Simulation of Inertial-Electrostatic Confinement of Fusion Plasma

A Project Report

submitted by

PRAVEEN GURRALA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 27, 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **Simulation of Inertial-Electrostatic Confinement of Fusion Plasma** , submitted by **Praveen Gurrala**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Harishankar Ramachandran
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 27th May 2014

ACKNOWLEDGEMENTS

I wish to first express my thanks to my research advisor Prof. Harishankar Ramachandran for his guidance and support in the completion of this thesis. His guidance has been very essential in the development of the simulation models.

I would like to thank all the professors of IIT Madras who have taught me. This work would not have been possible without the knowledge given by them.

I would also like to thank Rahul Vadaga for helping me with the software for visualizing the results of the simulation.

ABSTRACT

KEYWORDS: Inertial-Electrostatic Confinement; Monte-carlo Simulation; Plasma Simulation; Particle-in-cell

Inertial-Electrostatic Confinement (IEC) is a method of trapping ions in convergent geometries using electrostatic fields. The IEC concept is used in thermonuclear fusion reactors to improve their efficiency. The present work is aimed at modelling and simulating the operation an IEC fusion reactor.

Particles in the reactor are assumed to have both radial and azimuthal velocities, and a one-dimensional particle-in-cell model is used to track the motion of the particles in their plane of motion. Monte Carlo methods for simulating nuclear fusion, elastic scattering and charge exchange processes have been developed.

The results of the simulation show general agreement with the experiments. However, significant differences do exist. An explanation has been provided as to how the model could be improved to reconcile the differences.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

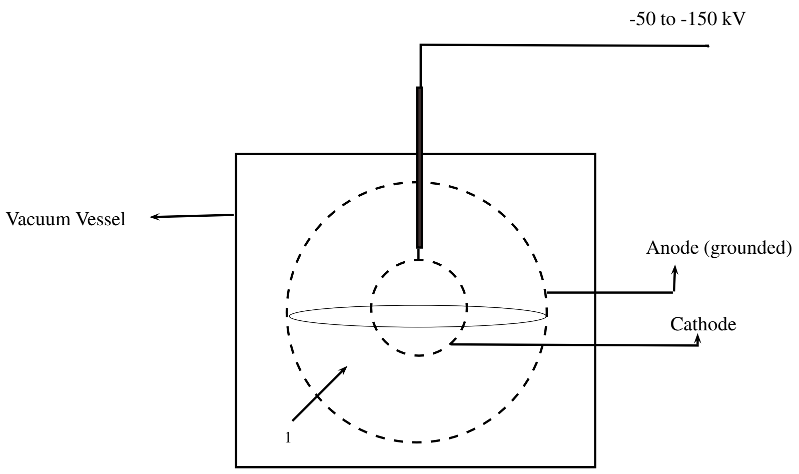
Inertial-electrostatic confinement(IEC) is a concept of trapping ions in convergent geometries like spherical or cylindrical ones using electrostatic fields. Philio Farnsworth (Farnsworth, 1966, 1968) and Oleg Laverent'yev (Lavrentev, 1975) independently formulated means of electrostatic confinement in the 1950's to 1970's, although in his work Laverent'yev studied plasma confinement using both electrostatic and magnetic fields. The term interial was borrowed from Farnsworth, who used the term 'inertial containment' for the trapping of positive and negative ions between equipotential boundaries. The IEC concept is used in thermonuclear fusion reactors to produce particle energies near the core that satisfy the conditions for nuclear fusion reactions.

1.1 Principle of Operation

In this section the method of operation of an IEC device of a specific configuration is explained. While the method of operation varies significantly from one device to another, the basic principle of operation remains same across various device configurations. The device considered here (see figure 1.1) consists of a spherical cathode grid that is surrounded concentrically by an anode grid. Both the anode and the cathode are highly (at least 90 %) transparent. A large negative voltage is applied to the cathode, while the anode is held at ground potential. Ions are introduced into the inter-electrode space by an ion source that is located outside the anode. The entire arrangement is housed inside a cylindrical vacuum vessel, and the fuel gas (D_2) is fed into this vessel. Electron-emitters placed outside the anode act as ion-sources by ionizing the fuel gas.

The ions that enter the inter-electrode space are accelerated towards the center of the device during which they gain fusion relevant energies. Moreover, because of the spherical focusing of ions towards the center, large ion-densities are produced thereat

Figure 1.1: Structure of an IEC device. The label 1 shows the space outside the anode, where ions are generated. Ions enter the inter-electrode space through the anode.



allowing fusion reactions between the ions. It should be noted that the ion trajectories are not perfectly radial as the ions start with some non-zero azimuthal velocity. In addition, processes such as elastic collision with the neutral gas atoms contribute to the deviation from perfectly radial motion. The high transparency of the cathode allows the ions to make multiple passes around the center, increasing the chances of fusion reactions. Although the IEC concept was first conceived as a means of facilitating ion-ion fusion reactions (also called as converged core and counter-streaming-ion fusion), it has been obvserved that the fusion reactivity is dominated by fusion reactions between the ions and the fast neutrals with the background gas (Meyer, 2007).

