การจำลองแบบเชิงตัวเลขสำหรับการเคลื่อนไหวของพลาสมามิติเดียวในสนามแม่เหล็กไฟฟ้า

นาย ชวิศนัช อิงชาติเจริญ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชานิวเคลียร์เทคโนโลยี ภาควิชานิวเคลียร์เทคโนโลยี

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2543

ISBN 974-13-0274-6

NUMERICAL SIMULATION FOR MOTION OF ONE DIMENSIONAL PLASMA IN AN ELECTROMAGNETIC FIELD

MR CHAWISNACH ENGCHATCHAROEN

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Nuclear Technology
Department of Nuclear Technology
Faculty of Engineering
Chulalongkorn University
Academic Year 2000
ISBN 974-13-0274-6

Thesis Title	Numerical simulation for motion of one dimensional plasma in an electromagnetic field.
By	Mr. Chawisnach Engchatcharoen
Field of Study	Nuclear Technology
Thesis Advisor	Dr. Sunchai Nilsuwankosit

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

…………..…………………..……. Dean of Faculty of Engineering

(Professor Somsak Punyakaew, D.Eng)

THESIS COMMITTEE

….…………………………….... Chairman

(Associate Professor Somyot Srisatit)

……………………………… Thesis Advisor

(Dr. Sunchai Nilsuwankosit)

……………………………… Member

(Associate Professor Nares Chankow)

……………………………… Member

(Assistant Professor Dr. Supitcha Chanyotha)

ชวิศนัช อิงชาติเจริญ : การจำลองแบบเชิงตัวเลขสำหรับการเคลื่อนไหวของพลาสมามิติเดียวในสนามแม่เหล็กไฟฟ้า. (NUMERICAL SIMULATION FOR MOTION OF ONE DIMENSIONAL PLASMA IN AN ELECTROMAGNETIC FIELD) อ. ที่ปรึกษา : อ.ดร. สัญชัย นิลสุวรรณโฆษิต, 86 หน้า. ISBN 974-13-0274-6

วัตถุประสงค์หลักของวิทยานิพนธ์คือ เพื่อพัฒนาแบบจำลองทางคอมพิวเตอร์ของพลาสมาแบบมิติเดียวบนเครื่องคอมพิวเตอร์ส่วนบุคคล พลาสมาคือสถานะที่สี่ของสสารซึ่งสสารในระบบสุริยะจักรวาลส่วนใหญ่อยู่ในสถานะของพลาสมา พลาสมามีประโยชน์หลายด้านโดยเฉพาะอย่างยิ่งในด้านนิวเคลียร์ฟิวชันในแง่ของแหล่งกำเนิดพลังงาน ดังนั้นการศึกษาพลาสมาจึงเป็นพื้นฐานที่สำคัญในการทำความเข้าใจและศึกษาขบวนการ การเกิดฟิวส์ชันแต่เนื่องจากการทำการทดลองกับระบบจริง ต้องใช้งบประมาณที่สูง ก่อให้เกิดการสิ้นเปลือง ดังนั้นการสร้างแบบจำลองในทางคอมพิวเตอร์ขึ้นก่อน จึงน่าจะเป็นทางเลือกหนึ่งที่เหมาะสม

ในเบื้องต้นนี้จะทำการสร้างแบบจำลองพลาสมาในระบบหนึ่งมิติ ภาษาคอมพิวเตอร์ที่ใช้ในการพัฒนาแบบจำลองคือ จาวา โปรแกรมที่พัฒนา ใช้วิธีทางการประมาณเชิงตัวเลข ผลต่างสืบเนื่อง มีขั้นตอนหลักอยู่สองขั้นตอน คือ ขั้นตอนแรก หาค่าสนามไฟฟ้า และสนามแม่เหล็ก ด้วยสมการแม็กซ์เวลล์ และขั้นตอนที่สอง หาการเคลื่อนที่ของอนุภาคด้วย สมการการเคลื่อนที่ของลอเร็นซ์ การจำลองการเคลื่อนไหวของพลาสมาถูกกระทำภายใต้เงื่อนไขโดยกำหนดศักย์ไฟฟ้าค่าศูนย์ที่ขอบทั้งสองของระบบ ซึ่งผลที่ได้ ความยาวเดอร์บาย และความถี่ของพลาสมา จะถูกตรวจสอบและเปรียบเทียบกับค่าทางทฤษฎี ความแตกต่างที่เกิดขึ้นเนื่องจากความไม่แม่นยำของการประมาณตลอดจนความผิดพลาดซึ่งเกิดขึ้นเองจากเทคนิคการคำนวณที่ใช้โดยอาศัยการประมาลผลเหล่านี้ จึงสรุปได้ว่ารูปแบบจำลองที่ใช้สามารถจำลองลักษณะการเคลื่อนไหวของพลาสมาแบบมิติเดียวได้ในระดับหนึ่งสำหรับทรัพยากรการคำนวณที่มีจำกัด

ภาควิชา……นิวเคลียร์เทคโนโลยี…….....	ลายมือชื่อนิสิต………..
สาขาวิชา......นิวเคลียร์เทคโนโลยี…......…	ลายมือชื่ออาจารย์ที่ปรึกษา…………............................
ปีการศึกษา 2543	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม................................

4070246721 : MAJOR NUCLEAR TECHNOLOGY

KEY WORD: PLASMA / PLASMA SIMULATION / NUCLEAR / FUSION / SIMULATION

CHAWISNACH ENGCHATCHAREON : NUMERICAL SIMULATION FOR MOTION OF ONE DIMENSIONAL PLASMA IN AN ELECTRO-MAGNETIC FIELD. THESIS ADVISOR : SUNCHAI NILSUWANKOSIT, Ph.D. , 86 pp. ISBN 974-13-0274-6

The purposes of this research are to develop a computer model to simulate the motion of the one dimensional plasma on microcomputer. Plasma is the fourth state of matter. Most of matter in the universe is in the plasma state. Plasma plays the important role in many applications. One of the most challenging is to develop the controlled thermonuclear fusion as an economical energy source. In order to achieve these conditions, an understanding of how the plasma behaves is essential. Because of some technical and economical difficulties, the experimental method may be impossible or difficult. Therefore, the computer simulation technique is an alternative.

We restrict the scope on one dimensional plasma. The model is developed with JAVA language with calculating Finite difference technique. The study consists of two main parts; the fields produced by particles which are solved by Maxwell's equations and the motion of the particles caused by the forces that resulted from the fields which are computed based on Lorentz equation. The motion of a one dimensional plasma is analyzed and simulated under zero potential that is applied at the boundary condition. The characteristic time and characteristic size are estimated to compare with the theoretical value. The results show the discrepancy which might have been due to the lack of accuracy of estimation and the inherent error caused by the technique used for the simulation. Based on these assessments, it is concluded that the model as proposed is capable to simulate the motion of the one dimensional plasma up to a level of certainty and with the limited computing resources.

Department…Nuclear Technology…… Student's signature…………………………..

Field of study…Nuclear Technology… Advisor's signature…………………………..

Academic year…2000……………..… Co-advisor's signature..…..……………………..

Acknowledgment

I would like to thank all the great people at the department of Nuclear Technology at Chulalongkorn University, especially Assistant Professor Attaporn Pattarasumunt for the help and advice. Specifically, I thank Dr. Sunchai Nilsuwankosit, my thesis advisor, who gave me the opportunity to do this thesis and who encouraged me every step of the way to ensure its success.

On a personal note, thanks to all of my friends who encouraged and prayed for me during the writing of this thesis. I would like to sincerely thank Dr. Kevin Bowers at Plasma Theory and Simulation Group, Electrical Engineering and Computer Science Department at University of California, Berkeley for the help and more useful suggestions. I also appreciate Dr. Vinai Pruksawan, a former advisor at Electrical Engineering, Kasetsart University, for useful guidance. Most importantly and finally, I thank my parents who highly supported, encouraged and truly impacted my personal life.

Contents

Contents(Cont.)

Figure contents

Figure contents (Cont.)

CHAPTER I

INTRODUCTION

1.1 Background

Plasma is the fourth state of matter. It is most similar to gas but is fundamentally different in that gas is made up of molecules and is electrically neutral while plasma is a collection of the free moving electrons and ions. Plasma can be made in a number of ways but the end result is always a combination of the positively charged ions (atoms with one or more missing electrons) and the negatively charged electrons. The movement of charged particles in plasma generates current and charge density gradient which in turn give rise to electric and magnetic field which affect to the motion of other charged particles over a large distance. This characteristic is called the exhibition of collective behavior. Plasma is not only electrically charged it is an efficient electrical conductor. The state of plasma can exist anywhere from inside the room temperature metal to the star. Plasma must conform these criteria, the size of system length must be larger than the Debye length, the number of particles are enough in a Debye length and the multiplication of the mean time between collision with neutral atoms and the frequency of plasma are greater than unity. At low density, plasma has collisionless behavior which means the collective behavior is much more important in determining the characteristic of plasma than local collision which can be neglected.

In order to understand the behavior of plasma, it is necessary to study its behaviors. Because of some technical and economical difficulties, the experimental method may be impossible or very difficult. Therefore, the simulation method is an alternative. Because plasma can be used in the innumerable applications, this mean to control the behavior of plasma is needed. As plasma is conductive and respond to the electric and magnetic fields therefore plasma can be accelerated and steered by the electric and magnetic fields. Plasma research also yields a greater understanding of the universe. It also provides many practical uses in new manufacturing techniques, the consumer products, and the prospect of abundant energy.

One of the most challenging applications of plasma is to develop the controlled thermonuclear fusion as an economical energy source. Fusion is a process that combines the atomic nuclei of light elements, like the isotopes of hydrogen called deuterium and tritium, to form heavier elements. There are many advantages of using fusion energy such as abundant and inexpensive fusion fuel, short life of radioactivity less than 100 years. The key to fusion is to heat hydrogen to an extreme temperature, which strip electrons from the nuclei of the atoms. The resultant mix of the positive (the nuclei) and negative (the electrons) particles creates an ionized gas called plasma. Importantly, the fact that the particles in plasma have these positive and negative charges allows them to be confined by magnetic fields in a fusion reactor.

1.2 Confinement schemes

In order to control the fusion process, two commonly forms of plasma confinements can be implemented.

Magnetic confinement

The magnetic field acts like a net to the charged particles of plasma preventing the plasma from leaking out. In deed, the very best way to trap the plasma is to form magnetic field like a ball but the magnetic field in the form donut shape called torus is made in stead. This method uses the fact that plasma is in the conducting state, that it contains charged particles (ions and electrons), it can be acted up on by a magnetic field so that it is shaped and confined. If you arrange the magnetic field carefully, the particles will be trapped by it. There are several major schemes to confine plasma with magnetic field as following.

Tokamak

One of the greatest innovations for fusion science was the Tokamak concept which was invented in the Soviet Union in 1950s by the Russians Tamm and

Sakharov. The word “Tokamak” is contraction of Russian word: toroidalnaya, kamera and magnitnaya these mean toroidal chamber magnetic. The Tokamak is a device which employs magnetic fields in a toroidal configuration to confine the plasma. The magnetic field in a tokamak are produced by a combination of the currents flowing within the plasma itself and the current flowing in external coils. Although the tokamak has excellent confinement properties, it still has the disadvantage that the current in the plasma tends to make it unstable. This can lead to a disruption where the plasma dramatically crashed against the inside of container. In currently, there are several major tokamak facilities throughout the world such as the Joint European Tokamak (JET) in England, JT-60 in Japan, International Thermonuclear Experimental Reactor (ITER). Much of the knowledge of plasma science gained in Tokamak experiments is directly applicable to alternate fusion systems. The present generation of these experiments, therefore, are effective tools for advancing the foundation of science and technology required for developing a commercially viable fusion power plant.

<u>Stellarator</u>

Stellarator have a magnetic topology similar to that of tokamak. The traditional stellarator consist of a set of continuous helical coils that create the twisted magnetic field needed to confine plasma. The advantage of this type of confinement scheme is the magnetic field can be created independently of plasma therefore, plasma does not influence the magnetic field. This method of trapping plasma is more stable than tokamak. However, the disadvantage is helical ripple created by the helical field coils which does not confine plasma efficiently. This lead to Quatos (Quasi Toroidal Stellarator) which is unique stellarator design that fool the plasma into thinking there is no helical ripple as conventional stellarator. Quatos incorporates the advantage of tokamak with its good plasma confinement properties and the advantage of stellarator with its stability.

<u>Alternative magnetic confinement schemes</u>

There are several alternative magnetic confinement schemes. Reverse Field Pinch looks similar to tokamak with a lot current in plasma which creates a strong poloidal magnetic field. The toroidal field is about ten times smaller thus, the field line make many poloidal circuits per toroidal circuit. The smaller field yields potential cost saving reactor advantage for instance, in copper magnets. Speromak is another idea is similar to reverse field pinch but in spherical containment vessel rather than a torus. This also eliminates any conductors through the hole in the doughnut.

Inertial confinement

No magnetic field are needed in this scheme. This technique contains the gaseous mixture within a small pellet or bead then have it bombarded from all directions by the high power laser beams. If the process is done quickly and compress it enough the subsequent shock wave causes the pellet to implode. The inertia then holds it together long enough for fusion reaction. This method is known as Inertial Confinement Fusion (ICF). It offers a different approach to developing the fusion energy. To achieve ICF, the powerful lasers or particle beams are focused on a small target of hydrogen fuel for a few billionths of a second. The target is compressed to the density 1000 times the normal density of a solid material and heated to the temperature of about 100 million degrees. In this condition, the hydrogen nuclei fuse to form helium, and release a significant amount of energy. The process is the same as that in the sun, except that a laser or particle beam heats and compresses the hydrogen fuel rather than the sun's gravity. The primary mission of ICF is to carry out the experiment on the nuclear weapon physics and its effects, under the extreme condition of density and temperature created in the inertial fusion. The major problem is getting the lasers to hit the pellet appropriately so that it compresses symmetrically rather than one side.

1.3 The method for heating plasma

There are several methods for heating plasmas. These include Ohmic Heating, Neutral Beam Injection, Magnetic Compression, Radio-Frequency Heating, and Inertial Compression. Each of these is discussed below.

1: Ohmic Heating

Since the plasma is an electrical conductor, it is possible to heat the plasma by passing a current through it; in deed, the current that generates the poloidal field also heats the plasma. This is called ohmic (or resistive) heating, it is the same kind of heating that occurs in an electric light bulb or in an electric heater. The heat generated depends on the resistance of the plasma and the current.

2: Neutral-Beam Injection

Neutral-beam injection involves the introduction of high-energy (neutral) atoms into the ohmically heated, magnetically confined plasma. The atoms are immediately ionized and are trapped by the magnetic field. The high-energy ions then transfer part of their energy to the plasma particles in repeated collisions, thus increasing the plasma temperature.

3: Magnetic Compression

A gas can be heated by sudden compression. In the same way, the temperature of a plasma is increased if it is compressed rapidly by increasing the confining magnetic field. In a tokamak system this compression is achieved simply by moving the plasma into a region of higher magnetic field (i.e. radially inward). Since plasma compression brings the ions closer together, the process has an additional benefit of facilitating attainment of the required density for a fusion reactor.

4: Radio Frequency Heating

In radio frequency heating, high frequency waves are generated by oscillators outside the torus. If the waves have a particular frequency (or wavelength), their energy can be transferred to the charged particles in the plasma, which in turn collide with other plasma particles, thus increasing the temperature of the bulk plasma. This process is similar to how a microwave oven heats food.

5: Inertial Compression

In the inertial approach the compression is achieved by using laser or particle beams to heat the outer layer of a target pellet; the outer layer vaporizes and the pressure that the vaporized layer exerts back on the core of the pellet accelerates the plasma inward on itself, and the inertia of the imploding atoms in the pellet allows the pellet to be compressed (for a very short time), and thus heated.

1.4 Plasma simulation

Computer simulation plays an important role in the development of plasma theory. In order to provide prediction in plasma, there are a large variety of models for simulating plasma but, basically, there are two major types;

Particle Model

This involves following the motion of a large number of charged particles in their self-consistent electric and magnetic fields. It is limited to only simulate the phenomenon in which only a small fraction of plasma is involved and only for a short period of time.

Fluid Model

This adopts a set of the fluid equations to describe plasma by solving numerically the magnetohydrodynamic (MHD) equation. The method is possible because of the fact that plasma behaves sometimes like a fluid and sometimes like a collection of individual particles. In the fluid approximation , plasma is considered to compose of two or more interpenetrating fluids. By treating plasma as a magnetized conducting fluid, this method can be applied to the large scale problems.

In some plasma problem, neither fluid theory nor kinetic theory is sufficient to describe the plasma's behavior. The alternate model is Hybrid model by treating the electrons as a fluid and ions are represented by macro particles.

1.5 One dimensional plasma simulation

This thesis uses the particle model to simulate one dimensional plasma. The calculation also employs the finite difference technique. The procedure comprises of two main parts, the field produced by particles according to Maxwell's equations and the motion produced by forces that use Lorentz equation of motion. This process is repeated over many time steps.

1.6 Objectives

1.5.1 To simulate one dimensional plasma on a microcomputer in order to investigate the motion and the primary characteristics of plasma.
1.5.2 To develop the technique to analyze and simulate the motion of plasma.

1.7 Scope of thesis

1.6.1 Develop the code based on solving Maxwell's equations together and Lorentz's equation of motion.
1.6.2 Obtain the fundamental characteristics of one dimensional plasma.

1.8 Methodology

1.5.1 Study the theory of plasma and relating subjects.
1.5.2 Choose the appropriate numerical technique to develop the simulation code.
1.5.3 Program and debug the code for simulation.

1.5.4 Analyze and compare the obtained results with the theoretical data.

1.5.5 Conclude the research and write the thesis.

1.9 Potential application

1.6.1 Used for initial exploration of plasma simulations.

1.6.2 The simulation of one dimensional plasma can be used to test the numerical techniques against the theoretical predictions.

1.6.3 To further develop two and three dimensional model to describe the complete characteristic of plasma.

1.10 Relative researches

- Charles K. Birdsall and A.Bruce Langdon developed the plasma simulation using particles model in 1985, their works originated as a set of class notes intended for use by the graduate students at University of California, Berkeley. To simulate a laboratory plasma in one dimension with the numerical methods of the fast Fourier transform (FFT) and written in Fortran language. The results from their works are enough to explain the background behaviors of the plasma.

- In 1996, Kevin Bowers at Plasma Theory and Simulation Group, Electrical Engineering and Computer Science Department at University of California, Berkeley developed the two dimensional plasma simulation using the Monte-Carlo of the numerical technique. The simulating code employed a hybrid of C and MATLAB. It is optimized for modern processors and shared memory operation. The code can run on any platform that has MATLAB V5+ and an ANSI C. His work is used to simulate physics of plasma.

- In 1998, Z.Lin, T.S. Hahm, W.W.Lee, W.M.Tang andR.B.White at department of energy's Princeton Plasma Physics Laboratory used the power of the SGI/CrayT3E supercomputer to create the three-dimensional nonlinear particle simulation of microturbulence in the plasma. The simulation involved the number of particle as large as 400 million plasma particles. The results were achieved with the ability of the massively parallel processor (MPP). The research discovered the suppression of the turbulence. This helps enhancing the administration of the plasma confinement.

- Dr. Chin S. Lin at Aurora Science Inc. developed the plasma simulation codes on the parallel computers. The codes are useful for investigating kinetic plasma processes associated with the global space plasma phenomena. A key element of developing the PIC(Particle in cell) codes on the parallel computers is the gather-scatter scheme. The gather-scatter scheme is used to communicate between particle quantities and the grid quantities. This project created a PIC code and a particle trajectory code on Thinking Machines Corp. CM-2 with 32,768 processors.

CHAPTER II

THEORY OF ONE DIMENSIONAL PLASMA

2.1 Overview

As we shall see, every plasma can be completely described by five equations: the four Maxwell's equations and the Lorentz Force. Maxwell's equations describe the electric and magnetic field over time and space.

The first equation of Maxwell's equations is Poisson's Law describes the electric field produced by the distribution of the electric charges. The second law is the Faraday's Law of induction , which states the relationship between the fluctuating magnetic field and the electric field it induces. The Ampere's Law is the third law that describes the magnetic field induced by the current density and the electric displacement . The last law has no name. It states that there is no magnetic monopole. In other words, all magnetic field lines must be closed. The Lorentz equation explain the total force on a particle due to its interaction with the electric and magnetic field.

2.2 Maxwell's equations

- Maxwell equation

In vacuum ;

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0} \qquad \text{(Poisson's equation)} \qquad (2.1)$$

$$\nabla \cdot \mathbf{B} = 0 \qquad \text{(Absence of free magnetic poles)} \qquad (2.2)$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \qquad \text{(Faraday's law)} \qquad (2.3)$$

$$\nabla \times \mathbf{B} = \mu_0 \left(\mathbf{j} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right) \qquad \text{(Ampere's law)} \qquad (2.4)$$

In a medium ;

$$\nabla \cdot \mathbf{D} = \rho \tag{2.5}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{2.6}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.7}$$

$$\nabla \times \mathbf{H} = J + \frac{\partial \mathbf{D}}{\partial t} \tag{2.8}$$

where

$$\mathbf{D} = \varepsilon_0 \mathbf{E} + \mathbf{P}$$

$$\mathbf{H} = \frac{1}{\mu_0} \mathbf{B} - \mathbf{M}$$

For linear and isotropic media

$$\mathbf{D} = \varepsilon \mathbf{E}$$

$$\mathbf{B} = \mu \mathbf{H}$$

- Lorentz force equation

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{2.9}$$

These field variables are defined as follow :

E = electric field (V/m)

H = magnetic field (A/m)

D = electric displacement or electric flux density (C/m^2)

B = magnetic induction or magnetic flux density (Wb/m^2)

ρ = charge density (C/m^3)

J = current density (A/m^2)

P = polarization (C/m^2)

M = magnetization (A/m)

ε = permittivity of the medium (F/m)

ε_0 = permittivity of free space or vacuum (F/m)

μ_0 = permeability of free space (H/m)

2.3 Debye Length

Because of its collective behavior, a plasma tend to resist to the external influence and thus, is able to shield out the electric potential that is applied to it as seen in Figure 2.1. The shielding distance or thickness of the sheath that screens the external field is called the Debye length (λ_D) and is equal to

$$\lambda_D = \left(\frac{\varepsilon_0 k_B T}{n_0 q_e^{2}}\right)^{\frac{1}{2}} \qquad (2.10)$$

where q_e is the electrical charge of an electron (C),

n_o is the equilibrium particle number density (m^{-3}),

k_B is Boltzman's constant (J/K),

T is temperature (K).

