1
1336                zloc=zloc+dim;
1337            end
1338        end
1339        qboundx(end+1)=xloc;
1340        qboundy(end+1)=yloc;
1341        if zloc==dim
1342            qboundz(end+1)=1;
1343        else
1344        qboundz(end+1)=zloc+1;
1345        end
1346        zloc=z;                                              ...
               %reset zloc to original z (where max q exists)
1347        %------------------------------end large ...
               loop----------------
1348
1349        %---------------till now, part of boundry points ...
               are found and recorded, and are marked as 'nth' ...
               in a new matrix ('nth' vortex)
1350    else
1351        thismaxpoint=false;
1352    end
1353    if thismaxpoint == true
1354        repeatboundlocs(1,1)=0;
1355        repeatboundlocs(:)=[];
1356    repeatboundlocs(1:length(qboundx),:) ...
1357     =[qboundx(:),qboundy(:),qboundz(:)];
1358    boundlocs(1,1)=0;
1359    boundlocs(:)=[];
```

```
1360      boundlocs=unique(repeatboundlocs,'rows');
1361      [row, colum]=size(boundlocs);
1362
1363      tempqboundx(1)=0;
1364      tempqboundx(:)=[];
1365      tempqboundy(1)=0;
1366      tempqboundy(:)=[];
1367      tempqboundz(1)=0;
1368      tempqboundz(:)=[];
1369      tempqboundx(1:length(boundlocs(:,1))) ...
1370          =boundlocs(:,1);
1371      tempqboundy(1:length(boundlocs(:,2))) ...
1372          =boundlocs(:,2);
1373      tempqboundz(1:length(boundlocs(:,3))) ...
1374          =boundlocs(:,3);
1375      tempvolume1=-1;
1376      tempvolume2=-2;
1377      %----------start with found boundary points---------------
1378      while tempvolume1~=tempvolume2
1379      repeatboundlocs(1:length(qboundx),:) ...
1380       =[qboundx(:),qboundy(:),qboundz(:)];
1381      boundlocs=unique(repeatboundlocs,'rows');
1382      tempqboundx(1:length(boundlocs(:,1)))=boundlocs(:,1);
1383      tempqboundy(1:length(boundlocs(:,2)))=boundlocs(:,2);
1384      tempqboundz(1:length(boundlocs(:,3)))=boundlocs(:,3);
1385      tempvolume1=tempvolume2;
1386      for boundnums=1:length(tempqboundx)
1387      xloc=tempqboundx(boundnums);
1388      yloc=tempqboundy(boundnums);
1389      zloc=tempqboundz(boundnums);
1390      if qchange(xloc,yloc,zloc)>= threshold
```

```
1391        %--------------------large ...
               loop------------------------------
1392        while qchange(xloc,yloc,zloc) >= threshold     ...
               %move to +x direction
1393            qnum(xloc,yloc,zloc)=n;              %marked ...
                   this point in "nth" vortex
1394            xx=xloc;                     %point ...
                   (xloc,yloc,zloc) is regarded as a temporary ...
                   origin
1395            yy=yloc;
1396            zz=zloc;
1397            %-----litte loop----------------------
1398            while qchange(xx,yy,zz) >= threshold
1399                qnum(xx,yy,zz)=n;
1400                xxx=xx;
1401                yyy=yy;
1402                zzz=zz;
1403                %----------very little loope-----
1404                while qchange(xxx,yyy,zzz) >= threshold
1405                    qnum(xxx,yyy,zzz)=n;
1406                    zzz=zzz+1;  %move to +zzz direction
1407                    if zzz>dim
1408                        zzz=zzz-dim;
1409                    end
1410                end
1411                qboundx(end+1)=xxx;
1412                qboundy(end+1)=yyy;
1413                if zzz==1
1414                    qboundz(end+1)=dim;
1415                else
1416
```

```
1417                qboundz(end+1)=zzz-1;
1418                end
1419                zzz=zz;
1420
1421                while qchange(xxx,yyy,zzz) >= threshold
1422                    qnum(xxx,yyy,zzz)=n;
1423                    zzz=zzz-1;  %move to -zzz
1424                    if zzz<1
1425                        zzz=zzz+dim;
1426                    end
1427                end
1428                qboundx(end+1)=xxx;
1429                qboundy(end+1)=yyy;
1430                if zzz==dim
1431                    qboundz(end+1)=1;
1432                else
1433                qboundz(end+1)=zzz+1;
1434                end
1435                zzz=zz;
1436                %-----------end very little loop-----
1437
1438                yy=yy+1;            %move to +yy direction
1439                if yy>dim
1440                    yy=yy-dim;
1441                end
1442            end
1443            qboundx(end+1)=xx;
1444            if yy==1
1445                qboundy(end+1)=dim;
1446            else
1447            qboundy(end+1)=yy-1;
```

```
1448            end
1449            qboundz(end+1)=zz;
1450            yy=yloc;
1451
1452            while qchange(xx,yy,zz) >= threshold
1453                qnum(xx,yy,zz)=n;
1454                %----------very little loope-----
1455                while qchange(xxx,yyy,zzz) >= threshold
1456                    qnum(xxx,yyy,zzz)=n;
1457                    zzz=zzz+1;  %move to +zzz direction
1458                    if zzz>dim
1459                        zzz=zzz-dim;
1460                    end
1461                end
1462                qboundx(end+1)=xxx;
1463                qboundy(end+1)=yyy;
1464                if zzz==1
1465                    qboundz(end+1)=dim;
1466                else
1467                qboundz(end+1)=zzz-1;
1468                end
1469                zzz=zz;
1470
1471                while qchange(xxx,yyy,zzz) >= threshold
1472                    qnum(xxx,yyy,zzz)=n;
1473                    zzz=zzz-1;  %move to -zzz
1474                    if zzz<1
1475                        zzz=zzz+dim;
1476                    end
1477                end
1478                qboundx(end+1)=xxx;
```