1.2 Fusion Reactions and Atomic Processes

In this section a description of the significant atomic processes that occur in the IEC device is given, followed by a classification of the fusion reactions occuring in the same. A large number of atomic and molecular processes occur within the space inside the anode, and some of the significant processes are mentioned below:

1. **Charge exchange reactions** :
   In such reactions electrons are transferred from one reactant species to another. In the present work only charge exchange reactions of the following type are considered.

$$D^+\ (fast)\ + D\ (slow) \longrightarrow D^+\ (slow)\ + D\ (fast)$$

The net effect can be thought of as a momentum transfer from the ion to the neutral background gas atom. In general, charge exchange might lead to many other effects such as creation of Deuterium anions (Santarius and Emmert, 2012).

2. **Ion-impact ionization**:
   These reactions occur primarily in the regions where the ions acquire energies high enough to knock off electrons from the gas atoms. Such processes can lead to avalanche ionization, if the electric field is high enough.

3. **Secondary electron emission**:
   Ions intercepting the cathode grid lead to secondary electron emission, which inturn can lead to many important effects such as 4.

4. **Electron-impact ionization and dissociation**:
   Fast moving electrons ionize the background gas atoms and also cause dissociation and subsequent ionization of background gas molecules.

5. **Ion-neutral elastic collisions**:
   Elastic scattering or collisions lead to significant deviations in the ion orbits.

In this work, it assumed that the fuel gas is completely dissociated, and only elastic scattering and charge exchange processes are considered. The Fusion reactions occuring in an IEC device are classified into several types as explained below (Krupakar Murali, 2004):

1. **Converged-core and Counter-streaming-ion (beam-beam)**:
   In counter-streaming-ion type of fusion, fusion occurs because of the intersection of ion orbits primarily near the center of the device. In converged-core type, the streaming ions interact with the dense core formed by the convergence of the ions into the cathode (inner-grid).

2. **Embedded Source (beam-target)**:
   Embedded source reactions occur between the fast moving ions and those trapped on the outer surface layers of the grid wires.

3. **Volume Source(charge-exchange neutral–neutral)**:
   Volume source fusion reactions occur between the fast neutrals(created by charge-exchange) and the neutral gas. Such reactions occur throughout the volume of the device.

4. **Beam-Background Reactions**:
   These reactions occur between the energitic ions and the background gas. As Child-Langmuir like potentials prevail in the device, the ions gain most of their energy near the cathode, and thus these reactions occur mostly in that region.

5. **Wall-Surface Reactions (charge-exchange neutral–target)**:
   The reactions between fast neutrals (created by charge-exchange) and the atoms trapped in the surface layers of the chamber walls are classified under this type.

Depending on the operating regime (gas pressure, power, fuel used etc.), each of the aforementioned reaction types become important. In the present work, only converged-core, volume source and beam-background reactions are taken into consideration because the device is assumed to operate at high pressures, wherein embedded source and wall-surface reactions have marginal contributions to the total fusion reactivity.

CHAPTER 2

THEORY, MODELING AND SIMULATION

In this chapter the model used for simulating Inertial-electrostatic plasma confinement is given. A brief, incomplete summary of the model is presented in section 2.1 for a better understanding of the the detailed explanation that follows. In each of the sections starting from section 2.2 a particular module of the model is discussed including the theory necessary for developing the same.

2.1 Summary of the model

A summary of the model is given below :

- The model used is a particle-in-cell model.
- We track only Deuterium ions and the fast neutral Deuterium atoms created by charge exchange processes.
- Only the particles with radius less than that of the anode are tracked.
- The motion of every particle (in its plane of motion) is tracked using its radial postion, and its radial and azimuthal velocities. The model presented here is therefore a 1-d model.
- For every particle i at a given radius it is assumed that N_i similar particles are present at the same radial distance but uniformly distributed in the polar and azimuthal directions. This assumption is made primarily to reduce the computational complexity associated with keeping track of every particle in the device.
- For reactions or processes with high probabilities, we assume that in any time step either all the N_i particles (corresponding to the particle i at a particular radius) participate in the reaction or all of them do not participate in the reaction.
- For reactions or processes with low probabilities, in any time step some of the N_i particles (corresponding to the particle i at a particular radius) participate in the reaction and some do not.
- Only the converged-core, volume source and beam-background type fusion reactions are considered in this model.

- Only charge exchange processes of the type discussed in section 1.2 and elastic collisions between the Deuterium ions and the background fuel gas are considered.
- The background fuel gas is assumed to be completely dissociated.

The pseudocode of the entire simulation is given below. How the individual parts of the simulation are implemented is discussed in the following sections.

```
1: for every time step do
2:     Generate ions
3:     Move all the ions according to the electrostatic potential in the device
4:     Simulate nuclear fusion reactions
5:     Simulate elastic scattering of Deuterium ions
6:     Simulate charge exchange process
7:     Recalculate the potential inside the device
8: end for
```

2.2 Electrostatic potential inside the device

2.2.1 Theoretical Analysis

An analytical solution for the vacuum potential is presented in this section. The initial (when no ions are present) potential distribution in the device can be obtained by solving the Laplace's equation. The Laplace's equation for the electric potential in spherical coordinates is

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial V}{\partial r}\right)+\frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial V}{\partial\theta}\right)+\frac{1}{r^2\sin^2\theta}\frac{\partial^2 V}{\partial\phi^2}=0 \tag{2.1}$$

Because of spherical symmetry equation (2.1) can be written as

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial V(r)}{\partial r}\right)=0 \tag{2.2}$$