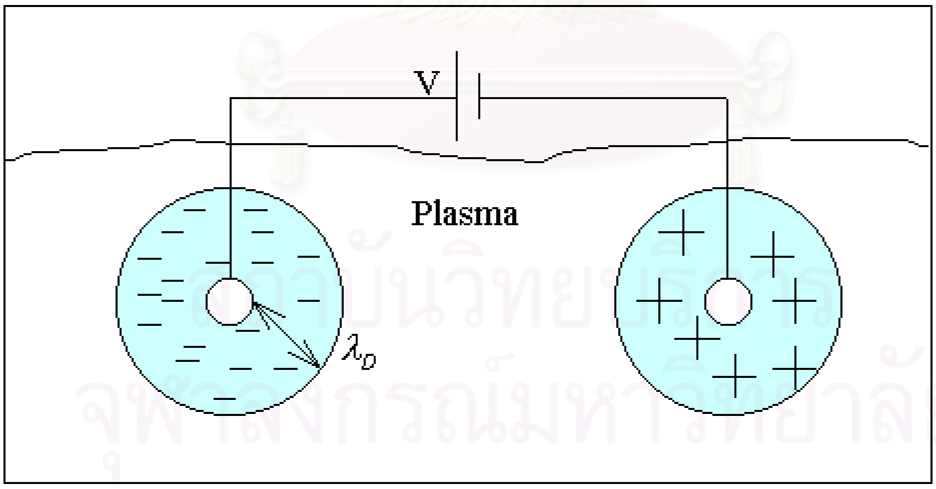


Figure 2.1 Debye length

The calculation of λ_D can be done in the following steps. When an electric field is applied to a plasma, the field only affects the distribution of the charged particle in the plasma within the distance of Debye length (λ_D). Within this distance of λ_D , the densities of the ions and the electrons can be found from the Boltzman distribution (each species can be in its own distribution).

That is $n_i = n_0 e^{\left(\frac{-q_e\phi}{k_B T}\right)}$ and $n_e = n_0 e^{\left(\frac{q_e\phi}{k_B T}\right)}$

n_i and n_e are the density of ions and electrons respectively.

ϕ is the actual electric field .

Therefore , $$\rho = q_e\,(n_0\, e^{\frac{-q_e\phi}{k_B T}} - n_0 e^{\frac{q_e\phi}{k_B T}}) \qquad (2.11)$$

from $\sinh x = \dfrac{e^x - e^{-x}}{2}$

thus $$\rho = q_e\,(n_0\, e^{\frac{-q_e\phi}{k_B T}} - n_0 e^{\frac{q_e\phi}{k_B T}})\frac{2}{2} \qquad (2.12)$$

become $$\rho = 2n_0 q_e \sinh\left(\frac{-q_e\phi}{k_B T}\right) \qquad (2.13)$$

and $$\rho = -\,2n_0 q_e \sinh\left(\frac{q_e\phi}{k_B T}\right) . \qquad (2.14)$$

From this result, we substitute it into Poisson's equation to get

$$\nabla^2\phi = \frac{-\rho}{\varepsilon_0} = \frac{1}{\varepsilon_0} 2n_0 q_e \sinh\left(\frac{q_e\phi}{k_B T}\right) . \qquad (2.15)$$

Assume $q_e\,\phi \langle\langle k_B T$ then $\sinh\left(\frac{q_e\phi}{k_B T}\right) \approx \frac{q_e\phi}{k_B T}$

and $$\nabla^2\phi = \frac{2n_0 q_e}{\varepsilon_0}\,\frac{q_e\phi}{k_B T} , \qquad (2.16)$$

$$\nabla^2\phi = 2\phi\left[\left(\frac{n_0 q_e{}^2}{\varepsilon_0 k_B T}\right)^{\frac{1}{2}}\right]^2 . \qquad (2.17)$$

That $$\nabla^2\phi = \frac{2\phi}{\lambda_D^2} \qquad (2.18)$$

where $\lambda_D = \left(\frac{\varepsilon_0 k_B T}{n_0 q_e^2}\right)^{\frac{1}{2}}$ and is the characteristic size for a plasma .

The screening of the applied electric field as discussed so far has a physical meaning only if a large number of particles, N_D , are contained within a volume that has the Debye length as its characteristic length. Assume that N_D particles are contained within a sphere of radius λ_D , it is followed that

$$N_D = \frac{4\pi}{3}\lambda_D^3 n_0 \gg 1 \tag{2.19}$$

With the value of λ_D as derived, we then have

$$\frac{4\pi}{3} n_0 \left(\frac{\varepsilon_0 k_B T}{n_0 q_e^2}\right)^{\frac{3}{2}} \gg 1 \tag{2.20}$$

or

$$\frac{4\pi}{3}\left(\frac{\varepsilon_0 k_B T}{q_e^2}\right)^{\frac{3}{2}} \gg n_0^{\frac{1}{2}} . \tag{2.21}$$

This value of n_0 is another requirement for a given matter to be classified as a plasma.

2.4 Plasma frequency

The particles in a plasma respond to an electric field by adjusting their positions. The effect of this movement is to set up a local electric field within a plasma that counteracts to the applied field. As a result, the energy shifts from the electric potential to the kinetic energy and back. The frequency in which this oscillation occurs is known as a plasma frequency (ω_p) This frequency characterizes the period in which plasma responses to an external field and is described as

$$\omega_p = \left(\frac{n_0 q_e^{\ 2}}{m\varepsilon_0}\right)^{\frac{1}{2}} \tag{2.22}$$

where m is the mass of an electron.

The above definition is calculated based on the following assumption. Consider in a plasma, as the electron is much less massive than the ion, the electrons oscillate so fast around their equilibrium positions that the massive ions may be considered as being relatively stationary. If the electrons are moved to a certain distance x , this will give rise to electric field that, according to Poison's equation is described as

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0} \quad .$$

If the displacement is only in x direction thus, electric field $\mathbf{E} = E\mathbf{e}_x$ then

$$\frac{dE}{dx} = \frac{q_e}{\varepsilon_0}(n_i - n_e) \quad . \tag{2.23}$$

Thus

$$\int dE = \frac{q_e}{\varepsilon_0} n_0 \int dx \tag{2.24}$$

$$E = \frac{q_e n_0 x}{\varepsilon_0} \tag{2.25}$$

where n_0 = mean density of charged particle in plasma .

Given that the equation of motion of an electron in an electric field is

$$m\ddot{x} = -q_e E \tag{2.26}$$

or

$$m\ddot{x} = -q_e\left(\frac{q_e n_0 x}{\varepsilon_0}\right) \tag{2.27}$$

then

$$\ddot{x} = -\left(\frac{n_0 q_e^{\ 2}}{m\varepsilon_0}\right)x \tag{2.28}$$

and $$\omega_p = \left(\frac{n_0 q_e^2}{m\varepsilon_0}\right)^{\frac{1}{2}} \tag{2.29}$$

$$\ddot{x} = -\omega_p^2 x \quad . \tag{2.30}$$

The solution of the above equation is

$$x = x_1 \sin \omega_p t + x_2 \cos \omega_p t \ . \tag{2.31}$$

This solution simply states in that the electrons in the plasma oscillate with the frequency of ω_p .

<u>Cyclotron frequency</u>

When a charged particle moves in a uniform magnetic field which does not vary in time, in the absence of an electric field, it circulates around an axis called guiding center with a radius r_L which is known as Larmor radius and with the frequency(ω_c) where

$$\omega_c = \frac{qB}{m} \tag{2.32}$$

and $$r_L = \frac{mv}{qB} = \frac{v}{\omega_c} \quad . \tag{2.33}$$

The value of ω_c and r_L can proved by considering the following;

From Lorentz force equation $$m\dot{\mathbf{v}} = q\mathbf{v} \times \mathbf{B} \tag{2.34}$$

given $\mathbf{B} = B\mathbf{e_Z}$, $$m\dot{v}_z = 0 \tag{2.35}$$

and $$m\dot{v}_x = eBv_y \tag{2.36}$$

Thus $$m\ddot{v}_x = eB\dot{v}_y \tag{2.37}$$

or $$\ddot{v}_x = \frac{eB}{m}\left(\frac{-eBv_x}{m}\right) \quad . \tag{2.38}$$

That is $$\ddot{v}_x = -\left(\frac{eB}{m}\right)^2 v_x \qquad (2.39)$$

Similarly, we have $$\ddot{v}_y = -\left(\frac{eB}{m}\right)^2 v_y \qquad (2.40)$$

As we define $\omega_c = \dfrac{eB}{m}$, therefore $\ddot{v} = \omega_c^2 v$. The solution of this equation is $\mathbf{v} = \mathbf{v}_\perp e^{i\omega_c t} = \dot{\mathbf{x}}$ where v = speed in the perpendicular plan to **B**.

Therefore $$m\dot{v}_x = eBv_y \qquad (2.41)$$

and $$v_y = \frac{m\dot{v}_x}{eB} \qquad (2.42)$$

$$v_y = \frac{\dot{v}_x}{\omega_c} \qquad (2.43)$$

$$v_y = \frac{\left(v_\perp \dot{e}^{i\omega_c t}\right)}{\omega_c} i^2 \qquad (2.44)$$

$$v_y = \frac{v_\perp i\omega_c e^{i\omega_c t}}{\omega_c} \qquad (2.45)$$

$$v_y = iv_\perp e^{i\omega_c t} \qquad (2.46)$$

$$\frac{dy}{dt} = iv_\perp e^{i\omega_c t} \qquad (2.47)$$

$$\int dy = \frac{v_\perp}{\omega_c}\int e^{i\omega_c t}\, di\,\omega_c t \qquad (2.48)$$

Integrating the above equation gives

$$y - y_0 = \frac{v_\perp}{\omega_c} e^{i\omega_c t} \qquad (2.49)$$

and $$x - x_0 = -\frac{v_\perp}{\omega_c} i e^{i\omega_c t}. \qquad (2.50)$$

The real part of these equations give ,

$$y - y_0 = r_L \cos \omega_c t \qquad (2.51)$$

and $$x - x_0 = r_L \sin \omega_c t \qquad (2.52)$$

From the above equations , it is immediately seen that a plasma particle moves on an perpendicular plan to the constant magnetic field along a circular orbit with a guiding center at (x_0 , y_0).

CHAPTER III

NUMERICAL MODEL FOR ONE DIMENSIONAL PLASMA

3.1 Overview

The basic technique for creating the model for the simulation is to choose the appropriate numerical method to describe the physical system. For this thesis, the finite difference technique is chosen . Once the finite difference scheme is set up, each plasma particle is given the initial position and velocity. With these initial conditions, the distribution of the charge densities in system can be calculated. Once these quantities have been determined, Maxwell's equations are solved to find the electric and the magnetic field. After the fields are known, the Lorentz Force equation is then used to find the force on each particle. We can then push each of the particles in the plasma one time step based on the force it feels. After being pushed, every particle will have a new position and a new velocity. At the next time index, we can start the calculation process again. Because we know all the new positions and velocities, we can recalculate the charge and current densities. Continuing the cycle of calculations with the Lorentz Force will then determine the next set of forces, and so on.

3.2 Simulation of Plasma in one dimension using finite difference method

The finite difference method proceeds according to the following steps. First, identify a finite number of discrete points within the domain of interest. Next, the derivatives that appear in the governing differential equations are replaced by the discrete difference approximation. These approximating equations are written in terms of nodal evaluation of the unknown functions.

Given the standard definition for the derivative of a continue function u(x)

$$\frac{du}{dx} \equiv \lim_{h->0} \frac{u(x+h)-u(x)}{(x+h)-x} = \lim_{h->0} \frac{u(x+h)-u(x)}{h} \quad (3.1)$$

$$\left.\frac{du}{dx}\right|_{x_i} \cong \frac{u(x_i+h)-u(x_i)}{(x_i+h)-x_i} = \frac{u_{i+1}-u_i}{x_{i+1}-x_i} \quad (3.2)$$

where point x_i and x_{i+1} are referred to as node points or grid points.

A similar approximation in concept for the first order derivatives can be written for the higher derivatives. Therefore, an approximation for the second derivative of u(x) may be derived as follows.

$$\left.\frac{d^2u}{dx^2}\right|_{x_i} \cong \frac{\left.\frac{du}{dx}\right|_{x_{i+1/2}} - \left.\frac{du}{dx}\right|_{x_{i-1/2}}}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}} \quad (3.3)$$

$$\cong \frac{\frac{u_{i+1}-u_i}{x_{i+1}-x_i} - \frac{u_i-u_{i-1}}{x_i-x_{i-1}}}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}} \quad (3.4)$$

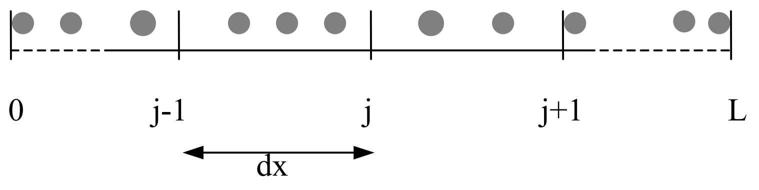
where $x_{i\pm\frac{1}{2}}$ represents the x-location of the midpoint between x_i and $x_{i\pm1}$. If the node spacing is constant ,thus $x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}} = x_{i+1} - x_i = x_i - x_{i-1} = \Delta x$, the above equation is simplified to

$$\left.\frac{d^2u}{dx^2}\right|_{x_i} \cong \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} \tag{3.5}$$

With the finite difference expressions in place of the true continuous partial differential equations and with the spatial grids for calculating the interacting forces, evaluating the interacting forces can be done in a much more efficient manner than summing all the direct interacting forces caused by the particles in the simulation. By accumulating the charge and the current due to the particles on each grid, their densities are used to calculate the acting forces that are applied on the particles. Though each particle actually exerts a force on each other, the grid method (and the alternated view) treats the process as if each particle feels a force due to the field at its location. The advantage of the grid method is that the number of operations is greatly reduced. The grid method procedure consists of three parts. The first part is to accumulate the charge at a grid. Then, the tasks are to calculate the field caused by the accumulation of charge and calculate the acting force that is to be applied to the particles.

3.3 Initial conditions on the particles

The first step is to have an appropriate system length L as shown in Figure 3.1 which is divided into N grids of length dx. From the definition of Debye length, the system length should be many Debye length long. However, a Debye length is not a hard number (temperature is not uniquely defined for a plasma as they are almost always far from thermodynamic equilibrium). In deed, there is no need to have the system be an exact multiple of a debye length ultimately, the plasma which is wanted to simulate will dictate the system length. For instance, the long wavelength behavior in an unbounded system, there must be the system length larger than the longest wavelength which is interested in it. The longer making system length, the more computational intensive the simulation confront. In calculating the Debye length, we also want to know the temperature and the number of the particle to generate the uniform distribution for the particle.



L 〉〉 Debye length

Figure 3.1   System length

Typically there must be number of particles such that are enough particles per grid cell so that the simulation level is acceptably meaningful, the number of the particles per grid cell is suggested that

$$\text{number of particles per grid cell} \gg 10^{\text{(number of dimensions)}} . \tag{3.6}$$

That is, for a 1D simulation, there need more than 10 particles per grid cell and for a 2D simulation, use over 100 particles per grid cell. However, using the different numerical techniques and weighting schemes can change this rule. Commonly, random number generator, which gives the number in the range of [0,1] will be modified with the suitable multiplier (the number of the grids) in order to generate the position of the particles in terms of the distance between grid points and the calculated grid width as shown in figure 3.2. In addition, the desired initial condition for velocity including the initial perturbation are also selected to place the particles in simulation for example, adding a sinusoidal perturbation in particle positions and/or velocities to study kinetic effects such as Landau damping. The cold plasma means no random thermal motion therefore, the electrons and ions are initially motionless. However, the particles could be drifting such as an electron beam, in which case velocity for all particle initially would be the same value. The cold drifting plasma tend to be violently unstable though. The warm plasma requires that each particles be given a velocity as following

$$v[i] = v_{\text{drift}} + \sqrt{\frac{kT}{m}} \times \text{normal random number} \qquad (3.7)$$

where normal random is Gausian (Maxwellian) random number (mean = 0, standard deviation =1). The kind of random generation and the placement of particles are properly considered in initial.

(Number of grid -1) × Random number generator = Initial particle's positions

$$dx = \frac{L}{N-1} \quad \text{and} \quad \frac{dx}{\lambda_D} \approx 1 \qquad (3.8)$$

N = Number of grid cell

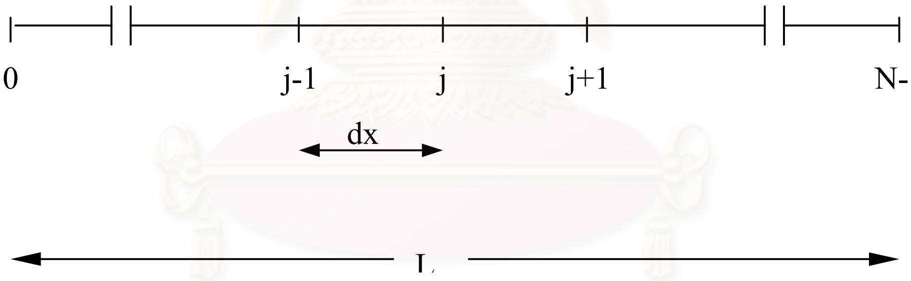


Figure 3. 2 Grid width

3.4 Charge density

The charge density ρ_j (at the grid with an index j) is obtained by summing the charge q_i locates at the position surrounding the grid within the distance of $\pm\frac{\Delta x}{2}$. This method is known as NGP (nearest grid point) which is commonly used in computer simulation model because this method is much more efficient than summing the direct all particles in the simulation. There numerous ways to find the number of particles which are distributed around each grid. Two of the possible methods are shown following. The first way is straightforwardly done by counting the particles directly as seen below in Figure 3.3.

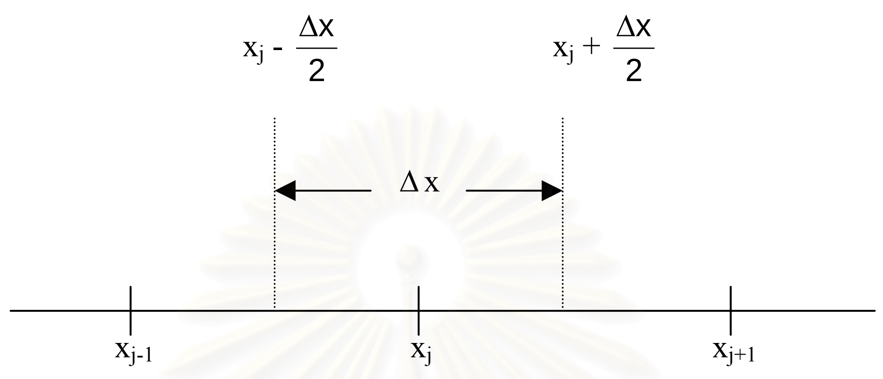


Figure 3.3 Dividing grid for counting particle

By using Loop structure such as For loop in the outer loop to run the number of the particle and the inner loop employ selection structure , IF statement to determine criteria if a particle fall between $x_j - \frac{\Delta x}{2}$ and $x_j + \frac{\Delta x}{2}$ then the given counting variable is added by one according to the number of particle.

The second way is to transform the position of each particle , which is often real number to an integer number then use this number as an array's index of charging particle variable (charge particle is grid quantity) to count the particle. For example, a particle at the location of 8.4 will give the array index of 8 . Therefore, the value of counting variable with the index of 8 must be increased by one . However , a particles at the location of 8.8 will also give the array index of 8 which is not desirable. In such case , 0.5 might be added to the particle's position before the conversion. As a result, 8.8+0.5 and 8.4+0.5, which are equal to 9.3 and 8.9, will be are converted to 9 and 8, respectively.

3.5 Potential field and electric field

From ρ_j , an electric field is found on the grid by solving the finite difference forms of $\nabla\cdot\mathbf{E} = \frac{\rho}{\varepsilon_0}$ and $\mathbf{E} = -\nabla\phi$ that can be approximated with the finite difference technique.

$$E = -\nabla\phi \tag{3.9}$$

$$\nabla\cdot E = \frac{\rho}{\varepsilon_0} \tag{3.10}$$

Both are combined to obtain Poisson's equation ϕ, where ϕ is the electric potential

$$\nabla\cdot\nabla\phi = -\frac{\rho}{\varepsilon_0} \quad . \tag{3.11}$$

In cartesian coordinates,

$$\nabla\cdot\nabla\phi = \frac{\partial}{\partial x}\left(\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(\frac{\partial\phi}{\partial y}\right)+\frac{\partial}{\partial z}\left(\frac{\partial\phi}{\partial z}\right) \tag{3.12}$$

or
$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2} \tag{3.13}$$

where operator $\nabla\cdot\nabla$ is abbreviated ∇^2 .