```
1479                qboundy(end+1)=yyy;
1480                if zzz==dim
1481                    qboundz(end+1)=1;
1482                else
1483                qboundz(end+1)=zzz+1;
1484                end
1485                zzz=zz;
1486                %-----------end very little loop-----
1487                yy=yy-1;           %move to -yy direction
1488                if yy<1
1489                    yy=yy+dim;
1490                end
1491            end
1492            qboundx(end+1)=xx;
1493            if yy==dim
1494                qboundy(end+1)=1;
1495            else
1496            qboundy(end+1)=yy+1;
1497            end
1498            qboundz(end+1)=zz;
1499            yy=yloc;
1500
1501            while qchange(xx,yy,zz) >= threshold
1502                qnum(xx,yy,zz)=n;
1503                %----------very little loope-----
1504                while qchange(xxx,yyy,zzz) >= threshold
1505                    qnum(xxx,yyy,zzz)=n;
1506                    yyy=yyy+1;  %move to +yyy direction
1507                    if yyy>dim
1508                        yyy=yyy-dim;
1509                    end
```

```
1510                end
1511                qboundx(end+1)=xxx;
1512                if yyy==1
1513                    qboundy(end+1)=dim;
1514                else
1515                qboundy(end+1)=yyy-1;
1516                end
1517                qboundz(end+1)=zzz;
1518                yyy=yy;
1519
1520                while qchange(xxx,yyy,zzz) >= threshold
1521                    qnum(xxx,yyy,zzz)=n;
1522                    yyy=yyy-1;  %move to -yyy
1523                    if yyy<1
1524                        yyy=yyy+dim;
1525                    end
1526                end
1527                qboundx(end+1)=xxx;
1528                if yyy==dim
1529                    qboundy(end+1)=1;
1530                else
1531                qboundy(end+1)=yyy+1;
1532                end
1533                qboundz(end+1)=zzz;
1534                yyy=yy;
1535                %-----------end very little loop-----
1536                zz=zz+1;           %move to +zz direction
1537                if zz>dim
1538                    zz=zz-dim;
1539                end
1540            end
```

```
1541            qboundx(end+1)=xx;
1542            qboundy(end+1)=yy;
1543            if zz==1
1544                qboundz(end+1)=dim;
1545            else
1546            qboundz(end+1)=zz-1;
1547            end
1548            zz=zloc;
1549
1550            while qchange(xx,yy,zz) >= threshold
1551                qnum(xx,yy,zz)=n;
1552                %----------very little loope-----
1553                while qchange(xxx,yyy,zzz) >= threshold
1554                    qnum(xxx,yyy,zzz)=n;
1555                    yyy=yyy+1;  %move to +yyy direction
1556                    if yyy>dim
1557                        yyy=yyy-dim;
1558                    end
1559                end
1560                qboundx(end+1)=xxx;
1561                if yyy==1
1562                    qboundy(end+1)=dim;
1563                else
1564                qboundy(end+1)=yyy-1;
1565                end
1566                qboundz(end+1)=zzz;
1567                yyy=yy;
1568
1569                while qchange(xxx,yyy,zzz) >= threshold
1570                    qnum(xxx,yyy,zzz)=n;
1571                    yyy=yyy-1;  %move to -yyy
```

```
                    if yyy<1
                        yyy=yyy+dim;
                    end
                end
                qboundx(end+1)=xxx;
                if yyy==dim
                    qboundy(end+1)=1;
                else
                qboundy(end+1)=yyy+1;
                end
                qboundz(end+1)=zzz;
                yyy=yy;
                %-----------end very little loop-----
                zz=zz-1;           %move to -zz direction
                if zz<1
                    zz=zz+dim;
                end
            end
            qboundx(end+1)=xx;
            qboundy(end+1)=yy;
            if zz==dim
                qboundz(end+1)=1;
            else
            qboundz(end+1)=zz+1;
            end
            zz=zloc;
            %--------------------end little loop------
            xloc=xloc+1;
            if xloc>dim
                xloc=xloc-dim;
            end
```

```
1603            end
1604            if xloc==1
1605                qboundx(end+1)=dim;
1606            else
1607            qboundx(end+1)=xloc-1;
1608            end
1609            qboundy(end+1)=yloc;
1610            qboundz(end+1)=zloc;
1611            xloc=tempqboundx(boundnums); ...
                                              %reset xloc to ...
                original x (boundry point)
1612            %------------------------------end large ...
                loop----------------
1613
1614            %--------------------large ...
                loop------------------------------
1615            while qchange(xloc,yloc,zloc) >= threshold     ...
                %move to -x direction
1616             qnum(xloc,yloc,zloc)=n;
1617             %-----litte loop----------------------
1618             while qchange(xx,yy,zz) >= threshold
1619                 qnum(xx,yy,zz)=n;
1620                 %----------very little loope-----
1621                 while qchange(xxx,yyy,zzz) >= threshold
1622                     qnum(xxx,yyy,zzz)=n;
1623                     zzz=zzz+1;  %move to +zzz direction
1624                     if zzz>dim
1625                         zzz=zzz-dim;
1626                     end
1627                 end
1628                 qboundx(end+1)=xxx;
```

```
1629                qboundy(end+1)=yyy;
1630                if zzz==1
1631                    qboundz(end+1)=dim;
1632                else
1633                qboundz(end+1)=zzz-1;
1634                end
1635                zzz=zz;
1636
1637                while qchange(xxx,yyy,zzz) >= threshold
1638                    qnum(xxx,yyy,zzz)=n;
1639                    zzz=zzz-1;  %move to -zzz
1640                    if zzz<1
1641                        zzz=zzz+dim;
1642                    end
1643                end
1644                qboundx(end+1)=xxx;
1645                qboundy(end+1)=yyy;
1646                if zzz==dim
1647                    qboundz(end+1)=1;
1648                else
1649                qboundz(end+1)=zzz+1;
1650                end
1651                zzz=zz;
1652                %-----------end very little loop-----
1653                yy=yy+1;           %move to +yy direction
1654                if yy>dim
1655                    yy=yy-dim;
1656                end
1657            end
1658            qboundx(end+1)=xx;
1659            if yy==1
```