The solution of equation (2.2) with the boundary conditions $V(r_a) = V_a$ and $V(r_c) = V_c$ is the initial potential distribution, where r_a and r_c are the anode and cathode radii respectively, and V_a and V_c are the anode and cathode voltages respectively. For $V_a = 0$, the solution for equation (2.2) for the above mentioned boundary conditions is as follows:

$$V(r) = \begin{cases} k\left(1 - \frac{r_a}{r}\right), & \text{if } r_c < r \leq r_a \\ V_c & \text{if } r \leq r_c \end{cases} \tag{2.3}$$

$$\text{where } k = \frac{V_c}{(r_a/r_c) - 1}$$

Figure 2.1: Plot of Vacuum Potential as function of radius

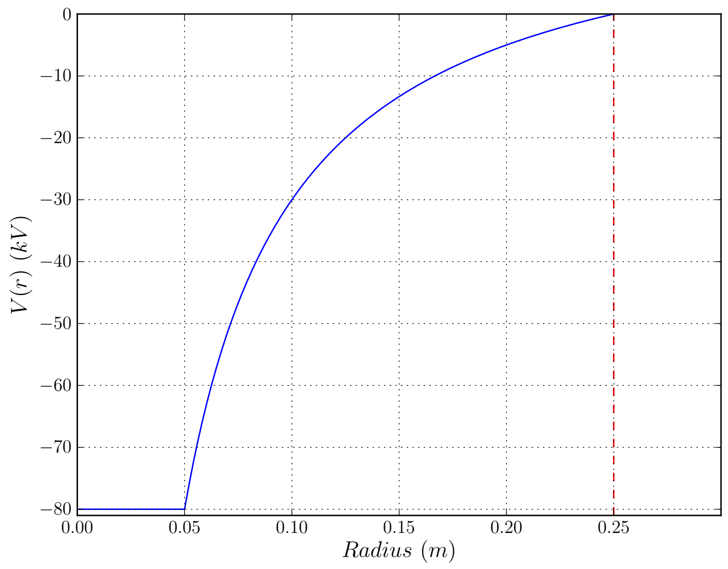


Figure 2.1 shows the potential distribution for $V_a = 0$, $r_a = 0.25m$, $V_c = -80kV$ and $r_c = 0.05m$. The rest of this section involves the derivation of equation (2.3).

Starting with equation (2.2) the following equations can be written:

$$\frac{\partial}{\partial r}\left(r^2 \frac{\partial V}{\partial r}\right) = 0$$

$$r^2 \frac{\partial V}{\partial r} = k_1$$

$$V(r) = \frac{-k_1}{r} + k_2$$

where k_1 and k_2 are some constants. Noting the fact that k_1 and k_2 take different values in the regions $r \leq r_c$ and $r \in (r_c, r_a]$, one can write

$$V(r) = \begin{cases} -\frac{k_1}{r} + k_2 & \text{if } r \in (r_c, r_a] \\ -\frac{k_1'}{r} + k_2' & \text{if } r \leq r_c \end{cases} \tag{2.4}$$

The first boundary condition, $V(r_a) = 0$, gives $k_1 = k_2 r_a$. A physically significant solution must satisfy

$$\lim_{r \to 0} V(r) = \text{finite}$$

Hence $k_1' = 0$. Substituting this in equation (2.4) and then using the boundary condition $V(r_c) = V_c$ gives $k_2' = V_c$.

Using the fact that $V(r)$ is continuous at r_c in equations (2.4)

$$-\frac{k_1}{r_c} + k_2 = V_c$$

Therefore

$$k_2 = \frac{V_c}{\frac{r_a}{r_c} - 1} \text{ and } k_1 = \frac{r_a V_c}{\frac{r_a}{r_c} - 1}$$

By substituting the values of k_1, k_2 k_1' and k_2' in equation (2.4) the potential $V(r)$ is obtained.

2.2.2 Model

Because of the spherically symmetric charge distribution inside the device, the space inside the reactor can be assumed to be radially divided into a number of small regions, forming a 1-dimensional grid of nodes of size $n + 1$. Approximating differntials with finite-differences, Poisson's equation can be written in the difference-form. The potential inside the device has to be recalculted in every time-step until a steady state is reached. Poisson's equation in spherical co-ordinates, with spherical symmetry is

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial V(r)}{\partial r}\right) = \frac{\rho}{\epsilon_0} \tag{2.5}$$

At any node j ($j \neq 0, n, r_c/\mathit{\Delta} r$), the difference equation can be written as

$$\left(r_j + \frac{\mathit{\Delta} r}{2}\right)^2 \left(\frac{V_{j+1} - V_j}{\mathit{\Delta} r}\right) - \left(r_j - \frac{\mathit{\Delta} r}{2}\right)^2 \left(\frac{V_j - V_{j-1}}{\mathit{\Delta} r}\right) = \frac{r_j^2 \rho_j}{\epsilon_0}$$

$$V_j = \frac{1}{2}\left(V_{j+1} + V_{j-1} + \frac{(V_{j+1} - V_{j-1}) \times r\mathit{\Delta} r}{r^2 + \mathit{\Delta} r^2/4} + \frac{\rho_j r^2 \mathit{\Delta} r}{r^2 + \mathit{\Delta} r^2/4}\right) \tag{2.6}$$