Therefore
$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2} = -\frac{\rho}{\varepsilon_0} \tag{3.14}$$

in one dimension the above equation is reduced into

$$\nabla^2\phi = \frac{d^2\phi}{dx^2} = -\frac{\rho}{\varepsilon_0} \quad . \tag{3.15}$$

With the finite difference, E and ϕ are approximated as

$$E_j = \frac{\phi_{j-1}-\phi_{j+1}}{2\Delta x} \tag{3.16}$$

and $$\frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{(\Delta x)^2} = -\frac{\rho_j}{\varepsilon_0} . \qquad (3.17)$$

Once ρ_j is known, it is used to obtain ϕ_j and E_j for j starting from 0 to N-1. A set of the linear algebraic equations are resulted as illustrated in the following matrix equation,

$$\begin{bmatrix} -2 & 1 & & & . & \\ 1 & -2 & 1 & & . & \\ & 1 & -2 & 1 & . & \\ . & . & . & . & . & . \\ & & & . & 1 & -2 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ . \\ \phi_{N-2} \\ \phi_{N-1} \end{bmatrix} = \begin{bmatrix} -\frac{\rho_1}{\varepsilon_0}(\Delta x)^2 - \phi_0 \\ -\frac{\rho_2}{\varepsilon_0}(\Delta x)^2 \\ -\frac{\rho_3}{\varepsilon_0}(\Delta x)^2 \\ . \\ -\frac{\rho_{N-1}}{\varepsilon_0}(\Delta x)^2 - \phi_N \end{bmatrix} \qquad (3.18)$$

There are various boundary conditions that can be applied for the calculation. For this simulation employs bounded system by applying zero potential at both ends, x = 0, x = L ($\phi_0 = 0$ and $\phi_N = 0$ are given), this is homogeneous problem which plasma is bounded between grounded metal parallel plates. For inhomogeneous Dirchlet boundary condition in Poisson system, there must be added a linear voltage as below.

$$\text{phi[i]} = \text{phi\_homog[i]} + V \times \frac{(N-i)}{i} \quad , i = 0, 1 ,.., N \qquad (3.19)$$

Here V is the applied voltage across the metal parallel plates and the right electrode is used as the reference or ground electrode, phi[i] is potential field at each grid cell.

3.6 Motion of the particles

The particles are advanced in time using the particle equations of motion and Lorentz equation with the self-consistently calculated electric and magnetic fields. Eventually, the motion of a particle can be obtained by combination of Lorentz equation with Newton Law.

$$m \, . \, a_i = q\,(E_j + v_i \times B_j)\, . \tag{3.20}$$

$$a_i = \frac{dv_i}{dt} \, . \tag{3.21}$$

$$v_i = \frac{dx_i}{dt} \, . \tag{3.22}$$

These equations are replaced by the finite difference equations as following

$$a_{\,new} = \frac{v_{new} - v_{old}}{\Delta t} \tag{3.23}$$

$$v_{\,new} = \frac{x_{new} - x_{old}}{\Delta t} \tag{3.24}$$

Where x , v and a are displacement , velocity and acceleration, respectively. The index old indicate the previous time and the index new represent the current time. As the velocity and position of each particle are not known at the same time therefore the time must be centered properly. The algorithm generally used to solve this set of equations is known as time-centered leap frog scheme. Time and space centered roughly means that the finite difference approximations which are using is better than first order accurate. This is time centered properly and for sufficiently small dt, stable. The position and field are calculate at integer time-steps and velocities at half time-steps of positions as following

$$a(t) = \frac{v(t+\frac{dt}{2}) - v(t-\frac{dt}{2})}{dt} \tag{3.25}$$

and $$v(t+\frac{dt}{2}) = \frac{x(t+dt)-x(t)}{dt} \quad . \qquad (3.26)$$

$$\omega_P \times dt \approx 0.2 \qquad (3.27)$$

where ω_P is plasma frequency.

If the time step too small, round off errors will screw up and if this take too big value (>2), it becomes unstable and phase errors in the particles orbit also increase as dt is increased. This rule of thumb is suitable for particle in cell with Monte Carlo Collision simulation. However, if everything normalize correctly, there are no even needs the multiplies for dt. In this case, velocity is in units of grid cells per time step and displacement is normalized to the cell spacing.

3.7 Computing cycle

The calculations can be repeated to simulate the motion of a plasma over a time period. The positions and the velocities of the particles are updated by calculating the forces on the particles from the interpolation of the field values at the grid points. The updated fields are found by solving the field equations on the grids using the new charge and the new current densities. The new charges and the new current densities at the grid points are then calculated with the interpolation from the new positions and the new velocities of the particles. This procedure is repeated for many time steps as illustrated in Figure 3.4 .

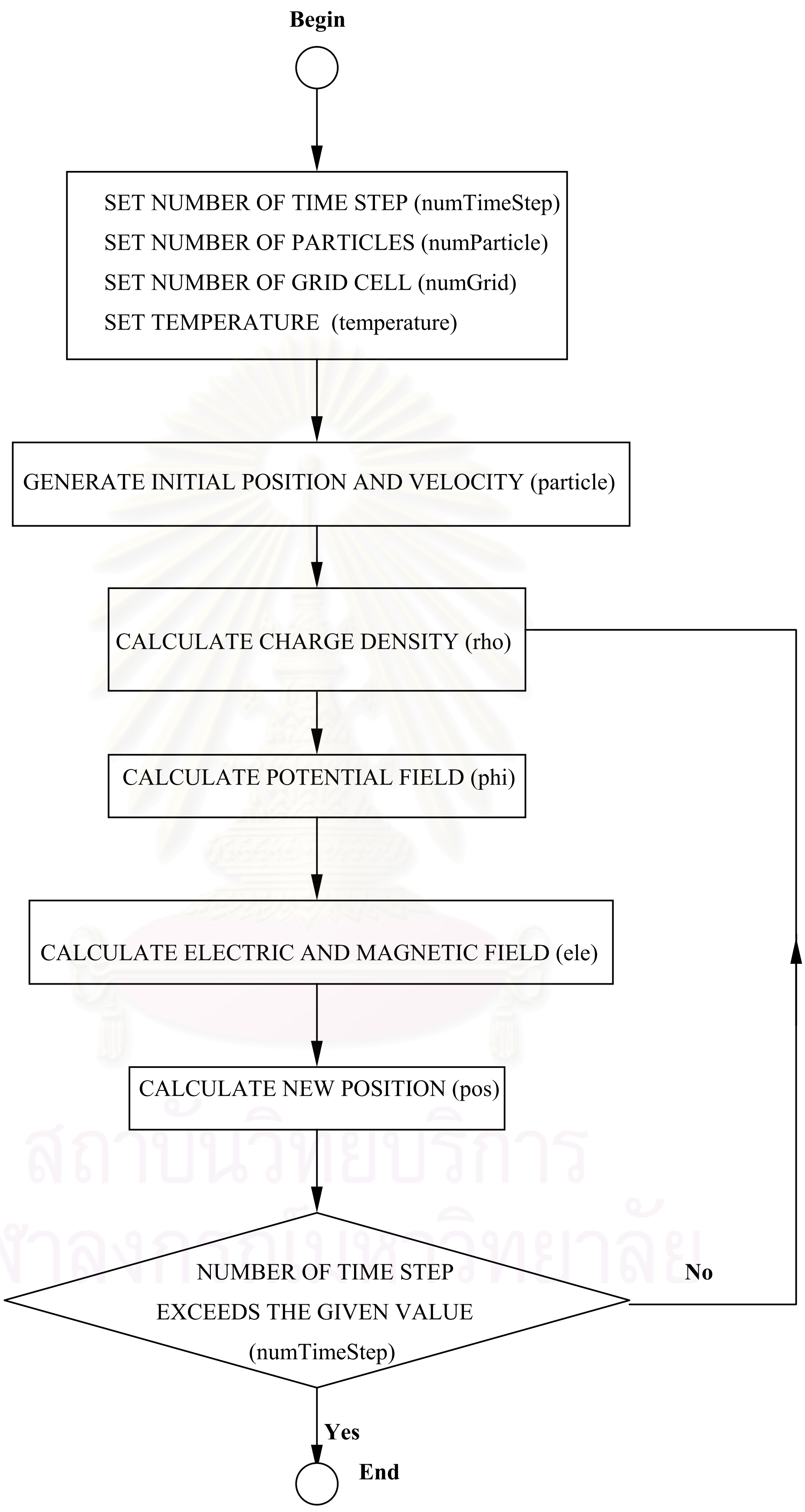

Begin
SET NUMBER OF TIME STEP (numTimeStep)
SET NUMBER OF PARTICLES (numParticle)
SET NUMBER OF GRID CELL (numGrid)
SET TEMPERATURE (temperature)
GENERATE INITIAL POSITION AND VELOCITY (particle)
CALCULATE CHARGE DENSITY (rho)
CALCULATE POTENTIAL FIELD (phi)
CALCULATE ELECTRIC AND MAGNETIC FIELD (ele)
CALCULATE NEW POSITION (pos)
NUMBER OF TIME STEP EXCEEDS THE GIVEN VALUE (numTimeStep)
No
Yes
End

Figure 3.4 Flow chart for plasma simulation in one dimension

CHAPTER IV

RESULTS FROM SIMULATIONS

4.1 Overview

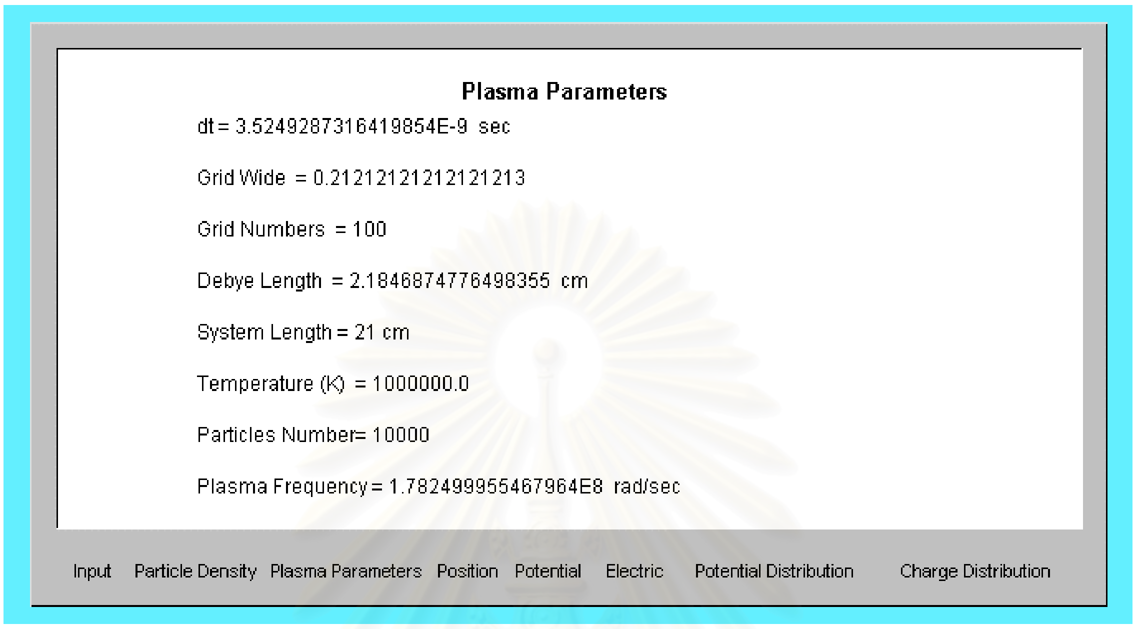
In this chapter, the computer model for one dimensional plasma is developed to simulate the distribution and motion of the particles in an ideal one dimensional plasma. The code is written in Java language which can be interpreted by the browser such as Netscape (version 4.0 or later) or Internet explorer(version 4.0 or later). The source code is also provided in the appendix of this thesis. Figure 4.1 shows the introduction screen of the program when it is run by such browser. The calculation of the plasma parameters and the result will be illustrated. In addition, the investigation into the effect of using the different number of particles, the number of grids and the time step will be also conducted.

Figure 4.1 Plasma simulation program

4.2 Results from simulation for the motion

We will start the investigation with 10,000 particles at the temperature of 1,000,000 K and with 100 grid points. Other necessary information is as given in

Figure 4.2. The output from the calculation consists of the position of each particle , the particle density ,the potential field and the electric field at each grid .

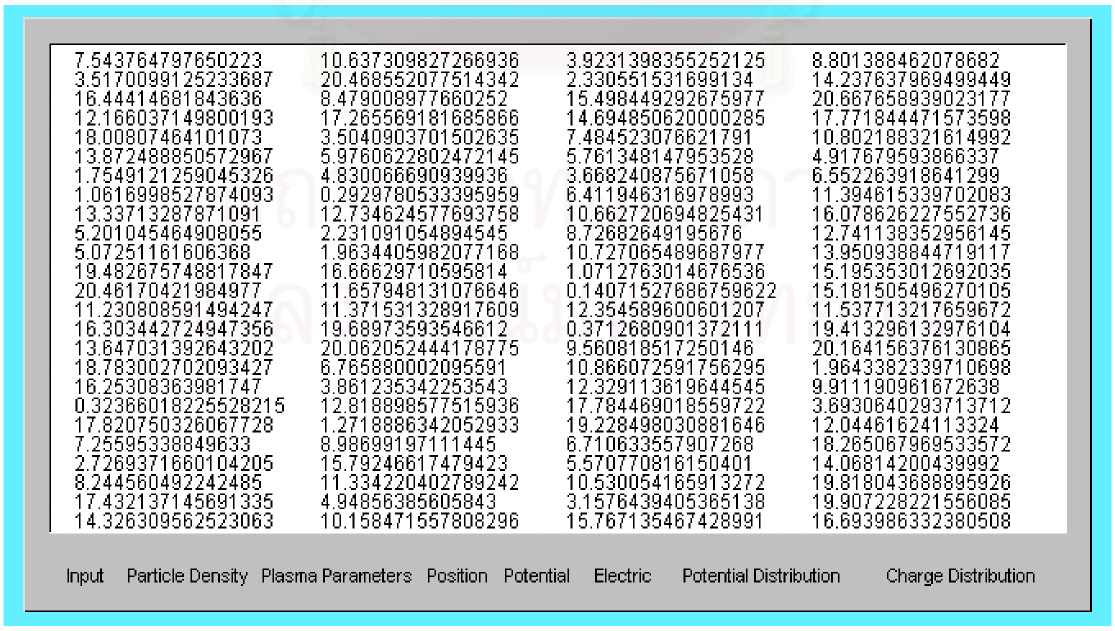


Plasma Parameters

```
dt= 3.5249287316419854E-9  sec

Grid Wide  = 0.21212121212121213

Grid Numbers  = 100

Debye Length  = 2.1846874776498355  cm

System Length = 21 cm

Temperature (K)  = 1000000.0

Particles Number= 10000

Plasma Frequency= 1.782499955467964E8  rad/sec
```

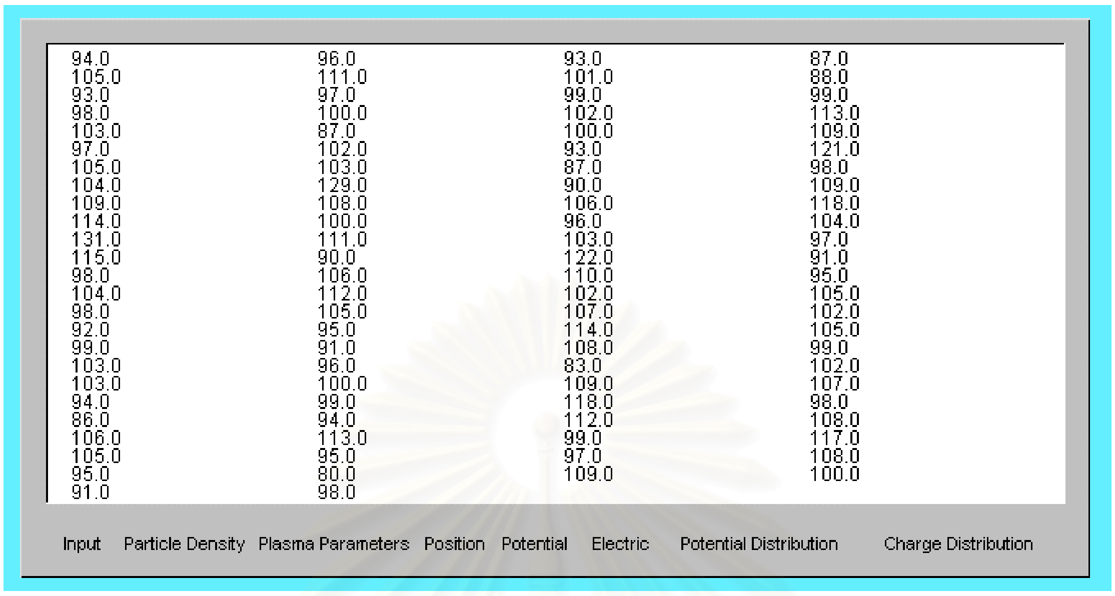
Figure4.2 plasma parameters

Some of the results obtained from the calculation with the input as given in Figure 4.2 are illustrated in Figure 4.3, 4.4, 4.5 and 4.6



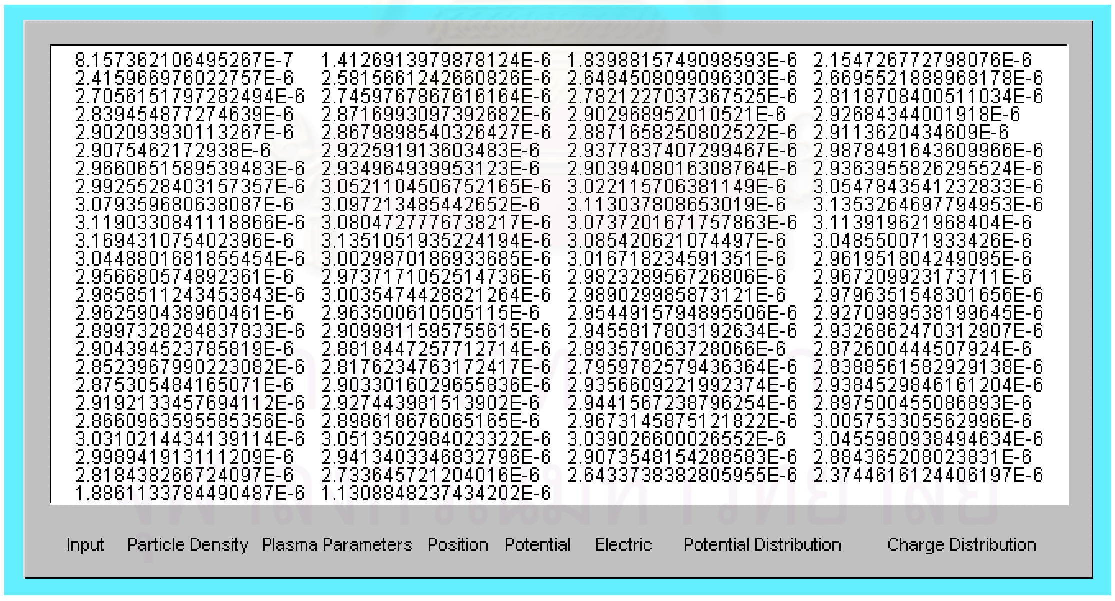
```
7.543764797650223      10.637309827266936     3.9231398355252125     8.801388462078682
3.5170099125233687     20.468552077514342     2.330551531699134      14.237637969499449
16.44414681843636      8.479008977660252      15.498449292675977     20.667658939023177
12.166037149800193     17.265569181685866     14.694850620000285     17.771844471573598
18.00807464101073      3.5040903701502635     7.484523076621791      10.802188321614992
13.872488850572967     5.9760622802472145     5.761348147953528      4.917679593866337
1.7549121259045326     4.830066690939936      3.668240875671058      6.552263918641299
1.0616998527874093     0.2929780533395959     6.411946316978993      11.394615339702083
13.33713287871091      12.734624577693758     10.662720694825431     16.078626227552736
5.201045464908055      2.231091054894545      8.72682649195676       12.741138352956145
5.07251161606368       1.9634405982077168     10.727065489687977     13.950938844719117
19.482675748817847     16.66629710595814      1.0712763014676536     15.195353012692035
20.46170421984977      11.657948131076646     0.14071527686759622    15.181505496270105
11.230808591494247     11.371531328917609     12.354589600601207     11.537713217659672
16.303442724947356     19.68973593546612      0.3712680901372111     19.413296132976104
13.647031392643202     20.062052444178775     9.560818517250146      20.164156376130865
18.783002702093427     6.765880002095591      10.866072591756295     1.9643382339710698
16.25308363981747      3.861235342253543      12.329113619644545     9.911190961672638
0.32366018225528215    12.818898577515936     17.784469018559722     3.6930640293713712
17.820750326067728     1.2718886342052933     19.228498030881646     12.04461624113324
7.25595338849633       8.98699197111445       6.710633557907268      18.265067969533572
2.7269371660104205     15.79246617479423      5.570770816150401      14.06814200439992
8.244560492242485      11.334220402789242     10.530054165913272     19.818043688895926
17.432137145691335     4.94856385605843       3.1576439405365138     19.907228221556085
14.326309562523063     10.158471557808296     15.767135467428991     16.693986332380508
```

Figure 4.3 Displacement of the particles (cm.)