```
1660                qboundy(end+1)=dim;
1661            else
1662            qboundy(end+1)=yy-1;
1663            end
1664            qboundz(end+1)=zz;
1665            yy=yloc;
1666
1667            while qchange(xx,yy,zz) >= threshold
1668                qnum(xx,yy,zz)=n;
1669                %----------very little loope-----
1670                while qchange(xxx,yyy,zzz) >= threshold
1671                    qnum(xxx,yyy,zzz)=n;
1672                    zzz=zzz+1;  %move to +zzz direction
1673                    if zzz>dim
1674                        zzz=zzz-dim;
1675                    end
1676                end
1677                qboundx(end+1)=xxx;
1678                qboundy(end+1)=yyy;
1679                if zzz==1
1680                    qboundz(end+1)=dim;
1681                else
1682                qboundz(end+1)=zzz-1;
1683                end
1684                zzz=zz;
1685
1686                while qchange(xxx,yyy,zzz) >= threshold
1687                    qnum(xxx,yyy,zzz)=n;
1688                    zzz=zzz-1;  %move to -zzz
1689                    if zzz<1
1690                        zzz=zzz+dim;
```

```
1691                    end
1692                end
1693                qboundx(end+1)=xxx;
1694                qboundy(end+1)=yyy;
1695                if zzz==dim
1696                    qboundz(end+1)=1;
1697                else
1698                qboundz(end+1)=zzz+1;
1699                end
1700                zzz=zz;
1701                %-----------end very little loop-----
1702                yy=yy-1;            %move to -yy direction
1703                if yy<1
1704                    yy=yy+dim;
1705                end
1706            end
1707            qboundx(end+1)=xx;
1708            if yy==dim
1709                qboundy(end+1)=1;
1710            else
1711            qboundy(end+1)=yy+1;
1712            end
1713            qboundz(end+1)=zz;
1714            yy=yloc;
1715
1716            while qchange(xx,yy,zz) >= threshold
1717                qnum(xx,yy,zz)=n;
1718                %----------very little loope-----
1719                while qchange(xxx,yyy,zzz) >= threshold
1720                    qnum(xxx,yyy,zzz)=n;
1721                    yyy=yyy+1;  %move to +yyy direction
```

```
1722                     if yyy>dim
1723                         yyy=yyy-dim;
1724                     end
1725                 end
1726                 qboundx(end+1)=xxx;
1727                 if yyy==1
1728                     qboundy(end+1)=dim;
1729                 else
1730                 qboundy(end+1)=yyy-1;
1731                 end
1732                 qboundz(end+1)=zzz;
1733                 yyy=yy;
1734
1735                 while qchange(xxx,yyy,zzz) >= threshold
1736                     qnum(xxx,yyy,zzz)=n;
1737                     yyy=yyy-1;  %move to -yyy
1738                     if yyy<1
1739                         yyy=yyy+dim;
1740                     end
1741                 end
1742                 qboundx(end+1)=xxx;
1743                 if yyy==dim
1744                     qboundy(end+1)=1;
1745                 else
1746                 qboundy(end+1)=yyy+1;
1747                 end
1748                 qboundz(end+1)=zzz;
1749                 yyy=yy;
1750                 %-----------end very little loop-----
1751                 zz=zz+1;           %move to +zz direction
1752                 if zz>dim
```

```
1753                    zz=zz-dim;
1754                end
1755            end
1756            qboundx(end+1)=xx;
1757            qboundy(end+1)=yy;
1758            if zz==1
1759                qboundz(end+1)=dim;
1760            else
1761            qboundz(end+1)=zz-1;
1762            end
1763            zz=zloc;
1764
1765            while qchange(xx,yy,zz) >= threshold
1766                qnum(xx,yy,zz)=n;
1767                %----------very little loope-----
1768                while qchange(xxx,yyy,zzz) >= threshold
1769                    qnum(xxx,yyy,zzz)=n;
1770                    yyy=yyy+1;  %move to +yyy direction
1771                    if yyy>dim
1772                        yyy=yyy-dim;
1773                    end
1774                end
1775                qboundx(end+1)=xxx;
1776                if yyy==1
1777                    qboundy(end+1)=dim;
1778                else
1779                qboundy(end+1)=yyy-1;
1780                end
1781                qboundz(end+1)=zzz;
1782                yyy=yy;
1783
```

```
1784                while qchange(xxx,yyy,zzz) >= threshold
1785                    qnum(xxx,yyy,zzz)=n;
1786                    yyy=yyy-1;  %move to -yyy
1787                    if yyy<1
1788                        yyy=yyy+dim;
1789                    end
1790                end
1791                qboundx(end+1)=xxx;
1792                if yyy==dim
1793                    qboundy(end+1)=1;
1794                else
1795                qboundy(end+1)=yyy+1;
1796                end
1797                qboundz(end+1)=zzz;
1798                yyy=yy;
1799                %-----------end very little loop-----
1800                zz=zz-1;           %move to -zz direction
1801                if zz<1
1802                    zz=zz+dim;
1803                end
1804            end
1805            qboundx(end+1)=xx;
1806            qboundy(end+1)=yy;
1807            if zz==dim
1808                qboundz(end+1)=1;
1809            else
1810            qboundz(end+1)=zz+1;
1811            end
1812            zz=zloc;
1813            %--------------------end little loop------
1814            xloc=xloc-1;
```

```
1815            if xloc<1
1816                xloc=xloc+dim;
1817            end
1818        end
1819        if xloc==dim
1820            qboundx(end+1)=1;
1821        else
1822        qboundx(end+1)=xloc+1;
1823        end
1824        qboundy(end+1)=yloc;
1825        qboundz(end+1)=zloc;
1826        xloc=tempqboundx(boundnums); ...
                                                    %reset ...
            xloc to original x (where boundry point)
1827        %------------------------------end large ...
            loop----------------
1828
1829        %--------------------large ...
            loop------------------------------
1830
1831        while qchange(xloc,yloc,zloc) >= threshold     ...
            %move to +y direction
1832            qnum(xloc,yloc,zloc)=n;
1833            %-----litte loop----------------------
1834            while qchange(xx,yy,zz) >= threshold
1835                qnum(xx,yy,zz)=n;
1836                %----------very little loope-----
1837                while qchange(xxx,yyy,zzz) >= threshold
1838                    qnum(xxx,yyy,zzz)=n;
1839                    zzz=zzz+1;  %move to +zzz direction
1840                    if zzz>dim
```