The differnce form of the boundary conditions is

$$V_{j_a} = V_a \tag{2.7}$$

$$V_{j_c} = V_c \tag{2.8}$$

$$V_0 = V_1 \tag{2.9}$$

where $j_a = r_a/\mathit{\Delta} r$ and $j_c = r_c/\mathit{\Delta} r$. Equation (2.9) is obtained by writing the left-hand side of equation (2.5) as

$$\frac{\partial^2 V}{\partial r^2} + \frac{2}{r}\frac{\partial V}{\partial r}$$

In the limit r tending to zero, for the above expression to be finite, the derivative of $V(r)$ at $r = 0$ should vanish. This gives the equation (2.9). The set of equations in (2.6) along with the boundary conditions forms a system of linear equations in the variables V_j ($j = 0$ *to* n). This system of linear equations can written in the form

$$AV = B$$

where A is an $(n+1) \times (n+1)$ matrix, B is a vector of size $n+1$, and V is the vector of unknowns V_j. It is easy to verify that the matrix A is irreducibly diagonally dominant. Therefore, the system of linear equations can be solved iteratively using the *Gauss-Siedel* method. The next section shows how the Gauss-Siedel method is implemented in the simulation.

2.2.3 Pseudocode

The following is the pseudocode for solving the equations (2.14) through (2.18).

```
 1: repeat
 2:     Copy old potential array
 3:     for Every node $j$ do
 4:         $V_j \longleftarrow \frac{1}{2}\left(V_{j+1} + V_{j-1} + \frac{(V_{j+1}-V_{j-1})\times r\varDelta r}{r^2+\varDelta r^2/4} + \frac{\rho_j r^2 \varDelta r}{r^2+\varDelta r^2/4}\right)$
 5:     end for
 6:     $V_{j_a} \longleftarrow V_a$ {$j_a$ is the index of the node at anode }
 7:     $V_{j_c} \longleftarrow V_c$ {$j_c$ is the index of the node at cathode }
 8:     $V_0 \longleftarrow V_1$
 9:     Find residual
10: until residual < tolerance
```

2.3 Ion generation

Deuterium ions are generated in the space outside the anode by ionizing the background gas using electron-emitters. The generated Deuterium ions enter the inter-electrode space with non-zero velocities and in random directions. The cathode current depends on the number of ions entering the inter-electrode space in any time-step. In this model, it is assumed that a fixed number of ions with a fixed energy are introduced into the inter-electrode space through the anode. However, their direction is assumed to be uniformly distributed in $[-\pi/2, \pi/2]$, where the angle is taken with reference to the radius directed towards the center.

2.4 Equations of motion

2.4.1 Theory

Under steady state, the ions inside the reactor undergo motion in a central force defined by a time-invariant potential $u(r)$. The total energy of any ion is given by

$$\begin{aligned} E &= \frac{m\dot{r}^2}{2} + \frac{mr^2\dot{\phi}^2}{2} + u(r) \\ &= \frac{m\dot{r}^2}{2} + \underbrace{\frac{M^2}{2mr^2} + u(r)}_{u_{\mathit{eff}}(r)} \qquad (2.10) \\ &= \frac{m\dot{r}^2}{2} + u_{\mathit{eff}}(r) \qquad (2.11) \end{aligned}$$

where m is the mass of the ion, $M = mr^2\dot{\phi}$ is the angular momentum of the ion, and $u_{\mathit{eff}}(r)$ is the effective potential seen by the ion. Taking the anode potential to be zero, the trajectory of any ion with total energy less than zero will be bounded. In general, the above condition does not hold true because ions enter the inter-electrode space from the region outside the anode with some non-zero velocity. However, when they lose energy by charge exchange processes, their total energy might become negative.

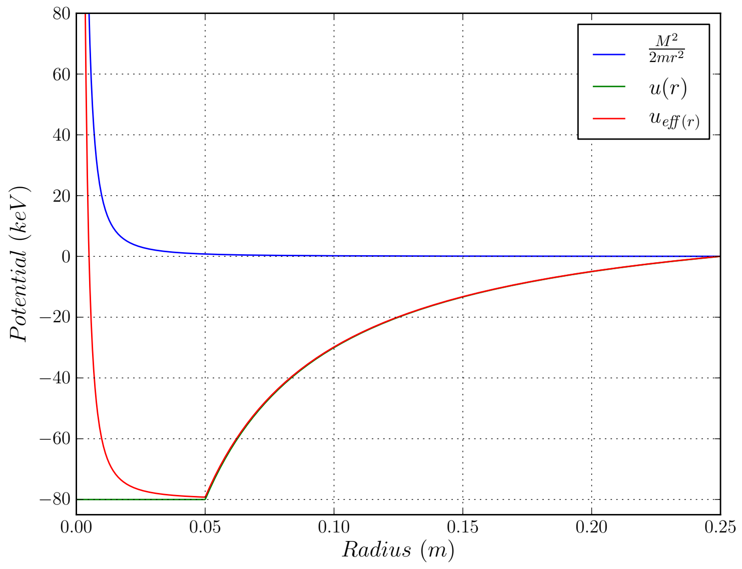
A reflecting boundary is used at the centre of the device i.e., velocities and radial positions of ions crossing the center are inverted. This is done because in this model only the radial positions (and not azimuthal or polar coordinates) of ions are tracked.

In order to understand the motion of the ions, let $u(r)$ in equation (2.10) above be replaced with the vacuum potential. Figure 2.2 shows a typical plot of the effective potential, taking the vacuum potential as the steady state potential $u(r)$.