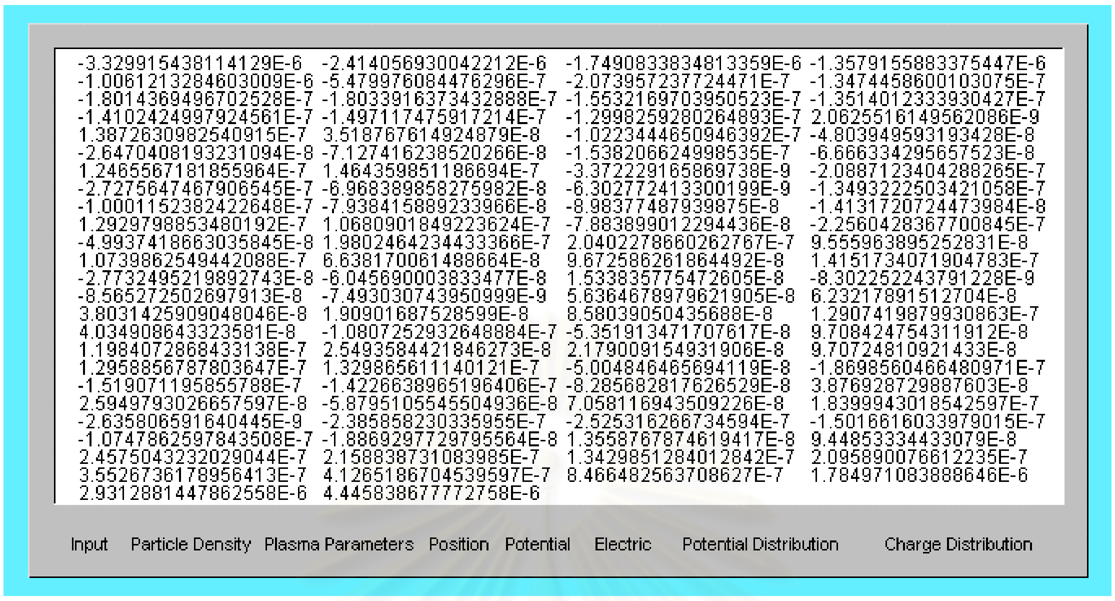
```
94.0      96.0      93.0      87.0
105.0     111.0     101.0     88.0
93.0      97.0      99.0      99.0
98.0      100.0     102.0     113.0
103.0     87.0      100.0     109.0
97.0      102.0     93.0      121.0
105.0     103.0     87.0      98.0
104.0     129.0     90.0      109.0
109.0     108.0     106.0     118.0
114.0     100.0     96.0      104.0
131.0     111.0     103.0     97.0
115.0     90.0      122.0     91.0
98.0      106.0     110.0     95.0
104.0     112.0     102.0     105.0
98.0      105.0     107.0     102.0
92.0      95.0      114.0     105.0
99.0      91.0      108.0     99.0
103.0     96.0      83.0      102.0
103.0     100.0     109.0     107.0
94.0      99.0      118.0     98.0
86.0      94.0      112.0     108.0
106.0     113.0     99.0      117.0
105.0     95.0      97.0      108.0
95.0      80.0      109.0     100.0
91.0      98.0
```

Figure 4.4 Particle density at each grid



```
8.157362106495267E-7    1.412691397987812 4E-6  1.839881574909859 3E-6  2.154726772798076E-6
2.415966976022757E-6    2.581566124266082 6E-6  2.648450809909630 3E-6  2.669552188896817 8E-6
2.705615179728249 4E-6  2.745976786761616 4E-6  2.782122703736752 5E-6  2.811870840051103 4E-6
2.839454877274639E-6    2.871699309739268 2E-6  2.902968952010521E-6    2.92684344001918E-6
2.902093930113267E-6    2.867989854032642 7E-6  2.887165825080252 2E-6  2.911362043460 9E-6
2.9075462172938E-6      2.922591913603483E-6    2.937783740729946 7E-6  2.987849164360996 6E-6
2.966065158953948 3E-6  2.934964939953123E-6    2.903940801630876 4E-6  2.936395582629552 4E-6
2.992552840315735 7E-6  3.052110450675216 5E-6  3.022115706381149E-6    3.054784354123283 3E-6
3.079359680638087E-6    3.097213485442652E-6    3.113037808653019E-6    3.135326469779495 3E-6
3.119033084111886 6E-6  3.080472777673821 7E-6  3.073720167175786 3E-6  3.113919621968404E-6
3.169431075402396E-6    3.135105193522419 4E-6  3.085420621074497E-6    3.048550071933426E-6
3.044880168185545 4E-6  3.002987018693368 5E-6  3.016718234591351E-6    2.961951804249095E-6
2.956680574892361E-6    2.973717105251473 6E-6  2.982328956726806E-6    2.967209923173711E-6
2.985851124345384 3E-6  3.003547442882126 4E-6  2.989029985873121E-6    2.979635154830165 6E-6
2.962590438960461E-6    2.963500610505115E-6    2.954491579489550 6E-6  2.927098953819964 5E-6
2.899732828483783 3E-6  2.909981159575561 5E-6  2.945581780319263 4E-6  2.932686247031290 7E-6
2.904394523785819E-6    2.881844725771271 4E-6  2.893579063728066E-6    2.872600444507924E-6
2.852396799022308 2E-6  2.817623476317241 7E-6  2.795978257943636 4E-6  2.838856158292913 8E-6
2.875305484165071E-6    2.903301602965583 6E-6  2.935660922199237 4E-6  2.938452984616120 4E-6
2.919213345769411 2E-6  2.927443981513902E-6    2.944156723879625 4E-6  2.897500455086893E-6
2.866096359558535 6E-6  2.898618676065165E-6    2.967314587512182 2E-6  3.005753305562996E-6
3.031021443413911 4E-6  3.051350298402332 2E-6  3.039026600026552E-6    3.045598093849463 4E-6
2.998941913111209E-6    2.941340334683279 6E-6  2.907354815428858 3E-6  2.884365208023831E-6
2.818438266724097E-6    2.733645721204016E-6    2.643373838280595 5E-6  2.374461612440619 7E-6
1.886113378449048 7E-6  1.130884823743420 2E-6
```

Figure 4.5 Potential field at each grid



```
-3.32991543811412 9E-6  -2.41405693004221 2E-6  -1.7490833834813359E-6 -1.35791558833754 47E-6
-1.0061213284603009E-6 -5.4799760844762 96E-7  -2.07395723772447 1E-7  -1.3474458600103075E-7
-1.8014369496702528E-7 -1.8033916373432888E-7 -1.5532169703950523E-7 -1.3514012333930427E-7
-1.4102424997924561E-7 -1.497117475917214E-7  -1.2998259280264893E-7 2.0625516149562086E-9
1.3872630982540915E-7  3.518767614924879E-8   -1.0223444650946392E-7 -4.803949593193428E-8
-2.6470408193231094E-8 -7.127416238520266E-8  -1.5382066249985 35E-7  -6.6663342956575 23E-8
1.2465567181855964E-7  1.464359851186694E-7   -3.37222916586973 8E-9  -2.0887123404288265E-7
-2.7275647467906545E-7 -6.968389858275982E-8  -6.302772413300199E-9  -1.3493222503421058E-7
-1.0001152382422648E-7 -7.938415889233966E-8  -8.98377487939875E-8   -1.4131720724473984E-8
1.2929798853480192E-7  1.0680901849223624E-7  -7.883899012294436E-8  -2.2560428367700845E-7
-4.9937418663035845E-8 1.9802464234433366E-7  2.0402278660262767E-7  9.555963895252831E-8
1.0739862549442088E-7  6.638170061488664E-8   9.672586261864492E-8   1.4151734071904783E-7
-2.7732495219892743E-8 -6.045690003833477E-8  1.533835775472605E-8   -8.302252243791228E-9
-8.565272502697913E-8  -7.493030743950999E-9  5.6364678979621905E-8  6.23217891512704E-8
3.8031425909048046E-8  1.90901687528599E-8    8.58039050435688E-8    1.2907419879930863E-7
4.034908643323581E-8   -1.0807252932648884E-7 -5.351913471707617E-8  9.708424754311912E-8
1.1984072868433138E-7  2.5493584421846273E-8  2.179009154931906E-8   9.70724810921433E-8
1.2958856787803647E-7  1.329865611140121E-7   -5.004846465694119E-8  -1.8698560466480971E-7
-1.519071195855788E-7  -1.4226638965196406E-7 -8.285682817626529E-8  3.876928729887603E-8
2.5949793026657597E-8  -5.8795105545504936E-8 7.058116943509226E-8   1.8399943018542597E-7
-2.635806591640445E-9  -2.385858230335955E-7  -2.525316266734594E-7  -1.5016616033979015E-7
-1.0747862597843508E-7 -1.8869297729795564E-8 1.3558767874619417E-8  9.44853334433079E-8
2.4575043232029044E-7  2.158838731083985E-7   1.3429851284012842E-7  2.095890076612235E-7
3.5526736178956413E-7  4.1265186704539597E-7  8.466482563708627E-7   1.784971083888646E-6
2.9312881447862558E-6  4.445838677772758E-6
```

Figure 4.6 Electric field at each grid

If all the electrons of plasma are removed from a uniform background of ions with certain distance as seen Figure 4.3, the electric field in Figure 4.6 are generated due to distribution of particle density as illustrated in Figure 4.4 according to Poisson’s equation. Then the forces on the particles are found by using the electric field and magnetic field in Lorentz equation of motion and recalculate the field from the particles at their new positions.

4.3 The various parameters in simulation

By investigating the results at one time step. The distribution of the potential field with one grid and with many grids according to the theory is illustrated in Figure4.7 and Figure4.8, respectively. The simulation with 10,000 and 1000 particles give the potential distribution as shown in Figure 4.10 and 4.11.When the density is increased (more particles), the local concentration of the charge particles around each grid is also increased which in turn gives rise to the potential field at each grid and having smoother profile of distribution than the one obtained with the less number of particles.

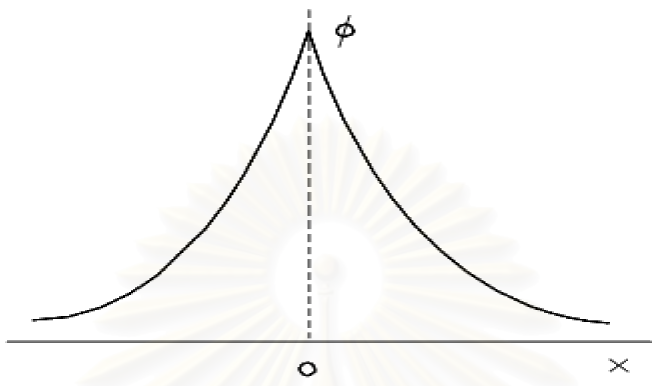


Figure 4.7 Potential distribution near a single grid in a plasma

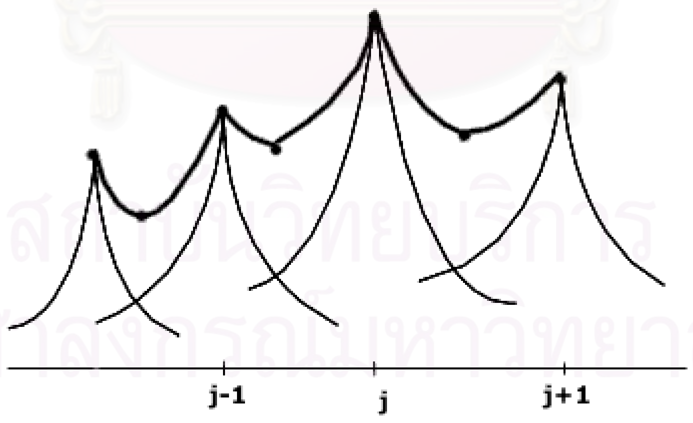


Figure 4.8 Potential distribution with a small number of grids .

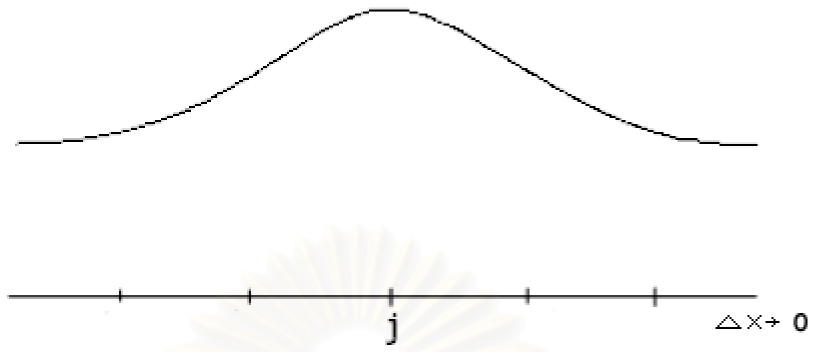


Figure 4.9 Potential distribution with a large number of grids.

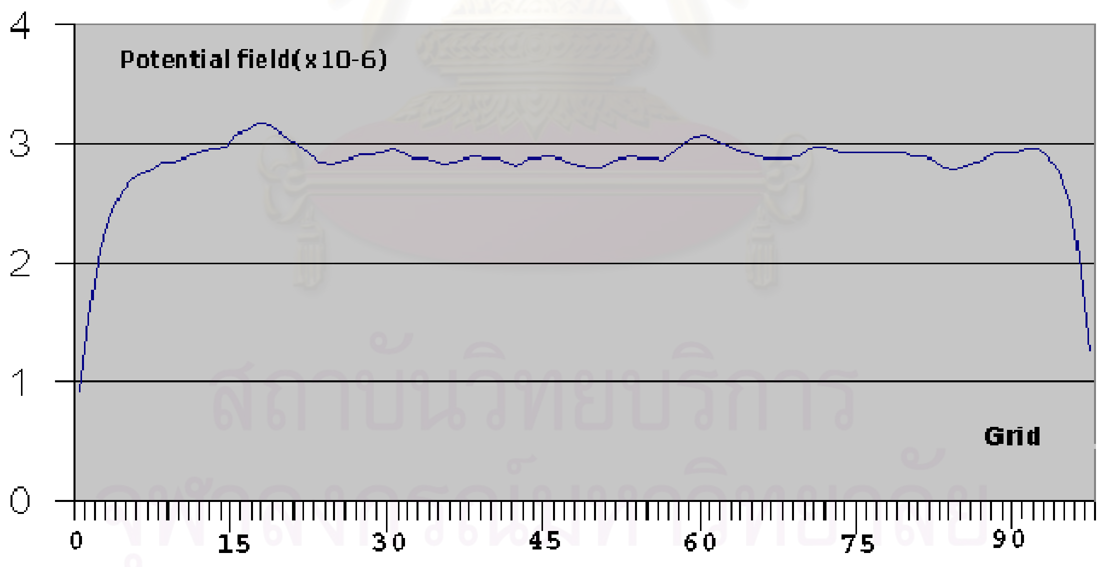


Figure 4.10 Potential Distribution from the simulation with 10,000 particles

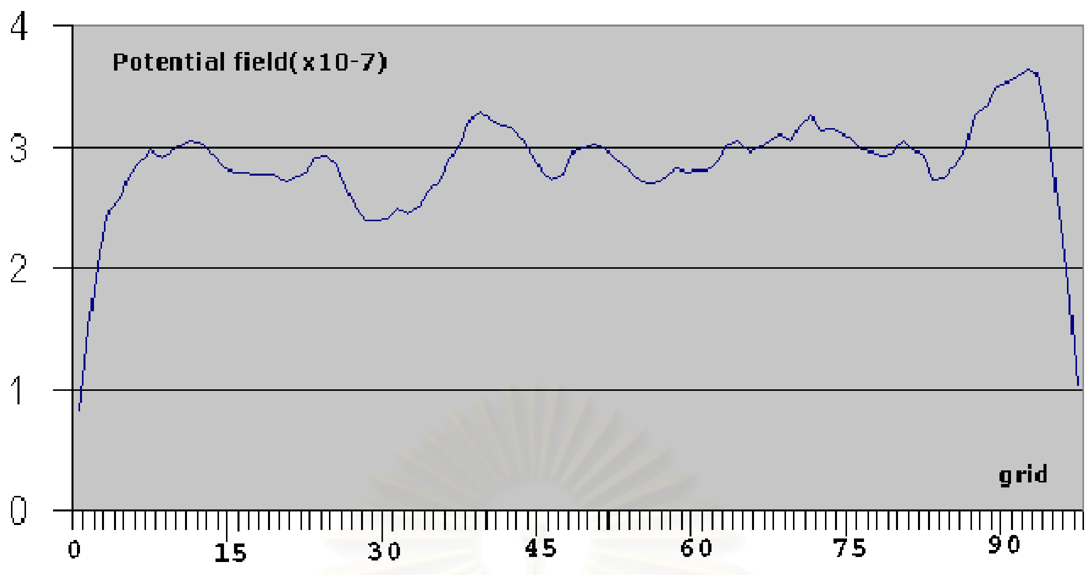


Figure 4.11 Potential Distribution from the simulation with 1,000 particles

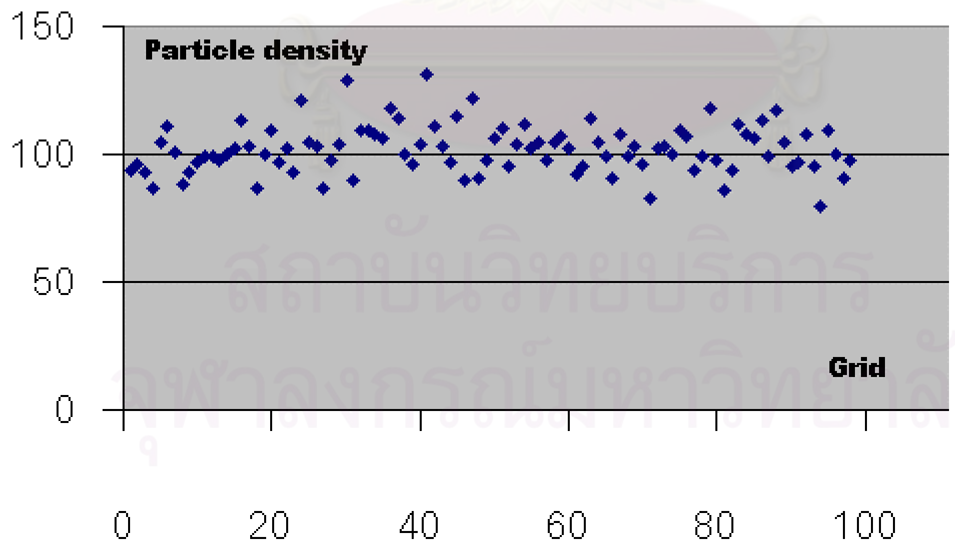
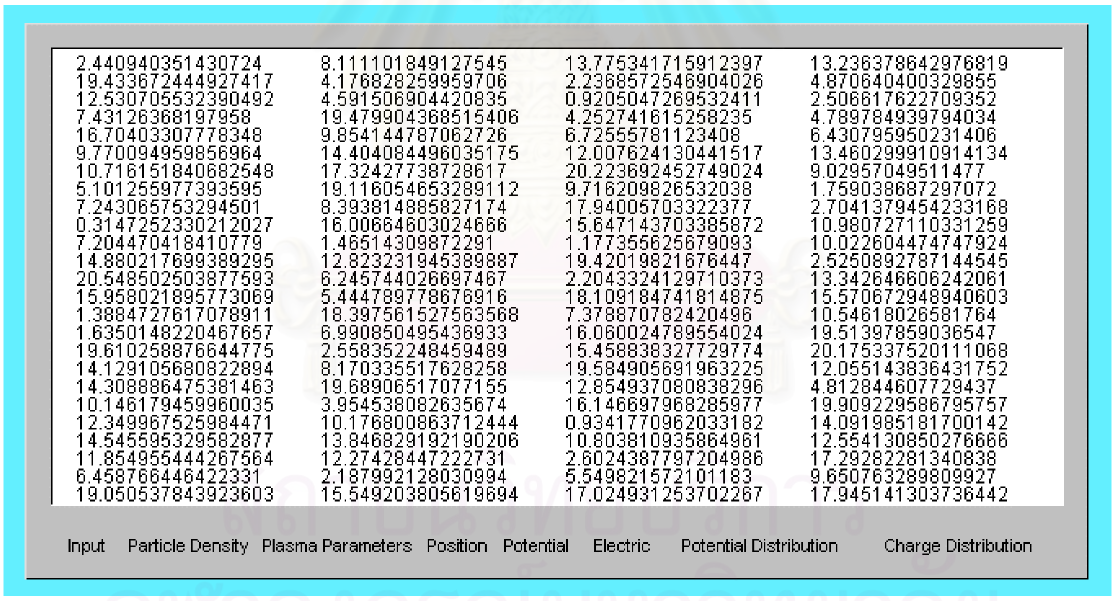


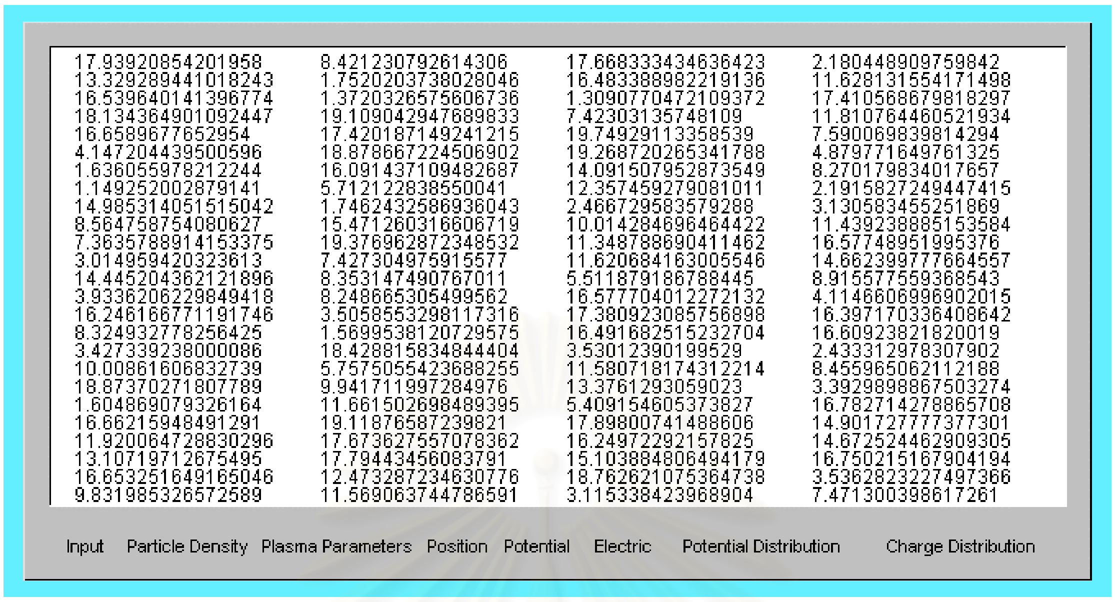
Figure 4.12 Density Distribution from the simulation with 10,000 particles and 100 grids

When the effect of the time step is considered, with the time step that more than 100 times the time scale of electron oscillations, the result from the calculation for the new displacement of the electrons leads to the error in the displacement of the electrons. In such case, the electrons are moved beyond the boundary of system. Consequentially, the new computing cycle is continued with the erroneous initiated condition. With the estimation of the plasma frequency of 10^6 Hz, time step of 10^{-7} sec. and the system length of 21 cm. , the displacement of each electron is obtained as shown in Figure 4.13. When the time step is 10^{-4} sec. , 10^{-3} sec. and 10^{-2} sec. The displacement are shown in Figure 4.14, 4.15 and 4.16, respectively. The erroneous results are clearly seen with the time step 10^{-3} sec. and 10^{-2} sec.