```
1841                         zzz=zzz-dim;
1842                     end
1843                 end
1844                 qboundx(end+1)=xxx;
1845                 qboundy(end+1)=yyy;
1846                 if zzz==1
1847                     qboundz(end+1)=dim;
1848                 else
1849                 qboundz(end+1)=zzz-1;
1850                 end
1851                 zzz=zz;
1852
1853                 while qchange(xxx,yyy,zzz) >= threshold
1854                     qnum(xxx,yyy,zzz)=n;
1855                     zzz=zzz-1;  %move to -zzz
1856                     if zzz<1
1857                         zzz=zzz+dim;
1858                     end
1859                 end
1860                 qboundx(end+1)=xxx;
1861                 qboundy(end+1)=yyy;
1862                 if zzz==dim
1863                     qboundz(end+1)=1;
1864                 else
1865                 qboundz(end+1)=zzz+1;
1866                 end
1867                 zzz=zz;
1868                 %-----------end very little loop-----
1869                 xx=xx+1;           %move to +xx direction
1870                 if xx>dim
1871                     xx=xx-dim;
```

```
1872                 end
1873             end
1874             if xx==1
1875                 qboundx(end+1)=dim;
1876             else
1877             qboundx(end+1)=xx-1;
1878             end
1879             qboundy(end+1)=yy;
1880             qboundz(end+1)=zz;
1881             xx=xloc;
1882
1883             while qchange(xx,yy,zz) >= threshold
1884                 qnum(xx,yy,zz)=n;
1885                 %----------very little loope-----
1886                 while qchange(xxx,yyy,zzz) >= threshold
1887                     qnum(xxx,yyy,zzz)=n;
1888                     zzz=zzz+1;  %move to +zzz direction
1889                     if zzz>dim
1890                         zzz=zzz-dim;
1891                     end
1892                 end
1893                 qboundx(end+1)=xxx;
1894                 qboundy(end+1)=yyy;
1895                 if zzz==1
1896                     qboundz(end+1)=dim;
1897                 else
1898                 qboundz(end+1)=zzz-1;
1899                 end
1900                 zzz=zz;
1901
1902                 while qchange(xxx,yyy,zzz) >= threshold
```

```
1903                    qnum(xxx,yyy,zzz)=n;
1904                    zzz=zzz-1;  %move to -zzz
1905                    if zzz<1
1906                        zzz=zzz+dim;
1907                    end
1908                end
1909                qboundx(end+1)=xxx;
1910                qboundy(end+1)=yyy;
1911               if zzz==dim
1912                   qboundz(end+1)=1;
1913               else
1914                qboundz(end+1)=zzz+1;
1915               end
1916                zzz=zz;
1917                %-----------end very little loop-----
1918                xx=xx-1;           %move to -xx direction
1919                if xx<1
1920                    xx=xx+dim;
1921                end
1922           end
1923           if xx==dim
1924               qboundx(end+1)=1;
1925           else
1926           qboundx(end+1)=xx+1;
1927           end
1928           qboundy(end+1)=yy;
1929           qboundz(end+1)=zz;
1930           xx=xloc;
1931
1932           while qchange(xx,yy,zz) >= threshold
1933               qnum(xx,yy,zz)=n;
```

```
1934                    %----------very little loope-----
1935                    while qchange(xxx,yyy,zzz) >= threshold
1936                        qnum(xxx,yyy,zzz)=n;
1937                        xxx=xxx+1;  %move to +xxx direction
1938                        if xxx>dim
1939                            xxx=xxx-dim;
1940                        end
1941                    end
1942                    if xxx==1
1943                        qboundx(end+1)=dim;
1944                    else
1945                    qboundx(end+1)=xxx-1;
1946                    end
1947                    qboundy(end+1)=yyy;
1948                    qboundz(end+1)=zzz;
1949                    xxx=xx;
1950
1951                    while qchange(xxx,yyy,zzz) >= threshold
1952                        qnum(xxx,yyy,zzz)=n;
1953                        xxx=xxx-1;  %move to -xxx
1954                        if xxx<1
1955                            xxx=xxx+dim;
1956                        end
1957                    end
1958                    if xxx==dim
1959                        qboundx(end+1)=1;
1960                    else
1961                    qboundx(end+1)=xxx+1;
1962                    end
1963                    qboundy(end+1)=yyy;
1964                    qboundz(end+1)=zzz;
```

```
1965                xxx=xx;
1966                %-----------end very little loop-----
1967                zz=zz+1;            %move to +zz direction
1968                if zz>dim
1969                    zz=zz-dim;
1970                end
1971            end
1972            qboundx(end+1)=xx;
1973            qboundy(end+1)=yy;
1974            if zz==1
1975                qboundz(end+1)=dim;
1976            else
1977            qboundz(end+1)=zz-1;
1978            end
1979            zz=zloc;
1980
1981            while qchange(xx,yy,zz) >= threshold
1982                qnum(xx,yy,zz)=n;
1983                %----------very little loope-----
1984                while qchange(xxx,yyy,zzz) >= threshold
1985                    qnum(xxx,yyy,zzz)=n;
1986                    xxx=xxx+1;  %move to +xxx direction
1987                    if xxx>dim
1988                        xxx=xxx-dim;
1989                    end
1990                end
1991                if xxx==1
1992                    qboundx(end+1)=dim;
1993                else
1994                qboundx(end+1)=xxx-1;
1995                end
```

```
1996                qboundy(end+1)=yyy;
1997                qboundz(end+1)=zzz;
1998                xxx=xx;
1999
2000                while qchange(xxx,yyy,zzz) >= threshold
2001                    qnum(xxx,yyy,zzz)=n;
2002                    xxx=xxx-1;  %move to -xxx
2003                    if xxx<1
2004                        xxx=xxx+dim;
2005                    end
2006                end
2007                if xxx==dim
2008                    qboundx(end+1)=1;
2009                else
2010                qboundx(end+1)=xxx+1;
2011                end
2012                qboundy(end+1)=yyy;
2013                qboundz(end+1)=zzz;
2014                xxx=xx;
2015                %-----------end very little loop-----
2016                zz=zz-1;           %move to -zz direction
2017                if zz<1
2018                    zz=zz+dim;
2019                end
2020            end
2021            qboundx(end+1)=xx;
2022            qboundy(end+1)=yy;
2023            if zz==dim
2024                qboundz(end+1)=1;
2025            else
2026            qboundz(end+1)=zz+1;
```