The equations of motion of an ion can be derived by starting with the Lagrangian of the ion

$$L(r, \dot{r}, \phi, \dot{\phi}) = \frac{m\dot{r}^2}{2} + \frac{mr^2\dot{\phi}^2}{2} - u(r)$$

Figure 2.2: Plot of Effective Potential as function of radius



The equations of motion will be given by

$$\frac{\partial L}{\partial r} = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{r}}\right) \quad \text{and} \quad \frac{\partial L}{\partial \phi} = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right)$$

which reduce to

$$m\ddot{r} = mr\dot{\phi}^2 - \frac{\partial u}{\partial r} \tag{2.12a}$$

$$mr^2\dot{\phi}^2 = \text{constant} \tag{2.12b}$$

2.4.2 Model

To simulate the motion of the ions according to equations (2.12a) and (2.12b), the grid that is used to calculate the potential should be sufficiently large so that the electric field (or the derivative of the potential) is constant in the region between any two nodes in the grid. Assuming that the above condition is satisfied, equation (2.12a) between nodes j

and $j-1$ can be written as:

$$\begin{aligned} m\ddot{r} &= \frac{mv_\phi^2}{r} - \frac{q(V_j - V_{j-1})}{\Delta r} \\ \ddot{r} &= \frac{v_\phi^2}{r} + \frac{q(V_{j-1} - V_j)}{m\Delta r} \end{aligned} \tag{2.13}$$

where q is the charge of the ion, V_j is the electric potential at the node j, Δr is the length of each element of the grid, and $v_\phi = r\dot{\phi}$ is the azimuthal velocity of the ion for which the equation is written.

If r is almost constant over a time-step (which also implies from equation (2.12b) that v_ϕ is constant), $\ddot{r}$ remains constant in that time-step. Thus if the time-step dt is made sufficiently small that the acceleration remains constant in any time-step, in the n^{th} time-step the following equations can be written:

$$\ddot{r}_n = \frac{v_{\phi_{n-1}}^2}{r_{n-1}} + \frac{q(V_{j-1} - V_j)}{m\Delta r} \tag{2.14}$$

$$dr_n = \dot{r}_{n-1}dt + \frac{1}{2}\ddot{r}_n dt^2 \tag{2.15}$$

$$r_n = r_{n-1} + dr_n \tag{2.16}$$

$$\dot{r}_n = \dot{r}_{n-1} + \ddot{r}_n dt \tag{2.17}$$

$$v_{\phi_n} = \frac{r_{n-1}\, v_{\phi_{n-1}}}{r_n} \tag{2.18}$$

where dr_n and $\ddot{r}_n$ are the displacement and acceleration in the n^{th} time-step respectively, and $\dot{r}_n$, r_n and v_{ϕ_n} are the radial velocity, radial position and azimuthal velocity after the n^{th} time-step respectively.

2.4.3 Pseudocode

The pseudocode for simulating the motion of the ions is given below:

```
1: for every time step do
2:     for every ion do
3:         find the numbers of the nodes, $j$ and $j-1$,between which the ion is located
4:         find the acceleration using (2.14)
5:         calcluate the discplacement using (2.15)
```

```
 6:         if ion is crossing cathode then
 7:             run a die to find if the ion intercepts cathode {This step is explained in the
                following section}
 8:         end if
 9:         $temp \longleftarrow r\, v_\phi$ {temp is a temporary variable}
10:         update the position of the ion ($r$) using (2.16)
11:         update the velocity of the ion ($\dot{r}$) using (2.17)
12:         $v_\phi \longleftarrow temp/r$ {update the azimuthal velocity of the ion}
13:         if the particle passes through the center then
14:             invert its position and velocity
15:         end if
16:         if the particle reaches or crosses the anode then
17:             stop tracking that particle
18:         end if
19:     end for
20: end for
```

2.5 Reactions and atomic processes

2.5.1 Theory

2.5.1.1 Reaction Cross Section

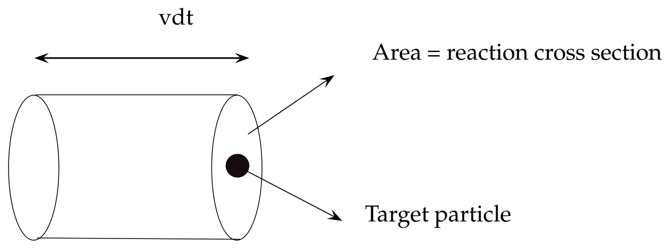
In this section the formula for the total number of reactions (of a particular type) that a particle participates in is derived in terms of the reaction cross section. Consider a reaction taking place between two particles. Let one of the particles be the target. We are interested in finding the total number of reactions that the target undergoes. Consider a circle centered around the target with area equal to the reaction cross section. By the definition of reaction cross section, in any infinitesimal time $\mathrm{d}t$, all the particles that cross the plane of the target through this circle react with the target (see figure 2.3)

In the frame of the target, the reactant particles of the other type arrive at the target at a velocity say v_{rel}. Then the total number of reactions that the target participates in is given by

$$r = \sigma \; v_{rel} \, \mathrm{d}t \; n \tag{2.19}$$

where n is the number density of the particles approaching the target. It is worth noting that for reactions in which the target is consumed[a] the number r must be smaller than one.