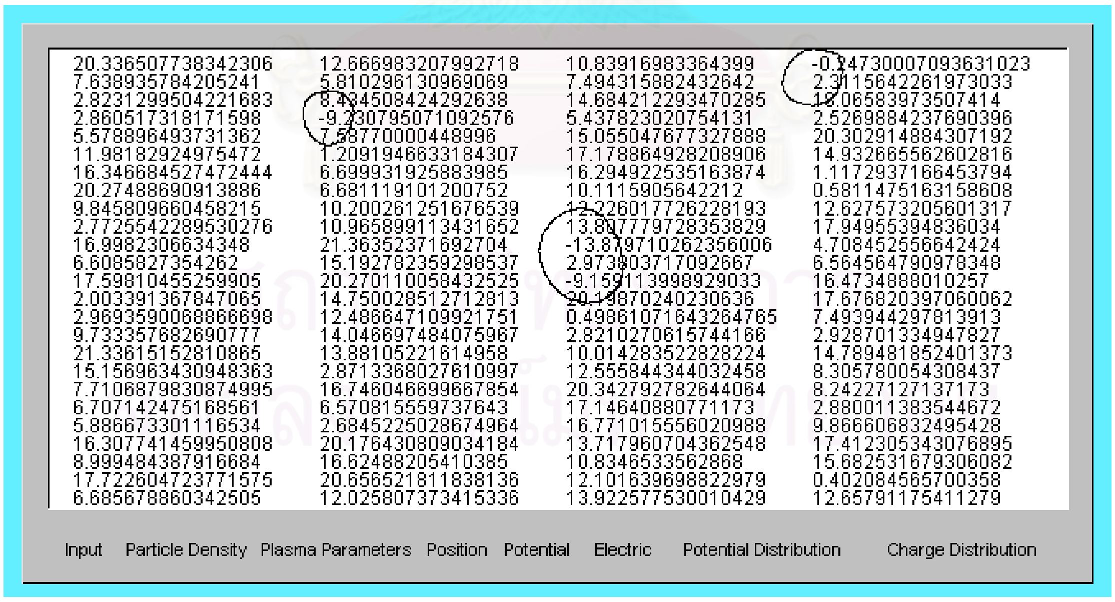
```
2.440940351430724     8.111101849127545     13.775341715912397    13.236378642976819
19.433672444927417    4.176828259959706     2.2368572546904026    4.870640400329855
12.530705532390492    4.591506904420835     0.9205047269532411    2.506617622709352
7.43126368197958      19.479904368515406    4.252741615258235     4.789784939794034
16.70403307778348     9.854144787062726     6.72555781123408      6.430795950231406
9.770094959856964     14.404084496035175    12.007624130441517    13.460299910914134
10.716151840682548    17.32427738728617     20.223692452749024    9.02957049511477
5.101255977393595     19.116054653289112    9.716209826532038     1.759038687297072
7.243065753294501     8.393814885827174     17.94005703322377     2.7041379454233168
0.3147252330212027    16.00664603024666     15.647143703385872    10.980727110331259
7.204470418410779     1.46514309872291      1.177355625679093     10.022604474747924
14.880217699389295    12.823231945389887    19.42019821676447     2.5250892787144545
20.548502503877593    6.245744026697467     2.2043324129710373    13.342646606242061
15.958021895773069    5.444789778676916     18.109184741814875    15.570672948940603
1.3884727617078911    18.397561527563568    7.378870782420496     10.54618026581764
1.6350148220467657    6.990850495436933     16.060024789554024    19.51397859036547
19.610258876644775    2.558352248459489     15.458838327729774    20.175337520111068
14.129105680822894    8.170335517628258     19.584905691963225    12.055143836431752
14.308886475381463    19.68906517077155     12.854937080838296    4.812844607729437
10.146179459960035    3.954538082635674     16.146697968285977    19.909229586795757
12.349967525984471    10.176800863712444    0.9341770962033182    14.091985181700142
14.545595329582877    13.846829192190206    10.803810935864961    12.554130850276666
11.854955444267564    12.27428447222731     2.6024387797204986    17.29282281340838
6.458766446422331     2.187992128030994     5.549821572101183     9.650763289809927
19.050537843923603    15.549203805619694    17.024931253702267    17.945141303736442
```

Figure 4.13 Displacement of the electrons from the simulation with 10,000 particles at time step 10^{-7} sec.



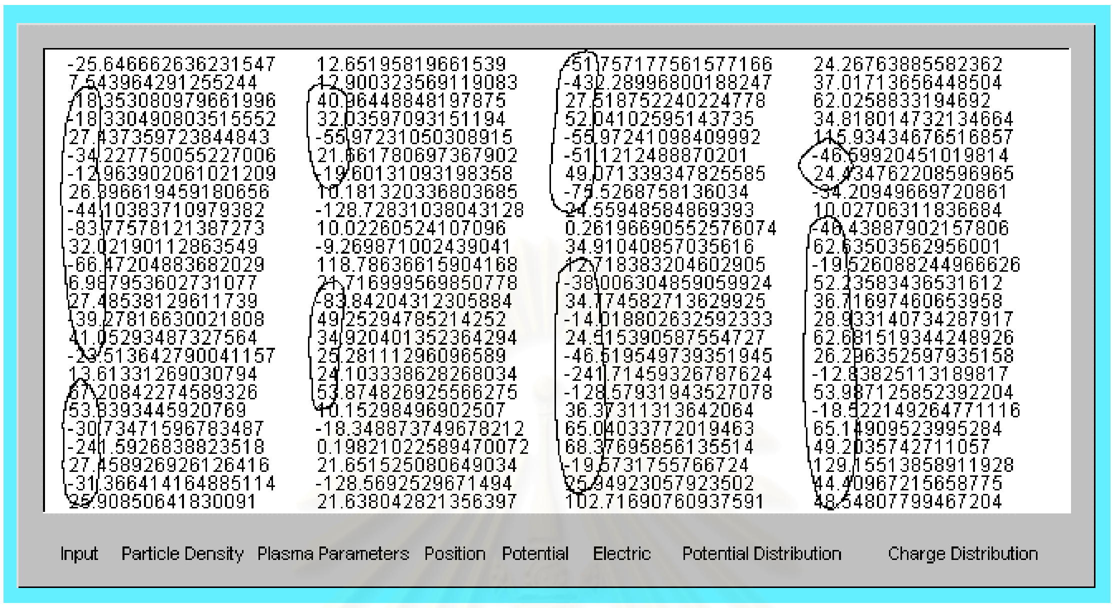
```
17.939208542019 58     8.421230792614306      17.668333434636423     2.180448909759842
13.329289441018243     1.7520203738028046     16.483388982219136     11.628131554171498
16.539640141396774     1.3720326575606736     1.3090770472109372     17.410568679818297
18.134364901092447     19.109042947689833     7.42303135748109       11.810764460521934
16.65896776529 54      17.420187149241215     19.74929113358539      7.590069839814294
4.147204439500596      18.878667224506902     19.268720265341788     4.879771649761325
1.636055978212244      16.091437109482687     14.091507952873549     8.270179834017657
1.149252002879141      5.712122838550041      12.357459279081011     2.1915827249447415
14.985314051515042     1.7462432586936043     2.466729583579288      3.130583455251869
8.564758754080627      15.471260316606719     10.014284696464422     11.439238885153584
7.3635788914153375     19.376962872348532     11.348788690411462     16.57748951995376
3.014959420323613      7.427304975915577      11.620684163005546     14.662399777664557
14.445204362121896     8.353147490767011      5.511879186788445      8.915577559368543
3.9336206229849418     8.248665305499562      16.577704012272132     4.1146606996902015
16.246166771191746     3.5058553298117316     17.380923085756898     16.397170336408642
8.324932778256425      1.5699538120729575     16.491682515232704     16.60923821820019
3.427339238000086      18.428815834844404     3.53012390199529       2.433312978307902
10.00861606832739      5.7575055423688255     11.580718174312214     8.455965062112188
18.87370271807789      9.941711997284976      13.3761293059023       3.3929898867503274
1.604869079326164      11.661502698489395     5.409154605373827      16.782714278865708
16.66215948491291      19.11876587239821      17.89800741488606      14.901727777377301
11.920064728830296     17.673627557078362     16.24972292157825      14.672524462909305
13.10719712675495      17.79443456083791      15.103884806494179     16.750215167904194
16.653251649165046     12.473287234630776     18.762621075364738     3.5362823227497366
9.831985326572589      11.569063744786591     3.115338423968904      7.471300398617261
```

Figure 4.14 Displacement of the electrons from the simulation with 10,000 particles at time step 10^{-4} sec.



```
20.336507738342306     12.666983207992718     10.83916983364399      -0.24730007093631023
7.638935784205241      5.810296130969069      7.494315882432642      2.3115642261973033
2.8231299504221683     8.434508424292638      14.684212293470285     18.06583973507414
2.860517318171598      -9.230795071092576     5.437823020754131      2.5269884237690396
5.578896493731362      7.58770000448996       15.055047677327888     20.302914884307192
11.98182924975472      1.2091946633184307     17.178864928208906     14.932665562602816
16.346684527472444     6.699931925883985      16.294922535163874     1.1172937166453794
20.27488690913886      6.681119101200752      10.1115905642212       0.5811475163158608
9.845809660458215      10.200261251676539     12.226017726228193     12.627573205601317
2.7725542289530276     10.965899113431652     13.807779728353829     17.94955394836034
16.9982306634348       21.36352371692704      -13.879710262356006    4.708452556642424
6.6085827354262        15.192782359298537     2.973803717092667      6.564564790978348
17.59810455259905      20.270110058432525     -9.159113998929033     16.4734888010257
2.003391367847065      14.750028512712813     20.19870240230636      17.676820397060062
2.9693590068866698     12.486647109921751     0.49861071643264765    7.493944297813913
9.733357682690777      14.046697484075967     2.821027061574416 6    2.928701334947827
21.33615152810865      13.88105221614958      10.014283522828224     14.789481852401373
15.156963430948363     2.8713368027610997     12.555844344032458     8.305780054308437
7.7106879830874995     16.746046699667854     20.342792782644064     8.24227127137173
6.707142475168561      6.570815559737643      17.14640880771173      2.880011383544672
5.886673301116534      2.6845225028674964     16.771015556020988     9.866606832495428
16.307741459950808     20.176430809034184     13.717960704362548     17.412305343076895
8.999484387916684      16.62488205410385      10.8346533562868       15.682531679306082
17.722604723771575     20.656521811838136     12.101639698822979     0.402084565700358
6.685678860342505      12.025807373415336     13.922577530010429     12.65791175411279
```

Figure 4.15 Displacement of the electrons from the simulation with 10,000 particles at time step 10^{-3} sec.



```
-25.646662636231547    12.65195819661539      -51.757177561577166    24.26763885582362
7.543964291255244      12.900323569119083     -432.28996800188247    37.01713656448504
-18.353080979661996    40.96448848197875      27.518752240224778     62.0258833194692
-18.330490803515552    32.03597093151194      52.04102595143735      34.818014732134664
27.437359723844843     -55.97231050308915     -55.97241098409992     115.93434676516857
-34.227750055227006    21.661780697367902     -51.1212488870201      -46.59920451019814
-12.963902061021209    -19.60131093198358     49.071339347825585     24.434762208596965
26.396619459180656     10.181320336803685     -75.5268758136034      -34.20949669720861
-44.10383710979382     -128.72831038043128    24.55948584869393      10.02706311836684
-83.77578121387273     10.02260524107096      0.26196690552576074    -40.43887902157806
32.02190112863549      -9.269871002439041     34.91040857035616      62.63503562956001
-66.47204883682029     118.78636615904168     12.718383204602905     -19.526088244966626
6.987953602731077      21.716999569850778     -38.006304859059924    52.23583436531612
27.48538129611739      -83.84204312305884     34.774582713629925     36.71697460653958
-39.27816630021808     49.25294785214252      -14.018802632592333    28.933140734287917
41.05293487327564      34.920401352364294     24.515390587554727     62.681519344248926
-23.513642790041157    25.28111296096589      -46.519549739351945    26.296352597935158
13.61331269030794      24.103338628268034     -241.71459326787624    -12.838251131898817
67.20842274589326      53.874826925566275     -128.57931943527078    53.987125852392204
53.8393445920769       10.15298496902507      36.37311313642064      -18.522149264771116
-30.73471596783487     -18.348873749678212    65.04033772019463      65.14909523995284
-241.5926838823518     0.19821022589470072    68.37695856135514      49.2035742711057
27.458926926126416     21.651525080649034     -19.57317557667724     129.15513858911928
-31.366414164885114    -128.5692529671494     25.94923057923502      44.40967215658775
25.90850641830091      21.638042821356397     102.71690760937591     48.54807799467204
```

Figure 4.16 Displacement of the electrons from the simulation with 10,000 particles at time step 10^{-2} sec.

The results obtained from the calculation with the different number of grids are considered. The calculation are conducted with the number of grids of 100 and with the number of grids of 50. While the system length remains the same, the width between two grids is increased when the number of grids is reduced. As a result, the fluctuation of the electric field is reduced as the field is averaged over the longer interval Δx. This effect is obviously shown in Figure 4.15 and 4.16.

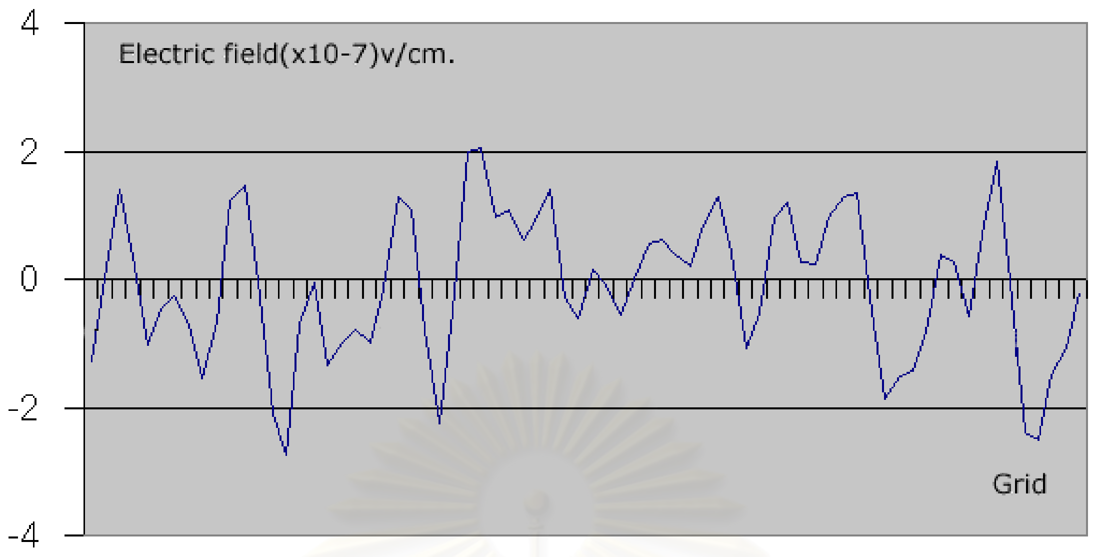


Figure 4.17 Simulations of Electric field distribution with 10,000 particles at 100 Grids

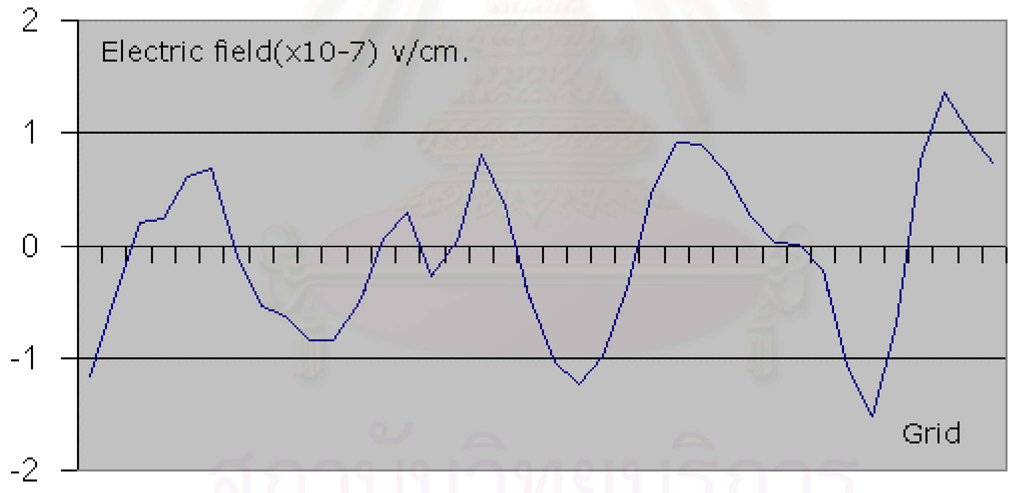


Figure 4.18 Simulation of Electric field distribution with 10,000 particles at 50 Grids

4.4 Results from simulation for the characteristics

Because the ions are much more massive than the electrons and so are considered stationary. Plasma oscillations are caused by the displacement of electrons from a uniform background of ions which electrons have distribution at each grid as shown in Figure 4.12. When the electrons are displaced, electric fields are generated due to charge separation. Due to the inertia of the electrons, they end up oscillating back and forth at a very high frequency which is called plasma frequency as seen in Figure 4.19 which is a plot of electric field at one grid versus time with the time step of 3.5×10^{-9} second. The approximately obtained frequency from graph and from theory are 5.9×10^{8} rad/sec. and 1.8×10^{8} rad/sec. , respectively.

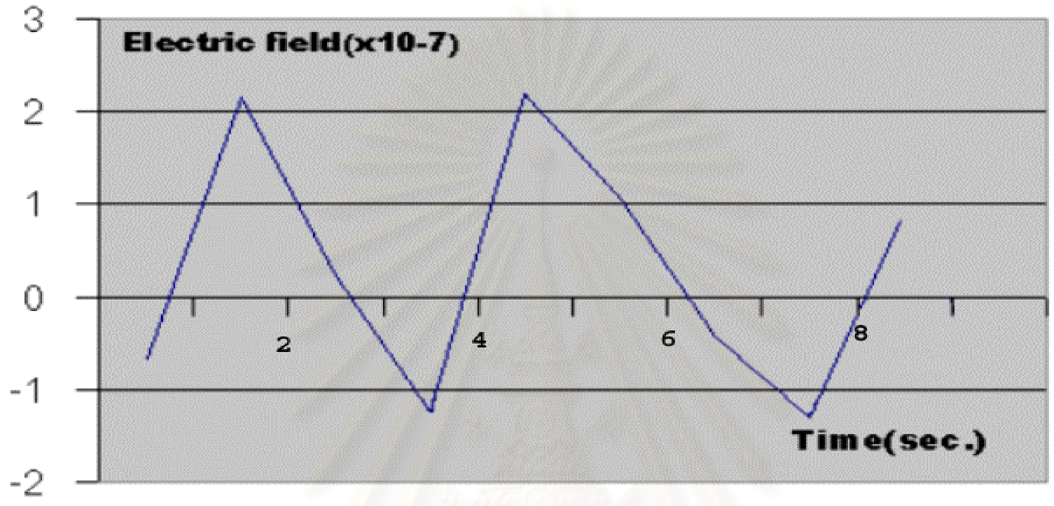


Figure 4.19 Simulations of Electric field distribution and Time at time step 3.5×10^{-9} second in one grid

In addition, the characteristic time which is determined by plasma frequency, plasma still have characteristic size which is determined by Debye length. The Debye length that is calculated from theory is 2.18 cm. and from simulation which can approximately measure by graphical technique as illustrated in Figure 4.20 is 1.89 cm. The difference of Debye length and plasma frequency of the both cause from various factors. The major difference is the reality of plasma have three dimensional characteristics but the obtained results from the simulation are done on a one dimensional plasma. The other owing to self limiting simulation of capable microcomputer and the selected technique in each step. For instance, the number of particle and the number grid these directly affect on the ability of computer including the used techniques for computer simulation, finite difference, this technique have truncation error which is decreased as Δx or grid width becomes sufficiently small.

