Title
Exploring the Limits of Dropwise Condensation on Nano-structured Surfaces

Author
Mendoza, Hector

Publication Date
2013

Peer reviewed|Thesis/dissertation

Exploring the Limits of Dropwise Condensation on Nano-structured Surfaces

by

Hector Mendoza

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Van P. Carey, Chair
Professor Costas Grigoropoulos
Professor Per F. Peterson

Spring 2013

Exploring the Limits of Dropwise Condensation on Nano-structured Surfaces

Abstract

Exploring the Limits of Dropwise Condensation on Nano-structured Surfaces

by

Hector Mendoza

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Van P. Carey, Chair

Within the types of condensation that can form on a surface, dropwise condensation has been previously shown to produce condensation heat transfer coefficients up to an order of magnitude greater than film condensation. Among dropwise condensation investigations, it has also been shown that smaller droplets result in higher heat transfer coefficients. An area that is currently under investigation within condensation advancements is creating superhydrophobic surfaces that can sustain smaller droplets during condensation. However, as droplet diameters are reduced to sizes comparable to the flow's mean free path, various mechanisms are expected to affect transport as the flow transitions from a continuum to free molecular flow: non-continuum transport effects, curvature effects on surface tension and on saturation conditions, and interactions with nearby droplets.

In this dissertation, we investigate the limits of heat transfer performance on surfaces that strive to sustain dropwise condensation for smaller droplets. We explore and compare the limitations of dropwise condensation as mean droplet sizes are reduced to micro and nanoscales using three different models: one that uses an approximation for micro and nanoscale transport on an array of droplets, one that uses the DSMC method to simulate transport on a single droplet, and a third model that uses the DSMC method to simulate transport on an array of droplets.

We found the three different models to show similar trends; dropwise condensation heat transfer coefficients increased as droplet sizes were reduced, but only up to a certain point where non-continuum transport and curvature effects became significant. For pure steam condensing on a cold wall at standard atmospheric condition with 3 °C of subcooling, dropwise condensation heat transfer coefficients were found to peak when droplets approached diameters near 200 nm. The effects of varying contact angle, thermal accommodation, pressure, amount of subcooling, spacing between droplets, and introduction of noncondensible particles into the system are also explored and discussed in detail.

Para mi familia tan linda que siempre me ha apoyado en todo...

Contents

List of Figures

List of Tables

Acknowledgments

I've had the privilege to have been influenced by a wide group of people while at UC Berkeley. I could not be any more thankful to all of those who have impacted my path in one way or another.

I would like to give a special thanks and my respects to Professor Van P. Carey for all his guidance, support, advice, patience, and encouragement over the years in his lab as my research advisor and dissertation chair. I especially want to thank Professor Per Peterson and Professor Costas Grigoropoulos for agreeing to review my dissertation on such a short notice. I would also like to thank Professor Carlos Fernandez-Pello for serving as my qualifying exam committee chair and being my advisor during the first couple of years at Berkeley.

Professor Ronald Bagley and Professor Hai-Chao Han highly influenced my decision to come to UC Berkeley, and for that I am very grateful. Their influence has literally enriched my life to come and meet the great people that I have at such a wonderful place.

I would also like to thank Beatriz Lopez Flores and Dr. Vanessa M. Rivera for their support and encouragement during my first years at Berkeley. They served as a tremendous source of encouragement as well as providing valuable moral support. Much of the financial assistance I received throughout my time at UC Berkeley was through their assistance.

Many thanks to those funders and sponsors who helped during my PhD research: The Alfred P. Sloan Foundation Fellowship, The Ralph A. Seban Heat Transfer Fellowship.

Thanks to everyone in the EIT lab for their help and camaraderie: Dave Lettieri, Sara Beaini, Maritza Ruiz, Vince Romanin.

An extra special thank you to Natalie Torres and Imelda Mendoza for editing sections of this dissertation.

I could not be at the point I am in my life without having crossed paths at UC Berkeley with a wonderful group of friends whom I would undoubtedly call my Berkeley family. In no particular order, I am very grateful to: Rene Sanchez, Ann E Nisenson (AKA Annie), Maria Gutierrez, Reynaldo Guerra, Esther Zeledon, Jorge Padilla, Rosailda Perez, Debora Ramirez, Kenneth Armijo, Keno Urquiza, Jigar Adhvaryu, William Casper Ortiz, Jennifer McDonald, Marilola Perez, Gustavo Buenrostro, Pedro el Piro Reynoso Mora, Erick Ulin Avila, Nienke Schouten, Jose Gines Garcia Cerdan, Tiernan Doyle, Daniel Shu, Bret Strogen, Naim Darghouth, Monica Kapil, Nancy Diaz, Karla Vega, Sonia Fereres, Amanda Dodd, Chris Hanneman, Adrien Monvoisin. Equal gratitude goes to those whove influenced me in my undergraduate years and continue to be a special part of my life: Ernesto Padilla,

Jorge Trevio, Jaciel Solis, Allan Beyer, Luis Elizondo, Mike Frazier, Cesar Gonzalez, Miguel Salinas, Aaron Campbell, Joe and Brandi Marroquin, Jared Brown, Luis Zavala, Rolando Jimenez, Jason Hernandez, Raul Reyes. Thanks to everyone from the Saturday Soccer group for all the good times and stress-relieving games.

Completing my Ph.D studies would not have been possible without the constant support from my family. Many, many thanks to them for everything they have done to help me achieve such a milestone. I am very fortunate to have had the guidance and unconditional love of my parents, Eutiquio and Aracely Mendoza. The determination, perseverance, and abundance of values I learned from them are indescribable. Of course, nothing would be the same without Imelda, Ricardo, Francisco, and Armando Mendoza as part of my life. Last but not least, I appreciate all the support from my extended family and anyone not previously mentioned that has influenced my life.

Chapter 1

Introduction and Motivation

1.1 Water in the 21st Century

As the global population continues to rise, demand for natural resources follows similar trends. According to The United Nations Department of Economic and Social Affairs (UN-DESA), water consumption has been growing at more than twice the rate of population growth in the last century [1]. Much of our demand for water on a global scale is not directly seen, but without fresh water availability, several sectors of an economy can suffer, not to mention the people dependent on those sectors. We not only depend on water for direct consumption, but fresh water plays a vital role in numerous indirect applications in our day to day lives. From supplying agriculture for the harvesting of food that we eat, to playing essential roles in a plethora of industrial applications that we depend on such as fabricating, processing, washing, diluting, sanitation, cooling, and energy generation, water is essential to all of our lives [2].

While there is no global water scarcity, around 1.6 billion people (almost one fourth of the world's population) face economic water shortages, and an increasing number of regions are chronically in shortage of water. To understand the severity that exists but that may sometimes be remote from the developing world, UNDESA [1] provides the following facts:

- Around 700 million people in 43 countries suffer today from water scarcity.
- By 2025, 1.8 billion people will be living in countries or regions with absolute water scarcity, and two-thirds of the world's population could be living under water stressed conditions.
- With the existing potential changes in climate, almost half the world's population could be living in areas of high water stress by 2030, including between 75 million and 250

million people in Africa. In addition, water scarcity in some arid and semi-arid places will displace between 24 million and 700 million people.

- Sub-Saharan Africa has the largest number of water-stressed countries of any region.

Paired with the advent of renewable and sustainable technologies in a wide range of engineering applications, it is not difficult to see that minimizing our depletion of the available natural resources is not only becoming an increasing trend, but a necessary one. Innovating renewable and sustainable engineering technologies is not limited to the popular areas of manufacturing and energy generation/storage, but it can and should also include sustainable ways of meeting our fresh water demands.

1.1.1 Water and Energy

The current, daily world-average consumption of fresh water is approximately at 1000 gallons per capita [3]. This consumption, coupled with continuously increasing water-demands, entails that finding ways to minimize the energy needed to meet this demand can prove to have an impact on a large scale. Depending on the geographical location, the demand for fresh water, along with the energy intensity required to obtain available fresh water, varies highly from region to region. Using the United States as a specific example, the energy intensity varies widely between the South Atlantic states to the Mountain states. Specifically, the energy intensity to attain fresh water can be from 350 kWh per year in the South Atlantic states to over 750 kWh per year in the Mountain states due to water accessibility, according to the U.S. Department of Energy [2]. Furthermore, according to the United States Geological Survey (USGS) in 2005, about 410 billion gallons of total water are withdrawn per day for the country as a whole. Of those 410 billion gallons, at least 85% of total water withdrawals are fresh water withdrawals - - directly indicating our high dependence on fresh water needs. Noting, however, that 97.5% of the world's water is salt water, it is apparent that making use of this source can result to be beneficial if the energy intensity to convert it to potable water, or at least useable water, can be minimized [4]. Finding ways to tap into this abundant source by desalinating brackish and seawater can prove to be promising for areas that are stressed by lack of water as well as areas that are not, so long as the methods used come from sustainable, low-energy intensive technologies.

1.2 Water Desalination

While various methods of water desalination currently exist, two main methods are through distillation (thermal) or by passing the salt water through a membrane processes [5].

Some of the main types of membrane processes are:

- Electrodialysis reversal (EDR)
- Reverse osmosis (RO)
- Membrane distillation (MD)

While some of the main types of distillation methods are:

- Multi-stage flash distillation (MSF)
- Multiple-effect distillation(MEDIME)
- Vapor-compression distillation (VCD)

The premise behind the membrane processes is that, by pushing the salt water through a permeable membrane, the salt can effectively be trapped. Aside from desalinating, these membranes also remove other contaminants from a water supply, and they can further serve to trap certain microbes and pathogens. In addition to potable water applications, depending on the type of membrane used, membrane technologies are also used for industrial, wastewater recovery, and refuse applications. While having the same applications, what distinguishes one type of membrane technology from the other is in how the solution passes through the membrane, whether it is by a driving pressure difference, differences in concentration, or through use of an electric field. Due to the filtering nature of membrane technologies, all membrane types of systems therefore require chemical cleaning and maintenance of the membrane [6].

Distillation methods, on the other hand, do not use a membrane but rather rely on thermal energy to separate the water from the salt. These methods have existed for a much longer time, and all are based on evaporating water and collecting the condensate. The differences between the various distillation processes have to do with how the salt water is heated and where it is condensed. These processes can take advantage of the use of a vacuum type of distillation, where water is boiled at lower pressures to lower the boiling temperature. Techniques of this type effectively lower the energy required to boil the water.

The desalinating capacities of the two main types of desalination vary depending on the specific technology being analyzed. Membrane types of systems are becoming more readily available as further research is conducted in manufacturing the membranes and as these types of systems become more scalable. Even though individual desalination plants using membrane technologies, mainly reverse osmosis, have smaller individual capacities than those using distillation methods, mainly multi-stage flash distillation, the larger number number of existing plants using reverse osmosis yield a higher overall capacity. More specifically,

according to the International Desalination Association (IDA) and the Global Water Intelligence(GWI), of the current capacity of 78.5 million m^3/day (19.8 billion U.S. gallons), approximately 60% is from reverse osmosis technologies and about 34% is from thermal desalination [5].

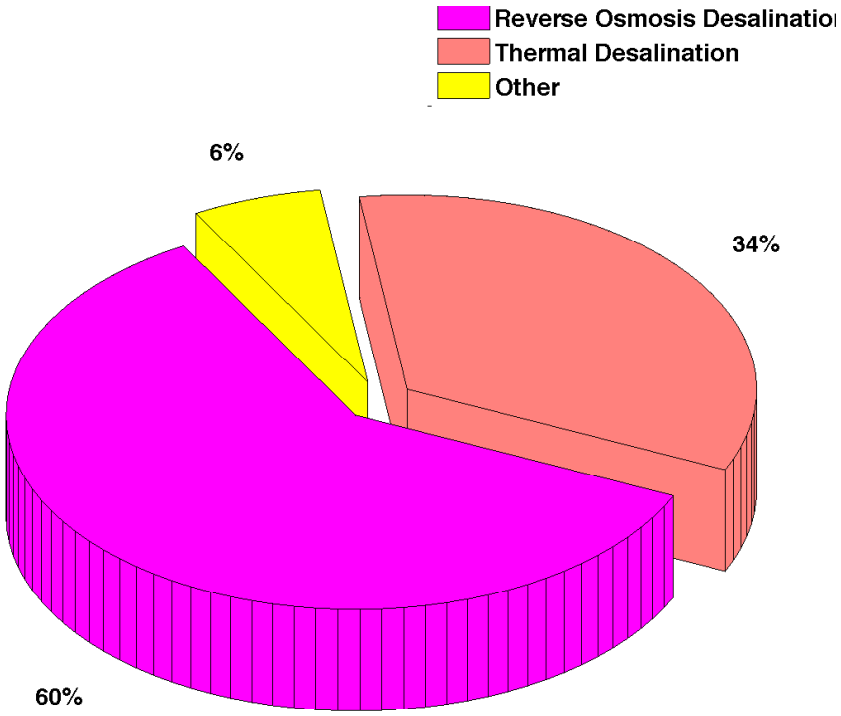


Figure 1.1: Distributions of Major Desalination Types in 2012 [5]

Nonetheless, the higher desalination capacity of thermal desalination processes still renders them very attractive. For example, according to the International Desalination Association and the Global Water Intelligence, the largest membrane desalination plant in the world is currently the 444,000 m^3/day Victoria Desalination Plant in Melbourne Australia, but the largest thermal desalination plant in the world is the 880,000 m^3/day plant in Saudi Arabia. The largest desalination plant in the world expected to start running in 2014 is the 1,025,000 m^3/day Ras Al Khair project in Saudi Arabia, which uses both membrane and thermal technologies. Both technologies have advantages and disadvantages, and, ultimately, a combination of these technologies can assist us in meeting our fresh water demands sustainably, especially considering that the past five years have seen a 57% increase in the capacity of desalination plants on line [5].

1.2.1 Solar Desalination

The most popular form of using solar energy to harvest fresh water from salt water is through solar humidification-dehumidification [6]. This method functions on a similar premise as multi-stage flash distillation, which, at the most basic level, is creating water vapor from salt water and then condensing the resulting vapor. The difference is that solar humidification-dehumidification uses incident sunlight as the heating force to produce the water vapor that is later condensed on a separate chamber. The type of solar-heating system can be as sophisticated as using solar collectors to concentrate the heat needed to vaporize the salt water, or it can be as simple as capitalizing on environments with relatively high humidity. The limitation that prevents these technologies from exploding on the market is low scalability [6].

Both solar and thermal technologies inspire an interest for improvement. Due to the high scalability of thermal desalination technologies, ultimately reducing the costs of the process can have profound implications. For solar desalination technologies, being able to collect the water in efficient forms and increasing scalability can also prove to have positive impacts.

1.2.2 The Condensation Phase of Distillation

Whether through traditional thermal distillation or via solar desalination techniques, a crucial step of the process, independent of how the vapor was generated, is to be able to collect the evaporated water through a condenser. Collecting the water entails exposing the vapor to a cool surface that is at a temperature below the saturation temperature of the vapor at its partial pressure. For any substance in a saturated state, the amount of energy required to condense the vapor per unit mass is its latent heat of condensation Δh. In a system where a vapor is condensing on a cold wall, the amount of potential cooling q(in terms of $\frac{\text{energy}}{\text{unit time}}$) can be as expressed as equation (1.1)

$$q = h_c A_s (\Delta T), \tag{1.1}$$

where h_c is the condensation heat transfer coefficient, A_s is the surface area of the cold wall, and ΔT is the temperature difference between the cold wall and the bulk vapor ($T_{sat} - T_{wall}$). The amount of vapor that can be condensed per unit time, $\dot{m}$, then becomes

$$\dot{m} = \frac{q}{\Delta h}. \tag{1.2}$$

Through inspection of equations (1.1) and (1.2), it can be seen that the amount of vapor that can be condensed is a direct function of h_c and ΔT. This implies that higher rates of condensation can be obtained by increasing either h_c, ΔT, or a combination of both. For a saturated vapor, increasing ΔT means reducing the wall temperature to colder levels,

but this could require an undesired additional energy input to cool the wall. However, if ΔT is desired to be kept at a minimum, this implies that finding ways to increase h_c can prove to be beneficial in any condensation process. Finding ways to maximize h_c means that higher condensation rates can be obtained for a specified condensation wall temperature, or that less energy would be required (as opposed to a situation with lower h_c) to cool the condenser wall for a specified desired condensation rate. In effect, maximizing h_c enhances the condensation processes.

1.2.3 Relevance of Motivation to Proposed Research and Mission Statement

The condensation heat transfer coefficient h_c can be a function of multiple phenomena, with one of the principal variables being the condensing surface. Therefore, investigating the behavior of h_c under different conditions becomes the focus of this dissertation. The above sections lay the framework for the motivation behind the research presented in this dissertation, but the implications of the analyses are not limited to those applications. In fact, the research presented here can serve as guidance in any application where efficient forms of condensation are desired. Due to the motivations presented above, the research presented in this dissertation aims to theoretically explore materials that can promote enhanced condensation. Ultimately, materials that can enhance condensation will result to have higher heat transfer coefficients, thus h_c is used as a principal metric for the condensation processes of the research presented here.

1.2.4 External Condensation

External condensation can be divided into two main types: Film condensation and dropwise Condensation. Multiple factors can influence the type of condensation that will form, including the shape of the condensing surface, the surface tension of the vapor near the surface, and the surface-energy of the condensing surface. Film condensation is more likely to occur with surfaces of high surface-energy which result in a well-wetted surface. Various theories exist as to how dropwise condensation forms varying on whether it is a derivative of an initial film formation bursting, or whether droplet formation starts occurring on active nucleation sites of a cold surface.

What is certain, however, is that dropwise condensation is more likely to be induced by having a poorly wetted surface. According to Carey [7], the poorly wetted surface can be achieved for steam condensation by:

1.) Temporarily injecting a non-wetting chemical into the vapor which subsequently deposits

on the surface,

2.) Temporarily introducing a substance such as a fatty acid or wax onto the solid surface, or

3.) Permanently coating the surface with a low-surface-energy polymer.

The distinguishing point to make is that, for comparable conditions, dropwise condensation can prove to be a more efficient form of condensation, up to an order of magnitude [7]. Therefore, if we are looking to enhance condensation, investigating and enhancing dropwise condensation becomes of greater interest. For research purposes, the work presented in this dissertation does not focus on the initial formation of droplets, but it is rather an analysis after nucleation has occurred (post-nucleation analysis).

1.3 Methodology

When looking to enhance dropwise condensation, the mechanisms at play must be well understood. Experimental studies of dropwise condensation have generally indicated that higher heat transfer coefficients correspond to smaller mean sizes of droplets growing through condensation on the surface [7]. Furthermore, recent investigations of dropwise condensation on nano-structured surfaces suggest that optimizing the design of such surfaces can push mean droplet sizes down to smaller values and significantly enhance heat transfer [8, 9, 10, 11, 12, 13, 14]. The physical reasoning that further supports these studies at macroscopic droplet sizes (diameters greater than 0.5 mm) comes purely from the conduction resistance [15]. Specifically, as the droplet size decreases, the conduction resistance from the interface of the droplet to the cold solid surface under the droplet diminishes. This reduction in conduction resistance clearly leads to the higher heat transfer coefficients, and hence could lead to potentially better designs for enhancing dropwise condensation heat transfer. However, as droplet sizes become ultra-small, other physical mechanisms that affect droplet condensation growth (which are not normally prominent at larger sizes), and hence compete with the effect of the reduced conduction resistance, begin to come into play.

To generate a model capable of performing a comprehensive analysis, accounting for physical mechanisms that amplify at reduced diameters is essential. The research presented here stresses to incorporate the following:

1.) Account for non-continuum transport effects at smaller diameters

2.) Account for changes in surface tension for reduced diameters

3.) Account for interface curvature on the droplet vapor pressure

1.3.1 The Non-Continuum Transport Effects

The importance of non-continuum rarefied gas effects is generally indicated by the Knudsen number Kn_d, defined here as the mean free path in the gas λ_m divided by the droplet diameter d:

$$\mathrm{Kn}_d = \lambda_m / d. \tag{1.3}$$

What occurs as the Knudsen number increases for a specific flow is that the length scale, in our case d, becomes small enough to the point where it starts approximating the magnitude of the mean free path of the flow. At these length scales, the flow can no longer be modeled as a continuum fluid. That is, the continuum assumption of fluid mechanics, where properties such as density, pressure, temperature, and velocity are well-defined and are assumed to vary continuously from one point to another, is no longer a good approximation and the fact that the fluid is made up of discrete molecules can no longer be ignored. Therefore, in situations where length scales become comparable to the mean free path, the flow has to be modeled using alternate methods involving statistical mechanics.

Figure 1.2 captures the different types of flow regimes for increasing Knudsen numbers. Flows deviating from a pure continuum model starts to occur around Knudsen numbers of about .01, where the continuum assumption of no-slip at a fluid-solid boundary is no longer negligible. Microflows typically fall in this regime, and non-continuum transport effects are expected to become increasingly important as Kn_d increases above 0.05 [16]. The transition from a pure continuum to free molecular flow happens for flows characterized by Knudsen numbers between 0.01 to 10, and it becomes challenging to accurately model these flows.

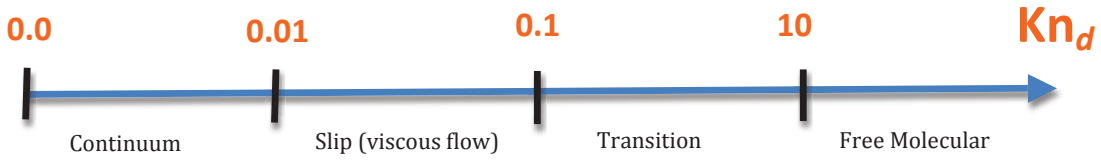


Figure 1.2: Flow regimes for increasing Knudsen number [16]

For saturated pure water vapor at atmospheric pressure and 70% of atmospheric pressure, the mean free path is about 70 nm and 100 nm, respectively. It follows that for condensation of saturated steam at atmospheric pressure and at 70% of atmospheric pressure, droplet sizes

of 0.5 μm correspond to Knudsen numbers between about 0.01 and 0.2. For some applications of this type, it is clear that Kn_d will be in a range for which non-continuum transport effects will be important. As reduced diameters tend to increase Kn_d for atmospheric conditions held constant, this may be true in the small droplet size limit. Furthermore, for sub-atmospheric condensation, which can be of interest in some applications, the Knudsen number may be even higher than 0.2 as the mean free path becomes larger.

1.3.2 Effects of Curvature on Surface Tension

Aside from the non-continuum transport effects present at smaller scales, the reduced droplet sizes also induce changes in surface tension and on the saturation conditions. According to Carey [7], the flat interface surface tension is valid when the thickness of the interfacial region of a liquid is very small compared to the radius of curvature of the surface defining the outer boundary of the interfacial region. As the interfacial region is proven to be on the order of a few nanometers, the interface of interest, in our case the droplet, must have a very small radius of curvature to deviate from the flat-interface theory. Molecular theories of capillarity and thermodynamic analyses, discussed by various references [7] and [17], predict that the surface tension will vary with the radius of curvature when the radius of curvature becomes comparable to the thickness of its interfacial region. The effects of curvature on surface tension are described by the correction to the flat interface surface tension by equation (1.4) [17]

$$\sigma_{lv} = \sigma_\infty(T)\left[1 + \frac{4\delta_T}{d}\right]^{-1}, \tag{1.4}$$

where $\sigma_\infty(T)$ is the flat interface surface tension at a given temperature T, and δ_T is the Tolman length. For water, the Tolman length is recommended to be 0.157 nm [18]. In inspecting equation (1.4), it can become obvious that the correction factor, $[1+\frac{4\delta_T}{d}]^{-1}$, becomes smaller as droplet sizes are reduced. For water, Pruppacher et al. [19] recommend using a Tolman length of 0.157 nm. Using that value, Figure 1.3 shows how the droplet surface tension starts to slowly decrease around radii of 150 μm, but greatly reduces below radii of 60 μm.

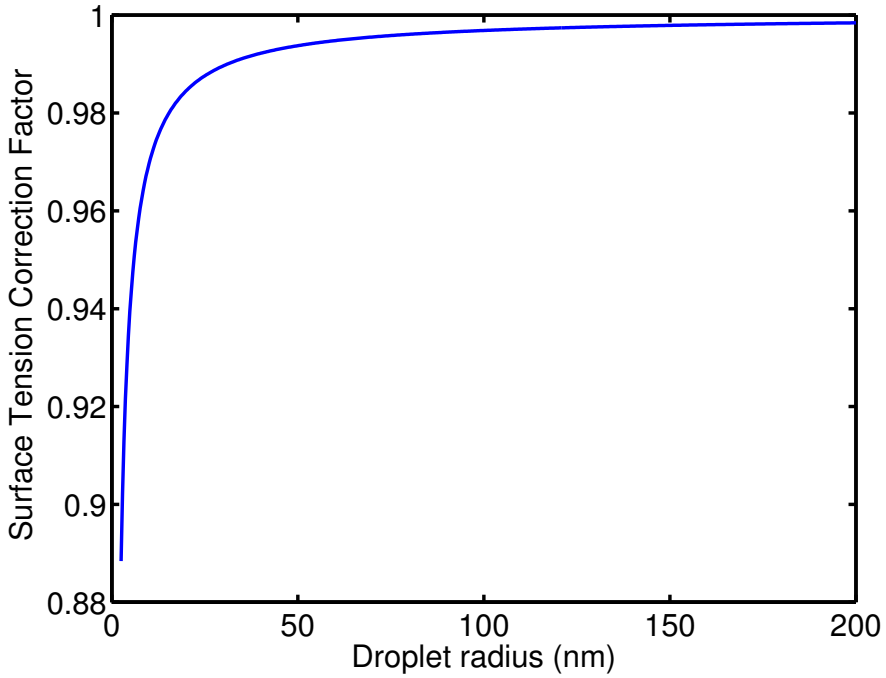


Figure 1.3: Effects of droplet curvature on surface tension

.

1.3.3 Effects of Curvature on Saturation Conditions

For a liquid droplet in equilibrium with its vapor, thermodynamic analyses on liquid droplets in a supersaturated vapor reveal the dependence of the equilibrium vapor pressure P_v on the surface tension and droplet size. The equilibrium vapor pressure for a liquid droplet of radius r_d in a supersaturated vapor can be represented as equation (1.5)

$$P_v = P_{sat}(T)\exp\left(\frac{2v_l\sigma_{lv}}{r_d RT}\right), \tag{1.5}$$

where $P_{sat}(T)$ is the flat-interface saturation pressure at a given temperature T, v_l is the specific volume at that temperature, and R is the gas constant [20] (a derivation and discussion of the assumptions made for the droplet vapor pressure expressed in this form are discussed in detail in Chapter 2). If all other parameters besides σ_{lv} and r_d are held constant, which is true for a given temperature, the effects of reduced droplet radius on the equilibrium vapor pressure can be visualized from Figure 1.4. The sample figure shown is specifically for pure steam with properties evaluated at atmospheric conditions. Two points to note from the trend are that, 1) curvature effects start to show around radii of 400 nm, and 2) these curvature effects can significantly increase below 200 nm radii. The figure is for visualization

purposes only, and it should be noted that different substances under different conditions will behave in a similar way, but to different magnitudes.

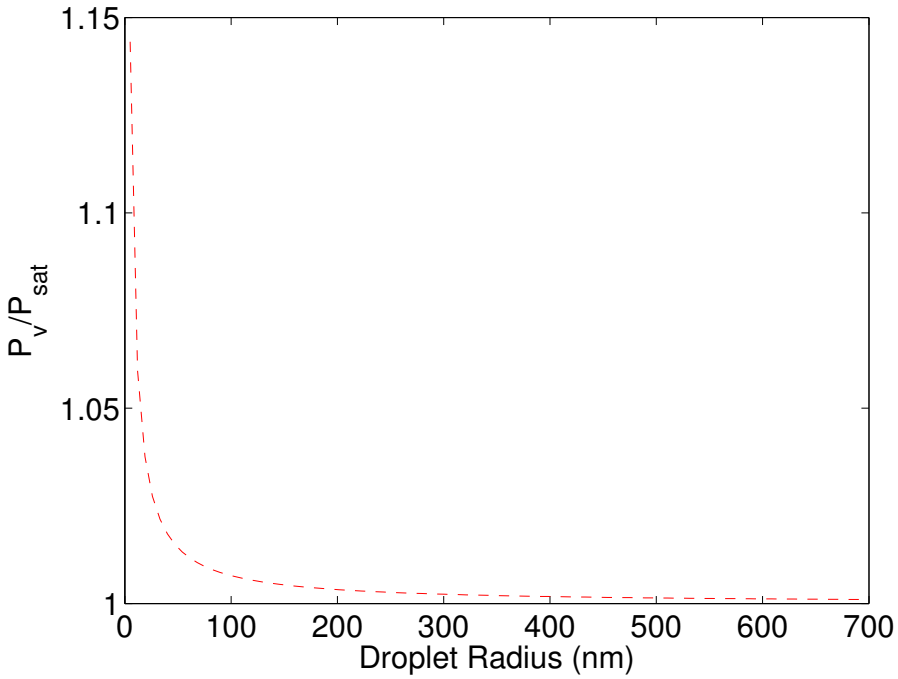


Figure 1.4: Effects of droplet curvature on equilibrium vapor pressure

.

The aforementioned mechanisms affecting dropwise condensation transport as droplet sizes are reduced, which are the factors of interest to explore in this research, can be visually appreciated by by Figure 1.5.

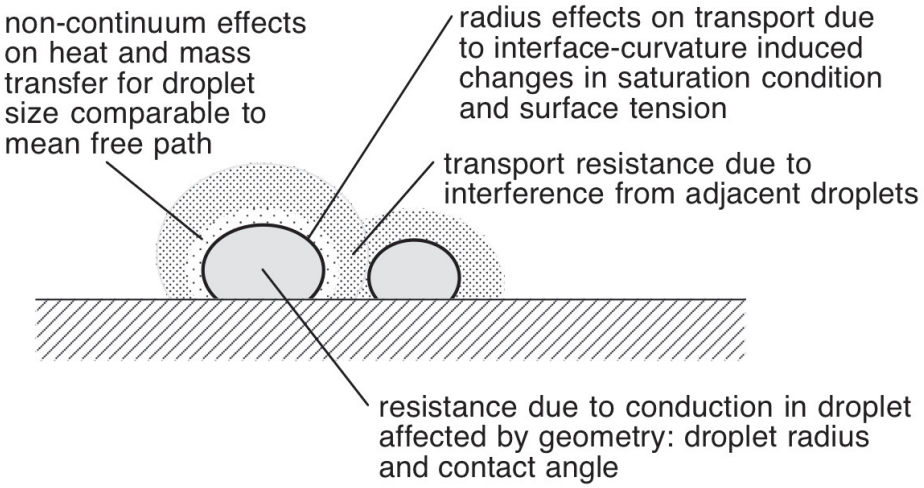


Figure 1.5: Mechanisms affecting dropwise condensation

These mechanisms and trends discussed above indicate that, for pure water vapor, at macroscopic mean droplet diameters ($> 500\mu$m), the dropwise condensation heat transfer coefficient will tend to increase as mean droplet diameter decreases, and at smaller diameters ($< 5\mu$m) the added mechanisms described above will become important. These added mechanisms occurring at the smaller scales could diminish the effectiveness of the heat transfer, and the overall effect on the condensing heat transfer coefficient becomes of question. *Does the heat transfer coefficient continue to increase as mean droplet sizes decrease in spite of these added mechanisms, or does the coefficient peak at a particular size and then begin to diminish*? This question is important both because its answer is central to understanding the mechanism of dropwise condensation, and because the answer will help guide the development of nano-structured enhanced dropwise condensation surfaces for applications involving condensation of water. It therefore becomes paramount to explore the limits of heat transfer enhancement that can be achieved by pushing mean droplet sizes to progressively smaller sizes.

The above description not only outlines potential competing mechanisms at smaller scales of dropwise condensation, but it also presents challenges that exist to adequately model dropwise condensation in a way that incorporates and considers the various mechanisms at play. As mentioned above, the transition from a pure continuum to free molecular flow happens for flows characterized by Knudsen numbers between 0.01 to 10, and it thus becomes challenging to accurately model these flows. To explore the various mechanisms at play as mean droplet sizes are reduced considering the transition regime, three different models that attempt to capture the combined effects were developed and are presented in subsequent chapters of this dissertation. Each of the models developed attempted to build upon and improve on the previous model/s. These models consist of:

1.) A preliminary model that uses an approximate technique to account for the non-continuum transport effects on a cluster of droplets undergoing dropwise condensation, developed using MATLAB;

2.) A more robust model on a single droplet that uses Direct Simulation Monte Carlo techniques to model the transport to a water droplet immersed in pure steam, developed in .C language;

3.) A second Direct Simulation Monte Carlo (DSMC) model was developed, on a droplet cluster rather than a single droplet, which attempts to account for interference effects of nearby droplets using the more robust methods. This model is further extended to incorporate and explore the effects of condensation in the presence of a non-condensible gas.

1.3.4 Evolution of the Different Models

Model 1: The Approximation Model

The first model lays the foundation and basic framework of this research. All the previously mentioned mechanisms of concern were considered, but with limitations. The model was limited in the accuracy it had in accounting for the non-continuum transport effects. That is to say, the predicted flux of molecules to the droplets at reduced diameters was based on an approximation method using the Knudsen number describing the system. This approximation is discussed with detail in further chapters, but the premise was that, as molecular flow to the droplets transitioned from a pure continuum to completely ballistic, transport to the droplets was divided into two sources: one was a flux of molecules coming from the ambient and described by kinetic theory, and the other was a flux from the ambient described to be as purely ballistic with no intermolecular collisions. As the droplet diameters decreased thereby increasing the Knudsen number defining the system conditions, the flux of molecules incident on droplets described by kinetic theory decreased while the flux described to be purely ballistic progressively increased with increasing Knudsen number. This approximation of the first model inspired the development of the more robust second and third models using the Direct Monte Carlo Simulation method. These models developed using DSMC were expected to provide validation to the approximations made in the first model.

Models 2 and 3: The Single Droplet DSMC and Droplet Cluster DSMC models

The DSMC method was chosen as the preferred method to accurately attempt to model the condensation flows for smaller diameters simply because these methods have a respectable

reputation for providing accurate results for a wide range of Knudsen numbers. The DSMC method was first used by Bird in the 1960s and has since been used in a wide range of applications to model rarefied gas flows [21]. The approach the DSMC method takes is that flows can be modeled using simulation particles representing a large group of molecules in a probabilistic fashion. These particles are moved throughout the simulation in a realistic manner through a physical space with real physical boundaries. Inter-particle collisions along with particle-boundary collisions are calculated using probabilistic models. Since DSMC techniques execute collisions only in between time-steps, the limitations are that the simulations have to iterate through time-steps that are smaller than the mean collision time. The advantage of the DSMC method is that the bundling of large groups of molecules into particles reduces the necessary computing power. DSMC techniques can therefore become reasonable to manage flows characterized by Knudsen numbers near one, where other methods, such as molecular dynamics models, would otherwise become unfeasible due to the required large amount of computing power [22, 23, 24].

1.3.5 Additional Factors Considered

Condensation in the Presence of Air

In many of the applications of interest, condensation will not occur in pure systems of water/steam mixtures, but rather in the presence of air. The introduction of non-condensible molecules into the systems studied presents changing dynamics of the condensation process. Recalling that condensation of vapor molecules occurs when a vapor is cooled to a temperature below its saturation temperature at its partial pressure in a mixture helps understand the effect of non-condensible molecules. As non-condensible molecules are introduced into a mixture, the partial pressure of the vapor is reduced, resulting in a lower saturation temperature corresponding to that partial pressure. Furthermore, as a consequence of the condensation process at the interface, where only the vapor is condensed, the concentration of the non-condensable gas at the interface can be higher than it is in the bulk mixture at the ambient. This results in a decreased partial pressure of the vapor at the interface below its ambient value. What this entails when the system is not in equilibrium is that the corresponding saturation temperature at the interface is even lower than it is for the bulk fluid. This described scenario can lead to a relatively high concentration of the non-condensible molecules at the interface whenever the non-condensible gas is of low concentration in the bulk fluid. Consequently, the resulting depression of the saturation temperature at the interface potentially reduces the condensation heat transfer rate below what would result if the system were a pure vapor under the same conditions. It therefore becomes of interest to also explore the introduction of a non-condensible substance into the systems being analyzed to visualize and understand the combined effects.

Contact Angle

Depending on the techniques used to develop superhydrophobic surfaces that attempt to sustain dropwise condensation at reduced diameters, a different wettability can result on these surfaces, leading to droplets condensing at varying contact angles. Because controlling wettability is recognized as key in promoting dropwise condensation, variations of different materials are theoretically explored through the incorporation of varying contact angles for the droplets in our analyses. Mainly, less wettable materials sustaining dropwise condensation, denoted by contact angles of 110°, are compared to materials that can withstand dropwise condensation with more wettable surfaces (those that can sustain 90° and 70° contact angles) without resulting in droplets merging into a film. The changing contact angle manifests itself in the transport problem as reducing the conduction resistance through the droplet for lower contact angles. This is a direct result of the conduction path through the droplet, and onto the cold wall, being reduced as contact angles diminish. The work of Nijaguna was essential in accurately accounting for his phenomenon [25]. These effects of varying contact angle on dropwise condensation are thus implemented and discussed in all three models.

1.4 Organization

This dissertation explores various models developed to investigate the effects of reduced droplet sizes for dropwise condensation. The remaining structure of this thesis is as follows:

Chapter 2 provides the literature review and background that inspired this research. It covers previous experimental and computational work that has been done related to this research.

Chapter 3 discusses the first model developed using MATLAB to explore the behavior of the condensation heat transfer coefficient for reduced droplet sizes on a cluster of droplets undergoing dropwise condensation.

Chapter 4 discusses a Direct Simulation Monte Carlo model developed for a single droplet undergoing dropwise condensation. The model is developed using .C programming to explore the behavior of the condensation heat transfer coefficient for reduced droplet sizes.

Chapter 5 discusses a second Direct Simulation Monte Carlo model developed for a droplet cluster undergoing dropwise condensation. The model is developed using .C programming to explore the behavior of the condensation heat transfer coefficient for reduced droplet sizes.

Chapter 6 compares the different models developed and compares them to experimental work.

Chapter 7 provides an overall summary and conclusion to this dissertation.

Chapter 2

Theory and Literature Review of Previous Work

2.1 Introduction

This chapter provides a literature review for this dissertation, covering topics related to modeling dropwise condensation heat transfer at reduced diameters. The chapter provides relevant theories and background on the work done in producing some of these specific theories that are employed in this dissertation. The literature review specifically aims to address the following topics:

- A general review of the theories and concepts behind condensation
- A review of theories and concepts specific to dropwise condensation
- A review of the approaches that can be used to model condensation
- A review of the the mechanisms expected to influence dropwise condensation at the scales of interest

2.2 Overview of Condensation

Condensation of a vapor occurs when the vapor is cooled to a temperature below the equilibrium saturation temperature corresponding to its vapor pressure, also known as its dew point. Condensation can occur naturally, such as when water droplets condense on an ice cold glass of water, or when dew drops form on grass in the mornings, just to mention some examples. Condensation can also be induced in various industrial applications for favorable uses, such as in power plant heat exchangers to condense exhaust steam from a turbine to

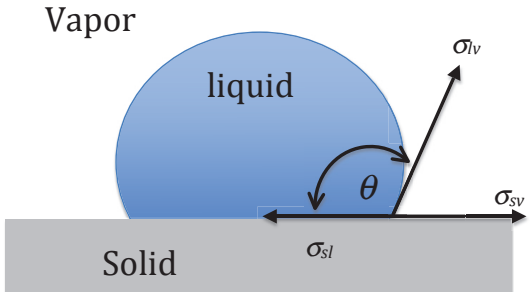
obtain maximum efficiency, or in condensers used in distillation processes to collect the water.

Condensation can manifest itself in two main forms:

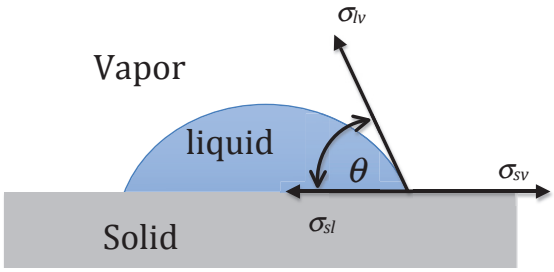
1) Film Condensation, or

2) Dropwise Condensation

The type of condensation that forms can highly depend on the nature of the medium that vapors are condensing on. If condensing on a solid surface, the surface-energy of the liquid relative to the surface-energy of the solid can be a good indicator of the behavior the liquid will have on the solid [26].



(a) Hydrophobic contact angle



(b) Hydrophilic contact angle

Figure 2.1: Contact angles for a liquid droplet condensing on a solid surface

Although highly simplified, the interaction of a liquid on a solid can be understood at a conceptual level by observing Figure 2.1, which depicts a force balance at the contact line of a liquid droplet condensing on a solid surface for two different types of surfaces. Here, σ_{lv} is the liquid surface-free-energy, σ_{sv} is the solid surface-free-energy, σ_{sl} is the solid/liquid interfacial-free-energy, and θ is the contact angle that can form for a liquid on a solid surface, measured

through the liquid as shown in the figure. At equilibrium, by following the illustrations in Figure 2.1, a force balance in the horizontal direction at the contact line can be expressed as

$$\sigma_{lv}\cos\theta = \sigma_{sv} - \sigma_{sl}, \tag{2.1}$$

which is well known as *Young's equation* [7]. Although this is a simplistic model assuming a uniform smooth surface ignoring surface roughness and impurities, a lot of insight can be gained into the potential type of condensation that can form. Scenarios where the surface-free-energy of the solid is less than the surface-free-energy of the liquid are more likely to produce situations as in case (a), where the surface is considered hydrophobic to the liquid and can result in droplet formation with contact angles greater than 90°. This scenario considers the solid to have poor wettability. Scenarios where the surface-free-energy of the liquid is less than that in the solid are more likely to produce situations as in case (b), where the surface is considered hydrophilic to the liquid which can result in droplet formation with contact angles less than 90°. This scenario considers the solid wettable (or to have high wettability) and is more likely to induce dropwise condensation. In these cases where the surface-free-energy of the liquid is much less than that of the solid surface, the likelihood of droplet formation is greatly diminished and can result in the liquid fully wetting the solid, thereby inducing film condensation [7].

Extensive research comparing filmwise versus dropwise condensation, such as that of Takeyama and Shimizu [27], has shown that, for comparable conditions, the resulting heat transfer for dropwise condensation can be as much as an order of magnitude higher than that for film condensation [7]. For example, in their work for steam condensing on a short vertical copper surface, Takeyama and Shimizu showed how the disparity between heat transfer coefficients grew as the amount of wall subcooling ($T_{sat} - T_{wall}$) increased. Specifically, for a wall subcooling of 10 °C, heat transfer coefficients for filmwise condensation were on the order of 6x10^4 (W/m^{2}K), and for dropwise condensation the heat transfer coefficient was on the order of 4x10^5 (W/m^{2}K). Similarly, Marto et al. [28] compared several low surface-free-energy polymer coatings to promote and sustain dropwise condensation of steam. They obtained dropwise condensation heat transfer coefficients as much as six times larger than film condensation heat transfer coefficients. In separate and independent studies for steam condensing on metallic surfaces at standard atomospheric pressure, Koch et al. and Rausch et al. [29, 30] also showed that dropwise condensation heat transfer could prove to be up to 5 times greater than filmwise condensation heat transfer for the same amount of subcooling under comparable conditions.

Due to the fact that dropwise condensation can prove to be a more efficient form of condensation, as corroborated by the multiple research studies discussed above [27, 31, 28, 29, 30, 32], this dissertation focuses on investigations for dropwise condensation.

2.3 Inducing Dropwise Condensation

In Chapter 1 it was mentioned how dropwise condensation can be promoted by one of three methods:

1.) Temporarily injecting a non-wetting chemical into the vapor which subsequently deposits on the condensing surface,

2.) Temporarily introducing a substance such as a fatty acid or wax onto the solid surface, or

3.) Permanently coating the condensing surface with a low-surface-energy polymer.

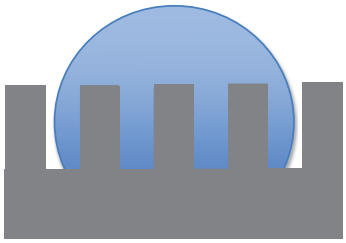
The third method of promoting dropwise condensation has become of particular interest because of its enduring nature and higher chance of sustaining continuous dropwise condensation [7]. As a result, much of the recent research in dropwise condensation has focused on developing and investigating the behavior of textured superhydrophobic surfaces that can sustain dropwise condensation [15]. These textured superhydrophobic surfaces can lead to enhanced heat transfer by inducing dropwise condensation to the point that condensate droplets nucleate and can roll down the surface at smaller sizes. In theory, these reduced droplet sizes should offer lower thermal resistances, and in effect enhance the heat transfer, rendering these superhydrophobic surfaces ideal candidates for dropwise condensation.

Numerous research studies have investigated the effects of producing such superhydrophobic surfaces through various methods [8, 9, 10, 11, 12, 13, 14]. To produce these surfaces, hydrophobic materials are usually applied onto micro and nano rough textured surfaces to promote hydrophobicity. Common techniques for fabricating these micro and nano structured surfaces vary from lithography, etching, and deposition. Advanced techniques can even attempt to impose hierarchical roughness, created by superposition of two rough patterns at different length scales, but the end goal to attain superhydrophobicity is still the same [8].

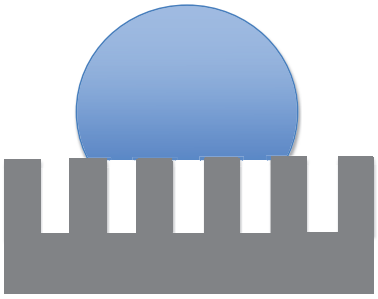
Johnson and Dettre [11] showed that two different equilibrium states can result from these superhydrophobic surfaces, corresponding to previous studies by Wenzel [14] and Cassie et al. [9]:

1) A Wenzel state, where the droplet is impaled on the surface roughness (Figure 2.2a), or

2) A Cassie-Baxter state, where the droplets are suspended on top of the surface roughness(Figure 2.2b).



(a) Wenzel State



(b) Cassie-Baxter State

Figure 2.2: Wetting modes for a superhydrophobic surface

Analyzing these wetting states and their viability of sustaining dropwise condensation on these superhydrophobic surfaces has been studied on multiple levels [33, 34, 35, 36, 37]. For example, Dorrer and Ruhe [33] studied the wetting behavior of microstructured post surfaces coated with a hydrophobic fluoropolymer. Narhe et al. [34] sudied growth dynamics on a micro-milled copper surface. Yoshimitsu et al. [35] studied the effects of surface structure on hetptadecafluorodecyltrimethoxysilane ($CF_3(CF_2)_7CH_2CH_2Si(OCH_3)_3$ coated micro-milled surfaces. The results of these investigations showed how, under specific prescribed conditions, steam drops condensing in the Wenzel state can transition into a Cassie state and roll off the surface at smaller diameters. In practice, however, this did not happen uniformly for all droplets on the surface. Dorrer and Ruhe studied pillared surfaces with variable spacing, but of particular importance were surfaces with a pillar width of 4 μm and 8 μm spacing. These studies showed that when droplets in a Wenzel state were small enough to cover no more than four pillars, coalescence occurred for droplets coming into contact with Cassie drop, resulting in a coalesced drop that was in a Cassie state. However, if the droplet was large enough so that more than four pillars were covered by the droplet in the Wenzel state, a Cassie state was not reached due to the stronger pinning by the multiple pillars. That particular scenario resulted in larger droplets that did not roll off as easily, hindering drop-

wise condensation. These studies showed that, in principal, a superhydrophobic surface can result in small droplets rolling off with high contact angles. In practice, however, droplets were of mixed Wenzel and Cassie states and the surface did not *uniformly* produce these higher contact angle droplets. In addition, many of the pinned Wenzel droplets trapped other droplets, preventing them from rolling off and coalescing into bigger and bigger droplets.

The above studies show that in theory, superhydrophobic surfaces can lead to enhanced heat transfer by inducing dropwise condensation where condensate droplets nucleate and can roll down the surface at smaller sizes. Hence, superhydrophobic surfaces should be better candidates at promoting and enhancing dropwise condensation. Due to the difficulties encountered in practice, however, continuing research such as those stated above are still ongoing with the aim of sustaining dropwise condensation for reduced diameters [10]. The efforts of such research studies that continuously aim to create these superhydrophobic surfaces to sustain dropwise condensation for reduced droplet diameters thus further motivates our research to theoretically investigate the limits of dropwise condensation at these potential reduced droplet sizes.

2.4 Modeling

2.4.1 Modeling of Heat Transfer for Dropwise Condensation

The theory of dropwise condensation from Rose and co-workers led to some of the earlier analytical models for dropwise condensation [38, 39, 40]. Their work predicted that the average condensation surface heat flux $q^{''}$ is the integrated effect of condensation heat transfer on a single droplet q_D with size r, multiplied by $n_d^{''}(r)dr$, the number of droplets per unit surface area with radius values between r and $r + dr$

$$q^{''} = \int_{r_{min}}^{r_{max}} q_D n_d^{''}(r)\,\mathrm{d}r. \tag{2.2}$$

This type of model predicts that the condensation heat transfer to smaller droplets is more efficient than to larger ones (q_D increases as r decreases), which suggests that inducing conditions that cause droplets to grow and depart the surface at small sizes would enhance dropwise condensation.

The work of Wu and Maa [41, 42] employ the Le-Fevre and Rose model above (Equation (2.2)) in conjunction with the droplet distribution of Rose and Glicksman [43] for drops greater than 5μm that grow by coalescence. Equations (2.3) and (2.4) show this model and the droplet distribution from Rose and Glicksman represented as

$$n_d''(r) = \frac{1}{3\pi r^2 \hat{r}} \left(\frac{r}{\hat{r}}\right)^{-2/3}, \tag{2.3}$$

where the expression for $\hat{r}$ can be obtained as

$$\hat{r} = K_1 \left(\frac{\sigma}{\rho g}\right)^{1/2} \tag{2.4}$$

where K_1 is a best-fit constant for steam data at atmospheric pressure, found to be equal to 0.4 [38, 43, 44, 45]. Wu and Maa used a population balance method to estimate the droplet distribution for smaller sizes, where it is assumed that these smaller droplets grow mainly by direct condensation where no coalescence occurs. However, in their model for heat transfer through a single drop, only the conduction resistance through the drop was considered, which tended to overestimate the heat transfer. Abu-Orabi [46] later built on that model by considering the resistance of a hypothetical promoter layer to develop his analytical model for dropwise condensation. When compared to the Nusselt model for filmwise condensation, these predictions showed to be higher, as would be expected from dropwise condensation. When his model was compared to the experimental work of Wilmshurst and Rose [47] for steam condensing on a wall at 306 K, however, while the model was close in predicting transport, it tended to overestimate the predicted heat flux [48].

Modeling of this type, where a population balance method is used to attain the droplet distribution for smaller drops and Equations (2.3) and (2.4) are used for larger droplet sizes, has been widely used for dropwise condensation due to its simplicity [48]. The more recent work of Vemuri et al.[44] and Kim et al. [49] also make use of Equation 2.2 in developing their analytical model with similar methods of approximating $n_d''(r)$. These models attempted to account for effects that Abu-Orabi omitted, such as contact angle, interfacial resistance, and superhydrophibicity. None of these models, however, attempt to account for droplets deviating extremely too far off from a continuum solution for very small droplet sizes.

2.4.2 Modeling at Smaller Sizes

As mentioned in the Introduction, when droplet sizes are reduced to droplet diameters d on the order of (or greater than) the mean free path of the flow lambda$_{\mathrm{m}}$, the continuum fluid approximation starts to break down and the particle nature of matter must be taken into account[50]. Using the Knudsen number ($\mathrm{Kn}_d = \lambda_m/d$) as a characterizing parameter with characteristic length d, flow behavior can be categorized into different regimes.

At one end of the Knudsen number spectrum, where the characteristic length of the system is much greater than the mean free path, is the flow behaving as a continuum with

the classical no-slip condition. When droplet sizes are reduced to the point where Knudsen numbers approach 0.01, the flow behavior starts to deviate from classical mechanics. At the other extreme is the free molecular flow, where the characteristic length of the system is on the order of the mean free path and molecules seldom collide with each other, if at all. Transport in between the extremes, in the transition regime, cannot be explained by purely using free molecular theory nor continuum theory, and it therefore becomes challenging to develop models accounting for transitions through this regime. To visualize these regimes, Figure 1.2 is placed again below as Figure 2.3 for the reader's convenience.

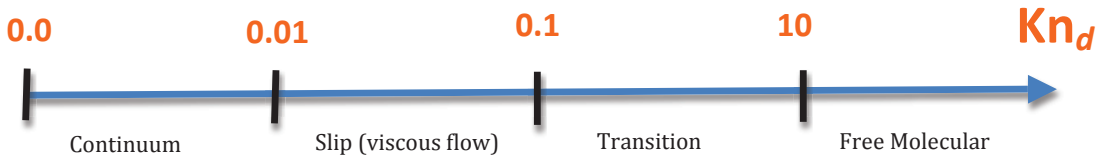


Figure 2.3: Flow regimes for increasing Knudsen number [16]

2.4.3 The Direct Simulation Monte Carlo Method

As the Knudsen numbers characterizing a flow start deviating away from a continuum, modeling approaches considering the deviation from classical mechanics have to be taken into account. Models for rarefied flow, where the Knudsen number characterizing the flow is greater than 0.1, can be described by directly solving the Boltzmann equation, which describes the statistical behavior of a flow in non-equilibrium statistical mechanics. Solving the Boltzmann equation directly can become extremely complex, however. These flows can also be modeled using the Direct Simulation Monte Carlo(DSMC) method of Bird, which solves the Boltzmann equation in a probabilistic fashion [21]. According to Bird, the applications and the extensions of the DSMC method have evolved to go beyond the range of direct solutions to the Boltzmann equation [51, 22].

In DSMC simulations modeling a fluid, the flow is modeled as particle simulators moving in a simulated physical space in such a way that each particle simulator represents a group of molecules, resulting in the number of simulated particles as a fraction of the number of molecules that would represent the actual flow. The simulation moves in incremental time-steps that are less than the mean free time so that particles can move distances no larger than the mean free path. In this way, particle motions and particle collisions are decoupled over time allowing the particle collisions to be calculated in a probabilistic fashion in be-

tween time-steps. During particle simulator collisions, conservation laws of particle collisions are obeyed. The general objective behind the DSMC method is to calculate practical gas flows through the use of a collision mechanics model of molecules, producing results that are equivalent to directly solving the Boltzmann equation for comparable conditions. When simulated for sufficiently long times, DSMC methods can prove to be highly accurate for flows characterized by higher Knudsen numbers [24].

As components of the DSMC method have evolved, such as the collison models [50, 52], the DSMC method has been used in a range of applications [53, 54, 23, 55, 56, 22]. The DSMC method has been been applied to (and comparisons from such simulations have been shown to be in agreement) flows in the range where the Navier Stokes equations can be used to consider slip ($0.01 < \mathrm{Kn}_d < 0.1$) [53, 54], through the transition range ($0.1 < \mathrm{Kn}_d < 10$) [23, 55], and down to flows resembling dilute gases ($\mathrm{Kn}_d > 10$) [56, 22]. The DSMC method has been used in a plethora of applications since it was first used by Bird in 1963 [57, 23], and the reader is encouraged to reference specific studies for details on applications of interest.

The DSMC method has been previously employed in condensation applications. Successful and relevant applications of DSMC for homogeneous condensation, where nucleation initiates within the vapor as opposed to a preferential site on a surface (heterogeneous), include the work of Zheng et al. [58]. In 2009 Zheng et al. worked on modeling condensation in free expansion plumes, where Knudsen numbers in the range of .01 are usually observed [58]. Zheng's DSMC studies attempting to predict the effect that Raleigh scattering intensities had on cluster growth were compared to experimental results, showing fair agreement. A heterogeneous nucleation example employing the DSMC method includes the work of Li [59]. In Li's DSMC model, N_2 molecules condensing on CO_2 nuclei in an expanding jet were simulated, and condensate growth rates were shown to be in agreement with experimental data [59].

2.4.4 DSMC and Mechanisms at Reduced Droplet Sizes

The above are just a couple of general and successful examples amongst many showing the effectiveness DSMC simulations have had within condensation. Specific DSMC studies by Carey have explored the mechanisms that affect droplet condensation growth as droplet size decreases [60, 61, 62]. The DSMC studies of Carey et al. [61] indicated that for water droplets growing in a supersaturated argon and steam mixture, at very small diameters, non-continuum effects tend to diminish the growth rate of the droplets below that predicted by continuum theory. Their results indicated that non-continuum transport effects significantly alter the droplet growth rates for droplet diameters below about 1.5 μm. Studies by Carey have also shown that for droplet diameters in the nanometer range, interface curvature effects on surface tension and vapor pressure significantly affect droplet condensation transport [62, 18]. Carey and Oyumi [63, 60] used DSMC molecular simulations to study

transport near small clusters of condensing water droplets in supersaturated steam and in steam/air mixtures. These studies considered symmetric clusters of closely spaced condensing droplets. The studies indicated that for droplets with diameters near 1 μm separated by about 0.5 μm, the presence of adjacent droplets modifies the temperature fields near the droplets, which in turn modifies the rate of condensation over the surfaces of the droplets. The studies further suggested that as droplet sizes are reduced, the interface curvature significantly affects saturation conditions and surface tension.

To conceptualize the effects of surface tension and vapor pressure at reduced droplet sizes, a concise description of the relative importance is provided below. Direct applications of the DSMC method with details are discussed later in Chapters 4 and 5.

2.4.5 Surface Tension and the Tolman Length

In 1948, Richar C. Tolman predicted that the surface tension of a liquid droplet should be expected to decrease with reductions in droplet sizes for a wide range of circumstances compared to its flat-interface value. Tolman predicted that the variations in surface tension became especially significant for very small drops [17].

In his work, Tolman used a thermodynamic analysis to obtain a correction for the surface tension of a droplet of radius r, commonly regarded as Tolman's relation, which was introduced in Chapter 1. For the reader's convenience, the deviation for a droplet's surface tension σ_{lv} from the flat-interface surface tension σ_∞ is shown here as

$$\sigma_{lv} = \sigma_\infty(T)\left[1 + \frac{2\delta_T}{r}\right]^{-1}. \tag{2.5}$$

The Tolman length δ, which is a constant that varies from substance to substance, is a measure of the extent to which the surface tension of a small liquid drop differs from its flat-interface value. For details on his analysis and derivation, the reader is advised to reference Tolman [17].

Tolman's findings have proven to have impacted a wide range of research areas. Numerous methods to predict the Tolman length include theoretical models and molecular dynamics simulations such as that of Haye et al., Kalikmaov, and Nijmeijer et al. [64, 65, 66]. These studies, among others, predict that away from the critical point, the Tolman length is on the order of the effective diameter of the molecule. For water, which is of interest for the investigations presented here, Pruppacher et al. [19] specifically recommend using a Tolman length of 0.157 nm. This is about half of the effective diameter of a water molecule, which is in agreement with the theories finding the Tolman length to be on the order of the effective

diameter.

Researchers using the Tolman relation in their work have found it valuable for their models in a range of applications. For example, Glesen et al, incorprate the relation in their studies to predict formation and consumption rates of iron atoms [67]. Zhu et al. concluded that it was necessary to consider the Tolman correction to properly predict fluid dynamics on carbon nanotubes [68]. Carey also incorporates the Tolman relation in a DSMC study of interface curvature effects on transport near the interface of droplet embryos formed by nucleation in a supersaturated gas mixture, concluding that using the Tolman length is essential to help predict transport for such cases [62]. The above studies indicate that use of the Tolman correction is especially crucial at reduced droplet sizes as surface tension effects play important roles in transport associated with postnucleation growth of droplets. Theoretical treatments of related problems must have the capability to incorporate such complexities to accurately predict transport.

2.4.6 Droplet Vapor Pressure

According to Carey [7], the vapor phase of a pure substance may be brought to a supercooled or supersaturated state either by transferring heat through the walls of a containing structure, or by rapidly changing the pressure of the gas. Once a saturated state is reached, condensation of some of the vapor to liquid may be initiated if nuclei of the liquid phase are present in the system with changes in the system pressure, or if further removal of heat occurs. These potential nuclei could be molecules in a near-liquid state that have been adsorbed on a wall or on dust particles suspended in the vapor.

To consider the effects of curvature on vapor pressure, we consider the analysis provided by Carey [7] in a system for a liquid droplet of radius r in equilibrium with a surrounding vapor held at fixed temperature T_v and pressure P_v. At equilibrium, the temperature and chemical potential in the vapor phase μ_v and liquid phase (the droplet) μ_l must be the same.

$$\mu_v = \mu_l, \tag{2.6}$$

Then, the Gibbs-Duhem equation for a constant temperature process to evaluate the chemical potential of the vapor at equilibrium μ_{ve} gives:

$$\mu - \mu_{sat} = \int_{P_{sat}T_v}^{P} v\mathrm{d}P, \tag{2.7}$$

where the integral can be evaluated by using the ideal gas law to obtain an expression for the specific volume of the vapor as a function of pressure ($v = RT_v/P$). Evaluating the integral for the vapor phase gives:

$$\mu_{ve} = \mu_{sat,v} + RT_v \ln\left[\frac{P_v}{P_{sat}(T_v)}\right]. \tag{2.8}$$

For the liquid inside the droplet, Equation (2.7) can be used again to determine its chemical potential. If the liquid is taken to be incompressible, with v equal to the value for a saturated liquid at T_v, evaluation of the integral up to a pressure $P = P_{le}$ gives

$$\mu_{le} = \mu_{sat,l} + v_l[P_{le} - P_{sat}(T_v)]. \tag{2.9}$$

Using equation (2.6) and equating the values of μ_v and μ_l given by Equations (2.8) and (2.9), and using the fact that $\mu_{sat,v} = \mu_{sat,l}$ at equilibrium, a relation for P_v can be obtained:

$$P_v = P_{sat}(T_v)\exp\left\{\frac{v_l[P_{le} - P_{sat}(T_v)}{RT_v}\right\} \tag{2.10}$$

For the system considered, the pressures in the two phases at equilibrium can be related through the Young-Laplace equation:

$$P_{le} = P_v + \frac{2\sigma_{lv}}{r_e}, \tag{2.11}$$

where P_{le} is the pressure of the liquid with surface tension σ_{lv}, which is at equilibrium with the vapor pressure P_v.

Substituting Equation (2.11) into Equation (2.10) results in

$$P_v = P_{sat}(T_v)\exp\left\{\frac{v_l[P_v - P_{sat}(T_v) + 2\sigma_{lv}/r_e}{RT_v}\right\} \tag{2.12}$$

From Figure 2.4 showing the liquid and vapor states for a liquid droplet in equilibrium with its surrounding vapor, it can be seen that P_v is much closer to P_{sat} than to P_{le}. From this we can infer that the value of $P_v - P_{sat}(T_v)$ is small compared to $2\sigma_{lv}/r_e = (P_{le} - P_v)$, and Equation (2.12) can be well approximated by Equation (2.13)

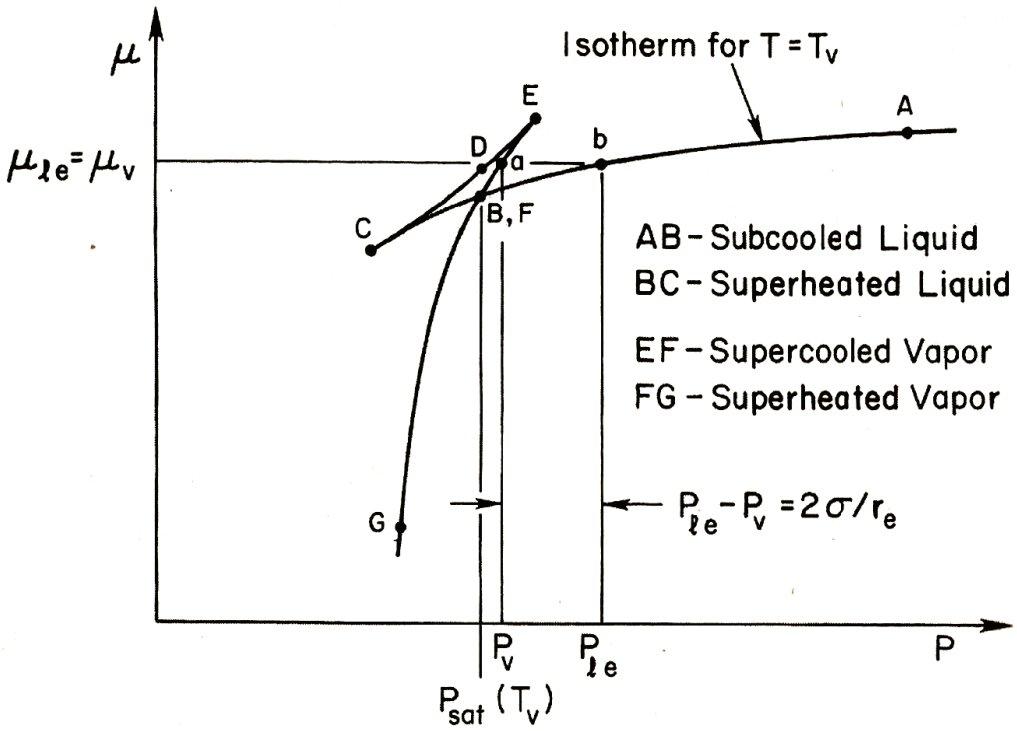


Figure 2.4: The liquid and vapor states for a liquid droplet in equilibrium with its surrounding vapor (**figure borrowed from source, where $\sigma = \sigma_{lv}$* [7])

$$P_v = P_{sat}(T_v)\exp\left(\frac{v_l 2\sigma_{lv}}{r_e R T_v}\right). \tag{2.13}$$

For the models discussed in this dissertation, Equation (2.13) is the functional form that is used as the droplet vapor pressure.

2.5 Closing the Literature Review

All the previously mentioned theories and applications described in this chapter lay the basic framework for details incorporated into our approach to model the limits of dropwise condensation at reduced diameters. Direct applications of the fundamentals described in this chapter are mentioned in detail in the following chapters.

Chapter 3

Approximation Model on a Droplet Cluster

3.1 Introduction

The work presented in this chapter discusses the preliminary model used to simulate dropwise condensation heat transfer for an array of droplets. To simplify the actual, yet more complicated, nonuniform array of droplet sizes that occurs during actual dropwise condensation, the modeled system is defined as a cluster of droplets with a single mean droplet size arranged in a hexagonal pattern. The model attempts to illustrate the behavior of the heat transfer coefficient as mean droplet diameters are reduced. This model includes sizes large enough for the condensation process to be modeled as a continuum, down to droplet sizes where the space in between the droplets is smaller than the mean free path such that the transport is completely ballistic. For this ballistic limit, A Monte Carlo scheme was developed to obtain the fraction of molecules from the ambient that would be incident on droplets ignoring intermolecular collisions. The transition regime from fully continuum to fully ballistic transport was modeled using an approximation method, which is described in detail below.

This chapter is organized as follows:

- The nomenclature for this chapter is presented in Section 3.2.
- The computational domain and boundary conditions for this approximation model are defined in Section 3.3.
- A detailed description of our analysis and approach to the model is described in Section 3.4.