```
2027            end
2028            zz=zloc;
2029            %--------------------end little loop------
2030            yloc=yloc+1;
2031            if yloc>dim
2032                yloc=yloc-dim;
2033            end
2034        end
2035        qboundx(end+1)=xloc;
2036        if yloc==1
2037            qboundy(end+1)=dim;
2038        else
2039        qboundy(end+1)=yloc-1;
2040        end
2041        qboundz(end+1)=zloc;
2042        yloc=tempqboundy(boundnums); ...
                                                  %reset ...
            yloc to original y (where max q exists)
2043        %------------------------------end large ...
            loop----------------
2044
2045        %--------------------large ...
            loop------------------------------
2046
2047        while qchange(xloc,yloc,zloc) >= threshold     ...
            %move to -y direction
2048            qnum(xloc,yloc,zloc)=n;
2049            %-----litte loop----------------------
2050            while qchange(xx,yy,zz) >= threshold
2051                qnum(xx,yy,zz)=n;
2052                %----------very little loope-----
```

```
2053                 while qchange(xxx,yyy,zzz) >= threshold
2054                     qnum(xxx,yyy,zzz)=n;
2055                     zzz=zzz+1;  %move to +zzz direction
2056                     if zzz>dim
2057                         zzz=zzz-dim;
2058                     end
2059                 end
2060                 qboundx(end+1)=xxx;
2061                 qboundy(end+1)=yyy;
2062                 if zzz==1
2063                     qboundz(end+1)=dim;
2064                 else
2065                 qboundz(end+1)=zzz-1;
2066                 end
2067                 zzz=zz;
2068
2069                 while qchange(xxx,yyy,zzz) >= threshold
2070                     qnum(xxx,yyy,zzz)=n;
2071                     zzz=zzz-1;  %move to -zzz
2072                     if zzz<1
2073                         zzz=zzz+dim;
2074                     end
2075                 end
2076                 qboundx(end+1)=xxx;
2077                 qboundy(end+1)=yyy;
2078                 if zzz==dim
2079                     qboundz(end+1)=1;
2080                 else
2081                 qboundz(end+1)=zzz+1;
2082                 end
2083                 zzz=zz;
```

```
2084                %-----------end very little loop-----
2085                xx=xx+1;           %move to +xx direction
2086                if xx>dim
2087                    xx=xx-dim;
2088                end
2089            end
2090            if xx==1
2091                qboundx(end+1)=dim;
2092            else
2093            qboundx(end+1)=xx-1;
2094            end
2095            qboundy(end+1)=yy;
2096            qboundz(end+1)=zz;
2097            xx=xloc;
2098
2099            while qchange(xx,yy,zz) >= threshold
2100                qnum(xx,yy,zz)=n;
2101                %----------very little loope-----
2102                while qchange(xxx,yyy,zzz) >= threshold
2103                    qnum(xxx,yyy,zzz)=n;
2104                    zzz=zzz+1;  %move to +zzz direction
2105                    if zzz>dim
2106                        zzz=zzz-dim;
2107                    end
2108                end
2109                qboundx(end+1)=xxx;
2110                qboundy(end+1)=yyy;
2111                if zzz==1
2112                    qboundz(end+1)=dim;
2113                else
2114                qboundz(end+1)=zzz-1;
```

```
2115                end
2116                zzz=zz;
2117
2118                while qchange(xxx,yyy,zzz) >= threshold
2119                    qnum(xxx,yyy,zzz)=n;
2120                    zzz=zzz-1;  %move to -zzz
2121                    if zzz<1
2122                        zzz=zzz+dim;
2123                    end
2124                end
2125                qboundx(end+1)=xxx;
2126                qboundy(end+1)=yyy;
2127                if zzz==dim
2128                    qboundz(end+1)=1;
2129                else
2130                qboundz(end+1)=zzz+1;
2131                end
2132                zzz=zz;
2133                %-----------end very little loop-----
2134                xx=xx-1;           %move to -xx direction
2135                if xx<1
2136                    xx=xx+dim;
2137                end
2138            end
2139            if xx==dim
2140                qboundx(end+1)=1;
2141            else
2142            qboundx(end+1)=xx+1;
2143            end
2144            qboundy(end+1)=yy;
2145            qboundz(end+1)=zz;
```

```
2146            xx=xloc;
2147
2148            while qchange(xx,yy,zz) >= threshold
2149                qnum(xx,yy,zz)=n;
2150                %----------very little loope-----
2151                while qchange(xxx,yyy,zzz) >= threshold
2152                    qnum(xxx,yyy,zzz)=n;
2153                    xxx=xxx+1;  %move to +xxx direction
2154                    if xxx>dim
2155                        xxx=xxx-dim;
2156                    end
2157                end
2158                if xxx==1
2159                    qboundx(end+1)=dim;
2160                else
2161                qboundx(end+1)=xxx-1;
2162                end
2163                qboundy(end+1)=yyy;
2164                qboundz(end+1)=zzz;
2165                xxx=xx;
2166
2167                while qchange(xxx,yyy,zzz) >= threshold
2168                    qnum(xxx,yyy,zzz)=n;
2169                    xxx=xxx-1;  %move to -xxx
2170                    if xxx<1
2171                        xxx=xxx+dim;
2172                    end
2173                end
2174                if xxx==dim
2175                    qboundx(end+1)=1;
2176                else
```

```
2177                qboundx(end+1)=xxx+1;
2178                end
2179                qboundy(end+1)=yyy;
2180                qboundz(end+1)=zzz;
2181                xxx=xx;
2182                %-----------end very little loop-----
2183                zz=zz+1;           %move to +zz direction
2184                if zz>dim
2185                    zz=zz-dim;
2186                end
2187            end
2188            qboundx(end+1)=xx;
2189            qboundy(end+1)=yy;
2190            if zz==1
2191                qboundz(end+1)=dim;
2192            else
2193            qboundz(end+1)=zz-1;
2194            end
2195            zz=zloc;
2196
2197            while qchange(xx,yy,zz) >= threshold
2198                qnum(xx,yy,zz)=n;
2199                %----------very little loope-----
2200                while qchange(xxx,yyy,zzz) >= threshold
2201                    qnum(xxx,yyy,zzz)=n;
2202                    xxx=xxx+1;  %move to +xxx direction
2203                    if xxx>dim
2204                        xxx=xxx-dim;
2205                    end
2206                end
2207                if xxx==1
```