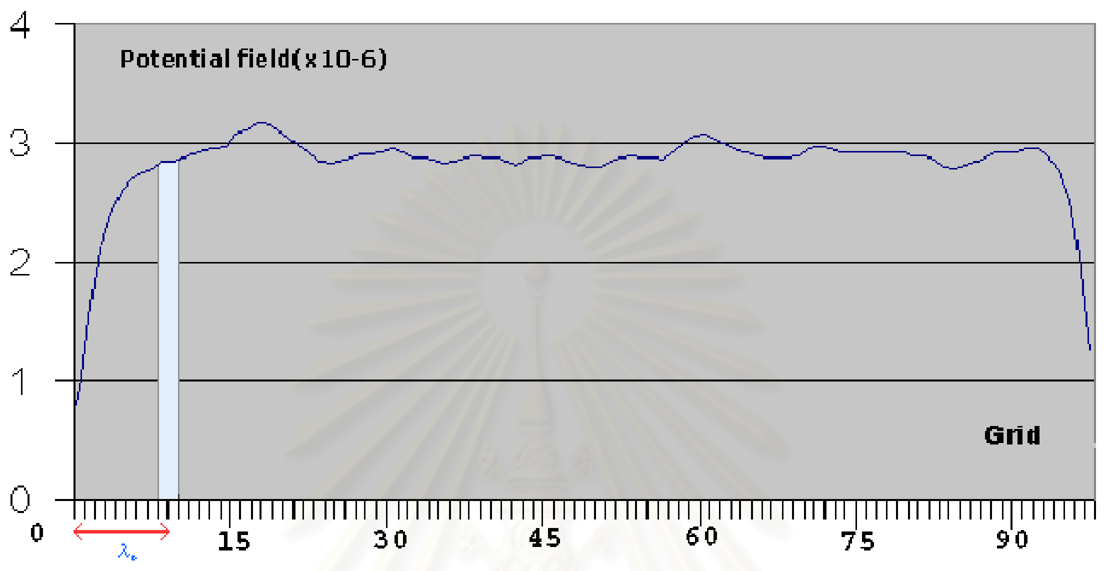


Figure 4.20 The measurement of Debye length

CHAPTER V

CONCLUSION AND SUGGESTIONS

5.1 Overview

The traditional method for obtaining a physical understanding of a physical phenomena is through the experimental approach. From the experiments, the results are interpreted in terms of the analytical relations. However, there are still some situations of interest where the experiments cannot be conducted under a given circumstance. In that case, the computer simulation is an alternative method that offers many advantages over the conventional experiments. It can be tested much more quickly while, at the same time, various physical effects can be specifically included or omitted. It also costs less than actually conducting the experiment.

5.2 Simulation in one dimensional plasma

There are a large variety of techniques for simulating plasmas but basically, they are of two types; the particle model and the fluid model. This thesis adopts the particle model which consist of calculating the charge densities on the discrete grid points using the nearest grid point (NGP), then the field is calculated from Poisson's equation and by using Lorentz equation the electric and the magnetic field are obtained. Later, by applying the calculated force on the particles, the new positions of the particles are known. This study focus on the motion of a one dimensional plasma in order to obtain various parameters, displacement, velocity and acceleration. The parameters that characterize plasma such as Debye length and plasma frequency are also estimated and compared with the theoretical values. The selected numerical technique is the finite difference with the condition of zero potential at the boundary. In addition, the inspection of the number of particles, the number of grid cells and time step are also conducted. These parameters determines the precision of the simulation.

5.3 The result of simulation

This program simulates the motion of plasma and in validating results, it also gives the primary characteristics of a plasma the Debye length and the plasma frequency. Plasma frequency is obtained from simulation and from theory as 5.9×10^8 rad/sec. and 1.8×10^8 rad/sec. , respectively.

In addition, the characteristic time which is determined by plasma frequency, plasma still has the characteristic size which is determined by Debye length. The Debye length that is calculated from theory is 2.18 cm. and from simulation, which is approximately measured by graphical technique as showed in Figure 4.20, is 1.89 cm.

The different results from the simulation and the theory take place due to various factors. The main difference is due to the fact that the actual plasma is three dimension but the obtained results from the simulation are conducted on one dimensional plasma. This error may be reduced for the simulation of the two or three dimensional plasma. The others are due to the self limit in the simulation by the capable microcomputer and the selected technique. The number of particles and the number of grids directly affect on the ability of computer. The more number of particles and the number of grids, the higher performance computer is required to cope with the larger number of equations. The techniques used in the simulation, the finite difference, also causes the truncation error, which can be decreased as Δx or grid width is reduced toward zero. As an improvement, the other numerical technique may be considered.

This study is limited for a one dimensional plasma. Though the simulation for the two and the three dimensional plasma are much more realistic, the one dimensional plasma still has its merit. This is because, for the one dimensional plasma, the simulation uses less computing resource. Since the theory of the one dimensional plasma is well developed, it is easier to test the suitability of a numerical technique against the theoretical prediction. In contrast, the two and the three dimensional simulation are very computationally intensive. Therefore the optimal result that might be selected between the time-consuming calculation and the accuracy of the obtained result may be chosen. By convention, this is compromised by limiting the resource for the simulation while still retains all the relevant physics.

5.4 Suggestions

In deed, there are numerous methods and conditions in each step for improving the simulation. Some other methods that can be further implemented are given and briefly discussed.

CIC(Cloud-in-cell) technique

In finding the charge density on a discrete grid point with NGP(Nearest grid point) technique, we simply count the number of particles within distance. By using CIC technique, the additional expenses are required for each particle. The advantage of this technique that it is more accurate than NGP technique due to the smoother distribution of the charge density. This technique employs two nearest grid points for each particle in order to deposit the fractional charge, which is proportional to the separation from each grid point as seen in Figure 5.1

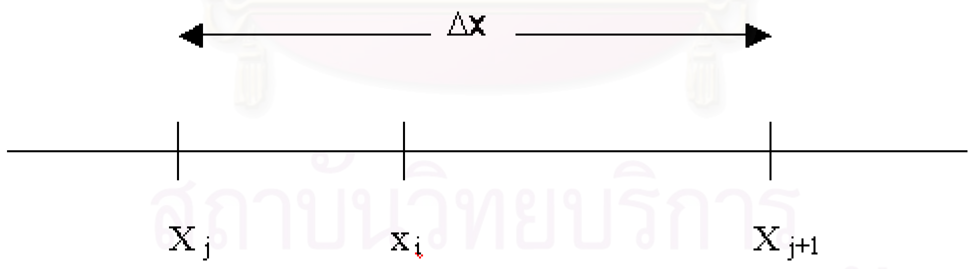


Figure 5.1 Position of particle between two grid points.

For the above figure, x_i is the position of the particle and q_c is the total charge.

Therefore, the fractions of charge particle that is assigned to j and j+1 as following

$$q_j = q_c \left[\frac{X_{j+1} - x_i}{\Delta x} \right] \tag{5.1}$$

and

$$q_{j+1} = q_c \left[\frac{x_i - X_j}{\Delta x} \right] \tag{5.2}$$

<u>Electromagnetic field</u>

Given V_x and V_y are velocity in x axis and velocity in y axis, respectively. The velocity in y axis can be obtained from velocity in x axis and the angle, theta at first time is evaluated from random generator

$$V_x = V \cos\theta \tag{5.3}$$

$$V_y = V \sin\theta \tag{5.4}$$

thus,

$$V_y = V_x \tan\theta \tag{5.5}$$

and

$$\theta = \text{random number} \times 2\pi \quad . \tag{5.6}$$

From Lorentz equation

$$F_B = qV \times B \tag{5.7}$$

the electromagnetic field in z axis can be calculated by cross product as following

$$F_B = q\, V_y \times B_z \tag{5.8}$$

$$m\, \dot{V}_x = q V_y B_z \tag{5.9}$$

$$B_z = \frac{ma_x}{qV_y} \tag{5.10}$$

<u>Possible value for one dimensional plasma</u>

For a three dimensional plasma, a particle is free to move in any direction but only the component in one specific direction, namely x-direction, is of interest. On the other hand, in a one dimensional plasma, a particle only move in one direction, x-direction. As a result, starting with the same number of particle, less particle would contribute to the motion in x-direction in the use of the three dimensional plasma. With this assumption, assume that each particle has the same drifting velocity, $\vec{v}$, the average distance a particle can move in x-direction for three dimensional over time Δt, H is half hemisphere and $\int_H d\Omega$ is for the positive direction, these are given as

$$\bar{x} = \frac{\int_H \vec{v} \cdot \hat{e}_x d\Omega}{\int_H d\Omega} \Delta t \tag{5.11}$$

$$= \frac{v\Delta t}{2\pi} \int_0^{2\pi} \int_0^{\pi/2} \hat{\Omega} \cdot \hat{e}_x \sin\theta d\theta d\phi \tag{5.12}$$

$$= v\Delta t \int_0^{\pi/2} \cos\theta \sin\theta d\theta \tag{5.13}$$

$$= \frac{1}{2} v\Delta t \tag{5.14}$$

The above relation shows that a particle contributes only half the distance it should have moved if it is a particle in a one dimensional plasma. Therefore, it is necessary that the value of n_0 is multiplied by the factor of two to compensate for this effect in order to obtain the similar result as compared with that of the three dimensional plasma.

Two dimensional Plasma

There are various problems that one dimensional plasma is not suitable. Therefore, two or three dimensional plasma must be considered. The method in two dimensional plasma is similar to one dimensional plasma but it take more calculating time and resource. The first step is to use the suitable random number to generate the position of the particles (x_i , y_i). The interesting random number is the random walk or the drunkard's walk which is used to describe the random motion at any given point in a two dimensional space. In this scheme, the motion can be equally either to the left or right or forward or backward. The principle for this random number generator is to produce random number in the range of 0 to 1. If the number is between 0 and 0.25, the direction is up , for 0.25 to 0.50 the direction is right , for 0.50 to 0.75 represent the down direction and the remaining range is equivalent the left direction The next step is to find the charge density each grid in order to supply the right hand side of Poisson's equation.

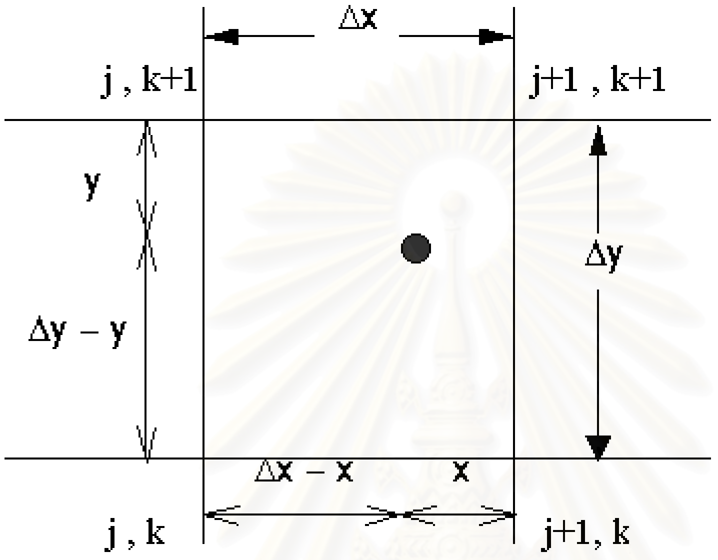


Figure 5.2 Position of a particle in the grid cell .

As seen from Figure 5.2, the weights are given by

$$\rho_{j,k} = \rho_c \frac{(\Delta x - x)(\Delta y - y)}{\Delta x \Delta y} \tag{5.15}$$

$$\rho_{j+1,k} = \rho_c \frac{x(\Delta y - y)}{\Delta x \Delta y} \tag{5.16}$$

$$\rho_{j+1,k+1} = \rho_c \frac{xy}{\Delta x \Delta y} \tag{5.17}$$

$$\rho_{j,k+1} = \rho_c \frac{(\Delta x - x)y}{\Delta x \Delta y} \tag{5.18}$$

$$\nabla^2\phi(x,y) = -\rho(x,y) \tag{5.19}$$

In finite difference form, this becomes

$$\frac{\left(\phi_{j-1} - 2\phi_j + \phi_{j+1}\right)_k}{\Delta x^2} + \frac{\left(\phi_{k-1} - 2\phi_k + \phi_{k+1}\right)_j}{\Delta y^2} = -\rho_{j,k} \tag{5.20}$$

and

$$(E_x)_{j,k} = \frac{(\phi_{j-1} - \phi_{j+1})_k}{2\Delta x} \tag{5.21}$$

$$(E_y)_{j,k} = \frac{(\phi_{k-1} - \phi_{k+1})_j}{2\Delta y} \quad . \tag{5.22}$$

Where ρ_c is the charge density uniformly filling a cell , Δx and Δy are the width of grid cell and the height of a grid cell, respectively. After obtaining the charge density, the remaining steps are similar to that of the one dimensional case. In general, we then have the following relations.

References

1. Birdsall, C. K., and Langdon, A. B. Plasma Physics via computer simulation. 1st ed. Singapore: McGraw-Hill, 1985.
2. Dendy, R. Plasma Dynamics. 1st ed. New York: Oxford University Press, 1999.
3. Tajima, T. Computational Plasma Physics with application to fusion and astrophysics . 1st ed. Singapore: Addison Wesley, 1987.
4. Smirnov, B.M. Introduction to Plasma Physics. 1st ed. Moscow: Mir, 1977.
5. Celia, M. A., and Gray, W. G. Numerical Method for Differential Equation. 1st ed. Singapore: Prentice-Hall, 1992.
6. Press, W. H., Teukoisky, S. A., Vetterling, W. T., and Flannery, B. P.Numerical Recipes in C. 2nded. New York: Cambridge University Press, 1992.
7. Rojiani, K. B. Programming in C with Numerical Methods for engineers. 1st ed. Singapore: Prentice-Hall, 1996.
8. Curtis, F., and Patrick, O. Applied Numerical Analysis. 5st ed. Singapore:Addison Wesley, 1994.

Appendix

```
/*==========================================================================================*/
/*                         Plasma simulation in one dimension program                       */
/*                              last modified     1/12/2000                                 */
/*                          By   Chawisnach   Engchatcharoen                                */
/*==========================================================================================*/

//   How   to  compile
//    javac  Thesis14.java  ➔ Thesis14.class
//    appletviewer     fileName.html
//    in   fileName.html
//     <html>
//     <applet code="Thesis14.class" width=750 height =440>
//     </applet>
//     </html>
//      recommended  Netscape or Explorer version 4.0 or later

/*
```

- JavaSoft is the divisoin of Sun Microsystem responsible for Java.
- Java's first official beta release was in November 1995.
- In OOP, program is considered to be a group of objects that interact with each other.These objects exist independently of each other since Java inherits OOP concepts from C++ and other languages such as Smalltake(pioneered OOP in the 1970s).
- Java can be used to create programs that execute from WWW pages,these programs are called Applets.
- Java program is created as text file with extension .java that is compiled into one or more files of bytecodes with the extension .class .
- Bytecodes are sets of instructions similar to the machine code instructions created when a computer program is compiled since the difference is that machine code must run on the computer system it was compiled for but bytecodes can run on any computer system equipped to handle Java programs.
- Java has been described as C++ minus because of complex parts of C++ were excluded from Java such as pointer , memory management(occurs automatically in Java).
- Java program that executes from a Web page is called an applet, all other Java programs are called applications.
- Pointer can be used to forge access to parts of a program where access is not allowed, by eliminating all pointer except for a limited from of references to objects thus Java is a much more secure language.
- Platform independence is the ability of the same program to work on different operating systems therefore Java's variable types have the same size across all Java development platforms so an integer is always the same size on matter which system a Java program was written and compiled on.
- Java Developers Kit (JDK) is a set of command line tools that can be used to create Java programs which includes compiler , interpreter to run compiled Java standalone applications, applet viewer to run Java applets ,sample code for developing Java program and other utilities.
- Java Application Programming Interface (API) is set of classes used to develop Java programs as these classes are organized into groups called packages
- Java API includes enough functionality to create sophisticated applets and applications and it must be supported by all operating systems and Web software .
- JavaOs is a compact operating systems intended to run Java programs and used to embedded on processor chips in appliances
- Load JDK at www.javasoft.com/
- Applet demos are interesting examples of Java applets which all come with complete source code.
- Java runtime interpreter is stand-alone version of Java interpreter built into HotJava browser which acts as a command line tool for running nongraphical Java programs
- Class is a template for multiple objects with similar features.
- When you write a program in OOP ,you don't define individual objects, instead you define class of objects.
- Objects are made up of many kinds of smaller objects.
- Term instance and object often are used interchangeably in OOP.
- Generally, every class you write in Java is made up of two components
- # Attributes are the individual things that differentiate one object from another and determine the appearance, state, or
- other qualities of that object.
- # Behavior is the only way objects can do anything to themselves in order to define an object's behavior, you create methods that is just like function in other language but it is defined inside classes unlike C++,Java does not have function of any kind defined outside of classes.
- Java header and stub file generator (javah) is used to generate C header and source files for implementing Java methods in C.
- Java has single inheritance that each class can have only one superclass although any superclass can have many multiple subclasses, unlike C++, class can has more than one superclass called multiple inheritance.
- HotJava is Web browser that not only support Java applets but also was written using Java language.