- The results from this model are discussed in Section 3.5 in regards to predicted heat transfer coefficients.
- The implications based on the predictions of this model are described in Section 3.6

3.2 Nomenclature

A_{di}	area of three droplet interfaces within the unit cell
A_{us}	area of unit cell footprint on surface
A_w	area of cold wall surface between droplets in unit cell
d	droplet diameter
F_{id}	fraction of molecules emitted from interface that a droplet interface
F_{ii}	fraction of molecules emitted from interface that return to the same droplet
F_{us}	fraction of molecules from upper surface of unit cell that hit a droplet interface
h_d	dropwise condensation heat transfer coefficient
j_{bi}	molecular flux to interface in ballistic limit
j_{ds}	molecular flux to droplet interface from surrounding vapor space
j_d	molecular flux from liquid to vapor at droplet interface
j_s	molecular flux from vapor space in cell to interface
j_∞	molecular flux from ambient into unit cell
k_B	Boltzmann constant
Kn_d	Knudsen number, $=\lambda_\mathrm{m}/d$
N_A	avogadro's number
M_w	molecular weight
P_{vd}	equilibrium vapor pressure for curved droplet interface
$P_{sat}(T_d)$	flat interface saturation pressure at temperature T_d
s	distance between droplet interfaces at mid plane
S_f	shape factor for droplet
T_d	interface temperature
T_s	temperature of vapor in space between droplets
T_w	wall temperature
$T_{sat}(P_\infty)$	flat interface saturation temperature at pressure P_∞
λ_m	mean free path of molecules in vapor
ρ_l	specific volume of liquid at interface temperature T_d
σ_{lv}	surface tension at liquid-vapor interface temperature T_d
σ	interface accommodation coefficient
θ	contact angle

3.3 Definition of Computational Domain

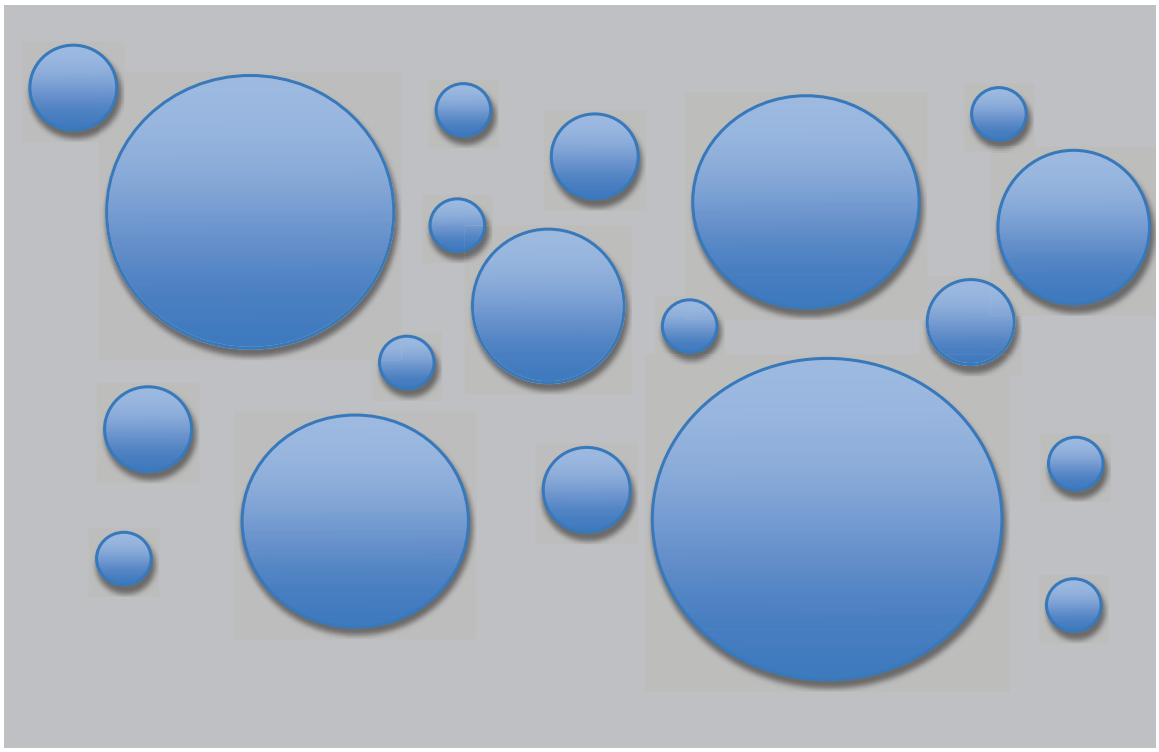


Figure 3.1: Actual Array of Droplet Size Distribution

Actual dropwise condensation on a surface occurs in a very nonuniform fashion, where droplet sizes can vary over a range as illustrated in Figure 3.1. For the purposes of this model, to simplify this nonuniform array of droplet sizes that occurs during actual dropwise condensation, the modeled system is defined here as a cluster of droplets with a single mean droplet diameter d arranged in a hexagonal pattern. Figure 3.2 illustrates the idealized droplet array used for this model, with the dashed lines outlining the hexagonal layout.

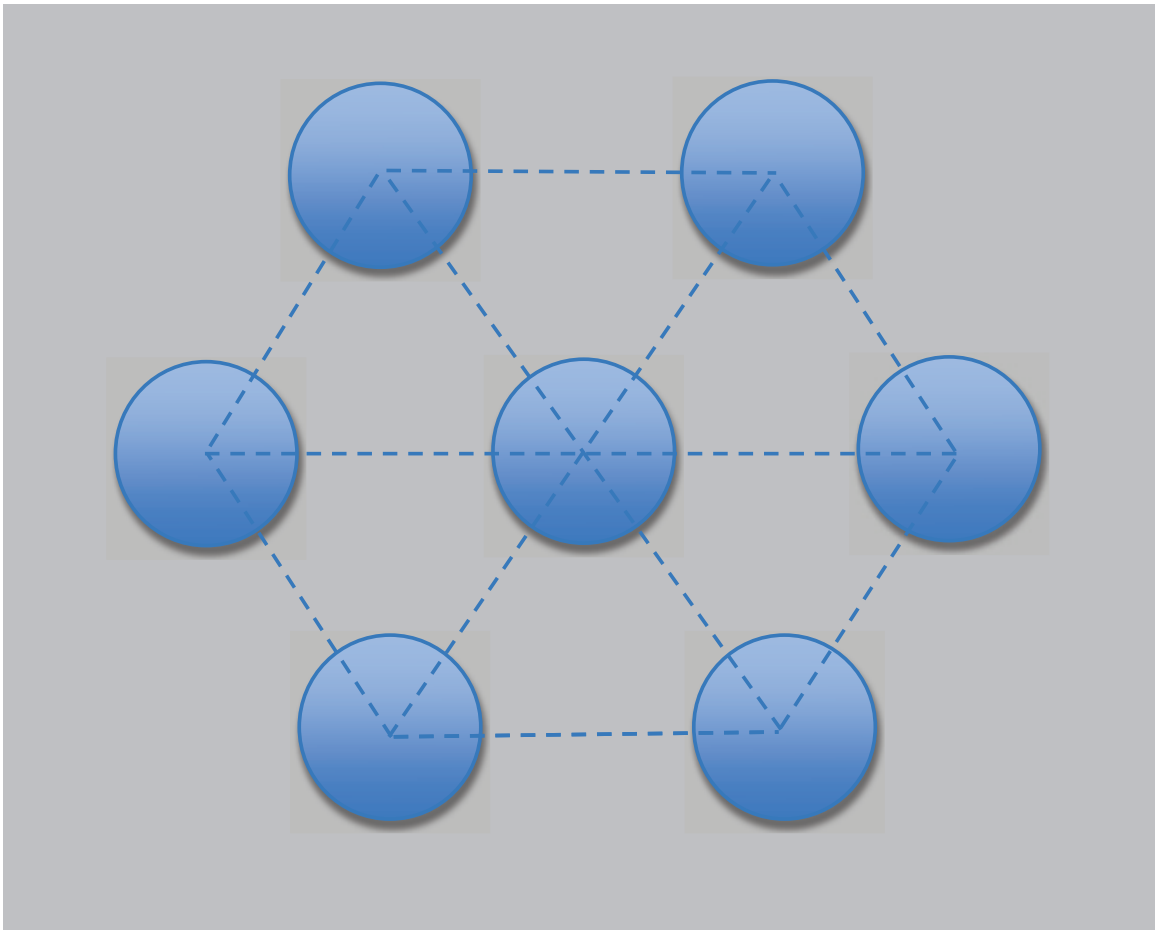


Figure 3.2: Idealized Array of Droplet Size Distribution

The hexagonal pattern gives the idealized layout symmetry, as outlined by the triangular cross-sections, which in turn allows our analysis to be focused on a single triangular cross-section. The array of droplets of uniform size is considered here to explore how the transport mechanisms of dropwise condensation change with droplet size at very small droplet diameters. Figure 3.3 shows the equilateral triangular unit cell of this array. The droplets are modeled as sphere segments with a liquid-vapor interface that meets the smooth solid surface at contact angle θ.

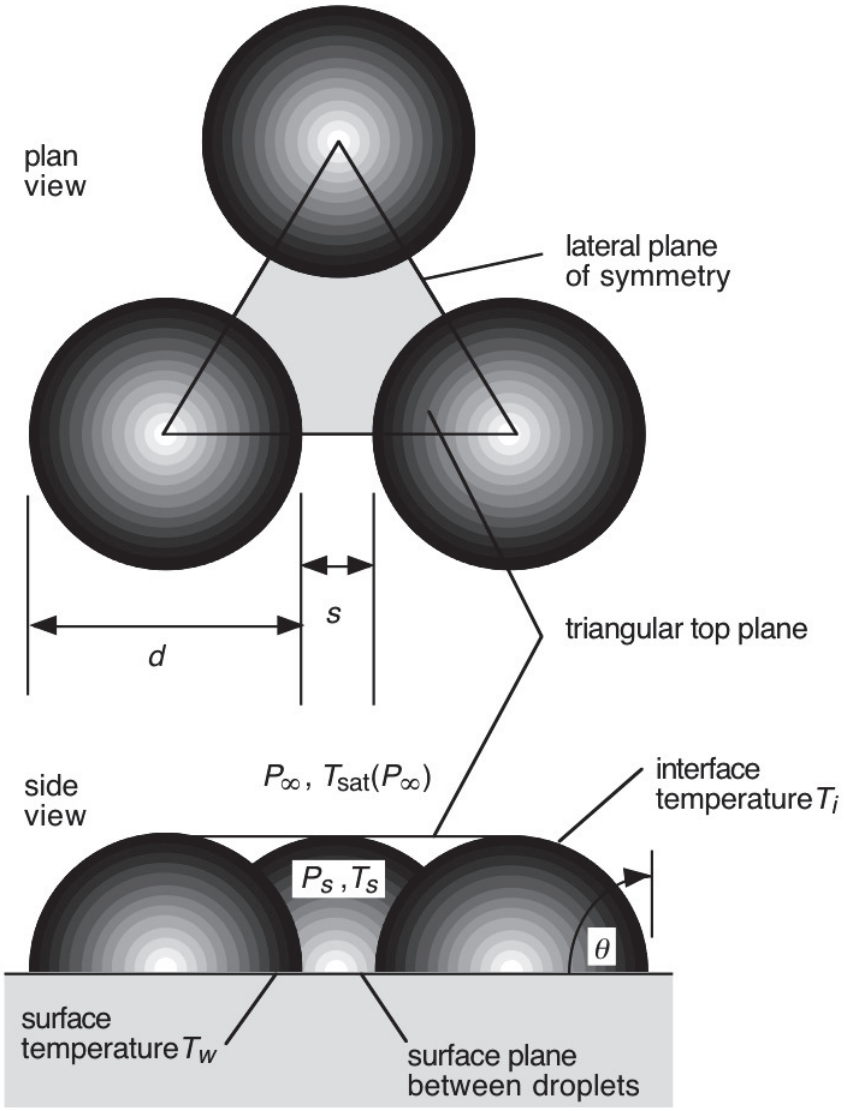


Figure 3.3: Model unit-cell system

Throughout this study, the radial separation between the interfaces of adjacent droplets in the center-plane of the droplet spheres is referred to as s. Beyond the triangular top plane, temperature and pressure are are taken to be the ambient saturation conditions $T_{sat}(P_\infty)$ and P_∞. This array of droplets of uniform size is considered here to be condensing on a cold wall at temperature T_w. Figure 3.3 comprehensively illustrates the domain used here

to explore how the transport mechanisms of dropwise condensation change with droplet size at very small droplet diameters.

3.4 Approach to Modeling

3.4.1 Modeling of the Mechanism

For continuum transport, kinetic theory dictates that the flux of molecules from the space within the cell shown in Figure 3.3 that will impact the surface of the droplets is given by

$$j_s = \frac{1}{4}\left(\frac{P_s}{k_B T_s}\right)\sqrt{\frac{8N_A k_B T_s}{\pi M_w}}, \tag{3.1}$$

where T_s and P_s are the temperature and pressure in the space between the droplet in the cell [60].

If the interface of the droplet is at temperature T_d, kinetic theory arguments as stated by Carey [7, 20], also indicate that the flux of molecules emitted from the interface is given by

$$j_d = \frac{\sigma}{4}\left(\frac{P_{vd}}{k_B T_d}\right)\sqrt{\frac{8N_A k_B T_d}{\pi M_w}}, \tag{3.2}$$

where σ is the interface accommodation coefficient. Here the condensing and evaporation accommodation coefficients are taken to be equal. In (3.2) P_{vd} is the equilibrium vapor pressure discussed in Chapter 2, which for a droplet of finite size, is given by

$$P_{vd} = P_{sat}(T_d)\exp\left(\frac{4\sigma_{lv}}{\rho_l d R T_d}\right), \tag{3.3}$$

where $P_{sat}(T_d)$ is the flat-interface saturation pressure and σ_{lv} is the surface tension at the interface temperature T_d. Considering changes in droplet exposure due to contact angle, the net heat transfer due to condensation at the interfaces of the droplet in the unit cell with total interface area $A_{di} = (\pi d^2/4)(1 - cos\theta)$ is dictated by the difference in molecular fluxes:

$$\dot{q}_{cc} = \left(\frac{M_w h_{lv}}{N_A}\right)(j_{ds} - j_d)\, A_{di}, \tag{3.4}$$

where $j_{ds} = j_s$ is the flux of molecules from the surrounding vapor space hitting the interfaces of the droplets.

At steady state, the energy delivered to the interface by the latent heat associated with the net molecule flux must equal the heat transfer transfer $\dot{q}_{cd}$ conducted from the interface to the cold wall surface through the liquid of the three droplet segments in the unit cell. The latter is given by

$$\dot{q}_{cd} = \frac{1}{2}\left(T_d - T_w\right) S_f k_l, \tag{3.5}$$

where S_f is the conduction shape factor for a single droplet, to be discussed later. Equating the two heat transfer rates yields a non-linear equation that must be satisfied at steady state.

$$\left(\frac{M_w h_{lv}}{N_A}\right)\left[j_{ds}(T_s) - j_d(T_d)\right] A_{di} = \frac{1}{2}\left(T_d - T_w\right) S_f k_l, \tag{3.6}$$

We expect that both j_d and j_{ds} in Equations (3.4) and (3.6) are affected by the accommodation coefficient σ. As discussed by Carey and coworkers [60, 61, 63, 18, 62], prior studies suggest that the accommodation coefficient should be close to one for pure water droplets in contact with pure vapor. Since we are modeling a pure water system, a value of σ near one is anticipated for the systems of interest here. The interface temperature adjusts to satisfy the steady state energy balance Equation (3.6). This non-linear equation can be solved to determine the interface temperature. Once T_d is determined, the dropwise condensation heat transfer coefficient for the unit cell is computed as Eq. (3.7), where $A_{us} = \sqrt{3}(dsin\theta + s)^2/4$ is the triangular area covered by the unit cell and imposed on the cold wall surface.

$$h_d = \dot{q}_{cd}/[A_{us}(T_{sat}(P_\infty) - T_w], \tag{3.7}$$

This implicitly assumes that condensation occurs only on the interface of the droplets and that vapor convection heat transfer to the solid wall in the region between the droplets is negligible. These are reasonable idealizations given the high heat transfer rate associated with the condensation process on the interfaces.

When the diameter of the droplet is smaller than the mean free path of vapor molecules in the space between the droplets, as discussed in Chapter 2, molecules from the ambient travel ballistically from the triangular plane at the top of the unit cell to the droplet interfaces. The diffusive flux across the top plane is dictated by kinetic theory:

$$j_\infty = \frac{1}{4}\left(\frac{P_\infty}{k_B T_{sat}(P_\infty)}\right)\sqrt{\frac{8 N_A k_B T_{sat}(P_\infty)}{\pi M_w}}, \tag{3.8}$$

The fraction of this diffusive flux over the triangular surface at the top of the unit cell that travels ballistically to impact the interfaces of the three droplets is defined here as F_{us}. Droplet emission from the interface of the droplets occurs as described by Equation (3.2), just as in the continuum case. Of the molecules emitted from the interface of the droplets, some will escape into the ambient, and some will strike a nearby droplet interface. F_{id} is the fraction of molecules emitted from interfaces that hit an interface of a different droplet, and F_{ii} is the fraction of molecules emitted from interfaces that hit the interface of the droplet from which they were emitted. It follows that the molecular flux from the vapor incident on the droplet interfaces in the ballistic limit j_{bi} can be computed as:

$$j_{bi} = \sigma \left[j_\infty \left(\frac{A_{us}}{A_{di}} \right) F_{us} + j_d F_{id} + j_d F_{ii} \right]. \tag{3.9}$$

For the purposes of this model, we adopt the hypothesis that the transport near the droplets will undergo a transition from continuum to ballistic over the range of droplet diameters corresponding to Knudsen numbers of $0.05 < \mathrm{Kn}_d < 20$, where $\mathrm{Kn}_d = \lambda_\mathrm{m}/d$. The mean free path λ_m for the steam molecules was computed from kinetic theory as $\lambda_\mathrm{m} = k_B T_{sat}(P_\infty)/[\sqrt{2}\pi D_m^2 P_\infty]$. In this calculation, the effective diameter D_m for water molecules was taken to be 3.61 Angstroms, which is the value that matches the steam viscosity predicted by kinetic theory to the value recommended in standard steam saturation tables at 100 °C.

In this transition range, we therefore postulate a smooth transition and adopt the following relation to predict the flux of molecules on the droplet interfaces:

$$j_{ds} = j_s \left(1 - e^{-0.5/Kn_d}\right) + j_{bi} e^{-0.5/Kn_d}. \tag{3.10}$$

Note that in Eq. (3.10), j_s is to be evaluated using Eq. (3.1) and j_{bi} is evaluated using Eq. (3.9). The constant 0.5 in Eq. (3.10) was chosen so transition from full continuum transport to full ballistic transport occurs over the hypothesized range $0.05 < \mathrm{Kn}_d < 20$. In the general case considered here, molecules with an energy characterized by the far field temperature $T_{sat}(P_\infty)$ are entering the space between the droplets from the ambient, and molecules with an energy characterized by T_d are entering this space from the droplet interfaces. Based on a steady state energy balance in a control volume around this region, the mean temperature of molecules in this region T_s must be given by

$$T_s = \frac{[j_d A_{di} T_d + j_\infty A_{us} T_{sat}(P_\infty) + j_{sm} A_w T_w]}{j_{sm}[A_{di} + A_{us} + A_w]}, \tag{3.11}$$

where $A_w = A_{us} - \pi(d\sin\theta)^2/8$ is the area of cold wall surface between droplets in the unit cell, and $j_{sm} = (j_d A_{di} + j_\infty A_{us})/(A_{di} + A_{us})$ is the molecular flux from the space between droplets incident on the surfaces bounding that space.

With the model relations developed above, the following iterative scheme can be used to solve for T_d and predict the dropwise heat transfer coefficient for the unit cell for specified system conditions:

i. A value is guessed for T_d

ii. j_d is computed using Equations (3.2) and (3.3)

iii. T_s is computed using Equation (3.11) and j_s using Equation (3.1)

iv. Equations (3.8) and (3.9) are used to compute j_{bi}

v. Equation (3.12) is used to compute j_{ds}

vi. The difference between the left and right sides of Eq. (3.6) is computed as an error parameter E_f:

$$E_f = [j_{ds}(T_s) - j_d(T_d)]A_{di} - \frac{1}{2}(T_d - T_w)\,S_f k_l. \tag{3.12}$$

If $|E_f| < 10^{-4}(\frac{1}{2})[T_{sat}(P_\infty) - T_w]S_f k_l$, the guessed value of T_d was taken to be correct and iteration of T_d ends. If E_f is not small enough to meet this condition, a Newton-Raphson method is used to generate an improved guess for T_d, and the process loops back to step [ii].

vii. From the converged value of T_d, Eq. (3.7) is used to compute the dropwise condensation heat transfer coefficient h_d.

For specified values of d, s, T_w, P_∞ and σ, the above solution scheme can be executed if relations for the associated physical properties are available, and if S_f, F_{us}, F_{id}, and F_{ii}, are known. In this investigation, water saturation properties were determined using the X Steam water and steam property package within MATLAB [69]. Because we are specifically interested in small droplets, the correction for the effect of droplet curvature on surface tension described in Chapter 2 for very small droplets

$$\sigma_{lv} = \sigma_\infty(T_d)\left[1 + \frac{4\delta_T}{d}\right]^{-1} \tag{3.13}$$

was also applied to the value of the flat interface surface tension determined from the property package X Steam. Here $\sigma_\infty(T_d)$ is the flat interface surface tension evaluated at temperature T_d and δ_T is the Tolman length, taken to be the recommended value of 0.157 nm for water [18]. Methods used to determine the transport parameters S_f, F_{us}, F_{id}, and F_{ii}, are handled appropriately and details are described in the subsequent subsections.

3.4.2 Modeling of Droplet Conduction

In this investigation, the conduction shape factor, S_f, for a drop as a function of its contact angle, θ, was computed based on the analytical model developed by Nijaguna [25]. Nijaguna solved the boundary value conduction problem for a hemispherical drop and for a spherical segment of a drop, assuming a constant temperature for the drop's base and for the drop's curved surface. His analysis results indicate that the shape factor can be computed as a function of the droplet diameter and contact angle using the following relation:

$$S_f = \frac{\pi\, d \sin\theta}{4}\Phi, \tag{3.14}$$

where

$$\Phi = \frac{4\left\{\sum_{n=0}^{\infty} \frac{(4n+3)\,(2n+1)\,(-1)^{2n}(2n!)^2}{(2n+2)^2(2)^{4n}(n!)^4}\right\}}{\left\{\sum_{n=0}^{\infty} \frac{(4n+3)\,(2n!)\,(-1)^n}{(2n+2)\,(2)^{2n}(n!)^2}[\tan(\theta/2)]^{2n+1}\right\}}, \tag{3.15}$$

Equations 3.14 and 3.15 were used to determine the shape factor in our model calculations. In doing so, the convergence of the summations is fast enough that inclusion of ten terms is estimated to predict the shape factor to within a fractional difference of less than 10^{-11} of the limiting value. We therefore used $n = 10$ for our calculations of S_f in the heat transfer analysis described int he previous subsection.

3.4.3 Modeling of Ballistic Transport

To model the ballistic transport limit described in Section 3.4.1, we consider molecules entering the triangular aperture in the tangent plane to the top of the droplets shown in Figures 3.3 and 3.4. We assumed that the molecules striking this surface were coming at ambient conditions described by $T_{sat}(P_\infty)$ and P_∞. Upon passing through this aperture,

molecules travel down to the condensing surface in a ballistic fashion. We therefore treat the triangular aperture in this tangent plane as a diffusely emitting surface, where the total flux of molecules on that plane is dictated by kinetic theory for the prescribed ambient conditions. However, only a fraction of those molecules emitted from the triangular top surface will strike one of the droplets. To determine the fraction of those molecules that would strike a droplet, F_{us}, a Monte Carlo approach was used.

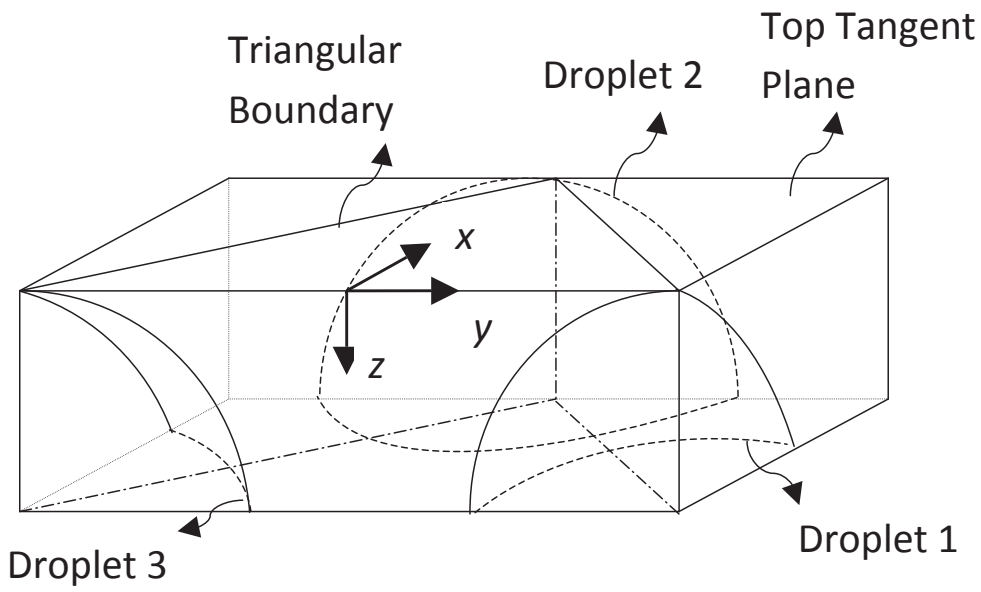


Figure 3.4: Model of computational domain used in Monte Carlo method determination of F_{us}, F_{id}, and F_{ii}

The Monte Carlo scheme developed to determine F_{us} begins by picking a random position within the top surface of the triangular unit cell. Locations of the particles representing the molecules are tracked in terms of the Cartesian coordinates shown on Fig. 3.4. Once a random position was chosen, a random direction for a particle to travel was generated by randomly setting the azimuthal angle between 0 and 2π and the latitude angle between 0 and π in spherical coordinates with their origin at the starting location, aiming the particle into the triangular domain. The particle was then moved along its trajectory given its starting position and direction. For these purposes, the molecules were modeled as infinitesimal particles moving at a velocity slow enough that the particle will take a few hundred steps to cross the cell domain to another boundary. If the particle crossed the interface of a droplet as it travels, it was counted as one hitting a droplet. If it crossed the cold solid surface or one of the lateral bounding planes of the unit cell, appropriate action was taken, consistent

with the boundary condition there.

By symmetry, an identical (symmetric) Monte Carlo simulation is presumed to be occurring in adjacent unit cells, with the result that for each particle exiting the simulation through a lateral wall, another would enter and follow the specular reflection of the first. Thus, in compliance with the symmetry and boundaries of the unit cell domain for the Monte Carlo simulation, the lateral boundaries were treated as specularly reflecting surfaces. The velocity vector of particles that strike them are changed so that the component of their velocity vector normal to the wall is reversed. Any particles striking the cold solid wall surface were reflected diffusely by picking a random direction of travel from the point of contact. Particles were tracked as they moved within the unit cell until they either crossed the interface of a droplet or passed through the triangular top aperture back into the ambient. If the latter happened, they were counted as not striking a droplet. The reader is advised to refer to Appendix A for the distribution sampling theory used in generating random particle directions and for the algorithm used in obtaining the fraction F_{us}.

After a particle either hit a droplet or escaped the unit cell, the value of F_{us} was recomputed as the ratio of the number of particles that hit a droplet surface to the total number of particles tracked in the simulation. Generally, the value of F_{us} fluctuated a bit initially, but beyond 5000 particles, it converged to an essentially constant value as demonstrated by Figure 3.5. Simulations were run for 7000 particles to determine F_{us} for each unit cell geometry of interest.

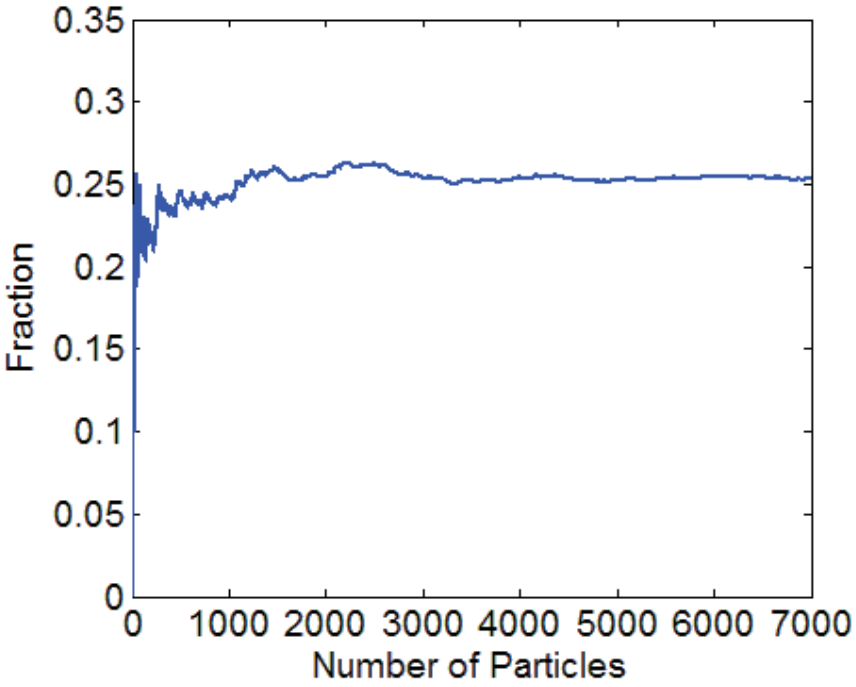


Figure 3.5: Sample calculation showing convergence of F_{us} for $s/d = 2$ and $\theta = 90°$

The results of the simulations indicated that the fraction of particles striking a droplet F_{us} was dependent on the ratio of separation distance to droplet diameter s/d, and droplet contact angle θ. The variation of F_{us} with these parameters determined from the simulations is plotted in Fig. 3.6.

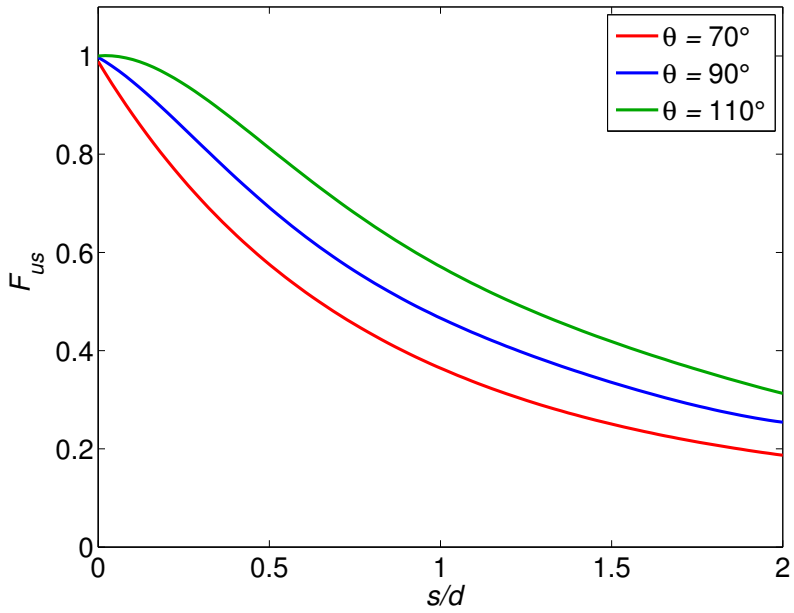


Figure 3.6: Predicted variation of the fraction of particles passing through the upper aperture surface of the unit cell striking a droplet (F_{us}) with s/d and θ in the ballistic limit

To determine the fraction of molecules emitted by the droplet interfaces that strike the other two droplets (F_{id}) and the fraction that return to their interface of origin (F_{ii}), a Monte Carlo scheme similar to the one described above was used. The only difference was that particles representing water molecules were emitted (diffusely) with random location and direction from one of the droplet interfaces, rather than from the triangular aperture at the top of the unit cell. In other respects, this second simulation was identical to the simulation to determine F_{us}. The variations of F_{id} and F_{ii} with s/d, and droplet contact angle θ determined from the simulations are shown in Figs. 3.7 and 3.8.

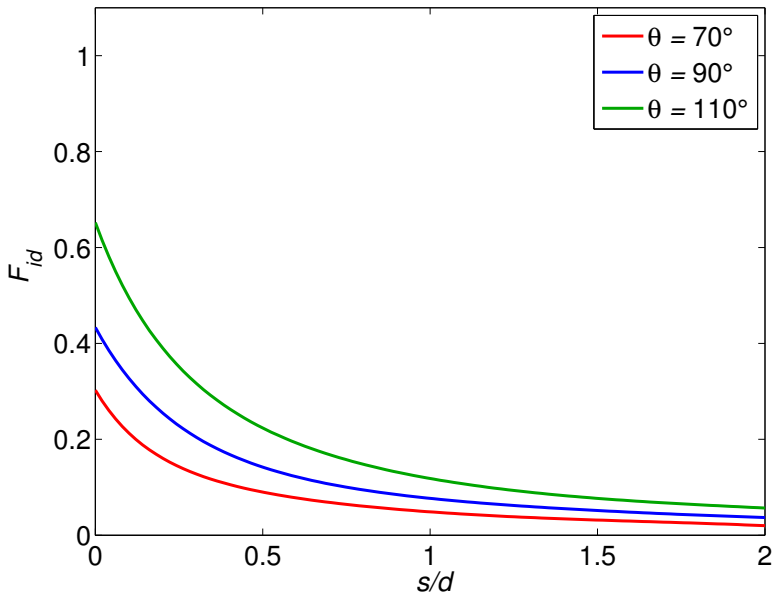


Figure 3.7: Predicted variation of the fraction of particles emitted from a droplet interface that strike a different droplet (F_{id}) with s/d and θ in the ballistic limit

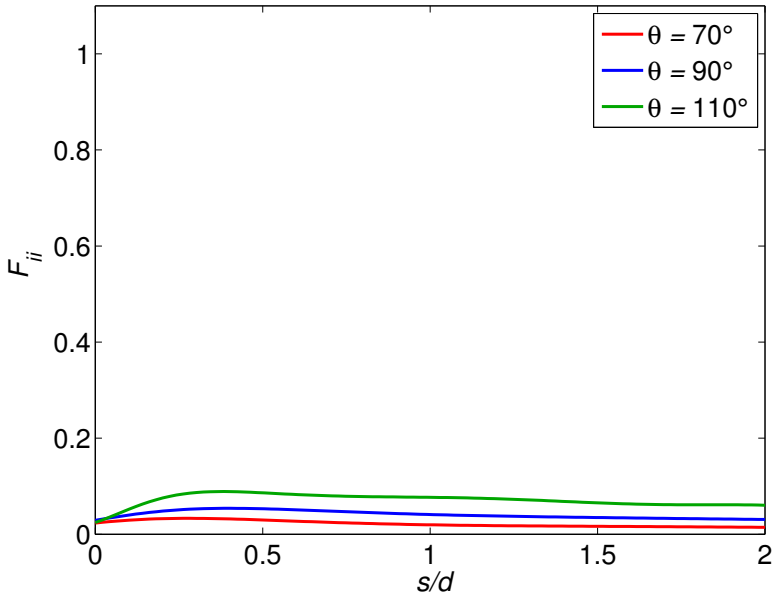


Figure 3.8: Predicted variation of the fraction of particles emitted from a droplet interface that return to the same droplet (F_{ii}) with s/d and θ in the ballistic limit

To facilitate determination of values of F_{us}, F_{id}, and F_{ii} in the dropwise condensation heat transfer analysis described above, the variations of these parameters with s/d and θ were curve-fit. These curve-fit relations are documented in Appendix A.

3.5 Model Results

Variation of heat transfer coefficient with droplet diameter predicted by the model analysis for various combinations of $(T_\infty - T_w) = (T_{sat}(P_\infty) - T_w)$, P_∞, and s/d are shown in Figures 3.9 through 3.13. Figure 3.9 shows results for condensing steam at atmospheric pressure for $s/d = 0.4$ and a wall sub-cooling of 3.0 K with the accommodation coefficient σ taken to be 1.0. As discussed above, we expect the accommodation coefficient to be close to one for a pure water system. The model predictions for the same conditions as Figure 3.9 but with σ set to 0.9 are shown in Figure 3.10. Comparison of the figures indicates that the effect of accommodation coefficient on the predicted heat transfer coefficient is small for values of σ near one.

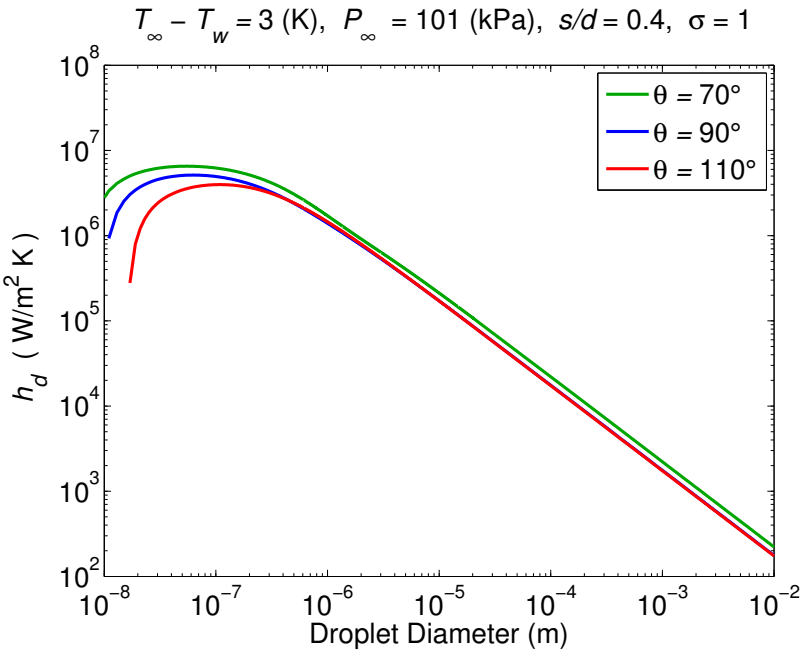


Figure 3.9: Variation of heat transfer coefficient with droplet diameter for $\sigma = 1$, $T_\infty - T_w = T_{sat}(P_\infty) - T_w = 3.0$K, $P_\infty = 101$ kPa, and $s/d = 0.4$.

In Figure 3.9, it can be seen that for diameters above 10 μm, as the diameter of the droplets decreases, the heat transfer coefficient increases, with the heat transfer coefficient being slightly higher for lower contact angle. At a certain diameter (about 1 μm in Figure 3.9) the curves for different contact angle diverge, with lower contact angle droplets showing

higher heat transfer coefficient. For diameters below this threshold, the values of the heat transfer coefficient h_d increase to a peak value and then diminish. This is consistent with the observations of Carey, et al. [61, 18] who found that a droplet growing in supersaturated vapor exhibits a maximum heat transfer rate at a particular diameter during post nucleation growth. This divergence of the heat transfer coefficient curves occurs at a diameter of about 1 μm, which corresponds to a $\mathrm{Kn}_d = 0.07$ for saturated steam at this pressure. As a consequence, diameters below this level are expected to exhibit increasing non-continuum effects. It can be seen in Figure 3.9 that below this size threshold, the variation of the heat transfer coefficients varies slightly with contact angle.

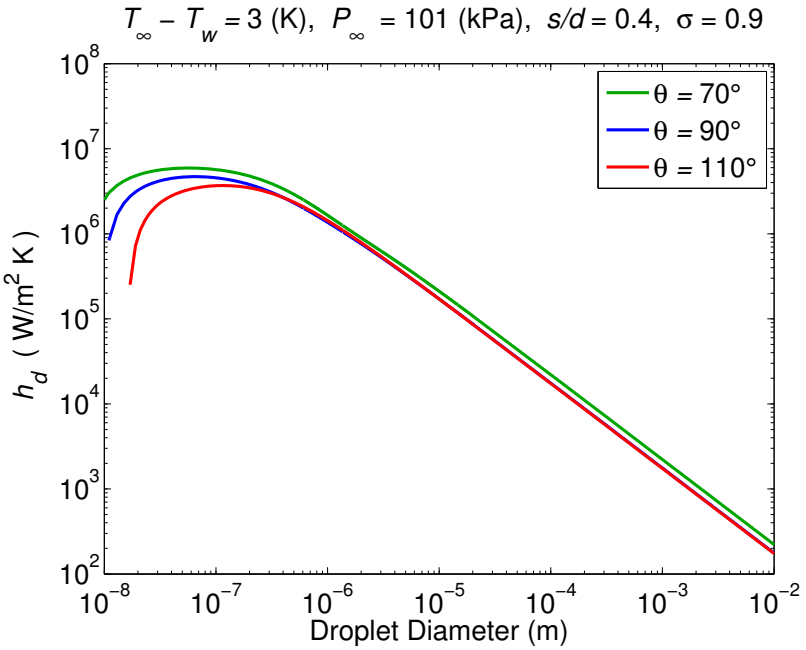


Figure 3.10: Variation of heat transfer coefficient with droplet diameter for $\sigma = 0.9$, $T_\infty - T_w = T_{sat}(P_\infty) - T_w = 3.0$K, $P_\infty = 101$ kPa, and $s/d = 0.4$.

Comparison of Figures 3.9 and 3.11 illustrates the effect of changing system pressure. Decreasing the pressure is seen to shift the onset of non-continuum effects and the peak heat transfer coefficient to larger droplet diameter. This is a direct consequence of the longer mean free path for lower vapor pressure, which results in larger Knudsen number at a given diameter. The transition Knudsen number range ($0.05 < \mathrm{Kn}_d < 20$) is first reached at larger droplet diameter when the mean free path is larger.

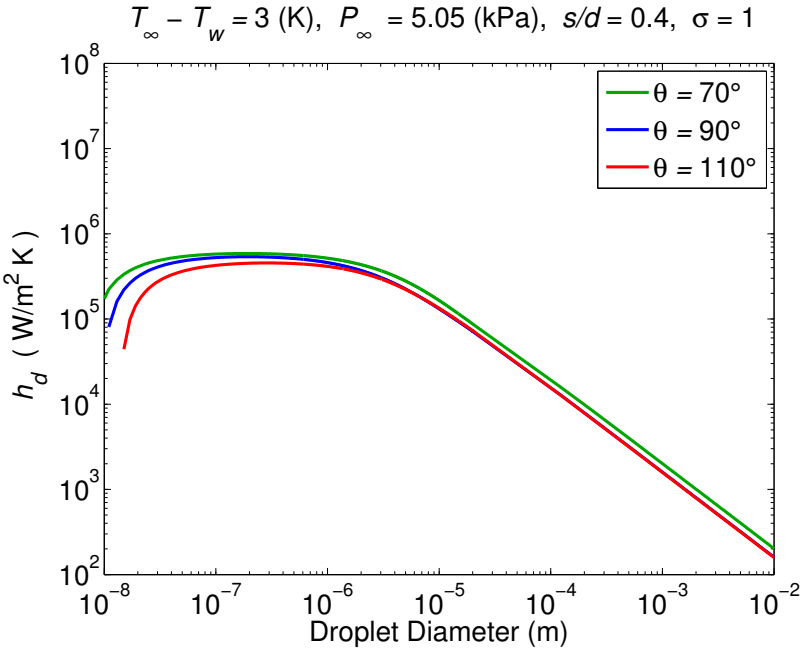


Figure 3.11: Variation of heat transfer coefficient with droplet diameter for $\sigma = 1$, $T_\infty - T_w = T_{sat}(P_\infty) - T_w = 3.0$K, $P_\infty = 5.05$ kPa, and $s/d = 0.4$.

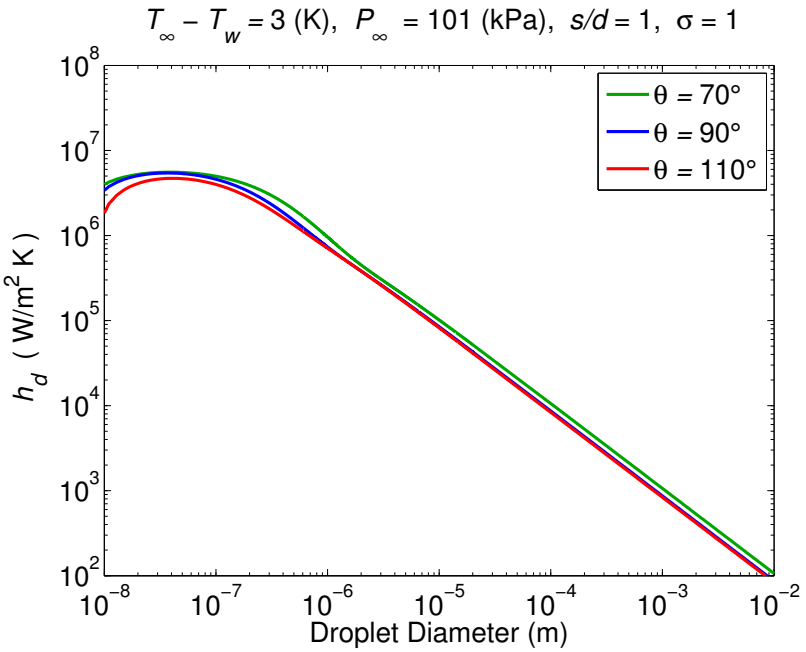


Figure 3.12: Variation of heat transfer coefficient with droplet diameter for $\sigma = 1$, $T_\infty - T_w = T_{sat}(P_\infty) - T_w = 3.0$K, $P_\infty = 101$ kPa, and $s/d = 1$

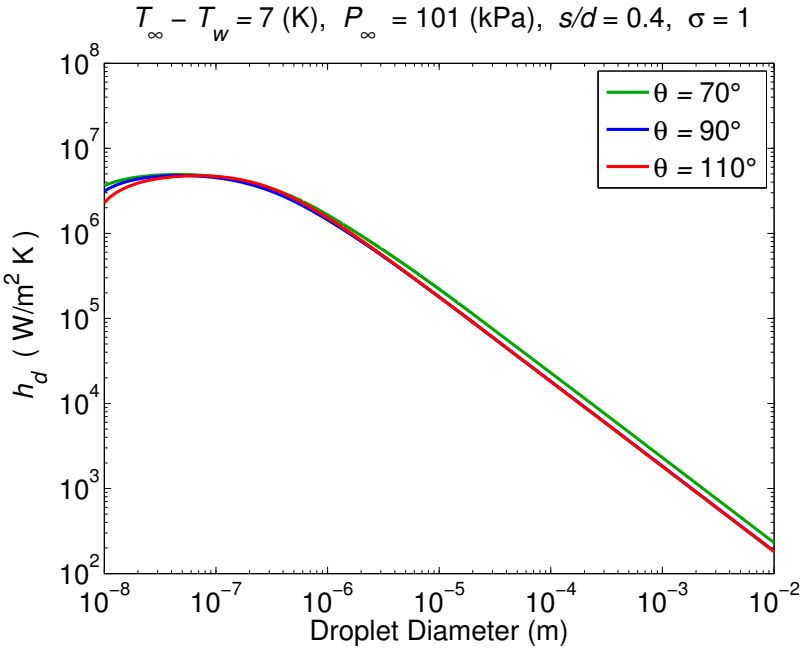


Figure 3.13: Variation of heat transfer coefficient with droplet diameter for $\sigma = 1$, $T_\infty - T_w = T_{sat}(P_\infty) - T_w = 7$K, $P_\infty = 101$ kPa, and $s/d = 0.4$.

Comparison of Figures 3.9 and 3.11 illustrates the effect of changing system pressure. Decreasing the pressure is seen to shift the onset of non-continuum effects and the peak heat transfer coefficient to larger droplet diameter. This is a direct consequence of the longer mean free path for lower vapor pressure, which results in larger Knudsen number at a given diameter. The transition Knudsen number range ($0.05 < \mathrm{Kn}_d < 20$) is first reached at larger droplet diameter when the mean free path is larger.

Comparison of Figures 3.9 and 3.12 indicates that changing the spacing of the droplets from $s/d = 0.4$ to $s/d = 1.0$ has a small effect on the variation of h with droplet diameter. This implies that at least in this range of separation distance, the diameter values where the onset of non-continuum effects occurs and where the peak heat transfer coefficient occurs are not strong functions of s/d.

Figures 3.9 and 3.13 illustrate the effect of surface subcooling on the variation of heat transfer coefficient with diameter. The figures indicate that increasing the surface subcooling changes the h_d versus d curves slightly and has the consequence of reducing the effect of contact angle in the non-continuum range at small diameters.

3.6 Implications of Model Predictions

The model of transport during dropwise condensation in the limit of ultra small droplet diameters developed here is extremely idealized, and it does not account for the spectrum of droplet sizes that exist on a cold surface during a real dropwise condensation process. If the droplet size and spacing input into this model are interpreted as mean values for a real process, an estimate of the resulting heat transfer coefficient can be generated. However, we present this model not as a predictive tool, but as a means of understanding the interplay among droplet conduction heat transfer, interfacial tension effects on thermodynamic equilibrium, noncontinuum transport effects, and interfacial curvature effects on surface tension, which become increasingly important as droplet size decreases.

The model provides insight into how the overall dropwise condensation heat transfer coefficient for a surface will be affected as the mean droplet diameter changes. The model predictions derived here indicate that the trend of increasing heat transfer coefficient with decreasing mean droplet size (observed for droplet diameters larger than 10 μm) eventually breaks down as droplet sizes become smaller and the above effects become increasingly important. The onset of the additional effects responsible for this breakdown occurs for droplet diameters between 0.2 μm and 7 μm for the typical cases considered in this study.

The predictions of this model are useful guidance for researchers aiming to design enhanced dropwise condensation heat transfer surfaces by creating nanostuctured surfaces that have localized hydrophilic areas on an otherwise hydrophobic surface. The model developed here indicates that creating a surface that results in smaller mean droplet sizes will tend to enhance the heat transfer coefficient, up to the point when the peak heat transfer coefficient is attained. The model further indicates that hydrophobic surfaces with contact angles in the range of 90°-110° perform about equally well as long as they do not result in droplet diameters in the 10s of nanometers range. The model also predicts that maximum possible performance of the surface could be attained if patterning of hydrophilic and hydrophobic areas (techniques for creating enhanced surfaces discussed in Chapter 2) resulted in mean droplet diameters no less than in the 100s of nanometers range. While the model developed here incorporates a number of idealizations, this type of information can be useful for designing nano-structured surfaces that aim to provide the highest possible dropwise condensation heat transfer performance.

Chapter 4

DSMC Model on a Single Droplet

4.1 Introduction

The model presented in Chapter 3 laid a starting framework to investigate dropwise condensation heat transfer, but it did have its limitations. The averaging techniques used to determine the transition from continuum to ballistic transport were a source of uncertainty. In an attempt to expand and improve on the model described in Chapter 3, in this chapter we incorporate a Direct Simulation Monte Carlo(DSMC) approach to attempt to model dropwise condensation. The work presented in this chapter discusses our first attempt to use a DSMC model to explore the limitations on dropwise condensation as droplet sizes are reduced. The DSMC model used here, however, simulates water condensation on a single droplet. It does account for the conduction resistance through the droplet and the radius effects on transport induced by deviations in saturation conditions and surface tension from the flat-interface conditions. As in the model presented in Chapter 3, an average droplet size was assumed for the array of droplets to simplify the modeled system. We analyze transport for water droplets condensing on a cold wall in pure steam with average droplet radii ranging between 1μm down to 10nm. The effects of surface wettability are explored by including variations in droplet conduction as droplet contact angle varies for hydrophobic and hydrophilic surfaces.

This chapter is organized as follows:

- The nomenclature for this chapter is presented in Section 4.2.
- The DSMC computational domain for a single droplet is defined in Section 4.3.
- A detailed description of our analysis and approach to the model is described in Section 4.4.
- Heat transfer coefficients derived from this model are discussed in Section 4.5.

- The implications based on the predictions of this model are discussed in Section 4.6

4.2 Nomenclature

A_{eff}	effective area on cold wall surface between r_d and 4_{rd}
A_{s4}	surface area of the outer boundary defined at $r = 4r_d$
A_{sd}	surface area of a droplet segment
d	droplet diameter
ε_{rot}	rotational energy for one molecule
ε_{tr}	translational energy for one molecule
$(E_{w,g})_i$	total energy gained by the droplet in one time-step i
$E_{w,o}$	total energy released by evaporating water particles for N_{steps} from droplet
h_d	dropwise condensation heat transfer coefficient
j_d	molecular vapor flux from droplet interface
k_B	Boltzmann constant
k_l	liquid thermal conductivity
Kn_d	Knudsen number, $=\lambda_\mathrm{m}/d$
m_w	mass of a water molecule
$\dot{n_w}$	rate of vapor particle addition from the ambient
N_{steps}	number of time-steps
N_{mpp}	number of molecules per particle
P_{vd}	droplet vapor pressure
$P_{sat}(T_d)$	flat interface saturation pressure at temperature T_d
r_d	droplet radius
S_f	shape factor for droplet
T_d	droplet interface temperature
T_w	wall temperature
$T_{sat}(P_\infty)$	flat interface saturation temperature at pressure P_∞
$\hat{u}_{lv}$	latent energy per water molecule merged into droplet
Δt	length of a time-step
$\Delta\varepsilon_{w,g}$	net energy gained by droplet from particle absorption
$\Delta\varepsilon_{w,o}$	net energy released from droplet by particle emission
λ_m	mean free path of molecules in vapor
ρ_l	liquid density
σ_{lv}	surface tension at interface temperature T_d
σ	droplet interface accommodation coefficient
θ	droplet contact angle

4.3 Definition of Computational Domain

In this model, again, we consider an array of droplets condensing on a cold wall. To simplify the nonuniform array of droplet sizes that occurs during actual dropwise condensation, as mentioned in Chapter 3, the modeled system here also considers a cluster of droplets with a single mean droplet size arranged in a hexagonal pattern. Different from the idealizations as that presented in Chapter 3, this analysis does not account for interactions with other droplets. The domain from this model instead focuses on a single droplet, where the layout assumed allows the results to be extrapolated to an entire array of droplets due to the symmetry illustrated in Figure 4.1. The dashed outlines encompass the effective area for the computational domain that we seek to model in this analysis.

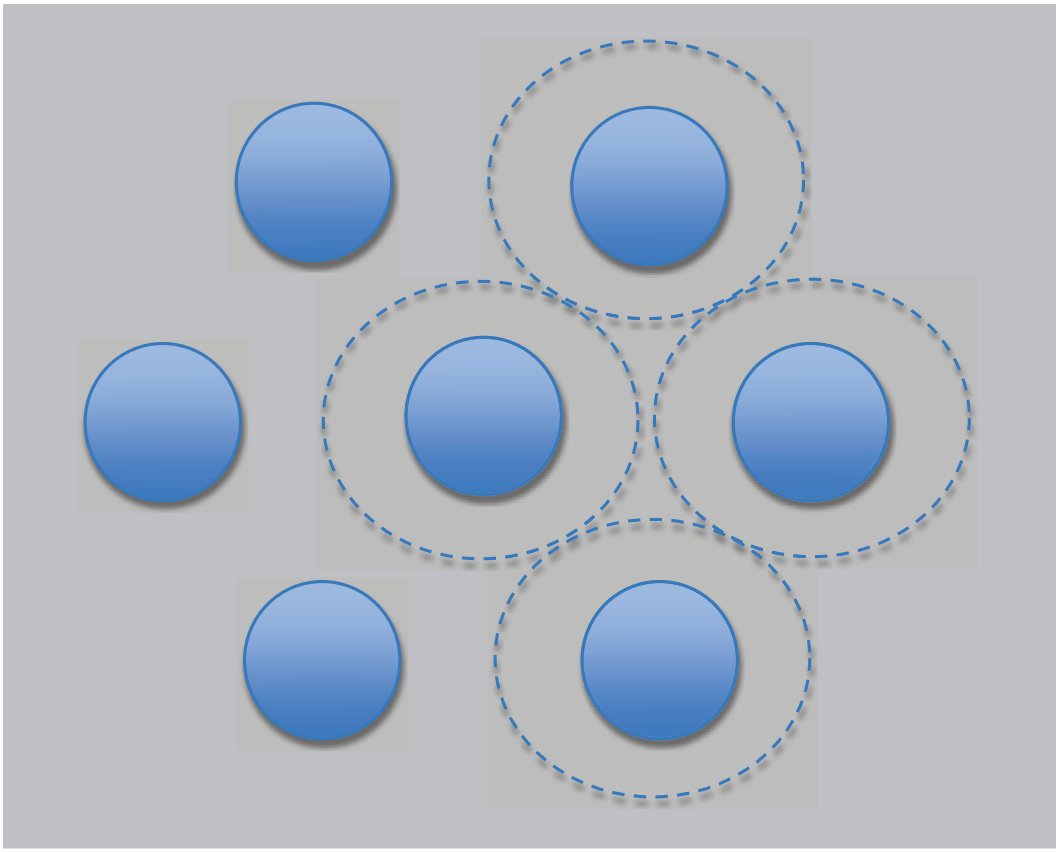


Figure 4.1: Idealized Array of Droplet Size Distribution

As mentioned in the introduction, to expand from the model described in Chapter 3, we were interested in creating this model using a Direct Simulation Monte Carlo(DSMC) approach. Due to the nature of DSMC as described in Chapter 2, steam is modeled as a collection of particles representing bundles of molecules. To compare the results to those

from Chapter 3, comparable conditions were set using a particle simulation model similar to that used by Carey et al. for water microdroplets in argon [61], which was based on the methods established by Bird [22]. The applications and fundamentals behind DSMC were described in Chapter 2, which serve as the basis for the description that follows (if further reference is desired, the reader is encouraged to search the works of Bird and Garcia [22, 50] for detailed theory of DSMC). A general algorithm of the steps to perform the DSMC simlulation for this single droplet domain can be found in Appendix B. The specific domain and boundary conditions used are described below along with key features unique to this model.

4.3.1 The Unit Cell

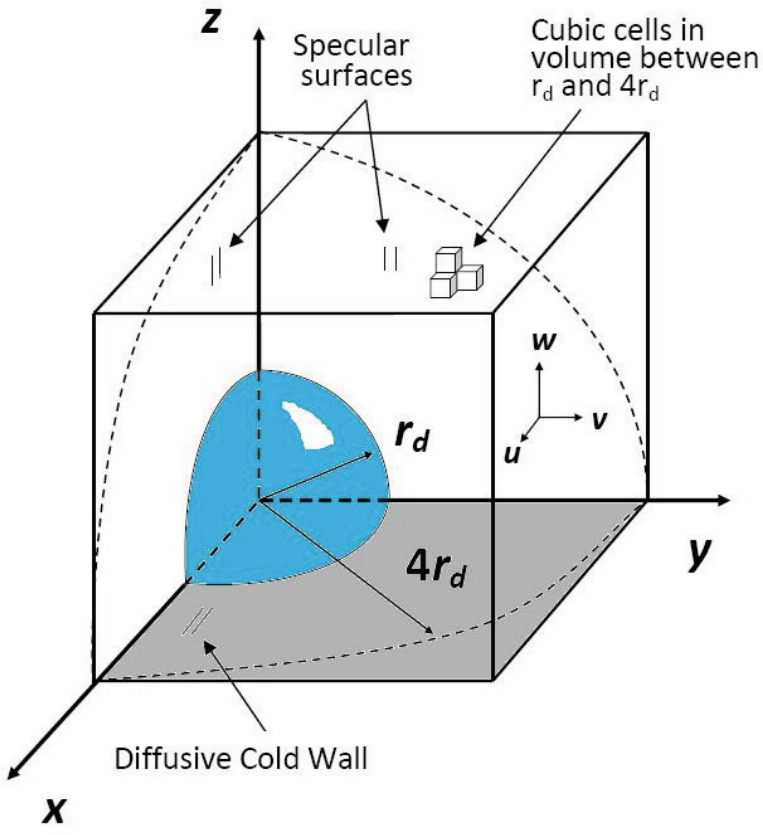


Figure 4.2: Simulation domain and boundary conditions for the single droplet DSMC model

In our DSMC model, droplets are assumed to be uniformly spaced on a cold wall in a hemispherical fashion. Because of the symmetrical nature of uniformly spaced drops, a single

condensing droplet with radius $r = r_d$ was focused on for this analysis with an effective area covered by the perimeter of $r = 4r_d$. To reduce computational time, and due to further symmetry of a droplet, the domain was restricted to one quadrant of a droplet condensing on a wall with its corresponding contact angle θ. Our simulation domain is best illustrated by Figure 4.2.

As mentioned in Chapter 2, DSMC models simulate particle movement throughout the simulation in a realistic manner through a physical space with real physical boundaries. Whenver symmetry is incorporated into the system, proper reflective boundaries have to be established. The DSMC model described here executes following these principals.

In our computations, to properly account for the symmetry of a quarter droplet analysis, the $x = 0$ and $y = 0$ planes were set as specularly reflecting surfaces, thereby providing the same statistical behavior as a full droplet simulation would [61]. Due to the theories of dropwise condensation described in Chapter 2 stating that dropwise condensation tends to occur on preferred nucleation sites, vapor particles were modeled to only be able to condense on a droplet. Therefore, particles interacting with the cold wall surface were diffusely reflected at the point of contact. To handle the transition from continuum transport to non-continuum behavior near the droplet, Carey et al. [61] previously used Langmuir's [70] method of setting up an outer boundary far away from the droplet where continuum transport is known to dominate. Because the local Knudsen number decreases rapidly with distance away from the sphere, Carey set the outer boundary at a radius of $r = 4r_d$. Beyond $4r_d$, the flow is matched to the continuum transport solution for ambient conditions. This method has been widely used for problems of this nature since first implemented by Langmuir, thus the same technique is applied to our proposed DSMC model. The boundary conditions described in this paragraph are summarized by the list below:

- The $x = 0$ and $y = 0$ planes are specularly reflective.
- The cold wall surface, at the $z = 0$ plane, is diffusely reflective.
- Condensation only occurs on the droplet interface.
- Ambient conditions were set at the outer boundary radius of $r = 4r_d$.

4.4 Approach to Modeling

4.4.1 The Particle Simulation Method

As previously stated, in DSMC methods, particles are chosen to represent a fixed number of molecules, N_{mpp}, which fill the simulation domain and move throughout successive time-

steps Δt, just as molecules would. To accurately perform the DSMC method, particles move freely throughout a time-step, and collisions are only executed on a probability basis at the end of each time-step. To execute collisions between particles, the region between r_d and $4r_d$ is filled with cubic cells to sample and collect candidate collision pairs to execute collisions in a probabilistic fashion, as recommended by Baganoff and McDonald [52]. To decouple particle motion from particle collisions, DSMC techniques require the length of the cell to be smaller than the mean free path of the flow being modeled. In accordance with these specifications, we chose cell lengths that varied between 0.5nm up to 50nm, depending on the size of the droplet, which subsequently determined the dimensions of the domain. For the same reasons that cell lengths are required to be smaller than the mean free path of the flow, it is also recommended that time-steps be smaller than the mean free collision time determined from kinetic theory [50]. Depending on the size of the cell, the time-step was chosen to traverse the cell length in approximately two time-steps, resulting in time-steps between 3 ps and 130 ps.

4.4.2 Particle Initiation

To initiate the simulation, a specified number of particles corresponding to the molecular density at ambient conditions was arbitrarily chosen to fill the $r_d - 4r_d$ volume. Cells were loaded with a set number of particles, between 12 and 18, and the corresponding N_{mpp} ratio was determined based on the prescribed conditions. At initiation, particle positions were uniformly distributed throughout the volume, while particle velocities and rotational energies were sampled from appropriate Boltzmann distributions [61].

Once particles were initiated and the simulation moved forward in time, particles were checked for boundary interaction in between time-steps before collisions were executed. The particles could move freely or have one of four interactions with the boundaries:

(1) They could strike the droplet and become absorbed if accommodated, or diffusely reflected if not.

(2) They could traverse the $r = 4r_d$ boundary and leave into the atmosphere.

(3) They could strike the $x = 0$ or $y = 0$ plane and specularly reflect.

(4) They could strike the cold wall and diffusely reflect.

For particles striking a droplet surface, prior experimental and computational work, such as that of Paul and Mills [71, 72] has shown that not all molecules from a surrounding gas will necessarily thermally interact with an interfacial region in the liquid phase. We incorporate this in our model by defining an accommodation coefficient σ as was done in the previous

chapter to determine absorption on a random basis. Specifically, the accommodation was treated by generating a random number $\Re$ between 0 and 1. When the value of $\Re$ was greater than σ, the particle was diffusely reflected from the striking point on the droplet, where its original speed and rotational energy was preserved but oriented in a different direction. If the value was less than a specified σ, the particle energy was considered as absorbed into the droplet, and the particle was removed from the simulation. Typical values for σ vary from system to system, but for systems of pure steam as the one analyzed here, σ is expected to be close to one [7].

To determine the amount of energy absorbed by a thermally interacting particle, it is noted that, on average, the energy of saturated water vapor molecules exceeds that of saturated liquid molecules by their latent energy of vaporization $\hat{u}_{lv}$. At a given droplet temperature T_d, the average energy of a vapor molecule can be defined as $3k_BT_d$, due to its six degrees of freedom from translational and rotational energy [20]. Consequently, for every selected vapor particle absorbed into the droplet with rotational energy ϵ_{rot} and kinetic energy ϵ_{tr}, the net energy delivered to the droplet $\Delta\varepsilon_{w,g}$ was defined as

$$\Delta\varepsilon_{w,g} = N_{mpp}(\varepsilon_{rot} + \varepsilon_{tr} - 3k_BT_d + \hat{u}_{lv}), \tag{4.1}$$

where k_B is the Boltzmann constant. Since this step required prior knowledge of the droplet temperature, for the first 100 iterations the droplet temperature was set to be halfway in between the cold wall temperature and the ambient temperature. For subsequent time-steps, the temperature was determined from an energy balance to be discussed in sections below.

To computationally account for particle interactions with the boundaries other than with the droplet interface, the following actions were taken:

(1) Particles traversing the outer $r = 4r_d$ boundary were considered as going into the ambient and were removed from the simulation.

(2) Particles striking either of the $x = 0$ or $y = 0$ planes, a reversal of the normal component of their velocities accounted for their specular reflections.

(3) Particles striking the cold wall were diffusely reflected and were not considered as condensed. Similar to the diffuse reflection from non-accommodated particles on the droplet, a new random velocity was generated and sampled from an accommodated wall temperature.

4.4.3 Emission From Ambient

After the above boundary checks were performed and appropriate actions were taken within a time-step, new particles were loaded on the $r = 4r_d$ boundary to simulate emission from

the ambient. The rate of vapor particle addition $\dot{n}_w$ from the ambient was determined as

$$\dot{n}_w = N_{mpp} A_{s4} j_\infty, \tag{4.2}$$

where A_{s4} is the surface area outlined by the $r = 4r_d$ surface, which varied depending on droplet size and droplet contact angle θ. The molecular flux from the ambient j_∞ is the flux defined by kinetic theory at ambient pressure P_∞ and ambient temperature $T_\infty = T_{sat}(P_\infty)$ as

$$j_\infty = \frac{1}{4}\left(\frac{P_\infty}{k_B T_\infty}\right)\sqrt{\frac{8k_B T_\infty}{\pi\, m_w}}. \tag{4.3}$$

The total number of particles added in each time-step varied depending on the ambient conditions as well as on the duration of the prescribed time-step.

4.4.4 Emission from Droplet

Emission from the droplet also required prior knowledge or evaluation of the droplet temperature. As stated above, its value is initially unknown, but it is predicted as part of the simulation calculation. As stated by Carey et al. [61], the steady-state energy exchange at the interface must satisfy conservation of energy in the limit of long times and can be expressed as

$$\sum_{i=1}^{N_{steps}} \left(E_{w,g}\right)_i = E_{w,o} + E_{c,o}, \tag{4.4}$$

where $\left(E_{w,g}\right)_i$ is the total energy gained by the droplet through absorbed particles at a time-step i, $E_{w,o}$ is the energy released by evaporating water particles from the droplet as determined by kinetic theory, $E_{c,o}$ is the energy conducted through the droplet and into the cold wall, and N_{steps} is the number of time-steps iterated up to that particular point in time. For an interface temperature T_d, kinetic theory arguments [7] predict the flux of emitted water vapor molecules as

$$j_d = \frac{\sigma}{4}\left(\frac{P_{vd}}{k_B T_d}\right)\sqrt{\frac{8k_B T_d}{\pi m_w}}, \tag{4.5}$$

where the droplet equilibrium vapor pressure P_{vd} accounting for curvature and surface tension effects is corrected from the flat-interface saturation pressure $P_{sat}(T_d)$ by

$$P_{vd} = P_{sat}(T_d)\exp\left(\frac{2\sigma_{lv}}{\rho_l r_d R T_d}\right). \tag{4.6}$$

Accounting for interfacial curvature effects on surface tension σ_{lv} are included as shown in Equation (4.7)

$$\sigma_{lv} = \sigma_{\infty}(T_d)\left[1 + \frac{4\delta_T}{d}\right]^{-1}, \tag{4.7}$$

where $\sigma_{\infty}(T_d)$ is the flat interface surface tension evaluated at the droplet temperature T_d, and δ_T is the Tolman length, taken to be the recommended value of 0.157 nm for water [18]. Consequently, Equation 4.4 can be expressed as

$$\begin{aligned}\sum_{i=1}^{N_{steps}} (E_{w,g})_i = A_{sd} j_d N_{steps} \Delta t (\hat{u}_{lv} + \tfrac{1}{2} k_B T_d) \\ + k_l S_f (T_d - T_w) N_{steps} \Delta t\end{aligned}, \tag{4.8}$$

where the last term represents the conduction through the droplet calculated by using the shape factor S_f. The shape factor for a droplet segment as a function of contact angle θ and radius r_d was obtained from the work of Nijaguana [25] as was done in Chapter 3, and it is shown here as

$$S_f = \frac{\pi\, r_d \sin\theta}{2}\Phi, \tag{4.9}$$

where

$$\Phi = \frac{4\left\{\sum_{n=0}^{\infty} \frac{(4n+3)\,(2n+1)\,(-1)^{2n}(2n!)^2}{(2n+2)^2(2)^{4n}(n!)^4}\right\}}{\left\{\sum_{n=0}^{\infty} \frac{(4n+3)\,(2n!)\,(-1)^n}{(2n+2)\,(2)^{2n}(n!)^2}[\tan(\theta/2)]^{2n+1}\right\}}, \tag{4.10}$$

Since our analysis was only on one-fourth of a symmetrical droplet segment, the shape factor above was multiplied by 0.25. All thermodynamic properties in the equations above were functions of the droplet temperature, so subroutines were developed to perform multiple sixth-order Lagrangian interpolations from values obtained from the National Institute of Standards and Technology (NIST) website [73]. Properties obtained in this fashion had accuracy over the temperature range between 5 °C up to 370 °C.