```
2208                      qboundx(end+1)=dim;
2209                  else
2210                  qboundx(end+1)=xxx-1;
2211                  end
2212                  qboundy(end+1)=yyy;
2213                  qboundz(end+1)=zzz;
2214                  xxx=xx;
2215
2216                  while qchange(xxx,yyy,zzz) >= threshold
2217                      qnum(xxx,yyy,zzz)=n;
2218                      xxx=xxx-1;  %move to -xxx
2219                      if xxx<1
2220                          xxx=xxx+dim;
2221                      end
2222                  end
2223                  if xxx==dim
2224                      qboundx(end+1)=1;
2225                  else
2226                  qboundx(end+1)=xxx+1;
2227                  end
2228                  qboundy(end+1)=yyy;
2229                  qboundz(end+1)=zzz;
2230                  xxx=xx;
2231                  %-----------end very little loop-----
2232                  zz=zz-1;           %move to -zz direction
2233                  if zz<1
2234                      zz=zz+dim;
2235                  end
2236              end
2237              qboundx(end+1)=xx;
2238              qboundy(end+1)=yy;
```

```
2239            if zz==dim
2240                qboundz(end+1)=1;
2241            else
2242            qboundz(end+1)=zz+1;
2243            end
2244            zz=zloc;
2245            %--------------------end little loop------
2246            yloc=yloc-1;
2247            if yloc<1
2248                yloc=yloc+dim;
2249            end
2250        end
2251        qboundx(end+1)=xloc;
2252        if yloc==dim
2253            qboundy(end+1)=1;
2254        else
2255        qboundy(end+1)=yloc+1;
2256        end
2257        qboundz(end+1)=zloc;
2258        yloc=tempqboundy(boundnums); ...
                                                  %reset ...
           yloc to original  (where max q exists)
2259        %------------------------------end large ...
           loop----------------
2260
2261        %--------------------large ...
           loop------------------------------
2262
2263        while qchange(xloc,yloc,zloc) >= threshold     ...
           %move to +z direction
2264            qnum(xloc,yloc,zloc)=n;
```

```
2265            %-----litte loop----------------------
2266            while qchange(xx,yy,zz) >= threshold
2267                qnum(xx,yy,zz)=n;
2268                %----------very little loope-----
2269                while qchange(xxx,yyy,zzz) >= threshold
2270                    qnum(xxx,yyy,zzz)=n;
2271                    yyy=yyy+1;  %move to +yyy direction
2272                    if yyy>dim
2273                        yyy=yyy-dim;
2274                    end
2275                end
2276                qboundx(end+1)=xxx;
2277                if yyy==1
2278                    qboundy(end+1)=dim;
2279                else
2280                qboundy(end+1)=yyy-1;
2281                end
2282                qboundz(end+1)=zzz;
2283                yyy=yy;
2284
2285                while qchange(xxx,yyy,zzz) >= threshold
2286                    qnum(xxx,yyy,zzz)=n;
2287                    yyy=yyy-1;  %move to -yyy
2288                    if yyy<1
2289                        yyy=yyy+dim;
2290                    end
2291                end
2292                qboundx(end+1)=xxx;
2293                if yyy==dim
2294                    qboundy(end+1)=1;
2295                else
```

```
2296                qboundy(end+1)=yyy+1;
2297                end
2298                qboundz(end+1)=zzz;
2299                yyy=yy;
2300                %-----------end very little loop-----
2301                xx=xx+1;           %move to +xx direction
2302                if xx>dim
2303                    xx=xx-dim;
2304                end
2305            end
2306            if xx==1
2307                qboundx(end+1)=dim;
2308            else
2309            qboundx(end+1)=xx-1;
2310            end
2311            qboundy(end+1)=yy;
2312            qboundz(end+1)=zz;
2313            xx=xloc;
2314
2315            while qchange(xx,yy,zz) >= threshold
2316                qnum(xx,yy,zz)=n;
2317                %----------very little loope-----
2318                while qchange(xxx,yyy,zzz) >= threshold
2319                    qnum(xxx,yyy,zzz)=n;
2320                    yyy=yyy+1;  %move to +yyy direction
2321                    if yyy>dim
2322                        yyy=yyy-dim;
2323                    end
2324                end
2325                qboundx(end+1)=xxx;
2326                if yyy==1
```

```
2327                    qboundy(end+1)=dim;
2328                else
2329                qboundy(end+1)=yyy-1;
2330                end
2331                qboundz(end+1)=zzz;
2332                yyy=yy;
2333
2334                while qchange(xxx,yyy,zzz) >= threshold
2335                    qnum(xxx,yyy,zzz)=n;
2336                    yyy=yyy-1;  %move to -yyy
2337                    if yyy<1
2338                        yyy=yyy+dim;
2339                    end
2340                end
2341                qboundx(end+1)=xxx;
2342                if yyy==dim
2343                    qboundy(end+1)=1;
2344                else
2345                qboundy(end+1)=yyy+1;
2346                end
2347                qboundz(end+1)=zzz;
2348                yyy=yy;
2349                %-----------end very little loop-----
2350                xx=xx-1;           %move to -xx direction
2351                if xx<1
2352                    xx=xx+dim;
2353                end
2354            end
2355            if xx==dim
2356                qboundx(end+1)=1;
2357            else
```

```
2358            qboundx(end+1)=xx+1;
2359            end
2360            qboundy(end+1)=yy;
2361            qboundz(end+1)=zz;
2362            xx=xloc;
2363
2364            while qchange(xx,yy,zz) >= threshold
2365                qnum(xx,yy,zz)=n;
2366                %----------very little loope-----
2367                while qchange(xxx,yyy,zzz) >= threshold
2368                    qnum(xxx,yyy,zzz)=n;
2369                    xxx=xxx+1;  %move to +xxx direction
2370                    if xxx>dim
2371                        xxx=xxx-dim;
2372                    end
2373                end
2374                if xxx==1
2375                    qboundx(end+1)=dim;
2376                else
2377                qboundx(end+1)=xxx-1;
2378                end
2379                qboundy(end+1)=yyy;
2380                qboundz(end+1)=zzz;
2381                xxx=xx;
2382
2383                while qchange(xxx,yyy,zzz) >= threshold
2384                    qnum(xxx,yyy,zzz)=n;
2385                    xxx=xxx-1;  %move to -xxx
2386                    if xxx<1
2387                        xxx=xxx+dim;
2388                    end
```