- Applet is Java program that executes on a World Wide Web page as be a part of Web page by attach Java applet to HTML page using two HTML tags <APPLET> and <PARAM >.This HTML code is included on a Web page along with all other HTML code like putting a picture on web page.
- The browser acts as the operating system for applets, you cann't run an applet as a standalone program in the same way you can run an executable file.
- Once the applet is written and compiled then add it to HTML pages , if the applet makes use of class files that are not part of the standard Java API ,these class files must be located in the same directory as the applet's class file.
- Abstract Windowing Toolkit (AWT) is set of classes used to build a graphical user interface for Java application and applet , it also includes classes to handle fonts, mouse click, keyboard input, and so on.
- AWT is one of useful packages included with JavaAPI that comes with JDK.
- Numerous ways to develop GUI for Java programs such Java Workshop, Jfactory ,etc or even using AWT .
-

```
    */
import java.awt.*;
import java.util.*;
import java.applet.*;
import java.awt.event.*;


public class Thesis14 extends Applet implements ActionListener, MouseListener,
                            MouseMotionListener,Runnable
{

        RECT3D  rect3d = new RECT3D();
        PLASMA  plasma = new PLASMA();


        Date      time1,time2;

        Label     titleNameL,enterPasswordL;
        Thread    moveGraph1,moveGraph2;
        String    password;
        Button    okButton;
        Graphics  g;
        TextField  enterPasswordTF, dtTF, numTimeStepTF, numGridTF, numParticleTF,
                       temperatureTF, densityTF;

        double    temperature;
        int       numTimeStep, numGrid, numParticle;
        long      startTime1,endTime1,startTime2,endTime2;
        boolean   firstPage = true, callMouseMoveTaskbar = true,
                       callMouseMoveAtPlasmaSimulation = true ,callMouseMoveButton = true,
            callAllButton = true, callEnterInput = true ,
                       paintFrame = true, nullPage = true, paintAnimaGraphPhi = true,
            paintAnimaGraphCharge = true;

        Image graphTempDensPic,definitionPic;

        Color myBlue    = new Color(150,150,200);
        Color myGray    = new Color(100,100,100);
        Color myCyan    = new Color(201,230,241);
        Color myYellow  = new Color(248,252,84);
        Color myGreen1  = new Color(214,241,201);
        Color myGreen2  = new Color(204,255,153);
        Color myCyan2   = new Color(103,235,250);
        Color myOrange1 = new Color(251,162,102);
        Color myGreen3  = new Color(9,255,163);
        Color myViolet  = new Color(217,142,244);


        Font f35cu = new Font("Courier", Font.BOLD+Font.ITALIC, 35);
        Font f20cu = new Font("Courier", Font.BOLD, 20);
        Font f10co = new Font("Cordia",  Font.PLAIN,10);
        Font f11co = new Font("Cordia",  Font.PLAIN,11);
        Font f12co = new Font("Cordia",  Font.PLAIN,12);
        Font f13coB= new Font("Cordia",  Font.PLAIN+Font.BOLD,13);


        public void init()
        {
                setLayout(null);


                titleNameL = new Label("Plasma Simulation in One Dimension");
                titleNameL.setFont(f35cu);
                titleNameL.setBackground(myBlue);
                titleNameL.reshape(10,40,725,60);
                add(titleNameL);

                enterPasswordL = new Label("Enter Password");
                enterPasswordL.setFont(f20cu);
                enterPasswordL.setBackground(myCyan);
                enterPasswordL.reshape(285,200,172,27);
                add(enterPasswordL);

                enterPasswordTF = new TextField(15);
                password = "";
                enterPasswordTF.setEchoChar('*');
                enterPasswordTF.reshape(295,260,150,20);
                add(enterPasswordTF);
                enterPasswordTF.addActionListener(this);


                this.addMouseListener(this);
                this.addMouseMotionListener(this);


                //{{INIT_CONTROLS
                //}}
        }

        public void paint(Graphics g)
        {
          if(firstPage)
```

```
{
		setBackground(myCyan);

		/*--------Draw Plasma simulation profile--------*/

		g.setColor(Color.black);
		g.drawLine(9,100,735,100);    // hor line
		g.drawLine(735,39,735,100);   // ver line

		g.setColor(Color.white);
		g.drawLine(9,39,736,39);      // hor line
		g.drawLine(9,39,9,100);       // ver line

		g.setColor(myBlue);
		rect3d.paintRect(g,230,170,280,180);  // paint rectangular of enter password


		/*----------------------Times----------------------*/
		/*
		while (firstPage)
		{
		  clock.paint(g,320,380);      // call paint in CLOCK class

		  repaint();
		}		*/
}

if(!nullPage)
{
		setBackground(Color.white);

}

if(!paintFrame)
{
		g = getGraphics();

		g.setColor(myCyan2);
   rect3d.paintRect(g,10,10,727,389);

		g.setColor(Color.lightGray);
		rect3d.paintRect(g,50,35,650,340);

		g.setColor(Color.black);
		g.drawLine(65,50,685,50);    // hor line
		g.drawLine(65,50,65,330);    // ver line

		g.setColor(Color.white);
		g.drawLine(65,330,685,330);    // hor line
		g.drawLine(685,50,685,330);    // ver line
```

```
}

if(!paintAnimaGraphPhi)
{
            plasma.debye(numParticle,temperature);
            plasma.generateParticle(numParticle,numGrid);
            plasma.adjustParticle(numParticle, numGrid);
            plasma.assignCharge(numParticle, numGrid);
            plasma.computeField(numParticle,numGrid);
            plasma.pushParticle(numParticle,numGrid);
            double   yPosition, xPosition = 0.0;
   double   increase  = 0.0;
            g.setColor(myGreen1);
            g.fillRect(66,51,619,279);
            g.setColor(Color.blue);

            g.setFont(f12co);
            g.drawString("Potential Field Distribution",75,65);
            g.drawString("Time step = "+plasma.dt*numTimeStep+" "+"sec.",75,80);
            g.drawString("X-axis : Grid",75,95);
            g.drawString("Y-axis : Potential at Grid",75,115);
            g.setColor(Color.black);

            for (int i =0; i< numGrid; i++)
            {
                        increase  = ( 692 / (numGrid-1) );
                        xPosition = xPosition + increase ;
                        yPosition = plasma.phi[i]*100/10e-6;
                        g.fillOval(70+(int)xPosition,100+195-(int)yPosition,3,3) ;
             }
}

if(!paintAnimaGraphCharge)
{
     plasma.debye(numParticle,temperature);
              plasma.generateParticle(numParticle,numGrid);
              plasma.adjustParticle(numParticle, numGrid);
              plasma.assignCharge(numParticle, numGrid);
              plasma.computeField(numParticle,numGrid);
              plasma.pushParticle(numParticle,numGrid);

     g.setColor(myViolet);
                             g.fillRect(66,51,619,279);
                             g.setColor(Color.black);
                             g.setFont(f12co);
                             g.drawString("Charge Density Distribution",75,65);
              g.drawString("Time step = "+plasma.dt*numTimeStep+" "+"sec.",75,80);
              g.drawString("X-axis : Grid",75,95);
              g.drawString("Y-axis : Charge Density at Grid",75,115);
              g.setColor(Color.white);
```

```
                    double add ,xPos=0.0,yPos=0.0;
                    for (int i =0; i< numGrid; i++)
                    {

                                add  = ( 690 / (numGrid-1) );
                                xPos = xPos + add ;
                                yPos = plasma.rho[i]*250/10e-17;
                                g.fillOval(70+(int)xPos,250+80-(int)yPos,3,3) ;
                    }
 }

}

public void start()
{
 if(!paintAnimaGraphPhi)
 {
             if(moveGraph1 == null)
             {
                         moveGraph1 = new Thread(this);
                         moveGraph1.start();
             }
 }
 if(!paintAnimaGraphCharge)
 {
    if(moveGraph2 == null)
             {
                         moveGraph2 = new Thread(this);
                         moveGraph2.start();
             }
 }
}

public void stop()
{
            if(moveGraph1 != null)
            {
                         moveGraph1.stop();
      moveGraph1 = null;
            }
            if(moveGraph2 != null)
            {
                         moveGraph2.stop();
      moveGraph2 = null;
            }
}

public void update(Graphics g)
{
```

```
                        paint(g);
}

public void run()
{
  if(!paintAnimaGraphPhi)
  {
            do
            {
              Date time1 = new Date();
                     endTime1   = (time1.getTime())/1000;

            try
                     {
                              Thread.sleep(1000);
                     }
                     catch(InterruptedException e)
                     {
                     }
                     repaint();
             }
             while( (endTime1 - startTime1) != numTimeStep );
             {
               moveGraph1 = null;

             }
  }

  if(!paintAnimaGraphCharge)
  {
     do
            {
              Date time2 = new Date();
                     endTime2   = (time2.getTime())/1000;

            try
                     {
                              Thread.sleep(1000);
                     }
                     catch(InterruptedException e)
                     {
                     }
                     repaint();
             }
             while( (endTime2 - startTime2) != numTimeStep );
             {
               moveGraph2 = null;

             }
  }
```

```
}

public void actionPerformed(ActionEvent event)
{
            String str,str1;
   Object obj;
            obj  = event.getSource();
            str  = event.getActionCommand();
            str1 = event.getActionCommand();


            if(obj == enterPasswordTF && str.equals(password))
            {
                        titleNameL.setVisible(false);
               enterPasswordL.setVisible(false);
                        enterPasswordTF.setVisible(false);

                        firstPage = false;                // close
                        nullPage  = false;                // open
                        callMouseMoveTaskbar = false;     // open

                        repaint();
            }


            if (numTimeStepTF != null)
            {
               str1 = numTimeStepTF.getText();
               numTimeStep = Integer.valueOf(str1).intValue();
            }

            if (numGridTF != null )
            {
                str1 = numGridTF.getText();
                numGrid = Integer.valueOf(str1).intValue();

            }

            if( numParticleTF != null )
            {
                str1  = numParticleTF.getText();
                numParticle = Integer.valueOf(str1).intValue();
            }


            if(temperatureTF != null)

            {
                        str1 = temperatureTF.getText();
               temperature = Double.valueOf(str1).doubleValue();
            }

            if( str.equals("O.K."))
            {

                plasma.debye(numParticle,temperature);
                plasma.generateParticle(numParticle,numGrid);
                            plasma.advanceSimulation(numParticle,numGrid,numTimeStep);

                            okButton.setVisible(false);
                            numGridTF.setVisible(false);
                            numParticleTF.setVisible(false);
                            numTimeStepTF.setVisible(false);
                            temperatureTF.setVisible(false);

                            repaint();
            }

}

public void mousePressed(MouseEvent event)
{
            int  xx,yy;

               g = getGraphics();

              xx = event.getX();
              yy = event.getY();

              /*---------------------Press Start---------------------*/

              if ( (xx > 10) && (xx < 65) && (yy > 400) && (yy < 425) )
              {
                          callMouseMoveButton = true;                    // close
                             callMouseMoveTaskbar = true;                   // close
                             callMouseMoveAtPlasmaSimulation = false;       // open

                                        g.setColor(Color.lightGray);
                 rect3d.paintRect(g,10,220,140,178);

                                        g.setColor(Color.black);
                                        g.setFont(f12co);
                          g.drawString("Plasma Simulation ",30,240);
                                        g.drawString("Plasma  Theoy", 38,270);
                                        g.drawString("Plasma Condition",34,300);
                                        g.drawString("Help",65,330);
                                        g.drawString("Others",60,360);
                                        g.drawString("Clear",63,390);

              /*------------Draw white line between message----------*/
```

```
                        g.setColor(Color.white);
                        g.drawLine(10,250,149,250);
                        g.drawLine(10,280,149,280);
                        g.drawLine(10,310,149,310);
                        g.drawLine(10,340,149,340);
                        g.drawLine(10,370,149,370);
}
/*---------------- Press Plasma Condition ----------------*/
if ((xx > 10) && (xx < 149) && (yy > 280) && (yy < 310))
{
        graphTempDensPic = getImage(getCodeBase(),"graphTempDens.jpg");
        g.drawImage(graphTempDensPic,0,0,this);
        callMouseMoveAtPlasmaSimulation = true;
}
/*---------------- Press Bounded Plasma ----------------*/

if ((xx > 155) && (xx < 290) && (yy > 190) && (yy < 220))
{
                paintFrame = false;                     // open
                callAllButton = false;                  // open
                callMouseMoveButton = false;            // open
                callMouseMoveTaskbar = false;           // open
                callMouseMoveAtPlasmaSimulation = true;   // close
                repaint();

}
/*---------------------- Press Clear --------------------*/

if ((xx > 10) && (xx < 149) && (yy > 370) && (yy < 400))
{

              nullPage  = false;                    // open
              paintFrame = true;                    // close
              callMouseMoveButton  = true;          // close
              callMouseMoveTaskbar = false;         // open
              callMouseMoveAtPlasmaSimulation = true; // close

              g.setColor(getBackground());
              Dimension size = getSize();
              g.fillRect(0,0,size.width,size.height);
              //paint(g);

              okButton.setVisible(false);
              dtTF.setVisible(false);
              numGridTF.setVisible(false);
              densityTF.setVisible(false);
              numParticleTF.setVisible(false);
              numTimeStepTF.setVisible(false);
              temperatureTF.setVisible(false);
```

```
                    paintAnimaGraphPhi = true;      // close
                    moveGraph1.stop();



                    paintAnimaGraphCharge = true;    // close
                    moveGraph2.stop();

                    repaint();
            }

if(!callAllButton)
{
            /*----------------------Press Input---------------------*/

            if( (xx > 67) && (xx < 105) && (yy > 345) && (yy < 365))
            {
                     callEnterInput = false;          // open
                        enterInputPanel();

            }

            /*--------------Press Particle Density Button --------------*/

            if( (xx > 110) && (xx < 190) && (yy > 345) && (yy < 365) )
            {
                        //g.setColor(myGreen3);
                        g.setColor(Color.white);
                                g.fillRect(66,51,619,279);
                                g.setColor(Color.black);
                                g.setFont(f12co);

                                int k = 64, z = 80;

                                for(int i=0; i<100; i++)
                                {
                                        g.drawString(String.valueOf(plasma.countParticle[i]),z,k);
                                        if (z == 530)
                                        {
                                                z  = -70;
                                                k += 11;
                                        }
                                        z += 150;
                        }
            }
            /*--------------Press Plasma Parameters Button -----------*/

            if( (xx > 193) && (xx < 288) && (yy > 345) && (yy < 365) )
            {
                        //g.setColor(myCyan);
```

```
                g.setColor(Color.white);
                        g.fillRect(66,51,619,279);

                        g.setColor(Color.black);
                        g.setFont(f13coB);
                        g.drawString("Plasma Parameters",310,80);
                        g.setFont(f12co);
                        g.drawString("dt ="+" "+plasma.dt+"  "+"sec",150,100);

                        g.drawString("Grid Wide  ="+" "+plasma.dx,150,130);
                        g.drawString("Grid Numbers  ="+" "+numGrid,150,160);
                        g.drawString("Debye Length  ="+" "+plasma.db*100.0+"  "+"cm",150,190);
                g.drawString("System Length ="+" "+plasma.systemLength+" "+"cm",150,220);
                g.drawString("Temperature (K)  ="+" "+temperature,150,250);
                        g.drawString("Particles Number="+" "+numParticle,150,280);
                                g.drawString("Plasma     Frequency     ="+"     "+plasma.frequency     +"
"+"rad/sec",150,310);


        }

        /*-----------------Press Position Button ----------------*/

        if( (xx > 290) && (xx < 338) && (yy > 345) && (yy < 365) )
        {
            //g.setColor(myGreen2);
            g.setColor(Color.white);
                        g.fillRect(66,51,619,279);
                        g.setColor(Color.black);
                        g.setFont(f12co);
                        int k = 64, z = 80;

                        for(int i=0; i<100; i++)
                        {
                                g.drawString(String.valueOf(plasma.pos[i]),z,k);
                                if (z == 530)
                                {
                                        z  = -70;
                                        k += 11;
                                }
                                z += 150;
                }
        }

        /*----------------- Press Potential Button --------------*/

        if( (xx > 340) && (xx < 388) && (yy > 345) && (yy < 365) )
        {
                //g.setColor(myOrange1);
                g.setColor(Color.white);
```

```
                    g.fillRect(66,51,619,279);
                    g.setColor(Color.black);
                    g.setFont(f12co);
                    int k = 64, z = 80;

                    for(int i=0; i<100; i++)
                    {
                            g.drawString(String.valueOf(plasma.phi[i]),z,k);
                            if (z == 530)
                            {
                                    z  = -70;
                                    k += 11;
                            }
                            z += 150;
                    }

}
/*---------------Press Electric field Button --------------*/

if( (xx > 390) && (xx < 438) && (yy > 345) && (yy < 365) )
{
            //g.setColor(myYellow);
            g.setColor(Color.white);
                    g.fillRect(66,51,619,279);
                    g.setColor(Color.black);
                    g.setFont(f12co);
                    int k = 64, z = 80;

                    for(int i=0; i<100; i++)
                    {
                            g.drawString(String.valueOf(plasma.ele[i]),z,k);
                            if (z == 530)
                            {
                                    z  = -70;
                                    k += 11;
                            }
                            z += 150;
            }
}

/*--------------Press Potential field Button --------------*/

if( (xx > 440) && (xx < 558) && (yy > 345) && (yy < 365) )
{
        paintAnimaGraphPhi    = false;    // open
        paintAnimaGraphCharge = true;     // close
        start();
        moveGraph1.resume();

        time1 = new Date();
```

```
                startTime1 = (time1.getTime())/1000;

        }
        /*-----------Press Charge Density Distribution Button -----------*/


        if( (xx > 560) && (xx < 683) && (yy > 345) && (yy < 365) )
        {

           paintAnimaGraphCharge = false;    // open
           paintAnimaGraphPhi    = true;     // close
                start();
                moveGraph2.resume();

                time2 = new Date();
                startTime2 = (time2.getTime())/1000;
        }
  }
}

public void mouseMoved(MouseEvent event)
{
       int  xx,yy;

        g  = getGraphics();
        xx = event.getX();
        yy = event.getY();


       if(!callMouseMoveTaskbar)
       {

        if ( (xx > 10) && (xx < 700) && (yy > 400) && (yy < 425) )
        {

                      g.setColor(Color.lightGray);
           rect3d.paintRect(g,10,400,55,25);       // draw start button
           rect3d.paintRect(g,67,400,670,25);       // draw taskbar

           g.setColor(Color.black);
           g.setFont(f12co);
           g.drawString("Start",25,418);
        }
        else
        {
                g.setColor(Color.white);
                rect3d.deleteRect(g,10,400,727,25);
        }
       }
```

```
if(!callMouseMoveAtPlasmaSimulation)
{
  if ( (xx > 10) && (xx < 150) && (yy > 220) && (yy < 250) )
  {
                 g.setColor(Color.lightGray);
      rect3d.paintRect(g,152,190,140,60);
                         g.setColor(Color.black);
                         g.setFont(f12co);
                         g.drawString("One Dimension",175,210);
                         //g.drawString("",175,240);
                         g.setColor(Color.white);
                         g.drawLine(152,220,290,220);
  }

}

if(!callMouseMoveButton)
{
             /*------------------Build Input Button------------------*/

   if( (xx > 67) && (xx < 105) && (yy > 345) && (yy < 365) )
             {
               g.setColor(Color.lightGray);
               rect3d.paintRect(g,67,344,40,20);
               g.setFont(f11co);
               g.setColor(Color.black);
               g.drawString("Input",76,359);
             }
             else
             {
               g.setColor(Color.lightGray);
               rect3d.deleteRect(g,67,344,40,20);
               g.setFont(f11co);
               g.setColor(Color.black);
               g.drawString("Input",76,359);
             }

             /*------------ Build Particle Density Button ---------------*/

             if( (xx > 110) && (xx < 190) && (yy > 345) && (yy < 365) )
             {
               g.setColor(Color.lightGray);
               rect3d.paintRect(g,110,344,80,20);
               g.setFont(f11co);
               g.setColor(Color.black);
               g.drawString("Particle Density",113,359);
             }
             else
             {
               g.setColor(Color.lightGray);
  rect3d.deleteRect(g,110,344,80,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Particle Density",113,359);
}

/*------------ Build Plasma Parameters Button ------------*/

if( (xx > 193) && (xx < 288) && (yy > 345) && (yy < 365) )
{
  g.setColor(Color.lightGray);
  rect3d.paintRect(g,193,344,95,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Plasma Parameters",195,359);
}
else
{
  g.setColor(Color.lightGray);
  rect3d.deleteRect(g,193,344,95,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Plasma Parameters",195,359);
}

/*---------------- Build Position Button ----------------*/

if( (xx > 290) && (xx < 338) && (yy > 345) && (yy < 365) )
{
  g.setColor(Color.lightGray);
  rect3d.paintRect(g,290,344,48,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Position",296,359);
}
else
{
  g.setColor(Color.lightGray);
  rect3d.deleteRect(g,290,344,48,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Position",296,359);
}

/*----------------- Build Potential Button ----------------*/

if( (xx > 340) && (xx < 388) && (yy > 345) && (yy < 365) )
{
  g.setColor(Color.lightGray);
  rect3d.paintRect(g,340,344,48,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Potential",343,359);
}
else
{
  g.setColor(Color.lightGray);
  rect3d.deleteRect(g,340,344,48,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Potential",343,359);
}

/*----------------- Build Potential Button ----------------*/

if( (xx > 390) && (xx < 438) && (yy > 345) && (yy < 365) )
{
  g.setColor(Color.lightGray);
  rect3d.paintRect(g,390,344,48,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Electric",398,359);
}
else
{
  g.setColor(Color.lightGray);
  rect3d.deleteRect(g,390,344,48,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Electric",398,359);
}

/*--------------- Build Potential field Button --------------*/

if( (xx > 440) && (xx < 558) && (yy > 345) && (yy < 365) )
{
  g.setColor(Color.lightGray);
  rect3d.paintRect(g,440,344,118,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Potential Distribution",452,359);
}
else
{
  g.setColor(Color.lightGray);
  rect3d.deleteRect(g,440,344,118,20);
  g.setFont(f11co);
  g.setColor(Color.black);
  g.drawString("Potential Distribution",452,359);
}
```

```
                /*--------- Build Charge Density Button ---------*/

                if( (xx > 560) && (xx < 683) && (yy > 345) && (yy < 365) )
                {
                  g.setColor(Color.lightGray);