Due to all the droplet temperature dependencies seen above, it can be seen that Equation (4.8) becomes highly non-linear when attempting to solve for the droplet temperature.

Therefore, a Newton Raphson scheme was used to iteratively solve for the droplet temperature at each time-step. Once a droplet temperature was obtained for a given time-step, the difference between the energy absorbed and the energy conducted away for that time-step determined the number of particles to be emitted. Similar to the amount of energy gained by an absorbed particle as shown in Equation (4.1), each emitted particle reduced the droplet energy by an amount $\Delta\epsilon_{w,o}$

$$\Delta\varepsilon_{w,o} = N_{mpp}(\varepsilon_{rot} + \varepsilon_{tr} - 3k_B T_D + \hat{u}_{lv}\,). \tag{4.11}$$

Again, a particle's rotational energy and velocity were sampled from appropriate Boltzmann distributions. The number of vapor particles emitted n_e at each time-step i was determined from

$$\sum_{j=1}^{n_e} \left(\Delta\varepsilon_{w,o}\right)_j = \left(E_{w,g}\right)_i - k_l S_f (T_d - T_w)\Delta t. \tag{4.12}$$

Essentially, particles were emitted until their energetic sum equated the difference between the energy absorbed and the energy conducted away to the wall in a given time-step.

The last component of each time-step was to execute collisions between particles. The probabilistic selection rule for finding candidate collision pairs was the same as used by Carey et al. [61], which was that recommended by Baganoff and McDonald [52]. The principal behind the selection process is that particle collision pairs are chosen based on their relative velocities. For the execution of molecular collisions, a hard sphere model was employed, which is known to be widely used in direct molecular simulation calculations[61]. In this hard sphere type of model, post-collision relative velocities must be isotropically scattered by choosing velocity vectors at random. Details of the theory behind the selection process and collision model can be found in the work of Baganoff [52] and Bird [22]. The fundamentals of the process are described in Appendix B.

4.4.5 Determining Heat Transfer Coefficients

As the simulation progressed over time, the droplet temperature converged to a stable value. At chosen intervals throughout the simulation, the heat transfer coefficient was determined as

$$h_d = \frac{\sum_{i}^{N_{steps}} \left(E_{w,g}\right)_i}{A_{eff}(T_\infty - T_w)N_{steps}\Delta t}, \tag{4.13}$$

where A_{eff} was the effective area outlined by the $4r_d$ footprint on the cold wall surface.

Simulations were iterated anywhere between 30,000 time-steps up to 400,000 time-steps, depending on the length of each time-step. The idea was to run simulations for a sufficiently long enough time until the value for the heat transfer coefficient stabilized. Figure 4.3 below exemplifies the nature of the simulation for a 400 nm radius droplet under a subcooling of 3 K at one atmosphere. The time-step used in this simulation was for 60 pico-seconds. For these conditions, it can be observed that the simulation is almost fully converged at 20,000 time-steps, which converts to a simulation time of 1.2 μs.

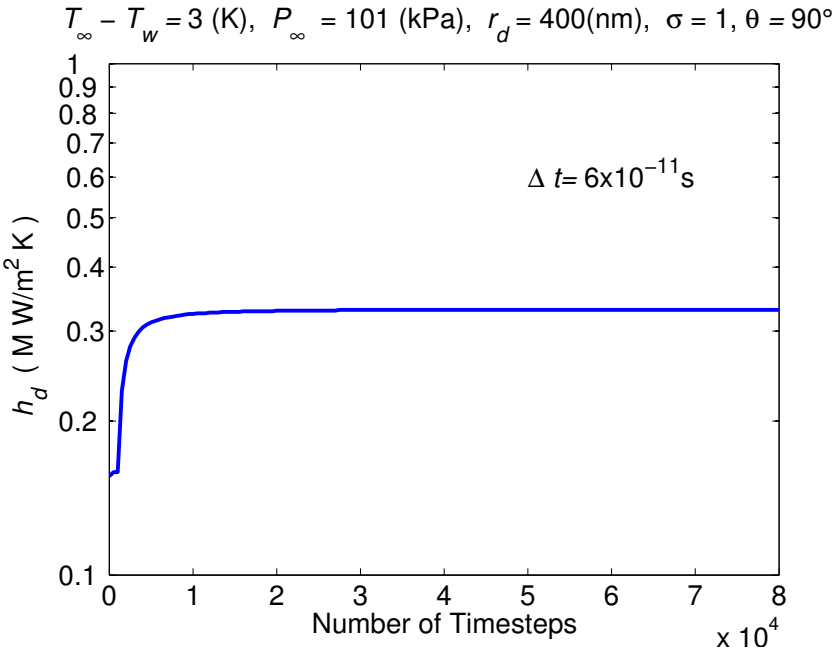


Figure 4.3: Convergence of the heat transfer coefficient calculations for perfect accommodation and $\theta = 90°$.

4.5 Model Results

Variations of the heat transfer coefficient for different prescribed conditions as predicted by the DSMC model are shown in Figures 4.4-4.6. For all cases, the ambient temperature T_∞ corresponded to the flat interface saturation temperature $T_{sat}(P_\infty)$ for steam at pressure P_∞. Each differnt scenario also explores the hydrophobicity and hydrophilicity of surfaces by varying the droplet contact angle. In all instances, trends similar to those seen in the approximation model of Chapter 3 were observed.

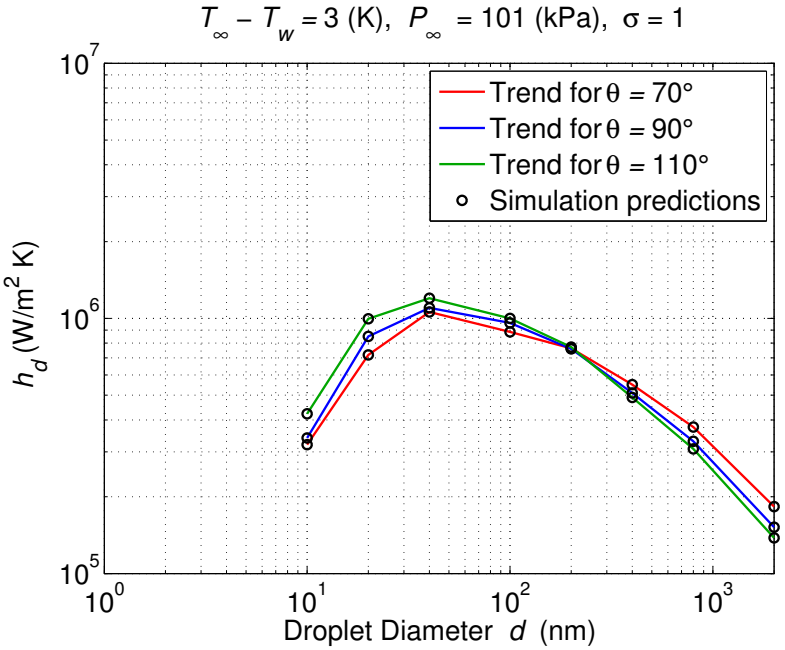


Figure 4.4: Single droplet DSMC predictions of heat transfer coefficients of pure steam for the base case of 3 K subcooling, one atmosphere, and perfect accommodation.

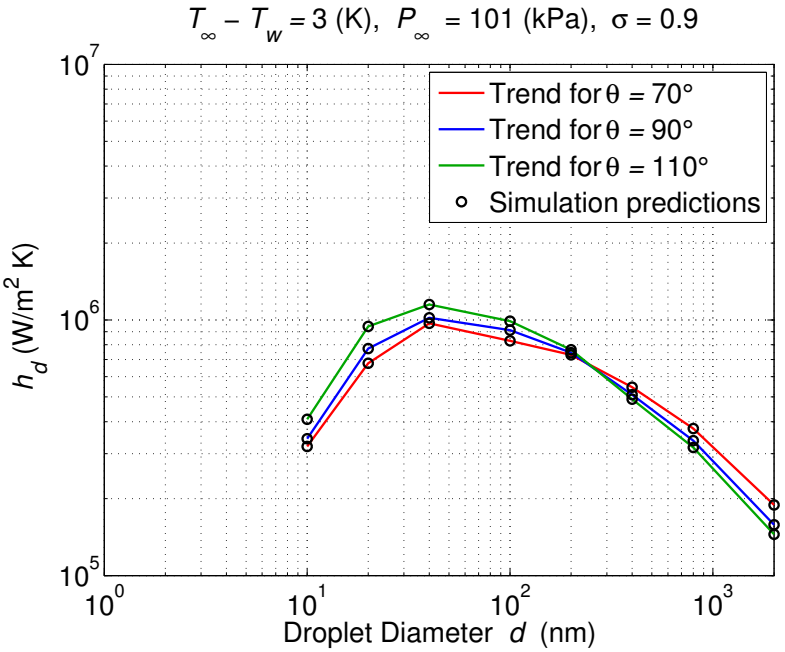


Figure 4.5: Single droplet DSMC predictions of heat transfer coefficients at reduced accomodation.

Due to computational limitations, simulations were only run for diameters up to 2000 nanometers, but significant knowledge is still gained from this size range. For example, as droplet sizes were reduced from 2000 nm diameters, heat transfer coefficients were observed to peaked in the 10s of nanometers range and dropped off as sizes were further reduced. These observations are consistent with those from the approximation model discussed in Chapter 3 and with the observations of Carey, et al. [61, 18], who found that a droplet growing in a supersaturated vapor exhibits a maximum heat transfer rate at a particular diameter during post nucleation growth. This further indicates two main points that the DSMC model showed at these smaller sizes: (1)transition is occurring from continuum transport to free molecular transport, and (2) curvature effects on saturation conditions and surface tension are coming into play, which is what was anticipated.

For standard atmospheric conditions (Figure 4.4) in droplet diameters greater than 1000 nanometers, the expected trends of continuum theory are illustrated; a reduction in droplet size results in smaller conduction thermal resistance through the droplet and thus showing higher heat transfer coefficients. This also agrees with the fact that sizes above 1000 nanometers have a Knudsen numbers near or less than those that identify continuum flow. Similar to the previous observations, for standard atmospheric, this 1000 nanometer droplet size corresponds to an approximate Knudsen number of $\mathrm{Kn}_d = 0.07$.

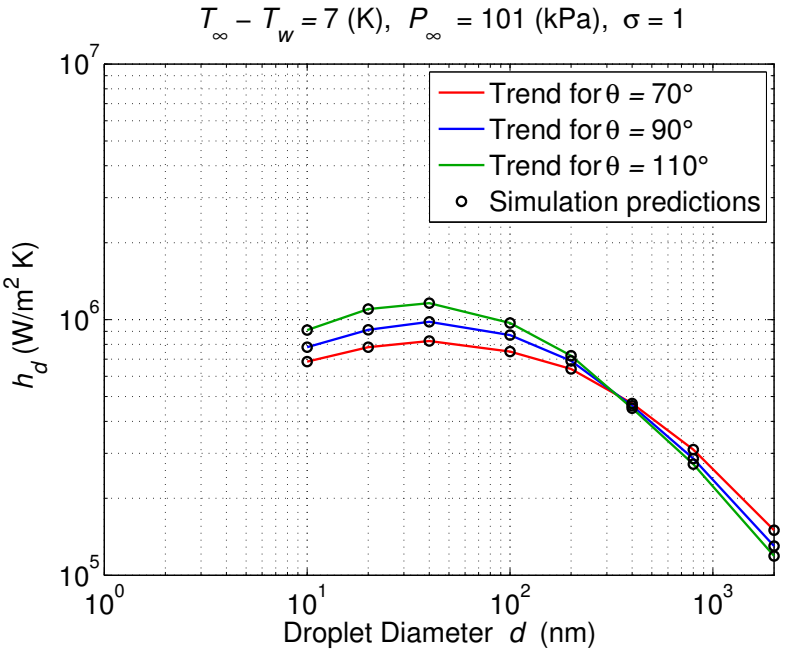


Figure 4.6: Single droplet DSMC predictions of heat transfer coefficients for increased subcooling.

Of particular note were the trends that occurred between changing contact angle at smaller sizes in each scenario. The correlation between heat transfer coefficient and contact angle is reversed from continuum theory, where it is then that those drops with hydrophobic contact angles that show the higher heat transfer coefficients. At these reduced diameters, this implies that the changes in conduction resistance between varying contact angles are not as dominant as in the continuum regime, but it is rather the changes in exposed surface area that promote these differences. That is, droplets with higher contact angle (110° in our case) have greater exposed surface area than those with lower contact angle (70° in our case).

Figure 4.4 shows the parameters chosen as the base case, where a subcooling of 3K is modeled for one atmosphere and perfect accommodation. Figure 4.5 shows the effect of reduced accommodation relative to the base case. Comparison of Figures 4.4 and 4.5 indicates that the variations in heat transfer for accommodation coefficients between 0.9 and 1.0 are nominal with only slight changes being present. Figure 4.6 shows the effect of higher subcooling, with wall temperature at 7 K below the saturation temperature for one atmosphere. Comparing Figures 4.4 and 4.6 reveals that, at smaller diameters, the increased subcooling stabilizes the heat transfer coefficient and prevents it from decreasing as drastically as it does for the lower subcooling of 3 K.

4.6 Implications of Model Predictions

The model in this study explores the limitations that can occur for researchers investigating nanostructured surfaces aiming to enhance dropwise condensation through reduced droplet sizes. The model discussed in this chapter takes a different approach than the model discussed in Chapter 4 as it approaches the problem using DSMC techniques. While the model presented is also idealized to uniform droplet distribution with an effective mean diameter, it still has similar implications that can impact the design of superhydrophobic surfaces aiming to reduce droplet sizes.

Even though this model ignores interactions with neighboring droplets, because of the agreement in observed trends, the implications of this model are similar to those discussed in the previous chapter. That is, due to the idealizations made for this study, this model is not presented as predicting precise heat transfer coefficients, but rather as a conceptual tool to understand the various mechanisms that interplay during dropwise condensation at reduced diameters; conduction through the droplet, interfacial curvature effects on surface tension and saturation conditions, and non-continuum transport effects. To make practical sense out of the idealizations, the droplet sizes modeled can be interpreted as an average size for a cluster of different droplet sizes. If this approach is taken, the implications from this model are that heat transfer coefficients tend to peak for mean droplet diameters of

about 60 nm for the conditions described here. While these sizes are smaller than what has been sustained by experimental research such as that by Dietz et al. [10], it implies that there is room for improvement. Different from the approximation model presented in the previous chapter, this DSMC model shows that as non-continuum effects become significant at reduced mean droplet sizes, it is actually those droplets sustaining higher contact angles that showed higher heat transfer coefficients. However, as long as mean droplet size of the condensate is sustained in the continuum regime, the effects of contact angle and thermal accommodation are shown to be nominal compared to the enhancement that can be attained through the design of surfaces that sustain smaller droplet sizes.

To understand the differences between this model and the others developed for this dissertation, comparisons between different models and their implications are discussed in Chapter 6. For now, this concludes the discussion of the DSMC model developed for a single droplet condensing on a cold wall.

Chapter 5

DSMC Model on a Droplet Cluster

5.1 Introduction

The work presented in this chapter extends the work outlined in Chapters 3 and 4. Similar to the model presented in Chapter 4, this model takes a Direct Simulation Monte Carlo(DSMC) approach. As in Chapter 4, steam is modeled as a collection of particles interacting as bundles of molecules to investigate transport effects on dropwise condensation as mean droplet sizes are reduced within an array of droplets. However, in addition to accounting for the changes in conduction resistance through the droplets, the changes in saturation conditions, and the changes in surface tension due to interface curvature, interaction between droplets is accounted for. This interaction between droplets was observed in the model presented in Chapter 3, but the model here considers it with DSMC techniques. While this model employs the Direct Simulation Monte Carlo method just as the model presented in Chapter 4, the computational domain is modified to be the same as the domain from Chapter 3. Therefore, this DSMC model changes from the previous chapter in that the analysis is on a cross-section encompassing multiple droplets rather than just one. This analysis still focuses on transport for water droplets condensing on a cold wall with average droplet diameters ranging between 2 μm down to 20 nm. Unlike the previous simulations where the focus was on pure steam, the simulations presented in this chapter explore the introduction of non-condensible particles into the system to simulate air. The effects of surface wettability are also explored by including variations in droplet conduction as droplet contact angle varies for hydrophobic and hydrophilic surfaces.

This chapter is organized as follows:

- The nomenclature for this chapter is presented in Section 5.2.
- The computational domain along with boundary conditions are defined in Section 5.3.

- A detailed description of our analysis and approach to the model is described in Section 5.4.
- Heat transfer coefficients predicted in this model are discussed in Section 5.6.
- The implications based on the predictions of this model are discussed in Section 5.7

5.2 Nomenclature

A_{di}	area of three droplet interfaces within the unit cell
A_{us}	area of unit cell footprint on cold wall surface
A_w	area of cold wall surface between droplets in unit cell
A_{sd}	surface area of a droplet segment
d	droplet diameter
ε_{rot}	rotational energy for one molecule
ε_{tr}	translational energy for one molecule
$(E_{w,g})_i$	total energy gained by the droplet in one time-step i
$E_{w,o}$	total energy released by evaporating water particles for N_{steps} from droplet
h_d	dropwise condensation heat transfer coefficient
j_d	molecular vapor flux from droplet interface
k_B	Boltzmann constant
k_l	liquid thermal conductivity
Kn_d	Knudsen number, $=\lambda_\mathrm{m}/d$
m_w	mass of a water molecule
N_A	avogadro's number
$\dot{n_w}$	rate of vapor particle addition from the ambient
N_{steps}	number of time-steps
N_{mpp}	number of molecules per particle
P_{vd}	droplet vapor pressure
$P_{sat}(T_d)$	flat interface saturation pressure at temperature T_d
r_d	droplet radius
S_f	shape factor for droplet
T_d	droplet interface temperature
T_w	wall temperature
$T_{sat}(P_\infty)$	flat interface saturation temperature at pressure P_∞
$\hat{u}_{lv}$	latent energy per water molecule merged into droplet
Δt	length of a time-step
$\Delta\varepsilon_{w,g}$	net energy gained by droplet from particle absorption
$\Delta\varepsilon_{w,o}$	net energy released from droplet by particle emission
λ_m	mean free path of molecules in vapor
ρ_l	liquid density

σ_{lv} surface tension at interface temperature T_d
σ droplet interface accommodation coefficient
θ droplet contact angle

5.3 Definition of Computational Domain

5.3.1 Idealizations in Model

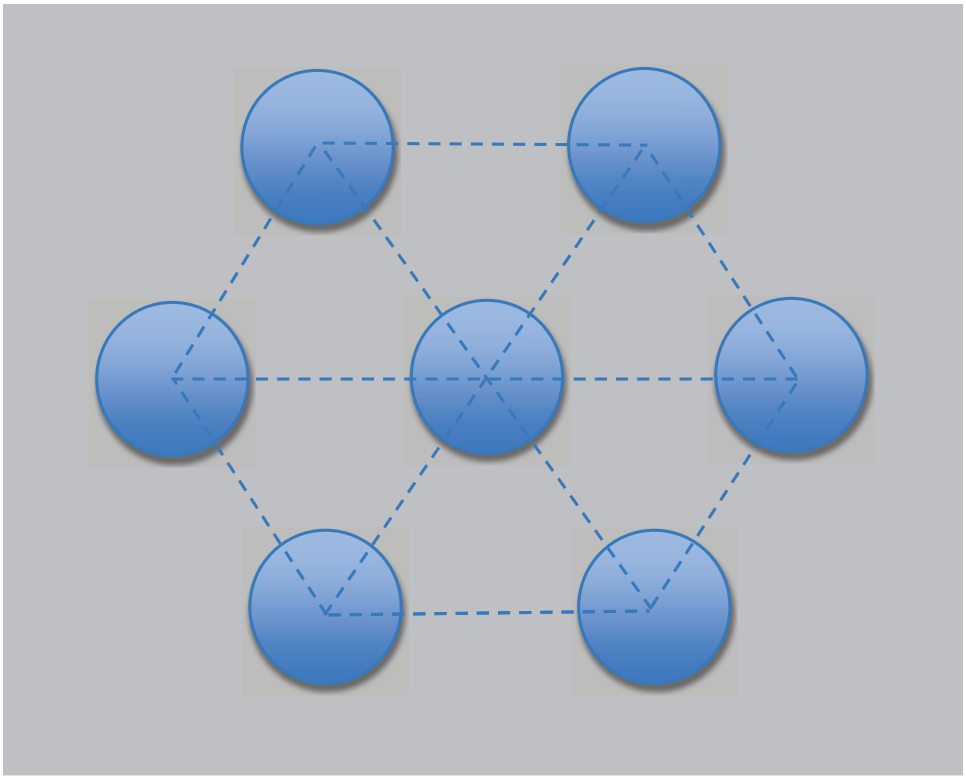


Figure 5.1: Idealized array of droplet size distribution (Figure 3.2 from Chapter 3)

As in Chapters 3 and 4, the non-uniform size pattern of droplets undergoing dropwise condensation is idealized. An array of droplets with mean diameters and equidistant from one another is considered here to explore how the transport mechanisms of dropwise condensation change with droplet size at very small droplet diameters. The same hexagonal pattern assumed in Chapters 3 and 4 was assumed for this model, as shown in Figure 5.1. The triangular cross-sections outlined by the dashed lines are cells of symmetry that allow this analysis to focus on a single triangular cross-section. Figure 5.2 shows the equilateral triangular unit cell of this array, where the droplets are modeled as spherical segments with a

liquid-vapor interface that meets the smooth solid surface (cold wall) at a contact angle θ. Figures 3.2 and 3.3 were inserted here as Figures 5.1 and 5.2 for emphasis and for the readers convenience. Here we take s as the radial separation between the interfaces of adjacent droplets in the center-plane of the droplets, d is the droplet diameter, P and T represent pressure and temperature respectively, and the subscript ∞ denotes the ambient.

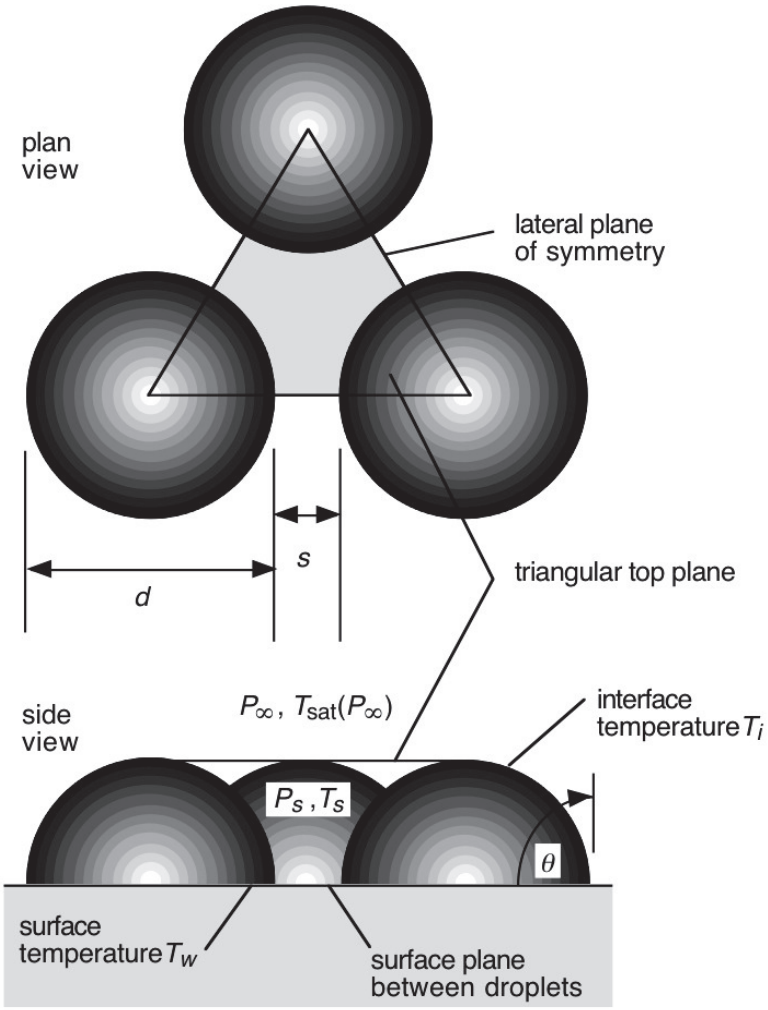


Figure 5.2: Model unit-cell system (Figure 3.3 from Chapter 3)

Based on the above description, the computational domain then becomes the space in between the droplets within the triangular-prism. As stated in the introduction, in this chapter

we employ the DSMC method to analyze the triangular prism computational domain rather than using the approximation methods introduced in Chapter 3 or the single droplet system of Chapter 4. Due to the nature of DSMC techniques, simulations can be time intensive, especially as the computational domains get larger. Therefore, focusing on one symmetrical triangular cross-section encompassing three droplets is further justified.

Various algorithms exist to simulate gases through DSMC models. The particle simulation model used here is similar to that used in the model presented in Chapter 4. This is based on the model Carey et al. used for water microdroplets in argon [61], which was based on the methods established by Bird [22]. The details and theory behind DSMC and its applications are thoroughly described in the works of Bird and Garcia [22, 50]. A general algorithm of the steps used to perform the DSMC simulation for the triangular domain described in this chapter can be found the Appendix C. Key features unique to this model and the details of the boundary conditions used for this domain are described below.

5.3.2 System Boundary Conditions

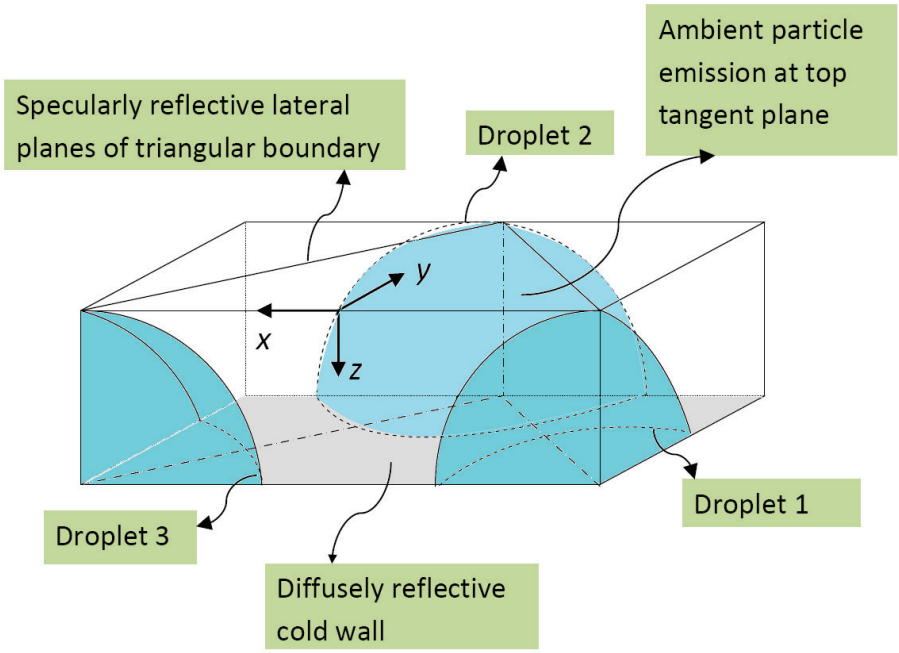


Figure 5.3: Simulation domain and boundary conditions for triangular-prism DSMC model

The premise of DSMC simulations is that molecular dynamics of a gas can be modeled on a statistical basis by simulating particles representing groups of molecules. To computa-

tionally account for the symmetry of using a single triangular cross-section, all three sides of the triangle, labeled as lateral planes of symmetry, were treated as specularly reflective surfaces. This is in accordance with guidelines for DSMC described by Garica and Bird [50, 22] to attain the same statistical behavior as a simulation on a bigger array would [61]. As mentioned in Chapters 2 and 3 in the theory for dropwise condensation, we assumed that particles could only condense on a droplet surface. Therefore, particles interacting with the cold wall surface in the space between the droplets were diffusely reflected.

Reiterating the expected behavior of flow, as droplet sizes are reduced to diameters comparable to the mean free path, transport dynamics transition from continuum to free molecular flow, as characterized by the Knudsen number. In the previous chapter, to handle the transition from continuum transport in the ambient to non-continuum behavior near the droplets, the model presented in Chapter 4 is forced to use a method previously used by Langmuir [70]. In his work, Langmuir considered an outer boundary far away from the droplet to define a point where continuum transport is known to dominate. In this aspect, the triangular-prism computational domain chosen here results to be advantageous. This model directly takes the triangular top plane as a surface where a continuum flux will be incident at prescribed ambient conditions. The triangular top plane thereby serves as the area where the flux of particles coming in from the ambient was known, and any influx of ambient molecules modeled were initiated on this triangular surface.

The computational domain and boundary conditions discussed above are illustrated in Figure 5.3. The following summarizes the boundary conditions specific to the triangular-prism computational domain.

- The lateral planes of the triangular boundary are taken to be specularly reflective, accounting for symmetry of neighboring unit-cells.
- The cold wall surface at the $z = r(1 - \cos\theta)$
- Condensation only occurs on one of the three droplet surfaces within the triangular domain and not on the cold wall space in between the droplets.
- Ambient conditions are set to be known on the top tangent plane where $z = 0$.

5.4 Approach to Modeling

5.4.1 Standard Features and General Overview of DSMC

The DSMC (Direct Simulation Monte Carlo) techniques used here are essentially the same as those used in Chapter 4, but they are summarized here in the context of this new domain. These DSMC techniques are common numerical methods ideal for modeling rarefied

gas flows where the mean free path of the gas is of the same order of magnitude as a physical length scale in the domain being analyzed. Here again, the DSMC method used models flows through simulation particles, which represent a group of actual molecules of the fluid being simulated. The particle simulators are moved through a simulated physical space in time-steps, where the dynamics of the actual gas molecules become statistically accurate over time. This physical space becomes the triangular boundary that we have defined in the section above.

To attain statistically accurate results using the DSMC techniques mentioned above, certain constraints guide how the simulation progresses and its overall general structure. Here, particles are chosen to represent a fixed number of molecules, N_{mpp}, which fill the triangular simulation domain and move throughout successive time-steps Δt, just as molecules would. Furthermore, the simulation domain is divided into cells, which are used in finding particle-pairs to execute collisions, and the requirement that the characteristic cell length be smaller than the mean free path of the fluid being modeled is satisfied here again. For the simulations in this model, cubic cell lengths varied between 10 nm and 60 nm, depending on the size of the droplet, which subsequently determined the dimensions of the domain. The number of molecules represented by each particle simluator, defined as N_{mpp} in here, is based on the molecular density of the simulations and the number of particles desired per unit volume.

According to Garcia [50], the accuracy of the model is compromised as the number of particles per cell is increased, and it is therefore recommended that there should be between 10-20 particles per cell. Consequently, N_{mpp} was then determined based on the atmospheric conditions of the of simulation along with limiting the domain to anywhere between 10-15 particles per cell. Based on these restrictions, the value of N_{mpp} used here was between 1.6 for simulations of smaller droplets, up to 170 for simulations with larger droplet and hence larger computaional domain. Lastly, as in Chapter 4, the requirement for time-steps to be less than the mean free collison time determined from kinetic theory is satisfied here. Depending on the size of the cell, the timestep was chosen to traverse the cell length in approximately one or two time-steps, resulting in time-steps between 10 and 130 ps [50].

5.4.2 Particle Initiation

To initiate the simulation, a specified number of particles corresponding to the molecular density at ambient conditions is arbitrarily chosen to fill the volumetric space in between the droplets. Cells are loaded with an assigned number of particles, with simulations for larger droplets having 10 particles per cell, and those with smaller droplet sizes (and therefore smaller domains) having 15 particles per cell. As stated above, the corresponding N_{mpp} ratio is determined based on these prescribed conditions. These particles are distributed uniformly throughout the volumetric space in between the cells, while particle velocities and

rotational energies were sampled from appropriate Boltzmann distributions [61]. Appendix A describes the process to randomly choose positions, and Appendix B discusses the process to randomly choose velocities sampled from a Boltzmann distribution.

The overall sequence of particle dynamics is the following, with steps (2s) through (4s) being repeated for each time-step as the simulation cycles through the total number of steps:

(1s) Simulation is initiated and computational domain is preloaded with particles

(2s) Particles are advanced Δt seconds and potential interactions with boundaries are checked

(3s) Particles are emitted from ambient

(4s) Particles are emitted from the droplet surface

5.4.3 Particle Progression

Once particles are initiated in the volumetric space between the droplets according to the atmospheric conditions, the simulation can move forward in time, progressing throughout each time-step and cycling through steps (2s) through (4s) for subsequent time-steps. In each time-step, as particles are advanced in the direction of their velocity, particles are checked for boundary interaction after their movement is completed. Based on the boundary conditions described in Section 5.3.2, particles can move freely or have one of four different boundary interactions:

(1) They could strike the droplet and become absorbed if accommodated, or diffusely reflect if not.

(2) They could traverse the z=0 plane and leave into the atmosphere.

(3) They could strike one of the lateral sides of the triangular prism and specularly reflect.

(4) They could strike the cold wall surface and diffusely reflect.

For particles striking a boundary, energy exchange between the particles and the boundary is not always 100% efficient. Just as in Chapter 4, this manifests itself in the form of an accommodation coefficient. For particles specifically striking a droplet surface, prior experimental and computational work from Paul and Mills [71, 72] has shown that not all vapor molecules from a surrounding gas can thermally interact with an interfacial region in the liquid phase. Essentially, this means that not all vapor molecules striking a droplet surface will necessarily become absorbed into the droplet. We incorporate this in our model by defining a thermal accommodation coefficient σ to determine absorption on a random basis where its value can vary anywhere between 0 and 1. If σ is multiplied by 100, it can

be thought of as the percentage of molecules interacting with a droplet boundary that are in fact absorbed into the droplet. Typical values for σ vary from system to system, but for the types of systems analyzed here, σ is expected to be close to one [7]. Consequently, simulations were varied for σ values between 0.9 and 1 throughout the work presented in this chapter. Such variations of σ are implemented in the model by generating a random number from a uniform distribution, $\Re$, between 0 and 1 whenever a particle strikes a droplet boundary. If the value of $\Re$ is greater than σ, the particle is diffusely reflected from the striking point on the droplet, where its original speed and rotational energy are preserved but oriented in a different direction. If the value of the generated $\Re$ is less than the value of σ, the particle is considered absorbed into the droplet, and the particle is removed from the simulation.

When particles become absorbed into the droplet, it is necessary to determine and record the amount of energy absorbed into a droplet by the thermally interacting particle. To account for this properly, it is noted that, on average, the energy of saturated water molecules exceeds that of saturated liquid molecules by their latent heat of vaporization, $\hat{u}_{lv}$. Furthermore, at a given droplet temperature T_d, molecular theory dictates that the average energy of any molecule is $\frac{1}{2}k_BT_d$ for every degree of freedom the molecule possesses, where k_B is the Boltzmann constant. Due to its six degrees of freedom from translational and rotational energy, a water vapor molecule has an average energy of $3k_BT_d$ [20]. Therefore, for accommodated vapor particle into the droplet with rotational energy ϵ_{rot} and kinetic energy ϵ_{tr}, the net energy delivered to the droplet $\Delta\varepsilon_{w,g}$ was defined as

$$\Delta\varepsilon_{w,g} = N_{mpp}(\varepsilon_{rot} + \varepsilon_{tr} + (\hat{u}_{lv} - 3k_BT_d\,). \tag{5.1}$$

Here again, for the first 100 iterations, the droplet temperature was set to be halfway in between the cold wall temperature and the ambient temperature. For subsequent time-steps, the temperature is determined from an energy balance to be discussed in sections below.

In this chapter we further consider the addition of a non-condensible gas into the system. Particles that are selected to represent a group of non-condensible molecules do not deliver their latent energy to the droplet, and their effects on the energy of the droplet when interacting with its boundary is less pronounced than the vapor molecules. Instead, these particles are simply diffusely reflected, where their emitted translational and rotational energy are sampled from a Boltzmann distribution at the droplet temperature T_d. This in effect mimics the actual molecular dynamics that occurs between a non-condensible gas and a water droplet. This implies that, on average, if non-condensible particles striking a droplet are initially sampled from the ambient temperature T_∞, which should be higher than the droplet temperature, they should still have a net energy transfer to the droplet. This net energy transfer is handled by considering the translational and rotational energy of all striking

particles as absorbed, but then removing the translational and rotational energies of all the non-condensible particles as they are all emitted later (step (4s) in Section 5.4.2) but sampled from T_d. Since the droplet surface temperature T_d is not solved for until a subsequent stage, the striking non-condensible particles are temporarily held at the droplet surface until the droplet surface temperature is solved for, and it is then that they are emitted from the droplet surface. As the details of the solution to the droplet temperature are explained further below, this reason to hold the non-condensible particles at the droplet surface until T_d is solved for becomes obvious. Again, it's important to distinguish from the dynamics of the condensed vapor particles. The main difference is that the net energy delivered by the non-condensible particles will be much less as no latent energy is deposited on the droplet.

Particles traveling in the direction outside of the computational domain and traversing the $z = 0$ boundary are considered as going into the ambient and are removed from the simulation as soon as the $z = 0$ plane is crossed. For particles striking either of the lateral planes of the triangular prism, their specular reflections were simulated by reversing the component of their velocities that is normal to the stricken plane. Particles striking the cold wall were not considered as condensed and were diffusely reflected. The diffuse reflection was done by generating a new random velocity sampled from a Boltzmann distribution at the wall temperature T_w.

5.4.4 Emission from ambient

After particles advanced in time and the above checks were performed with the appropriate actions being taken, molecules from the ambient coming into the computational domain were considered. As stated above, the molecular flux from the ambient was handled by introducing particles at the $z = 0$ plane with energies and velocities sampled from a Boltzmann distribution at the ambient conditions. The vapor particle flux from the ambient $\dot{n}_w$ is therefore determined as

$$\dot{n}_w = X_{w,\infty} N_{mpp} A_{ts} j_{w,\infty}, \tag{5.2}$$

where $X_{w,\infty}$ is the concentration of water in the ambient, and A_{ts} is the triangular surface area outlined by the droplet peaks at the $z = 0$ plane, which varied depending on droplet sizes, contact angle θ, and droplet separation distance s. The molecular flux from the ambient $j_{w,\infty}$ is the flux defined by kinetic theory at ambient pressure P_∞ and ambient temperature $T_\infty = T_{sat}(P_\infty)$ as

$$j_{w,\infty} = \frac{1}{4}\left(\frac{P_\infty}{k_B T_\infty}\right)\sqrt{\frac{8 k_B T_\infty}{\pi\, m_w}}. \tag{5.3}$$

Similarly, the rate of non-condensible particles addition $\dot{n}_{nc}$ is determined as

$$\dot{n}_{nc} = X_{nc,\infty} N_{mpp} A_{ts} j_{\infty w}, \tag{5.4}$$

where $X_{nc,\infty}$ is the concentration of non-condensible particles in the ambient. The molecular flux from the ambient $j_{nc,\infty}$ is the flux defined by kinetic theory at ambient pressure P_∞ and ambient temperature $T_\infty = T_{sat}(P_\infty)$ as

$$j_{nc,\infty} = \frac{1}{4}\left(\frac{P_\infty}{k_B T_\infty}\right)\sqrt{\frac{8 k_B T_\infty}{\pi\, m_{nc}}}. \tag{5.5}$$

The total number of particles added from each species varied depending on the ambient conditions and N_{mpp}, as well as on the duration of the prescribed time-step.

5.4.5 Emission From Droplet

Following the emission from the ambient is the emission from the droplet. However, emission from the droplet requires knowledge of the droplet surface temperature. It's value is initially unknown, but it is determined as part of the simulation calculation. As stated by Carey et al. [61], steady-state energy exchange at the interface must satisfy conservation of energy in the limit of long times

$$\sum_{i=1}^{N_{steps}} \left(E_{gain}\right)_i = E_{w,o} + E_{nc,o} + E_{cond,o}, \tag{5.6}$$

where $\left(E_{gain}\right)_i$ is the energy gained by the droplets as particles (either condensible or non-condensible) collide with the droplet interface at a time-step i, giving off their respective energy. $E_{w,o}$ is the energy released by evaporating water particles at temperature T_d, $E_{nc,o}$ is the energy released by the reflected non-condensible particles at temperature T_d, $E_{cond,o}$ is the energy conducted through the droplet into the cold wall, and N_{steps} is the number of time-steps iterated up to that particular point in time. $E_{w,o}$ is calculated as

$$E_{w,o} = A_{sd} j_d N_{steps} \Delta t (\hat{u}_{lv} + \frac{1}{2} k_B T_d), \tag{5.7}$$

where A_{sd} represents the surface area of the three droplet segments within the computational domain, and j_d is the vapor molecular flux emitted from the droplet at its corresponding temperature and vapor pressure. From kinetic theory, the vapor molecular flux j_d is calculated as

$$j_d = \frac{\sigma}{4}\left(\frac{P_{vd}}{k_B T_d}\right)\sqrt{\frac{8 k_B T_d}{\pi m_w}}, \tag{5.8}$$

where P_{vd} is the droplet equilibrium vapor pressure accounting for curvature and surface tension effects corrected from the flat-interface saturation pressure $P_{sat}(T_d)$ as

$$P_{vd} = P_{sat}(T_d) \exp\left(\frac{2\sigma_{lv}}{\rho_l r_d R T_d}\right). \tag{5.9}$$

The interfacial curvature effects on surface tension σ_{lv} are included as shown in Equation (5.10)

$$\sigma_{lv} = \sigma_\infty(T_d)\left[1 + \frac{4\delta_T}{d}\right]^{-1}, \tag{5.10}$$

where $\sigma_\infty(T_d)$ is the flat interface surface tension evaluated at the droplet temperature T_d, and δ_T is the Tolman length, taken to be the recommended value of 0.157nm for water [18]. Substituting in equations (5.7) through(5.10) into (5.6) leads to

$$\begin{aligned}\sum_{i}^{N_{steps}} (E_{gain})_{i=1} = A_{sd} j_d N_{steps} \Delta t (\hat{u}_{lv} + \tfrac{1}{2} k_B T_d) \\ +3 k_B T_d N_{mpp} \sum_{i}^{N_{steps}} (N_{non})_i + k_l S_f (T_d - T_w) N_{steps} \Delta t\end{aligned}, \tag{5.11}$$

where S_f in the last term represents the conduction shape factor for a droplet segment. This conduction shape factor as a function of contact angle θ and radius r_d was obtained from the the work of Nijaguana [25] as

$$S_f = \frac{\pi\, r_d \sin\theta}{2}\Phi, \tag{5.12}$$

where

$$\Phi = \frac{4\left\{\sum_{n=0}^{\infty} \frac{(4n+3)(2n+1)(-1)^{2n}(2n!)^2}{(2n+2)^2(2)^{4n}(n!)^4}\right\}}{\left\{\sum_{n=0}^{\infty} \frac{(4n+3)(2n!)(-1)^n}{(2n+2)(2)^{2n}(n!)^2}\left[\tan(\theta/2)\right]^{2n+1}\right\}}, \tag{5.13}$$

Since our analysis is on three 60° droplet segments, this only totals up to 180° of a whole droplet segment, therefore the shape factor above was multiplied by 0.5 throughout the simulation.

All thermodynamic properties in the foregoing equations were functions of either the droplet or the ambient temperature. Subroutines were therefore developed to perform multiple sixth-order Lagrangian interpolations from values obtained from the National Institute of Standards and Technology (NIST) website [73]. Properties obtained in this fashion had accuracy over a temperature range between 5 °C up to 370 °C, which broadly covered the temperatures used for the investigations of this chapter.

To finally solve for the temperature, Equation (5.11) is used along with the aforementioned definitions of all the parameters in it. The left hand side of the equation is simply a tally of the energy collected from all the condensible particles absorbed into the droplets and from the net energy delivered by the interacting non-condensible particles. This tally of energy is collected throughout the simulation at each time-step as described in section 5.4.3. In analyzing the right hand side, it can be observed from the first and second terms that because of the dependency on T_d, Equation (5.11) becomes highly nonlinear. Due to the molecular fluxes and droplet vapor pressure, solving for T_d cannot be done in a direct fashion. To solve for T_d, a Newton Raphson scheme was employed at each time-step which iteratively solved for the temperature through the use of Equation (5.11).

Once T_d is obtained for a given time-step, particle emission from the droplet surface followed. The initial step is to emit all the non-condensible particles that were recorded to strike the droplet surface during the "particle progression" phase discussed in section 5.4.3. As mentioned in the same section, these particles were held off at the droplet surface until reaching this point in the simulation, where T_d has been solved for at the given time-step. At this point in the time-step, all non-condensible particles are finally emitted, where their translational and rotational energies are sampled from a Boltzmann distribution at the droplet temperature T_d. Each non-condensible particle reduces the droplet's energy by an amount $\Delta\varepsilon_{nc,o}$

$$\Delta\varepsilon_{nc,o} = N_{mpp}(\varepsilon_{rot} + \varepsilon_{tr}). \tag{5.14}$$

The number of vapor particles emitted is then dependent on the total amount of energy needed to be released in order to satisfy the energy balance for the droplet surface. In that energy balance, all energy gained by the droplets for that time-step should be equivalent to the sum of the energies released through the non-condensible particles, the vapor particles, and the energy conducted through the droplet and into the cold wall. Following the same logic behind Equation (5.1), each emitted vapor particle reduces the droplet energy by an amount $\Delta\varepsilon_{w,o}$

$$\Delta\varepsilon_{w,o} = N_{mpp}(\varepsilon_{rot} + \varepsilon_{tr} + \hat{u}_{lv} - 3k_B T_d\,). \tag{5.15}$$

Therefore, for a known number of non-condensible particles emitted (equivalent to the amount absorbed) n_{nce}, the number of vapor particles to emit n_{we} at a specific time-step i is determined from

$$\sum_{j=1}^{n_{we}} (\Delta\varepsilon_{w,o})_j = (E_{w,g})_i - \sum_{k=1}^{n_{nce}} (\Delta\varepsilon_{nc,o})_k - k_l S_f (T_d - T_w)\Delta t. \tag{5.16}$$

Vapor particles are continuously emitted until a value of n_{we} is reached where the left-hand side of Equation (5.16) equates the right-hand side.

If zero non-condensible particles were introduced into the system, then the simulation proceeded in the same fashion except that Equation (5.16) changes by eliminating the second factor in the right-hand side and it becomes

$$\sum_{j=1}^{n_e} (\Delta\varepsilon_{w,o})_j = (E_{w,g})_i - k_l S_f (T_d - T_w)\Delta t. \tag{5.17}$$

The last stage of the time-step was to execute collisions between particles in the volumetric space between the droplets. The model used for determining candidate collision pairs and executing collisions is the same as described in Section drpemisCh3. For details of the theory behind the selection process and collision models, the reader is referred to the work of Baganoff [52] and Bird [22]. The fundamentals of the process are described in Appendix B.

5.5 Determining Heat Transfer Coefficients

The simulation loops through the algorithm and iterates through the time-steps until Equation (5.11) stabilizes over time, and hence the droplet surface temperature T_d converges. The convective heat transfer coefficient is derived by equating the theoretical heat flux derived from Newton's law of cooling to the time-averaged heat flux from the simulation up to that point in time. As the simulation moves through, variable time-intervals were chosen to calculate the heat transfer coefficient h_d, and it was determined as

$$h_d = \frac{\sum_{i}^{N_{steps}} (E_{w,g})_i}{A_{ts}(T_\infty - T_w)N_{steps}\Delta t}. \tag{5.18}$$

Basically, the total energy collected by the droplets up to the number of steps iterated up to that point in time, N_{steps}, is divided by the product of the temperature difference (or the

amount of subcooling), the area covered on the cold wall by the computational domain, and the total time up to that point.

Simulations were iterated anywhere between 10,000 time-steps up to 80,000 time-steps, depending on the length of each time-step. Simulations for larger droplets had larger cell sizes, which implies that each time-step is larger since the time-steps were chosen so that the average molecule would traverse a cell in about 2 time-steps. Regardless of the length of each time-step, however, simulations were iterated through the time-steps until the values of T_d and h_d stabilized. For this computational domain, h_d converges similarly to the model presented in Chapter 4, as shown by Figure 5.4. The plot shows the behavior of h_d for an increasing number of time-steps. The figure corresponds to a domain under pure steam exposure for 200 nm diameter droplets, standard atmospheric pressure, a subcooling of 3 K, and $s/d = 4$. The time-step used in this simulation was for $4.8\text{x}10^{-11}$ seconds and it is shown that this particular simulation starts to converge around 10,000 time-steps (a total of $4.8\text{x}10^{-7}$ computational seconds).

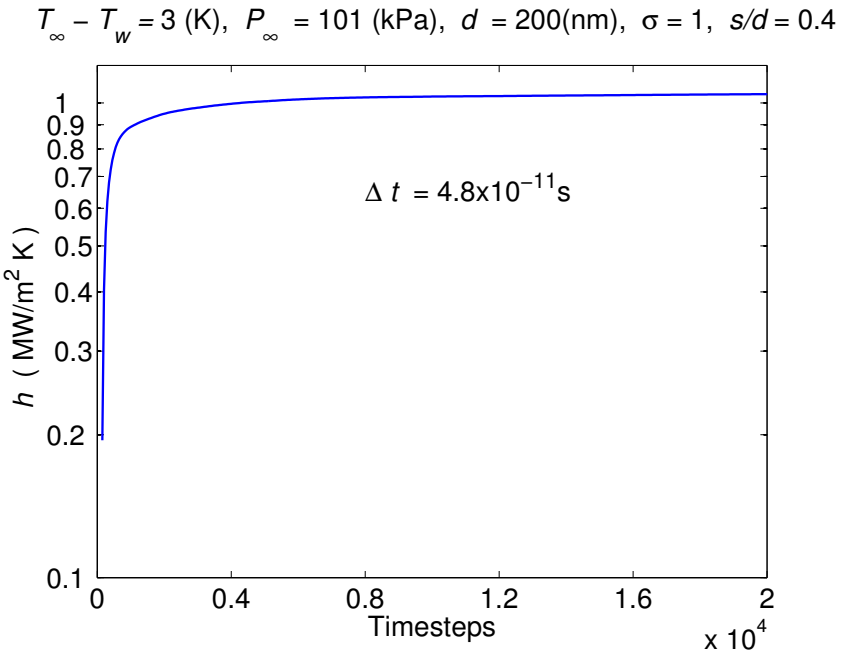


Figure 5.4: Convergence of the heat transfer coefficient calculations for perfect accommodation and $\theta = 90°$.

5.6 Model Results

Simulations for a range of conditions were run using the DSMC model developed here. The discussion below focuses on the resultant heat transfer coefficients behavior as droplet sizes were reduced for the conditions simulated. A base case was chosen for comparison to distinguish the effects of the varied parameters. The effect of contact angle in each case was explored by varying between contact angles of $70°, 90°$, and $110°$, just as done in the models from Chapters 3 and 4. The base case, shown as Figure 5.5 below, was chosen for the following conditions:

- Perfect accommodation($\sigma = 1$)
- Wall subcooling of 3 Kelvin ($T_\infty - T_w = 3.0$ K)
- Ambient at atmospheric pressure ($P_\infty = 101$ kPa)
- A droplet spacing to diameter ratio of 0.4 ($s/d = 0.4$)
- A pure saturated vapor at the ambient ($X_{w\infty}$)

Figures 5.6 through 5.10 below depict individual variations for each of these prescribed conditions.

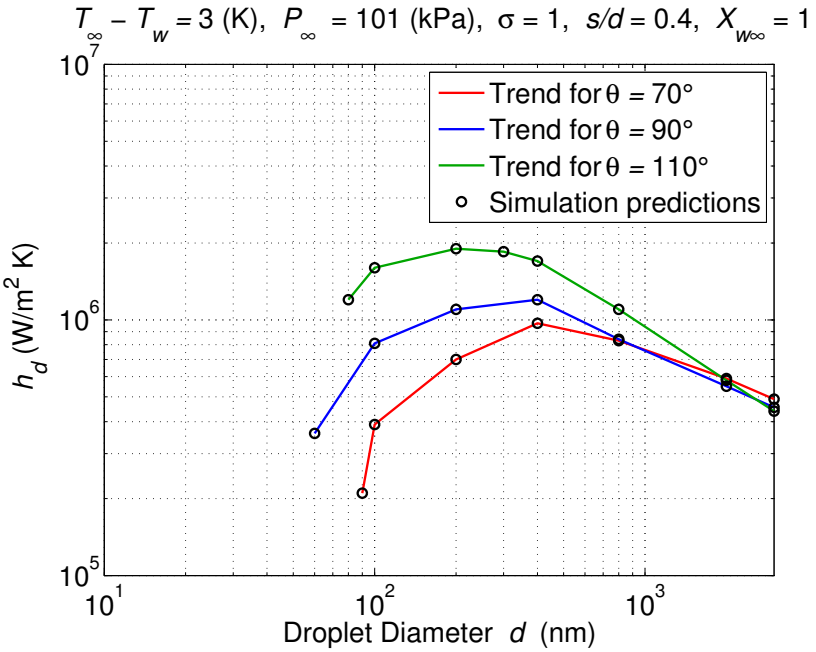


Figure 5.5: Droplet cluster DSMC h_d prediction for the base case of perfect accommodation, 3 K subcooling, one atmosphere, s/d=0.4, and a pure saturated vapor at the ambient.

Due to computational limitations, simulations beyond 3000 nm in diameter were not run, but appreciable knowledge is still gained from the range of sizes explored here. These sizes approximately correspond to Kn_d=0.02, which is near what is accepted to be in the continuum range. For diameters greater than 2000 nm, the simulations predicted behavior similar to what would be expected from continuum theory; while the variations were not large between contact angles, droplets with smaller contact angles (with a smaller conduction resistance) showed slightly higher h_d than droplets for the same size but possessing larger contact angles. However, at smaller sizes, the correlation between heat transfer coefficient and contact angle is reversed, where the more hydrophobic contact angles revealed the higher heat transfer coefficients. The further away droplet sizes deviated from the continuum range, the more pronounced these differences between varying contact angle became. At diameters near 300 nm, h_d seemed to peak and diminished as droplet sizes were further reduced, displaying the increasing non-continuum effects. This is consistent with the trends observed in the DSMC model for a single droplet, discussed in Chapter 4.

Figure 5.6 shows the behavior for a reduced thermal accommodation coefficient on the droplet. As discussed above, for a system with high $X_{w\infty}$, the thermal accommodation coefficient is expected to be close to unity, so we chose to explore this by reducing the value to $\sigma = 0.9$. Comparison of Figures 5.5 and 5.6 shows that the reduction of accommodation for values near one has a small effect in the variation of h_d, in agreement with models discussed in Chapters 3 and 4. In fact, the trends seen in both figures are almost identical, with only a nominal shift for the reduced accommodation.

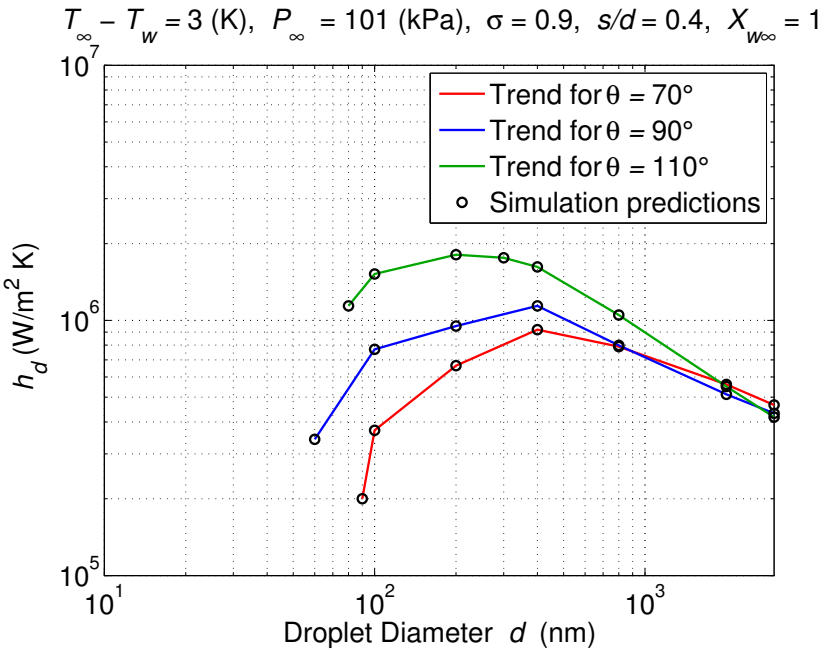


Figure 5.6: Droplet cluster DSMC h_d prediction at reduced accomodation

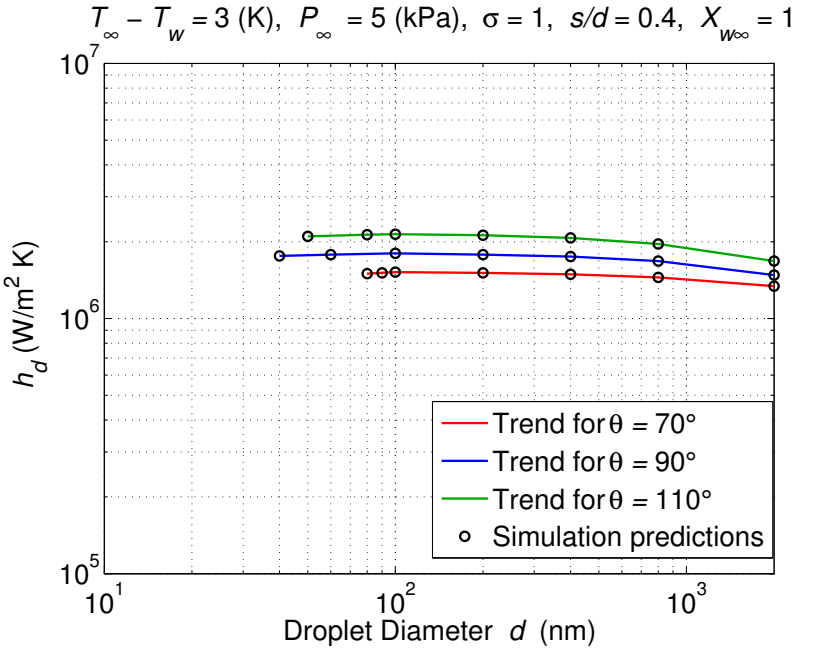


Figure 5.7: Droplet cluster DSMC h_d prediction at reduced pressures

Comparison of Figures 5.5 and 5.7 shows the effect of changing the ambient pressure. The trends seen for this reduced pressure at the modeled droplet sizes are consistent with those trends seen in the base case away from the continuum range. This is due to the reduced pressure increasing the mean free path of the flow and thus shifting the point at which non-continuum effects become significant to larger droplet sizes. The range of droplet sizes modeled here was well below diameters corresponding to continuum flow for this reduced pressure. At this pressure, a $\mathrm{Kn}_d = 0.05$, at which non-continuum effects should already be present, corresponds to droplet diameters of about 3500 nm. Therefore, the largest diameter shown in Figure 5.7 already includes non-continuum effects, and the flow only moves further away from continuum transport as the droplet sizes are reduced.

Comparison of Figures 5.5 and 5.8 illustrate the effect of wider spacing between the droplets. The more notable difference is the reduction in peak heat transfer coefficients at each contact angle when comparing the different conditions. Since the effective cold wall area where vapor drops don't condense increases with the increased spacing, this reduction in heat transfer coefficients makes physical sense.

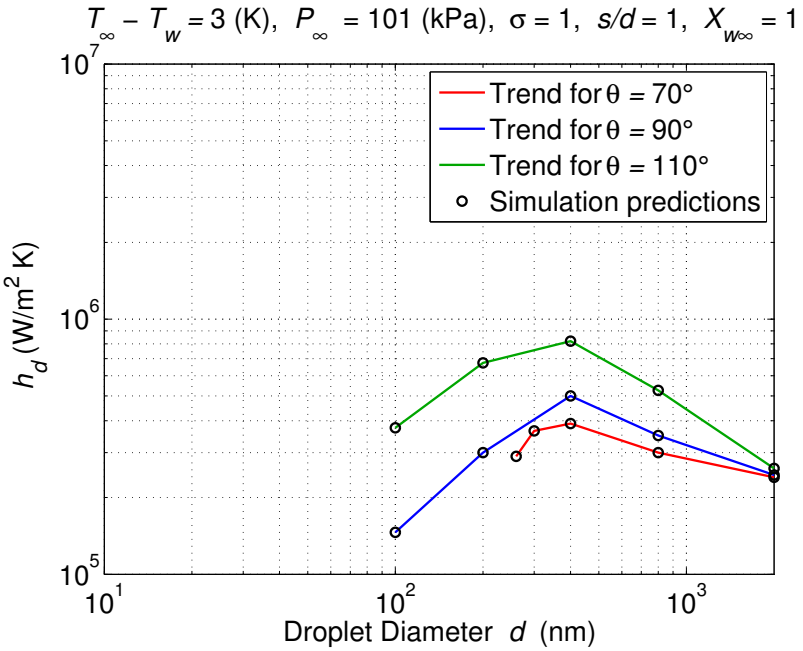


Figure 5.8: Droplet cluster DSMC h_d prediction for increased spacing

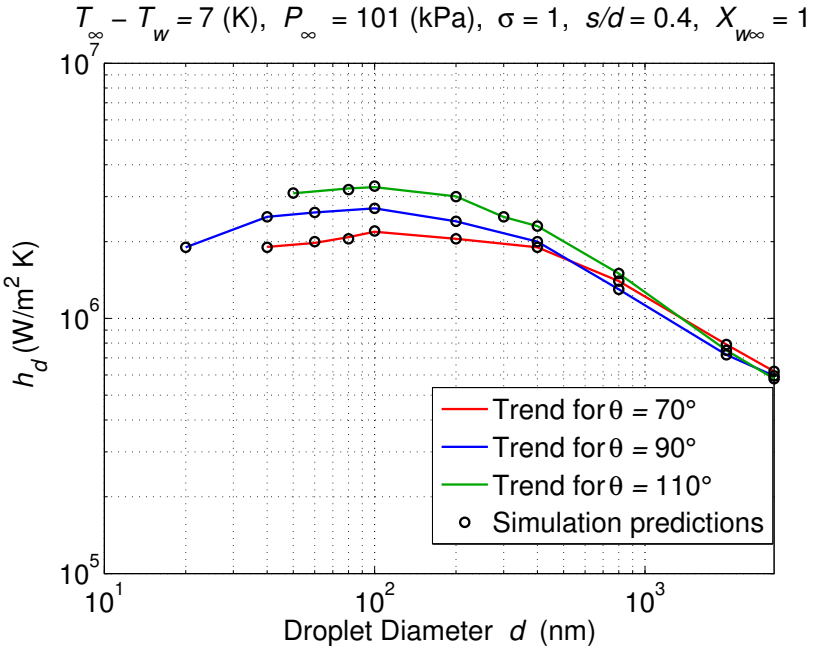


Figure 5.9: Droplet cluster DSMC h_d prediction for increased subcooling

Consistent with the predictions from the approximation model in Chapter 3, Figure 5.9 shows that the reduced diameters in the non-continuum range have less of an effect in the variations between contact angles for this higher subcooling. As in the single droplet DSMC model, h_d also levels off at the smaller diameters rather than dropping as drastically as happens in the base case.

Lastly, for the reduction in $X_{w\infty}$, comparing Figures 5.5 and 5.10 shows that the trends are very similar. There is a slight difference in peak heat transfer coefficients supported by the discussion in Chapter 1 stating that the presence of non-condensible particles at the droplet interface will tend to lower the vapor partial pressure there.

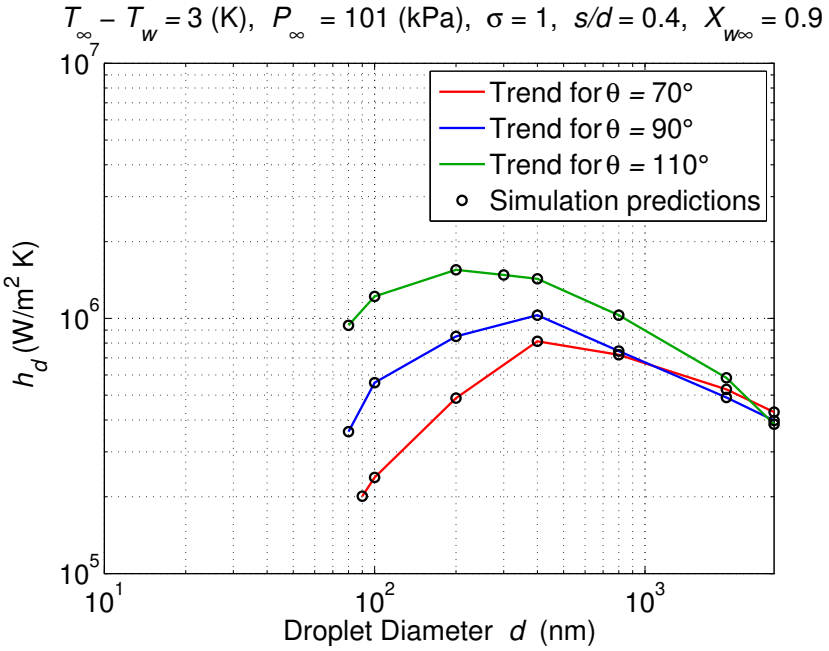


Figure 5.10: Droplet cluster DSMC h_d prediction for reduced water concentration $\theta = 90°$.

5.7 Implications of Model Predictions

As trends similar to those observed in Chapters 3 and 4 were shown here, the same conjectures can be made for this model. The conceptual implications are reiterated below with some specifics about this model.

- Due to the idealizations made, this model does not account for the spectrum of sizes occuring during real dropwise condensation proccesses and is therefore not presented as predicting precise heat transfer coefficients.
- This model is rather presented as a tool to understand the various mechanisms involved during dropwise condensation as diameters are reduced; conduction through the droplet, interfacial curvature effects on surface tension and saturation conditions, and non-continuum transport effects.
- Taking the droplet diameters presented here as mean droplet sizes for an array of droplets with a range of sizes, this model predicts heat transfer coefficients peaking near 200 nm mean droplet diameters as sizes are reduced. The peaking occurs due to the effects mentioned above becoming more significant.

- As in the single droplet DSMC model, effects of contact angle are more significant for mean droplet sizes away from the continuum range, where surfaces sustaining more hydrophobic droplets would attain higher heat transfer coefficients.
- As long as mean droplet sizes remain in the continuum regime, a reduction in diameter will continue to increase heat transfer coefficients. However, within this regime, the effects of varying contact angle on h_d are nominal for droplets sustaining contact angles in the range of 70° to 110°.

To appreciate the implications of all three models as a group and how they paired with one another, Chapter 6 discusses the results of the models in the context of continuum theory and in relation to one another.

Chapter 6

Model Comparisons and Validation

6.1 Introduction

In Chapters 3 through 5 we presented different models that attempted to simulate dropwise condensation on an array of droplets at reduced mean droplet sizes. It was stressed how we attempted to account for deviations of dropwise condensation from continuum theory at those reduced sizes. In this chapter we provide a visualization of those deviations. Since different techniques were used in each of the models, in this chapter we also seek to compare the different models with one another and identify the strengths and weaknesses of the models in relation to each other. We end this chapter by discussing the models in the context of previous experimental work.