```
2389                end
2390                if xxx==dim
2391                    qboundx(end+1)=1;
2392                else
2393                qboundx(end+1)=xxx+1;
2394                end
2395                qboundy(end+1)=yyy;
2396                qboundz(end+1)=zzz;
2397                xxx=xx;
2398                %-----------end very little loop-----
2399                yy=yy+1;            %move to +yy direction
2400                if yy>dim
2401                    yy=yy-dim;
2402                end
2403            end
2404            qboundx(end+1)=xx;
2405            if yy==1
2406                qboundy(end+1)=dim;
2407            else
2408            qboundy(end+1)=yy-1;
2409            end
2410            qboundz(end+1)=zz;
2411            yy=yloc;
2412
2413            while qchange(xx,yy,zz) >= threshold
2414                qnum(xx,yy,zz)=n;
2415                %----------very little loope-----
2416                while qchange(xxx,yyy,zzz) >= threshold
2417                    qnum(xxx,yyy,zzz)=n;
2418                    xxx=xxx+1;  %move to +xxx direction
2419                    if xxx>dim
```

```
2420                         xxx=xxx-dim;
2421                     end
2422                 end
2423                 if xxx==1
2424                     qboundx(end+1)=dim;
2425                 else
2426                 qboundx(end+1)=xxx-1;
2427                 end
2428                 qboundy(end+1)=yyy;
2429                 qboundz(end+1)=zzz;
2430                 xxx=xx;
2431
2432                 while qchange(xxx,yyy,zzz) >= threshold
2433                     qnum(xxx,yyy,zzz)=n;
2434                     xxx=xxx-1;  %move to -xxx
2435                     if xxx<1
2436                         xxx=xxx+dim;
2437                     end
2438                 end
2439                 if xxx==dim
2440                     qboundx(end+1)=1;
2441                 else
2442                 qboundx(end+1)=xxx+1;
2443                 end
2444                 qboundy(end+1)=yyy;
2445                 qboundz(end+1)=zzz;
2446                 xxx=xx;
2447                 %-----------end very little loop-----
2448                 yy=yy-1;           %move to -yy direction
2449                 if yy<1
2450                     yy=yy+dim;
```

```
2451                end
2452            end
2453            qboundx(end+1)=xx;
2454            if yy==dim
2455                qboundy(end+1)=1;
2456            else
2457            qboundy(end+1)=yy+1;
2458            end
2459            qboundz(end+1)=zz;
2460            yy=yloc;
2461            %--------------------end little loop------
2462            zloc=zloc+1;
2463            if zloc>dim
2464                zloc=zloc-dim;
2465            end
2466        end
2467        qboundx(end+1)=xloc;
2468        qboundy(end+1)=yloc;
2469        if zloc==1
2470            qboundz(end+1)=1;
2471        else
2472        qboundz(end+1)=zloc-1;
2473        end
2474        zloc=tempqboundz(boundnums); ...
                                                  %reset ...
           zloc to original z (where max q exists)
2475        %------------------------------end large ...
           loop----------------
2476
2477        %--------------------large ...
           loop-----------------------------
```

```
2478
2479          while qchange(xloc,yloc,zloc) >= threshold     ...
                  %move to -z direction
2480              qnum(xloc,yloc,zloc)=n;
2481              %-----litte loop----------------------
2482              while qchange(xx,yy,zz) >= threshold
2483                  qnum(xx,yy,zz)=n;
2484                  %----------very little loope-----
2485                  while qchange(xxx,yyy,zzz) >= threshold
2486                      qnum(xxx,yyy,zzz)=n;
2487                      yyy=yyy+1;  %move to +yyy direction
2488                      if yyy>dim
2489                          yyy=yyy-dim;
2490                      end
2491                  end
2492                  qboundx(end+1)=xxx;
2493                  if yyy==1
2494                      qboundy(end+1)=dim;
2495                  else
2496                  qboundy(end+1)=yyy-1;
2497                  end
2498                  qboundz(end+1)=zzz;
2499                  yyy=yy;
2500
2501                  while qchange(xxx,yyy,zzz) >= threshold
2502                      qnum(xxx,yyy,zzz)=n;
2503                      yyy=yyy-1;  %move to -yyy
2504                      if yyy<1
2505                          yyy=yyy+dim;
2506                      end
2507                  end
```

```
2508                qboundx(end+1)=xxx;
2509                if yyy==dim
2510                    qboundy(end+1)=1;
2511                else
2512                qboundy(end+1)=yyy+1;
2513                end
2514                qboundz(end+1)=zzz;
2515                yyy=yy;
2516                %-----------end very little loop-----
2517                xx=xx+1;            %move to +xx direction
2518                if xx>dim
2519                    xx=xx-dim;
2520                end
2521            end
2522            if xx==1
2523                qboundx(end+1)=dim;
2524            else
2525            qboundx(end+1)=xx-1;
2526            end
2527            qboundy(end+1)=yy;
2528            qboundz(end+1)=zz;
2529            xx=xloc;
2530
2531            while qchange(xx,yy,zz) >= threshold
2532                qnum(xx,yy,zz)=n;
2533                %----------very little loope-----
2534                while qchange(xxx,yyy,zzz) >= threshold
2535                    qnum(xxx,yyy,zzz)=n;
2536                    yyy=yyy+1;  %move to +yyy direction
2537                    if yyy>dim
2538                        yyy=yyy-dim;
```

```
2539                    end
2540                end
2541                qboundx(end+1)=xxx;
2542                if yyy==1
2543                    qboundy(end+1)=dim;
2544                else
2545                qboundy(end+1)=yyy-1;
2546                end
2547                qboundz(end+1)=zzz;
2548                yyy=yy;
2549
2550                while qchange(xxx,yyy,zzz) >= threshold
2551                    qnum(xxx,yyy,zzz)=n;
2552                    yyy=yyy-1;  %move to -yyy
2553                    if yyy<1
2554                        yyy=yyy+dim;
2555                    end
2556                end
2557                qboundx(end+1)=xxx;
2558                if yyy==dim
2559                    qboundy(end+1)=1;
2560                else
2561                qboundy(end+1)=yyy+1;
2562                end
2563                qboundz(end+1)=zzz;
2564                yyy=yy;
2565                %-----------end very little loop-----
2566                xx=xx-1;           %move to -xx direction
2567                if xx<1
2568                    xx=xx+dim;
2569                end
```