Figure 2.3: All the particles in the cylinder with length $v\mathrm{d}t$ react with the target



2.5.1.2 Fusion reaction and charge exchange reaction

A fusion reaction between two Deuterium atoms can lead to two equally probable sets of products as shown below.

$$\begin{aligned} {}^2_1D \quad + \quad {}^2_1D \quad &\longrightarrow \quad {}^3_2He \quad + \quad {}^1_0n \\ &\longrightarrow \quad {}^3_1T \quad + \quad {}^1_1p \end{aligned}$$

Each of the Deuterium atoms in the reaction shown above can be either ionized or neutral.

The charge exchange reaction that is considered in this model is as follows:

$$D^+\,(fast) \quad + D\,(slow) \quad \longrightarrow \quad D^+\,(slow) \quad + \quad D\,(fast)$$

[a] Here and elsewhere in this document the term consumption is used to refer to a change in the constitution.

2.5.1.3 Elastic Scattering

Figure 2.4 shows the process of elastic scattering between two particles. We are interested in deriving a formula for the scattering angle of the particle impinging the target. The target can be made stationary by choosing the frame of reference as the frame of the target. As shown in the figure 2.4, let particle A be the impinging particle and let particle B be the target. Let r and ϕ denote the radial and polar coordinates of particle A when particle B is chosen as the origin. Because of the conservation of angular momentum of A about the target B,

$$mr^2\frac{\mathrm{d}\phi}{\mathrm{d}t} = constant = mv_0 b$$

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = \frac{v_0 b}{r^2} \tag{2.20}$$

where m is the mass of particle A and b is the impact factor (shown in the figure 2.4). With reference to figure 2.4, the force balance equation in the vertical direction is

$$F\,Sin(\phi) = \frac{Z_A Z_B\, e^2 Sin(\phi)}{4\pi\epsilon_0\, r^2} = m\frac{\mathrm{d}v_y}{\mathrm{d}t} \tag{2.21}$$

where Z_A and Z_B are the atomic numbers of particles A and B respectively, e is the electronic charge, ϵ_0 is the vacuum permittivity, F is the central force seen by the particle A and v_y is the velocity of particle A in the y-direction. Equation (2.21) can be rewritten as

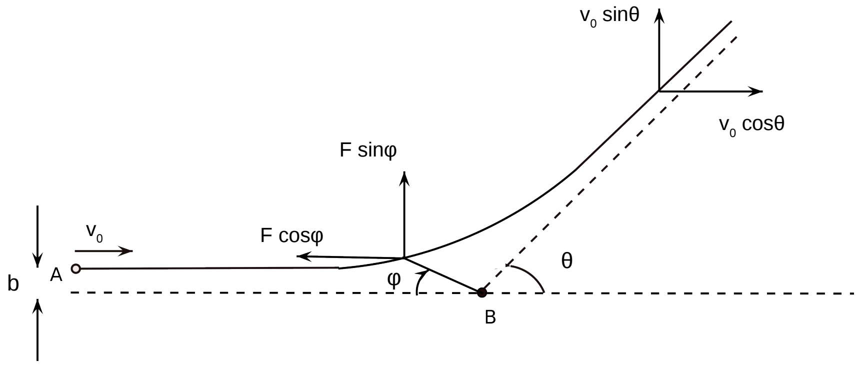
$$m\frac{\mathrm{d}v_y}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}\phi} = \frac{Z_A Z_B\, e^2 \sin\phi}{4\pi\epsilon_0\, r^2}\frac{r^2}{v_0 b} \tag{2.22}$$

$$\frac{\mathrm{d}v_y}{\mathrm{d}\phi} = \frac{Z_A Z_B\, e^2 \sin\phi}{4\pi\epsilon_0\, m v_0 b} \tag{2.23}$$

where equation (2.20) is used to write the right hand side of equation (2.22). Equation (2.23) upon integration gives

$$v_y(\phi) = \frac{Z_A Z_B\, e^2 \cos\phi}{4\pi\epsilon_0\, m v_0 b} + c \tag{2.24}$$

Figure 2.4: Elastic scattering between two particles



The constant c is set by using the fact that particle A has zero velocity in the y-direction when it is far from the target i.e., $v_y(0) = 0$. This gives

$$v_y(\phi) = \frac{Z_A Z_B \, e^2 (1 - \cos\phi)}{4\pi\epsilon_0 \, m v_0 b} \tag{2.25}$$

From the figure 2.4 it can also be seen that

$$v_y(\pi - \theta) = v_0 \sin\theta \tag{2.26}$$

Using equation (2.25) for the left hand side of equation (2.26), we can write

$$\frac{Z_A Z_B \, e^2 (1 + \cos\phi)}{4\pi\epsilon_0 \, m v_0 b} = v_0 \sin\theta$$

$$\frac{Z_A Z_B \, e^2}{8\pi\epsilon_0 \, E} = b \tan\left(\frac{\theta}{2}\right) \tag{2.27}$$

$$\theta = 2\cot^{-1}\left(b\frac{8\pi\epsilon_0 \, E}{Z_A Z_B \, e^2}\right) \tag{2.28}$$

where E is the kinetic energy of the impinging particle. Equation (2.28) gives the scattering angle in terms of the impact parameter b and the kinetic energy of the impinging particle.