This chapter is organized as follows:

- The nomenclature for this chapter is presented in Section 6.2
- Continuum theory is discussed in Section 6.3. The simulation results of each model from Chapters 3 through 5 are compared to a corresponding continuum solution in this same section.
- The results from each model are compared with one another and discussed in Section 6.4. A discussion of the limitations of each model relative to one another is also presented in this section.
- The models are discussed in the context of experimental work in Section 6.5.

6.2 Nomenclature

A	the effective area of the cold wall cross-section that is being analyzed
d	droplet diameter
h_d	dropwise condensation heat transfer coefficient
j_{bi}	molecular flux to interface in ballistic limit
j_{ds}	molecular flux to droplet interface from surrounding vapor space
j_s	molecular flux from vapor space in cell to interface
k_l	liquid thermal conductivity
Kn_d	Knudsen number, $=\lambda_\mathrm{m}/d$
P_{vd}	droplet vapor pressure
$P_{sat}(T_d)$	flat interface saturation pressure at temperature T_d
q_{dc}	energy conducted through the droplet
q_{dcn}	the net energy onto the droplet due to condensation
r_d	droplet radius
s	distance between droplet interfaces at mid plane
S_f	shape factor for droplet
T_d	droplet interface temperature
T_w	wall temperature
T_∞	ambient temperature
$T_{sat}(P_\infty)$	flat interface saturation temperature at pressure P_∞
λ_m	mean free path of molecules in vapor
ρ_l	liquid density
σ_{lv}	surface tension at interface temperature T_d
σ	droplet interface accommodation coefficient

6.3 Continuum Theory

Throughout this dissertation, we have stressed that flow can be characterized into different regimes depending on a characteristic length of the system compared to the mean free path λ_m of the flow. We described how, for the dropwise condensation domains mentioned here with diameter d, these regimes were categorized based on the Knudsen number defined as $\mathrm{Kn}_d = \lambda_\mathrm{m}/d$. From this, we can see that as droplet diameters are reduced, the Knudsen number increases, indicating transitions between different flow types, as was shown in Figure 2.3. In this section, we want to visualize this deviation from continuum theory and see how well our models accounted for transitions into different regimes. Specifically we want to visualize the deviation from what a continuum model would indicate.

6.3.1 The Continuum Model

In Chapter 2, it was described how the equilibrium vapor pressure on a droplet can vary from a flat-interface saturation pressure due to its curvature. This equilibrium droplet vapor pressure is described as

$$P_{vd} = P_{sat}(T_d)\exp\left(\frac{2\sigma_{lv}}{\rho_l r_d R T_d}\right), \tag{6.1}$$

where $P_{sat}(T_d)$ is the flat-interface saturation pressure corresponding to the vapor at temperature T_d. From analyzing the parameter $\exp\left(\frac{2\sigma_{lv}}{\rho_l r_d R T_d}\right)$, and from having seen its behavior with droplet radius r_d in Figure 1.4, it can be shown that as $\rho_l r_d R T_d$ becomes much larger than $2\sigma_{lv}$, the equilibrium droplet vapor pressure approximates the flat-interface saturation pressure. For the systems we are interested in, this occurs for very large droplet diameters which result to be in the continuum range. For the saturation condition at standard atmospheric pressure depicted in Figure 1.4, the drop vapor pressure approximates the flat-interface saturation pressure over droplet diameters of 900 nm. Furthermore, from analyzing Figure 1.3 in Chapter 1, it's observed that the surface tension considering curvature effects approximates the flat-interface surface tension for droplet diameters even below 900 nm. Therefore, for droplet sizes in the continuum range, the droplet interface is at or very near the flat-interface saturation conditions [7, 20].

Considering this occurrence that, for droplet sizes in the continuum range, the droplet interface approximates flat-interface saturation conditions, a relatively simple analysis can follow in order to determine heat transfer coefficients for a continuum. To make the continuum analysis for dropwise condensation on a cold wall, the same underlying assumptions from the models discussed in Chapters 3 through 5 are considered:

- Dropwise condensation occurs for a uniformly spaced array.
- A mean droplet diameter is used to simplify the analysis.
- Condensation only occurs on preferred nucleation sites, droplets. Specifically, there is no condensation in the space between the droplets.

If an energy balance is done at the droplet interfaces on a wall cross-section, the net energy coming in from condensation q_{dcn} has to be equal to the conduction going through the droplets q_{dc}:

$$q_{dc} = q_{dcn}. \tag{6.2}$$

In the models we have developed here, we have let the conduction through the droplet be expressed as

$$q_{dc} = k_l S_f (T_d - T_w), \tag{6.3}$$

where k_l is the thermal conductivity of the water, S_f is the shape factor described in Chapters 3 through 5 , T_d is the droplet interface temperature, and T_w is the cold wall temperature. Similarly, for a condensation heat transfer coefficient h_d, the condensation can be expressed as

$$q_{dcn} = h_d A (T_\infty - T_w), \tag{6.4}$$

where T_∞ is the temperature from the ambient. The area A in this case would be the effective area of the cold wall cross-section that is being analyzed. To put this into perspective, if we assume a triangular cross-section as was assumed in Chapters 3 and 5 (see Figure 3.4), the effective area A is the area of the triangular cross-section imposed on the cold wall. The shape factor S_f corresponds to $\frac{1}{2}$ of a droplet hemisphere since three 60° droplet cross-sections lie within the triangular domain. Substituting Equations (6.3) and (6.4) into Equation (6.2) gives

$$k S_f (T_d - T_w) = h_d A (T_\infty - T_w). \tag{6.5}$$

However, for a saturated vapor, T_∞ is $T_{sat}(P_\infty)$. In addition, as was shown above for a continuum, droplet interface conditions at equilibrium become the flat interface saturation conditions and $T_d = T_{sat}(P_\infty)$. Substituting into Equation (6.5) yields an expression for h_d

$$h_{d,continuum} = \frac{k S_f}{A}. \tag{6.6}$$

This shows that, for a continuum, the heat transfer coefficient is independent of the ambient and wall temperature difference. Furthermore, because the shape factor varies linearly with d, the continuum heat transfer coefficient is proportional to $1/d$.

6.3.2 Comparing Models to Continuum Theory

To appreciate the effects of the models developed for this dissertation, Figures 6.1 through 6.3 below illustrate how, as droplet sizes are reduced, each one of the models deviates further and further away from what continuum theory would predict.

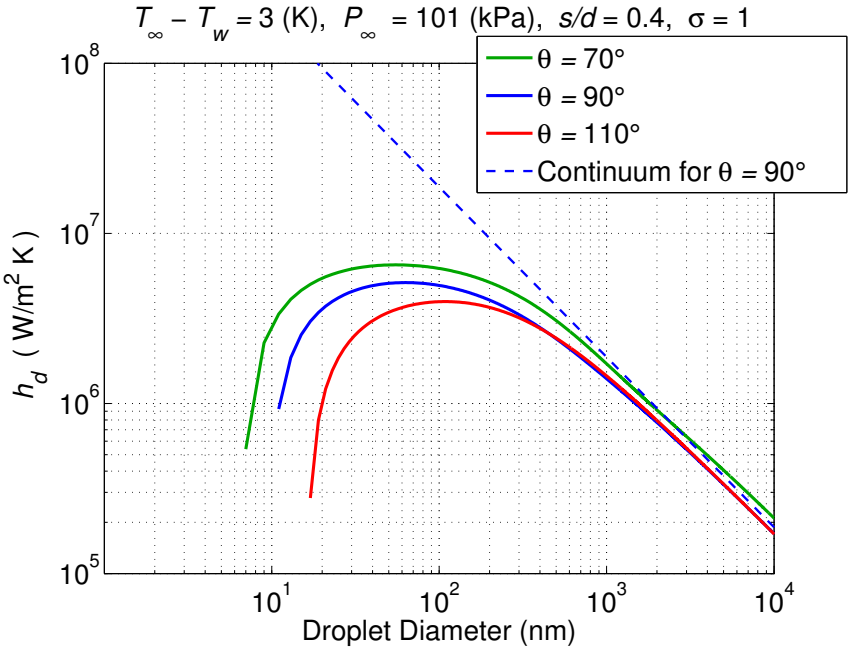


Figure 6.1: Continuum solution superimposed on the approximation model (from Chapter 3)

Of particular note from the figures above is the point at which they start deviating from continuum theory. Inspection of the figures indicates that appreciable deviation from continuum theory starts to occur around droplet diameters between 2000 nm and 4000 nm, respectively corresponding to Knudsen numbers of approximately $\mathrm{Kn}_d = 0.05$ and $\mathrm{Kn}_d = 0.02$ for the prescribed conditions. This is consistent with what would be expected, as described in Chapters 1 and 2 by the different flow regimes corresponding to different Knudsen numbers.

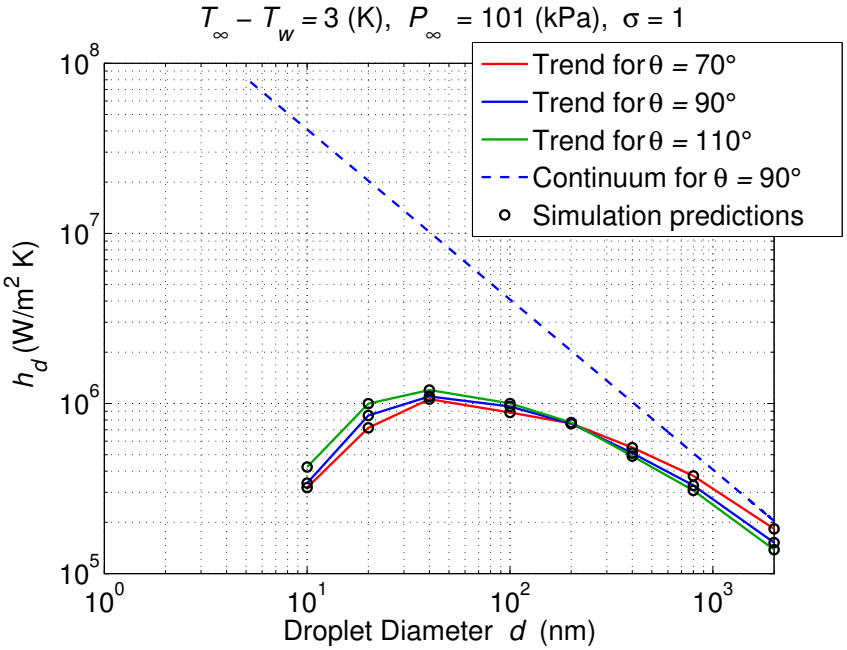


Figure 6.2: Continuum solution superimposed on the single droplet DSMC model (from Chapter 4)

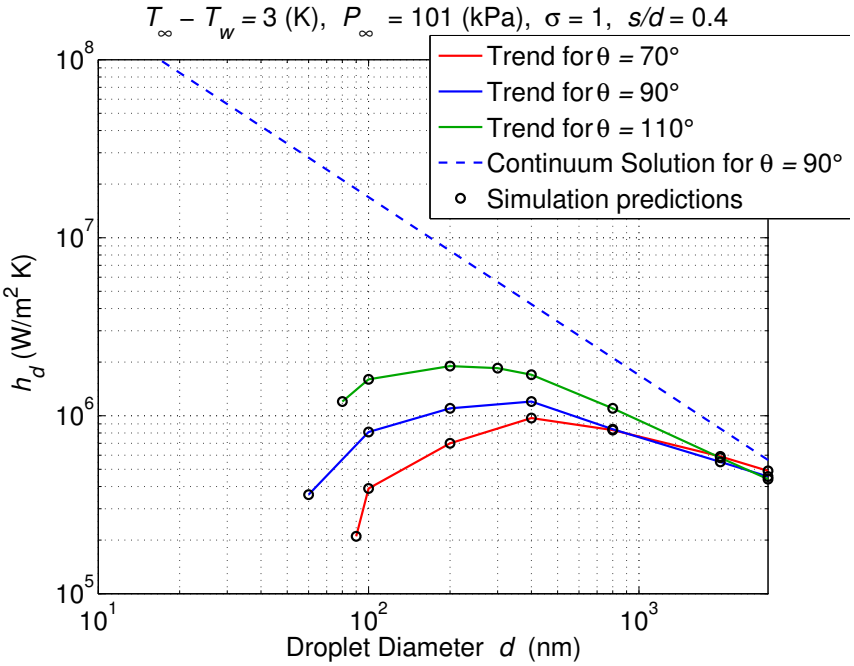


Figure 6.3: Continuum solution superimposed on the droplet cluster DSMC model (from Chapter 5)

6.4 Comparing Models with One Another

6.4.1 Approximation Model and the Single Droplet DSMC

Developing the DSMC model on the single droplet was the step following the initial approximate model in the evolution of the models. Although the single droplet DSMC model did not account for the effects of nearby droplets, it was still a stepping stone in the direction of verifying the validity of the approximate model. An attempt at comparing the single droplet DSMC model to the approximation model was done by adjusting the spacing between the droplets (s/d) in the approximation model. Recalling that in the single droplet DSMC model the ambient conditions were simulated at four radii from the center of the droplet, adjusting the spacing between the droplets was a justified approach in attempting to connect the two models. Comparison to the approximation model was then approached by finding an equivalent spacing within the droplet cluster that would approach the solution of the single droplet DSMC model. Figure 6.4 below shows this comparison, where the equivalent (s/d) was found to be 2.6.

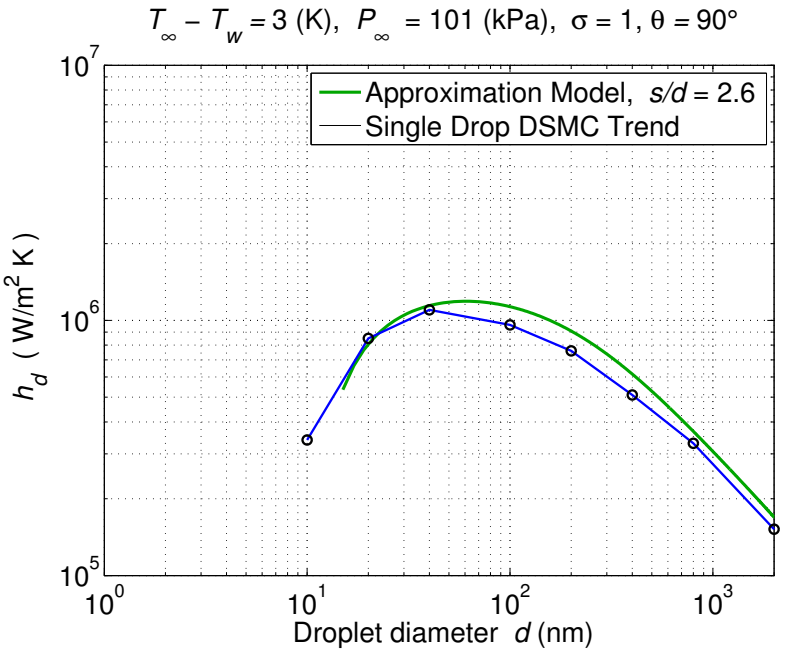


Figure 6.4: Comparison between the approximation model and the single droplet DSMC model for the prescribed conditions

This adjustment therefore implies that, in order for the approximate model to simulate a single droplet condensing on a wall at standard atmospheric conditions, which essentially ignores interactions between droplets, a spacing in between droplets of 2.6 times the diameter is needed in the approximate model. By using this spacing, the approximation model paired very close to the DSMC model, showing the same trend of increasing heat transfer coefficient as droplet diameters were reduced. For the prescribed conditions, this superimposing also shows heat transfer coefficients peaking around the same droplet diameters for both models, approximately around 50 nm droplet diameters. This resultant equivalent spacing is not so arbitrary. Since the single droplet DSMC model assumed atmospheric conditions at four radii from the center of the droplet, this corresponds to three radii from the droplet surface. The ratio of spacing between the atmosphere and the droplet surface is therefore three, which is fairly close to the 2.6 value needed for the approximation model to approach the single droplet DSMC model.

6.4.2 Approximation Model and the Droplet Cluster DSMC Model

While comparison of the approximation model and the single droplet DSMC model served as a step in the right direction, it wasn't a direct comparison due to the nature of having different computational domains. Ultimately, comparison between the approximation model and the droplet cluster DSMC model would be key to validating the development of different models

attempting to capture the same concepts. Initial comparison between the approximation model and the droplet cluster DSMC model at reduced sizes showed close agreement between the two for standard atmospheric conditions on a 90° contact angle droplet. Figure 6.5 below shows this comparison.

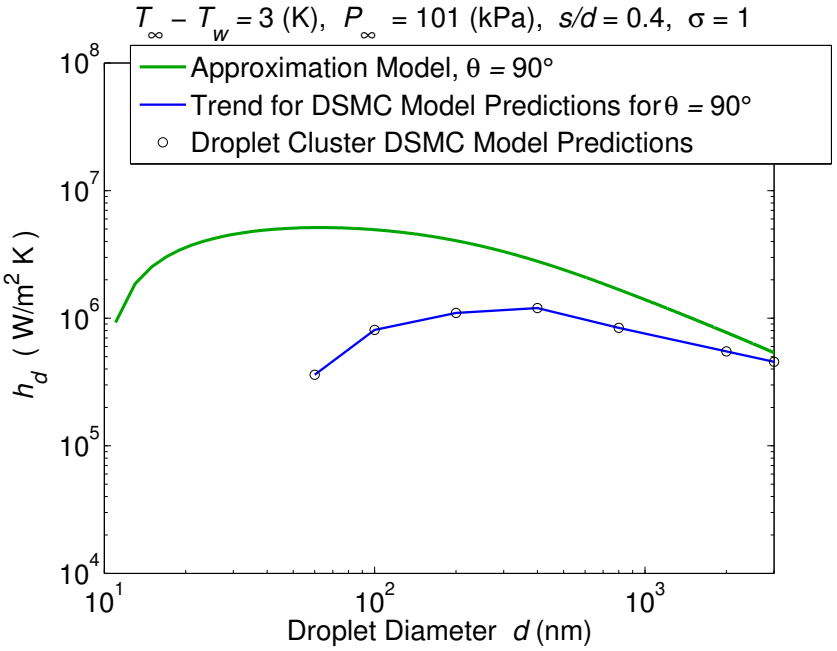


Figure 6.5: Comparison between the approximation model and the droplet cluster DSMC model for the prescribed conditions

While there was close agreement between the two models, there was still some disparity. The approximation model overestimated the heat transfer coefficient relative to the droplet cluster DSMC model for comparable conditions. Taking the DSMC model as a better approach, it was of interest to define the discrepancy.

Particular attention was given to the fashion in which the approximation model accounts for deviations from continuum. Recalling from Chapter 3 that the way the incident flux on a droplet was modeled was as Equation (6.7) below (same as Equation (3.10), rewritten here for reference), some insight is given to the possibility of adjusting the approximation model.

$$j_{ds} = j_s \left(1 - e^{-0.5/\mathrm{Kn_d}}\right) + j_{bi} e^{-0.5/\mathrm{Kn_d}}. \tag{6.7}$$

Equation (6.7) represents two potential sources of incident molecules on the droplet cluster: j_s which represents a kinetic theory flux coming from the space in between the droplets,

and j_{bi} which represents the flux that would be incident on the droplet from the ambient if it were in a completely ballistic regime. The multiplying terms $1 - e^{-0.5/\mathrm{Kn}_d}$ and $e^{-0.5/\mathrm{Kn}_d}$ are used to create the transition from continuum theory to purely ballistic; $1 - e^{-0.5/\mathrm{Kn}_d}$ goes from 1 to zero as droplet sizes are reduced and Kn_d increases, and $e^{-0.5/\mathrm{Kn}_d}$ simultaneously goes from zero to one. This has the effect of transitioning the contributions of j_s and j_{bi} to j_{ds} from continuum transport to purely ballistic. The 0.5 constant helps control when each term becomes dominant relative to the other, depending on the value of the Knudsen number. With a constant of 0.5, the flow effectively goes from fully continuum to fully ballistic over the range of $0.05 < \mathrm{Kn_d} < 20$.

Considering the nature of the approximation model as explained above, the 0.5 constant was manually swept through to identify a better fit for the approximation model to the predictions from the DSMC droplet cluster model. After iterating through a range of values, a constant of 0.3 instead of 0.5 in Equation (6.7) resulted in a better fit to the DSMC droplet cluster model, changing Equation (6.7) to Equation (6.8).

$$j_{ds} = j_s \left(1 - e^{-0.3/\mathrm{Kn_d}}\right) + j_{bi} e^{-0.3/\mathrm{Kn_d}}. \tag{6.8}$$

The new constant of 0.3 slightly changes the range over which the flow transitions from a continuum to fully ballistic. Rather than having the transition occur between Knudsen numbers in the range of $0.05 < \mathrm{Kn}_d < 20$ (as occurred with a constant of 0.5), a constant of 0.3 predicts the transition over the range of $0.03 < \mathrm{Kn}_d < 15$. Figure 6.6 shows the modified approximation model compared to the DSMC droplet cluster model for standard atmospheric conditions.

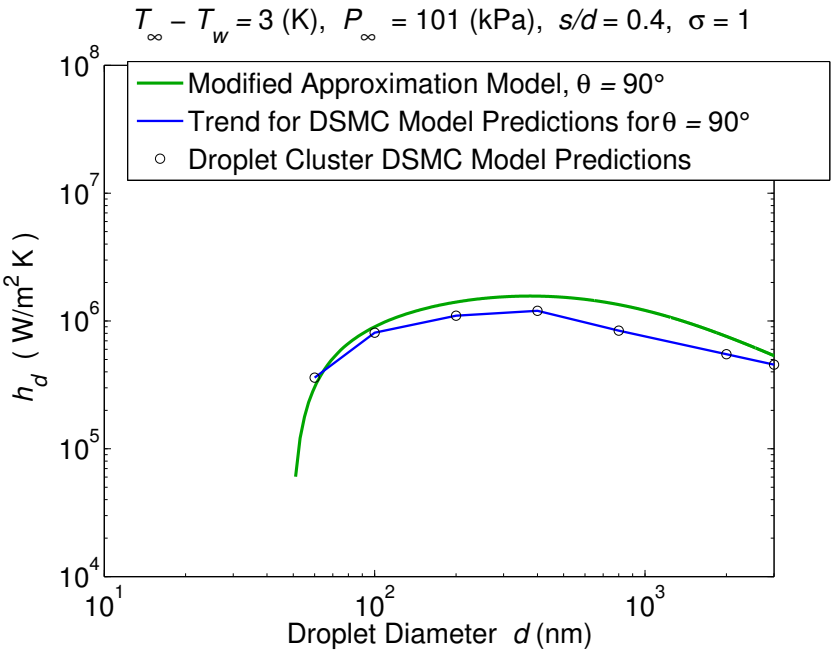


Figure 6.6: Comparison between the modified approximation model and the droplet cluster DSMC model for the prescribed conditions

While the 0.3 constant doesn't make the results identical, the predictions are fairly close for the range of 3000 nm down to 60nm droplet diameters. Considering that this modification doesn't affect diameters large enough in the continuum solution, this still renders the approximation valid over the continuum range. In fact, this transition range between $0.03 < \mathrm{Kn}_d < 15$ for the new constant is still within the range observed by rarefied flow studies and that which was discussed earlier in Chapters 1 and 2 [74, 59, 55, 62, 75, 16].

The above modification proves to hold valid for the conditions shown. However, Figures 6.7 through 6.9 below also show that the modified approximation model still holds valid for increased subcooling, lower accommodation, and is not too far off for varied contact angles. Thus, the modification to the approximation model is taken as a better correction factor than the one initially used.

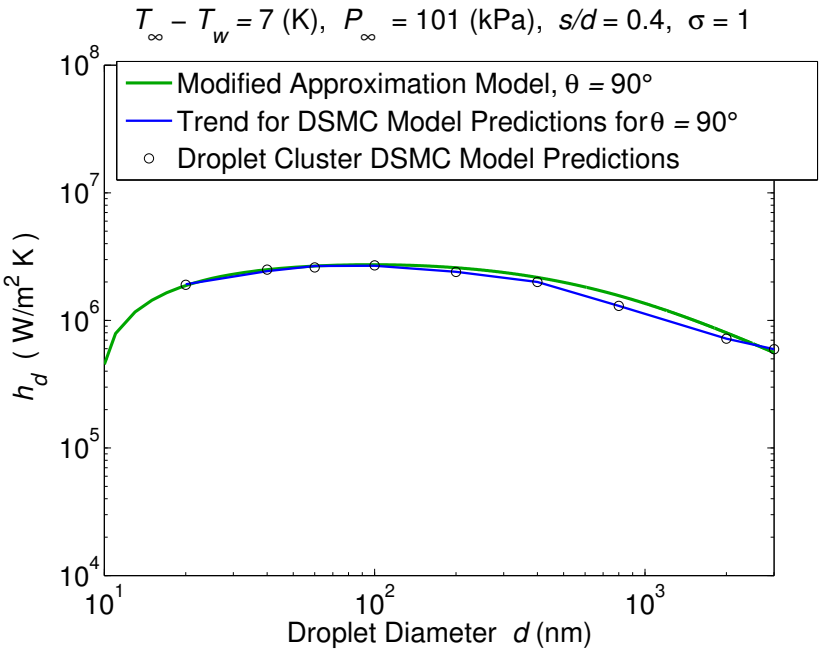


Figure 6.7: Comparison between the modified approximation model and the droplet cluster DSMC model for the prescribed conditions considering higher subcooling

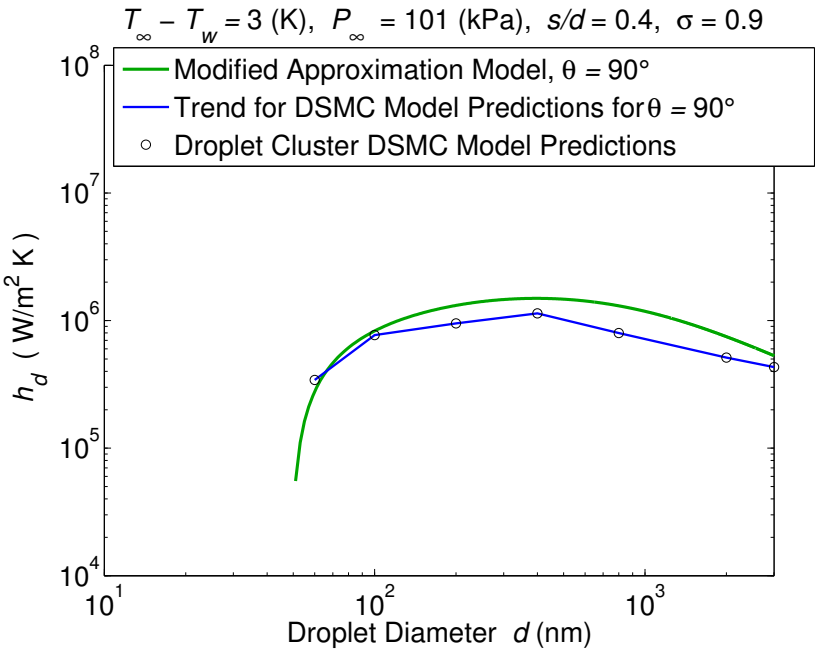


Figure 6.8: Comparison between the modified approximation model and the droplet cluster DSMC model for the prescribed conditions considering lower accomodation

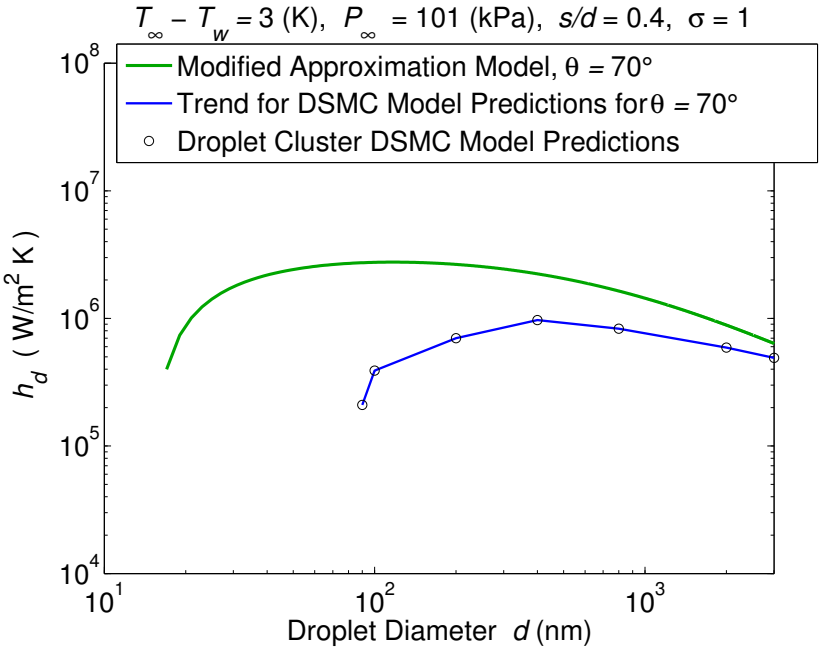


Figure 6.9: Comparison between the modified approximation model and the droplet cluster DSMC model for the prescribed conditions considering lower contact angles

6.4.3 Discussing the Limitations of the Models

Going through the development of the different models showed varying limitations from model to model. The most robust model is the droplet cluster DSMC. Further comparison to the approximation showed good agreement between the two models with some modifications. Aside from the robustness, the model also has the capability to incorporate non-condensible gases into the system. The robustness of the model comes at a computational cost though. Simulations to determine a single point on the plots shown above, using a 2 GHz Intel Core 2 Duo with 4 GB 1067 MHz DDR3 RAM running on Mac OS X10.7.5, took anywhere between 1 hour up to several hours to process depending on the size of the domain. Compared to the several minutes the approximation model took to develop an entire plot using the same machine, the slow speed of the DSMC model becomes an obvious limitation. The single droplet DSMC model had similar limitations to the droplet cluster DSMC model. The issue of speed arose in this model to the same extent that it arose in the droplet cluster DSMC model. Furthermore, as was shown in the discussion above, due to the nature of the model only considering a single droplet, it lacks the ability to account for interaction with the nearby droplets.

The approximation model has the speed that the DSMC models lack, but it is deficient in other aspects. The approximation model is only as accurate as calibrated by the corre-

Reference	h_{max} (kW/m^2K)	Wall Subcooling (K)	Contact Angle (°)	Predicted Mean Diameter(μm)
Kandlikar et al. [26]	280.0	3.5	90	6.0
Xuehu et al. [76]	800.0	10	106	2.0
Chung et al. [77]	50.0	10	90-110	35.0
Takeyama et al. [27]	600.0	3	90-110	2.7

Table 6.1: Maximum heat transfer coefficients recorded using steam at atmospheric pressure

sponding DSMC model. Furthermore, in order for the approximation model to function, correlations have to be developed externally to determine the fraction of molecules that are incident on the system of droplets being analyzed (F_{ii}, F_{us} and F_{id} as discussed in Chapter 3). Therefore, the approximation model can only predict h_d for the range of s/d that these correlations are developed. Once the correlations are developed, though, the model can be used with ease. The approximation model, however, does have the advantage of using the Xsteam water and steam property package developed for MATLAB [69] to determine water saturation properties.

6.5 Relevance to Experimental Work

While the draining mechanism of the condensate that occurs during real dropwise condensation is not taken into account, significant insight and implications are revealed from these studies.

To consider this in more detail, we note that the investigations indicated in Table 6.1 have reported heat transfer coefficient data for dropwise condensation of saturated steam at atmospheric pressure. Table 6.1 summarizes maximum dropwise condensation heat transfer coefficients reported in each study. For each of these studies, we found a corresponding mean droplet diameter predicted by our models for the conditions prescribed. In two of the investigations (Takeyama et al. [27] and Chung, et al.[77]), contact angle information was not provided for the conditions studied, so contact angles between 90° and 110° were assumed. On a vertical copper surface, Takeyama and Shimizu [27] measured a dropwise heat transfer coefficient of 6.00 x 10^5 W/m^2K at a surface sub-cooling of 3.0 K. Assuming a contact angle between 90° and 110°, results from the droplet cluster DSMC model and the adjusted approximation model (see Figures 6.3 and 6.6) suggest that the mean effective droplet diameter was about 2.7 μm. The model results for $\theta = 90°$ and $\theta = 110°$ in Figure 6.3 indicate two things about the potential for enhancing the dropwise condensation process. First, it is predicted that the measured performance in Takeyama's study is well below the maximum attainable heat transfer coefficient for dropwise condensation predicted at these conditions, which is indicated to be about 2.0 x 10^6 W/m^2K. The results of the DSMC model for the droplet cluster shown in Figure 6.3 also indicate that decreasing the effective mean

droplet size near 1.0 μm could raise the heat transfer coefficient to about 1.0 x 10^6 W/m^{2}K, which is almost twice the value measured by Takeyama and Shimizu. In a similar way, we have used the droplet cluster DSMC model to estimate a mean effective droplet diameter for the peak heat transfer coefficients in the other studies in Table 6.1. Taken as a whole, these estimates suggest that these studies typically show mean effective droplet diameters in the 2-40 μm range. Real condensation processes have a distribution of sizes, and the prediction should be interpreted as an estimate of the mean effective size for the process under the indicated conditions. The estimates of the effective mean droplet diameter produced with these models are approximate. They do, however, provide at least two indications: (1)the potential enhancement of the heat transfer coefficient that can be achieved by creating circumstances that result in smaller mean droplet diameters, and (2)the limitations resulting from non-continuum effects.

The predictions of these models are useful guidance for researchers aiming to design enhanced dropwise condensation heat transfer by creating nano-stuctured surfaces that have localized hydrophilic areas on an otherwise hydrophobic surface [78, 45, 33, 36, 79, 80, 81, 82, 83]. The models developed here indicate that creating a surface that results in smaller mean droplet sizes will tend to enhance the heat transfer coefficient, but only up to a point where the peak heat transfer coefficient is attained. As noted above, our models predict that a small decrease in mean droplet size can substantially increase the dropwise condensation coefficient. For the continuum regime, the models also all indicate that hydrophobic surfaces sustaining droplets with contact angles in the range of 90°-110° perform about equally well.

For water condensing at atmospheric pressure on a surface with a (hydrophobic) contact angle of 110°, the DSMC model on a droplet cluster indicates that the peak heat transfer coefficient would occur for droplet diameters of about 200 nm. This implies that the maximum possible performance of the surface could be attained if patterning of hydrophilic and hydrophobic areas resulted in mean droplet diameters near 200 nm. While patterning at this resolution may be difficult to achieve in practice, the model indicates that a substantial enhancement of the condensing heat transfer coefficient could be attained if patterning can reduce the mean droplet size down to a range between 300nm-2μm, which is below those that have been observed in the studies mentioned in Table 6.1.

Chapter 7

Conclusion

In this dissertation we investigated the effects of reducing mean droplet size on heat transfer coefficients during dropwise condensation. The objective was to capture the different mechanisms that become significant as the flow type transitions from continuum transport at larger droplet sizes to free molecular flow at sizes where the droplet diameters are comparable to the flow's mean free path. We developed three different models that capture the same mechanisms and effects using different modeling techniques.

7.1 Summary of Models

In Chapter 3 we presented a model developed in MATLAB for a cluster of water droplets condensing on a cold wall within saturated steam. The distribution of droplet sizes was approximated by assuming a mean droplet size for the array. The model uses an approximation technique to account for the transition from continuum transport to free molecular flow. Deviations from flat-interface saturation conditions were accounted for vapor pressure by considering a hemispherical droplet in equilibrium with its surroundings. Deviations from the flat-interface surface tension were accounted for by using a correction based on the Tolman length. Water saturation thermodynamic properties were obtained using the X-Steam water and steam property package within MATLAB. A molecular simulation type of model was used to determine the fraction of molecules from the ambient that strike a droplet during condensation, where ambient conditions were assumed to be known on the top plane tangent to the droplet peaks.

In Chapter 4 we presented a DSMC model(programmed in C) on a single water droplet condensing on a cold wall within saturated steam, which in effect ignored interactions with neighboring droplets. Symmetry allowed the computational domain to be reduced to one-fourth of a droplet hemisphere. Deviations from flat-interface saturation conditions were

accounted for in the same fashion as they were in Chapter 3. Ambient saturation conditions were assumed to be known at four droplet radii away from the center of the droplet, as has been done in previous studies of similar applications presented in the Literature Review portion of this dissertation (Chapter 2). To obtain water saturation thermodynamic properties, subroutines were developed to perform sixth-order Lagrangian interpolations between known values as a function of saturation temperature. The known values were obtained from the National Institute of Standards and Technology (NIST) website, and the interpolations were valid for properties evaluated at any temperature between 5 °C and 370 °C.

In Chapter 5 we presented a DSMC model on a droplet cluster condensing on a cold wall. As in Chapter 4, symmetry was used to reduce the computational domain to a smaller, more manageable size. Assuming uniform droplet distribution, this computational domain reduced to a single triangular unit cell. Deviations from flat-interface saturation conditions were accounted for in the same fashion as they were in Chapter 3. Ambient conditions were assumed to be known on the top plane tangent to the droplet peaks, where water saturation thermodynamic properties were also obtained from subroutines developed to perform sixth-order Lagrangian interpolations between 5 °C and 370 °C.

In Chapter 6 we compared the three different models to predictions for corresponding continuum theory models. The three models were further compared to one another. The DSMC model on the droplet cluster was taken as the most comprehensive model, and a correction to the approximation model presented in Chapter 3 was suggested as a way of calibrating it.

7.2 Concluding Statements

While the different models showed slightly varying heat transfer coefficients due to different assumptions and approximations, the general trends were the same. Heat transfer coefficients increased as droplet sizes were reduced, but only up to the point where non-continuum and curvature effects started to become significant. Once heat transfer coefficients peaked at a given diameter, heat transfer coefficients dropped as mean droplet sizes were further reduced, where non-continuum transport and droplet curvature effects dominated. Heat transfer coefficients peaked when droplet diameters were on the order of 100s of nanometers. In the DSMC droplet cluster model, compared to pure steam at atmospheric pressure condensing on a cold wall with 3 °C of sub-cooling, the following effects were observed:

- Lower thermal accommodation on the droplet results in nominal differences as long as the thermal accomodation remains near one.

- Reduced pressure results in non-continuum effects affecting transport at larger diameters, starting near 3500 nm, as opposed to near 200 nm for the base case. This is directly due to the mean free path of the flow increasing as pressure is reduced.
- Wider spacing between the droplets results in a decrease of peak heat transfer coefficients from about $2x10^6 W/m^2K$ to about $5.8x10^5 W/m^2K$.
- Increased wall sub-cooling from 3 °C up to 7 °C has the effect of minimizing the differences in heat transfer coefficient between varying contact angles at smaller sizes.

The models developed here predict the local dropwise condensation heat transfer coefficient in an array of droplets having specified mean diameter, spacing, and contact angle. A real dropwise condensation process generally involves a surface covered by droplets of different size and spacing, with the distribution of droplet sizes dictated by the coalescence and sweeping effects of large droplets as they run down the surface. The model analysis presented here does not consider the drainage mechanism. It does, however, provide insight into how the overall dropwise condensation heat transfer coefficient for a surface will change as the mean droplet diameter changes.

The predictions of the models discussed in this dissertation are useful guidance for researchers aiming to improve dropwise condensation heat transfer by creating superhydrophobic nanostuctured surfaces that result in smaller condensate sizes. The models developed here indicate that creating a surface which results in smaller mean droplet sizes can substantially enhance the heat transfer coefficient, up to the point where the peak heat transfer coefficient is attained.

Appendix A

Chapter 3 Appendix

A.1 Monte Carlo Model of Molecular Transport in Ballistic Limit

To accurately proceed with the Monte Carlo approach, initiating a particle at a random position to move in a random direction required random sampling of the position and direction. Furthermore, reflecting a particle diffusely from the condensing surface required similar random sampling methods.

A.1.1 Sampling Direction

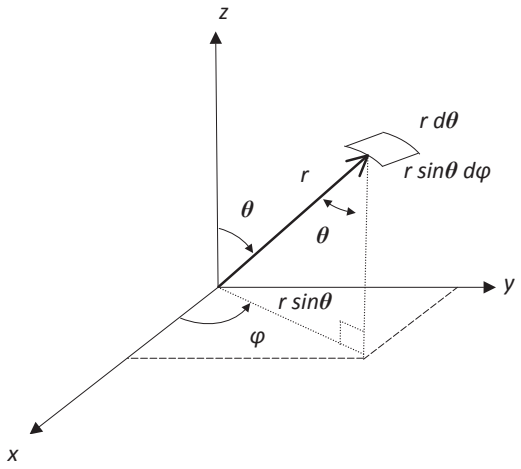


Figure A.1: General Coordinate System

When initiating a particle using the coordinate system in Figure A.1, a random direction for the particle to move in was chosen by randomly choosing θ and ϕ angles. By letting $f_{\theta\phi}$ represent the probability distribution function, $f_{\theta\phi}d\theta d\phi$ then represents the probability that θ and ϕ are within the ranges θ to $(\theta + d\theta)$ and ϕ to $(\phi + d\phi)$, where

$$f_{\theta\phi}d\theta d\phi = \frac{differential\ area\ for\ r=1}{total\ hemispherical\ area\ for\ r=1} = \frac{[r^2 \sin\theta\, \mathrm{d}\theta\, \mathrm{d}\phi]}{[2\pi\, r^2]_{r=1}} = \frac{\sin\theta\, \mathrm{d}\theta\, \mathrm{d}\phi}{2\pi}. \quad (A.1)$$

A cumulative distribution function for f_θ and f_ϕ are then defined respectively as F_θ and F_ϕ, where

$$F_\theta = \int_0^\theta f_\theta \mathrm{d}\theta = \int_0^\theta \sin\theta\, \mathrm{d}\theta = [-\cos\theta]_0^\theta = -\cos\theta + 1 = 1 - \cos\theta, \quad (A.2)$$

and

$$F_\theta = \int_0^\phi f_\phi \mathrm{d}\phi = \int_0^\phi \frac{1}{2\pi}\mathrm{d}\phi = \frac{\phi}{2\pi}. \quad (A.3)$$

By setting F_θ and F_ϕ equal to a random number $\Re$, where $0 < \Re < 1$, sampling relations for θ and ϕ can be derived by solving for the respective variables [75]. Therefore, to sample θ, we obtain:

$$1 - \cos\theta = \Re \Rightarrow \cos\theta = 1 - \Re = \Re \quad (A.4)$$

which gives

$$\theta = \cos^{-1}\Re \quad (A.5)$$

with $0 \leq \theta \leq \pi/2$. Similarly, to sample ϕ, we obtain:

$$\frac{\varphi}{2\pi} = \Re \Rightarrow \varphi = 2\pi\, \Re \quad (A.6)$$

with $0 \leq \phi \leq 2\pi$.

Using MATLAB's function *rand* to generate $\Re$, random directions were generated with these relations.

A.1.2 Random Starting Position

Choosing random starting **x** and **y** positions, x_{start} and y_{start}, was done using the same approach and resulted in:

$$x_{start} = (s + d)\Re \tag{A.7}$$

and

$$y_{start} = \frac{\sqrt{3}(s + d)\Re}{2}, \tag{A.8}$$

with $\Re$ being a random number where $0 < \Re < 1$. These starting positions, however, have a range that covers a rectangular area. To correct for this, any starting positions that resulted outside the triangular boundary were discarded and another random point was generated until it landed within the triangular boundary, thereby only creating random positions for the triangular boundary.

A.1.3 Diffuse Reflection

After a particle was initiated and allowed to travel through the space down to the condensing surface, if it happened to reach the condensing surface, the particle then had to be reflected diffusely. Using the random sampling results from above for the θ and ϕ directions, a new random direction was chosen between $0 \leq \theta \leq \pi/2$ and $0 \leq \varphi \leq 2\pi$ from the point where the particle struck the condensing surface.

A.1.4 Specular Reflection at Lateral Boundaries

When a particle reached a specularly reflective boundary, the particle was reflected about the normal unit vector of the boundary, as shown in Fig. A.2, where $\hat{r}$ is the reflected unit vector, $\hat{n}$ is the normal unit vector, $\hat{i}$ is the incident unit vector, and β is the angle of incidence and reflection. The reflected unit vector was then given as:

$$\hat{r} = -2(\hat{n} \cdot \hat{i})\hat{n} + \hat{i}. \tag{A.9}$$

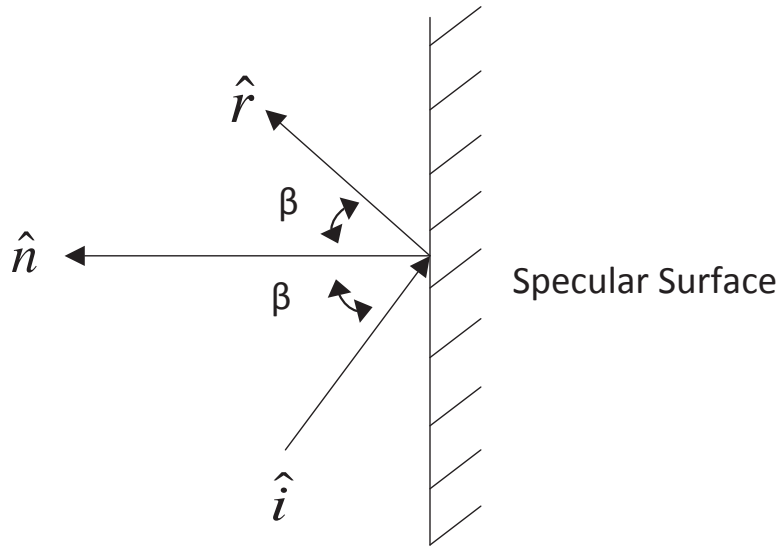


Figure A.2: Specular reflection at lateral planes of unit cell

A.1.5 Algorithm for Modeling Molecules from Unit Cell Upper Boundary Aperture to Droplets

Based on the theory described above for the Monte Carlo scheme, the following procedure was used to determine the fraction of molecules from the far field that pass through the triangular aperture in the upper unit cell boundary and hit one of the droplet interfaces (F_{us}):

(1) A random starting position for a new particle is chosen on the peak tangent plane within the triangular boundary.

(2) A random moving direction between $0 \leq \theta \leq \pi/2$ and $0 \leq \varphi \leq 2\pi$ is chosen for the particle in (1).

(3) The particle is moved one time-step in the direction chosen in (2) by giving it an arbitrary velocity of 100 m/s and choosing to divide the trajectory into 100 time steps.

(4) After advancing the particle one time step, four checks are performed in the following order:

(a.) Check to see if the particle strikes a droplet. If true, the particle is counted as one that strikes a droplet, and the simulation for that particle is terminated. Then, a

new particle is initiated and the simulation starts back at step (1) again. If false, the simulation proceeds to step (b.).

(b.) Check to see if the particle strikes one of the specularly reflective boundaries. If true, the particles direction is changed as a specular reflection and the algorithm goes back to step (3). If false, the simulation proceeds to step (c.).

(c.) Check to see if the particle strikes the condensing surface. If true, the particles direction is changed as a diffuse reflection, and the algorithm goes back to step (3). If false, the simulation proceeds to step (d.).

(d.) Check to see if the particle has gone back into the atmosphere by checking its position. If true, the particle is counted as one that didnt strike a droplet throughout its trajectory, and the simulation goes back to step (1) again with a new particle. If false, the simulation goes back to step (3) as the particle is moved another time step to perform the four checks. Every particle is moved until it either strikes a droplet or goes back into space.

After every particle was counted as either striking a droplet or one that went back into the atmosphere, the ratio of particles that struck a droplet to the total number of particles simulated up to that point was recorded (F_{us}). The above procedure was done until enough particles had been simulated to stabilize the fraction. To determine the fraction of molecules emitted by the droplet interfaces that strike the other two droplets (F_{id}) and the fraction that return to their interface of origin (F_{ii}), a Monte Carlo simulation similar to the one described above was used. The only difference was that the particles were emitted with random location and direction from one of the droplet interfaces, rather than from the triangular aperture at the top of the unit cell.

The variations of F_{us}, F_{id}, and F_{ii} with $\xi = s/d$ and θ determined from the Monte Carlo simulations were curve-fit to obtain the following relations:

For $\theta = 70°$:

$$F_{us} = \exp(0.1654\xi^2 - 1.1634\xi - 0.0116) \tag{A.10}$$

$$F_{id} = \exp(-0.34\xi^5 + 1.838\xi^4 - 3.948\xi^3 + 4.543\,\xi^2 - 3.9191\xi - 1.1953) \tag{A.11}$$

$$F_{ii} = \exp(0.4444\xi^5 - 2.9489\xi^4 + 7.2561\xi^3 - 7.857\,\xi^2 + 2.9295\xi - 3.7526) \tag{A.12}$$

For $\theta = 90°$:

$$F_{us} = \exp(0.1292\xi^5 - 0.716\xi^4 + 1.4642\xi^3 - 1.2421\xi^2 - 0.3953\xi - 0.0028) \tag{A.13}$$

$$F_{id} = \exp(-0.0091\xi^6 + 0.0631\xi^5 - 0.0756\xi^4 - 0.447\xi^3 + 1.704\xi^2 - 2.9651\xi - 0.8362) \tag{A.14}$$

$$F_{ii} = \exp(-0.1082\xi^6 + 1.0937\xi^5 - 4.3736\xi^4 + 8.768\xi^3 - 9.0585\xi^2 + 4.0275\xi - 3.5404) \tag{A.15}$$

For $\theta = 110°$:

$$F_{us} = \exp(0.0727\xi^5 - 0.5020\xi^4 + 1.300\xi^3 - 1.4973\xi^2 + 0.0664\xi - 3.000 \times 10^{-5}) \tag{A.16}$$

$$F_{id} = \exp(-0.0896\xi^5 + 0.5482\xi^4 - 1.3885\xi^3 + 2.158\xi^2 - 2.9350\xi - 0.4276) \tag{A.17}$$

$$F_{ii} = \exp(-1.5165\xi^6 + 10.742\xi^5 29.868\xi^4 + 41.299\xi^3 - 29.634\xi^2 + 10.124\xi - 3.7119) \; . \tag{A.18}$$

Appendix B

Chapter 4 Appendix

B.1 Simulation Algorithm

The simulation begins by loading the cells with a set number of vapor particles in the $r_d - 4r_d$ space. Depending on the prescribed conditions for the simulation, 12-30 particles were loaded in each cell, each representing up to 200 molecules. The average number of particles is arbitrarily set to match the ambient number density. Each particle is given a random location sampled from a uniform distribution, while random velocities and rotational energies are sampled from a Boltzmann distribution at the outer boundary temperature. The simulation subsequently marches forward in time Δt seconds for N_{steps} steps, and the following actions are taken at each time-step:

(1) Statistical information is collected for the particles in each cell

(2) Each particle is moved according to their given velocities for the length of the time-step

(3) The position after the movement for each particle is checked for any interaction with the boundaries:

   (a.) If a particle struck a droplet, the particle was either absorbed or reflected, depending on the accommodation coefficient. If it was considered absorbed, the net energy added to the droplet was noted on a tally and the particle was removed from the simulation.

   (b.) If a particle traversed the outer boundary, the particle was simply removed from the simulation

   (c.) If a particle struck a specular surface at $x = 0$ or $y = 0$, the normal component of the velocity was reversed

   (d.) If a particle struck the cold wall, the particle was diffusely reflected by preserving its speed but choosing a new random direction

(4) To simulate emission from the ambient, particles are randomly added to cells at the outer boundary surface. Velocities and rotational energies were sampled from appropriate Boltzmann distributions at the outer boundary temperature.

(5) The droplet surface temperature was solved for and particles were subsequently added to the surface using the methods described in Section 4.4.4 above. Velocities and rotational energies were sampled from appropriate Boltzmann distributions at the surface temperature.

(6) Candidate collision pairs were randomly selected in each cell. A probabilistic selection rule was used to determine whether a collision occurred or not [52].

(7) For each of the pairs selected for collision, the collision was executed based on the hard sphere interaction model, where energy and momentum are conserved. Details can be found in work from Carey et al. and Baganoff [61, 52].

(8) Steps (1)-(7) were iterated anywhere between 30,000 up to 400,000 time-steps until the value for the droplet temperature T_d and the heat transfer coefficient h_d converged.

B.2 Sampling Random Positions and Velocity Directions

Finding an arbitray direction for particle velocities was done by choosing a random polar (zenith) and azimuthal angle, exactly as described in Section A.1.1 in Appendix A. In a similar way, this same procedure was used to find arbitrary particle positions in the $r_d - 4r_d$ space. The only addition was to uniformly sample the radius. This gives

$$r_{start} = \Re(4r_d - r_d) \tag{B.1}$$

where r_{start} is a random radial distance used in combination with a random polar (zenith) and azimuthal angle to initiate random particle positions. A single random position therefore consists of choosing three different random numbers $\Re_1, \Re_2, \Re_3$ in combination with the definitions of

$$\theta = cos^{-1}\Re_1 \tag{B.2}$$

$$\phi = 2\pi\Re_2 \tag{B.3}$$

$$r_{start} = \Re_3(4r_d - r_d) \tag{B.4}$$

B.3 Specular Reflections

Specular reflections were performed in the same way as described in Appendix A. The reader is encouraged to refer to that section for further details.

Appendix C

Chapter 5 Appendix

C.1 Simulation Algorithm

labelalgoACh4 The simulation begins by loading the cells with a set number of vapor particles in the computational domain in the volumetric space between the droplets. The average number of particles for the computational domain is arbitrarily set to match the ambient number density. Depending on the prescribed conditions for the simulation, 10-15 particles were loaded in each cell, each representing up to 200 molecules. Each particle is given a random location sampled from a uniform distribution, while random velocities and rotational energies are sampled from a Boltzmann distribution at the outer ambient temperature. The simulation subsequently marches forward in time Δt seconds for N_{steps} steps, and the following actions are subsequently taken at each time-step:

(1) Statistical information is collected for the particles in each cell

(2) Each particle is moved according to their given velocities for the length of the time-step

(3) The position after the movement of each particle is checked for any interaction with the boundaries:

   (a.) If a particle struck one of the droplet interfaces, the particle was either absorbed or reflected, depending on the thermal accommodation coefficient. If it was considered absorbed, the net energy added to the droplet was noted on a tally and the particle was removed from the simulation. Non-condensible particles were kept on a separate tally to be removed at a later step.

   (b.) If a particle traversed the top tangential plane (the ambient boundary where z=0), the particle was simply removed from the simulation as it was considered lost into the ambient and outside of the computational domain

(c.) If a particle struck one of the specular surfaces (the lateral sides of our triangular computational domain), the particle was specularly reflected by reversing the velocity component normal to the stricken plane

(d.) If a particle struck the cold wall, the particle was diffusely reflected by choosing a new random direction from a uniform distribution while rotational and translational energies were sampled from a Boltzmann distribution at the cold wall temperature.

(4) To simulate emission from the ambient, particles are randomly added to cells at the top tangential plane within the computational domain(the ambient boundary where z=0). Velocities and rotational energies were sampled from appropriate Boltzmann distributions at the prescribed ambient temperature.

(5) The droplet surface temperature is solved for and particles were subsequently added to the droplet surfaces using the methods described in section 5.4.5 above. Velocities and rotational energies are sampled from appropriate Boltzmann distributions at the droplet surface temperature.

(6) Candidate collision pairs from the volumentric space between the droplets are randomly selected in each cell. A probabilistic selection rule based on their relative velocities is used to determine whether a collision occurs or not [52].

(7) For each of the pairs selected for collision, the collision was executed based on the hard sphere interaction model, where energy and momentum are conserved. Details can be found in work from Carey et al. and Baganoff [61, 52].

(8) Steps (1)-(7) were iterated anywhere between 10,000 up to 80,000 time-steps until the value for the droplet temperature T_d and the heat transfer coefficient h_d converged.

C.2 Diffuse Reflections and Droplet Emission

To perform diffuse reflections and particle emission from droplets, it was necessary to establish local coordinate systems at the center of each droplet. It simply became convenient to have coordinates local to each droplet to know exactly where particles struck a droplet relative to its center. This was especially useful in two cases:

a) When particles were not fully accommodated and needed to be diffusely reflected, or

b) In the droplet emission stage when positions and velocities were to be generated randomly on the surface of each droplet.

The coordinate system below shown by Figure C.1 depicts how the local coordinate systems(blue) were oriented relative to the global coordinate system (orange). Subscripts 1,

2, and 3 denote local coordinate systems corresponding to the respective droplet number, where the numbering was chosen arbitrarily and proceeding in a counter-clockwise fashion.

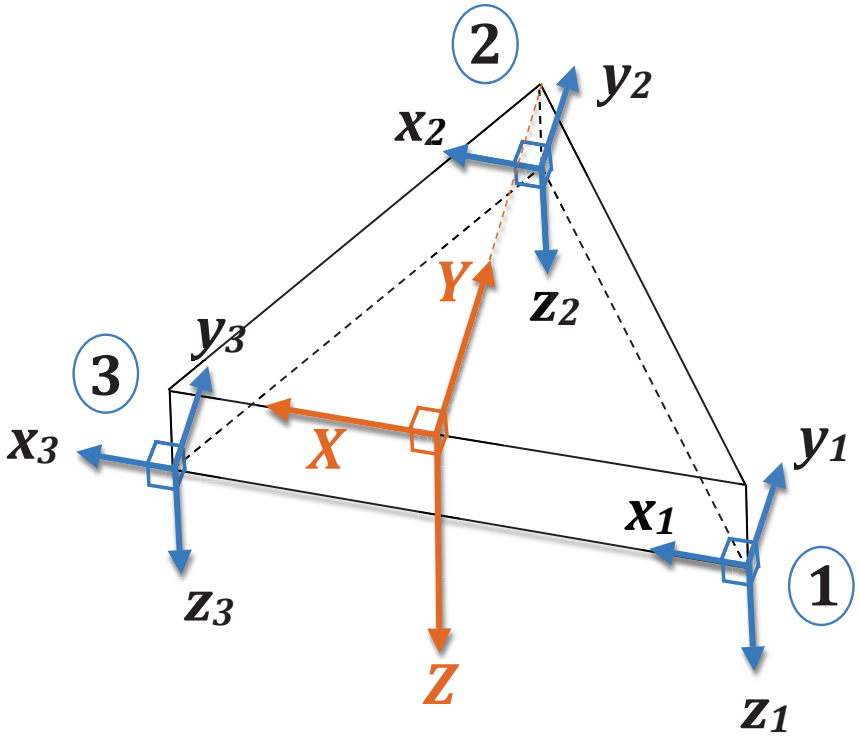


Figure C.1: Global and Local Coordinate Systems.

These local coordinate systems 1, 2, and 3 were excellent for choosing a random position on a droplet as all that was needed was choosing random polar(zenith) and azimuthal angles (as discussed in Appendix A) on the local coordinate systems.

Establishing a random velocity direction for a particle set at an arbitrary interface position, however, was more far more complex. To do this successfully, a normal-tangential type of coordinate system had to be set up at the point of particle emission or reflection from a droplet. Figure C.1 shows this type of system, where n refers to a direction normal to the droplet surface, tp refers to the direction that is tangential to the droplet surface and simultaneously parallel to the cold wall surface, and t refers to the other direction that is simply tangential to the droplet surface.

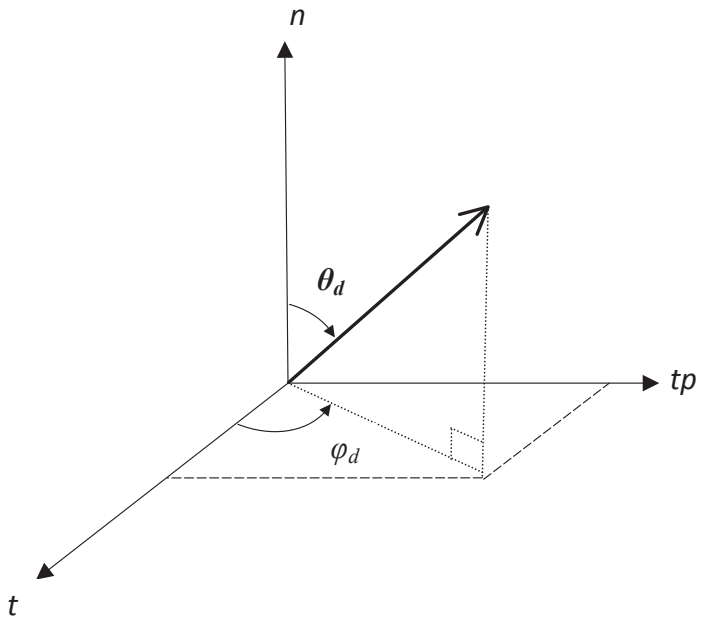


Figure C.2: Tangential coordinate system

The type of coordinate system shown by Figure C.2 was positioned at a droplet surface anytime a particle had to be released from any one of the droplet surfaces, whether it was from a reflection or droplet particle emission. Thus, it was convenient to establish coordinates and nomenclature corresponding to the normal and tangential directions. In this way, with positive n pointing away from a droplet surface, directions for particle velocities were chosen in the positive n orientation with any t and tp direction. This was established by choosing a random azimulthal angle φ_d between 0° and 360°, and a random polar angle θ_d between 0° and 90° sampled from a uniform distribution.

Eventually, these local coordinates had to be converted to the global coordinate system, which implied that, depending on which droplet a particle was emitted from, multiple coordinate rotations had to be performed to convert the tangential and normal coordinates to global X, Y, and Z coordinates. For any direction chosen on any drop, the conversion to global coordinates was done in 3 rotations.

- Figures C.3 through C.5 below demonstrate the rotations needed for a random velocity direction chosen at any point from the surface of droplet 1.
- Figures C.6 through C.8 below demonstrate the rotations needed for a random velocity direction chosen at any point from the surface of droplet 2.
- Figures C.9 through C.11 below demonstrate the rotations needed for a random veloc-

ity direction chosen at any point from the surface of droplet 3.

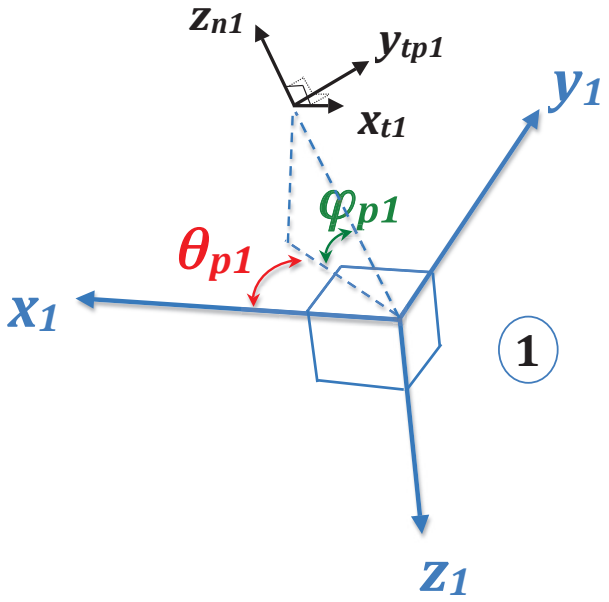


Figure C.3: Coordinates with respect to droplet 1, first position.

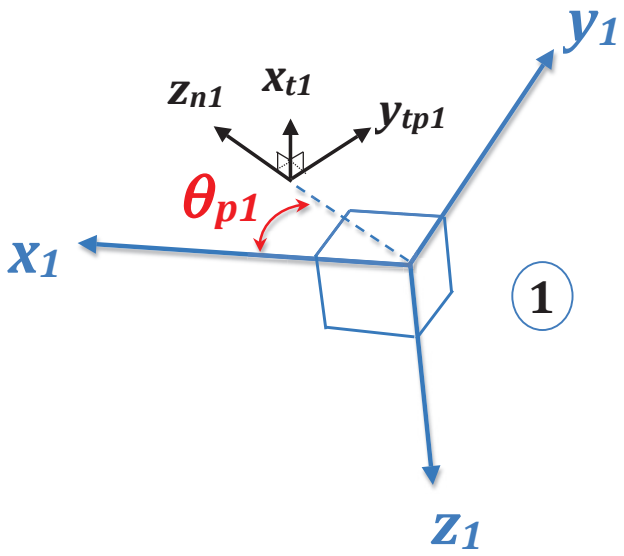


Figure C.4: Coordinates with respect to droplet 1, second position.

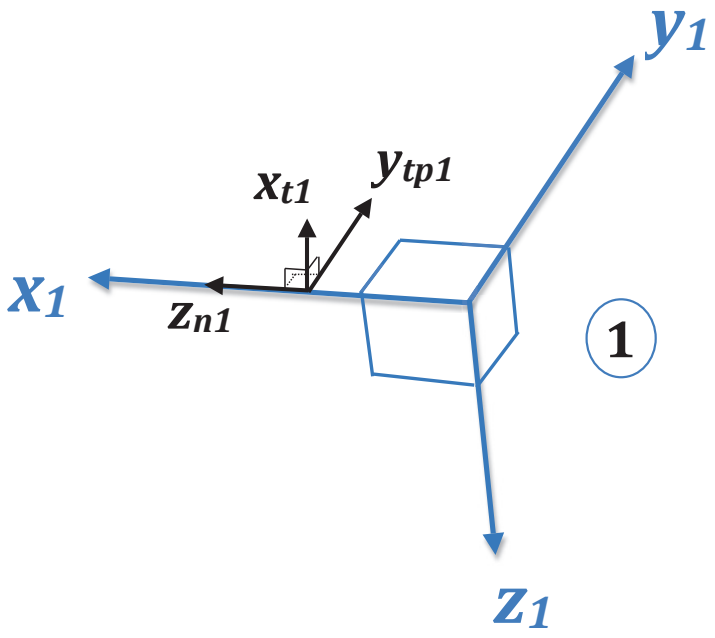


Figure C.5: Coordinates with respect to droplet 1, third position

The rotations for a particle released from droplet 1 to convert from the local tangential coordinate system to the global coordinate system were as follows (Figures C.3 through C.5):

1.) The tangential coordinates generated were rotated φ_{p1}° about the y_{tp1} axis.

2.) Those coordinates were then rotated θ_{p1}° about the x_{t1} axis.

3.) Then finally, the coordinates were rotated 90° about the y_{tp1} axis in the direction from x_{t1} towards z_{n1}.

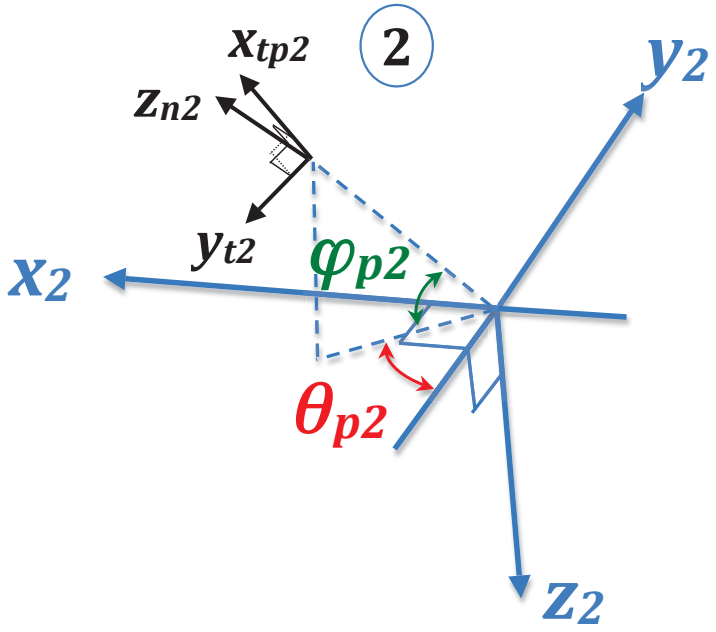


Figure C.6: Coordinates with respect to droplet 2, first position

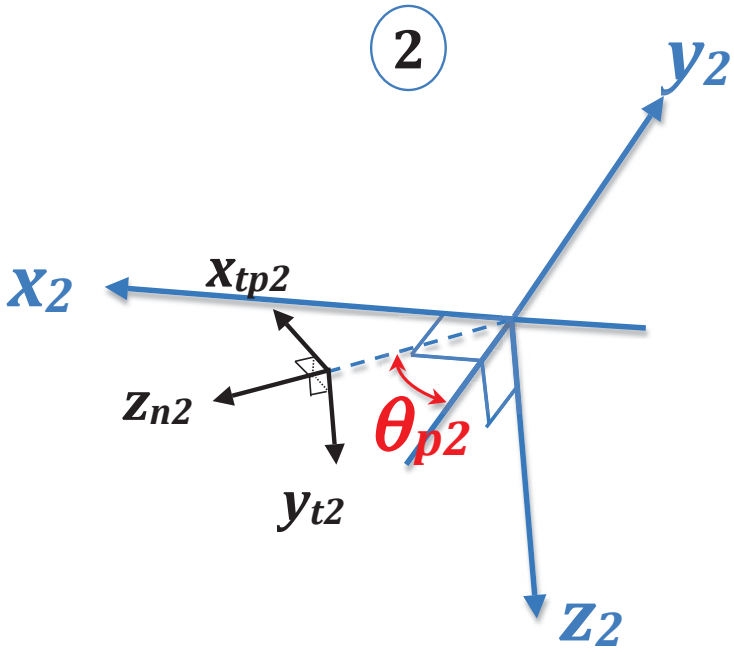


Figure C.7: Coordinates with respect to droplet 2, second position

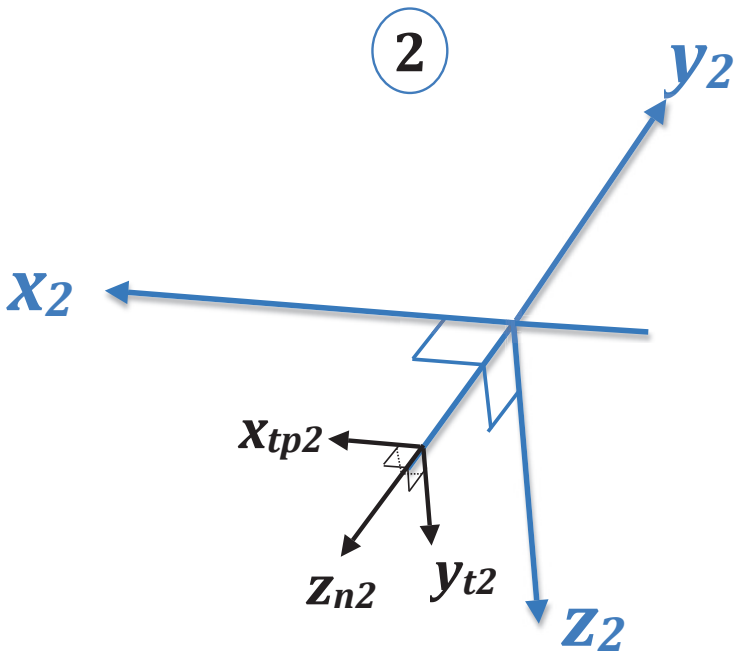


Figure C.8: Coordinates with respect to droplet 2, third position

The rotations for a particle released from droplet 2 to convert from the local tangential coordinate system to the global coordinate system were as follows (Figures C.6 through C.8):

1.) The tangential coordinates generated were rotated φ_{p2}° about the x_{tp2} axis.