```
2570            end
2571            if xx==dim
2572                qboundx(end+1)=1;
2573            else
2574            qboundx(end+1)=xx+1;
2575            end
2576            qboundy(end+1)=yy;
2577            qboundz(end+1)=zz;
2578            xx=xloc;
2579
2580            while qchange(xx,yy,zz) >= threshold
2581                qnum(xx,yy,zz)=n;
2582                %----------very little loope-----
2583                while qchange(xxx,yyy,zzz) >= threshold
2584                    qnum(xxx,yyy,zzz)=n;
2585                    xxx=xxx+1;  %move to +xxx direction
2586                    if xxx>dim
2587                        xxx=xxx-dim;
2588                    end
2589                end
2590                if xxx==1
2591                    qboundx(end+1)=dim;
2592                else
2593                qboundx(end+1)=xxx-1;
2594                end
2595                qboundy(end+1)=yyy;
2596                qboundz(end+1)=zzz;
2597                xxx=xx;
2598
2599                while qchange(xxx,yyy,zzz) >= threshold
2600                    qnum(xxx,yyy,zzz)=n;
```

```
2601                    xxx=xxx-1;  %move to -xxx
2602                    if xxx<1
2603                        xxx=xxx+dim;
2604                    end
2605                end
2606                if xxx==dim
2607                    qboundx(end+1)=1;
2608                else
2609                qboundx(end+1)=xxx+1;
2610                end
2611                qboundy(end+1)=yyy;
2612                qboundz(end+1)=zzz;
2613                xxx=xx;
2614                %-----------end very little loop-----
2615                yy=yy+1;           %move to +yy direction
2616                if yy>dim
2617                    yy=yy-dim;
2618                end
2619            end
2620            qboundx(end+1)=xx;
2621            if yy==1
2622                qboundy(end+1)=dim;
2623            else
2624            qboundy(end+1)=yy-1;
2625            end
2626            qboundz(end+1)=zz;
2627            yy=yloc;
2628
2629            while qchange(xx,yy,zz) >= threshold
2630                qnum(xx,yy,zz)=n;
2631                %----------very little loope-----
```

```
2632                    while qchange(xxx,yyy,zzz) >= threshold
2633                        qnum(xxx,yyy,zzz)=n;
2634                        xxx=xxx+1;  %move to +xxx direction
2635                        if xxx>dim
2636                            xxx=xxx-dim;
2637                        end
2638                    end
2639                    if xxx==1
2640                        qboundx(end+1)=dim;
2641                    else
2642
2643                    qboundx(end+1)=xxx-1;
2644                    end
2645                    qboundy(end+1)=yyy;
2646                    qboundz(end+1)=zzz;
2647                    xxx=xx;
2648
2649                    while qchange(xxx,yyy,zzz) >= threshold
2650                        qnum(xxx,yyy,zzz)=n;
2651                        xxx=xxx-1;  %move to -xxx
2652                        if xxx<1
2653                            xxx=xxx+dim;
2654                        end
2655                    end
2656                    if xxx==dim
2657                        qboundx(end+1)=1;
2658                    else
2659
2660                    qboundx(end+1)=xxx+1;
2661                    end
2662                    qboundy(end+1)=yyy;
```

```
2663                qboundz(end+1)=zzz;
2664                xxx=xx;
2665                %-----------end very little loop-----
2666                yy=yy-1;            %move to -yy direction
2667                if yy<1
2668                    yy=yy+dim;
2669                end
2670            end
2671            qboundx(end+1)=xx;
2672            if yy==dim
2673                qboundy(end+1)=1;
2674            else
2675            qboundy(end+1)=yy+1;
2676            end
2677            qboundz(end+1)=zz;
2678            yy=yloc;
2679            %--------------------end little loop------
2680            zloc=zloc-1;
2681            if zloc<1
2682                zloc=zloc+dim;
2683            end
2684        end
2685        qboundx(end+1)=xloc;
2686        qboundy(end+1)=yloc;
2687        if zloc==dim
2688            qboundz(end+1)=1;
2689        else
2690        qboundz(end+1)=zloc+1;
2691        end
2692        zloc=tempqboundz(boundnums); ...
                                                    %reset ...
```

```
            zloc to original z (where max q exists)
2693        %------------------------------end large ...
            loop----------------

2694
2695    end
2696    end
2697    tempvolume2=sum(qnum(:)==n);
2698    end
2699
2700    if tempvolume2 >= cutoffvolume
2701       disp(['The volume of ',num2str(n),' th vortex is ...
              ',num2str(tempvolume2)])
2702    [x1,y1,z1]=ind2sub(size(qnum),find(qnum==n));
2703        for positions=1:length(x1)
2704        qfinal(x1(positions),y1(positions),z1(positions)) ...
2705            =q(x1(positions),y1(positions),z1(positions));
2706        qchange(x1(positions),y1(positions),z1(positions))=0;
2707        end
2708    else
2709        [x1,y1,z1]=ind2sub(size(qnum),find(qnum==n));
2710        for positions=1:length(x1)
2711        qnum(x1(positions),y1(positions),z1(positions))=0;
2712        qchange(x1(positions),y1(positions),z1(positions))=0;
2713        end
2714        n=n-1;
2715    end
2716    n=n+1;
2717    end
2718    qboundx(:)=[];
2719    qboundy(:)=[];
2720    qboundz(:)=[];
```

```
2721      end
2722    qmax=max(max(max(qchange)));
2723  end
2724  saveqnum=reshape(qnum,[dim^3,1]);
2725
2726  fid = fopen([filepath2,'qnub',num2str(t,'%04i'),'.dat'], 'w');
2727      fprintf(fid, '%8.1f\n', saveqnum);
2728      fclose(fid);
2729  hintnumber=((t-aa1)/interv+1)/((aa2-aa1)/interv+1);
2730  hints=[num2str(hintnumber*100),'% is finished'];
2731  waitbar(hintnumber,hh,hints)
2732  % filename=[filepath2,'qnum',num2str(t,'%04i'),'.dat'];
2733  % save(filename,'saveqnum','-ascii')
2734  end
2735  toc
2736  close(hh)
```