For the case of scattering of Deuterium ions from the background Deuterium gas atoms, $Z_A = Z_B = 1$, and because of debye shielding by the electrons of the target Deuterium atoms, the impact parameter b has a certain limiting value beyond which the central force exerted by the target on the impinging ion becomes very small. The limiting value of the impact parameter depends on the distribution of the electrons inside the target Deuterium atom, and a closed form expression for it may not exist. However, the Bohr's radius a_0 can be taken as an upper bound on this limiting value. Paar (Paar, 2010) discusses how the same result can be obtained using quantum mechanical scattering theory. The impact parameter of any ion can thus be assumed to be uniformly distributed in $(-a_0, a_0)$.

2.5.2 Model

2.5.2.1 Model for Fusion Reactions

The number of reactions that a particle participates in any small time $\mathrm{d}t$ is given in equation (2.19). It has also been mentioned that for reactions in which the reactant particle is consumed, like nuclear fusion reactions, this number (r) will be less than one[a] [b]. This number can therefore be interpreted as the probabiltity that the particle undergoes a reaction. As mentioned in section 2.1, we assume that for every particle i at a given radius r_i, N_i similar particles are present at the same radial distance uniformly distributed in the polar and azimuthal directions. Let X be the random variable denoting the total number of reactions due to these N_i particles in any infinitesimal time $\mathrm{d}t$. Then X will have a Binomial distribution with parameters (N_i, r). In every time step, a realization of X should be generated, and the value of N_i should be decremented by this value. Since only Deuterium ions and fast Deuterium atoms are being tracked in this model, the only effect of the nuclear fusion reactions as per the model will be a decrement in the value of this number N_i.

[a]While calculating this number for reactions with the background gas atoms, the background gas atoms are assumed to be at rest as their velocities are negligible compared to the velocities of ions and fast neutrals

[b]It turns out that this number is less than one for elastic scattering also, which is not a reactant consuming process.

2.5.2.2 Model for Elastic Scattering and Charge Exchange

Unlike nuclear fusion reactions, in the processes of elastic scattering and charge exchange, the products consist of particles that need to be tracked. Therefore the effect of such processes will be that out of the N_i particles some particles change their state and some do not. If the process being considered is elastic scattering, the change of state is a change in the direction of motion (scattering), and if the process is charge exchange, the change in the state can be seen as a change in the momentum of the particle. In addition to changing the momentum of the reactant ion, a charge exchange process also creates a fast neutral Deuterium.

Because the effects of elastic scattering and charge exchange are more than those of fusion reactions, modelling them in a way similar to nuclear fusion reactions would increase the computational time and memory. An alternative to the model suggested in the previous section is to assume that in any time step, either all the N_i particles participate in the process of elastic scattering or charge exchange, or all of them do not participate, the probability with which they participate being given by r in equation (2.19). Such an assumption eliminates the computational cost associated with simulating all the effects of these processes. However, an approach such as this can capture the effects of charge exchange and elastic scattering to a reasonable degree of accuracy if the total number of time steps in the simulation is large. To see why this is so, consider N trials of a Bernoulli process with parameter p. Let the random variable Z denote the number of successes in the N trials. Then Z will have a binomial distribution with parameters (N, p). The mean and the variance of Z can be verified to be

$$\langle Z \rangle = Np$$
$$\left\langle (\Delta Z)^2 \right\rangle = Np\,(1-p)$$

Therefore

$$\frac{\sqrt{\left\langle (\Delta Z)^2 \right\rangle}}{\langle Z \rangle} = \sqrt{\frac{(1-p)}{Np}} \tag{2.29}$$

For $p \ll 1$,

$$\frac{\sqrt{\langle(\Delta Z)^2\rangle}}{\langle Z\rangle} \simeq \frac{1}{\sqrt{Np}} \tag{2.30}$$

Therefore if number of trials N is made large such that $Np \gg 100$, then realizations of Z lie close to the mean Np with a high probability.

Relating the above result to the number of charge exchange and elastic processes, it can be seen that increasing the number of time steps increases the number of trials of each of these processes. The method of simulating reactions that is described in the previous section and the one descibed in this section differ only in the number of trials made. From the result (2.30) we can then conclude that these two methods differ only in their accuracy if it can be ensured that the product Np is large, where N is the number of trials perfomed in the second method and p is the probability of the process that is being simulated. In other words, both methods yield approximately the same number of reactions in any shell inside the device, albeit, to varying degrees of accuracy.

The model mentioned in this section can also be applied to simulate the loss of particles due to their interception by the cathode. The fraction of cathode opacity can be used as the probability for such a process to occur.

2.5.3 Pseudocode

The pseudocode for simulating the fusion reactions is as follows:

```
1: for every particle $i$ do
2:     Find the probability of reaction $(r)$
3:     Generate a realization of a random variable $X$ distributed binomially with parameters $(N_i, r)$
4:     $N_i \longleftarrow N_i - x$
5: end for
```

The pseudocode for simulating elastic scattering is as follows:

```
1: for every ion $i$ do
```

```
2:     Compute the total velocity of the ion
3:     Compute the probability of scattering $(r)$ from equation (2.19)
4:     Generate a realization of a random variable $K$ uniformly distributed in $[0, 1]$
5:     if $k < r$ then
6:         Generate a realization of the impact parameter {Impact parameter is uniformly
           distributed in $(-a_0, a_0)$ as mentioned in section 2.5.1.3}
7:         Compute the scattering angle according to equation (2.28)
8:         Compute the angle in which the ion is travelling from the radial and azimuthal
           velocities
9:         Update this angle by adding to it the scattering angle
10:        Update the radial and azimuthal velocities
11:    end if
12: end for
```