2.) Those coordinates were then rotated θ_{p2}° about the y_{t2} axis.

3.) Then finally, the coordinates were rotated 90° about the x_{tp2} axis in the direction from z_{n2} towards y_{t2}.

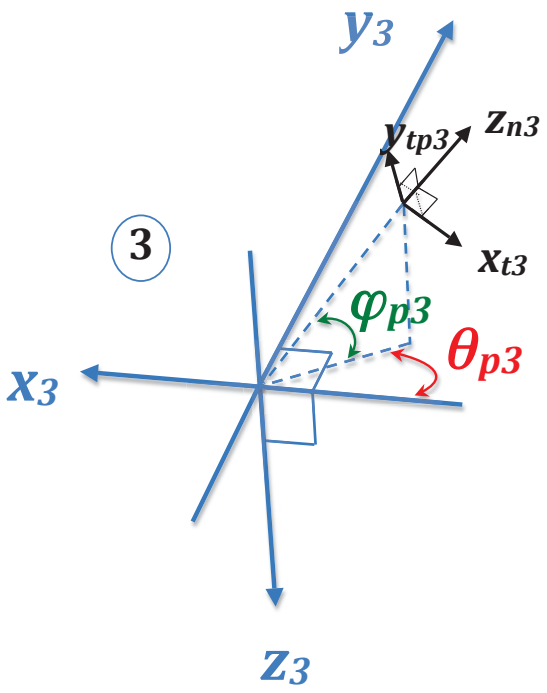


Figure C.9: Coordinates with respect to droplet 3, first position

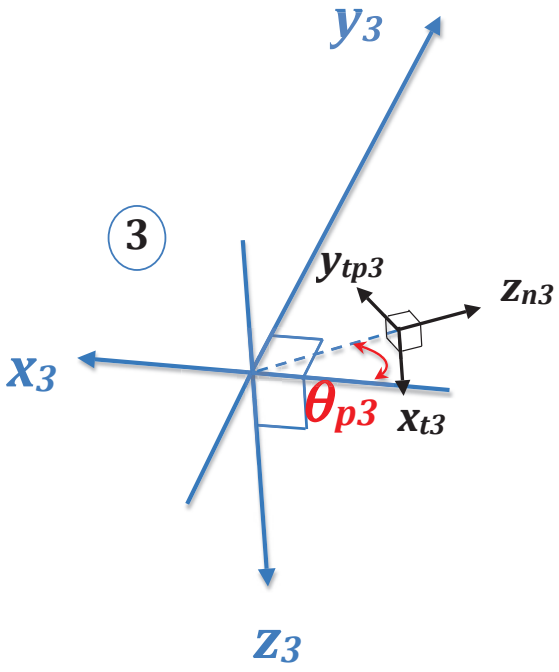


Figure C.10: Coordinates with respect to droplet 3, second position

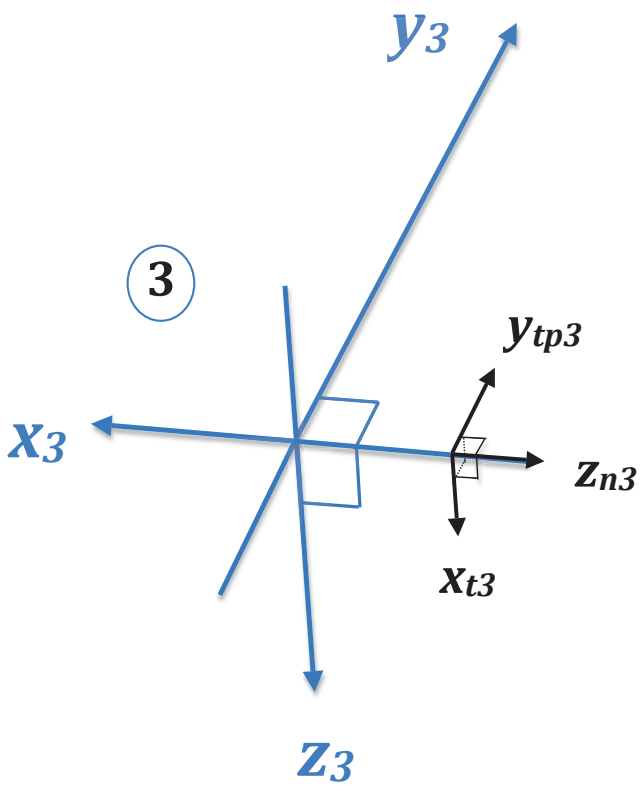


Figure C.11: Coordinates with respect to droplet 3, third position

The rotations for a particle released from droplet 3 to convert from the local tangential coordinate system to the global coordinate system were as follows (Figures C.9 through C.11):

1.) The tangential coordinates generated were rotated φ_{p3}° about the y_{tp3} axis.

2.) Those coordinates were then rotated θ_{p3}° about the x_{t3} axis.

3.) Then finally, the coordinates were rotated 90° about the y_{tp3} axis in the direction from z_{n3} towards x_{t3}

C.3 Specular Reflections

Specular reflections were performed and handled in the same way as described in Appendix A.

Appendix D

MATLAB Code for the Approximation Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This file draws lines of constant contact angle on  pressure figures for%
%the Approximation Model on a Droplet Cluster
%
%H. Mendoza, S. Beaini, and V.Carey 2011
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all
tic

cont_ang=[70 90 110]; %contact angle degrees
Press=[1]; %Pressure [atm]
subcool= [3];  %[3 5 7];amount of SubCooling degrees C
sddr = [0.4];  % set s/dd ratio...can't do sddr>2 because of F limitations
max_it=15; %maximum number of iterations for
ex=.2;
AC=1;
acs=num2str(AC);
pred=1;
for si=1:length(subcool)
    sc=subcool(si);
for l=1:length(Press)%loops through the different pressures
    P=Press(l);

    for k=1:length(cont_ang)%makes one figure with three lines
        ca=cont_ang(k); %contact angle

        %d = linspace(0.000000001,.000001,1000); % 1[micro-meter] to 1 [nano-
meter]
        %d=linspace(100e-6,1e-9,100000); %100 microns (visible range) down to
when it becomes zero
        d1=linspace(10000e-6,20e-6,1000);
        d2=linspace(20e-6,1e-9,10000);
        d=[d1 d2];

        for j=1:length(d)
            dd=d(j);



            s = sddr*dd; % compute droplet separation, s is defined by the
diamter of curvature...consistent with the monte carlo calculations

            rd = dd/2.; % compute droplet radius

            %Sf = 2.0*3.14159*dd;   %Graham and Griffith shape factor for
conduction in droplet
            Sf = (1/2)*DropSF(ca, rd);
            Adi=(rd^2)*pi*(1-cosd(ca));%Adi = .25*(2*pi*(rd)^2*(-
cosd(90+ca)+1));%0.25*3.14159*dd*dd; (pi*d^2 represents area of whole
sphere)%surface area of three droplet interfaces
            Adel = 0.25*sqrt(3.)*(s+(dd*sind(ca)) )*(s+(dd*sind(ca)) );
```

```
%surface areas of upper triangle
            del = rd*(1-cosd(ca) );  %vertical distance between cold surface
and triangle
            Aw=Adel-0.5*pi*(rd*sind(ca))^2; %Area in between the
            % % %     lam = 5.85e-07;  % mean free path in meters



            %Fdeld = Fraction of molecule flux from top triangle that hit a
droplet
            %Fdelw = Fraction of molecule flux from top triangle that hits
bottom wall triangle and returns to ambient
            %Fid = Fraction of emitted from interface that hits other
droplets
            %Fii= Fraction of molecule flux emitted from interface that hits
its own interface
            if round(ca)==70
                Fdeld= exp(0.1654*(sddr) ^2- 1.1634*(sddr)  -0.0116);%check
                Fdelw = 1.-Fdeld;
                Fid= exp(-0.34*(sddr)^5 + 1.838*(sddr)^4 - 3.948*(sddr)^3 +
4.543*(sddr)^2 - 3.9191*(sddr) - 1.1953);%check
                Fii = exp(0.4444*(sddr)^5 - 2.9489*(sddr)^4 + 7.2561*(sddr)^3
- 7.857*(sddr)^2 + 2.9295*(sddr) - 3.7526);
            elseif round(ca)==80
                Fdeld= exp(0.1947*(sddr)^5 - 0.9935*(sddr)^4 +
1.8117*(sddr)^3 - 1.2842*(sddr)^2 - 0.591*(sddr) - 0.0155);
                Fdelw = 1.-Fdeld;
                Fid= exp(0.0635*(sddr)^5 - 0.0924*(sddr)^4 - 0.622*(sddr)^3 +
2.0767*(sddr)^2 - 3.161*(sddr) - 1.0259);
                Fii = exp(-0.493*(sddr)^6 + 3.5266*(sddr)^5 - 10.472*(sddr)^4
+ 16.313*(sddr)^3 - 13.533*(sddr)^2 + 4.8297*(sddr) - 3.7338);
            elseif round(ca)==90
                Fdeld= exp(0.1292*(sddr)^5 - 0.716*(sddr)^4 + 1.4642*(sddr)^3
- 1.2421*(sddr)^2 - 0.3953*(sddr) - 0.0028);
                Fdelw = 1.-Fdeld;
                Fid= exp(-0.0091*(sddr)^6 + 0.0631*(sddr)^5 - 0.0756*(sddr)^4
- 0.447*(sddr)^3 + 1.704*(sddr)^2 - 2.9651*(sddr) - 0.8362);
                Fii = exp(-0.1082*(sddr)^6 + 1.0937*(sddr)^5 -
4.3736*(sddr)^4 + 8.768*(sddr)^3 - 9.0585*(sddr)^2 + 4.0275*(sddr) - 3.5404);
            elseif round(ca)==110
                Fdeld= exp(0.0727*(sddr)^5 - 0.502*(sddr)^4 + 1.3*(sddr)^3 -
1.4973*(sddr)^2 + 0.0664*(sddr) - 3E-05);%check
                Fdelw = 1.-Fdeld;
                Fid= exp(-0.0896*(sddr)^5 + 0.5482*(sddr)^4 - 1.3885*(sddr)^3
+ 2.158*(sddr)^2 - 2.935*(sddr) - 0.4276);%Check
                Fii = exp(-1.5165*(sddr)^6 + 10.742*(sddr)^5 -
29.868*(sddr)^4 + 41.299*(sddr)^3- 29.634*(sddr)^2 + 10.124*(sddr) - 3.7119);
            elseif round(ca)==100
                Fdeld90= exp(0.1292*(sddr)^5 - 0.716*(sddr)^4 +
1.4642*(sddr)^3 - 1.2421*(sddr)^2 - 0.3953*(sddr) - 0.0028);
                Fdelw90 = 1.-Fdeld90;
                Fid90= exp(-0.0091*(sddr)^6 + 0.0631*(sddr)^5 -
0.0756*(sddr)^4 - 0.447*(sddr)^3 + 1.704*(sddr)^2 - 2.9651*(sddr) - 0.8362);
                Fii90 = exp(-0.1082*(sddr)^6 + 1.0937*(sddr)^5 -
4.3736*(sddr)^4 + 8.768*(sddr)^3 - 9.0585*(sddr)^2 + 4.0275*(sddr) - 3.5404);
```

```
                Fdeld110= exp(0.0727*(sddr)^5 - 0.502*(sddr)^4 + 1.3*(sddr)^3
- 1.4973*(sddr)^2 + 0.0664*(sddr) - 3E-05);%check
                Fdelw110 = 1.-Fdeld110;
                Fid110= exp(-0.0896*(sddr)^5 + 0.5482*(sddr)^4 -
1.3885*(sddr)^3 + 2.158*(sddr)^2 - 2.935*(sddr) - 0.4276);%Check
                Fii110 = exp(-1.5165*(sddr)^6 + 10.742*(sddr)^5 -
29.868*(sddr)^4 + 41.299*(sddr)^3- 29.634*(sddr)^2 + 10.124*(sddr) - 3.7119);

                Fdeld=mean(Fdeld90+Fdeld110)/2;
                Fdelw=mean(Fdelw90+Fdelw110)/2;
                Fid=(Fid90+Fid110)/2;
                Fii=(Fii90+Fii110)/2;
            end
            Fdeld=pred*Fdeld;
            Fdelw=pred*Fdelw;
            Fid=pred*Fid;
            Fii=pred*Fii;


            NA = 6.02e+26;  %Avogadro's number molecules/KMOL
            kB = 1.38e-23; %Boltzman constant
            Mw = 18.0;  %molecular mass of water kg/kmol
            Pinf_xsteam = P*1.01325; %[bar] (1.01325[bar]=1atm)
            Pinf=Pinf_xsteam*10^5; %[Pa]

            %anything with a suffix of xsteam is to use with "Xsteam
function"
            Tinf_xsteam=XSteam('Tsat_P',Pinf_xsteam); % degrees C
            Tw_xsteam=Tinf_xsteam - sc;%degrees C
            Tinf=Tinf_xsteam + 273.2;%Kelvin
            Tw=Tinf-sc;% Kelvin

            % % %     Tinf = 100 +273.2;   %steam ambient temperature
            % % %     Tw = 97. +273.2; %wall temperature

            %Start mean free path
            D=3.61e-10; %molecular diameter for water (should cite!)
            lam=kB*Tinf/(sqrt(2)*pi*Pinf*D^2);

            %End mean free path
            Kn = lam/del;  %droplet Knudsen number

            %flux across suface in ambient (valid at triangle)
            jinf = 0.25*(Pinf/(kB*Tinf))*((8.*NA*kB*Tinf/(3.14159*Mw))^0.5);
            %% properties from function (lagrange interpolation)...evaluated
at Tinf??
            %% half way in between??
            Te_xsteam= (Tinf_xsteam + Tw_xsteam)/2; %average temp at which
properties can be EEEvaluated
            R=NA*kB/Mw; % gas constant
            vl=XSteam('vL_T',Te_xsteam);%...vl = 0.001; %cu meters per kg
            vv = XSteam('vV_T',Te_xsteam); %cu meters per kg
```

```
            %% surface tension correction
            sig = XSteam('st_T',Te_xsteam); %N/m
            tml=0.157e-9; %tolman length [m]
            sig=sig*((1+ (2*tml/rd))^-1); %the main correction


            hlv = (XSteam('hV_T',Te_xsteam)-XSteam('hL_T',Te_xsteam))*1000;
%J/kg
            kl = XSteam('tcL_T',Te_xsteam); %W/mK
           % dPdT = hlv/(Tinf*vv); %Pa per K clapeyron clausius
            %%
            %guess Ti interface temp difference delt = Tinf - Ti
            delt = 0.1; %  initial value of Tinf - Ti

            %% begin delt = Tinf - Ti iterative loop here
[[[[[[[[[[[[[[[[[[[[[[[[[

            for i = 1:1:max_it;
                Ti = Tinf + delt; % interface temperature incremented
                %PsatTi = Pinf + dPdT*(Ti - Tinf); %Pa
                PsatTi = XSteam('psat_T',Ti-273)*10^5;% the 273 is subracted
because Xsteam takes in degrees Celsius, and it is multiplied by 10^5 because
xsteam outputs pressure in bar, but we want Pa
                Pvi = PsatTi*exp(2.*vl*sig/(rd*R*Ti));
                % ji = flux emitted from droplet surface at tempertaure Ti
                ji = 0.25*(Pvi/(kB*Ti))*((8.*NA*kB*Ti/(3.14159*Mw))^0.5);

                % space conditions between droplets
                % energy balance dictates mean temperature in space bewteen
droplets

                %pressure in space between droplet equal ambinet pressure
                Ps = Pinf; %ambient pressure approximation

                 %mass conservation dicates js, m denotes by mass
                jsm= (jinf*Adel + ji*Adi)/(Adel + Adi);

                % energy balance dictates mean temperature in space bewteen
droplets
                Ts = (ji*Adi*Ti + jinf*Adel*Tinf +Aw*jsm*Tw)/(
jsm*(Adel+Adi+Aw) );

                % js = the molecular flux incident on interface from space
between droplets
                % in the continuum limit
                js = 0.25*(Ps/(kB*Ts))*((8.*NA*kB*Ts/(3.14159*Mw))^0.5);
                % jbi = the molecular flux incident on interface in ballistic
limit
                jbi = (jinf*Adel*Fdeld/Adi) + ji*Fid + ji*Fii;
                %jds = flux indicent on droplet surface as weighted average
of continuum
                %and ballistic values
                jds = js*(1.-exp(-ex*del/lam)) + jbi*exp(-ex*del/lam);
```

```
                %second term is incident flux from the ambient plus added
incident flux
                %from other droplets

                % net heat transfer  dictated by difference in molecular
fluxes at
                % interface
                qdotcc = AC*(Mw*hlv/NA)*(jds - ji)*Adi;

                %interface heat echange must equal heat transfer through
droplets
                %Err = difference between interface heat transfer and droplet
conduction
                % iterate delt = Tinf-Ti until fErr = Err/qdotcc < 0.0001









                Err = qdotcc - (Ti - Tw)*(kl*Sf); %net in from condensation
should equal to net going out due to conduction.  however, here we set it
equal to the error.
                fErr1 = Err/qdotcc; %1st method of trying (carey's)... Cannot
                ...use it, because at some point(FOR REALLY SMALL RADII)
qdotcc becomes so small
                ...(very few net particles in) that fErr1 does not converge
                ...in the if abs(ferr)<0.0001 statement below

                Err_norm=(Tinf-Tw)*(kl*Sf);
                %Err_norm = qdotcc - (Ti - Tw)*(kl*Sf);
                fErr=Err/Err_norm; %current method (mendoza)...this one works
                ...better because Err_norm is a good meter because Tinf isn't
changing
                ...as Ti does, but it does take into consideration the radius
and the
                ...contact angle


                if abs(fErr)<0.0001
                    Ti(j) = Ti;
                    TidegC = Ti-273.2;
                    break;
                end

                %small increment in delt to determine Newton-Raphson
correction
                Ti = Tinf + 1.005*delt; % interface temperature
                PsatTi = XSteam('psat_T',Ti-273)*10^5;
                %PsatTi = Pinf + dPdT*(Ti - Tinf); %Pa
                Pvi = PsatTi*exp(2.*vl*sig/(rd*R*Ti));
                ji = 0.25*(Pvi/(kB*Ti))*((8.*NA*kB*Ti/(3.14159*Mw))^0.5);
```

```
                %Ps = ambient pressure approximation
                Ps = Pinf;

                 %mass conservation dicates js
                jsm= (jinf*Adel + ji*Adi)/(Adel + Adi);

                % space conditions between droplets
                % energy balance dictates mean temperature in space bewteen
droplets
                Ts = (ji*Adi*Ti + jinf*Adel*Tinf +Aw*jsm*Tw)/(
jsm*(Adel+Adi+Aw) );


                js = 0.25*(Ps/(kB*Ts))*((8.*NA*kB*Ts/(3.14159*Mw))^0.5);
                jbi = (jinf*Adel*Fdeld/Adi) + ji*Fid + ji*Fii;
                jds = js*(1.-exp(-ex*del/lam)) + jbi*exp(-ex*del/lam);

                qdotcc = AC*(Mw*hlv/NA)*(jds - ji)*Adi;

                Err2 = qdotcc - (Ti - Tw)*(kl*Sf);

                dErrdelt = (Err2-Err)/(0.005*delt);
                delt = delt - Err/dErrdelt;

                if i==max_it-1
                    i
                    j
                    k
                    l
                    si
                    error('Ti does not converge')
                end

            end

            %% end Ti loop here ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
            jbi_j(j)= jbi;
            jds_j(j)= jds;
            js_j(j)= js;
            ji_j(j)= ji;


            qdotcc(j) = AC*(Mw*hlv/NA)*(jds - ji)*Adi;
            h(j) = qdotcc(j)/(Adel*(Tinf-Tw));
            fcond(j) = (Ti(j)-Tw)/(Tinf-Tw);
            Adi = Adi;
            Adel = Adel;


            if h(j)<0
                h(j)=[];
                d(j:end)=[];
```

```
                qdotcc(j)=[];
                jbi_j(j)= [];
                jds_j(j)= [];
                js_j(j)= [];
                ji_j(j)= [];
                break
            end

        end
        h_coef{k}=h;
        diam{k}=d;

        %% end diameter loop above here


        %Dmin(l,k)=4*vl*sig*Tw/(hlv*(Tinf-Tw));
        [hmax(k),hmaxi(k) ]=max(h);%hmax is the value, hmaxi is the indice
        d_hmax(k)=d(hmaxi(k)) ;%diameter corresponding to hmax


        cas=num2str(ca); %contact angle in string format
        lgndstr(k)= {['\it\theta = \rm', cas,char(176)]};
        %lgndstr(k,:)= ['\theta_c_a= ', Ps,'[ATM]'];


        toc
        clear  h qdotcc Ti fcond



    end
    %%end the contact angle loop above here
    %% creating the heat tranfser coefficient figure
    dmax=linspace(max(d),min(d),100);
    hmaxx=(1/vv)*hlv*sqrt(R*Tinf/(2*pi));
    hmaxvec=hmaxx*(ones(size(dmax)));
    figure()
    lntp={'-',':','-.','--','-',':','-.','--',...
        '-',':','-.','--','-',':','-.','--',...
        '-',':','-.','--','-',':','-.','--',...
        '-',':','-.','--','-',':','-.','--'};%line type string
    cstr='gbrcmykbgrcmyk';%color string

    for ii=1:length(cont_ang)

        if ii==1
            loglog(diam{ii},h_coef{ii},'Color',[0,.65,0],'LineWidth',2)
            hold on
        else
            loglog(diam{ii},h_coef{ii},[cstr(ii),lntp{1}],'LineWidth',2)
            hold on
        end
```

```
%         plot(d_hmax,hmax,'kx','LineWidth',2)
%         hold on
    end
    plot(dmax,hmaxvec,'k')
    xlabel('Droplet Diameter (m)')
    ylabel('\it h  \rm( W/m^2 K )')%h=heat transfer coefficient
    Ps= num2str(101*P); %Pressure in string format in kPa









    title(['\it T_\infty \rm- \itT_w = \rm',num2str(sc),' (K), ','\it
P_\infty \rm = ',Ps,...
        ' (kPa), \it s/d\rm = ',num2str(sddr),', \it \sigma \rm= ',acs])
    legend(lgndstr,'Location','NorthEast')
    %loglog(d_hmax,hmax,'k.','LineWidth',5)
    xlim([1e-8 max(d)])
    ylim([10^2 10^8])
    set(gca,'XTick',[10^-8 10^-7 10^-6 10^-5 10^-4 10^-3 10^-2])
    set(gca,'YTick',[10^2 10^3 10^4 10^5 10^6 10^7 10^8])


end

%%end the pressure loop above here

end

%%end subcool loop above here
```

Appendix E

C Code for DSMC Model on a Single Droplet

```
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 ::
 ::
 ::  A protype zero-bulk-flow DSMC program in C.
 ::  Boundary conditions in this version set to
 ::  compute transport to a SINGLE HEMISPHERICAL DROPLET
 ::  condensing on a cold wall with variable contact angle
 ::
 :: This file specifically simulates for a 400nm droplet, 90 degree contact
angle
 ::
 ::
 ::  H. Mendoza 2012, based on V. Carey  1/95, 9/97, 7/99.
 ::

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*
/


#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>


/* BBBBBBBBBBB  Definition of constants */

#define NCX       32          /* number of cells in x direction */
#define NCY       32          /* number of cells in y direction */
#define NCZ       32          /* number of cells in z direction */
#define NCELLS  32770              /* total number of cells */

#define NPRTMAX 240000        /* maximum number of particles */
#define INPRTNO 197000        /* initial number of particles */

#define NSTEPS    24000       /* number of time steps to be done */
#define INTST     1000        /* no. of steps to Tsp iteration */
#define DT  6.0e-11     /* time step in seconds */
#define L_CELL    5.0e-08     /* length of cell in meters */
#define RSP 4.0e-07           /* sphere radius in meters */
#define PRESS     101.325e+03 /* ambient pressure Pa */
#define PARTRAT   204.95      /* number of molecules per particle */



#define RANMAX  2147483647.    /* max value returned by random() */

#define ACCOM     1.0   /* droplet surface accommodation coeficent */
#define WALL_ACCOM 1.0  //accomodation coeficient for condensing wall
#define RADFLUX   0.0         /* net radiation flux to drop W/sq m */
#define CONCA     1           /* ambient molar concentration of
species 1 (water) */
#define MASSA     2.99e-26    /* mass of species 1 molecule kg */
#define MASSB     6.64e-26    /* mass of species 2 molecule kg */
#define DEGFREB 3.           //Degrees of freedom of species B
#define MDA 1000.           /* mass density of liq species 1000 kg/cu m
```

```
#define DMA 4.51e-10    /* effective diameter of species 1
molecule meters */
#define DMB 3.62e-10    /* effective diameter of species 2
molecule meters */
#define IZROT     0.2         /* value of 1/Zrot */
#define CA  0.07583     /* surface tension constant N/m  */
#define CB  0.1477          /* surface tension constant N/mK */
#define GAM 1.566e-07   /* Tolman length scale parameter kg/sq m  */
#define KB  1.38e-23    /* Boltzmann constant J/K */
#define NA  6.02e+26    /* Avogadro's number molecules/kmol */
#define PI  3.14159
#define EPS 1.00e-20    /* a small but non-zero number */
#define PINT      1000        /* time steps in print interval */
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Definition of external varibles and functions */
FILE *fpo;  /* uses indirection to define pointer fpo to a file */
FILE *fopen(); /* function used to define files */
int nprt, lost, again, nopairs, nocoll, nocolrot, nncon;
int pt1[NPRTMAX], pt2[NPRTMAX], sampsize[NCELLS];
int losscheck, spherehits, sphemits, collcount, nconsum;
double fractbin[15], mnp[NCELLS];
double wfluxres, afluxres1, afluxres2, elost, egain, degain;
double xmax, ymax, zmax;
double freelam, den, freeparden;
double totfract1, Einlower, Ecgain, sepsum;
double Eg, Tsp, EvapT, delt, errsum;
double ca, ca4,L_CELLz,Tw,sc, TAMB, TSP, EVAPPM;
int wallhitcheck, scatter, check;
double uvelocitypre,vvelocitypre,uvelpost,vvelpost,thisphi,thisr,thiscT;
struct part {
    int kind;
    int cellno;
    double mass;
    double x;
    double y;
    double z;
    double u;
    double v;
    double w;
    double erot;
} prt[NPRTMAX];
struct cellstat {
    int countkind1;
    int countkind2;
    double temp;
    double molconc1;
    double umean;
    double etrsum;
    double erotsum;
    double usum;
} cell[NCELLS];
void initcon();
void advance(int k);
void chkspace(int n, int k);
```

```
void loseprt(int n);
void ambemit();
void dropemit(int n);
void wallhit(int n, double m);
void collect();
void swap(int n, int m);
void collide();
void statpr();
void statcell(int n);
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* FUNCTIONS THAT HECTOR ADDED */
double density_liquid(double n);
double enthalpy(double x);
double pressure(double x);
double specific_volume_vapor(double x);
double surface_tension(double x);
double thermal_cond_liquid(double x);
double temperature(double x);
double ar_prodsum(double arg[], int arg_length, int flag);
void ebe(double a1[], double a2[],double *a3,int alen);
double DropSF(double phi, double radius);
long factorial(int n);


/*EEEEEEEEEEE FUNCTIONS THAT HECTOR ADDED EEEEEEEEEEEEE*/




/* BBBBBBBBBBB  MAIN PROGRAM OF PARTICLE SIMULATION CODE */
main()
{

    int i, ndum, iprof, j, jmax, k, plim, ip;
    double xdum, rm12, D12, flam1, flam2, drdt;
    double collcount_time,kTCR,kTCRA,kTCRB,kTCRAB,sigab,
rmab,VbarA,VbarAB,VbarB, nA, nB, octvol, h,Aeff;

    fpo=fopen("outDSMC_400_ca90","w"); /* open file outDSMC to write
                                          output in fpo is pointer to this
file..sdwc =single droplet wall condensation*/
    /* Note that this section are all new additions */

    TAMB= temperature(PRESS);
    sc=3;//amount of subcooling in [K]
    Tw=TAMB-sc;//wall temperature
    TSP=Tw+sc/2.;//just a guess
    EVAPPM=enthalpy(TSP)*MASSA; // J/molecule-K



    ca= 90;//degrees
    ca=ca*PI/180;//converted to radians
    ca4= acos(cos(ca)/4);//This is the contact angle for the r=4rd sphere
```

```
//lcel=L_CELL;
L_CELLz= L_CELL*(1-cos(ca4));//ca4 is 100% correct...do the geometry!!


/* End section of all New Additions */


/* print-out set constants */

xdum = TAMB;
fprintf(fpo,"\n   TAMB = %8.3e", xdum);
xdum = TSP;
fprintf(fpo,"\n   TSP = %8.3e", xdum);
xdum = PRESS;
fprintf(fpo,"\n   PRESS = %8.3e", xdum);
xdum = CONCA;
fprintf(fpo,"\n   CONCA = %8.5e", xdum);
xdum = DT;
fprintf(fpo,"\n   DT = %8.3e", xdum);
ndum = INTST;
fprintf(fpo,"\n   INTST = %d", ndum);
ndum = NCX;
fprintf(fpo,"\n   NCX = %d", ndum);
ndum = NCY;
fprintf(fpo,"\n   NCY = %d", ndum);
ndum = NCZ;
fprintf(fpo,"\n   NCZ = %d", ndum);
ndum = NPRTMAX;
fprintf(fpo,"\n   NPRTMAX = %d", ndum);
xdum = PARTRAT;
fprintf(fpo,"\n   PARTRAT = %8.3e", xdum);
xdum = MASSA;
fprintf(fpo,"\n   MASSA = %8.3e", xdum);
xdum = MASSB;
fprintf(fpo,"\n   MASSB = %8.3e", xdum);
xdum = EVAPPM;
fprintf(fpo,"\n   EVAPPM = %8.3e", xdum);
xdum = DMA;
fprintf(fpo,"\n   DMA = %8.3e", xdum);
xdum = DMB;
fprintf(fpo,"\n   DMB = %8.3e", xdum);
xdum = MDA;
fprintf(fpo,"\n   MDA = %8.3e", xdum);
xdum = IZROT;
fprintf(fpo,"\n   IZROT = %8.3e", xdum);
xdum = L_CELL;
fprintf(fpo,"\n   L_CELL = %8.3e", xdum);
xdum = RSP;
fprintf(fpo,"\n   RSP = %8.3e", xdum);
xdum = ACCOM;
fprintf(fpo,"\n   ACCOM = %8.3e", xdum);
xdum = RADFLUX;
fprintf(fpo,"\n   RADFLUX = %8.3e", xdum);
xdum = CA;
fprintf(fpo,"\n   CA = %8.3e", xdum);
xdum = CB;
```

```
    fprintf(fpo,"\n   CB = %8.3e", xdum);
    xdum = GAM;
    fprintf(fpo,"\n   GAM = %8.3e", xdum);


    srandom(19);              /* initialize seed for random number generator
*/

    nprt = INPRTNO;           /* set initial particle count */
    wfluxres = 0.0;           /* initialize accumulators */
     nconsum = 0;
     errsum = 0.;
    afluxres1 = 0.0;
    afluxres2 = 0.0;




    xmax = NCX*L_CELL;
    ymax = NCY*L_CELL;
    zmax = NCZ*L_CELLz;//should still be equal to
     degain = RADFLUX*DT*(PI/2.)*RSP*RSP;
    lost = 0;
    again = 0;
     Ecgain = 0.0;
     Einlower = 0.;
     Eg = 0.0;
     sphemits = 0;
     spherehits = 0;
     nocoll = 0;
     collcount = 0;
     nocolrot = 0;
     for (i=0; i< NCELLS; i++) {
        /* set arrays to zero */
        cell[i].countkind1 = 0;
        cell[i].countkind2 = 0;
        cell[i].etrsum = 0.;
        cell[i].erotsum = 0.;
        cell[i].usum = 0.;
     }
     Tsp = TSP;
    EvapT = EVAPPM;
    den = PRESS/(KB*TAMB);    /* molecular density per cu meter */
     D12 = (DMA + DMB)/2.;
     rm12 = MASSA/MASSB;
     flam1 = 1./(1.414*den* CONCA*DMA*DMA
                 + sqrt(1. + rm12)*den*(1.- CONCA)*D12*D12);
     flam1 = flam1/PI;
     flam2 = 1./(1.414*den*(1. - CONCA)*DMB*DMB
                 + sqrt(1. + (1./rm12))*den*CONCA*D12*D12);
     flam2 = flam2/PI;
    /* ambient mean free path */
    freelam = CONCA*flam1 + (1. - CONCA)*flam2;
    fprintf(fpo,"\n freelam (m) = %8.3e", freelam);

    /* ambient particles per cell */
```

```
    freeparden = den*L_CELLz*L_CELL*L_CELL/PARTRAT;
     jmax = NCX*NCY;

    initcon();          /* fill particle structures */


    //printf("failed");
    //exit(EXIT_FAILURE);
     iprof = PINT;
     plim = PINT - 1;
    k=0;                        /* loop to step simulation */


    //calculation of kinetic theory collision rate
    sigab=(DMA/2)+(DMB/2);
    rmab= MASSA*MASSB/(MASSA+MASSB);//reduced mass
    VbarAB=sqrt(8*KB*TAMB/(PI*rmab));
    VbarA=sqrt(8*KB*TAMB/(PI*MASSA));
    VbarB=sqrt(8*KB*TAMB/(PI*MASSB));


    nA=(den*CONCA);
    nB=(den*(1-CONCA));
    octvol=PI/2*((4*RSP*4*RSP*4*RSP)-(RSP*RSP*RSP))*(1-cos(ca))/3; /*the /2
acccounts for the octant SINCE NOT DIVIDING BY FOUR GIVES FOR A HEMISPHERE*/
    kTCRAB=PI*nA*nB*sigab*sigab*VbarAB*octvol; /*Kinetic theory collision
rate[coll/s] one octant of a spherical shell*/

    kTCRA=sqrt(2)*nA*PI*DMA*DMA*VbarA*octvol;
    kTCRB=sqrt(2)*nB*PI*DMA*DMB*VbarB*octvol;
    kTCR=kTCRA+kTCRB+kTCRAB;


    while (k < NSTEPS) {            /* through time steps      */
        wallhitcheck=0;
        scatter=0;
        check=0;
      iprof++;
      nncon = 0;
        elost = 0.0;
        egain = 0.0;
      delt = ((double) (k + 1))*DT;

        statcell(k);    /* compile cell statistics */
        advance(k);      /* move particles one time step */
        ambemit();       /* add particles from ambient */
        dropemit(k);     /* add particles emitted from drop */
        collect();       /* collect candidate collision pairs */
        collide();       /* execute collisions */
        //exit(EXIT_FAILURE);

        if (iprof > plim){
            iprof -= PINT;
            fprintf(fpo,"\n step = %d done, nocoll = %d", k, nocoll);
            fprintf(fpo,"\n            nocolrot = %d", nocolrot);
            fprintf(fpo,"\n            nprt = %d", nprt);
```

```
                fprintf(fpo,"\n Ecgain = %8.3e", Ecgain);
                fprintf(fpo,"\n errsum = %8.3e", errsum);
            fprintf(fpo,"\n spherehits = %d ", spherehits);
            fprintf(fpo,"\n sphemits = %d ", sphemits);
                fprintf(fpo,"\n Eg = %8.3e", Eg);
            fprintf(fpo,"\n nconsum = %d ", nconsum);
            fprintf(fpo,"\n collcount = %d ", collcount);
                fprintf(fpo,"\n Tsp = %8.3e", Tsp);

            collcount_time=PARTRAT*collcount/delt; /*molecule
collisions/sec*/
            fprintf(fpo,"\n collcount_time = %8.3e ", collcount_time);
            fprintf(fpo,"\n  kinetic_collcount_time= %8.3e", kTCR);
            /* compute droplet growth rate */
            drdt = 2.*((double)(spherehits - sphemits))*PARTRAT;
            drdt *= MASSA/(PI*RSP*RSP*delt*MDA);
            drdt *= 1.0e+06;
                fprintf(fpo,"\n drdt (microns/sec) = %8.3e", drdt);

            //compute heat transfer coefficient
            Aeff= (1./4.)*PI*(4*RSP*sin(ca4))*(4*RSP*sin(ca4));
            h=(Einlower/delt)/(Aeff*(TAMB-Tw));
            fprintf(fpo,"\n Heat Transfer Coeff (W/m2-K) = %8.3e", h);



            /* print-out profiles for this time step */

            for (i=0; i < NCZ; i++){
                ip = i*jmax;
                xdum = zmax*(0.5 + ((double) i))/((double) NCZ);
                fprintf(fpo,"\n z = %8.3e cellden = %8.3e conc1 = %8.3e temp
= %8.3e",
                        xdum, mnp[ip], cell[ip].molconc1, cell[ip].temp);
            }
            for (i=0; i < NCZ; i++){
                ip = i*jmax + i;
                xdum = zmax*(0.5 + ((double) i))/((double) NCZ);
                xdum = 1.414*xdum;
                fprintf(fpo,"\n r = %8.3e cellden = %8.3e conc1 = %8.3e temp
= %8.3e",
                        xdum, mnp[ip], cell[ip].molconc1, cell[ip].temp);
            }
        }  /* end of intermittent write */
        k++;
    }   /* end of time step while loop */



    fclose(fpo);        /* close file outDSMC with pointer fpo */
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */
```

```
/* BBBBBBBBBBB  Function to initialize particle structures */
void initcon()          /* New-style definition of C function */
{
    int j, xc, yc, zc,rpcheck;
    double cT, phi, r, Rtest, eps, Ie;
    double costheta, sintheta, snx, sny, snz, rp,theta,d,R1;
    rpcheck=0;
    j=0;
    while (j < nprt) {
        prt[j].kind = (int) ((1. - CONCA) + 1.0
                             + ((double) random())/RANMAX);


        //random theta generation accounting for the contact angle
        theta = ca + 10;//just to initialize while loop
        if (ca <= PI/2.){
            while (theta > (ca4) ){
                costheta = 1. - ((double) random())/RANMAX;
                theta=acos(costheta);
            }
        }
        else /* (ca > PI/2)*/{
            while (theta > (ca)){
                costheta = 2*((double) random())/RANMAX-1;//...this should
sample between 0 and 180 degrees
                theta=acos(costheta);
            }
        }
        //random theta generation accounting for the contact angle

        /* randomly select entry
         point for new particle */
        rp = RSP + 0.1*L_CELL
        + (xmax - RSP - 0.1*L_CELL)*((double) random())/RANMAX;
        if (ca<=PI/2.){
            if (theta>ca){//remember that you're sweeping theta between 0 and
ca4, so all good
                d=RSP*cos(ca)/cos(theta);//hypotenuous
                rp=(4*RSP-0.1*L_CELL-d)*((double)
random())/RANMAX+d+0.1*L_CELL;
            }

        }

        if (ca>PI/2){
            if (theta>ca4){//remember that theta was only swept between 0 and
ca, so all good
                rpcheck=1;
                d=RSP*cos(ca)/cos(theta);//hypotenuous
                R1=((double) random())/RANMAX ;
                rp=(d-RSP)*R1+RSP;

                //rp=(d-RSP-0.1*L_CELL)*R1+RSP+0.1*L_CELL;
            }
        }
```

```
        //end random r generation accounting for contact angle



         sintheta = sqrt(1. - costheta*costheta);
         phi = 0.5*PI*((double) random())/RANMAX;
         snx = sintheta*cos(phi);
         sny = sintheta*sin(phi);
         snz = costheta;
        prt[j].x = rp*snx;
        prt[j].y = rp*sny;
        prt[j].z = rp*snz;



        xc = (int) (prt[j].x*NCX/xmax);
        yc = (int) (prt[j].y*NCY/ymax);
        zc = (int) ( (prt[j].z-RSP*cos(ca)) *NCZ/zmax);
        switch (prt[j].kind){
            case 1:  /* water */
                prt[j].mass = PARTRAT*MASSA;
                /* rotational energy for water */
                Rtest = 0.97*((double) random())/RANMAX;
                eps = 0.002;
                Ie = 0.0;
                while (Ie < Rtest) {
                    eps += 0.1;
                    Ie += (sqrt(eps - 0.1)*exp(-eps + 0.1)
                           + sqrt(eps)*exp(-eps))*0.05642;
                }
                prt[j].erot = PARTRAT*KB*TAMB*eps;
                break;
            case 2: /*argon */
                prt[j].mass = PARTRAT*MASSB;
                prt[j].erot = 0.0;
                break;
        }  /* end switch */
        cT = sqrt(PARTRAT*KB*TAMB/prt[j].mass);
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


        phi = 2.*PI*((double) random())/RANMAX;
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].u = cT*r*cos(phi);
        prt[j].v = cT*r*sin(phi);
        phi = 2.*PI*((double) random())/RANMAX;
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
```

```
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].w = cT*r*cos(phi);
        j++;
    }
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

/* BBBBBBBBBB  Function to advance particle to new location */
void advance(k)
{
    int j;
    j=0;
    while (j < nprt) {



        prt[j].x += prt[j].u * DT;
        prt[j].y += prt[j].v * DT;
        prt[j].z += prt[j].w * DT;
        losscheck = 1;
        chkspace(j,k);  /* see if it left cell grid or hit sphere */
        j += losscheck;
    }
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

/* BBBBBBBBBB  Function to check whether particle escapes and handle */
void chkspace(j,k)
{
     int  xc, yc, zc;
     double xold, yold, zold, A, B, C, D, dt1, dt2, dtmin, Epre;
     double R0, R1, R2, xi, yi, zi, cmag;
     double costheta, sintheta, phi, cn, cp1, cp2, snx, sny, snz;
     double a1, a2, snynz, b0, b1, b2, rsq, rgsq, dts;


    /* check if sphere is hit */
    dts=DT;
     xold = prt[j].x-prt[j].u*DT;
     yold = prt[j].y-prt[j].v*DT;
     zold = prt[j].z-prt[j].w*DT;

     A = prt[j].u*prt[j].u+prt[j].v*prt[j].v+prt[j].w*prt[j].w;
     B = 2.*(xold*prt[j].u+yold*prt[j].v+zold*prt[j].w);
     C = xold*xold+yold*yold+zold*zold - RSP*RSP;
     D = B*B - 4.*A*C;
     if (D > 0.){
         scatter=2;
         dt1 = 0.5*(-B + sqrt(D))/A;
         dt2 = 0.5*(-B - sqrt(D))/A;
         if (dt1 > 0.){
             dtmin = dt1;
             if (dt2 < dt1){
```

```
                dtmin = dt2;
                scatter=4;
            }
            //this (position before wallhit was already...) is happening when
dtmin is not less thanDT
            if (dtmin < DT){
                scatter=3;

                R0 = ((double) random())/RANMAX;
                if (ACCOM > R0){  /* emit later fully accommodated */
                    scatter=5;
                    spherehits++;
                    Epre = 0.5*prt[j].mass*(prt[j].u*prt[j].u
                                            + prt[j].v*prt[j].v +
prt[j].w*prt[j].w)
                    + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp);
                    if (prt[j].kind > 1) {
                        nncon++;
                        Epre -= PARTRAT*(EvapT - 3.*KB*Tsp);
                    }
                    Einlower += Epre;
                    egain += Epre;
                    /* replace adsorbed particle with last particle */
                    loseprt(j);
                }
                else { /* adiabatic diffuse scattering */
                    scatter=1;
                    xi = xold + prt[j].u*dtmin;
                    yi = yold + prt[j].v*dtmin;
                    zi = zold + prt[j].w*dtmin;
                    if (xi < 0.0) xi = -xi;
                    if (yi < 0.0) yi = -yi;
                    if (zi < RSP*cos(ca)) zi = (RSP*cos(ca)-
zi)+(RSP*cos(ca));//left alone since it's going to diffusely scatter from the
sphere anyway
                    cmag = sqrt(prt[j].u*prt[j].u
                                + prt[j].v*prt[j].v + prt[j].w*prt[j].w);
                    R1 = ((double) random())/RANMAX;
                    R2 = ((double) random())/RANMAX;
                    costheta = 1. - R1;
                    sintheta = sqrt(1. - costheta*costheta);
                    phi = 2.*PI*R2;
                    cn = cmag*costheta;
                    cp1 = cmag*sintheta*cos(phi);
                    cp2 = cmag*sintheta*sin(phi);
                    snx = xi/RSP;
                    sny = yi/RSP;
                    snz = zi/RSP;
                        /* compute constants for unit
                     vectors parallel to tangent surface */
                    a1 = 0.0;
                    a2 = 1.0;
                    snynz = sny*sny + snz*snz;
                    if (snynz > 0.0) {
                        a1 = -snz/sqrt(snynz);
                        a2 = sny/sqrt(snynz);
```

```
                    }
                    b0 = (a2*sny - a1*snz);
                    b1 = -a2*snx;
                    b2 = a1*snx;
                              /* convert velocity components
                     to xyz coordinate system */
                    prt[j].u = cn*snx + b0*cp2;
                    prt[j].v = cn*sny + a1*cp1 + b1*cp2;
                    prt[j].w = cn*snz + a2*cp1 + b2*cp2;

                    prt[j].x = xi + prt[j].u*(DT-dtmin);
                    prt[j].y = yi + prt[j].v*(DT-dtmin);
                    prt[j].z = zi + prt[j].w*(DT-dtmin);
                    dts=DT-dtmin;
                }
            }
        }
     }
    if (losscheck > 0){
        rgsq = xmax*xmax;
        rsq = prt[j].x*prt[j].x + prt[j].y*prt[j].y
      + prt[j].z*prt[j].z;
        if (rsq > rgsq)
            loseprt(j);
    }
    if (losscheck > 0){
        if (prt[j].x < 0.0 || prt[j].y < 0.0 || prt[j].z < RSP*cos(ca)){
            wallhitcheck=1;
            wallhit(j,dts);
        }
        xc = (int) (prt[j].x*NCX/xmax);
        yc = (int) (prt[j].y*NCY/ymax);
        zc = (int) ( (prt[j].z-RSP*cos(ca)) *NCZ/zmax);
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


    }
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



/* BBBBBBBBBBB  Function to deal with lost particles */
void loseprt(j)
{
    /* replace lost particle with last particle */
    int nmax;
    nmax = nprt - 1;
    /*      memcpy(&prt[j],&prt[nmax],sizeof(struct part));  */
    prt[j].kind = prt[nmax].kind;
    prt[j].cellno = prt[nmax].cellno;
    prt[j].mass = prt[nmax].mass;
    prt[j].x = prt[nmax].x;
    prt[j].y = prt[nmax].y;
    prt[j].z = prt[nmax].z;
    prt[j].u = prt[nmax].u;
```

```
    prt[j].v = prt[nmax].v;
    prt[j].w = prt[nmax].w;
    prt[j].erot = prt[nmax].erot;
    nprt--;
    lost++;
    losscheck = 0;
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to handle influx from ambient */
void ambemit()
{
    int i, j, n, nadd, xc, yc, zc;
    double mean_c, influx, den, dnadd;
    double r, phi, cT, cp1, cp2, cn;
     double a1, a2, snynz, b0, b1, b2;
     double costheta, sintheta, snx, sny, snz, theta;
     double Rtest, eps, Ie, xdum, coef, R1;
    /* emission of species 1 */
    /* determine number */
    /* flux per time step */
    den = CONCA*PRESS/(KB*TAMB);
    mean_c = sqrt( 8.*KB*TAMB/(PI*MASSA) );
    influx = 0.25*den*mean_c*DT*0.125*4.0*PI*xmax*xmax*(1-cos(ca4))/PARTRAT;
    influx += afluxres1;
    nadd = (int) influx;//(int) rounds down
    dnadd = (double) nadd;
    afluxres1 = influx - dnadd;  /* save residual for next time */
    nadd++;
    cT = sqrt(KB*TAMB/MASSA);//average speed from which to sample

     i = 1;
    while (i < nadd) {
        j = nprt;
        /* randomly select entry
         point for new particle */

        //random theta generation accounting for the contact angle
        theta = ca4 + 10.;//just to initialize while loop
        if (ca4 <= PI/2){
            while (theta > (ca4) ){
                costheta = 1. - ((double) random())/RANMAX;
                theta=acos(costheta);
            }
        }
        else /* (ca4 > PI/2)*/{
            while (theta > (ca4)){
                costheta = 2*((double) random())/RANMAX-1;//...this should
sample between 0 and 180 degrees
                theta=acos(costheta);
            }
        }
        //random theta generation accounting for the contact angle
         sintheta = sqrt(1. - costheta*costheta);
         phi = 0.5*PI*((double) random())/RANMAX;
```

```
 snx = sintheta*cos(phi);
 sny = sintheta*sin(phi);
 snz = costheta;
prt[j].x = (xmax - 0.5*L_CELL)*snx;
prt[j].y = (xmax - 0.5*L_CELL)*sny;
prt[j].z = (xmax - 0.5*L_CELL)*snz;
if ( prt[j].z<RSP*cos(ca) ){
    coef=0.5;

    while(prt[j].z<RSP*cos(ca)){
        if (coef<(-0.2)){
            printf("prt[j].z did not converge in ambemit1");
            exit(EXIT_FAILURE);
        }
        prt[j].z = (xmax - coef*L_CELL)*snz;
        coef=coef-0.1;


    }
}



/* sample velocity components normal and parallel to surface
 from appropriate distributions */

phi = 2.*PI*((double) random())/RANMAX;
R1=((double) random())/RANMAX;
if (R1<=0)  {
    while (R1<=0){
        R1=((double) random())/RANMAX;
    }
}
r = sqrt(-2.*log(R1));
cp1 = cT*r*cos(phi);
cp2 = cT*r*sin(phi);
R1=((double) random())/RANMAX;
if (R1<=0)  {
    while (R1<=0){
        R1=((double) random())/RANMAX;
    }
}
r = sqrt(-2.*log(R1));
cn = -cT*r;


/* compute constants for unit vectors parallel to surface */
a1 = 0.0;
a2 = 1.0;
snynz = sny*sny + snz*snz;
if (snynz > 0.0) {
    a1 = -snz/sqrt(snynz);
    a2 = sny/sqrt(snynz);
}
b0 = (a2*sny - a1*snz);
```

```
        b1 = -a2*snx;
        b2 = a1*snx;


        /* convert sampled velocity components to xyz coordinate system */
        prt[j].u = cn*snx + b0*cp2;
        prt[j].v = cn*sny + a1*cp1 + b1*cp2;
        prt[j].w = cn*snz + a2*cp1 + b2*cp2;

        prt[j].kind = 1;
        xc = (int) (prt[j].x*NCX/xmax);
        yc = (int) (prt[j].y*NCY/ymax);
        zc = (int) ( (prt[j].z-RSP*cos(ca)) *NCZ/zmax);
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


        prt[j].mass = PARTRAT*MASSA;
        /* rotational energy for water */
      Rtest = 0.97*((double) random())/RANMAX;
      eps = 0.002;
      Ie = 0.0;
      while (Ie < Rtest) {
            eps += 0.1;
            Ie += (sqrt(eps - 0.1)*exp(-eps + 0.1)
                   + sqrt(eps)*exp(-eps))*0.05642;
      }
        prt[j].erot = PARTRAT*KB*TAMB*eps;
            i++;
        nprt++;
    }
    /* emission of species 2 */
    /* determine number */
    /* flux per time step */
    den = (1. - CONCA)*PRESS/(KB*TAMB);
    mean_c = sqrt( 8.*KB*TAMB/(PI*MASSB));
    influx = 0.25*den*mean_c*DT*0.125*4.0*PI*xmax*xmax*(1-cos(ca4))/PARTRAT;
    influx += afluxres2;
    nadd = (int) influx;
    dnadd = (double) nadd;
    afluxres2 = influx - dnadd;  /* save residual for next time */
    nadd++;
    cT = sqrt(KB*TAMB/MASSB);//average speed from which to sample

     i = 1;
    while (i < nadd) {
        j = nprt;
        /* randomly select entry
         point for new particle */

        //random theta generation accounting for the contact angle
        theta = ca4 + 10.;//just to initialize while loop
        if (ca4 <= PI/2.){
            while (theta > (ca4) ){
                costheta = 1. - ((double) random())/RANMAX;
                theta=acos(costheta);
            }
```

```
        }
        else /* (ca > PI/2)*/{
            while (theta > (ca4)){
                costheta = 2*((double) random())/RANMAX-1;//...this should
sample between 0 and 180 degrees
                theta=acos(costheta);
            }
        }
        //random theta generation accounting for the contact angle

         sintheta = sqrt(1. - costheta*costheta);
         phi = 0.5*PI*((double) random())/RANMAX;
         snx = sintheta*cos(phi);
         sny = sintheta*sin(phi);
         snz = costheta;
        prt[j].x = (xmax - 0.5*L_CELL)*snx;
        prt[j].y = (xmax - 0.5*L_CELL)*sny;
        prt[j].z = (xmax - 0.1*L_CELL)*snz;
        if ( prt[j].z<RSP*cos(ca) ){
            coef=0.5;

            while(prt[j].z<RSP*cos(ca)){
                if (coef<(-0.2)){
                    printf("prt[j].z did not converge in ambemit1");
                    exit(EXIT_FAILURE);
                }
                prt[j].z = (xmax - coef*L_CELL)*snz;
                coef=coef-0.1;


            }
        }


        phi = 2.*PI*((double) random())/RANMAX;
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        cp1 = cT*r*cos(phi);
        cp2 = cT*r*sin(phi);
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        cn = -cT*r;


        /* compute constants for unit vectors parallel to surface */
        a1 = 0.0;
```

```
        a2 = 1.0;
        snynz = sny*sny + snz*snz;
        if (snynz > 0.0) {
            a1 = -snz/sqrt(snynz);
            a2 = sny/sqrt(snynz);
        }
        b0 = (a2*sny - a1*snz);
        b1 = -a2*snx;
        b2 = a1*snx;


        /* convert sampled velocity components to xyz coordinate system */
        prt[j].u = cn*snx + b0*cp2;
        prt[j].v = cn*sny + a1*cp1 + b1*cp2;
        prt[j].w = cn*snz + a2*cp1 + b2*cp2;

        prt[j].kind = 2;
        xc = (int) (prt[j].x*NCX/xmax);
        yc = (int) (prt[j].y*NCY/ymax);
        zc = (int) ( (prt[j].z-RSP*cos(ca)) *NCZ/zmax);
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;

        prt[j].mass = PARTRAT*MASSB;
        /* rotational energy for argon is zero */
        prt[j].erot = 0.0;
            i++;
        nprt++;
    }
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBB  Function to deal with particles hitting the wall surface */
void wallhit(int j, double dts)
{
    double R1, R2, cmag, phi, costheta, sintheta;
    double xold, yold, zold, DThit, DTleft;
    double upre, vpre, xpre, ypre, zpre, wpre, xdum, xhit, yhit;
    int tagx, tagy, tagz;
    double tx, ty, tz, tmin;
    double R0, cT,r;




    //This wallhit checks if the particle hits a wall surface before the
particle is moved dts seconds, rather than checking if it does it in the
trajectory!!!//

    //dts denotes residual time (residual from when it hits a boundary?)...

    //dts is how much you take the particle back in time!!!!!!!!!!!!!
    //basically, if the simulation goes inside this function, it means that
at some point the particle was attempted to move dts seconds but it ended up
outside the boundary for some reason, so the below checks at what time it hit
the boundary by first moving it back to the point it was before it hit
```

```
anything

    upre=prt[j].u;
    vpre=prt[j].v;
    wpre=prt[j].w;

    xpre=prt[j].x-upre*dts;
    ypre=prt[j].y-vpre*dts;
    zpre=prt[j].z-wpre*dts;



    tagx=0;
    tagy=0;
    tagz=0;


    //pre denotes prior to movement->movement that caused them to strike a
wall (pre-movment for dt seconds)
    //the below gives the time (tx, ty, tz) it would take to go from the
origin up to the point xpre(position before moving to a position that results
outside a specular-wall-boundary).  The negative sign is to make the time
positive for those that happen to be outside of a specular-wall-boundary
(essentially a negative position).  Note that all the "ifs" below basically
ignore times that come from positions inside the boundary (because they would
result in a negative time when multiplied by the negative sign) by only
considering those times that are greater than or equal (>=0)...times
corresponding to positions outside the boundary




    //The below is correct.  THINK ABOUT IT!
    tx=-xpre/upre;
    ty=-ypre/vpre;
    tz=-(zpre-RSP*cos(ca))/wpre;

    tmin=dts;



    //THE BELOW SHOULD NEVER GO INTO ALL 3




    if (tz <= tmin && tz >= 0)
    {
        tagx=0;
        tagy=0;
        // particle will hit z=0 before hitting x=0 or y=0, so don't revert
the x or y components YET
```

```
        tagz=1;
        tmin=tz;
        DTleft=dts-tz; //time left after hitting diffuse wall...here,
obviously dts>tz, which means that it takes less time than dts to hit the
diffuse wall.

    }

    if (tx <= tmin && tx >= 0)
        //here, tmin could be dts or tz, depending on whether it went in the
tz-if-statement
    {

        tagx=1;//will only be set to 1 if less than both, dts and tz (which
is the only way it can come into this if-statement)
        dts=dts-tx;//this is valid whether it went into the tz-if or not,
although don't really need dts if it never went into the tz-if (basically,
don't need it if tagz=0)
    }

    if (ty <= tmin && ty >= 0)
        //here, tmin could be dts, tz, BUT NOT tx!!! (dtmin isn't updtated in
"tx" on purpose, because it could reflect from both "y" and "x"
simultaneously, in the same time step.
    {

        tagy=1;
        if (tagx > 0 && ty<tx)//basically if tagx>0, tmin corresponds to tx,
NOT to tz.  By this point, already verified that both ty and tx are greater
than 0.   Will go into this to correct/reverse anything that was changed to
dts in "tx".
        {




            //why is the below a dts+tx as opposed to dts-tx? ->
            //It adds tx to make it back into the original dts to be able to
subtract ty from it (the original dts) because in this case it turns out that
ty<txu.  Note that tagx is not reverted back to 0, because it is still less
than dts, which means that both boundaries (x=0/y=0)can be hit, but this only
happens if dtmin is NEVER set to tz (if the z=0 boundary is hit), in which
case the velocities are all changed and the x=0/y=0 boundaries need to be
checked later




            dts=dts+tx; //only if ty<tx<tz<original-dts
            dts=dts-ty;// should only make dts the residual after the last
possible hit.  only want dts=dts-(longest dt less than dts...so dtx OR dty,
but not both)!!
```

```
        }

    }

    //If tz is hit first, ignore what would have happened with x and y.  This
is taken care of by changing tmin to tz at the end of the "z-check".
    //HOWEVER, if x or y are hit first, then definitely take them into
account, but still take z into account if hit.  The time would be the same
whether x/u or y/v are reverted or not




    //this works perfectly.  remember that each component is independent.  If
x or y are hit first, the particle is placed in the corresponding x or y, but
the z component shouldn't be affected by reflections in x or y.  therefore,
if x and/or y is hit first and then z is hit, as long as x and/or y is moved
first, the point where the diffuse reflection is made should be 100%
ACCURATE!!!  Therefore, the below is in PEFECT order!



     if (tagx > 0.){
        // if (prt[j].x < 0.)

        prt[j].x = -prt[j].x;
        prt[j].u = -prt[j].u;
     }
     if (tagy > 0.){
        //if (prt[j].y < 0.)

        prt[j].y = -prt[j].y;
        prt[j].v = -prt[j].v;
     }

    check=1;
     if (tagz > 0.){
        check=2;



        /* this puts particles at the diffuse wall in the spot where they
actually hit.  Note that DTleft is only defined in the "z" portion */
        //Also, note how DTleft here already acounts for changes in contact
angle since its calculation involved tz, which accounts for contact angle.
        prt[j].x=prt[j].x-prt[j].u*(DTleft+.001*DTleft);//the .001 is just to
push it a little inside the boundary
        prt[j].y=prt[j].y-prt[j].v*(DTleft+.001*DTleft);//the .001 is just to
push it a little inside the boundary
        prt[j].z=RSP*cos(ca)+.001*L_CELLz;



        xhit=prt[j].x;
        yhit=prt[j].y;
```

```
        if (xhit<0){
            prt[j].y = -prt[j].y;
        }

        if (yhit<0){
            prt[j].x = -prt[j].x;
        }


        //----CHECKING FOR ACCOMODATION----//
        R0 = ((double) random())/RANMAX;
        if (WALL_ACCOM > R0){  /* fully accommodated */
            check=3;

            R1=((double) random())/RANMAX;
            R2=((double) random())/RANMAX;
            if (R2<=0)  {
                while (R2<=0){
                    R2=((double) random())/RANMAX;
                }
            }

            cT = sqrt(PARTRAT*KB*Tw/prt[j].mass);
            phi = 2.*PI*R1;
            r = sqrt(-2.*log(R2));
            uvelocitypre=prt[j].u;
            vvelocitypre=prt[j].v;
            prt[j].u = cT*r*cos(phi);
            prt[j].v = cT*r*sin(phi);
            thisphi=phi;
            thisr=r;
            thiscT=cT;
            uvelpost=prt[j].u;
            vvelpost=prt[j].v;
            // phi = 2.*PI*((double) random())/RANMAX;
            R1=((double) random())/RANMAX;
            if (R1<=0)  {
                while (R1<=0){
                    R1=((double) random())/RANMAX;
                }
            }
            r = sqrt(-2.*log(R1));            //prt[j].w = cT*r*cos(phi);
            //note that the above, using phi, will create a random velocity
in both -z and +z directions.  we only want +z when coming off the wall
(hemispherically [not spherically] diffuse), therefore use the below
way(without phi)
            prt[j].w = cT*r;

        }
        //-----END VELOCITY GENERATION IF ACCOMODATED---//
        else/*diffuse reflection*/{
            R1=((double) random())/RANMAX;
            R2=((double) random())/RANMAX;
            cmag=sqrt(
prt[j].u*prt[j].u+prt[j].v*prt[j].v+prt[j].w*prt[j].w);
```

```
            phi=2*PI*R1;
            costheta=1-R2;
            sintheta=sqrt(1-costheta*costheta);
            prt[j].u= cmag*sintheta*cos(phi);
            prt[j].v= cmag*sintheta*sin(phi);
            prt[j].w= cmag*costheta;

        }//-----END if/else for WALL ACCOMODATION-----//

        //note that these are new velocities!!!!
        //Also, note how DTleft here already acounts for changes in contact
angle since its calculation involved tz, which accounts for contact angle.
        prt[j].x= prt[j].x+prt[j].u*DTleft;
        prt[j].y= prt[j].y+prt[j].v*DTleft;
        prt[j].z= prt[j].z+prt[j].w*DTleft;

        if (prt[j].y<0){
            prt[j].y = -prt[j].y;
            prt[j].v = -prt[j].v;
        }

        if (prt[j].x<0){
            prt[j].x = -prt[j].x;
            prt[j].u = -prt[j].u;
        }
        //YES, these above two if-statements make everything perfect!  there
is no way that "z" can be less than zero since the diffuse reflection takes
care of that!!!




    }


}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */
```

```
void dropemit(k)
{
    int j, n,  xc, yc, zc, iec;
     double Epost, cT, r, phi, cp1, cp2, cn;
     double costheta, sintheta, snx, sny, snz;
     double  a1, a2, snynz, b0, b1, b2, dts;
     double Rtest, eps, Ie;
     double dEdT, ae, be, u283, tcorr, error, ersinc, Eg0;
     double eta, rg, siglv, dedT, leta,R1;

    double Mw, tml, sig, kl, vl, ulv, PsatTi, Pvi, jdout, SF, R,Tsp2;
    double Err, Err_norm, fErr, theta;
    double delTemp, fun1, fun2, dfunddelTemp;
    int counter;


    Mw = 18.0;  //molecular mass of water kg/kmol

    ///* determine energy
    //transfer for this time step */
    Eg = Eg + egain; ///* calc. based on incident energy */
    //Eg is the sum up to this point. egain simply for this time step, it is
zeroed out at each time-step loop.
    nconsum = nconsum + nncon;//note that all the noncondensible particles up
to this point are added here.  nncon is zero-ed out at each k step
    //Note that Tsp is only solved after so many k timesteps (INTST)
    //INTST is number of time steps for Tsp iteration

    R= KB/MASSA;//specific gas constant
    SF =0.25*DropSF(ca, RSP);//ca is input in radians!!!  DropSF is for a
full hemisphere.  1/4 because of a quarter hemisphere!!