                  rect3d.paintRect(g,560,344,123,20);
                  g.setFont(f11co);
                  g.setColor(Color.black);
                  g.drawString("Charge Distribution",576,359);
                }
                else
                {
                  g.setColor(Color.lightGray);
                  rect3d.deleteRect(g,560,344,123,20);
                  g.setFont(f11co);
                  g.setColor(Color.black);
                  g.drawString("Charge Distribution",576,359);
                }

        }
}

public void mouseDragged(MouseEvent event)
{

}

public void mouseReleased(MouseEvent event)
{

}

public void mouseEntered (MouseEvent event)
{

}

public void mouseExited  (MouseEvent event)
{

}

public void mouseClicked (MouseEvent event)
{

}

public void enterInputPanel()
{
```

```
g = getGraphics();

                if(!callEnterInput)
                {
                        g.setColor(Color.lightGray);
                        rect3d.paintRect(g,67,90,233,253);

                        g.setColor(Color.blue);
                  g.setFont(f11co);

                        g.drawString("Grid Number",80,130);
                        g.drawString("Particle Number",80,160);
                        g.drawString("Time Step Number ",80,190);
                        g.drawString("Plasma Temperature (K)",80,220);

                        g.setColor(Color.lightGray);
                        rect3d.paintRect(g,155,315,50,20);
                        g.setColor(Color.black);
                        g.drawString("O.K.",171,330);

                        numGridTF = new TextField(20);
                        numGridTF.reshape(200,118,85,18);
                        add(numGridTF);
                        //numGridTF.addActionListener(this);

                        numParticleTF = new TextField(20);
                        numParticleTF.reshape(200,148,85,18);
                        add(numParticleTF);
                        //numParticleTF.addActionListener(this);



                        numTimeStepTF = new TextField(20);
                        numTimeStepTF.reshape(200,178,85,18);
                        add(numTimeStepTF);
                        //numTimeStepTF.addActionListener(this);


                        temperatureTF  = new TextField(20);
                        temperatureTF.reshape(200,208,85,18);
                        add(temperatureTF);
                        //temperatureTF.addActionListener(this);

                        okButton = new Button("O.K.");
                        okButton.setFont(f10co);
                        okButton.reshape(155,315,50,20);
                        add(okButton);
                        okButton.addActionListener(this);
                }
    }
```

```
            //{{DECLARE_CONTROLS
            //}}
}

class RECT3D extends Canvas
{

            public void paintRect(Graphics g, int x,int y,int w,int L)
            {

                        g.fillRect(x,y,w,L);

                        g.setColor(Color.black);
                        g.drawLine(x, y+L, x+w, y+L);       //hor line
                        g.drawLine(x+w, y, x+w, y+L);       //ver line

                        g.setColor(new Color(220,220,220));
                        g.drawLine(x-1, y, x+w, y);         //hor line
                        g.drawLine(x-1, y, x-1, y+L);       //ver line
            }
            public void deleteRect(Graphics g, int x,int y,int w,int L)
            {
              g.fillRect(x-1, y, w+2, L+1);
            }
}

class PLASMA
{

              double  temp,dx,db,frequency,Nd,density,dt;
                      double  particle[],rho[],a[][],b[],phi[],ele[],mag[],theta[];
                      double  acc[],velx[],vely[],pos[],countParticle[],plasmaFrequency[] ;
                      int     systemLength,j;

                      public void debye(int numParticle,double temperature)
                      {
                         density = 10.0*Math.pow((double)numParticle, 3.0);
                                 temp = (8.854e-12*temperature*1.38e-23)/(density*1.6e-19*1.6e-19);
                         db   = Math.sqrt(temp);
                                 Nd   = 4.0/3.0*Math.PI*density*db*db*db;
                      }

                      public void generateParticle(int numParticle,int numGrid)
                      {
                                particle = new double[numParticle];
                                theta    = new double[numParticle];

                                for (int i =0; i<numParticle ; i++)
                                {
```

```
                    particle[i] = Math.random()*(double)(numGrid-2);
                    theta[i] = Math.random()*Math.PI*2.0;
                }
            }

            public void adjustParticle(int numParticle,int numGrid)
        {

                    for(int i=0; i<numParticle; ++i)
                    {
                     if    ( (particle[i]+0.5)  >= (double)(numGrid-2) )
                                  particle[i]  = (particle[i]+0.5) - (double)(numGrid-2);

                     else if ( (particle[i]+0.5) <= 0.0 )
                                   particle[i] = (particle[i]+0.5) + (double)(numGrid-2);
                     }
        }

            public void assignCharge(int numParticle,int numGrid)
            {

                    systemLength = (int)(db*100.0*10.0);
            dx = (double)(systemLength)/(double)(numGrid-1);
                                 rho = new double[numGrid-2];
                   countParticle = new double[numGrid-2];

                     for (int i =0; i<numParticle; i++)
                     {
                                      j  = (int)(particle[i]+0.5);
                            countParticle[j] = countParticle[j]+1.0;

                     }

                     for (int j=0; j<numGrid-2; j++)
                     {
                             rho[j] = countParticle[j];
                             rho[j] = rho[j]*1.6e-19/dx;
                     }


                     frequency =  Math.sqrt( (density*2.56e-38)/(8.854*9.1e-43) );
                                           dt =  (2.0*Math.PI)/(frequency*10.0);
            }

            public void computeField(int numParticle,int numGrid)
            {

              int num_iter = 0;
                    int max_iter = 7;
                    boolean tol_exceed = true;
```

```
double tol = 0.05;
double sum, y_old, d_old;
double PHI0 = -0.00000001, PHIL = -0.00000001;

a   = new double[numGrid-2][numGrid-2];
b   = new double[numGrid-2];
phi = new double[numGrid-2];
ele = new double[numGrid-2];

try
{
   a[0][0] = -2.0;   a[0][1] = 1.0;

   for (int i=1; i<numGrid-3; i++)
     for (int j=0; j<numGrid-2; j++)
               {
                              if (i == j)
                              {
                                             a[i][j]   = -2.0;
         a[i][j-1] =  1.0;
                                             a[i][j+1] =  1.0;
                 }
                }

   a[numGrid-3][numGrid-4] =  1.0;
   a[numGrid-3][numGrid-3] = -2.0;
}

catch(Exception e)
{
               System.out.println ("Error =" + e);
}
/*----------------------------------------------------------------*/

   b[0] = (-rho[0]*(dx*dx)/8.854e-12)-PHI0;

   for (int i=1; i<numGrid-3; i++)
                 b[i] = -rho[i]*(dx*dx)/8.854e-12;

   b[numGrid-3] = (-rho[numGrid-3]*(dx*dx)/8.854e-12)-PHIL;


/*-----------------------------------------------------------------*/

for (int i=0; i<numGrid-2; i++)
                phi[i] = b[i]/a[i][i];

while (tol_exceed && num_iter < max_iter)
{
                for (int i=0; i<numGrid-2; i++)
```

```
                        {
                                y_old = phi[i];
                  tol_exceed = false;
                                sum = b[i];

                                for (int j=0; j<numGrid-2; j++)
                                        if(i != j)
                                                sum -= a[i][j]*phi[j];

                                phi[i] = sum/a[i][i];

                                if ( Math.abs(phi[i]-y_old) > Math.abs(y_old*tol))
                                        tol_exceed = true;
                        }
                        ++num_iter;
                }

  ele[0] = (PHI0-phi[1])/(2*dx);

  for (int j=1; j<numGrid-3; j++)
    ele[j] = (phi[j-1]-phi[j+1])/(2*dx);

  ele[numGrid-3] = (phi[numGrid-4]-PHIL)/(2*dx);

}

public void pushParticle(int numParticle,int numGrid)
{

        acc   = new double[numGrid-2];
        velx  = new double[numParticle];
        vely  = new double[numParticle];
        pos   = new double[numParticle];
        mag   = new double[numGrid-2];

          for (int j=0; j<numGrid-2; j++)
            acc[j] = 1.6e-19*ele[j]/9.1e-31;


          for (int i=0; i<numParticle; i++)
          {

                  j = (int)(particle[i]+0.5);

                        velx[i] = acc[j]*(dt-0.5);
                        pos[i]  = particle[i]*dx;
                  pos[i]  = pos[i] + velx[i]*dt;

            vely[i] = velx[i]*Math.tan(theta[i]);
            mag[j] = (9.1e-31*acc[j])/(vely[i]*1.6e-19);
```

```
                        theta[i] = theta[i]+ (dt*1.6e-19*mag[j]/9.1e-31);
                  }



            }

            public void advanceSimulation(int numParticle,int numGrid,double numTimeStep)
            {
              for(int i=0; i<numTimeStep; i++)
              {

                        adjustParticle(numParticle, numGrid);
                        assignCharge(numParticle, numGrid);
                        computeField(numParticle,numGrid);
                        pushParticle(numParticle,numGrid);

              }
            }
 }



/****************************************************************************************************
 *
 * mexFunction: mexAccumulate
 *
 * Weights the particle positions to a uniform 2D rectangular grid.
 * - Particle position given is normalized to the grid spacings (Thus
 *   xphys = xgiven * dx and similarly for y).
 * - The density is measured in the number of particles per grid cell.
 *   It is weighted bilinearly.
 * - Density are defined at the corners of the grid cells.
 * - Density along the edge and corners may need correcting depending on how
 *   density is defined there (i.e. might want to x2 edge cells and x4 corner
 *   cells).
 *
 * Usage: NumInboundParticles = mexAccumulate( 'Particles', 'Density',
 *                                  NumParticles );
 *
 * For below: J, L are the number of cells in the x and y directions rspctvly
 *
 * Particles is a real full single array with 5*MaxNumParticles members
 * - Note: Any layout in matlab will do but recommmend [5 MaxNumParticles]
 *   Then Particles(1,:) are the given x coords, Particles(2,:) are the y given
 *   y coords, Particles(3,:) are the given x grid vel, Particles(4,:) are the
 *   given y grid vel, and Particles(5,:) is the z grid vel (for 2D3V sims).
 *
 * Density is a real full single (J+1) x (L+1) arrays
 * - Note: This must be strictly adhered to!
 *
 * NumParticles is the number of active particles in the Particles array
 *
 * Upon return all particles which went out of bounds are located at
 * Particles(:,NumInbounds+1:NumParticles) (assuming [5 MaxNumParticles]
 * allocation was used). Particles(:,1:NumInbounds) are all inbounds.
 *
 * Warning: This function accesses and modifies arrays in the caller's
 * workspace! This was done to avoid the overhead for the rhs copy
 * mechanisms. This can have weird side effects (in particular if the
 * copy count an altered array is greater than one, the memoryless
 * copies will be altered - the API has no way to avoid this that I know of).
 *
 * Warning: Routine does no type checking. If you mess up the arguments or
 * they are the incorrect sizes and types, you are screwed bad.
 *
 * Warning: All arrays passed by name to this function must be allocated before
 * calling this function.
 *
 *******************************************************************************************************/

#include <string.h>
#include <mex.h>

#include "../mexCommon.h"

static int Accumulate( ParticleData *ptrParticle, float *ptrDensity,
                  int N, int J, int L );

/*******************************************************************************************************/

#define mNumInbounds plhs[0]
#define mParticles prhs[0]
#define mDensity prhs[1]
#define mN prhs[2]

void mexFunction( int nlhs, mxArray *plhs[],
              int nrhs, const mxArray *prhs[] )
  {
   ParticleData *ptrParticle;
   float *ptrDensity;
   int N, J, L;
   const mxArray *mx;

   if( mexIsLocked() )
     {
      if( ( nrhs == 0 ) && ( nlhs == 0 ) )
        {
         mexUnlock();
         return;
        }
```

```
    }
   else
    {
     mexLock();
    }

   /* Parse the input */

   GetMatrix( mParticles, mx, ptrParticle );
   GetMatrix( mDensity, mx, ptrDensity ); J = mxGetM(mx)-1; L = mxGetN(mx)-1;
   N = mxGetScalar( mN );

   /* Do the calculation */

   N = Accumulate( ptrParticle, ptrDensity, N, J, L );

   /* Store the output */

   mxSetScalar( mNumInbounds, N );
   mexAddFlops( 14*N );
  }

#undef mNumInbounds
#undef mParticles
#undef mDensity
#undef mN

/**************************************************************************************************/

int Accumulate( ParticleData *Particle, float *Density,
            int N, int Nx, int Ny )
  {
   int i, j, n;
   float dx, dy;
   ParticleData *LastParticle;
   ParticleData ptmp;

#ifdef GCC_X86_ROUNDING_HACK
   FPUCW oldcw;
   oldcw = SetRoundingMode( FPU_RC_DOWN );
#endif

   Nx++; /* Number of grid points along x */
   Ny++; /* Number of grid points along y */
   LastParticle = &Particle[N-1];

   memset(Density,0,Nx*Ny*sizeof(float));

   for( n = 0; n < N; n++, Particle++ )
    {
      i = FloatToInt( Particle->x );
      j = FloatToInt( Particle->y );
#ifdef GCC_X86_ROUNDING_HACK /* Round mode is towards -Infinity */
      if( (i >= 0) && (j >= 0) && (i < (Nx-1)) && (j < (Ny-1)) )
#else /* Round mode is truncate */
      if( (Particle->x >= 0.0) && (Particle->y >= 0.0) &&
         (i < (Nx-1)) && (j < (Ny-1)) )
#endif
        {
         dx = Particle->x - i;
         dy = Particle->y - j;
         Density[j*Nx+i] += (1-dx)*(1-dy);
         Density[j*Nx+i+1] += dx*(1-dy);
         Density[j*Nx+i+Nx] += (1-dx)*dy;
         Density[j*Nx+i+Nx+1] += dx*dy;
        }
      else
        {
         ptmp.x = Particle->x; ptmp.y = Particle->x;
         ptmp.vx = Particle->vx; ptmp.vy = Particle->vy;
         ptmp.vz = Particle->vz;
         Particle->x = LastParticle->x; Particle->y = LastParticle->y;
         Particle->vx = LastParticle->vx; Particle->vy = LastParticle->vy;
         Particle->vz = LastParticle->vz;
         LastParticle->x = ptmp.x; LastParticle->y = ptmp.y;
         LastParticle->vx = ptmp.vx; LastParticle->vy = ptmp.vy;
         LastParticle->vz = ptmp.vz;
         N--; n--; Particle--; LastParticle--;
        }
     }
#ifdef GCC_X86_ROUNDING_HACK
   SetFPUControlWord( oldcw );
#endif
   return( N );
  }
/***************************************************************************************************/

/***************************************************************************************************
 *
 * mexFunction: mexNegGrad
 *
 * Solves E = -grad PHI on a uniform 2D rectangular mesh. For the interior
 * points, central differencing is used (second order). That is:
 * Ex(i,j) = -( PHI(i+1,j) - PHI(i-1,j) ) / dx
 * Ey(i,j) = -( PHI(i,j+1) - PHI(i,j-1) ) / dy
 * On the edges, second order forward / backward differencing is used.
 *
 * Usage: mexNegGrad( 'Ex', 'Ey', 'PHI', deltax, deltay );
 *
 * Ex, Ey and PHI are all real full single precision arrays of the same
 * dimension. The arrays must be at least 3x3 in size.
 * deltax and deltay are double scalars
 *
 * Data in PHI in the caller's workspace is unchanged.
 * Data in Ex and Ey in the caller's workspace is replaced.
 *
 * Ex is the x-component of the electric field
 * Ey is the y-component of the electric field
 * PHI is the Potential
 * deltax and deltay are the x and y grid spacing.
 *
 * This routine locks itself in memory
 * If called with no rhs and lhs arguments, the routine is unlocked from memory
 *
 * Warning: This function accesses and modifies arrays in the caller's
 * workspace! This was done to avoid the overhead for the rhs copy
 * mechanisms. This can have weird side effects (in particular if the
 * copy count an altered array is greater than one, the memoryless
 * copies will be altered - the API has no way to avoid this that I know of).
 *
 * Warning: Routine does no type checking. If you mess up the arguments or
 * they are the incorrect sizes and types you are screwed bad.
 *
 * Warning: All arrays passed by name to this function must be allocated before
 * calling this function.
 *
 *****************************************************************************************************/

#include <mex.h>
#include "../mexCommon.h"

static void SolveElectricFields( double *Ex, double *Ey, double *PHI,
                         double deltax, double deltay,
                         int J, int L );

/*****************************************************************************************************/

#define mEx prhs[0]
#define mEy prhs[1]
#define mPHI prhs[2]
#define mdeltax prhs[3]
#define mdeltay prhs[4]

void mexFunction( int nlhs, mxArray *plhs[],
              int nrhs, const mxArray *prhs[] )
  {
   double *Ex, *Ey, *PHI;
   int J, L;
   double deltax, deltay;
   const mxArray *mx;
   if( mexIsLocked() )
     {
      if( ( nrhs == 0 ) && ( nlhs == 0 ) )
        {
          mexUnlock();
          return;
        }
     }
   else
     {
      mexLock();
     }

   GetMatrix( mEx, mx, Ex ); J = mxGetM(mx)-1; L = mxGetN(mx)-1;
   GetMatrix( mEy, mx, Ey );
   GetMatrix( mPHI, mx, PHI );
   deltax = mxGetScalar( mdeltax );
   deltay = mxGetScalar( mdeltay );

   SolveElectricFields( Ex, Ey, PHI, deltax, deltay, J, L );

   mexAddFlops( 6*L*J + 11*L + 11*J );
  }

#undef mEx
#undef mEy
#undef mPHI
#undef mdeltax
#undef mdeltay
#undef GetMatrix

/**********************************************************************************************/


#define PHI(j,l) PHI[ (l)*NgridsX + (j) ]
#define Ex(j,l) Ex[ (l)*NgridsX + (j) ]
#define Ey(j,l) Ey[ (l)*NgridsX + (j) ]

void SolveElectricFields( double *Ex, double *Ey, double *PHI,
                    double deltax, double deltay, int J, int L )
  {
   int j, l, NgridsX;
   double tmp;
   double *ptr1, *ptr2, *ptr3;

   NgridsX = J+1;

   /* Solve for Ex except on the left and right edges */
   tmp = -0.5 / deltax;
   for( l = 0; l <= L; l++ )
    {
      ptr1 = &Ex(1,l); ptr2 = &PHI(2,l); ptr3 = &PHI(0,l);
      for( j = 1; j < J; j++, ptr1++, ptr2++, ptr3++ )
         *ptr1 = ( *ptr2 - *ptr3 )*tmp;
    }

   /* Solve for Ex on the left and right edges */

   for( l = 0; l <= L; l++ )
      Ex(0,l) = ( -PHI(2,l) + 4.0*PHI(1,l) - 3.0*PHI(0,l) )*tmp;
   for( l = 0; l <= L; l++ )
      Ex(J,l) = ( 3.0*PHI(J,l) - 4.0*PHI(J-1,l) + PHI(J-2,l) )*tmp;

   /* Solve for Ey except on the top and bottom edges */

   tmp = -0.5 / deltay;
   for( l = 1; l < L; l++ )
    {
      ptr1 = &Ey(0,l); ptr2 = &PHI(0,l+1); ptr3 = &PHI(0,l-1);
      for( j = 0; j <= J; j++, ptr1++, ptr2++, ptr3++ )
         *ptr1 = ( *ptr2 - *ptr3 )*tmp;
    }

   /* Solve for Ey on the top and bottom edges */

   for( j = 0; j <= J; j++ )
      Ey(j,0) = ( -PHI(j,2) + 4.0*PHI(j,1) - 3.0*PHI(j,0) )*tmp;
   for( j = 0; j <= J; j++ )
      Ey(j,L) = ( 3.0*PHI(j,L) - 4.0*PHI(j,L-1) + PHI(j,L-2) )*tmp;

 }

#undef PHI
#undef Ex
#undef Ey

/***************************************************************************************************/

/***************************************************************************************************

 * mexFunction: mexPush
 *
 * Integrates the nonrelativistic equations of motion for a set of particles
 * pushed by a force known on a uniform rectangular 2D grid.
 * - The force is bilinearly interpolated to the particle positions
 * - Particle position and velocity given are normalized to the grid spacings
 *   and timestep (Thus xphys = xgiven * dx, vxphys = vgiven * dx / dt and
 *   similarly for y and vy).
 * - Ex and Ey are defined at the corners of the grid cells.
 *
 * Usage: mexPush( 'Particles', 'Ex', 'Ey', 'PushAux', ...
 *              EtoDVx, EtoDVy, NumParticles );
 *
 * Assume J, L are the number of cells in the x and y directions respectively
 *
 * Particles is a real full single array with 5*MaxNumParticles members
 * - Note: Any layout in matlab will do but recommmend [5 MaxNumParticles].
 *   Then Particles(1,:) are the given x coords, Particles(2,:) are the y given
 *   y coords, Particles(3,:) are the given x grid vel, Particles(4,:) are the
 *   given y grid vel, and Particles(5,:) is the z grid vel (for 2D3V sims).
 *
 * Ex and Ey are real full (J+1) x (L+1) arrays
 * - Note: This must be strictly adhered to!
 *
 * PushAux is a SINGLE precision array with 2*(J+1)*(L+1) members. Any layout
 * in Matlab will do
 *
 * EtoDVx, EtoDVy convert the units of Ex, Ey into acceleration with respect
 * to the grid and timestep
 *
 * NumParticles is the number of active particles in the Particles array
 * - Note: All particles must be inbounds initially
 *
 * This routine locks itself into memory
 * If called with no rhs and lhs arguments, the routine is unlocked from memory
 *
 * Warning: This function accesses and modifies arrays in the caller's
 * workspace! This was done to avoid the overhead for the lhs, rhs copy
 * mechanisms. This can have weird side effects (in particular if the
 * copy count an altered array is greater than one, the memoryless
 * copies will be altered - the API has no way to avoid this that I know of).
 *
 * Particles, PushAux are altered in the callers workspace
 *
 * Warning: Routine does no type checking. If you mess up the arguments or
 * they are the incorrect sizes and types, you are screwed bad.
 *
 * Warning: All arrays passed by name to this function must be allocated before
 * calling this function.
 *
 ***************************************************************************************************/

#include <mex.h>

#include "../mexCommon.h"

static void Push( ParticleData *ptrParticle,
          double *ptrEx, double *ptrEy,
              float *ptrdVx, float *ptrdVy,
              double EtoDVx, double EtoDVy,
              int N, int J, int L );

/*************************************************************************************************/

#define mParticles prhs[0]
#define mEx prhs[1]
#define mEy prhs[2]
#define mPushAux prhs[3]
#define mEtoDVx prhs[4]
#define mEtoDVy prhs[5]
#define mN prhs[6]

void mexFunction( int nlhs, mxArray *plhs[],
              int nrhs, const mxArray *prhs[] )
 {
   ParticleData *ptrParticle;
   double *ptrEx, *ptrEy;
   double EtoDVx, EtoDVy;
   int N, J, L;
   const mxArray *mx;
   float *ptrdV;

   if( mexIsLocked() )
     {
       if( ( nrhs == 0 ) && ( nlhs == 0 ) )
         {
           mexUnlock();
           return;
         }
     }
   else
     {
       mexLock();
     }

   /* Parse the function input */

   GetMatrix( mParticles, mx, ptrParticle );
   GetMatrix( mEx, mx, ptrEx ); J = mxGetM(mx)-1; L = mxGetN(mx)-1;
   GetMatrix( mEy, mx, ptrEy );
   GetMatrix( mPushAux, mx, ptrdV );
   EtoDVx = mxGetScalar( mEtoDVx );
   EtoDVy = mxGetScalar( mEtoDVy );
   N = mxGetScalar( mN );

   /* Do the calculation */

   Push( ptrParticle, ptrEx, ptrEy, &ptrdV[0], &ptrdV[(J+1)*(L+1)],
        EtoDVx, EtoDVy, N, J, L );

   mexAddFlops( 28*N + 2*(J+1)*(L+1) );
  }

#undef mParticles
#undef mEx
#undef mEy
#undef mPushAux
#undef mEtoDVx
#undef mEtoDVy
#undef mN

/***********************************************************************************************/

void Push( ParticleData *Particle,
         double *Ex, double *Ey,
         float *dVx, float *dVy,
         double Ex2dVx, double Ey2dVy,
         int N, int Nx, int Ny )
  {
   int i, j, n;
   float dx, dy;

#ifdef GCC_X86_ROUNDING_HACK
   FPUCW oldcw;
   oldcw = SetRoundingMode( FPU_RC_ZERO );
#endif

   Nx++; /* Ncx is the number of grid points along x */
   Ny++; /* Ncy is the number of grid points along y */

   /* Scale Ex and Ey into dVx and dVy */

   for( n = 0; n < Nx*Ny; n++ )
     {
      dVx[n] = Ex[n] * Ex2dVx;
      dVy[n] = Ey[n] * Ey2dVy;
     }

   for( n = 0; n < N; n++, Particle++ )
     {
      i = FloatToInt( Particle->x );
      j = FloatToInt( Particle->y );

      /* The below trusts CSE (common sub-expression elimination) to
         optimize this. Trusting CSE makes better assembly language code
         with the GCC compiler on x86 platforms. Mileage may vary with
         other compilers and architectures! */
      dx = Particle->x - i;
      dy = Particle->y - j;
      Particle->vx +=
        (1-dx)*(1-dy)*dVx[j*Nx+i] +
           dx *(1-dy)*dVx[j*Nx+i+1] +
        (1-dx)*   dy *dVx[j*Nx+i+Nx] +
           dx *   dy *dVx[j*Nx+i+Nx+1];
      Particle->vy +=
        (1-dx)*(1-dy)*dVy[j*Nx+i] +
           dx *(1-dy)*dVy[j*Nx+i+1] +
        (1-dx)*   dy *dVy[j*Nx+i+Nx] +
           dx *   dy *dVy[j*Nx+i+Nx+1];
      Particle->x += Particle->vx;
      Particle->y += Particle->vy;
     }

#ifdef GCC_X86_ROUNDING_HACK
   SetFPUControlWord( oldcw );
#endif

  }

/*************************************************************************************************/
```

Biography

Mr. Chawisnach Engchatcharoen was born on April,16, 1970 at Amphur Bangkokyai, Bangkok Thailand. As received the Bachelor degree in Electrical Engineering from Kasetsart University in 1995. To beginning study Master degree in Nuclear Engineering at Chulalongkorn University in 1997.