    //None of the above change with temperature



    if (k >= INTST)     {
        // surface tension correction for small radii
        sig = surface_tension(Tsp); //N/m
        tml=0.157e-9; //tolman length [m]
```

```
        sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
        //end surface tension correction

        //other properties
        kl = thermal_cond_liquid(Tsp); //W/mK
        vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
        ulv=enthalpy(Tsp)*MASSA; // J/molecule-K
        PsatTi = pressure(Tsp); //
        Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//

        jdout= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j
droplet out, does this work for a droplet of mixed substance?  We use Mw, but
this doesn't account that there are mixed non-condensible water particles in
there.  YES->Remember that the noncondensible particles don't exactly
penetrate the droplet, so using Mw here is perfectly fine.  They are simply
reflected later at an accomodated temperature (if accommodated)



        Eg0= jdout*( PI*RSP*RSP*(1.-cos(ca))/2. )*delt*(ulv + 0.5*KB*Tsp) +
(DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT + SF*kl*(Tsp - Tw)*delt; //total
energy out based on kinetic theory...note how each parameter is based on TSP
here!!!!...total up to this time, not just for this time step!!!
        printf("\n Eg0_1=%8.3e",Eg0);









        error = sqrt((Eg - Eg0)*(Eg - Eg0))/(Eg + EPS);

        Err = (Eg - Eg0);  //compare to using error from above!!!
        Err_norm=(TAMB-Tw)*(kl*SF)*delt;
        fErr=Err/Err_norm;

        //basically, the temperature is iterated to the point where the
        //theoretical ouflux is equal to the 'counted (from the simulation)'
        //influx. (remember that the outflux is nonlinear, that's why it has
to
        //be iterated)









        delTemp=.1;//just to initialize the loop
        counter=0;//just to initialize the loop
```

```
        while (fabs(fErr)>0.0002) //or simply use the error, or fErr...KNOW
what order of magnitude should be in the while loop!!
            //FIRST PART TESTS, SECOND PART ADDS CORRECTION
        {
            counter=counter+1;


            Tsp= TAMB+delTemp;







            // surface tension correction for small radii
            sig = surface_tension(Tsp); //N/m
            tml=0.157e-9; //tolman length [m]
            sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
            //end surface tension correction

            //other properties
            kl = thermal_cond_liquid(Tsp); //W/mK
            vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
            ulv=enthalpy(Tsp)*MASSA; // J/molecule-K
            PsatTi = pressure(Tsp); //
            Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//

            jdout= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j
droplet out

            Eg0= jdout*( PI*RSP*RSP*(1.-cos(ca))/2. )*delt*(ulv + 0.5*KB*Tsp)
+ (DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT + SF*kl*(Tsp - Tw)*delt; //total
energy out based on kinetic theory..note how each parameter is based on TSP
here!!!!...total up to this time, not just for this time step!!!







            fun1= Eg-Eg0;

            //********************IMPORTANT****//

            //Note that it is fun1 that is used for the error check of the
while loop.  This is because I only want to test the "adjusted" temperature.
fun2 is calculated to make the temperature adjustment, but it shouldn't be
used as a valid test for the error of the while loop since it is only used to
calculate the derivative in the Newton-Raphson correction (d(fun)/dT)


            //******************END IMPORTANT***//
```

```
            error = sqrt(fun1*fun1)/(Eg + EPS);

            Err = (fun1);  //compare to using error from above!!!
            Err_norm=(TAMB-Tw)*(kl*SF)*delt;
            fErr=Err/Err_norm; //for comparison only



            //need a temperature increment for the newton raphson correction

            Tsp= TAMB + 1.005*delTemp;



            // surface tension correction for small radii
            sig = surface_tension(Tsp); //N/m
            tml=0.157e-9; //tolman length [m]
            sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
            //end surface tension correction

            //other properties
            kl = thermal_cond_liquid(Tsp); //W/mK
            vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
            ulv=enthalpy(Tsp)*MASSA; // J/molecule-K
            PsatTi = pressure(Tsp); //
            Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//

            jdout= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j
droplet out

            Eg0= jdout*( PI*RSP*RSP*(1.-cos(ca))/2. )*delt*(ulv + 0.5*KB*Tsp)
+ (DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT + SF*kl*(Tsp - Tw)*delt; //total
energy out based on kinetic theory note how each parameter is based on TSP
here!!!!...total up to this time, not just for this time step!!!



            fun2=Eg - Eg0;
            //dfunddelTemp=(fun2-fun1)/delTemp;  These temperature
corrections have been removed to the top to accept the most recently tested
temperature!!!!!
            Tsp= TAMB + delTemp;//Now that I already calculated all my
properties at the modified Tsp, I want to make sure that the loop closes with
the Tsp used in fun1, because that is the one that is tested in the while
loop, and that is the correct one I want to keep when the while statment
holds true.  Therefore, that's why I retype the SAME equation as I did above
(at the very top of the while loop) to end the loop with the Tsp from the
fun1!!.  This should work, because delTemp shouldn't be any different and
should have not been modified at this point since the beginnning of the while
loop.
```

```
            Tsp2=Tsp;



            dfunddelTemp=(fun2-fun1)/(.005*delTemp);//don't want this to be
zero because it is in the denominator below
            delTemp=delTemp- fun1/dfunddelTemp;



            //This second section (what follows from [Tsp= TAMB +
1.005*delTemp;]) of calculated properties at Tsp+1.005*delTemp will only be
necessary if the delTemp needs to be modified..that is, if the fErr
calculated at the end of the first section didn't meet the criteria of the
while loop.  So, can think of this as sort of "useless" ONLY in the last
iteration.



        }//ends the while looop

        // surface tension correction for small radii
        sig = surface_tension(Tsp); //N/m
        tml=0.157e-9; //tolman length [m]
        sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
        //end surface tension correction
        //this is redone here so that sig can be calculated at the right
temperature, not a modifed one for a newton raphson calculation

        errsum=errsum+fErr;//should have already been zeroed out in the main
file
        EvapT=ulv-2.*sig*MASSA/(MDA*RSP);

    }//ends the if statement



    kl = thermal_cond_liquid(Tsp); //W/mK
    egain += (degain - wfluxres);// the "excess" that was lost in the last
"k"-step is now considered (subtracted) in this new iteration of "k" (new k-
step)
    //note that nncon is only for this time step, it is zeroed out at the
beginning of each time step loop
     while (nncon > 0) {
        j = nprt;
        prt[j].kind = 2;
        prt[j].mass = PARTRAT*MASSB;

        /* randomly select surface entry
         position vector for new particle */


         //random theta generation accounting for the contact angle
        theta = ca + 10.;//just to initialize while loop
        if (ca <= PI/2.){
```

```
            while (theta > (ca) ){
                costheta = 1. - ((double) random())/RANMAX;
                theta=acos(costheta);
            }
        }
        else /* (ca > PI/2)*/{
            while (theta > (ca)){
                costheta = 2.*((double) random())/RANMAX-1;//...this should
sample between 0 and 180 degrees
                theta=acos(costheta);
            }
        }
        //random theta generation accounting for the contact angle


        sintheta = sqrt(1. - costheta*costheta);
         phi = 0.5*PI*((double) random())/RANMAX;
         snx = sintheta*cos(phi);
         sny = sintheta*sin(phi);
         snz = costheta;

        /* sample velocity components normal and parallel to surface
         from appropriate distributions */

        cT = sqrt(PARTRAT*KB*Tsp/prt[j].mass);
        phi = 2.*PI*((double) random())/RANMAX;
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        cp1 = cT*r*cos(phi);
        cp2 = cT*r*sin(phi);
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        cn = cT*r;
        /* rotational energy for argon is zero */
        prt[j].erot = 0.0;

        /* compute constants for unit vectors parallel to surface */
        a1 = 0.0;
        a2 = 1.0;
        snynz = sny*sny + snz*snz;
        if (snynz > 0.0) {
            a1 = -snz/sqrt(snynz);
            a2 = sny/sqrt(snynz);
        }
        b0 = (a2*sny - a1*snz);
        b1 = -a2*snx;
```

```
        b2 = a1*snx;


        /* convert sampled velocity components to xyz coordinate system */
        prt[j].u = cn*snx + b0*cp2;
        prt[j].v = cn*sny + a1*cp1 + b1*cp2;
        prt[j].w = cn*snz + a2*cp1 + b2*cp2;
        /* determine entry
         point for new particle */

         dts = DT*((double) random())/RANMAX;
        prt[j].x = RSP*snx + prt[j].u*dts;
        prt[j].y = RSP*sny + prt[j].v*dts;
        prt[j].z = RSP*snz + prt[j].w*dts;


        /* specularly reflect if particles go beyond specular walls */
        if ( prt[j].x < 0.0 || prt[j].y < 0.0 || prt[j].z < RSP*cos(ca) ){
            wallhitcheck=2;
            wallhit(j,dts);
        }
        xc = (int) (prt[j].x*NCX/xmax);
        yc = (int) (prt[j].y*NCY/ymax);
        zc = (int) ( (prt[j].z-RSP*cos(ca)) *NCZ/zmax);
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


        /* remove energy of particle from sphere tally */
      Epost = 0.5*prt[j].mass*(prt[j].u*prt[j].u
                               + prt[j].v*prt[j].v + prt[j].w*prt[j].w)
        + prt[j].erot;
         Einlower -= Epost;
         sphemits++;
         elost += Epost;
         nprt++;
         nncon --;
     } /* end while */

    elost=elost + SF*kl*(Tsp - Tw)*DT;

    while (elost < egain) {
        j = nprt;
        prt[j].kind = 1;
        prt[j].mass = PARTRAT*MASSA;

        /* randomly select surface entry
         position vector for new particle */


         //random theta generation accounting for the contact angle
        theta = ca + 10.;//just to initialize while loop
        if (ca <= PI/2.){
            while (theta > (ca) ){
                costheta = 1. - ((double) random())/RANMAX;
                theta=acos(costheta);
            }
```

```
        }
        else /* (ca > PI/2)*/{
            while (theta > (ca)){
                costheta = 2.*((double) random())/RANMAX-1.;//...this should
sample between 0 and 180 degrees
                theta=acos(costheta);
            }
        }
        //random theta generation accounting for the contact angle

        sintheta = sqrt(1. - costheta*costheta);
         phi = 0.5*PI*((double) random())/RANMAX;
         snx = sintheta*cos(phi);
         sny = sintheta*sin(phi);
         snz = costheta;

        /* sample velocity components normal and parallel to surface
         from appropriate distributions */

        cT = sqrt(PARTRAT*KB*Tsp/prt[j].mass);
        phi = 2.*PI*((double) random())/RANMAX;
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        cp1 = cT*r*cos(phi);
        cp2 = cT*r*sin(phi);
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        cn = cT*r;
        /* rotational energy for water */
      Rtest = 0.97*((double) random())/RANMAX;
      eps = 0.002;
      Ie = 0.0;
      while (Ie < Rtest) {
            eps += 0.1;
            Ie += (sqrt(eps - 0.1)*exp(-eps + 0.1)
                   + sqrt(eps)*exp(-eps))*0.05642;
      }
        prt[j].erot = PARTRAT*KB*Tsp*eps;


        /* compute constants for unit vectors parallel to surface */
        a1 = 0.0;
        a2 = 1.0;
        snynz = sny*sny + snz*snz;
        if (snynz > 0.0) {
            a1 = -snz/sqrt(snynz);
```

```
            a2 = sny/sqrt(snynz);
        }
        b0 = (a2*sny - a1*snz);
        b1 = -a2*snx;
        b2 = a1*snx;


        /* convert sampled velocity components to xyz coordinate system */
        prt[j].u = cn*snx + b0*cp2;
        prt[j].v = cn*sny + a1*cp1 + b1*cp2;
        prt[j].w = cn*snz + a2*cp1 + b2*cp2;
        /* determine entry
         point for new particle */

         dts = DT*((double) random())/RANMAX;
        prt[j].x = RSP*snx + prt[j].u*dts;
        prt[j].y = RSP*sny + prt[j].v*dts;
        prt[j].z = RSP*snz + prt[j].w*dts;



        /* specularly reflect if particles go beyond specular walls */
        if (prt[j].x < 0.0 || prt[j].y < 0.0 || prt[j].z < RSP*cos(ca) ){
            wallhitcheck=3;
            wallhit(j,dts);
        }

        xc = (int) (prt[j].x*NCX/xmax);
        yc = (int) (prt[j].y*NCY/ymax);
        zc = (int) ( (prt[j].z-RSP*cos(ca)) *NCZ/zmax);
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;



        /* remove energy of particle from sphere tally */
      Epost = 0.5*prt[j].mass*(prt[j].u*prt[j].u
                               + prt[j].v*prt[j].v + prt[j].w*prt[j].w)
        + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp);
      Einlower -= Epost;
      sphemits++;
      elost += Epost;
        nprt++;
    }
    wfluxres = elost - egain;//this is the "excess" that was lost and should
be considered in the next iteration of "k" (basically subtracted from egain
in the next k-step)}

}



/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */
```

```
/* BBBBBBBBBBB  Function to select collision pairs and collect
 occupancy info for each cell */
void collect()
{
    double ptcount;
    int i, j, m1, m2, icount, icell, xdum;
    int iptcount, space[NCELLS];
```

```
    for (i=0; i< NCELLS; i++) {
        space[i] = -1;  /* set arrays to zero */
        sampsize[i] = 0;
    }
    ptcount = (double) nprt;
    iptcount = (int) (2.0*ptcount);
    /* randomly swap particle numbers */
    for (i=0; i< iptcount; i++) {
        m1 = (int) (0.9999*ptcount*((double) random())/RANMAX);
        m2 = (int) (0.9999*ptcount*((double) random())/RANMAX);
        swap(m1,m2);
    }

    icount = 0;
    for (j=0; j< nprt; j++) {
        icell = prt[j].cellno;




        if (space[icell] < 0)
        {
            space[icell] = j;
        }
        else {
            pt1[icount] = j;  /* store candidate pair */
            pt2[icount] = space[icell];

            space[icell] = -1;
            icount++;
            sampsize[icell]++;
        }
    }

    nopairs = icount;
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to swap particle numbers for two particles */
void swap(i,j)
{
    int ndum;
    double dum;
     struct part prtdum;
    /*      memcpy(&prtdum,&prt[i],sizeof(struct part));
     memcpy(&prt[i],&prt[j],sizeof(struct part));
     memcpy(&prt[j],&prtdum,sizeof(struct part));  */
    ndum = prt[i].kind;
    prt[i].kind = prt[j].kind;
    prt[j].kind = ndum;
    ndum = prt[i].cellno;
    prt[i].cellno = prt[j].cellno;
    prt[j].cellno = ndum;
```

```
    dum = prt[i].mass;
    prt[i].mass = prt[j].mass;
    prt[j].mass = dum;
    dum = prt[i].x;
    prt[i].x = prt[j].x;
    prt[j].x = dum;
    dum = prt[i].y;
    prt[i].y = prt[j].y;
    prt[j].y = dum;
    dum = prt[i].z;
    prt[i].z = prt[j].z;
    prt[j].z = dum;
    dum = prt[i].u;
    prt[i].u = prt[j].u;
    prt[j].u = dum;
    dum = prt[i].v;
    prt[i].v = prt[j].v;
    prt[j].v = dum;
    dum = prt[i].w;
    prt[i].w = prt[j].w;
    prt[j].w = dum;
    dum = prt[i].erot;
    prt[i].erot = prt[j].erot;
    prt[j].erot = dum;
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to execute collision of candidate pairs */
void collide()
{
     double vrel, np, Rtest, ucm, vcm, wcm, mr, Ecoll, r, rlim;
     double fracrt, vrelp, Erelp, Erotp, nx, ny, nz, eps, sinki;
     double prob, fzr, etransr, Epre, Epost;
     double d0, d1, d2, d3, d4, fracr1;
     double lamAA, lamBB, lamAB, D12, den;
     int i, j, m1, m2, ifzr, isw;
     den = PRESS/(KB*TAMB);
     D12 = (DMA + DMB)/2.;
     lamAA = 1.0/(PI*1.414*den*DMA*DMA);
     lamBB = 1.0/(PI*1.414*den*DMB*DMB);
     lamAB = 1.0/(PI*1.414*den*D12*D12);
     for (i=0; i< nopairs; i++) {  /* do for all candidate pairs */
        m1 = pt1[i];
        m2 = pt2[i];
        np = (int) mnp[prt[m1].cellno];
        vrel = sqrt( (prt[m1].u - prt[m2].u)*(prt[m1].u - prt[m2].u)
                    + (prt[m1].v - prt[m2].v)*(prt[m1].v - prt[m2].v)
                    + (prt[m1].w - prt[m2].w)*(prt[m1].w - prt[m2].w) );
        prob = 1.5*np*vrel*DT/(1.414*freeparden*freelam);

        Rtest =((double) random())/RANMAX;
        if (prob > Rtest){          /* test whether they collide
                                  -if yes, continue */

            ifzr = (int)(IZROT + ((double) random())/RANMAX);
```

```
            fzr = (double) ifzr;
            r = sqrt( prt[m1].x*prt[m1].x + prt[m1].y*prt[m1].y
                     + prt[m1].z*prt[m1].z );
            rlim = RSP + 2.0*L_CELL;
            if (r > RSP && r < rlim) collcount++;
            Epre = 0.5*prt[m1].mass*(prt[m1].u*prt[m1].u
                                     + prt[m1].v*prt[m1].v +
prt[m1].w*prt[m1].w)
            + prt[m1].erot
            +  0.5*prt[m2].mass*(prt[m2].u*prt[m2].u
                                 + prt[m2].v*prt[m2].v + prt[m2].w*prt[m2].w)
            + prt[m2].erot;

            ucm = (prt[m1].u*prt[m1].mass +
                   prt[m2].u*prt[m2].mass);
            ucm /= (prt[m1].mass + prt[m2].mass);
            vcm = (prt[m1].v*prt[m1].mass +
                   prt[m2].v*prt[m2].mass);
            vcm /= (prt[m1].mass + prt[m2].mass);
            wcm = (prt[m1].w*prt[m1].mass +
                   prt[m2].w*prt[m2].mass);
            wcm /= (prt[m1].mass + prt[m2].mass);
            mr = (prt[m1].mass*prt[m2].mass)/(prt[m1].mass+prt[m2].mass);
            etransr = 0.5*mr*vrel*vrel;

            Ecoll = etransr + prt[m1].erot + prt[m2].erot;

            /* sample distribution for fraction
             in relative translation */
            isw = prt[m1].kind*prt[m2].kind;
            switch (isw) {
                case 1:  /* water - water */
                    d0 = ((double) random())/RANMAX;
                    d1 = d0 - 0.1808;
                    d2 = d0 - 0.5248;
                    d3 = d0 - 0.8208;
                    d4 = d0 - 1.0;
                    fracrt = - 6.133*d0*d2*d3*d4 + 15.752*d0*d1*d3*d4
                    - 21.533*d0*d1*d2*d4 + 14.335*d0*d1*d2*d3;
                    fracrt *= 0.99999;

                    Erelp = Ecoll*fracrt*fzr + (1. - fzr)*etransr;
                    Erotp = Ecoll - Erelp;
                    d0 = ((double) random())/RANMAX;
                    d1 = d0 - 0.1424;
                    d2 = d0 - 0.5000;
                    d3 = d0 - 0.8576;
                    d4 = d0 - 1.0;
                    fracr1 = - 6.403*d0*d2*d3*d4 + 15.640*d0*d1*d3*d4
                    - 25.614*d0*d1*d2*d4 + 16.377*d0*d1*d2*d3;
                    fracr1 *= 0.9999;
                    fracr1 += 0.00002;
                    prt[m1].erot = fzr*Erotp*fracr1
                    + (1.-fzr)*prt[m1].erot;
                    prt[m2].erot = Erotp - prt[m1].erot;
                    break;
```

```
                case 2:  /* water - argon */
                    d0 = ((double) random())/RANMAX;
                    d1 = d0 - 0.1508;
                    d2 = d0 - 0.3813;
                    d3 = d0 - 0.7344;
                    d4 = d0 - 1.0;
                    fracrt = - 17.415*d0*d2*d3*d4 + 26.041*d0*d1*d3*d4
                    - 18.659*d0*d1*d2*d4 + 7.1661*d0*d1*d2*d3;
                    fracrt *= 0.99999;

                    Erelp = Ecoll*fracrt*fzr + (1. - fzr)*etransr;
                    Erotp = Ecoll - Erelp;
                    prt[m1].erot = ((double) (2.-prt[m1].kind))*Erotp;
                    prt[m2].erot = ((double) (2.-prt[m2].kind))*Erotp;
                    break;
                case 4:  /* argon - argon */
                    /* no rotation - all translation */
                    Erelp = Ecoll;
                    break;
            } /* end of switch */
            vrelp = sqrt(Erelp/(mr/2.));
            nx = 2.*((double) random())/RANMAX - 1.0;
            eps = 2.*PI*((double) random())/RANMAX;
            sinki = sqrt(1. - nx*nx);
            ny = sinki*cos(eps);
            nz = sinki*sin(eps);
            prt[m1].u = ucm + mr*vrelp*nx/prt[m1].mass;
            prt[m1].v = vcm + mr*vrelp*ny/prt[m1].mass;
            prt[m1].w = wcm + mr*vrelp*nz/prt[m1].mass;
            prt[m2].u = ucm - mr*vrelp*nx/prt[m2].mass;
            prt[m2].v = vcm - mr*vrelp*ny/prt[m2].mass;
            prt[m2].w = wcm - mr*vrelp*nz/prt[m2].mass;
            Epost = 0.5*prt[m1].mass*(prt[m1].u*prt[m1].u
                                      + prt[m1].v*prt[m1].v +
prt[m1].w*prt[m1].w)
            + prt[m1].erot
            +  0.5*prt[m2].mass*(prt[m2].u*prt[m2].u
                                 + prt[m2].v*prt[m2].v + prt[m2].w*prt[m2].w)
            + prt[m2].erot;
            Ecgain += Epost - Epre;
            nocoll++;
            nocolrot += ifzr;
        }
    }
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to collect statistics on particles
 and determine statistical properties */
void statpr()     /* not used in this version */
{
    double Hfunc, crms, erotm, cssum, erotsum, inopart;
    int i, j, k, nbin, part1count;
    for (i=0; i<15; i++) {
        fractbin[i] = 0.;
```

```
    }
    inopart = 1./((double) nprt);
    j = 0; cssum = 0.; erotsum = 0.; part1count = 0;
    while (j < nprt) {
      cssum += prt[j].u*prt[j].u + prt[j].v*prt[j].v
        + prt[j].w*prt[j].w;
      erotsum += prt[j].erot;
      if (prt[j].kind == 1) part1count++;
      j++;
    }
    crms = sqrt(cssum*inopart);
    erotm = erotsum*inopart;
    totfract1 = ((double) part1count)*inopart;

    j = 0;
    while (j < nprt) {
      nbin = (int) (7.499 + 5.0*prt[j].u/crms);
      if (nbin > -1 && nbin < 15) fractbin[nbin] += inopart;
      j++;
    }




}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

/* BBBBBBBBBBB  Function to collect statistics on particles to allow
 later determination of statistical properties for each cell */
void statcell(k)
{
    int i, j, icell;
    double count1, count2, stepcount, denom, xdum;

    for (j=0; j< nprt; j++) {
        icell = prt[j].cellno;


        if (prt[j].kind == 1)
        {cell[icell].countkind1++;


        }


        if (icell<0){
            printf("k=%d",k);
            printf("x=%8.3e",prt[j].x);
            printf("y=%8.3e",prt[j].y);
            printf("z=%8.3e",prt[j].z);
        }


        cell[icell].countkind2++;
        cell[icell].etrsum += 0.5*prt[j].mass*(prt[j].u
                                               *prt[j].u + prt[j].v*prt[j].v
```

```
                                                +prt[j].w*prt[j].w);
        cell[icell].erotsum += prt[j].erot;
        cell[icell].usum += prt[j].u;
    }
    /* refine cell statistics */
    stepcount = (double) (k+1);
    for (i=0; i < NCELLS; i++){
        count1 = (double) cell[i].countkind1;
        count2 = (double) cell[i].countkind2;
        denom = (3.0*count1+1.5*count2 + EPS)*KB*PARTRAT;
        cell[i].temp = (cell[i].etrsum + cell[i].erotsum)/denom;
      cell[i].molconc1 = count1/(count1 + count2 + EPS);
        cell[i].umean = cell[i].usum/(count1 + count2 + EPS);
      mnp[i] = ((double) (count1 + count2))/stepcount;
    }



}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */




double density_liquid(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of density of the liquid at given
    //%Temperature x in [kg/m3]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated

    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,75,     80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={999.8,      999.9,      999.7,      999.1,      998.2,
997, 995.6, 994,  992.2,      990.2,      988,  985.7,      983.2,
     980.5, 977.7,      974.8,      971.8,      968.6,      965.3,
     961.9, 958.4,      958.3,      954.7,      950.9,      947.1,
     943.1};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //density liquid
        double p262[]={943.11, 934.83, 926.13, 917.01, 907.45, 897.45,
887.00, 876.08, 864.66, 852.72, 840.22, 827.12, 813.37, 798.89, 783.63,
767.46, 750.28, 731.91, 712.14, 690.67, 667.09, 640.77, 610.67, 574.71,
527.59, 451.43};

        for (i=0;i<len26;i++){
```

```
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }

    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function


/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */
```

```
double enthalpy(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of enthalpy of vaporization at given
    //%Temperature x in [J/kg-K]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated
    //%This is h_lv!!!!  LATENT HEAT


    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={2500.92, 2489.04, 2477.19,    2465.35,    2453.52,
     2441.68,     2429.82,    2417.92,    2405.98,    2394, 2381.95,
     2369.83,     2357.65,    2345.38,    2333.03,    2320.57,    2308.01,
     2295.32,     2282.49,    2279.52,    2256.47,    2256.4,     2243.12,
     2229.64,     2215.99,    2202.12};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);



    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //enthalpy of vaporization
        double p262[]={2202.6, 2174.2, 2144.7, 2114.3, 2082.6, 2049.5,
2015.0, 1978.8, 1940.7, 1900.7, 1858.5, 1813.8, 1766.5, 1716.2, 1662.5,
1605.2, 1543.6, 1477.1, 1404.9,
            2727.9-1402.2,
            1238.6,
            2666.0-1525.9,
            1027.9,
            2563.6-1670.9,
            720.5,
            2334.5-1890.7};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }

    for (i=0;i<len26;i++)
    {
        p26[i]=p26[i]*1000;
    }

    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};
```

```
    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);

    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


double pressure(double x)
{
    //       %------------Saturation Properties of H2O--------------------%
    //       %L is the Lagrangian interpolation of Pressure at given
Temperature
    //       %pressure in [Pa]
    //       %x= ;%temperature in [Kelvin] at wich the property is to be
evaluated




    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5, 10, 15, 20, 25, 30 ,35 ,40 ,45 ,50 ,55 ,60 ,65 ,70
```

```
,75 ,80 ,85 ,90, 95 ,99.97 ,100 ,105 ,110 ,115, 120}  ;//vector of temps
    double p26[]={0.00061, 0.00087 ,0.00123 ,0.00171 ,0.00234 ,0.00317,
0.00425, 0.00563, 0.00738, 0.00959, 0.01235, 0.01576, 0.01995, 0.02504,
0.0312, 0.0386, 0.04741, 0.05787, 0.07018, 0.08461, 0.10133, 0.10142, 0.1209,
0.14338, 0.16918, 0.19867};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);


    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //pressure
        double p262[]={.19867, .27028, .36154, .47616, .61823, .79219,
1.0028, 1.2552, 1.5549, 1.9077, 2.3196, 2.7971, 3.3469, 3.9762, 4.6923,
5.5030, 6.4166, 7.4418, 8.5879, 9.8651, 11.284, 12.858, 14.601, 16.529,
18.666, 21.044};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }

    }

    for (i=0;i<len26;i++)
    {
        p26[i]=p26[i]*10e5;
    }


    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
```

```
        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


double specific_volume_vapor(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of the Specific Volume of the vapor
at
    //%given Temperature x in [m3/kg]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated


    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={205.99, 147.01,   106.3,      77.875,     57.757,
     43.337,32.878,     25.205,     19.515,     15.252,     12.027,
     9.5643,7.6672,     6.1935,     5.0395,     4.1289,     3.4052,
     2.8258,2.3591,     1.9806,     1.6732,     1.6718,     1.4184,
     1.2093,1.0358,     0.89121};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if(x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //specific volume vapor
        double p262[]={0.89121, 0.66800, 0.50845, 0.39245, 0.30678, 0.24259,
0.19384, 0.15636, 0.12721, 0.10429, 0.086092, 0.071503, 0.059705, 0.050083,
0.042173, 0.035621, 0.030153, 0.025555, 0.021660, 0.018335, 0.015471,
0.012979, 0.010781, 0.0088024, 0.0069493, 0.0049544};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }


    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[22], p26[len26-
```

```
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[22], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function


/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double surface_tension(double x)
{
    //                      %------------Saturation Properties of H2O---------
-----------%
    //                      %L is the Lagrangian interpolation of Surface
Tension at given Temperature
    //                      %x in [N/m]
    //                      %x= ;%temperature in [Kelvin] at which the
property is to be evaluated

    int len26,lenT,i,j,lsize,flag;
    double L;
```

```
    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={75.65, 74.94, 74.22, 73.49, 72.74, 71.97, 71.19, 70.4,
69.6, 68.78, 67.94, 67.1, 66.24, 65.37, 64.48, 63.58, 62.67, 61.75, 60.82,
59.87, 58.92, 58.91, 57.94, 56.96, 55.97, 54.97};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);


    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //surface tension
        double p262[]={0.054968*1000, 0.052932*1000, 0.050856*1000,
0.048741*1000, 0.046591*1000, 0.044406*1000, 0.042190*1000, 0.039945*1000,
0.037675*1000, 0.035381*1000, 0.033067*1000, 0.030736*1000, 0.028394*1000,
0.026043*1000, 0.023689*1000, 0.021337*1000, 0.018993*1000, 0.016664*1000,
0.014360*1000, 0.012089*1000, 0.0098644*1000, 0.0077026*1000, 0.0056255*1000,
0.0036654*1000, 0.0018772*1000, 0.00038822*1000};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }



    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
```

```
        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    L=L/1000;//to convert from mN/m to N/m
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double thermal_cond_liquid(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of the thermal conductivity of the
liquid
    //%liquid at given Temperature x in [mW/m-K]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated


    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={561, 570.5, 580,  589.3,      598.4,      607.2,
     615.5, 623.3,      630.6,      637.3,      643.6,      649.2,
     654.3, 659,  663.1,      666.8,      670,  672.8,      675.3,
     677.3, 679.1,      679.1,      680.5,      681.7,      682.6,
     683.2};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if (x>393.15){
        //thermal conductivity liquid...MAKE SURE TO PUT IN mW/m-K
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        double p262[]={0.68319*1000, 0.68370*1000, 0.68330*1000,
0.68204*1000, 0.67996*1000, 0.67705*1000, 0.67332*1000, 0.66875*1000,
0.66331*1000, 0.65697*1000, 0.64965*1000, 0.64131*1000, 0.63185*1000,
0.62119*1000, 0.60924*1000, 0.59591*1000, 0.58115*1000, 0.56496*1000,
0.54743*1000, 0.52875*1000, 0.50920*1000, 0.48907*1000, 0.46851*1000,
0.44737*1000, 0.42572*1000, 0.42504*1000};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }
```

```
    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    L=L/1000;// To convert it from mW/m-K to W/m-K
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double temperature(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of Saturation Temperature at given
Pressure
    //%x in [kelvin]
    //%x= ;%pressure in [Pa] at wich the property is to be evaluated
```

```
    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5, 10, 15, 20, 25, 30 ,35 ,40 ,45 ,50 ,55 ,60 ,65 ,70
,75 ,80 ,85 ,90, 95 ,99.97 ,100 ,105 ,110 ,115, 120}  ;//vector of temps
    double p26[]={0.00061, 0.00087 ,0.00123 ,0.00171 ,0.00234 ,0.00317,
0.00425, 0.00563, 0.00738, 0.00959, 0.01235, 0.01576, 0.01995, 0.02504,
0.0312, 0.0386, 0.04741, 0.05787, 0.07018, 0.08461, 0.10133, 0.10142, 0.1209,
0.14338, 0.16918, 0.19867};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if(x>0.19867e6){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //pressure
        double p262[]={.19867, .27028, .36154, .47616, .61823, .79219,
1.0028, 1.2552, 1.5549, 1.9077, 2.3196, 2.7971, 3.3469, 3.9762, 4.6923,
5.5030, 6.4166, 7.4418, 8.5879, 9.8651, 11.284, 12.858, 14.601, 16.529,
18.666, 21.044};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }

    }


    for (i=0;i<len26;i++)
    {
        p26[i]=p26[i]*10e5;//to convert from Mpa to Pa
    }


    double p[]={p26[0], p26[14],p26[17],p26[19],p26[21], p26[23],p26[24],
p26[len26-1]};
    double T[]={T26[0], T26[14], T26[17],T26[19],T26[21],T26[23],T26[24],
T26[len26-1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
```

```
            {
                l[j]= (x-p[j])/(p[i]-p[j]);
            }
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;

    }
    double ebeprod[lsize];
    ebe(ll,T,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function


/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

double ar_prodsum(double arg[], int arg_length, int flag)

{//%opens function-

    double *end = arg+arg_length;
    double sum = 0, prod = 1;
    double *p;

    for (p = arg; p!=end; ++p) {
        sum += *p;
        prod *= *p;
    }

    if (flag<2)
    {
        return(prod);
    }
    else
    {
        return(sum);
    }

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



void ebe(double a1[], double a2[],double *a3,int alen) //element by element
{
    /*a3 should be declared outside of here...a3 is the output, so make sure
that a3 is an array when using this function.
     //double a3[alen]; */
    int i;
```

```
    for (i=0;i<alen;i++)
    {
        *(a3+i)=a1[i]*a2[i];
    }
}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

double DropSF(double phi, double radius)
{//input the ange in radians
    int n;
    double HS_ratio, segment_ratio,HS_top,HS_base, segment_top,
segment_base,phi_term,HS_factor;
    double segment_factor, ShapeFactor,SF;


    //% Eq. 34 (hemispherical droplet) & Eq. 39 (Spherical segment) are
series functions that converge for n > 8
    HS_ratio = 0;
    segment_ratio = 0;

    for (n=0;n<11;n++)//n = 0:1:10
    {
        HS_top = (4*n+3)*(2*n+1)*4*factorial(n)*factorial(n)*(pow((-
1),(2*n)));   //% eq. 34 numerator
        HS_base = (2*n+2)*(2*n+2)*(pow(2,(4*n)))*(pow((factorial(n)),4)); //%
eq. 34 denominator
        HS_ratio = HS_ratio + HS_top/HS_base;
        segment_top = (4*n+3)*(2*factorial(n))*(pow((-1),(n))); //% eq. 39
numerator
        segment_base = (2*n+2)*factorial(n)*factorial(n)*(pow(2,(2*n)));//%
eq. 39 denominator
        phi_term= pow((tan(phi/2)),(2*n+1));
        segment_ratio = segment_ratio + segment_top*phi_term/segment_base;
    }

    HS_factor = 4*HS_ratio;   //% eq. 34 is for the hemispherical drop
Nusselt number
    segment_factor = HS_factor/segment_ratio;   //% eq. 39 is for the
spherical segment drop Nusseltnumber

    ShapeFactor = 0.5*segment_factor*PI*radius*sin(phi);  //% Shape factor =
Nu_drop*Area/Diameter

    SF = ShapeFactor;

    return (SF);
}

long factorial(int n)
{
    int c;
    long result = 1;

    for( c = 1 ; c <= n ; c++ )
```

```
        result = result*c;

    return ( result );
}
```

Appendix F

C Code for DSMC Model on a Droplet Cluster

```
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 ::
 ::  A protype zero-bulk-flow DSMC program in C.
 ::  Boundary conditions in this version set to
 ::  compute transport to a DROPLET CLUSTER
 ::  condensing on a cold wall with variable contact angle
 ::
 :: This file specifically simulates for a 400nm droplet, 90 degree contact
angle
 ::
 ::
 ::  H. Mendoza 2012/2013, based on V. Carey  1/95, 9/97, 7/99. ::

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*
/


#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>


/* BBBBBBBBBBB  Definition of constants */
//# MOLECULES PER PARTICLE
//#define TAMB    242.4       /* ambient temp K */
//#define TSP     244.0       /* guess of sphere temp K
#define NCX       28          /* number of cells in x direction */
#define NCY       25          /* number of cells in y direction */
#define NCZ1      11          /* number of cells in z direction */
#define NCELLS  7700          /* total number of cells */

#define NPRTMAX 70000         /* maximum number of particles */
#define INPRTNO 35000         /* initial number of particles */

#define NSTEPS    16600       /* number of time steps to be done */
#define INTST     1           /* no. of steps to Tsp iteration */
#define DT  .9638e-10   /* time step in seconds REMEMBER ABOUT EPS2!!!!!!!!*/
#define L_CELL    4.0e-08     /* length of cell in meters */
#define RSP 400.0e-09         /* sphere radius in meters */
#define SOD 0.4           /* separation distance over diameter ratio */
#define PRESS     101.325e+03 /* ambient pressure Pa */
#define PARTRAT   83.949      /* number of molecules per particle */

#define ACCOM     0.9         /* droplet surface accommodation coeficent */
#define WALL_ACCOM 1.0  //accomodation coeficient for condensing wall

#define CONCA     1.0         /* ambient molar concentration of species 1
(water) */



#define RANMAX  2147483647.         /* max value returned by random() */
/*#define NSTEPS  300000             number of time steps to be done */

/* #define L_CELL 5.00e-08 */ /* length of cell in meters */
```

```
/* #define RSP    4.0e-07     */    /* sphere radius in meters */

#define RADFLUX   0.0         /* net radiation flux to drop W/sq m */

#define MASSA     2.99e-26    /* mass of species 1 molecule kg */
#define MASSB     4.65e-26    /* mass of species 2 molecule kg */
#define DEGFREB 5.           //Degrees of freedom of species B
#define MDA 1000.           /* mass density of liq species 1000 kg/cu m */
//#define EVAPPM  7.016e-20   /* initial guess of effective energy of
vaporization per molecule J */
#define DMA 4.51e-10    /* effective diameter of species 1
molecule meters */
#define DMB 3.78e-10    /* effective diameter of species 2
molecule meters */
#define IZROT     0.2         /* value of 1/Zrot */
/* #define CA     0.07583     /* surface tension constant N/m  */
/* #define CA     0.000001  */      /* surface tension constant N/m  */
/* #define CB     0.1477            /* surface tension constant N/mK */
/* #define CB     0.0   */    /* surface tension constant N/mK */
/*#define GAM     1.566e-07   /* Tolman length scale parameter kg/sq m  */
/*#define PARTRAT 195.6       /* number of molecules per particle */
#define KB  1.38e-23    /* Boltzmann constant J/K */
#define NA  6.02e+26    /* Avogadro's number molecules/kmol */
#define PI  3.141592653589793
#define EPS 1.00e-20    /* a small but non-zero number */
#define EPS2 1.00e-20   /* a small but non-zero number */

#define PINT      50          /* time steps in print interval */
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



/* BBBBBBBBBBB  Definition of external varibles and functions */
FILE *fpo;  /* uses indirection to define pointer fpo to a file */
FILE *fpo2;  /* uses indirection to define pointer fpo to a file */
FILE *fpo3;  /* uses indirection to define pointer fpo to a file */
FILE *fpo4;  /* uses indirection to define pointer fpo to a file */
FILE *fpo5;  /* uses indirection to define pointer fpo to a file */



//FILE *fopen(); /* function used to define files */
int nprt, lost, again, nopairs, nocoll, nocolrot, nncon1, nncon2, nncon3;
int pt1[NPRTMAX], pt2[NPRTMAX], sampsize[NCELLS];
int losscheck, spherehits1, spherehits2, spherehits3,
sphemits1,sphemits2,sphemits3, collcount, nconsum;
double fractbin[15], mnp[NCELLS];
double wfluxres, afluxres1, afluxres2, elost, elostNcon,elostcon;
double elost1, elost2, elost3, egain, egain1, egain2, egain3, degain,
elostconduc, elostconducsum, egain_beta, elost1sum, elost2sum, elost3sum,
egain1sum, egain2sum, egain3sum;
double xmax, ymax, zmax;
double freelam, den, freeparden;
double totfract1, Einlower1, Einlower2, Einlower3, Ecgain, sepsum;
double Eg, Tsp, EvapT, delt, errsum;
double ca, L_CELLz,Tw,sc, TAMB, TSP, EVAPPM;
```

```
int wallhitcheck, scatter, check;
double uvelocitypre,vvelocitypre,uvelpost,vvelpost,thisphi,thisr,thiscT;

/* New External Variables Added*/
double L, H, d, s;
int boundnum, spacescape, dropnumber, NCZ;
double dtres, dtmin,xold, yold, zold, condwallZ;
double m1, m2, cz;
double d1p[3],d2p[3],d3p[3];
double cx1, cy1, cz1, cx2, cy2, cz2, cx3, cy3, cz3;
double gapz,gapx,gapy,gapx_perpendicular;
double theta11,ccc,RR, acosRR, j0print, Eg0h20print, wallhit1_DT,
wallhit1_delt ,wallhit2_DT;
double wallhit2_delt, wallhit3_DT, wallhit3_delt, Aside, condwallhit_DT,
condwallhit_delt, spacescape_DT, wfluxres_delt, wfluxres2;/*MAKE SURE TO
DELETE THESE*/
double sphemits1_DT, sphemits2_DT, sphemits3_DT, spherehits1_DT,
spherehits2_DT, spherehits3_DT, influxprint, At, Aw;
/* double d1p[], d2p[], d3p[] can't declare these here because of the way
they are defined below */

/* End New Variables Added*/



struct part {
    int kind;
    int cellno;
    double mass;
    double x;
    double y;
    double z;
    double u;
    double v;
    double w;
    double erot;
} prt[NPRTMAX];
struct cellstat {
    int countkind1;
    int countkind2;
    double temp;
    double molconc1;
    double umean;
    double etrsum;
    double erotsum;
    double usum;
} cell[NCELLS];
void initcon();
void spaceprtstart(int j);//NEW
void advance(int k);
void chkspace(int n);
void loseprt(int n);
void ambemit(int k);
void ambprtstart(int j);//New
void dropemit(int n);
void dropemitpos(int j, int k);//New
```

```
void dropemitvel(int j);//New
void dropemitcheck(int j);//New
void diffusewall_reflection(int j);//NEW
void wallhit(int j);
void collect();
void swap(int n, int m);
void collide();
void statpr();
void statcell(int n);
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

/* FUNCTIONS THAT HECTOR ADDED */
double density_liquid(double n);
double enthalpy(double x);
double pressure(double x);
double specific_volume_vapor(double x);
double surface_tension(double x);
double thermal_cond_liquid(double x);
double temperature(double x);
double ar_prodsum(double arg[], int arg_length, int flag);
void ebeadd(double a1[], double a2[],double *a3,int alen); /*element by
element addition*/
void ebe(double a1[], double a2[],double *a3,int alen);
double pointdist(double a1[], double a2[]);
long factorial(int n);
double DropSF(double phi, double radius);
void mat_mult3x3(double a[][3],double b[][3], double c[][3],int nrows,int
ncol,int mcol);
double mat_mult_vel(double a[][3],double b[][1], int nrow,int mcol);

/*EEEEEEEEEEE FUNCTIONS THAT HECTOR ADDED EEEEEEEEEEEEE*/




/* BBBBBBBBBBB  MAIN PROGRAM OF PARTICLE SIMULATION CODE */
main()
{

    int i, ndum, iprof, j, jmax, k, plim, ip;
    double xdum, rm12, D12, flam1, flam2, drdt, flux;
    double
collcount_time,kTCR,kTCRA,kTCRB,kTCRAB,sigab,rmab,VbarA,VbarAB,VbarB, nA, nB,
octvol, h,As1, Einlower, elost3dropseg_stepsum, Egain3dropseg_stepsum,
elost3dropseg_indiv_step, egain3dropseg_indiv_step;
    double  h_conden_delt_einlower, h_conden_delt_egainlost, h_conden_DT,
h_conduc_delt, h_conduc_DT;
    double h_main1, h_main2;



    printf("start");
    fpo=fopen("outDSMC_400nmrad_ca90","w"); /* open file outDSMC to write
output in fpo is pointer to this file..sdwc =single droplet wall
```

```
condensation*/
    fpo2=fopen("gainloss_DT_CoON_acc1_dropletsON_fixed.dat","w"); /* open
file outDSMC to write */
    fpo3=fopen("Eg0h20_CoON_acc1_dropletsON_fixed.dat","w"); /* open file
outDSMC to write Eg0h20print, Eg0h20print/delt,(Eg-Eg0h20print)/delt
...Con=collisons on */
    fpo4=fopen("Eg_O_delt_CoON_acc1_dropletsON_fixed.dat","w"); /* open file
outDSMC to write */



    /* Note that this section are all new additions */
    TAMB= temperature(PRESS);
    sc=3.;//amount of subcooling in [K]
    Tw=TAMB-sc;//wall temperature
    //TSP=Tw+sc/2.;//just a guess
    TSP=Tw;//just a guess
    EVAPPM=enthalpy(TSP)*MASSA; /* J/molecule-K */



    ca= 90.;//degrees
    ca=ca*PI/180.;//converted to radians

    /*New Global Variables*/
    L_CELLz=L_CELL;
    d=2.*RSP;
    s= SOD*d;
    L= s + d*sin(ca);/*%L=side length of triangle, s=space gap length,
d=droplet diameter*/
    H= L*sin(60.*PI/180.);

    gapz=.005*RSP;
    gapx=.005*L;
    gapx_perpendicular=gapx*sin(60.*PI/180.);/*keep in mind that this is a
little less than gapx */
    gapy=gapx_perpendicular;


    condwallZ=RSP*(1.-cos(ca)) + gapz;
    if (condwallZ<=L_CELL*NCZ1){
        NCZ=NCZ1;
    }
    else{
        NCZ=ceil(condwallZ/L_CELLz);
        //L_CELLz= L_CELL*(1.-cos(ca));//Here zmax is defined by RSP+gap,
therefore cos(ca) is what it should be.
    }
    /* The above basically ensures that we maximize our use of our cells in
the z-direction.  Remember that I take the ceiling of the cells for the z-
direction, therefore a lot of times I will have a cell that is inbetween a
boundary.  The above if/else ensures that I use that last cell to its maximum
before I start adding more cells. */
```

```
    //%the following is just used to define the lines that make the triangle
prefix "c" refers to center of droplet, [x,y,z] corresponds to the
coordinates, and the number corresponds to the droplet number

    cz= d/2.+ gapz;//%center of droplets is always at D/2 + gapz, regardless
of contact angle!!



    cx1=-L/2.;
    cy1=0;
    cz1=cz;

    cx2=0;
    cy2=H;
    cz2=cz;

    cx3=L/2.;
    cy3=0;
    cz3=cz;

    d1p[0]=cx1;
    d1p[1]=cy1;
    d1p[2]=cz1;

    d2p[0]=cx2;
    d2p[1]=cy2;
    d2p[2]=cz2;

    d3p[0]=cx3;
    d3p[1]=cy3;
    d3p[2]=cz3;




    m1=(cy2-cy1)/(cx2-cx1);// %positive slope between drops 1&2 bc of
reversed coordinates
    m2=(cy2-cy3)/(cx2-cx3);//%negative slope between drops 2&3 bc of reversed
coordinates




    /* print-out set constants */

    xdum = TAMB;
    fprintf(fpo,"\n   TAMB = %8.8e", xdum);
    xdum = TSP;
    fprintf(fpo,"\n   TSP = %8.8e", xdum);
```

```
    xdum = PRESS;
    fprintf(fpo,"\n   PRESS = %8.3e", xdum);
    xdum = CONCA;
    fprintf(fpo,"\n   CONCA = %8.5e", xdum);
    xdum = DT;
    fprintf(fpo,"\n   DT = %8.3e", xdum);
    ndum = INTST;
    fprintf(fpo,"\n   INTST = %d", ndum);
    ndum = NCX;
    fprintf(fpo,"\n   NCX = %d", ndum);
    ndum = NCY;
    fprintf(fpo,"\n   NCY = %d", ndum);
    ndum = NCZ;
    fprintf(fpo,"\n   NCZ = %d", ndum);
    ndum = NPRTMAX;
    fprintf(fpo,"\n   NPRTMAX = %d", ndum);
    xdum = PARTRAT;
    fprintf(fpo,"\n   PARTRAT = %8.3e", xdum);
    xdum = MASSA;
    fprintf(fpo,"\n   MASSA = %8.3e", xdum);
    xdum = MASSB;
    fprintf(fpo,"\n   MASSB = %8.3e", xdum);
    xdum = EVAPPM;
    fprintf(fpo,"\n   EVAPPM = %8.3e", xdum);
    xdum = DMA;
    fprintf(fpo,"\n   DMA = %8.3e", xdum);
    xdum = DMB;
    fprintf(fpo,"\n   DMB = %8.3e", xdum);
    xdum = MDA;
    fprintf(fpo,"\n   MDA = %8.3e", xdum);
    xdum = IZROT;
    fprintf(fpo,"\n   IZROT = %8.3e", xdum);
    xdum = L_CELL;
    fprintf(fpo,"\n   L_CELL = %8.3e", xdum);
    xdum = RSP;
    fprintf(fpo,"\n   RSP = %8.3e", xdum);
    xdum = ACCOM;
    fprintf(fpo,"\n   ACCOM = %8.3e", xdum);



    srandom(19);              /* initialize seed for random number generator
*/

    nprt = INPRTNO;           /* set initial particle count */
    wfluxres = 0.0;           /* initialize accumulators */
    wfluxres2 = 0.0;          /* initialize accumulators */
    wfluxres_delt = 0.0;            /* initialize accumulators */
     nconsum = 0;
     errsum = 0.;
    afluxres1 = 0.0;
    afluxres2 = 0.0;


    /*THE REFERENCE POINT FOR CALCULATING CELL NUMBER (and therefore
```

```
corresponding to xmax,ymax, and zmax) is at the [-L/2,0,0] global coordinate.
)*/

    xmax = NCX*L_CELL; /*STRICTLY USED FOR CALCULATING CELL NUMBER, AND
NOTHING ELSE.  NOTE THAT IT DOESN'T NECESSARILY COINCIDE WITH THE condwallZ
because I sometimes let my "last" cell be inbetween a boundary line...think
about it*/
    ymax = NCY*L_CELL;/*STRICTLY USED FOR CALCULATING CELL NUMBER, AND
NOTHING ELSE.  NOTE THAT IT DOESN'T NECESSARILY COINCIDE WITH THE condwallZ
because I sometimes let my "last" cell be inbetween a boundary line...think
about it*/
    zmax = NCZ*L_CELLz;/*STRICTLY USED FOR CALCULATING CELL NUMBER, AND
NOTHING ELSE.  NOTE THAT IT DOESN'T NECESSARILY COINCIDE WITH THE condwallZ
because I sometimes let my "last" cell be inbetween a boundary line...think
about it */ /*Consider modifying NCZ rather than L_CELLz to try to leave the
cells cubic!!!!*/


     degain = RADFLUX*DT*(PI/2.)*RSP*RSP;
    lost = 0;
    again = 0;
     Ecgain = 0.0;
    Einlower= 0.0;
     Einlower1 = 0.0;
    Einlower2 = 0.0;
    Einlower3 = 0.0;
    elostconducsum=0.0;
    elost3dropseg_stepsum=0.0;
    Egain3dropseg_stepsum=0.0;
     Eg = 0.0;
    egain1sum=0.0;
    egain2sum=0.0;
    egain3sum=0.0;
    elost1sum=0.0;
    elost2sum=0.0;
    elost3sum=0.0;
    Eg0h20print=0.0;

     sphemits1 = 0;
    sphemits2 = 0;
     sphemits3 = 0;

     spherehits1 = 0;
    spherehits2 = 0;
    spherehits3 = 0;
    spacescape = 0;


    wallhit1_delt=0;
    wallhit2_delt=0;
    wallhit3_delt=0;
    condwallhit_DT=0;


     nocoll = 0;
     collcount = 0;
```

```
     nocolrot = 0;
     for (i=0; i< NCELLS; i++) {
        /* set arrays to zero */
        cell[i].countkind1 = 0;
        cell[i].countkind2 = 0;
        cell[i].etrsum = 0.;
        cell[i].erotsum = 0.;
        cell[i].usum = 0.;
     }
     Tsp = TSP;
    EvapT = EVAPPM;
    den = PRESS/(KB*TAMB);    /* molecular density per cu meter */
     D12 = (DMA + DMB)/2.;
     rm12 = MASSA/MASSB;
     flam1 = 1./(1.414*den* CONCA*DMA*DMA
                + sqrt(1. + rm12)*den*(1.- CONCA)*D12*D12);
     flam1 = flam1/PI;
     flam2 = 1./(1.414*den*(1. - CONCA)*DMB*DMB
                + sqrt(1. + (1./rm12))*den*CONCA*D12*D12);
     flam2 = flam2/PI;
    /* ambient mean free path */
    freelam = CONCA*flam1 + (1. - CONCA)*flam2;

    /* ambient particles per cell */
    freeparden = den*L_CELLz*L_CELL*L_CELL/PARTRAT;
     jmax = NCX*NCY;


    initcon();    /* fill particle structures */


    //printf("failed");
    //exit(EXIT_FAILURE);
     iprof = PINT;
     plim = PINT - 1;
    k=0;                        /* loop to step simulation */


    //calculation of kinetic theory collision rate
    sigab=(DMA/2.)+(DMB/2.);
    rmab= MASSA*MASSB/(MASSA+MASSB);//reduced mass
    VbarAB=sqrt(8.*KB*TAMB/(PI*rmab));
    VbarA=sqrt(8.*KB*TAMB/(PI*MASSA));
    VbarB=sqrt(8.*KB*TAMB/(PI*MASSB));


    nA=(den*CONCA);
    nB=(den*(1-CONCA));
    octvol=PI/2.*((4.*RSP*4.*RSP*4.*RSP)-(RSP*RSP*RSP))*(1.-cos(ca))/3.;
/*the /2 acccounts for the octant SINCE NOT DIVIDING BY FOUR GIVES FOR A
HEMISPHERE*/
    kTCRAB=PI*nA*nB*sigab*sigab*VbarAB*octvol; /*Kinetic theory collision
rate[coll/s] one octant of a spherical shell*/

    kTCRA=sqrt(2)*nA*PI*DMA*DMA*VbarA*octvol;
    kTCRB=sqrt(2)*nB*PI*DMA*DMB*VbarB*octvol;
```

```
kTCR=kTCRA+kTCRB+kTCRAB;

//printf("\n initcon");
while (k < NSTEPS) {            /* through time steps      */

  iprof++;
  nncon1 = 0;
    nncon2 = 0;
    nncon3 = 0;
    elost1 = 0.0;
    elost2 = 0.0;
    elost3 = 0.0;

    egain  = 0.0;
    egain1 = 0.0;
    egain2 = 0.0;
    egain3 = 0.0;
    egain3dropseg_indiv_step = 0.0;
    elost3dropseg_indiv_step = 0.0;
    sphemits1_DT=0.0;
    sphemits2_DT=0.0;
    sphemits3_DT=0.0;
    spherehits1_DT=0.0;
    spherehits2_DT=0.0;
    spherehits3_DT=0.0;
    influxprint=0.0;

    wallhit1_DT=0;
    wallhit2_DT=0;
    wallhit3_DT=0;
    spacescape_DT = 0;

    condwallhit_DT=0;

  delt = ((double) (k + 1))*DT;

    statcell(k);    /* compile cell statistics */




    advance(k);      /* move particles one time step */

    ambemit(k);      /* add particles from ambient */

    dropemit(k);     /* add particles emitted from drop */
    As1=PI*RSP*RSP*(1-cos(ca))/3;
    elostconducsum+=elostconduc;
    Egain3dropseg_stepsum+=egain1+egain2+egain3;
    elost3dropseg_stepsum+=elost1+elost2+elost3;
    egain3dropseg_indiv_step=egain1+egain2+egain3;
    elost3dropseg_indiv_step=elost1+elost2+elost3;
    egain1sum+=egain1;
```

```
        egain2sum+=egain2;
        egain3sum+=egain3;
        elost1sum+=elost1;
        elost2sum+=elost2;
        elost3sum+=elost3;
        wfluxres_delt+=wfluxres2;



        collect();       /* collect candidate collision pairs */

        collide();       /* execute collisions */


        if (iprof > plim){
            iprof -= PINT;
            fprintf(fpo,"\n\n\n\n\n\n\n\n\n\n step = %d done, nocoll = %d", k,
nocoll);
            fprintf(fpo,"\n            nocolrot = %d", nocolrot);
            fprintf(fpo,"\n            nprt = %d", nprt);
                  fprintf(fpo,"\n Ecgain = %8.3e", Ecgain);
                  fprintf(fpo,"\n errsum = %8.3e", errsum);

            fprintf(fpo,"\n spherehits1 = %d ", spherehits1);
            fprintf(fpo,"\n spherehits2 = %d ", spherehits2);
            fprintf(fpo,"\n spherehits3 = %d ", spherehits3);

            flux=PARTRAT*spherehits1/As1/delt;
            fprintf(fpo,"\n influx1_delt= %8.3e", flux);
            flux=PARTRAT*spherehits2/As1/delt;
            fprintf(fpo,"\n influx2_delt= %8.3e", flux);
            flux=PARTRAT*spherehits3/As1/delt;
            fprintf(fpo,"\n influx3_delt= %8.3e", flux);
            flux=PARTRAT*(spherehits1+spherehits2+spherehits3)/(3*As1)/delt;
            fprintf(fpo,"\n average_influx_delt= %8.3e", flux);



            fprintf(fpo,"\n\n sphemits1 = %d ", sphemits1);
            fprintf(fpo,"\n sphemits2 = %d ", sphemits2);
            fprintf(fpo,"\n sphemits3= %d ", sphemits3);


            flux=PARTRAT*sphemits1/As1/delt;
            fprintf(fpo,"\n\n outflux1_delt= %8.3e", flux);
            flux=PARTRAT*sphemits2/As1/delt;
            fprintf(fpo,"\n outflux2=_delt %8.3e", flux);
            flux=PARTRAT*sphemits3/As1/delt;
            fprintf(fpo,"\n outflux3_delt= %8.3e", flux);
            flux=PARTRAT*(sphemits1+sphemits2+sphemits3)/(3*As1)/delt;
            fprintf(fpo,"\n average_outflux_delt= %8.3e", flux);


            //Aside=L*condwallZ; //FOR DROPLETS OFF
            Aside=L*condwallZ-2.*(   PI*RSP*RSP/4-( 0.5*RSP*RSP*(PI/2.-
ca+sin(ca)*cos(ca)) )   );//for DROPLETS ON
```

```
            flux=PARTRAT*wallhit1_delt/Aside/delt;
            fprintf(fpo,"\n\n wallflux1_delt= %8.3e", flux);
            flux=PARTRAT*wallhit2_delt/Aside/delt;
            fprintf(fpo,"\n wallflux2_delt= %8.3e", flux);
            flux=PARTRAT*wallhit3_delt/Aside/delt;
            fprintf(fpo,"\n wallflux3_delt= %8.3e", flux);

            flux=PARTRAT*wallhit1_DT/Aside/DT;
            fprintf(fpo,"\n wallflux1_DT= %8.3e", flux);
            flux=PARTRAT*wallhit2_DT/Aside/DT;
            fprintf(fpo,"\n wallflux2_DT= %8.3e", flux);
            flux=PARTRAT*wallhit3_DT/Aside/DT;
            fprintf(fpo,"\n wallflux3_DT= %8.3e", flux);


            //compute heat transfer coefficient
            At = 0.25*sqrt(3.)*(L)*(L); /* (same thing as 0.5*L*H since
H=L*sqrt(3)/2) surface areas of upper triangle*/
            //Aw=At;//FOR DROPLETS OFF

            Aw=At-.5*PI*(RSP*sin(ca))*(RSP*sin(ca)); /* FOR DROPLETS ONcold
wall area inbetween the droplet segments */


            flux=PARTRAT*spacescape/(At)/delt;
            fprintf(fpo,"\n spacescape_flux1_delt= %8.3e", flux);
            flux=PARTRAT*spacescape_DT/(At)/DT;
            fprintf(fpo,"\n spacescape_flux1_DT= %8.3e", flux);
            flux=PARTRAT*condwallhit_delt/(Aw)/delt;
            fprintf(fpo,"\n condensingwall_flux1_delt= %8.3e", flux);
            flux=PARTRAT*condwallhit_DT/(Aw)/DT;
            fprintf(fpo,"\n condensingwall_flux1_DT= %8.3e", flux);
            fprintf(fpo,"\n AmbientFlux = %8.3e", influxprint);



            fprintf(fpo,"\n\n j0print_delt= %8.3e", j0print);
            fprintf(fpo,"\n Eg0h20print= %8.16e", Eg0h20print);
            fprintf(fpo,"\n Eg0h20print/delt= %8.16e", Eg0h20print/delt);
            fprintf(fpo,"\n (Eg-Eg0h20print)/delt= %8.16e", (Eg-
Eg0h20print)/delt);

            fprintf(fpo3,"\n %8.16e %8.16e %8.16e", Eg0h20print,
Eg0h20print/delt,(Eg-Eg0h20print)/delt);
            fprintf(fpo4,"\n %8.16e", Eg/delt);



                  fprintf(fpo,"\n\n Eg = %8.16e", Eg/delt);
            fprintf(fpo,"\n nconsum = %d ", nconsum);
            fprintf(fpo,"\n collcount = %d ", collcount);
                  fprintf(fpo,"\n Tsp = %8.15e", Tsp);
            fprintf(fpo,"\n Tamb = %8.15e", TAMB);
                  fprintf(fpo,"\n Twall = %8.15e", Tw);
```

```
            fprintf(fpo,"\n Tamb-Tsp = %8.15e", TAMB-Tsp);
                  fprintf(fpo,"\n Tsp-Tw = %8.15e", Tsp-Tw);



            collcount_time=PARTRAT*collcount/delt; /*molecule
collisions/sec*/
            fprintf(fpo,"\n collcount_delt = %8.3e ", collcount_time);
            fprintf(fpo,"\n  kinetic_collcount_time= %8.3e", kTCR);









            Einlower=Einlower1+Einlower2+Einlower3;
            h_conden_delt_einlower=(Einlower/delt)/(At*(TAMB-Tw));
            h_conden_delt_egainlost=((Egain3dropseg_stepsum-
elost3dropseg_stepsum)/delt)/(At*(TAMB-Tw));


            h_conduc_delt=(elostconducsum/delt)/(At*(TAMB-Tw));
            h_main1=((Egain3dropseg_stepsum-Eg0h20print)/delt)/(At*(TAMB-
Tw));
            h_main2=((Eg-Eg0h20print)/delt)/(At*(TAMB-Tw));



            h_conden_DT=(( (egain3dropseg_indiv_step)-
(elost3dropseg_indiv_step) )/DT)/(At*(TAMB-Tw));

            h_conduc_DT=(elostconduc/DT)/(At*(TAMB-Tw));




            /*AT LEAST THESE SHOULD STABILIZE!!!*/
            fprintf(fpo,"\n\n HTC_main1(W/m2-K) = %8.3e", h_main1);
            fprintf(fpo,"\n HTC_main2(W/m2-K) = %8.3e", h_main2);

            fprintf(fpo,"\n HTC_fluxes_delt_einlower_main3(W/m2-K) = %8.3e",
h_conden_delt_einlower);
            fprintf(fpo,"\n\n HTC_fluxes_delt_egainlost(W/m2-K) = %8.3e",
h_conden_delt_egainlost);
            fprintf(fpo,"\n HTC_conduction_delt_einlower(W/m2-K) = %8.3e",
h_conduc_delt);

            fprintf(fpo,"\n HTC_fluxes_DT_egainlost(W/m2-K) = %8.3e",
h_conden_DT);
            fprintf(fpo,"\n HTC_conduction_DT(W/m2-K) = %8.3e", h_conduc_DT);


            fprintf(fpo,"\n\n egain1/DT= %8.10e", egain1/DT);
```

```
                fprintf(fpo,"\n egain2/DT = %8.10e", egain2/DT);
                fprintf(fpo,"\n egain3/DT = %8.10e", egain3/DT);

          fprintf(fpo,"\n elost1/DT = %8.10e", elost1/DT);
                fprintf(fpo,"\n elost2/DT = %8.10e", elost2/DT);
                fprintf(fpo,"\n elost3/DT = %8.10e", elost3/DT);

          fprintf(fpo,"\n\n egain1sum/delt= %8.10e", egain1sum/delt);
                fprintf(fpo,"\n egain2sum/delt = %8.10e", egain2sum/delt);
                fprintf(fpo,"\n egain3sum/delt = %8.10e", egain3sum/delt);

          fprintf(fpo,"\n elost1sum/delt = %8.10e", elost1sum/delt);
                fprintf(fpo,"\n elost2sum/delt = %8.10e", elost2sum/delt);
                fprintf(fpo,"\n elost3sum/delt = %8.10e", elost3sum/delt);
          fprintf(fpo,"\n Egain3dropseg_stepsum/delt = %8.10e",
Egain3dropseg_stepsum/delt);
          fprintf(fpo,"\n elost3dropseg_conden_stepsum/delt = %8.10e",
elost3dropseg_stepsum/delt);

          fprintf(fpo,"\n\n Epre1average = %8.10e", egain1/spherehits1_DT);
          fprintf(fpo,"\n Epre2average = %8.10e", egain2/spherehits2_DT);
          fprintf(fpo,"\n Epre3average = %8.10e", egain3/spherehits3_DT);
          fprintf(fpo,"\n EpreALLaverage = %8.10e",
(egain1/spherehits1_DT+egain2/spherehits2_DT+egain3/spherehits3_DT)/3. );


          fprintf(fpo,"\n Epost1average = %8.10e", elost1/sphemits1_DT);
          fprintf(fpo,"\n Epost2average = %8.10e", elost2/sphemits2_DT);
          fprintf(fpo,"\n Epost3average = %8.10e", elost3/sphemits3_DT);
          fprintf(fpo,"\n EpostALLaverage = %8.10e",
(elost1/sphemits1_DT+elost2/sphemits2_DT+elost3/sphemits3_DT)/3.);




          fprintf(fpo,"\n\n Egain3dropseg_stepsum = %8.10e",
Egain3dropseg_stepsum);
          fprintf(fpo,"\n elost3dropseg_conden_stepsum = %8.10e",
elost3dropseg_stepsum);//pure condensation, no conduction
          fprintf(fpo,"\n elost3dropseg_conden_ENERGYBAL_stepsum = %8.10e",
Eg0h20print);
          fprintf(fpo,"\n elost3dropseg_conden+conduc_ENERGYBAL_stepsum =
%8.10e", Eg0h20print+elostconducsum);
          fprintf(fpo,"\n elostconduc_stepsum = %8.10e", elostconducsum);
          fprintf(fpo,"\n ELOSTTOTAL_CONDENS_CONDUC_stepsum = %8.10e",
elostconducsum+elost3dropseg_stepsum);

          fprintf(fpo,"\n\n wfluxres_delt =%8.10e", wfluxres_delt/delt);
          fprintf(fpo,"\n
wfluxres_delt/3elost_stepsum_percent=%8.10e",100*wfluxres_delt/elost3dropseg_
stepsum);
          fprintf(fpo,"\n
wfluxres_delt/3egain_stepsum_percent=%8.10e",100*wfluxres_delt/Egain3dropseg_
stepsum);
          fprintf(fpo,"\n
```

```
wfluxres_delt/Einlower_percent=%8.10e",100*wfluxres_delt/Einlower);


            fprintf(fpo,"\n\n egain3dropseg_indiv_step = %8.10e",
egain3dropseg_indiv_step);
            fprintf(fpo,"\n elost3dropseg_conden_indiv_step = %8.10e",
elost3dropseg_indiv_step);
            fprintf(fpo,"\n elostconduc_indiv_step = %8.10e", elostconduc);
            fprintf(fpo,"\n ELOSTTOTAL_CONDENS_CONDUC_indiv_step = %8.10e",
elostconduc+elost3dropseg_indiv_step);
            fprintf(fpo,"\n 3dropgain - 3droploss = %8.10e
\n",(egain3dropseg_indiv_step)-(elost3dropseg_indiv_step));
            fprintf(fpo2,"\n %8.6e %8.10e",delt, (egain3dropseg_indiv_step)-
(elost3dropseg_indiv_step));


            fprintf(fpo,"\n\n wfluxres_DT=%8.10e",wfluxres2/DT);
            fprintf(fpo,"\n wfluxres=%8.10e",wfluxres2);
            fprintf(fpo,"\n
wfluxres/3elost_percent=%8.10e",100*wfluxres2/elost3dropseg_indiv_step);
            fprintf(fpo,"\n
wfluxres/3egain_percent=%8.10e",100*wfluxres2/egain3dropseg_indiv_step);
            fprintf(fpo,"\n
wfluxres/Einlower_percent=%8.10e",100*wfluxres2/(egain-elost));

            fprintf(fpo,"\n\n spherehits1_DT = %8.10e", spherehits1_DT);
            fprintf(fpo,"\n spherehits2_DT = %8.10e", spherehits2_DT);
            fprintf(fpo,"\n spherehits3_DT = %8.10e", spherehits3_DT);
            fprintf(fpo,"\n spherehitsALL_DT = %8.10e",
spherehits1_DT+spherehits2_DT+spherehits3_DT);



            fprintf(fpo,"\n sphemits1_DT = %8.10e", sphemits1_DT);
            fprintf(fpo,"\n sphemits2_DT = %8.10e", sphemits3_DT);
            fprintf(fpo,"\n sphemits3_DT = %8.10e", sphemits3_DT);
            fprintf(fpo,"\n sphemitsALL_DT = %8.10e",
sphemits1_DT+sphemits2_DT+sphemits3_DT);


            fprintf(fpo,"\n spherehitsALL_DT-sphemitsALL_DT = %8.10e", (
spherehits1_DT+spherehits2_DT+spherehits3_DT)-
(sphemits1_DT+sphemits2_DT+sphemits3_DT) );









            /* print-out profiles for this time step */

            for (i=0; i < NCZ; i++){
                ip = i*jmax;
```

```
                xdum = zmax*(0.5 + ((double) i))/((double) NCZ);
                fprintf(fpo,"\n z = %8.3e cellden = %8.3e conc1 = %8.3e temp
= %8.3e",
                        xdum, mnp[ip], cell[ip].molconc1, cell[ip].temp);
            }

            for (i=0; i < NCZ; i++){
                ip = i*jmax + i;
                xdum = zmax*(0.5 + ((double) i))/((double) NCZ);
                xdum = 1.414*xdum;
                fprintf(fpo,"\n r = %8.3e cellden = %8.3e conc1 = %8.3e temp
= %8.3e",
                        xdum, mnp[ip], cell[ip].molconc1, cell[ip].temp);
            }
        }  /* end of intermittent write */
        k++;
    }   /* end of time step while loop */



    fclose(fpo);        /* close file outDSMC with pointer fpo */
    fclose(fpo2);       /* close file outDSMC with pointer fpo */
    fclose(fpo3);       /* close file outDSMC with pointer fpo */
    fclose(fpo4);       /* close file outDSMC with pointer fpo */


}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */









void initcon()
{
    /* BBBBBBBBBBB  Function to initialize particle structures */

    /*MATLAB GLOBAL CONCA RSP L_CELL NCX NCY PI NCZ PARTRAT MASSA
     MASSB KB TAMB global nprt xmax ymax zmax prt global prt*/


    //%%%%%%%%These will be local variables%%%%%%%%%%%%%%%
    int j, xc, yc, zc;
    double cT, phi, r, Rtest, eps, Ie;
    double costheta, sintheta, snx, sny, snz, rp;
    double R1;
    double prt_en, average_prt_en, prt_en_indiv;
    //%%%%%%%%%%%%%%end local variables%%%%%%%%%%%%%%%%%%%
```

```
j=0; //j denotes particle number
prt_en=0;
prt_en_indiv=0;
average_prt_en=0;

while (j < nprt) {
    prt[j].kind = (int) ((1. - CONCA) + 1.0
                         + ((double) random())/RANMAX);
    /* (1-CONCA)=CONCB
     (1-CONCA)=molar concentration of species2*/


    // randomly select entry point for new particle */
    spaceprtstart(j);
    //end random select entry point for new particle



    xc = (int)( (prt[j].x + L/2) *NCX/xmax ); //
    yc = (int)(prt[j].y*NCY/ymax); //
    zc = (int)(prt[j].z*NCZ/zmax); //




    switch (prt[j].kind){
        case 1:  /* water */
            prt[j].mass = PARTRAT*MASSA;
            /* rotational energy for water */
            Rtest = 0.97*((double) random())/RANMAX;
            eps = 0.002;
            Ie = 0.0;
            while (Ie < Rtest) {
                eps += 0.1;
                Ie += (sqrt(eps - 0.1)*exp(-eps + 0.1)
                       + sqrt(eps)*exp(-eps))*0.05642;
            }
            prt[j].erot = PARTRAT*KB*TAMB*eps;
            break;
        case 2: /*Nitrogen */
            prt[j].mass = PARTRAT*MASSB;
            /* rotational energy for N2 */
            R1=((double) random())/RANMAX;
            if (R1<=0 || R1>=1)  {
                while (R1<=0 || R1>=1){
                    R1=((double) random())/RANMAX;
                }
            }
            prt[j].erot = -PARTRAT*KB*TAMB/log(R1);
            /* END rotational energy for N2 */
            break;
    }  /* end switch */
```

```
        cT = sqrt(PARTRAT*KB*TAMB/prt[j].mass); //max avg particle
speed...determined by ambient temperature
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


        if (prt[j].cellno>NCELLS || prt[j].cellno<0 || prt[j].y<0 ||
prt[j].y> m2*fabs(prt[j].x) + H || prt[j].z>condwallZ || prt[j].x>((prt[j].y-
H)/m2) || prt[j].x<(-(prt[j].y-H)/m2) ){
            printf("\n   ZZ = %8.3e", prt[j].z);
            printf("\n   YY = %8.3e", prt[j].y);
            printf("\n   XX = %8.3e", prt[j].x);
            printf("\n   max_x = %8.3e", (prt[j].y-H)/m2);
            printf("\n   cellno = %d", prt[j].cellno);
            printf("initcon");
            exit(EXIT_FAILURE);

        }

        phi = 2.*PI*( ((double) random())/RANMAX);//to determine random
velocity direction(this is the phi used as a sampling variable)



        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].u = cT*r*cos(phi);//random u velocity.  can these ever be
negative?  they should be able to be, no?
        prt[j].v = cT*r*sin(phi);//random v velocity.  can it be negative?
no!  why not?
        //the cos(phi) should take care of including a NEGATIVE since phi
goes from 0 to 2pi!!!!



        //here just generating another random number, but no need to generate
2 as in u and v
        phi = 2.*PI*( ((double) random())/RANMAX );
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].w = cT*r*cos(phi);
        prt_en_indiv=0.5*prt[j].mass*(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w) + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp);
```

```
        prt_en+=prt_en_indiv;
        average_prt_en=prt_en/(j+1);


        j=j+1;//leave this one ALONE!!!!


    }//ends the main while loop
}//ends function
// EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


// EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

void spaceprtstart(j)
{
    int flag,i;
    double xrangemin, xrangemax, yrangemin, yrangemax, xcoor, xrange, ycoor;
    double xpoint, ypoint,zpoint, d2d1, d2d2, d2d3;



    xrangemin=cx1 + gapx;//-L/2.;/*minimum coordinate at the base of the
triangle*/
    xrangemax=cx3 - gapx;//L/2.;/*maximum coordinate at the base of the
trianlge*/
    yrangemin=0. + gapy;/*minimum coordinate anywhere*/
    yrangemax=cy2 - gapy;/*maximum coordinate in the middle of the triangle*/
    flag=1;

    while (flag>0){
        /* X-range as a funcion of y-position for triangle*/
        /*RECENTLY MADE GLOBALcondwallZ= RSP*(1-cos(ca));*/ /*max z
coordinate, depending on the contact angle of the droplet*/

        /*just to initialize the loop*/
        xcoor=2.;/* xcoor is the actual possible coordinate, except here
where it is just used for initializing */
        xrange=1.;
        i=0;
        /*end initialization*/

        while (xcoor < -xrange || xcoor > xrange) {

            i=i+1;
            xcoor=xrangemin+ ((double) random())/RANMAX*(xrangemax-
xrangemin);
            ycoor=yrangemin+ ((double) random())/RANMAX*(yrangemax-
yrangemin);

            xrange=(yrangemax-ycoor)*(L/2./H); /*%y=mx + b, m= -H/(L/2),
x=(y-b)/m = -(b-y)/m, b=yrangemax, (and x is really xrangemax).
                                                %note that xrange is the
distance from the middle of the triangle (cut halfways down the middle)  to a
corresponding edge for the height!!  It's NOT the whole
                                                %width at that height! NOTE
```

```
THAT XRANGE IS JUST AN ABSOLUTE VALUE...also note that m has a negative in
front of the expression, therefore (L/2./H) is correct (no negative
needed)!!*/

            if (xcoor >= -xrange && xcoor <= xrange){
                /*If the particle is within acceptable range, designate them
as the coordinate*/
                xpoint=xcoor;
                ypoint=ycoor;
                zpoint=0.5*gapz + (condwallZ-gapz)*((double)
random())/RANMAX;
                /* don't forget that condwallZ points down
                 %note that it comes into this if it automatically does not
go
                 %into the next step of the while loop*/
                /*NOTE THAT THE zpoint IS CHOSEN TO START AT 0.5gapz DOWN
FROM THE AMBIENT, and ITS MAXIMUM IS 0.5*gapz RIGHT ABOVE THE condwallz,
hence the subraction of the WHOLE gapz, not just 0.5*gapz */
            }


        }



        /*Once coordinates are within range for the cooresponding y, It is
time to determine if a droplet was stricken*/

        /*Z= D/2;//Distance from coordinate system to the center of sphere
outlined by
         the droplet.  NOT the distance to the condensing surface!*/
        double ppos[3]={xpoint,ypoint,zpoint};/*particle position*/



        d2d1= pointdist(ppos,d1p); /*sqrt(  sum((ppos-d1p).^2)  ); //distance
to droplet center 1  */
        d2d2= pointdist(ppos,d2p); /*sqrt(  sum((ppos-d2p).^2)  ); //distance
to droplet center 2  */
        d2d3= pointdist(ppos,d3p); /*sqrt(  sum((ppos-d3p).^2)  ); //distance
to droplet center3 */



        if (d2d1 <= RSP || d2d2 <= RSP || d2d3 <= RSP){

            flag=1;
            /*the particle lies within a droplet, generate a new position*/
        }
        else{
            /*nothing has been stricken...accept particle and just continue*/
            prt[j].x = xpoint;
            prt[j].y = ypoint;
```

```
            prt[j].z = zpoint;
            flag=-1;
        }
    }


}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to advance particle to new location */
void advance(k)
{
    int i,j, xc, yc, zc;




    j=0;// j is the indice for particle number
    while (j < nprt)

    {
        dtres = DT;/* this initializes dtres, but it is ammended later*/

        losscheck = 1; /*set to one if it has not left, if it has left,
particle is removed from simulation and replaced by last particle and
losscheck is set to ZERO to make the check on that "replacING" (the one that
replaced the one that left the simulation) particle.  Basically, it is set to
one for every particle initially.  It can be set to zero inside of chkspace*/
        i=0;
        while (dtres>0) /*this moves the particle until it runs out of the
residual time "dtres"*/

        {
            i++;


            prt[j].x = prt[j].x + prt[j].u * dtres;
            prt[j].y = prt[j].y + prt[j].v * dtres;
            prt[j].z = prt[j].z + prt[j].w * dtres;


            chkspace(j);
            /*note that many actions here could move the particle back a bit
in comparison to the above movement. Basically the particle is moved
dtres[seconds] forward, but then chkspace analyzes that movement, and if a
boundary was struck within that movement, the particle is moved back a bit to
the the point (time and position) where it hit the boundary (note that this
keeps on going until dtres is subtracted down to zero from DT)

             //note that when dtres reaches zero, it only moves to the next
particle if losscheck was never set to zero(therefore, it makes sense to set
dtres to zero at end of loseprt).  In otherwords, it will not go into the
next particle if loseprt was somehow used (if the particle was either
absorbed or lost into space)...(make sure that this still makes sense to keep
doing this same procedure of replacing the lost particles within the same
```

```
time-step)*/

            /*loseprt sets dtres=0 so that it can immediately exit this
dtres-loop once the particle has been lost to space, and losscheck=0 so that
it can "move" the new "replacING" particle..so it leaves the same j
below...(j=j+losscheck)*/
        }



        xc = (int)( (prt[j].x + L/2) *NCX/xmax );
        yc = (int)(prt[j].y*NCY/ymax);
        zc = (int)(prt[j].z*NCZ/zmax);



        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;



        j = j + losscheck;
    }
}



/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

void chkspace(j) /* BBBBBBBBBBB  Function to check whether particle escapes
and handle
                  %I think it also checks whether it hit a specular surface
too*/
{
    //%%%%%%%%These will be local variables%%%%%%%%%%%%%%%
    int  xc, yc, zc;
    double dt1, dt2, Epre;
    double A,B1,C1,D1,B2,C2,D2, B3, C3, D3, b;
    double R0, R1, R2, xi, yi, zi, cmag;
    double costheta, sintheta, phi, cn, cp1, cp2, snx, sny, snz;
    double a1, a2, snynz, b0, b1, b2, rsq, rgsq;
    double denominator;
    double xhit, xhitc, yhit, yhitc, zhit, zhitc, phiv, thetav, phipos,
thetapos;
    double c,u,v,w;
    double rot1[3][3];
    double totrot[3][3];
    double randvel[3][1];
    //%%%%%%%%%%%%%%end local variables%%%%%%%%%%%%%%%%%%


    /*

     %Notice that Einlower is
     %used right after Epost, but notice that Epost is different in two
factors:
     %at the end of the condensible and noncondensible emission!!!!  */
```

```
    /*Epost is local to dropemit and collide!!!*/



    /*this initiates boundnum and assumes no drop is hit unless otherwise
determined from the tests below*/
    boundnum=0;



    b=H;//%(s+d*sin(ca))*sqrt(3)/2;


    /* check if sphere is hit   ... using time to determine if drop is hit*/
    xold = prt[j].x-prt[j].u*dtres;
    yold = prt[j].y-prt[j].v*dtres;
    zold = prt[j].z-prt[j].w*dtres;
    /*%equation for offset sphere: (x-offx)^2 + (y-offy)^2 + (z-offz)^2=
radius^2
     %the logic that we are using is that we can set x equal to (xold +
u*delt),
     %and y and z in the same analagous way.  Then we parameterize*/

    A = prt[j].u*prt[j].u + prt[j].v*prt[j].v + prt[j].w*prt[j].w;// %u^2 +
v^2 + w^2=c2 [m2/s2]... A=speed^2
    dtmin=dtres; /*%this ensures there is a dtmin regardless if all D1's
(below) are negative.*/
    /*%from here on, dtmin is basically the minimum positive time resulting
from
     %any of the checks.*/

    //%% droplet 1 time-check
    B1 = 2.*((xold - cx1)*prt[j].u+(yold - cy1)*prt[j].v+ (zold -
cz1)*prt[j].w); //%2*(xold*u + yold*v + zold*w)[m2/s]
    C1 = (xold*xold - 2.*xold*cx1 + cx1*cx1) + (yold*yold - 2.*yold*cy1 +
cy1*cy1) + (zold*zold - 2.*zold*cz1 + cz1*cz1) - RSP*RSP; /*% [m2]xold^2 +
yold^2 +zold^2 -RSP^2...comes from distance^2=x^2 + y^2 + z^2...therefore if
C is negative, inside droplet.  if positive, outside droplet.*/
    D1 = B1*B1 - 4.*A*C1;/*% if d is negative, outside droplet.  if positive,
hits droplet.*/
    if (D1 >= 0){
        dt1 = 0.5*(-B1 + sqrt(D1))/A;
        dt2 = 0.5*(-B1 - sqrt(D1))/A;
        if (dt1>EPS2 && dt1 < dtmin){
            dtmin=dt1;
            boundnum=1;
        }
        if (dt2>EPS2 && dt2<dtmin){
            dtmin=dt2;
            boundnum=1;
        }
    }

    /*%% droplet 2 time-check*/
```

```
    B2 = 2.*((xold - cx2)*prt[j].u+(yold - cy2)*prt[j].v+ (zold -
cz2)*prt[j].w);// %2*(xold*u + yold*v + zold*w)[m2/s]
    C2 = (xold*xold - 2.*xold*cx2 + cx2*cx2) + (yold*yold - 2.*yold*cy2+
cy2*cy2) + (zold*zold - 2.*zold*cz2 + cz2*cz2) - RSP*RSP; /*% [m2]xold^2 +
yold^2 +zold^2 -RSP^2...comes from distance^2=x^2 + y^2 + z^2...therefore if
C is negative, inside droplet.  if positive, outside droplet.*/
    D2 = B2*B2 - 4.*A*C2;/*%if d is negative, outside droplet.  if positive,
hits droplet.*/
    if (D2 >= 0){
        dt1 = 0.5*(-B2 + sqrt(D2))/A;
        dt2 = 0.5*(-B2 - sqrt(D2))/A;
        if (dt1 >EPS2 && dt1 < dtmin){
            dtmin=dt1;
            boundnum=2;
        }
        if (dt2>EPS2 && dt2<dtmin){
            dtmin=dt2;
            boundnum=2;
        }
    }

    //%% droplet 3 time-check
    B3 = 2.*((xold - cx3)*prt[j].u+(yold - cy3)*prt[j].v+ (zold -
cz3)*prt[j].w);/* %2*(xold*u + yold*v + zold*w)[m2/s]*/
    C3 = (xold*xold - 2.*xold*cx3 + cx3*cx3) + (yold*yold - 2.*yold*cy3+
cy3*cy3) + (zold*zold - 2.*zold*cz3 + cz3*cz3) - RSP*RSP;/* % [m2]xold^2 +
yold^2 +zold^2 -RSP^2...comes from distance^2=x^2 + y^2 + z^2...therefore if
C is negative, inside droplet.  if positive, outside droplet.*/
    D3 = B3*B3 - 4.*A*C3;/*%if d is negative, outside droplet.  if positive,
hits droplet.*/
    /*%D stands for discriminate...look up quadratic equation and definition
of
     %discriminate if don't understand*/
    if (D3 >= 0){
        dt1 = 0.5*(-B3 + sqrt(D3))/A;
        dt2 = 0.5*(-B3 - sqrt(D3))/A;
        if (dt1>EPS2 && dt1<dtmin){
            dtmin = dt1;
            boundnum = 3;
        }
        if (dt2>EPS2 && dt2<dtmin){
            dtmin = dt2;
            boundnum = 3;
        }
    }

    /*Condensing Wall hit time-check */
    dt1=( condwallZ - zold )/prt[j].w; //%ca=contact angle

    if (dt1>EPS2 && dt1<dtmin){
        dtmin = dt1;
        boundnum = 4;
        /*%% Now need to reflect particle diffusely and move it in the rest
of the time
         %oh snap, don't do anything yet, just set a marker...boundnum*/
```

```
    }

    //%% Specular Wall 1 hit time-check
    /*comes from parameterizing y=mx+b, where b=H y=yold+v*dt, x=xold+u*dt,
dt=(yold-m1*xold-b)/(m1*u-v)*/
    denominator=(m1*prt[j].u -prt[j].v);
    if (denominator != 0){
        dt1= (yold - m1*xold - b)/denominator;
        if ((dt1 >EPS2) && (dt1 < dtmin)){
            dtmin = dt1;
            boundnum = 5;
        }
    }



    //%% Specular Wall 2 hit Check Subfunctions?
    /*comes from parameterizing y=mx+b, where b=H y=yold+v*dt, x=xold+u*dt,
dt=(yold-m2*xold-b)/(m2*u-v)*/
    denominator=(m2*prt[j].u -prt[j].v);
    if (denominator != 0){
        dt1= (yold - m2*xold - b)/denominator;
        if ((dt1 >EPS2) && (dt1 < dtmin)){
            dtmin=dt1;
            boundnum=6;
        }
    }



    //%% Specular Wall 3 hit Check Subfunctions?
    //%y=yo + vt=0.... t=-yo/v
    if (prt[j].v !=0){
        dt1= -yold/prt[j].v; //remember, always based on old???
        if ((dt1 >EPS2) && (dt1 < dtmin)){
            dtmin=dt1;
            boundnum=7;
        }
    }


    //%% Space Escape Check
    if (prt[j].w !=0){
        dt1=-zold/prt[j].w;
        if ((dt1 >EPS2) && (dt1 < dtmin)){
            dtmin=dt1;
            boundnum=8;
            /*%account for that particle being lost*/
        }
    }
```

```
    if (boundnum==0 || dtmin > dtres){ //%it will skip all this if
(dtres<=0).
        /*%this means no droplet was struck, skip all the bottom by leaving
this
         %function*/

        dtres=0;
        /*%basically this says that nothing was hit so the residual dt goes
to
         %zero.
         %Setting dtres=0 just forces the dtres-loop in "advance.m" to end.*/



        /*Return in matlab exits the function...what's the equivalent in .C
programming?...SAME*/
        return;
    }
    /*printf("\n boundarynum=%d",boundnum);*/
    /*%absorption or diffuse reflection*/
    R0=((double) random())/RANMAX;
    if (ACCOM>R0 && boundnum<4){ /*note that the boundnum =0 case is
automatically excluded by the above statement where the function is exitted
when boundnum =0 with "return", since nothing was hit.  Boudaries 1-3 are all
the droplets*/




        /*%particle is absorbed into sphere denoted by boundnum*/
        switch (boundnum){
            case 1:

                spherehits1 = spherehits1 + 1;
                spherehits1_DT++;
                Epre = 0.5*prt[j].mass*(prt[j].u*prt[j].u + prt[j].v*prt[j].v
+ prt[j].w*prt[j].w)
                + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp); /* 1/2mv^2 +
erot + (EvapT is a guess of effective energy of vaporiztion per molecule).
*/

                if (prt[j].kind > 1){ //%if_6
                    /*%if particle is not water, then it is noted as nncon to
                     %emit later*/
                    nncon1=nncon1+1; /*this is added when not water...number
of noncondensable particles*/
                    Epre = Epre - PARTRAT*(EvapT - 3.*KB*Tsp);
                    /*noncondensible...therefore no energy of
vaporization!!*/
                } /*%end if_6*/

                Einlower1 = Einlower1 + Epre;/*note that it's a global
variable...used in dropemit*/
                egain1 = egain1 + Epre; /*%note that it's a global varible*/
```

```
                /*%Is egain the energy gained by the water droplet????!!!!
                 %why add Epre to both...YES.  Einlower is the NET energy, as
it is subtracted from every
                 time something is emitted from that droplet, nothing is
subracted from egain (zeroed out at every time-step*/

                /* replace adsorbed particle with last particle */
                loseprt(j); //%lost particle is replaced with last particle
                return;
                break;

            case 2:
                spherehits2 = spherehits2 + 1;
                spherehits2_DT++;
                Epre = 0.5*prt[j].mass*(prt[j].u*prt[j].u + prt[j].v*prt[j].v
+ prt[j].w*prt[j].w)
                + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp); /* 1/2mv^2 +
erot + (EvapT is a guess of effective energy of vaporiztion per molecule).
*/

                if (prt[j].kind > 1){ //%if_6
                    //%if particle is not water, then it is noted as nncon to
                    //%emit later
                    nncon2=nncon2+1; /*this is added when not water...number
of noncondensable particles*/
                    Epre = Epre - PARTRAT*(EvapT - 3.*KB*Tsp);
                } //%end if_6

                Einlower2 = Einlower2 + Epre;//%note that it's a global
variable...used in dropemit
                egain2 = egain2 + Epre; //%note that it's a global varible
                /*%Is egain the energy gained by the water droplet????!!!!
                 %why add Epre to both...YES.  Einlower is the NET energy, as
it is subtracted from every
                 time something is emitted from that droplet, nothing is
subracted from egain*/

                /* replace adsorbed particle with last particle */
                loseprt(j); /*%lost particle is replaced with last particle
*/
                return;
                break;

            case 3:
                spherehits3 = spherehits3 + 1;
                spherehits3_DT++;
                Epre = 0.5*prt[j].mass*(prt[j].u*prt[j].u
                                        + prt[j].v*prt[j].v +
prt[j].w*prt[j].w)
                + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp); /* 1/2mv^2 +
erot + (EvapT is a guess of effective energy of vaporiztion per molecule).
*/

                if (prt[j].kind > 1){ //%if_6
                    /*%if particle is not water, then it is noted as nncon to
                     %emit later*/
```

```
                    nncon3=nncon3+1; /*%this is added when not water...number
of noncondensable particles*/
                    Epre = Epre - PARTRAT*(EvapT - 3.*KB*Tsp);
                } //%end if_6

                Einlower3 = Einlower3 + Epre;/*%note that it's a global
variable...used in dropemit*/
                egain3 = egain3 + Epre; /*%note that it's a global varible*/
                /*%Is egain the energy gained by the water droplet????!!!!
                 %why add Epre to both...YES.  Einlower is the NET energy, as
it is subtracted from every
                 time something is emitted from that droplet, nothing is
subracted from egain*/

                /* replace adsorbed particle with last particle */
                loseprt(j); //%lost particle is replaced with last particle
                return;
                break;



        }//should end the switch
    }

    else{ //this ELSE corresponds to the "if (ACCOM>R0 && boundnum<4)"
        /*%this ELSE basically says "if particle wasn't absorbed, then the
         %particle is diffusely reflected!!"*/
        switch (boundnum){//it should cover boundnums 1-3

                /*it is just a diffuse reflection by preserving "speed" and
simply changing direction randomly.*/



            case 1: /*%diffuse reflection from droplet 1*/
                /*%these below are all with respect to the global
coordinates*/
                xhit= xold + dtmin*prt[j].u;
                yhit= yold + dtmin*prt[j].v;
                zhit= zold + dtmin*prt[j].w;

                /*%these below are all with respect to the CENTER of the
droplet, not global anymore*/
                xhitc= xhit-cx1;
                yhitc= yhit-cy1;
                zhitc= zhit-cz1;



                /*  Insert randomly generated velocity components with local
CS here, preserving speed, but a completely new direction perpendicular to
the tangent plane*/
                c=sqrt(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w);
```

```
                phiv=2.*PI*((double) random())/RANMAX ;//v denotes to
determine velocity
                thetav= acos(((double) random())/RANMAX);//ensures w is
positive!!

                u=c*sin(thetav)*cos(phiv);// These are randomly generated
                v=c*sin(thetav)*sin(phiv);
                w=c*cos(thetav);

                randvel[0][0]=u;
                randvel[1][0]=v;
                randvel[2][0]=w;
                //%end random generation here.

                phipos= atan(yhitc/xhitc); /*%pos denotes that these are
angles for determining position*/
                thetapos = atan(-zhitc/sqrt(xhitc*xhitc + yhitc*yhitc));

                double Yxz1[3][3]={
                    {cos(thetapos), 0, sin(thetapos)},
                    {0, 1, 0},
                    { -sin(thetapos), 0, cos(thetapos)}
                } ;/*%rotation #1, about y, and from x to z*/

                double Xyz2[3][3]={
                    {1, 0, 0},
                    {0, cos(phipos), sin(phipos)},
                    {0, -sin(phipos), cos(phipos)}
                }; /*%rotation #2, about x, and from y to z*/

                double Yxz3[3][3]={
                    {cos(90.*PI/180.), 0, sin(90.*PI/180.)},
                    {0, 1, 0},
                    {-sin(90.*PI/180.), 0, cos(90.*PI/180.)}
                };/*%rotation #3, about y, and from x to z*/



                /*First 2 Rotations*/
                mat_mult3x3(Yxz3,Xyz2,rot1,3,3,3);
                /*third Rotation*/
                mat_mult3x3(rot1,Yxz1,totrot,3,3,3);

                prt[j].u=mat_mult_vel(totrot,randvel,0,3);
                prt[j].v=mat_mult_vel(totrot,randvel,1,3);
                prt[j].w=mat_mult_vel(totrot,randvel,2,3);




                prt[j].x = xhit;
                prt[j].y = yhit;
                prt[j].z = zhit;
                dtres=dtres-dtmin;
```

```
                return;

            case 2: //% diffuse reflection from droplet 2
                xhit= xold + dtmin*prt[j].u;//%these are all with the global
coordinates
                yhit= yold + dtmin*prt[j].v;
                zhit= zold + dtmin*prt[j].w;

                xhitc= xhit-cx2;/*%these are all with respect to the center
of the droplet, not global anymore*/
                yhitc= yhit-cy2;
                zhitc= zhit-cz2;



                /*%  Insert randomly generated velocity components with local
CS here.*/
                c=sqrt(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w);

                phiv=2.*PI*((double) random())/RANMAX ;//%v denotes to
determine velocity
                thetav= acos(((double) random())/RANMAX);

                u=c*sin(thetav)*cos(phiv);//%These are randomly generated
                v=c*sin(thetav)*sin(phiv);
                w=c*cos(thetav);
                randvel[0][0]=u;
                randvel[1][0]=v;
                randvel[2][0]=w;
                //%end random generation here.

                phipos= atan(xhitc/-yhitc);/*pos denotes that these are
angles for determining position*/
                thetapos = atan(-zhitc/sqrt(xhitc*xhitc + yhitc*yhitc));


                double Xzy1[3][3]={
                    {1, 0, 0},
                    { 0, cos(thetapos), -sin(thetapos)},
                    {0, sin(thetapos), cos(thetapos)}
                }; /*rotation #, about x, and from z to y*/
                double Yxz2[3][3]={
                    {cos(phipos), 0, sin(phipos)},
                    { 0, 1, 0},
                    { -sin(phipos), 0, cos(phipos)}
                };/*rotation #, about y, and from x to z*/
                double Xzy3[3][3]={
                    {1, 0, 0},
                    {0, cos(90.*PI/180.), -sin(90.*PI/180.)},
                    {0, sin(90.*PI/180.), cos(90.*PI/180.)}
                }; /*rotation #, about x, and from z to y*/
```

```
                /*First Rotation*/
                mat_mult3x3(Xzy3,Yxz2,rot1,3,3,3);
                /*Second Rotation*/
                mat_mult3x3(rot1,Xzy1,totrot,3,3,3);

                prt[j].u=mat_mult_vel(totrot,randvel,0,3);
                prt[j].v=mat_mult_vel(totrot,randvel,1,3);
                prt[j].w=mat_mult_vel(totrot,randvel,2,3);



                /*%These are the new randomly diffused velocities in the
global coordinate system...(1x3)*(3x1)=1x1!!!!*/

                prt[j].x = xhit;
                prt[j].y = yhit;
                prt[j].z = zhit;
                dtres=dtres-dtmin;
                return;

            case 3: //%diffuse reflection from droplet 3
                xhit= xold + dtmin*prt[j].u; //%these are all with the global
coordinates
                yhit= yold + dtmin*prt[j].v;
                zhit= zold + dtmin*prt[j].w;

                xhitc= xhit-cx3;/*%these are all with respect to the center
of the droplet, not global anymore*/
                yhitc= yhit-cy3;
                zhitc= zhit-cz3;



                 /*%  Insert randomly generated velocity components with
local CS here.*/
                c=sqrt(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w);

                phiv=2.*PI*((double) random())/RANMAX;//v denotes to
determine velocity
                thetav= acos(((double) random())/RANMAX);

                u=c*sin(thetav)*cos(phiv);//These are randomly generated
                v=c*sin(thetav)*sin(phiv);
                w=c*cos(thetav);
                randvel[0][0]=u;
                randvel[1][0]=v;
                randvel[2][0]=w;
                //%end random generation here.

                phipos= atan(yhitc/-xhitc);/*%pos denotes that these are
angles for determining position*/
                thetapos = atan(-zhitc/sqrt(xhitc*xhitc + yhitc*yhitc));

                double Yzx1[3][3]={
```

```
                    {cos(thetapos), 0, -sin(thetapos)},
                    {0, 1, 0},
                    {sin(thetapos), 0, cos(thetapos)}
                };/*%rotation #, about y, and from z to x*/

                double Xyz22[3][3]={
                    {1, 0, 0},
                    {0, cos(phipos), sin(phipos)},
                    {0, -sin(phipos), cos(phipos)}
                }; /*%rotation #, about x, and from y to z*/

                double Yzx3[3][3]={
                    {cos(90.*PI/180.), 0, -sin(90.*PI/180.)},
                    {0, 1, 0},
                    {sin(90.*PI/180.), 0, cos(90.*PI/180.)}
                };/*%rotation #, about y, and from z to x*/



                /*First Rotation*/
                mat_mult3x3(Yzx3,Xyz22,rot1,3,3,3);
                /*Second Rotation*/
                mat_mult3x3(rot1,Yzx1,totrot,3,3,3);



                prt[j].u=mat_mult_vel(totrot,randvel,0,3);
                prt[j].v=mat_mult_vel(totrot,randvel,1,3);
                prt[j].w=mat_mult_vel(totrot,randvel,2,3);



                /*%These are the new randomly diffused velocities in the
global coordinate system...(1x3)*(3x1)=1x1!!!!*/


                prt[j].x = xhit;//%
                prt[j].y = yhit;//%
                prt[j].z = zhit;//%
                dtres=dtres-dtmin;
                return;

            }
        }



        switch (boundnum) {//%here only boundnum 4-8 are used
            case 4:
                //%condensing wall hit
                diffusewall_reflection(j);
                condwallhit_DT++;
                condwallhit_delt++;
                //%diffuse reflection..
                break;
```

```
        case 5:
            wallhit(j);
            wallhit1_DT++;
            wallhit1_delt++;
            //%specular wall 1 hit
            //%grab from wallhit
            break;

        case 6:
            wallhit(j);
            wallhit2_DT++;
            wallhit2_delt++;
            //%specualr wall 2 hit
            //%grab from wallhit
            break;

        case 7:
            wallhit(j);
            wallhit3_DT++;
            wallhit3_delt++;
            //%%specular wall 3 hit (y=0)
            break;

        case 8:
            spacescape= spacescape + 1;
            spacescape_DT++;
            loseprt(j);
            /*NOTE THAT dtres IS SET TO ZERO, BUT SO IS losscheck, which
means that having dtres equal to zero makes the "advance"-loop finish, but
with losscheck set to zero, it doesn't really start the loop again for
another particle, but rather for a new particle...PERFECT!!!*/
            //%space escape
            //%relatively simple.  just need to keep track of the number of
            //%particles and do a "loseprt()"

            //%% space escaped particles..when should this be zeroed out
(which loop)??


            /*
             //%does setting the LOSSCHECK to 0 still make sense for my
             //%simulation??? Think about it!!!!!!
             */
            break;

    }
    return;


}


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
```

```
void diffusewall_reflection(j)
{
    /*global dtres dtmin prt xold yold zold */
    double xhit, yhit, zhit, c, theta, phi,R,acosR;
    xhit= xold + dtmin*prt[j].u; //%these are all with the global coordinates
    yhit= yold + dtmin*prt[j].v;
    zhit= zold + dtmin*prt[j].w;



    //%% Random Sampling of theta
    c=sqrt(prt[j].u*prt[j].u + prt[j].v*prt[j].v + prt[j].w*prt[j].w);
    R=((double) random())/RANMAX;
    acosR=acos(R);
    theta=90.*(PI/180.)+acos(R); //%radians
    theta11=theta;
    ccc=c;
    RR=R;
    acosRR=acosR;

    //%% Random Sampling of phi

    phi=360.*(PI/180.)*((double) random())/RANMAX; //%radians

    //%% Setting up velocities based on random angles

    prt[j].u=c*sin(theta)*cos(phi);// in the x-direction
    prt[j].v=c*sin(theta)*sin(phi); //% in the y-direction
    prt[j].w=c*cos(theta);// %in the z-direction

    prt[j].x = xhit;//%
    prt[j].y = yhit;//%
    prt[j].z = zhit;//%
    dtres=dtres-dtmin;

}
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/






void wallhit(j)
{
    double norm_dist, i_vec, j_vec, norm_mag,c, xhit, yhit, zhit;
    double ref_vel[3];

/*
     BBBBBBB  Function to deal with particles hitting the wall surface
     %make sure that all input contact angles are in radians and no cosd/sind
```

```
     %are used!!!

     % % %the following is just used to define the lines that make the
triangle
     % % d1p= [-(s+d*sin(ca))/2, 0]; %droplet 1 center position [x,y]
     % % d2p= [0, sqrt(3)*(s+d*sin(ca))/2] ; %droplet 2 center position [x,y]
     % % d3p= [(s+d*sin(ca))/2,0] ; %droplet 3 center position [x,y]
     % %
     % % m1=(d2p(2)-d1p(2))/(d2p(1)-d1p(1)); %positive slope
     % % m2=(d2p(2)-d3p(2))/(d2p(1)-d3p(1)); %negative slope
     % %
     % %
     % % b=(s+d*sin(ca))*sqrt(3)/2;
     */

    /*%(unit vector 2) points to -i, -j, and (unit vector 1) points to +i,-j
     %% creating unit vector of the surface normal*/
    norm_dist=L*0.5*sin(60.*PI/180.);/*%60 and 30 degrees have to do with it
being in a triangle*/
    i_vec=norm_dist*cos(30.*PI/180.);
    j_vec=norm_dist*sin(30.*PI/180.);
    norm_mag= sqrt(i_vec*i_vec + j_vec*j_vec);/*%normal vector
magnitude...this should equal norm_dist*/
    double uv_n1[3]={i_vec/norm_mag, -j_vec/norm_mag,0} ;/*%normal unit
vector of line connecting drop 1 to 2*/
    double uv_n2[3]={ -i_vec/norm_mag, -j_vec/norm_mag,0} ;/*%normal unit
vector of line connecting drop 3 to 2*/


    double vel[]={prt[j].u,prt[j].v,prt[j].w};
    c=sqrt(prt[j].u*prt[j].u + prt[j].v*prt[j].v + prt[j].w*prt[j].w);
    double uv_v[]={vel[0]/c,vel[1]/c,vel[2]/c};/*%1x3 vector...unit vector of
incoming velocity,where the velocity in z-direction(w) shouldn't ever be
altered*/

    xhit= xold + dtmin*prt[j].u; /*%these are all with the global
coordinates*/
    yhit= yold + dtmin*prt[j].v;
    zhit= zold + dtmin*prt[j].w;

    //%specular reflected vector
    switch (boundnum){
        case 5: //MATLAB CODE %x<=0 && y>=m1*x+b
            /*uv_n=uv_n1;//note that this might not be valid..check!!
             ref_vel=-1*(2*(sum(uv_v.*uv_n))*uv_n-uv_v )*c; %in the for
(reflected unit vector)*c, where c is magnitude of velocity*/

            ref_vel[0]=-
1.*(2.*(uv_v[0]*uv_n1[0]+uv_v[1]*uv_n1[1]+uv_v[2]*uv_n1[2])*uv_n1[0]-
uv_v[0])*c;
            ref_vel[1]=-
1.*(2.*(uv_v[0]*uv_n1[0]+uv_v[1]*uv_n1[1]+uv_v[2]*uv_n1[2])*uv_n1[1]-
uv_v[1])*c;
            ref_vel[2]=-
1.*(2.*(uv_v[0]*uv_n1[0]+uv_v[1]*uv_n1[1]+uv_v[2]*uv_n1[2])*uv_n1[2]-
uv_v[2])*c;
```

```
            prt[j].u=ref_vel[0];
            prt[j].v=ref_vel[1];
            prt[j].w=ref_vel[2];



            break;

        case 6: //%x>=0 && y>=m2*x+b
            //MATLAB CODE
            /*uv_n=uv_n2;
             ref_vel=-1*(2*(sum(uv_v.*uv_n))*uv_n-uv_v )*c; //%in the for
(reflected unit vector)*c, where c is magnitude of velocity
             prt[j].u=ref_vel[1];
             prt[j].v=ref_vel[2];*/

            ref_vel[0]=-
1.*(2.*(uv_v[0]*uv_n2[0]+uv_v[1]*uv_n2[1]+uv_v[2]*uv_n2[2])*uv_n2[0]-
uv_v[0])*c;
            ref_vel[1]=-
1.*(2.*(uv_v[0]*uv_n2[0]+uv_v[1]*uv_n2[1]+uv_v[2]*uv_n2[2])*uv_n2[1]-
uv_v[1])*c;
            ref_vel[2]=-
1.*(2.*(uv_v[0]*uv_n2[0]+uv_v[1]*uv_n2[1]+uv_v[2]*uv_n2[2])*uv_n2[2]-
uv_v[2])*c;


            prt[j].u=ref_vel[0];
            prt[j].v=ref_vel[1];
            prt[j].w=ref_vel[2];
            break;


        case 7: //%y<=0
            prt[j].v=-prt[j].v;
            //%prt[j].y=-prt[j].y;
            break;
    }


    dtres=dtres-dtmin;
    prt[j].x = xhit;
    prt[j].y = yhit;
    prt[j].z = zhit;


}

void ambemit(k)
{
    // /* BBBBBBBBBBB  Function to handle influx from ambient */


    ////////////////These will be local variables////////////////////////////
```

```
    int i, j, n, nadd, xc, yc, zc;
    double mean_c, influx, den, dnadd;
    double r, phi, cT, cp1, cp2, cn;
    double a1, a2, snynz, b0, b1, b2;
    double costheta, sintheta, snx, sny, snz;
    double Rtest, eps, Ie;
    double R1;
    ////////////////These will be local variables//////////////////////////


    //            /* emission of species 1 */
    //                        /* determine number */
    //                        /* flux per time step */

    den = CONCA*PRESS/(KB*TAMB); //molecular density of species 1
    mean_c = sqrt( 8.*KB*TAMB/(PI*MASSA));//MEANspeed of a particle
    influx = 0.25*den*mean_c*DT*(At)/PARTRAT;//kinetic theory flux
eqn...TOTAL PARTICLES based on MOLECULAR flux (NOT per unit area)
    influxprint=influx*PARTRAT/At/DT;
    influx += afluxres1;
    nadd = (int) influx;//(int) rounds down
    dnadd = (double) nadd;
    afluxres1 = influx - dnadd;  /* save residual for next time */
    nadd++;
    cT = sqrt(KB*TAMB/MASSA);//speed from which to sample

     i = 1;



    while (i < nadd){

        //// begin random space entry point
       //-.5*L_cell beneath the TOP triangle SURFACE

        j = nprt; //nprt should start out as initial number of particles
        //basically we are adding more particles to our domain...which is
what j
        //indicates as the indice-particle-number


        ///* randomly select entry point for new particle */
        ambprtstart(j); //this should give prt.x,.y,.z
        //// end random space entry point




        //// begin random velocity generation


        phi = 2.*PI*( ((double) random())/RANMAX );
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
```

```
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].u = cT*r*cos(phi);
        prt[j].v = cT*r*sin(phi);

        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].w = cT*r;


        //// Modified to adjust for translated coordiante system...the
coordinate system that starts the cell counting is in the [cx1,cy1,cz1]
corner


        prt[j].kind = 1;
        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); // local cell# in the x dir
from the CORNER...this will leave A LOT of empty cells(should account for
them when considering total cells)!!!
        yc = (int)(prt[j].y*NCY/ymax); // local cell# in the y dir from the
CORNER
        zc = (int)(prt[j].z*NCZ/zmax); // local cell# in the z dir from the
CORNER
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;
        prt[j].mass = PARTRAT*MASSA; //total mass of particle by summing mass
of molecules

        //// end modification



        //              /* rotational energy for water */
        Rtest = 0.97*((double) random())/RANMAX;//
        eps = 0.002;
        Ie = 0.0;
        while (Ie < Rtest){
            eps = eps + 0.1;
            Ie = Ie + (sqrt(eps - 0.1)*exp(-eps + 0.1)
                        + sqrt(eps)*exp(-eps))*0.05642;
        } //end while1
        prt[j].erot = PARTRAT*KB*TAMB*eps;
        i++;
        nprt++;

    } //end while species 1 emission


    //// same from here down as above, just for a different species!!!
```

```
    //            /* emission of species 2 */
    //                        /* determine number */
    //                        /* flux per time step */
    den = (1. - CONCA)*PRESS/(KB*TAMB);
    mean_c = sqrt( 8.*KB*TAMB/(PI*MASSB));//MEANspeed of particle
    influx = 0.25*den*mean_c*DT*(At)/PARTRAT;
    influx += afluxres2;



    nadd = (int) influx;
    dnadd = (double) nadd;
    afluxres2 = influx - dnadd;  /* save residual for next time */
    nadd++;



    cT = sqrt(KB*TAMB/MASSB); //molecule speed?//m/s...WHY DOES THIS NOT HAVE
THE 8 NOR PI
    ////////////////////////////////////////////////
    ////


    i = 1;
    while (i < nadd){
        j = nprt;
        //                          /* randomly select entry
        //                          point for new particle */

        ambprtstart(j);


        //// begin random velocity generation (mine should be easier since
global
        //coordinate system is to be used for direction generation)


        phi = 2.*PI*( ((double) random())/RANMAX );
        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        prt[j].u = cT*r*cos(phi);
        prt[j].v = cT*r*sin(phi);

        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
```

```
        prt[j].w = cT*r;


        //// end random velocity generation




        prt[j].kind = 2;
        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); // local cell# in the x dir
from the origin?...this will leave A LOT of empty cells(should account for
them when considering total cells)!!!
        yc = (int)(prt[j].y*NCY/ymax); // local cell# in the y dir from the
origin?
        zc = (int)(prt[j].z*NCZ/zmax); // local cell# in the z dir from the
origin?
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;
        prt[j].mass = PARTRAT*MASSB; //total mass of particle by summing mass
of molecules
        // end modification



        /* rotational energy for N2 */
        R1=((double) random())/RANMAX;
        if (R1<=0 || R1>=1)  {
            while (R1<=0 || R1>=1){
                R1=((double) random())/RANMAX;
            }
        }
        prt[j].erot = -PARTRAT*KB*TAMB/log(R1);
        /* END rotational energy for N2 */
        i++;
        nprt++;
    }


}//function end


////////////////////////////////////////////////////////////////////////////
////////



void ambprtstart(j)
{
    int flag,i;
    double xrangemin, xrangemax, yrangemin, yrangemax, xcoor, xrange, ycoor;
    double xpoint, ypoint,zpoint,d2d1, d2d2, d2d3;
```

```
    xrangemin=cx1 + gapx;//-L/2.;/*minimum coordinate at the base of the
triangle*/
    xrangemax=cx3 - gapx;//L/2.;/*maximum coordinate at the base of the
trianlge*/
    yrangemin=0. + gapy;/*minimum coordinate anywhere*/
    yrangemax=cy2 - gapy;/*maximum coordinate in the middle of the triangle*/
    flag=1;

    while (flag>0){
        /* X-range as a funcion of y-position for triangle*/
        /*XCOOR=2. just to initialize the WHILE loop BELOW*/
        xcoor=2.;/* xcoor is the actual possible coordinate, except here
where it is just used for initializing */

        xrange=1.; /*Note that xrange is an absolute value of half the
available distance*/
        i=0;
        /*end initialization*/

        while (xcoor < -xrange || xcoor > xrange) {

            i=i+1;
            xcoor=xrangemin+ ((double) random())/RANMAX*(xrangemax-
xrangemin);
            ycoor=yrangemin+ ((double) random())/RANMAX*(yrangemax-
yrangemin);

            xrange=(yrangemax-ycoor)*(L/2./H); /*%y=mx + b, m= -H/(L/2),
x=(y-b)/m = -(b-y)/m, b=yrangemax, (and x is really xrangemax).
                                               %note that xrange is the
distance from the middle of the triangle (cut halfways down the middle)  to a
corresponding edge for the height!!  It's NOT the whole
                                               %width at that height! NOTE
THAT XRANGE IS JUST AN ABSOLUTE VALUE...also note that m has a negative in
front of the expression, therefore (L/2./H) is correct (no negative
needed)!!*/

            if (xcoor>= -xrange && xcoor<= xrange){
                /*If the particle is within acceptable range, designate them
as the coordinate*/
                xpoint=xcoor;
                ypoint=ycoor;
                zpoint=0.5*gapz;

                /*%note that it comes into this if it automatically does not
go
                 %into the next step of the while loop*/
            }


        }



        /*Once coordinates are within range for the cooresponding y, It is
```

```
time to determine if a droplet was stricken*/

        /*Z= D/2;//Distance from coordinate system to the center of sphere
outlined by
         the droplet.  NOT the distance to the condensing surface...because
they are not the same when contact angle is not 90degrees!*/
        double ppos[3]={xpoint,ypoint,zpoint};/*particle position*/



        d2d1= pointdist(ppos,d1p); /*sqrt(  sum((ppos-d1p).^2)  ); //distance
to droplet center 1  */
        d2d2= pointdist(ppos,d2p); /*sqrt(  sum((ppos-d2p).^2)  ); //distance
to droplet center 2  */
        d2d3= pointdist(ppos,d3p); /*sqrt(  sum((ppos-d3p).^2)  ); //distance
to droplet center 3 */



        if (d2d1 <= RSP || d2d2 <= RSP || d2d3 <= RSP){

            flag=1;
            /*the particle lies within a droplet, generate a new position*/
        }
        else{
            /*nothing has been stricken...accept particle and just continue*/
            prt[j].x = xpoint;
            prt[j].y = ypoint;
            prt[j].z = zpoint;
            flag=-1;
        }
    }


}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



void dropemit(int k)
{
    /* BBBBBBBBBBB  Function to handle influx from the sphere */
    /* %k is the timestep


    /*%%%%%%%%These will be local variables%%%%%%%%%%%%%%*/
    int j, n,  xc, yc, zc, iec;
     double Epost, cT, r, phi, cp1, cp2, cn;
     double costheta, sintheta, snx, sny, snz;
     double  a1, a2, snynz, b0, b1, b2, dts;
     double Rtest, eps, Ie;
     double dEdT, ae, be, u283, tcorr, error, ersinc, Eg0;
     double eta, rg, siglv, dedT, leta;
```

```
    double Mw, tml, sig, kl, vl, ulv, PsatTi, Pvi, jdout, SF, R,Tsp2;
    double Err, Err_norm, fErr, theta, R1;
    double delTemp, fun1, fun2, dfunddelTemp, Eg01, jdout1,
elostconduc_print;
    int counter;
    /*%%%%%%%%End local variables%%%%%%%%%%%%%%%*/

    Mw = 18.0;  //molecular mass of water kg/kmol

    ///* determine energy
    //transfer for this time step */
    Eg = Eg + egain1 + egain2 + egain3;
    //Eg is the sum up to this point. egain simply for this time step, it is
zeroed out at each time-step loop.
    nconsum = nconsum + nncon1 + nncon2 + nncon3;//note that all the
noncondensible particles up to this point are added here.  nncon is zero-ed
out at each k step

    R= KB/MASSA;/*specific gas constant*/
    SF = 0.5*DropSF(ca, RSP);//ca is input in radians!!!  DropSF is for a
full hemisphere.  1/2 because of a quarter hemisphere!!

    //None of the above change with temperature


    if (k >= INTST) //if1 //is this right?
    {
        // surface tension correction for small radii
        sig = surface_tension(Tsp); //N/m
        tml=0.157e-9; //tolman length [m]
        sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
        //end surface tension correction

        //other properties
        kl = thermal_cond_liquid(Tsp); //W/mK
        vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
        ulv=enthalpy(Tsp)*MASSA; // J/molecule-K
        PsatTi = pressure(Tsp); //
        Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//

        jdout= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j
droplet out, does this work for a droplet of mixed substance?  We use Mw, but
this doesn't account that there are mixed non-condensible water particles in
there.  YES->Remember that the noncondensible particles don't exactly
penetrate the droplet, so using Mw here is perfectly fine.  They are simply
reflected later at an accomodated temperature (if accommodated)


        Eg0= jdout*( PI*RSP*RSP*(1.-cos(ca)) )*delt*(ulv + 0.5*KB*Tsp) +
(DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT + SF*kl*(Tsp - Tw)*delt; //total
energy out based on kinetic theory ...note how each parameter is based on TSP
here!!!!...total up to this time, not just for this time step!!!

        error = sqrt((Eg - Eg0)*(Eg - Eg0))/(Eg + EPS);
```

```
        Err = (Eg - Eg0);  //compare to using error from above!!!
        Err_norm=(TAMB-Tw)*(kl*SF)*delt;
        fErr=Err/Err_norm; //current method (mendoza)...//compare to using
error from above!!!


        //basically, the temperature is iterated to the point where the
        //theoretical ouflux is equal to the 'counted (from the simulation)'
        //influx. (remember that the outflux is highly nonlinear, that's why
TSP has to
        //be iterated)



        delTemp=.1;//just to initialize the loop
        counter=0;//just to initialize the loop

        while (fabs(error)>0.0002) //or simply use the error, or fErr...KNOW
what order of magnitude should be in the while loop!!

            //while (fabs(fErr)>0.0002) //or simply use the error, or
fErr...KNOW what order of magnitude should be in the while loop!!
            //FIRST PART TESTS, SECOND PART ADDS CORRECTION
        {
            counter=counter+1;

            if (k>1000000){
                printf("\n here2");
                printf("\n k=%d", k);
                printf("\n here2");
            }

            Tsp= TAMB+delTemp;

            // surface tension correction for small radii
            sig = surface_tension(Tsp); //N/m
            tml=0.157e-9; //tolman length [m]
            sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
            //end surface tension correction

            //other properties
            kl = thermal_cond_liquid(Tsp); //W/mK
            vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
            ulv=enthalpy(Tsp)*MASSA; // J/molecule-K
            PsatTi = pressure(Tsp); //
            Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//

            jdout= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j
droplet out

            Eg0= jdout*( PI*RSP*RSP*(1.-cos(ca)) )*delt*(ulv + 0.5*KB*Tsp) +
(DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT + SF*kl*(Tsp - Tw)*delt; //total
energy out based on kinetic theory...note how each parameter is based on TSP
here!!!!...total up to this time, not just for this time step!!!
            Eg01=Eg0;
```

```
            jdout1=jdout;


            fun1= Eg-Eg0;

            //********************IMPORTANT****//

            //Note that it is fun1 that is used for the error check of the
while loop.  This is because I only want to test the "adjusted" temperature.
fun2 is calculated to make the temperature adjustment, but it shouldn't be
used as a valid test for the error of the while loop since it is only used to
calculate the derivative in the Newton-Raphson correction (d(fun)/dT)



            //******************END IMPORTANT***//



            error = sqrt(fun1*fun1)/(Eg + EPS);

            Err = (fun1);  //compare to using error from above!!!
            Err_norm=(TAMB-Tw)*(kl*SF)*delt;
            fErr=Err/Err_norm; //compare to using error from above!!!



            /*need a temperature increment for the newton raphson
correction*/

            Tsp= TAMB + 1.005*delTemp;


            // surface tension correction for small radii
            sig = surface_tension(Tsp); //N/m
            tml=0.157e-9; //tolman length [m]
            sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
            //end surface tension correction


            //other properties
            kl = thermal_cond_liquid(Tsp); //W/mK
            vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
            ulv=enthalpy(Tsp)*MASSA; // J/molecule-K
            PsatTi = pressure(Tsp); //
            Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//

            jdout= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j
droplet out

            Eg0= jdout*( PI*RSP*RSP*(1.-cos(ca)) )*delt*(ulv + 0.5*KB*Tsp) +
(DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT + SF*kl*(Tsp - Tw)*delt; //total
energy out based on kinetic theory...note how each parameter is based on TSP
here!!!!...total up to this time, not just for this time step!!!
```

```
            fun2=Eg - Eg0;
            //dfunddelTemp=(fun2-fun1)/delTemp;  These temperature
corrections have been removed to the top to accept the most recently tested
temperature!!!!!
            Tsp= TAMB + delTemp;//Now that I already calculated all my
properties at the modified Tsp, I want to make sure that the loop closes with
the Tsp used in fun1, because that is the one that is tested in the while
loop, and that is the correct one I want to keep when the while statment
holds true.  Therefore, that's why I retype the SAME equation as I did above
(at the very top of the while loop) to end the loop with the Tsp from the
fun1!!.  This should work, because delTemp shouldn't be any different and
should have not been modified at this point since the beginnning of the while
loop.  It isn't, I just did a "find delTemp"...5/22
            Tsp2=Tsp;


            dfunddelTemp=(fun2-fun1)/(.005*delTemp);//don't want this to be
zero because it is in the denominator below...don't worry about it, it should
never be.
            delTemp=delTemp- fun1/dfunddelTemp;




            if (counter>30){
                printf("\n \n Eg= %8.8e",Eg);
                printf("\n Eg01= %8.8e",Eg01);
                printf("\n k= %d", k);
                printf("\n Err= %8.18e",Err);
                printf("\n error= %8.18e",error);
                /*printf("\n Err_norm= %8.18e",Err_norm);
                 printf("\n fErr= %8.18e",fErr);*/

                printf( "\n Tsp did  \n not converge \n after 30 iterations
\n Tsp=%8.3e",Tsp );
                exit(EXIT_FAILURE);
            }


        }//ends the while looop
        //printf("\n counter= %d", counter);

        // surface tension correction for small radii
        sig = surface_tension(Tsp); //N/m
        tml=0.157e-9; //tolman length [m]
        sig=sig*(1./(1.+ (2.*tml/RSP))); //the main correction
        //end surface tension correction
        ulv=enthalpy(Tsp)*MASSA; // J/molecule-K

        //the above is redone so that sig and ulv can be calculated at the
right temperature, not the modifed one for a newton raphson calculation.
```

```
        errsum=errsum+fErr;//should have already been zeroed out in the main
file
        EvapT=ulv-2.*sig*MASSA/(MDA*RSP);



    }//ends the if statement

    vl=1./density_liquid(Tsp);//...vl = 0.001; //cu meters per kg
    PsatTi = pressure(Tsp); //
    Pvi = PsatTi*exp(2.*vl*sig/(RSP*R*Tsp));//
    j0print= ACCOM*(Pvi/(KB*Tsp))*(sqrt(8.*NA*KB*Tsp/(PI*Mw)))/4.;//j droplet
out
    Eg0h20print= j0print*( PI*RSP*RSP*(1.-cos(ca)) )*delt*(ulv + 0.5*KB*Tsp)
+ (DEGFREB*0.5 + 0.5)*KB*Tsp*nconsum*PARTRAT; //total energy out based on
kinetic theory (does it make sense to usethis?)...note how each parameter is
based on TSP here!!!!...total up to this time, not just for this time step!!

    elostconduc=SF*kl*(Tsp - Tw)*DT;
    elostconduc_print=SF*kl*(Tsp - Tw)*delt;



    kl = thermal_cond_liquid(Tsp); //W/mK









    /*note that nncon is only for this time step, it is zeroed out
     at the beginning of each time step loop*/

    /*NOTE that we assume uniform droplet temperature!!!*/
    while (nncon1 > 0){ //while nncon1
        j = nprt;
        prt[j].kind = 2;
        prt[j].mass = PARTRAT*MASSB;

        /* rotational energy for N2 */
        R1=((double) random())/RANMAX;
        if (R1<=0 || R1>=1)  {
            while (R1<=0 || R1>=1){
                R1=((double) random())/RANMAX;
            }
        }
        prt[j].erot = -PARTRAT*KB*Tsp/log(R1);
        /* END rotational energy for N2 */
```

```
        dropnumber=1;
        dropemitpos(j,k);
        dropemitvel(j);
        //dropemitcheck(j);


        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); // local cell#
        yc = (int)(prt[j].y*NCY/ymax); // local cell# in the y dir from the
corner
        zc = (int)(prt[j].z*NCZ/zmax); // local cell# in the z dir from the
corner
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


        /* remove energy of particle from sphere tally */
        Epost = 0.5*prt[j].mass*(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w) + prt[j].erot;
        Einlower1 = Einlower1 - Epost;
        sphemits1 = sphemits1 + 1;/*this is done to cancel the effect of...
                                    //spherehits that are kind2...creates a
zero net flux of kind 2 when...
                                    //doing (spherehits-sphemits)see chkspace
and you can see that...
                                    //spherehits does not discriminate whether
it is kind 1 or 2, ....
                                    //discrimination is until a few lines
afterwards.*/
        sphemits1_DT++;
        elost1 = elost1 + Epost;
        nprt = nprt + 1;
        nncon1 = nncon1 -1;//nncon refers to number of noncondensible
particles absorbing into a droplet.  Why not just reflect non-condensible
particles immediately?  why wait to emit them?  because of the energy balance
    } /* end while nncon1 */

    while (nncon2 > 0){
        j = nprt;
        prt[j].kind = 2;
        prt[j].mass = PARTRAT*MASSB;

        /* rotational energy for N2 */
        R1=((double) random())/RANMAX;
        if (R1<=0 || R1>=1)  {
            while (R1<=0 || R1>=1){
                R1=((double) random())/RANMAX;
            }
        }
        prt[j].erot = -PARTRAT*KB*Tsp/log(R1);
        /* END rotational energy for N2 */

        dropnumber=2;
        dropemitpos(j,k);
        dropemitvel(j);
        //dropemitcheck(j);
```

```
        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); // local cell#
        yc = (int)(prt[j].y*NCY/ymax); // local cell# in the y dir from the
corner
        zc = (int)(prt[j].z*NCZ/zmax); // local cell# in the z dir from the
corner
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;


        /* remove energy of particle from sphere tally */
        Epost = 0.5*prt[j].mass*(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w) + prt[j].erot;
        Einlower2 = Einlower2 - Epost;
        sphemits2 = sphemits2 + 1;//this is done to cancel the effect of...
        //spherehits that are kind2...creates a zero net flux of kind 2
when...
        //doing (spherehits-sphemits)see chkspace and you can see that...
        //spherehits does not discriminate whether it is kind 1 or 2, ....
        //discrimination is until a few lines afterwards.
        sphemits2_DT++;

        elost2 = elost2 + Epost;
        nprt = nprt + 1;
        nncon2 = nncon2 -1;//nncon refers to number of noncondensible
particles absorbing into a droplet.  Why not just reflect non-condensible
particles immediately?  why wait to emit them? because of the energy balance
    } /* end while nncon2 */

    while (nncon3 > 0) {

        j = nprt;
        prt[j].kind = 2;
        prt[j].mass = PARTRAT*MASSB;

        /* rotational energy for N2 */
        R1=((double) random())/RANMAX;
        if (R1<=0 || R1>=1)  {
            while (R1<=0 || R1>=1){
                R1=((double) random())/RANMAX;
            }
        }
        prt[j].erot = -PARTRAT*KB*Tsp/log(R1);
        /* END rotational energy for N2 */

        dropnumber=3;
        dropemitpos(j,k);
        dropemitvel(j);
        //dropemitcheck(j);


        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); //
        yc = (int)(prt[j].y*NCY/ymax); //
        zc = (int)(prt[j].z*NCZ/zmax); //
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;
```

```
        /* remove energy of particle from sphere tally */
        Epost = 0.5*prt[j].mass*(prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w) + prt[j].erot;
        Einlower3 = Einlower3 - Epost;
        sphemits3 = sphemits3 + 1;//this is done to cancel the effect of...
        //spherehits that are kind2...creates a zero net flux of kind 2
when...
        //doing (spherehits-sphemits)see chkspace and you can see that...
        //spherehits does not discriminate whether it is kind 1 or 2, ....
        //discrimination is until a few lines afterwards.
        sphemits3_DT++;
        elost3 = elost3 + Epost;
        nprt = nprt + 1;
        nncon3 = nncon3 -1;//nncon refers to number of noncondensible
particles absorbing into a droplet.  Why not just reflect non-condensible
particles immediately?  why wait to emit them?  because of the energy balance
    } /* end while nncon3 */



    /*%the residual "wfluxres" from last time is subtracted here*/

    elostNcon=elost1 + elost2 + elost3;
    //printf("\n elostncon=%8.8e",elostNcon);

    egain_beta=egain1 + egain2+ egain3;
    //printf("\n egain_beta=%8.8e",egain_beta);

    egain= egain1 + egain2 + egain3 - wfluxres;
    // printf("\n egain=%8.8e \n",egain);


    wfluxres2=wfluxres;

    elost=elostNcon + SF*kl*(Tsp - Tw)*DT;
    //SF should already account for 3 droplet segments of 60degrees


    elost1=0;
    elost2=0;
    elost3=0;
    //why is there egain used here instead of Eg??? this is based on a single
    //timestep, not on the entire simulation as the loop for the
determination
    //of temperature does.
    //okay, so to solve for the temperature, you need to consider the entire
    //simulation... to solve for the point at which to stop emitting, you
only
    //need the "dt" part of the simulation...if done right, it should give
you the same thing, or close to the same thing

    while (elost < egain){ //while3  BOUNDARY CONDITION!!! Basically enough
particles are emitted to make this boundary condition true!! Eout=Ein
        j = nprt;
        prt[j].kind = 1;
```

```
        prt[j].mass = PARTRAT*MASSA;
        /* rotational energy for water */
        Rtest = 0.97*((double) random())/RANMAX;
        eps = 0.002;
        Ie = 0.0;

        while (Ie < Rtest){
            eps = eps + 0.1;
            Ie = Ie + (sqrt(eps - 0.1)*exp(-eps + 0.1) + sqrt(eps)*exp(-
eps))*0.05642;
        }

        prt[j].erot = PARTRAT*KB*Tsp*eps;

        //1. generate random position and convert to GCS (global coord.
syst.)
        //2. generate random velocity and convert to GCS
        //3. determine rotational energy for corresponding type of particle
        //4. determine entry point for new particle (see below)



        dropnumber=1;
        dropemitpos(j,k);

        dropemitvel(j);

        /*dropemitcheck(j);*/

        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); // local cell# in the x dir
        yc = (int)(prt[j].y*NCY/ymax); // local cell# in the y dir
        zc = (int)(prt[j].z*NCZ/zmax); // local cell# in the z dir
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;



        /* remove energy of particle from sphere tally */
        Epost = 0.5*prt[j].mass*( prt[j].u*prt[j].u + prt[j].v*prt[j].v +
prt[j].w*prt[j].w ) + prt[j].erot + PARTRAT*(EvapT - 3.*KB*Tsp);

        Einlower1 = Einlower1 - Epost;//sphere tally, hence removal
        sphemits1 = sphemits1 + 1;
        sphemits1_DT++;
        elost1 = elost1 + Epost;
        nprt = nprt + 1;




        elostcon=elost1 + elost2 + elost3;
        elost=elostconduc + elostcon + elostNcon;/*should already include the
nncon and the conduction portion*/
        if (elost>egain) {
            //            /*basically reverse everything that was just
```

```
done...this eliminates the need for a wfluxres, HOWEVER it might
underestimate the amount emitted!!!!!!*/
            //            Einlower1 = Einlower1 + Epost;
            //            sphemits1 = sphemits1 -1;
            //            sphemits1_DT=sphemits1_DT-1;
            //            elost1 = elost1-Epost;
            //            nprt = nprt -1;
            //
            //
            //
            //
            //            elostcon=elost1 + elost2 + elost3;
            //            elost=elostconduc + elostcon + elostNcon;/*should
already include the nncon and the conduction portion*/
            break;
        }









        j = nprt;
        prt[j].kind = 1;
        prt[j].mass = PARTRAT*MASSA;
        /* rotational energy for water */
        Rtest = 0.97*((double) random())/RANMAX;
        eps = 0.002;
        Ie = 0.0;

        while (Ie < Rtest){
            eps = eps + 0.1;
            Ie = Ie + (sqrt(eps - 0.1)*exp(-eps + 0.1) + sqrt(eps)*exp(-
eps))*0.05642;
        }

        prt[j].erot = PARTRAT*KB*Tsp*eps;

        //1. generate random position and convert to GCS (global coord.
syst.)
        //2. generate random velocity and convert to GCS
        //3. determine rotational energy for corresponding type of particle
        //4. determine entry point for new particle (see below)
        //5.

        dropnumber=2;

        dropemitpos(j,k);

        dropemitvel(j);

        /*dropemitcheck(j);
                  }*/
```

```
        xc = (int)( (prt[j].x + L/2) *NCX/xmax );
        yc = (int)(prt[j].y*NCY/ymax); //
        zc = (int)(prt[j].z*NCZ/zmax); //
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;



        /* remove energy of particle from sphere tally */
        Epost = 0.5*prt[j].mass*(
prt[j].u*prt[j].u+prt[j].v*prt[j].v+prt[j].w*prt[j].w ) + prt[j].erot +
PARTRAT*(EvapT - 3.*KB*Tsp);

        Einlower2 = Einlower2 - Epost;//sphere tally, hence removal
        sphemits2 = sphemits2 + 1;
        sphemits2_DT++;
        elost2 = elost2 + Epost;
        nprt = nprt + 1;



        elostcon=elost1 + elost2 + elost3;
        elost=elostconduc + elostcon + elostNcon;/*should already include the
nncon and the conduction portion*/
        if (elost>egain) {
            //            /*basically reverse everything that was just
done...this eliminates the need for a wfluxres, HOWEVER it might
underestimate the amount emitted!!!!!!*/
            //            Einlower2 = Einlower2 + Epost;
            //            sphemits2 = sphemits2 -1;
            //            sphemits2_DT=sphemits2_DT-1;
            //            elost2 = elost2-Epost;
            //            nprt = nprt -1;
            //
            //
            //
            //
            //            elostcon=elost1 + elost2 + elost3;
            //            elost=elostconduc + elostcon + elostNcon;/*should
already include the nncon and the conduction portion*/
            break;
        }


        j = nprt;
        prt[j].kind = 1;
        prt[j].mass = PARTRAT*MASSA;
        /* rotational energy for water */
        Rtest = 0.97*((double) random())/RANMAX;
        eps = 0.002;
        Ie = 0.0;

        while (Ie < Rtest){
            eps = eps + 0.1;
            Ie = Ie + (sqrt(eps - 0.1)*exp(-eps + 0.1) + sqrt(eps)*exp(-
eps))*0.05642;
```

```
        }

        prt[j].erot = PARTRAT*KB*Tsp*eps;

        //1. generate random position and convert to GCS (global coord.
syst.)
        //2. generate random velocity and convert to GCS
        //3. determine rotational energy for corresponding type of particle
        //4. determine entry point for new particle (see below)
        //5.

        dropnumber=3;
        dropemitpos(j,k);

        dropemitvel(j);


        /*dropemitcheck(j);*/

        xc = (int)( (prt[j].x + L/2) *NCX/xmax ); // local cell# in the
        yc = (int)(prt[j].y*NCY/ymax); // local cell# in the y dir
        zc = (int)(prt[j].z*NCZ/zmax); // local cell# in the z dir
        prt[j].cellno = xc + NCX*yc + NCX*NCY*zc;




        /* remove energy of particle from sphere tally */
        Epost = 0.5*prt[j].mass*(
prt[j].u*prt[j].u+prt[j].v*prt[j].v+prt[j].w*prt[j].w ) + prt[j].erot +
PARTRAT*(EvapT - 3.*KB*Tsp);

        Einlower3 = Einlower3 - Epost; //sphere tally, hence removal
        sphemits3 = sphemits3 + 1;
        sphemits3_DT++;
        elost3 = elost3 + Epost;
        nprt = nprt + 1;


        elostcon=elost1 + elost2 + elost3;
        elost=elostconduc + elostcon + elostNcon;/*should already include the
nncon and the conduction portion*/
        if (elost>egain) {
            //            /*basically reverse everything that was just
done...this eliminates the need for a wfluxres, HOWEVER it might
underestimate the amount emitted!!!!!!*/            Einlower3 = Einlower3 +
Epost;
            //            sphemits3 = sphemits3 -1;
            //            sphemits3_DT=sphemits3_DT-1;
            //            elost3 = elost3-Epost;
            //            nprt = nprt -1;
            //
            //
            //
            //
```

```
            //            elostcon=elost1 + elost2 + elost3;
            //            elost=elostconduc + elostcon + elostNcon;/*should
already include the nncon and the conduction portion*/
            break;
        }




    }/*should end the loop*/


    wfluxres = elost - egain;/*considering that I have the above if
statements with "breaks", wfluxres should really remain zero throughout the
entire simulation */

} //ends function


////////////////////////////////////////////////////////////////////////////
/////




void dropemitpos(int j, int k)
{
    double x1,y1,z1,x2,y2,z2,x3,y3,z3, R;
    double theta, phi,del_phi, del_theta_hl, del_theta_ll,uh;



    uh=gapy/sin(30.*PI/180.);
    del_theta_ll= asin(uh/(RSP+0.5*gapz));//ll=lower limit
    del_theta_hl=asin(0.5*gapz/(RSP +0.5*gapz));//hl=high limit


    switch(dropnumber){



        case 1:
            /*REMEMBER THAT FOR POSITION-GENERATION THE COORDINATE SYSTEM IS
INVERTED about the y!!!!!*/
            //random position generation droplet 1
            R=((double) random())/RANMAX;
            phi= 120.*PI/180. + PI/3.*R;
```

```
            theta = ca + 10.;//just to initialize while loop


            if (ca <= PI/2.){
                while (theta > (ca-del_theta_hl) || theta < del_theta_ll){
                    theta=acos(((double) random())/RANMAX); //in
degrees...this should sample between 0 and 90 degrees
                }
            }
            else {/*elseif ca > PI/2*/
                while (theta > (ca-del_theta_hl) || theta < del_theta_ll){
                    theta= acos(2.*((double) random())/RANMAX-1); //in
degrees...this should sample between 0 and 180 degrees
                }
            }



            del_phi=asin( gapy/((RSP +0.5*gapz)*sin(theta)) );

            while (phi<(120.*PI/180.+del_phi) || phi>(120.*PI/180. + PI/3. -
del_phi)){
                R=((double) random())/RANMAX;
                phi= 120.*PI/180. + PI/3.*R;

            }//this is to avoid a particle on either boundary



            //// "1" subscript denotes the local fixed coordinate system at
center of droplet
            //note that the angles were sampled on a rotated coordinate
system (see
            //handout from prof. carey and make sure to document on writeup)

            x1= (RSP+ 0.5*gapz)*sin(theta)*cos(phi);
            y1= (RSP+ 0.5*gapz)*sin(theta)*sin(phi);
            z1= (RSP+ 0.5*gapz)*cos(theta);


            //translating to global coordinates
            /*REMEMBER THAT FOR POSITION-GENERATION THE COORDINATE SYSTEM IS
INVERTED about y!!!!!*/
            x1=-( (-cx1) + x1);/*note that for this case, the x1 generated
above is negative, and also note that cx1 is negative, so we multiply by -1*/
            y1= y1;
            z1= cz1-z1;
            //
            //



            prt[j].x=x1;
            prt[j].y=y1;
            prt[j].z=z1;
```

```
            return;

        case 2:
            /*REMEMBER THAT FOR POSITION-GENERATION THE local COORDINATE
SYSTEM IS INVERTED about y!!!!!*/
            //random position generation droplet 2
            R=((double) random())/RANMAX;
            phi= 240.*PI/180. + PI/3.*R;
            theta= ca + 10;//just to initialize while loop




            if (ca <= PI/2.){
                while (theta > (ca-del_theta_hl) || theta < del_theta_ll){
                    theta=acos(((double) random())/RANMAX); //in
degrees...this should sample between 0 and 90 degrees
                }
            }
            else {/*elseif ca > PI/2*/
                while (theta > (ca-del_theta_hl) || theta < del_theta_ll){
                    theta= acos(2.*((double) random())/RANMAX-1); //in
degrees...this should sample between 0 and 180 degrees
                }
            }


            del_phi=asin( gapy/((RSP +0.5*gapz)*sin(theta)) );

            while (phi<240.*PI/180.+del_phi || phi>(240.*PI/180. + PI/3.-
del_phi)){
                R=((double) random())/RANMAX;
                phi= 240.*PI/180. + PI/3.*R;
            }//this is to avoid a particle on either boundary


            x2= (RSP+ 0.5*gapz)*sin(theta)*cos(phi);//li denotes local
inverted
            y2= (RSP+ 0.5*gapz)*sin(theta)*sin(phi);
            z2= (RSP+ 0.5*gapz)*cos(theta);




            //translating to global coordinates
            /*REMEMBER THAT FOR POSITION-GENERATION THE local COORDINATE
SYSTEM IS INVERTED about y!!!!!*/
            x2=-x2;
            y2= cy2 + y2;/*where H is the height of the triangle predefined
as a global constant.  Eqn Makes sense because y2 (the first one generated
above) is negative.  H=L*sin(2*PI/3) =L*sind(60).*/
```

```
            z2=cz2-z2;



            prt[j].x=x2;
            prt[j].y=y2;
            prt[j].z=z2;
            return;


        case 3:
            /*REMEMBER THAT FOR POSITION-GENERATION THE local COORDINATE
SYSTEM IS INVERTED about y!!!!!*/

            //random position generation droplet 3
            R=((double) random())/RANMAX;
            phi= PI/3.*R;
            theta= ca + 10;//just to initialize while loop




            if (ca <= PI/2.){
                while (theta > (ca-del_theta_hl) || theta < del_theta_ll){
                    theta=acos(((double) random())/RANMAX); //in
degrees...this should sample between 0 and 90 degrees, but not exactly one or
the other
                }
            }
            else {/*elseif ca > PI/2*/
                while (theta > (ca-del_theta_hl) || theta < del_theta_ll){
                    theta= acos(2.*((double) random())/RANMAX-1); //in
degrees...this should sample between 0 and 180 degrees
                }
            }

            del_phi=asin( gapy/((RSP +0.5*gapz)*sin(theta)) );


            while (phi<del_phi || phi>(PI/3. - del_phi)){
                R=((double) random())/RANMAX;
                phi= PI/3.*R;
            }//this is to avoid a particle on either boundary



            x3= (RSP+ 0.5*gapz)*sin(theta)*cos(phi);
            y3= (RSP+ 0.5*gapz)*sin(theta)*sin(phi);
            z3= (RSP+ 0.5*gapz)*cos(theta);



            //translating to global coordinates
            /*REMEMBER THAT FOR POSITION-GENERATION THE local COORDINATE
SYSTEM IS INVERTED!!!!!*/
```

```
            x3= (cx3)-x3;
            y3= y3;
            z3=cz3-z3;




            prt[j].x=x3;
            prt[j].y=y3;
            prt[j].z=z3;
            return;
    }/*should end the switch*/
}/*should end the function*/
////////////////////////////////////////////////////////////////////////////
/////

void dropemitvel(int j)
{
    double xemit, yemit, zemit, cT, phi, r, u, v, w,R1;
    double xemitc, yemitc, zemitc, thetapos, phipos;
    double rot1[3][3];
    double totrot[3][3];
    double randvel[3][1];

    /*uses the positions that were created in dropemitpos--> [prt[j].x,
prt[j].y, prt[j].z]*/



    switch (dropnumber){ //remember that "switches" in .c need "breaks"
        case 1: //diffuse reflection from droplet 1
            xemit= prt[j].x;//xold + dtmin*prt[j].u; //these are all with the
global coordinates
            yemit= prt[j].y;//yold + dtmin*prt[j].v;
            zemit= prt[j].z;//zold + dtmin*prt[j].w;

            /* GCS=global coordinate system */


            xemitc= xemit-cx1;/*these are all with respect to the center of
the droplet, not global anymore...easy because [cx1,cy1,cz1] are all with
respect to the GCS*/
            yemitc= yemit-cy1;
            zemitc= zemit-cz1;




            //  Insert randomly generated velocity components with local CS
here.
            cT = sqrt(PARTRAT*KB*Tsp/prt[j].mass);
            phi = 2.*PI*( ((double) random())/RANMAX );
            R1=((double) random())/RANMAX;
            if (R1<=0)  {
                while (R1<=0){
```

```
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        u = cT*r*cos(phi);
        v = cT*r*sin(phi);

        R1=((double) random())/RANMAX;
        if (R1<=0)  {
            while (R1<=0){
                R1=((double) random())/RANMAX;
            }
        }
        r = sqrt(-2.*log(R1));
        w = cT*r;

        randvel[0][0]=u;
        randvel[1][0]=v;
        randvel[2][0]=w;
        //end random generation here.

        phipos= atan(yemitc/xemitc); //pos denotes that these are angles
for determining position
        thetapos = atan(-zemitc/sqrt(xemitc*xemitc + yemitc*yemitc));

        double Yxz1[3][3]={
            {cos(thetapos), 0, sin(thetapos)},
            {0, 1, 0},
            { -sin(thetapos), 0, cos(thetapos)}
        } ;/*%rotation #1, about y, and from x to z*/

        double Xyz2[3][3]={
            {1, 0, 0},
            {0, cos(phipos), sin(phipos)},
            {0, -sin(phipos), cos(phipos)}
        }; /*%rotation #2, about x, and from y to z*/

        double Yxz3[3][3]={
            {cos(90.*PI/180.), 0, sin(90.*PI/180.)},
            {0, 1, 0},
            {-sin(90.*PI/180.), 0, cos(90.*PI/180.)}
        };/*%rotation #3, about y, and from x to z*/

        /*First Rotation*/
        mat_mult3x3(Yxz3,Xyz2,rot1,3,3,3);
        /*Second Rotation*/
        mat_mult3x3(rot1,Yxz1,totrot,3,3,3);

        prt[j].u=mat_mult_vel(totrot,randvel,0,3);
        prt[j].v=mat_mult_vel(totrot,randvel,1,3);
        prt[j].w=mat_mult_vel(totrot,randvel,2,3);



        /*dts = DT*((double) random())/RANMAX;*/
        prt[j].x = xemit;
```

```
            prt[j].y = yemit;
            prt[j].z = zemit;

            return;

        case 2: // diffuse reflection from droplet 2
            //with respect to global coordinate system
            xemit= prt[j].x;
            yemit= prt[j].y;
            zemit= prt[j].z;


            xemitc= xemit-cx2;//these are all with respect to the center of
the droplet, not global anymore
            yemitc= yemit-cy2;
            zemitc= zemit-cz2;



            //  Insert randomly generated velocity components with local CS
here.
            cT = sqrt(PARTRAT*KB*Tsp/prt[j].mass);
            phi = 2.*PI*( ((double) random())/RANMAX );
            R1=((double) random())/RANMAX;
            if (R1<=0)  {
                while (R1<=0){
                    R1=((double) random())/RANMAX;
                }
            }
            r = sqrt(-2.*log(R1));
            u = cT*r*cos(phi);
            v = cT*r*sin(phi);


            R1=((double) random())/RANMAX;
            if (R1<=0)  {
                while (R1<=0){
                    R1=((double) random())/RANMAX;
                }
            }
            r = sqrt(-2.*log(R1));
            w = cT*r;
            randvel[0][0]=u;
            randvel[1][0]=v;
            randvel[2][0]=w;
            //end random generation here.

            phipos= atan(xemitc/-yemitc);//pos denotes that these are angles
for determining position
            thetapos = atan(-zemitc/sqrt(xemitc*xemitc + yemitc*yemitc));


            double Xzy1[3][3]={
                {1, 0, 0},
                { 0, cos(thetapos), -sin(thetapos)},
                {0, sin(thetapos), cos(thetapos)}
```

```
            }; /*rotation #, about x, and from z to y*/
            double Yxz2[3][3]={
                {cos(phipos), 0, sin(phipos)},
                { 0, 1, 0},
                { -sin(phipos), 0, cos(phipos)}
            };/*rotation #, about y, and from x to z*/
            double Xzy3[3][3]={
                {1, 0, 0},
                {0, cos(90.*PI/180.), -sin(90.*PI/180.)},
                {0, sin(90.*PI/180.), cos(90.*PI/180.)}
            }; /*rotation #, about x, and from z to y*/

            /*First Rotation*/
            mat_mult3x3(Xzy3,Yxz2,rot1,3,3,3);
            /*Second Rotation*/
            mat_mult3x3(rot1,Xzy1,totrot,3,3,3);



            prt[j].u=mat_mult_vel(totrot,randvel,0,3);
            prt[j].v=mat_mult_vel(totrot,randvel,1,3);
            prt[j].w=mat_mult_vel(totrot,randvel,2,3);


            prt[j].x = xemit;
            prt[j].y = yemit;
            prt[j].z = zemit;
            return;


        case 3: //diffuse reflection from droplet 3
            //with respect to global coordinate system
            xemit= prt[j].x;
            yemit= prt[j].y;
            zemit= prt[j].z;


            xemitc= xemit-cx3;//these are all with respect to the center of
the droplet, not global anymore
            yemitc= yemit-cy3;
            zemitc= zemit-cz3;



            //  Insert randomly generated velocity components with local CS
here.
            cT = sqrt(PARTRAT*KB*Tsp/prt[j].mass);
            phi = 2.*PI*( ((double) random())/RANMAX );
            R1=((double) random())/RANMAX;
            if (R1<=0)  {
                while (R1<=0){
                    R1=((double) random())/RANMAX;
                }
            }
            r = sqrt(-2.*log(R1));
            u = cT*r*cos(phi);
```

```
            v = cT*r*sin(phi);

            R1=((double) random())/RANMAX;
            if (R1<=0)  {
                while (R1<=0){
                    R1=((double) random())/RANMAX;
                }
            }
            r = sqrt(-2.*log(R1));
            w = cT*r;

            randvel[0][0]=u;
            randvel[1][0]=v;
            randvel[2][0]=w;

            //end random generation here.

            phipos= atan(yemitc/-xemitc);//pos denotes that these are angles
for determining position
            thetapos = atan(-zemitc/sqrt(xemitc*xemitc + yemitc*yemitc));

            double Yzx1[3][3]={
                {cos(thetapos), 0, -sin(thetapos)},
                {0, 1, 0},
                {sin(thetapos), 0, cos(thetapos)}
            };/*%rotation #, about y, and from z to x*/
            double Xyz22[3][3]={
                {1, 0, 0},
                {0, cos(phipos), sin(phipos)},
                {0, -sin(phipos), cos(phipos)}
            }; /*%rotation #, about x, and from y to z*/

            double Yzx3[3][3]={
                {cos(90.*PI/180.), 0, -sin(90.*PI/180.)},
                {0, 1, 0},
                {sin(90.*PI/180.), 0, cos(90.*PI/180.)}
            };/*%rotation #, about y, and from z to x*/

            /*First Rotation*/
            mat_mult3x3(Yzx3,Xyz22,rot1,3,3,3);
            /*Second Rotation*/
            mat_mult3x3(rot1,Yzx1,totrot,3,3,3);


            prt[j].u=mat_mult_vel(totrot,randvel,0,3);
            prt[j].v=mat_mult_vel(totrot,randvel,1,3);
            prt[j].w=mat_mult_vel(totrot,randvel,2,3);



            prt[j].x = xemit;
            prt[j].y = yemit;
            prt[j].z = zemit;
            return;
    }
```

```
}



////////////////////////////////////////////////////////////////////////////
/////
void dropemitcheck(int j)
{
    int test, i,k;
    double norm_dist, i_vec, j_vec, c, prtx_mock, prty_mock, prtx_mock2,
prty_mock2, dum, ref_vel[3];
    double norm_mag,b, dts;


    /*BASICALY, WE NEVER WANT A PARTICLE TO INITIATE ON A BOUNDARY BECAUSE IT
CAN BE MESSED UP IN CHKSPACE BY AUTOMATICALLY HAVING ONE OF THE "dt-checks"
EQUAL TO ZERO, SO IT WILL CORRECT IT AUTOMATICALLY, BUT IT CAN DO IT WRONGLY
SINCE THE PARTICLE HASN'T REALLY MOVED and rather just started there.
Basically, chkspace works great for particles that have already moved, not
initiated particles.  Also, note that the only one that is an exception is
the space check at the end of this function, since it doesn't matter if it
was already going to go into space.  Particles have NOT been moved when
reaching this point*/



    //this function checks that the particle randomly generated in dropemit
does
    //not end up in a conflict.  Mainly, that if a particle does happen to be
    //generated on an edge, that it does not go out of the domain of the
    //triangular prism, and if it does, the velocity is corrected (reflected)
    //appropriately depending on on which boundary it might go into



    /*%(unit vector 2) points to -i, -j, and (unit vector 1) points to +i,-j
     %% creating unit vector of the surface normal*/
    norm_dist=L*0.5*sin(60.*PI/180.);/*%60 and 30 degrees have to do with it
being in a triangle*/
    i_vec=norm_dist*cos(30.*PI/180.);
    j_vec=norm_dist*sin(30.*PI/180.);

    norm_mag= sqrt(i_vec*i_vec + j_vec*j_vec);/*%normal vector
magnitude...this should equal norm_dist*/
    double uv_n1[3]={i_vec/norm_mag, -j_vec/norm_mag,0} ;/*%normal unit
vector of line connecting drop 1 to 2*/
    double uv_n2[3]={ -i_vec/norm_mag, -j_vec/norm_mag,0} ;/*%normal unit
vector of line connecting drop 3 to 2*/

    double vel[]={prt[j].u,prt[j].v,prt[j].w};
    c=sqrt(prt[j].u*prt[j].u + prt[j].v*prt[j].v + prt[j].w*prt[j].w);
    double uv_v[]={vel[0]/c,vel[1]/c,vel[2]/c};/* %1x3 vector...unit vector
of incoming velocity,where the velocity in z-direction(w) shouldn't ever be
altered */
```

```
    b=H;

    dts = 0.5*DT*((double) random())/RANMAX;
    while (dts<=0){
        dts = 0.5*DT*((double) random())/RANMAX;

    }
    i=0;
    k=0;

    test=2; //just to initiate the loop

    while (test >0){
        i++;
        if (i>10){
            while (dts<=0){
                dts = 0.1*DT*((double) random())/RANMAX;

            }

            k++;
            if (k>10){
                /*comes in here if dts was changed 10 times, unsuccessfully*/
                printf("failed!!!!");
                exit(EXIT_FAILURE);
            }
        }

        /*this mock position is basically to test what would happen if it
were to go move one time step*/
        prtx_mock = prt[j].x + prt[j].u*DT;/*this is just a mock move to see
if it is on a boundary,but directed to go out.*/
        prty_mock = prt[j].y + prt[j].v*DT;

        prtx_mock2=(prt[j].x+prt[j].u*dts);
        prty_mock2=(prt[j].y+dts*prt[j].v);
        //// right leg of triangular boundary(think about uniting the 2 ifs
with an
        //"&&" statment)
        if (prt[j].y >= m1*prt[j].x + b || prty_mock2>=m1*prtx_mock2+b){

            if (prty_mock > m1*prtx_mock + b){/*here, if the particle did
happen to land on a boundary, it is checked to see if it is directed to go
out of the boundary by looking at the "mock position" and corrected if that's
the case*/


                ref_vel[0]=-
1.*(2.*(uv_v[0]*uv_n1[0]+uv_v[1]*uv_n1[1]+uv_v[2]*uv_n1[2])*uv_n1[0]-
uv_v[0])*c;
                ref_vel[1]=-
1.*(2.*(uv_v[0]*uv_n1[0]+uv_v[1]*uv_n1[1]+uv_v[2]*uv_n1[2])*uv_n1[1]-
uv_v[1])*c;
                ref_vel[2]=-
1.*(2.*(uv_v[0]*uv_n1[0]+uv_v[1]*uv_n1[1]+uv_v[2]*uv_n1[2])*uv_n1[2]-
uv_v[2])*c;
```

```
                /*ref_vel stands for reflected velocity*/


                prt[j].u=ref_vel[0];
                prt[j].v=ref_vel[1];
                prt[j].w=ref_vel[2];


                test=1; //actually, this could/should be zero and would
probably be just fine if everything else goes smoothly
            }
        }

        else {
            test=0; /*now that the loop has started, set test to zero*/
        }



        //// left leg of triangular boundary (think about uniting the 2 ifs
with an
        //"&&" statment)
        if (prt[j].y >= m2*prt[j].x + b || prty_mock2>=m2*prtx_mock2+b){

            if (prty_mock > m2*prtx_mock + b){/*here, if the particle did
happen to land on a boundary, it is checked to see if it is directed to go
out of the boundary by looking at the "mock position" and corrected if that's
the case*/

                ref_vel[0]=-
1.*(2.*(uv_v[0]*uv_n2[0]+uv_v[1]*uv_n2[1]+uv_v[2]*uv_n2[2])*uv_n2[0]-
uv_v[0])*c;
                ref_vel[1]=-
1.*(2.*(uv_v[0]*uv_n2[0]+uv_v[1]*uv_n2[1]+uv_v[2]*uv_n2[2])*uv_n2[1]-
uv_v[1])*c;
                ref_vel[2]=-
1.*(2.*(uv_v[0]*uv_n2[0]+uv_v[1]*uv_n2[1]+uv_v[2]*uv_n2[2])*uv_n2[2]-
uv_v[2])*c;

                /*ref_vel stands for reflected velocity*/

                prt[j].u=ref_vel[0];
                prt[j].v=ref_vel[1];
                prt[j].w=ref_vel[2];

                /*uv_n=uv_n2;
                 ref_vel=-1*(2*(sum(uv_v.*uv_n))*uv_n-uv_v )*c; //in the for
(reflected unit vector)*c, where c is magnitude of velocity
                 prt[j].u=ref_vel(1);
                 prt[j].v=ref_vel(2);*/

                /*prt[j].x += prt[j].u*DT*dts;
                 prt[j].y += prt[j].v*DT*dts;
                 prt[j].z += prt[j].w*DT*dts;
                 */
```

```
                test= test + 1;
            }
        }


        //// base of trianglular boundary (think about uniting the 2 ifs with
an
        //"&&" statment)
        if ( prt[j].y<=0 || (prt[j].y+dts*prt[j].v)<=0 ){
            if (prty_mock<0){/*here, if the particle did happen to land on a
boundary, it is checked to see if it is directed to go out of the boundary by
looking at the "mock position" and corrected if that's the case*/

                prt[j].v = -prt[j].v;
                /*
                 prt[j].x += prt[j].u*DT*dts;
                 prt[j].y += prt[j].v*DT*dts;
                 prt[j].z += prt[j].w*DT*dts;

                 */


                test= test + 1;
            }
        }




        //// condensing wall
        if (prt[j].z >= RSP*(1-cos(ca)) || (prt[j].z + prt[j].w*dts)>=
RSP*(1-cos(ca)) ){/*here, if the particle did happen to land on a boundary,
it is checked to see if it is directed to go out of the boundary by looking
at its velocity (prt[j].w) and corrected if that's the case.  Note that w>0
is actually in the downward direction, away from the simulation domain*/

            if (prt[j].w>0){
                /*drop number should still be active, so using dropemitvel
will just generate a new normally distributed velocity*/


                dropemitvel(j); //basically generate new velocity since
condensing wall is diffusive

                /*            prt[j].x += prt[j].u*DT*dts;
                 prt[j].y += prt[j].v*DT*dts;
                 prt[j].z += prt[j].w*DT*dts;

                 */
                test= test + 1;
            }
        }
```

```
        //// going into space
        if (prt[j].z <= 0 || (prt[j].z + prt[j].w*dts) <=0 ){/*NOTE THAT the
.w<0 is redundant since no particle should in theory have a velocity directed
towards the droplet, since the only way ".z=0 is if the position is at the
peak of the droplet */

            prt[j].z = 0;
            //leave alone, it will fix itself in "advance"...WILL IT???


            /*spacescape= spacescape + 1;
             nprt=nprt-1; //this results to be nprt= nprt when we do
"nprt=nprt+1" afterwards, outside this file in dropemit.
             //might not want to have this if something is done with nprt
before
             //"1" is added to it.*/
            test=0; //make sense?  I think so!!!  basically don't consider
this particle anymore!!
        }


    }


    prt[j].x += prt[j].u*dts;
    prt[j].y += prt[j].v*dts;
    prt[j].z += prt[j].w*dts;

}



/* BBBBBBBBBBB  Function to deal with lost particles */
void loseprt(j)
{
    /* replace lost particle with last particle */
    int nmax;
    nmax = nprt - 1;
    /*      memcpy(&prt[j],&prt[nmax],sizeof(struct part));  */
    prt[j].kind = prt[nmax].kind;
    prt[j].cellno = prt[nmax].cellno;
    prt[j].mass = prt[nmax].mass;
    prt[j].x = prt[nmax].x;
    prt[j].y = prt[nmax].y;
    prt[j].z = prt[nmax].z;
    prt[j].u = prt[nmax].u;
    prt[j].v = prt[nmax].v;
    prt[j].w = prt[nmax].w;
    prt[j].erot = prt[nmax].erot;
    nprt--;
    lost++;
    losscheck = 0;
    dtres=0;
```

```
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to select collision pairs and collect
 occupancy info for each cell */
void collect()
{
    double ptcount;
    int i, j, m1, m2, icount, icell, xdum;
    int iptcount, space[NCELLS];

    for (i=0; i< NCELLS; i++) {
        space[i] = -1;  /* set arrays to zero */
        sampsize[i] = 0;
    }
    ptcount = (double) nprt;
    iptcount = (int) (2.0*ptcount);
    /* randomly swap particle numbers */
    for (i=0; i< iptcount; i++) {
        m1 = (int) (0.9999*ptcount*((double) random())/RANMAX);
        m2 = (int) (0.9999*ptcount*((double) random())/RANMAX);
        swap(m1,m2);
    }

    icount = 0;
    for (j=0; j< nprt; j++) {
        icell = prt[j].cellno;




        if (space[icell] < 0)
        {
            space[icell] = j;
        }
        else {
            pt1[icount] = j;  /* store candidate pair */
            pt2[icount] = space[icell];

            space[icell] = -1;
            icount++;
            sampsize[icell]++;
        }
    }

    nopairs = icount;
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to swap particle numbers for two particles */
void swap(i,j)
{
```

```
    int ndum;
    double dum;
     struct part prtdum;
    /*      memcpy(&prtdum,&prt[i],sizeof(struct part));
     memcpy(&prt[i],&prt[j],sizeof(struct part));
     memcpy(&prt[j],&prtdum,sizeof(struct part));  */
    ndum = prt[i].kind;
    prt[i].kind = prt[j].kind;
    prt[j].kind = ndum;
    ndum = prt[i].cellno;
    prt[i].cellno = prt[j].cellno;
    prt[j].cellno = ndum;
    dum = prt[i].mass;
    prt[i].mass = prt[j].mass;
    prt[j].mass = dum;
    dum = prt[i].x;
    prt[i].x = prt[j].x;
    prt[j].x = dum;
    dum = prt[i].y;
    prt[i].y = prt[j].y;
    prt[j].y = dum;
    dum = prt[i].z;
    prt[i].z = prt[j].z;
    prt[j].z = dum;
    dum = prt[i].u;
    prt[i].u = prt[j].u;
    prt[j].u = dum;
    dum = prt[i].v;
    prt[i].v = prt[j].v;
    prt[j].v = dum;
    dum = prt[i].w;
    prt[i].w = prt[j].w;
    prt[j].w = dum;
    dum = prt[i].erot;
    prt[i].erot = prt[j].erot;
    prt[j].erot = dum;
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


/* BBBBBBBBBBB  Function to execute collision of candidate pairs */
void collide()
{
     double vrel, np, Rtest, ucm, vcm, wcm, mr, Ecoll, r, rlim;
     double fracrt, vrelp, Erelp, Erotp, nx, ny, nz, eps, sinki, theta;
     double prob, fzr, etransr, Epre, Epost;
     double d0, d1, d2, d3, d4, fracr1;
     double lamAA, lamBB, lamAB, D12, den;
     int i, j, m1, m2, ifzr, isw;
     den = PRESS/(KB*TAMB);
     D12 = (DMA + DMB)/2.;
     lamAA = 1.0/(PI*1.414*den*DMA*DMA);
     lamBB = 1.0/(PI*1.414*den*DMB*DMB);
     lamAB = 1.0/(PI*1.414*den*D12*D12);
     for (i=0; i< nopairs; i++) {  /* do for all candidate pairs */
        m1 = pt1[i];
```

```
        m2 = pt2[i];
        np = (int) mnp[prt[m1].cellno];
        vrel = sqrt( (prt[m1].u - prt[m2].u)*(prt[m1].u - prt[m2].u)
                    + (prt[m1].v - prt[m2].v)*(prt[m1].v - prt[m2].v)
                    + (prt[m1].w - prt[m2].w)*(prt[m1].w - prt[m2].w) );
        prob = 1.5*np*vrel*DT/(1.414*freeparden*freelam);
        switch (prt[m1].kind*prt[m2].kind){
            case 1: /* water-water */
                /* prob = 1.5*np*vrel*DT/(1.414*freeparden*lamAA); */
                break;
            case 2: /* water-argon */
                /* prob = 1.5*np*vrel*DT/(1.414*freeparden*lamAB); */
                break;
            case 4: /* argon-argon */
                /* prob = 1.5*np*vrel*DT/(1.414*freeparden*lamBB); */
                break;
        } /* end of switch */
        Rtest =((double) random())/RANMAX;
        if (prob > Rtest){          /* test whether they collide
                                   -if yes, continue */

            ifzr = (int)(IZROT + ((double) random())/RANMAX);
            fzr = (double) ifzr;
            r = sqrt( prt[m1].x*prt[m1].x + prt[m1].y*prt[m1].y
                     + prt[m1].z*prt[m1].z );
            rlim = RSP + 2.0*L_CELL;
            if (r > RSP && r < rlim) collcount++;
            Epre = 0.5*prt[m1].mass*(prt[m1].u*prt[m1].u
                                     + prt[m1].v*prt[m1].v +
prt[m1].w*prt[m1].w)
            + prt[m1].erot
            +  0.5*prt[m2].mass*(prt[m2].u*prt[m2].u
                                   + prt[m2].v*prt[m2].v + prt[m2].w*prt[m2].w)
            + prt[m2].erot;

            ucm = (prt[m1].u*prt[m1].mass +
                   prt[m2].u*prt[m2].mass);
            ucm /= (prt[m1].mass + prt[m2].mass);
            vcm = (prt[m1].v*prt[m1].mass +
                   prt[m2].v*prt[m2].mass);
            vcm /= (prt[m1].mass + prt[m2].mass);
            wcm = (prt[m1].w*prt[m1].mass +
                   prt[m2].w*prt[m2].mass);
            wcm /= (prt[m1].mass + prt[m2].mass);
            mr = (prt[m1].mass*prt[m2].mass)/(prt[m1].mass+prt[m2].mass);
            etransr = 0.5*mr*vrel*vrel;

            Ecoll = etransr + prt[m1].erot + prt[m2].erot;

            /* sample distribution for fraction
             in relative translation */
            isw = prt[m1].kind*prt[m2].kind;
            switch (isw) {
                case 1:  /* water - water */
                    d0 = ((double) random())/RANMAX;
                    d1 = d0 - 0.1808;
```

```
                    d2 = d0 - 0.5248;
                    d3 = d0 - 0.8208;
                    d4 = d0 - 1.0;
                    fracrt = - 6.133*d0*d2*d3*d4 + 15.752*d0*d1*d3*d4
                    - 21.533*d0*d1*d2*d4 + 14.335*d0*d1*d2*d3;
                    fracrt *= 0.99999;

                    Erelp = Ecoll*fracrt*fzr + (1. - fzr)*etransr;
                    Erotp = Ecoll - Erelp;
                    d0 = ((double) random())/RANMAX;
                    d1 = d0 - 0.1424;
                    d2 = d0 - 0.5000;
                    d3 = d0 - 0.8576;
                    d4 = d0 - 1.0;
                    fracr1 = - 6.403*d0*d2*d3*d4 + 15.640*d0*d1*d3*d4
                    - 25.614*d0*d1*d2*d4 + 16.377*d0*d1*d2*d3;
                    fracr1 *= 0.9999;
                    fracr1 += 0.00002;
                    prt[m1].erot = fzr*Erotp*fracr1
                    + (1.-fzr)*prt[m1].erot;
                    prt[m2].erot = Erotp - prt[m1].erot;
                    break;
                case 2:  /* water - Nitrogen*/
                    d0 = ((double) random())/RANMAX;
                    d1 = d0 - 0.1413;
                    d2 = d0 - 0.4423;
                    d3 = d0 - 0.7470;
                    d4 = d0 - 1.0;
                    fracrt = - 9.041*d0*d2*d3*d4 + 17.681*d0*d1*d3*d4
                    - 17.202*d0*d1*d2*d4 + 8.253*d0*d1*d2*d3;
                    fracrt *= 0.99999;

                    Erelp = Ecoll*fracrt*fzr + (1. - fzr)*etransr;
                    Erotp = Ecoll - Erelp;
                    Rtest=((double) random())/RANMAX;
                    fracr1=pow(Rtest, 0.66667);

                    prt[m1].erot = fzr*Erotp*fracr1 + ( 1.-
fzr)*prt[m1].erot;
                    prt[m2].erot = Erotp - prt[m1].erot;
                    break;

                case 4:  /* Nitrogen-Nitrogen */
                    /* no rotation - all translation */
                    Rtest= ((double) random())/RANMAX;
                    theta= acos(1.-2.*Rtest);
                    fracrt = 0.5 + cos(0.3333333*theta + 1.3333333*PI);
                    Erelp = Ecoll*fracrt*fzr + (1.-fzr)*etransr;
                    Erotp = Ecoll - Erelp;
                    prt[m1].erot = fzr*(Erotp*((double) random())/RANMAX) +
(1.-fzr)*prt[m1].erot;
                    prt[m2].erot= Erotp - prt[m1].erot;
                    break;
            } /* end of switch */
```

```
            vrelp = sqrt(Erelp/(mr/2.));
            nx = 2.*((double) random())/RANMAX - 1.0;
            eps = 2.*PI*((double) random())/RANMAX;
            sinki = sqrt(1. - nx*nx);
            ny = sinki*cos(eps);
            nz = sinki*sin(eps);
            prt[m1].u = ucm + mr*vrelp*nx/prt[m1].mass;
            prt[m1].v = vcm + mr*vrelp*ny/prt[m1].mass;
            prt[m1].w = wcm + mr*vrelp*nz/prt[m1].mass;
            prt[m2].u = ucm - mr*vrelp*nx/prt[m2].mass;
            prt[m2].v = vcm - mr*vrelp*ny/prt[m2].mass;
            prt[m2].w = wcm - mr*vrelp*nz/prt[m2].mass;
            Epost = 0.5*prt[m1].mass*(prt[m1].u*prt[m1].u
                                      + prt[m1].v*prt[m1].v +
prt[m1].w*prt[m1].w)
            + prt[m1].erot
            +  0.5*prt[m2].mass*(prt[m2].u*prt[m2].u
                                 + prt[m2].v*prt[m2].v + prt[m2].w*prt[m2].w)
            + prt[m2].erot;
            Ecgain += Epost - Epre;
            nocoll++;
            nocolrot += ifzr;
        }
    }
}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



/* BBBBBBBBBBB  Function to collect statistics on particles to allow
 later determination of statistical properties for each cell */
void statcell(k)
{
    int i, j, icell;
    double count1, count2, stepcount, denom, xdum;

    for (j=0; j< nprt; j++) {
        icell = prt[j].cellno;
        /* if (icell<0)
         {
         xdum = prt[j].x;
         printf("\n   x = %8.3e", xdum);
         xdum = prt[j].y;
         printf("\n   y = %8.3e", xdum);
         xdum = prt[j].z;
         printf("\n   z = %8.3e", xdum);
         xdum = prt[j].v;
```

```
          printf("\n   v = %8.3e", xdum);
          xdum = icell;
          printf("\n   icell = %8.3e", xdum);
          xdum = cell[icell].countkind2;
          printf("\n   countkind = %8.3e", xdum);
          xdum = stepcount;
          printf("\n   stepcount = %8.3e", xdum);
          xdum = prt[j].u;
          printf("\n   u = %8.3e", xdum);
          xdum = prt[j].u;
          printf("\n   u = %8.3e", xdum);



          } */


         if (prt[j].kind == 1)
         {cell[icell].countkind1++;


         }




         cell[icell].countkind2++;
         cell[icell].etrsum += 0.5*prt[j].mass*(prt[j].u
                                                *prt[j].u + prt[j].v*prt[j].v
                                                +prt[j].w*prt[j].w);
         cell[icell].erotsum += prt[j].erot;
         cell[icell].usum += prt[j].u;
     }
     /* refine cell statistics */
     stepcount = (double) (k+1);
     for (i=0; i < NCELLS; i++){
         count1 = (double) cell[i].countkind1;
         count2 = (double) cell[i].countkind2;
         denom = (3.0*count1+1.5*count2 + EPS)*KB*PARTRAT;
         cell[i].temp = (cell[i].etrsum + cell[i].erotsum)/denom;
       cell[i].molconc1 = count1/(count1 + count2 + EPS);
         cell[i].umean = cell[i].usum/(count1 + count2 + EPS);
       mnp[i] = ((double) (count1 + count2))/stepcount;
     }



}
/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double density_liquid(double x)
{
     //%------------Saturation Properties of H2O--------------------%
     //%L is the Lagrangian interpolation of density of the liquid at given
```

```
    //%Temperature x in [kg/m3]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated

    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,75,     80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={999.8,      999.9,      999.7,      999.1,      998.2,
997, 995.6,  994,  992.2,      990.2,      988,  985.7,      983.2,
     980.5,  977.7,      974.8,      971.8,      968.6,      965.3,
     961.9,  958.4,      958.3,      954.7,      950.9,      947.1,
     943.1};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //density liquid
        double p262[]={943.11, 934.83, 926.13, 917.01, 907.45, 897.45,
887.00, 876.08, 864.66, 852.72, 840.22, 827.12, 813.37, 798.89, 783.63,
767.46, 750.28, 731.91, 712.14, 690.67, 667.09, 640.77, 610.67, 574.71,
527.59, 451.43};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }




    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
```

```
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
        //printf("\n lsize=%2.2i\n",lsize);

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function


/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double enthalpy(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of enthalpy of vaporization at given
    //%Temperature x in [J/kg-K]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated
    //%This is h_lv!!!!  LATENT HEAT


    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={2500.92, 2489.04, 2477.19,    2465.35,    2453.52,
     2441.68,     2429.82,    2417.92,    2405.98,    2394, 2381.95,
     2369.83,     2357.65,    2345.38,    2333.03,    2320.57,    2308.01,
     2295.32,     2282.49,    2279.52,    2256.47,    2256.4,     2243.12,
     2229.64,     2215.99,    2202.12};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);
```

```
    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //enthalpy of vaporization
        double p262[]={2202.6, 2174.2, 2144.7, 2114.3, 2082.6, 2049.5,
2015.0, 1978.8, 1940.7, 1900.7, 1858.5, 1813.8, 1766.5, 1716.2, 1662.5,
1605.2, 1543.6, 1477.1, 1404.9,
            2727.9-1402.2,
            1238.6,
            2666.0-1525.9,
            1027.9,
            2563.6-1670.9,
            720.5,
            2334.5-1890.7};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }

    for (i=0;i<len26;i++)
    {
        p26[i]=p26[i]*1000;
    }

    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);

    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
        //printf("\n lsize=%2.2i\n",lsize);
```

```
        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


double pressure(double x)
{
    //       %------------Saturation Properties of H2O--------------------%
    //       %L is the Lagrangian interpolation of Pressure at given
Temperature
    //       %pressure in [Pa]
    //       %x= ;%temperature in [Kelvin] at wich the property is to be
evaluated



    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5, 10, 15, 20, 25, 30 ,35 ,40 ,45 ,50 ,55 ,60 ,65 ,70
,75 ,80 ,85 ,90, 95 ,99.97 ,100 ,105 ,110 ,115, 120}  ;//vector of temps
    double p26[]={0.00061, 0.00087 ,0.00123 ,0.00171 ,0.00234 ,0.00317,
0.00425, 0.00563, 0.00738, 0.00959, 0.01235, 0.01576, 0.01995, 0.02504,
0.0312, 0.0386, 0.04741, 0.05787, 0.07018, 0.08461, 0.10133, 0.10142, 0.1209,
0.14338, 0.16918, 0.19867};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);


    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //pressure
        double p262[]={.19867, .27028, .36154, .47616, .61823, .79219,
1.0028, 1.2552, 1.5549, 1.9077, 2.3196, 2.7971, 3.3469, 3.9762, 4.6923,
5.5030, 6.4166, 7.4418, 8.5879, 9.8651, 11.284, 12.858, 14.601, 16.529,
18.666, 21.044};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
```

```
    }

    for (i=0;i<len26;i++)
    {
        p26[i]=p26[i]*10e5;
    }


    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
        //printf("\n lsize=%2.2i\n",lsize);

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */


double specific_volume_vapor(double x)
```

```
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of the Specific Volume of the vapor
at
    //%given Temperature x in [m3/kg]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated


    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={205.99, 147.01,   106.3,      77.875,     57.757,
     43.337,32.878,     25.205,     19.515,     15.252,     12.027,
     9.5643,7.6672,     6.1935,     5.0395,     4.1289,     3.4052,
     2.8258,2.3591,     1.9806,     1.6732,     1.6718,     1.4184,
     1.2093,1.0358,     0.89121};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if(x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //specific volume vapor
        double p262[]={0.89121, 0.66800, 0.50845, 0.39245, 0.30678, 0.24259,
0.19384, 0.15636, 0.12721, 0.10429, 0.086092, 0.071503, 0.059705, 0.050083,
0.042173, 0.035621, 0.030153, 0.025555, 0.021660, 0.018335, 0.015471,
0.012979, 0.010781, 0.0088024, 0.0069493, 0.0049544};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }


    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[22], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[22], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
```

```
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
        //printf("\n lsize=%2.2i\n",lsize);

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function


/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double surface_tension(double x)
{
    //                      %------------Saturation Properties of H2O---------
-----------%
    //                      %L is the Lagrangian interpolation of Surface
Tension at given Temperature
    //                      %x in [N/m]
    //                      %x= ;%temperature in [Kelvin] at which the
property is to be evaluated

    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={75.65, 74.94, 74.22, 73.49, 72.74, 71.97, 71.19, 70.4,
69.6, 68.78, 67.94, 67.1, 66.24, 65.37, 64.48, 63.58, 62.67, 61.75, 60.82,
59.87, 58.92, 58.91, 57.94, 56.96, 55.97, 54.97};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);


    if (x>393.15){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //surface tension
```

```
        double p262[]={0.054968*1000, 0.052932*1000, 0.050856*1000,
0.048741*1000, 0.046591*1000, 0.044406*1000, 0.042190*1000, 0.039945*1000,
0.037675*1000, 0.035381*1000, 0.033067*1000, 0.030736*1000, 0.028394*1000,
0.026043*1000, 0.023689*1000, 0.021337*1000, 0.018993*1000, 0.016664*1000,
0.014360*1000, 0.012089*1000, 0.0098644*1000, 0.0077026*1000, 0.0056255*1000,
0.0036654*1000, 0.0018772*1000, 0.00038822*1000};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }



    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
        //printf("\n lsize=%2.2i\n",lsize);

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    L=L/1000;//to convert from mN/m to N/m
```

```
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double thermal_cond_liquid(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of the thermal conductivity of the
liquid
    //%liquid at given Temperature x in [mW/m-K]
    //%x= ;%temperature in [Kelvin] at wich the property is to be evaluated


    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5,    10,   15,   20,   25,   30,   35,   40,   45,
     50,    55,  60, 65, 70,  75,   80,   85,   90,   95,   99.97, 100, 105,
110, 115, 120} ;//vector of temps
    double p26[]={561, 570.5, 580,  589.3,      598.4,      607.2,
     615.5, 623.3,      630.6,      637.3,      643.6,      649.2,
     654.3, 659,  663.1,      666.8,      670,  672.8,      675.3,
     677.3, 679.1,      679.1,      680.5,      681.7,      682.6,
     683.2};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if (x>393.15){
        //thermal conductivity liquid...MAKE SURE TO PUT IN mW/m-K
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        double p262[]={0.68319*1000, 0.68370*1000, 0.68330*1000,
0.68204*1000, 0.67996*1000, 0.67705*1000, 0.67332*1000, 0.66875*1000,
0.66331*1000, 0.65697*1000, 0.64965*1000, 0.64131*1000, 0.63185*1000,
0.62119*1000, 0.60924*1000, 0.59591*1000, 0.58115*1000, 0.56496*1000,
0.54743*1000, 0.52875*1000, 0.50920*1000, 0.48907*1000, 0.46851*1000,
0.44737*1000, 0.42572*1000, 0.42504*1000};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }
    }


    double p[]={p26[0], p26[4], p26[8], p26[12], p26[16], p26[20], p26[len26-
1]};
    double T[]={T26[0], T26[4], T26[8], T26[12], T26[16], T26[20], T26[len26-
1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
```

```
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-T[j])/(T[i]-T[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
        //printf("\n lsize=%2.2i\n",lsize);

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,p,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    L=L/1000;// To convert it from mW/m-K to W/m-K
    return(L);

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



double temperature(double x)
{
    //%------------Saturation Properties of H2O--------------------%
    //%L is the Lagrangian interpolation of Saturation Temperature at given
Pressure
    //%x in [kelvin]
    //%x= ;%pressure in [Pa] at wich the property is to be evaluated



    int len26,lenT,i,j,lsize,flag;
    double L;

    double T26[]={0.01, 5, 10, 15, 20, 25, 30 ,35 ,40 ,45 ,50 ,55 ,60 ,65 ,70
,75 ,80 ,85 ,90, 95 ,99.97 ,100 ,105 ,110 ,115, 120}  ;//vector of temps
```

```
    double p26[]={0.00061, 0.00087 ,0.00123 ,0.00171 ,0.00234 ,0.00317,
0.00425, 0.00563, 0.00738, 0.00959, 0.01235, 0.01576, 0.01995, 0.02504,
0.0312, 0.0386, 0.04741, 0.05787, 0.07018, 0.08461, 0.10133, 0.10142, 0.1209,
0.14338, 0.16918, 0.19867};//vector of property
    len26=sizeof(T26)/sizeof(T26[0]);

    if(x>0.19867e6){
        double T262[]={120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220,
230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370};
        //pressure
        double p262[]={.19867, .27028, .36154, .47616, .61823, .79219,
1.0028, 1.2552, 1.5549, 1.9077, 2.3196, 2.7971, 3.3469, 3.9762, 4.6923,
5.5030, 6.4166, 7.4418, 8.5879, 9.8651, 11.284, 12.858, 14.601, 16.529,
18.666, 21.044};

        for (i=0;i<len26;i++){
            T26[i]=T262[i];
            p26[i]=p262[i];
        }

    }


    for (i=0;i<len26;i++)
    {
        p26[i]=p26[i]*10e5;//to convert from Mpa to Pa
    }


    double p[]={p26[0], p26[14],p26[17],p26[19],p26[21], p26[23],p26[24],
p26[len26-1]};
    double T[]={T26[0], T26[14], T26[17],T26[19],T26[21],T26[23],T26[24],
T26[len26-1]};

    //THE LENGTH OF P AND T SHOULD ALWAYS BE THE SAME!!!!

    lenT=sizeof(T)/sizeof(T[0]);
    for (i=0;i<lenT;i++)
    {
        T[i]=T[i] + 273.15;
    }
    double l[lenT],ll[lenT];

    for (i=0;i<lenT;i++){
        for (j=0;j<lenT;j++){

            if (i==j){
                l[j]=1;
            }
            else
            {
                l[j]= (x-p[j])/(p[i]-p[j]);
            }
            //printf("el=%3.5e\n",l[j]);
        }
        lsize=  (int)(sizeof(l)/sizeof(l[0]));
```

```
        //printf("\n lsize=%2.2i\n",lsize);

        flag=1;
        L=ar_prodsum( l,lsize,flag);
        ll[i]=L;
        //printf("\nll=%3.5e\n",L);

    }
    double ebeprod[lsize];
    //ll=prod(l,2);
    ebe(ll,T,ebeprod,lsize);
    flag=2;//sum up all elements of array
    L=ar_prodsum( ebeprod,lsize,flag);
    return(L);

}//closes function


/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */

double ar_prodsum(double arg[], int arg_length, int flag)

{//%opens function-
    //int arg[] = { 1,2,3,4,5 };
    //int arg_length = sizeof(arg)/sizeof(arg[0])
    double *end = arg+arg_length;
    double sum = 0, prod = 1;
    double *p;

    for (p = arg; p!=end; ++p) {
        sum += *p;
        prod *= *p;
    }

    if (flag<2)
    {
        return(prod);
    }
    else
    {
        return(sum);
    }

}//closes function

/* EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE */



void ebeadd(double a1[], double a2[],double *a3,int alen) /*element by
element addition*/
{
    /*a3 should be declared outside of here...a3 is the output, so make sure
that a3 is an array when using this function.
     //double a3[alen]; */
```

```
    int i;

    for (i=0;i<alen;i++)
    {
        *(a3+i)=a1[i]+a2[i];
    }
}//closes function



void ebe(double a1[], double a2[],double *a3,int alen) //element by element
{
    /*a3 should be declared outside of here...a3 is the output, so make sure
that a3 is an array when using this function.
     //double a3[alen]; */
    int i;

    for (i=0;i<alen;i++)
    {
        *(a3+i)=a1[i]*a2[i];
    }
}//closes function




double pointdist(double a1[], double a2[])
{
    double out;
    /*both arrays should be 1x3, defined by [x,y,z] coordinates */
    out=sqrt( (a1[1]-a2[1])*(a1[1]-a2[1]) +(a1[2]-a2[2])*(a1[2]-a2[2]) +
(a1[3]-a2[3])*(a1[3]-a2[3]) );
    return(out);


}


double DropSF(double phi, double radius)
{//input the ange in radians
    int n;
    double HS_ratio, segment_ratio,HS_top,HS_base, segment_top,
segment_base,phi_term,HS_factor;
    double segment_factor, ShapeFactor,SF;


    //% Eq. 34 (hemispherical droplet) & Eq. 39 (Spherical segment) are
series functions that converge for n > 8
    HS_ratio = 0;
    segment_ratio = 0;

    for (n=0;n<11;n++)//n = 0:1:10
    {
        HS_top = (4.*n+3)*(2.*n+1)*4*factorial(n)*factorial(n)*(pow((-
1),(2.*n)));   //% eq. 34 numerator
```

```
        HS_base = (2.*n+2)*(2.*n+2)*(pow(2,(4.*n)))*(pow((factorial(n)),4));
//% eq. 34 denominator
        HS_ratio = HS_ratio + HS_top/HS_base;
        segment_top = (4.*n+3)*(2.*factorial(n))*(pow((-1),(n))); //% eq. 39
numerator
        segment_base = (2.*n+2)*factorial(n)*factorial(n)*(pow(2,(2.*n)));//%
eq. 39 denominator
        phi_term= pow((tan(phi/2.)),(2.*n+1));
        segment_ratio = segment_ratio + segment_top*phi_term/segment_base;
    }

    HS_factor = 4*HS_ratio;   //% eq. 34 is for the hemispherical drop
Nusselt number
    segment_factor = HS_factor/segment_ratio;   //% eq. 39 is for the
spherical segment drop Nusseltnumber

    ShapeFactor = 0.5*segment_factor*PI*radius*sin(phi);  //% Shape factor =
Nu_drop*Area/Diameter

    SF = ShapeFactor;

    return (SF);
}

long factorial(int n)
{
    int c;
    long result = 1;

    for( c = 1 ; c <= n ; c++ )
        result = result*c;

    return ( result );
}


void mat_mult3x3(double a[][3],double b[][3], double c[][3],int nrows,int
ncol,int mcol)

{
    /*nrows is the number of rows c (the resulting matrix) should have, and
ncol is the number of columns c (the same resulting matrix) should have.*/
    /* mcol is the #of colums in the first matrix which has to match the
number of rows in the second matrix for matrix multiplication to work.  For
example, in a [2x3]*[3x4], mcol=3 */
    int i,j,k;

    for (i=0;i<nrows;i++){
        for (j=0;j<ncol;j++){
            c[i][j]=0;
            for (k=0;k<mcol;k++){
                c[i][j]+=a[i][k] * b[k][j];
            }
        }
    }
```

```
}


double mat_mult_vel(double a[][3],double b[][1], int nrow,int mcol)
{
    /* use nrows=0 for u, nrows=1 for v, nrows=2 for w */
    /* mcol is the #of colums in the first matrix which has to match the
number of rows in the second matrix for matrix multiplication to work.  For
example, in a [2x3]*[3x4], mcol=3 */
    /* b is always an input matrix "randvel" that is a 3x1, therefore j==0 */

    int i,j,k;
    double out;
    i=nrow;
    j=0;
    out=0;


    for (k=0;k<mcol;k++){
        out +=a[i][k] * b[k][j];
    }
    return(out);



}
```

Bibliography

[1] United Nations: Department of Economic and Social Affairs. *Water Scarcity*. URL: `http://www.un.org/waterforlifedecade/scarcity.shtml` (visited on 04/08/2012).

[2] U.S. Department of Energy. *Energy Demands on Water Resources: Report to Congres on the Interdependency of Energy and Water*. Tech. rep. U.S. Department of Energy, 2006.

[3] Dutch Water Sector. *Real water use of the average world citizens is an astonishing 4000 liters per day*. URL: `http://www.dutchwatersector.com/news/news/2012/02/real-water-use-of-the-average-world-citizens-is-an-astonishing-4000-liter-per-day/` (visited on 04/08/2013).

[4] U.S. Department of the Interior: U.S. Geological Survey. *The USGS Water Science School*. USGS - U.S. Geological Survey.

[5] The International Desalination Association (IDA) and The Global Water Intelligence (GWI). *IDA Desalination Yearbook*. Tech. rep. The International Desalination Association (IDA) and The Global Water Intelligence (GWI), 2012-2013.

[6] A Al-Karaghouli and LL Kazmerski. "Renewable Energy Opportunities in Water Desalination". In: *National Renewable Energy Laboratory (NREL), Golden, Colorado* 80401 (2011).

[7] V.P. Carey. *Liquid-Vapor Phase-Change Phenomena*. Second. Taylor & Francis, 2008.

[8] Bharat Bhushan and Yong Chae Jung. "Natural and biomimetic artificial surfaces for superhydrophobicity, self-cleaning, low adhesion, and drag reduction". In: *Progress in Materials Science* 56.1 (2011), pp. 1–108.

[9] A.B. Cassie and S Baxter. "Wettability of porous surfaces". In: *Transactions of the Faraday Society* 40 (1944), pp. 546–551.

[10] C. Dietz et al. "Visualization of droplet departure on a superhydrophobic surface and implications to heat transfer enhancement during dropwise condensation". In: *Applied Physics Letters* 97.3 (2010), pp. 033104–033104.

[11] R.E. Johnson and Robert H. Dettre. "Contact angle hysteresis". In: *Contact angle, wettability, and adhesion. Advances in Chemistry Series* 43 (1964), pp. 112–135.

[12] K.K. Varanasi and T. Deng. "Controlling nucleation and growth of water using hybrid hydrophobic-hydrophilic surfaces". In: *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*. IEEE. 2010, pp. 1–5.

[13] K.K. Varanasi et al. "Spatial control in the heterogeneous nucleation of water". In: *Applied Physics Letters* 95.9 (2009), pp. 094101–094101.

[14] Robert N. Wenzel. "Resistance of solid surfaces to wetting by water". In: *Industrial & Engineering Chemistry* 28.8 (1936), pp. 988–994.

[15] H. Mendoza, S. Beaini, and V.P. Carey. "An Exploration of Transport Within Micro and Nano Droplet Clusters During Dropwise Condensation of Water on Nanostructured Surfaces". In: ASME. 2011.

[16] Gang Chen. *Nanoscale energy transport and conversion: a parallel treatment of electrons, molecules, phonons, and photons*. Oxford University Press, USA, 2005.

[17] Richard C Tolman. "The effect of droplet size on surface tension". In: *The journal of chemical physics* 17 (1949), p. 333.

[18] V.P. Carey. "Surface tension effects on post-nucleation growth of water microdroplets in supersaturated gas mixtures". In: *Journal of heat transfer* 122.2 (2000), pp. 294–302.

[19] Hans R Pruppacher and James D Klett. "Microphysics of clouds and precipitation". In: *Nature* 284.5751 (1980), pp. 88–88.

[20] V.P. Carey. *Statistical Thermodynamics and Microscale Thermophysics*. Cambridge University Press, 1999.

[21] GA Bird. "Direct simulation and the Boltzmann equation". In: *Physics of Fluids* 13 (1970), p. 2676.

[22] GA Bird. "Molecular gas dynamics and the direct simulation monte carlo of gas flows". In: *Clarendon, Oxford* (1994).

[23] GA Bird. "Monte Carlo simulation of gas flows". In: *Annual Review of Fluid Mechanics* 10.1 (1978), pp. 11–31.

[24] GA Bird. "Monte-Carlo simulation in an engineering context". In: *Progress in Astronautics and Aeronautics* 74 (1981), pp. 239–255.

[25] BT Nijaguna. "Drop Nusselt numbers in dropwise condensation". In: *Applied Scientific Research* 29.1 (1974), pp. 226–236.

[26] S.G. Kandlikar. *Handbook of phase change: boiling and condensation*. CRC, 1999.

[27] T. Takeyama and S. Shimizu. "On the transition of dropwise-film condensation". In: *Proc Fifth Int Heat Transfer Conf*. Vol. 3. 1974, pp. 274–290.

[28] PJ Marto et al. "Evaluation of organic coatings for the promotion of dropwise condensation of steam". In: *International journal of heat and mass transfer* 29.8 (1986), pp. 1109–1117.

[29] G. Koch, K. Kraft, and A. Leipertz. "Parameter study on the performance of dropwise condensation". In: *Revue générale de thermique* 37.7 (1998), pp. 539–548.

[30] MH Rausch, AP Fröba, and A Leipertz. "Dropwise condensation heat transfer on ion implanted aluminum surfaces". In: *International Journal of Heat and Mass Transfer* 51.5 (2008), pp. 1061–1070.

[31] Lan Zhong et al. "Effects of surface free energy and nanostructures on dropwise condensation". In: *Chemical Engineering Journal* 156.3 (2010), pp. 546–552.

[32] IC Bang and JH Jeong. "Nanotechnology for advanced nuclear thermal-hydraulics and safety: boiling and condensation". In: *Nuclear Engineering and Technology* 43.3 (2011), pp. 217–242.

[33] Christian Dorrer and Jürgen Rühe. "Condensation and wetting transitions on microstructured ultrahydrophobic surfaces". In: *Langmuir* 23.7 (2007), pp. 3820–3824.

[34] RD Narhe and DA Beysens. "Growth dynamics of water drops on a square-pattern rough hydrophobic surface". In: *Langmuir* 23.12 (2007), pp. 6486–6489.

[35] Zen Yoshimitsu et al. "Effects of surface structure on the hydrophobicity and sliding behavior of water droplets". In: *Langmuir* 18.15 (2002), pp. 5818–5822.

[36] Ryan Enright et al. "Condensation on Superhydrophobic Surfaces: The Role of Local Energy Barriers and Structure Length Scale". In: *Langmuir* 28.40 (2012), pp. 14424–14432.

[37] Kenneth KS Lau et al. "Superhydrophobic carbon nanotube forests". In: *Nano Letters* 3.12 (2003), pp. 1701–1705.

[38] EJ Le Fevre and JW Rose. "A theory of heat transfer by dropwise condensation". In: *Proceedings of the Third International Heat Transfer Conference, Chicago*. Vol. 2. 1966, p. 362.

[39] JW Rose. "Dropwise condensation theory". In: *International Journal of Heat and Mass Transfer* 24.2 (1981), pp. 191–194.

[40] JW Rose. "Some aspects of condensation heat transfer theory". In: *International communications in heat and mass transfer* 15.4 (1988), pp. 449–473.

[41] Jer Ru Maa. "Drop size distribution and heat flux of dropwise condensation". In: *The Chemical Engineering Journal* 16.3 (1978), pp. 171–176.

[42] Hai Wu Wen and Ru Maa Jer. "On the heat transfer in dropwise condensation". In: *The Chemical Engineering Journal* 12.3 (1976), pp. 225–231.

[43] JW Rose and LR Glicksman. "Dropwise condensation-the distribution of drop sizes". In: *International Journal of Heat and Mass Transfer* 16.2 (1973), pp. 411–425.

[44] S. Vemuri and KJ Kim. "An experimental and theoretical study on the concept of dropwise condensation". In: *International journal of heat and mass transfer* 49.3 (2006), pp. 649–657.

[45] Sara S Al-Beaini. "Biomimicry using Nano-Engineered Enhanced Condensing Surfaces for Sustainable Fresh Water Technology". PhD thesis. University of California, Berkeley, 2012.

[46] Mousa Abu-Orabi. "Modeling of heat transfer in dropwise condensation". In: *International journal of heat and mass transfer* 41.1 (1998), pp. 81–87.

[47] R Wilmshurst and JW Rose. "Dropwise condensation-further heat transfer measurements". In: *Proc 4th International Heat Transfer Conference*. 1970.

[48] Nirmal Kumar Battoo et al. "Mathematical modeling and simulation of dropwise condensation and inclined surfaces exposed to vapor flux". In: *Proceedings of the 20th National and 9th International ISHMT-ASME Heat and Mass Transfer Conference*. 2010.

[49] Sunwoo Kim and Kwang J Kim. "Dropwise condensation modeling suitable for superhydrophobic surfaces". In: *Journal of heat transfer* 133.8 (2011).

[50] F.J. Alexander and A.L. Garcia. "The direct simulation Monte Carlo method". In: *Computers in Physics* 11.6 (1997), p. 588.

[51] GA Bird. "Recent advances and current challenges for DSMC". In: *Computers & Mathematics with Applications* 35.1 (1998), pp. 1–14.

[52] D. Baganoff and JD McDonald. "A collision-selection rule for a particle simulation method suited to vector computers". In: *Physics of Fluids A: Fluid Dynamics* 2 (1990), p. 1248.

[53] GA Bird. "Thermal and pressure diffusion effects in high altitude flows". In: *AIAA, Thermophysics, Plasmadynamics and Lasers Conference*. Vol. 1. 1988.

[54] Gerald C Pham-Van-Diep, Daniel A Erwin, and E Phillip Muntz. "Testing continuum descriptions of low-Mach-number shock structures". In: *Journal of Fluid Mechanics* 232.1 (1991), pp. 403–413.

[55] ES Oran, CK Oh, and BZ Cybyk. "Direct Simulation Monte Carlo: Recent Advances and Applications". In: *Annual Review of Fluid Mechanics* 30.1 (1998), pp. 403–441.

[56] Sydney Chapman and Thomas George Cowling. *The mathematical theory of nonuniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge university press, 1991.

[57] GA Bird. "Approach to translational equilibrium in a rigid sphere gas". In: *Physics of Fluids* 6 (1963), p. 1518.

[58] Li Zheng et al. "Development of Homogeneous Water Condensation Models Using Molecular Dynamics". In: *AIAA journal* 47.5 (2009), pp. 1241–1251.

[59] Zheng Li. "Direct Simulation Monte Carlo Modeling of Condensation in Supersonic Plume Expansions of Small Polyatomic Systems". PhD thesis. The Pennsylvania State University, 2009.

[60] V.P. Carey and SM Oyumi. "Condensation growth of single and multiple water microdroplets in supersaturated steam: Molecular simulation predictions". In: *Microscale Thermophysical Engineering* 1.1 (1997), pp. 31–38.

[61] V.P Carey, SM Oyumi, and S. Ahmed. "Post-nucleation growth of water microdroplets in supersaturated gas mixtures: a molecular simulation study". In: *International journal of heat and mass transfer* 40.10 (1997), pp. 2393–2406.

[62] V.P. Carey. "Dsmc modeling of interface curvature effects on near-interface transport". In: *Microscale Thermophysical Engineering* 6.1 (2002), pp. 55–74.

[63] V.P. Carey and S.M. Oyumi. "Molecular Simulation of Dropwise Condensation on an Adiabatic Surface Surrounded by a Supersaturated Air-Stream Mixture". In: *Proceedings of the 1999 ASME/JSME Thermal Engineering Joint Conference, San Diego, CA* AJTE99-6177 (1999).

[64] MJ Haye and C Bruin. "Molecular dynamics study of the curvature correction to the surface tension". In: *The Journal of chemical physics* 100 (1994), p. 556.

[65] VI Kalikmanov. "Semiphenomenological theory of the Tolman length". In: *Physical review E* 55.3 (1997), p. 3068.

[66] MJP Nijmeijer et al. "Molecular dynamics of the surface tension of a drop". In: *The Journal of chemical physics* 96 (1992), p. 565.

[67] A Giesen, A Kowalik, and P Roth. "Iron-atom condensation interpreted by a kinetic model and a nucleation model approach". In: *Phase Transitions* 77.1-2 (2004), pp. 115–129.

[68] S. Cui and R. Zhu et al. *Tolman Effect on Fluid Dynamics in Carbon Nanotubes*. Taylor & Francis Group LLC. 2007.

[69] *Matlab Central: XSteam, thermodynamic properties of water and steam*. June 2007. URL: `http://www.mathworks.com/matlabcentral/fileexchange/9817-http://www.mathworks.com/matlabcentral/fileexchange/9817-x-steam-thermodynamic-properties-of-water-and-steam` (visited on 04/15/2011).

[70] I. Langmuir. "The Dissociation of Hydrogen into Atoms: Calculation of the Degree of Dissociation and the Heat of Formation (Part II)". In: *Journal of the American Chemical Society* 37.3 (1915), pp. 417–458.

[71] B. Paul. "Compilation of evaporation coefficients". In: *ARS Journal* (1962).

[72] A.F. Mills. *The condensation of steam at low pressures*. Tech. rep. NSF GP-2520. Space Sciences Laboratory, University of California, Berkeley, 1965.

[73] The National Institute of Standards and Technology: U.S. Department of Commerce. *Thermal physical properties of fluid systems*. URL: `http://webbook.nist.gov/chemistry/fluid/` (visited on 05/15/2012).

[74] JK Harvey. "Direct simulation Monte Carlo method and comparison with experiment". In: *Progress in Astronautics and Aeronautics: Thermophysical Aspects of Re-Entry Flows* (1986), pp. 25–43.

[75] M.H. Kalos and P.A. Whitlock. *Monte carlo methods*. Wiley-VCH, 2009.

[76] X. Ma et al. "Dropwise condensation heat transfer of steam on a polytethefluoroethylene film". In: *Journal of Thermal Science* 10.3 (2001), pp. 247–253.

[77] C. Bum-Jin et al. "Experimental comparison of film-wise and drop-wise condensations of steam on vertical flat plates with the presence of air". In: *International communications in heat and mass transfer* 31.8 (2004), pp. 1067–1074.

[78] Yang-Tse Cheng and Daniel E Rodak. "Is the lotus leaf superhydrophobic?" In: *Applied physics letters* 86.14 (2005), pp. 144101–144101.

[79] Kevin A Wier and Thomas J McCarthy. "Condensation on ultrahydrophobic surfaces and its effect on droplet mobility: ultrahydrophobic surfaces are not always water repellant". In: *Langmuir* 22.6 (2006), pp. 2433–2436.

[80] C.H. Chen et al. "Dropwise condensation on superhydrophobic surfaces with two-tier roughness". In: *Applied physics letters* 90.17 (2007), pp. 173108–173108.

[81] R.W. Bonner III. "Dropwise Condensation on Surfaces With Graded Hydrophobicity". In: ASME. 2009.

[82] Nenad Miljkovic, Ryan Enright, and Evelyn N Wang. "Effect of droplet morphology on growth dynamics and heat transfer during condensation on superhydrophobic nanostructured surfaces". In: *ACS nano* 6.2 (2012), pp. 1776–1785.

[83] N Miljkovic, R Enright, and EN Wang. "Growth Dynamics During Dropwise Condensation on Nanostructured Superhydrophobic Surfaces". In: *3rd Micro/Nanoscale Heat & Mass Transfer International Conference, Atlanta, GA*. 2012.