The following is the pseudocode for the charge exchange process:

```
1: for every ion $i$ do
2:     Compute the total velocity of the ion $(v_{ion})$
3:     Compute the probability of charge exchange $(r)$ from equation (2.19)
4:     Generate a realization of a random variable $K$ uniformly distributed in $[0, 1]$
5:     if $k < r$ then
6:         Generate a realization of a random variable $L$ uniformly distributed in $[0, 1]$
7:         Compute the angle in which the ion is travelling from the radial and azimuthal
           velocities
8:         $v_{CX} \longleftarrow (1 - l)\, v_{ion}$ {$v_{CX}$ is the total velocity of the fast neutral created}
9:         $v_{ion} \longleftarrow l\, v_{ion}$
10:        Update the radial and azimuthal velocities of the ion
11:        Compute the radial and azimuthal velocities of the fast neutral assuming that
           it travels in the same direction as the ion
12:        Total number of fast neutrals of type $i$ at the given radius $\longleftarrow N_i$
13:    end if
14: end for
```

CHAPTER 3

RESULTS AND DISCUSSION

3.1 Results of the Simulation

The results of the simulation for the conditions shown in table 3.1 are presented in this chapter. The results show general agreement with the experimental observations. However, significant differences do exist.

Table 3.1: Simulation conditions under which the results shown were obtained

Cathode current	$\sim 1.2\ mA$
Neutral gas pressure	$2\ mtorr$
Anode radius	$0.25\ m$
Cathode radius	$0.05\ m$
Anode voltage	0
Cathode voltage	$-80\ kV$

Figure 3.3 shows the plot of the vacuum potential in steady state. A large potential well is seen near the center of the device. While the presence of such potential wells has been predicted (see figure 3.2 (Hirsch, 2004)) and experimentally verified (see figure 3.1 (Thorson *et al.*, 1997)), the cathode current in such cases is much larger than what is observed here [a]. The particle-in-cell code simulation of Tomiyasu (Tomiyasu *et al.*, 2003) also indicates only a small potential well ($\sim\ 200V$) for a cathode current of $40mA$.

A possible reason for this deviation could be the absence of electrons in the model presented here. Electrons with small energies taking birth near the cathode (either by secondary electron emission or by ion impact ionization as described in section 1.2) will be trapped in the potential well created by the Deuterium ion space charge, and this, in turn, decreases the size of the well.

[a]It should be noted that direct control over the cathode current does not exist. The cathode current can be controlled only by controlling the flux of the ions from the ion source.

Figure 3.1: Potential structure observed by Thorson (Thorson *et al.*, 1997)

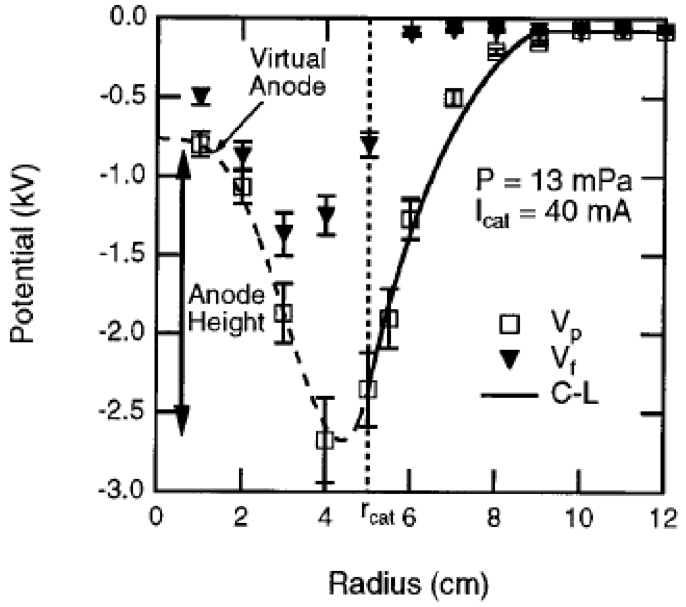


FIG. 6. Radial profile of the floating (V_f) and plasma (V_p) potentials for a 13 mPa, 5.0 kV, 40 mA case. Also included (solid line) is a normalized fit of the Child–Langmuir ($C-L$) potential distribution as expected from Eq. (6) (for $r > r_{cat}$).

Figure 3.4 shows the radial distribution of the Deuterium ions. A peak is observed near the anode ($r_a = 0.25m$) because of the large residence time of the ions thereat. The peak near the center is due to spherical focusing of ions. A small rise in the ion density is found near the cathode ($r_c = 0.05m$) as the low energy ions created by charge exchange processes are trapped in the potential well near the cathode.

Figures 3.5 and 3.6 are plots of the total number of Deuterium ions and the total number of fast neutrals in the device as a function of time. It can be seen that the number of ions reaches a steady state (steady oscillations) in $2\mu s$, while the total number of fast neutrals appears to become steady only after a much longer time. Due to the spherical focusing of ions, the current of the ions through the cathode is much higher than that of the fast neutrals. Thus the rate of loss of ions due to collisions with cathode (the main loss process for ions) will be much higher than that of the fast neutrals. Consequently, the number of fast neutrals is expected to become steady only after a longer time.

Figures 3.7 through 3.9 show the rates of various reactions or processes. No fusion reactions of the converged core or counter-streaming-ion type have been observed.

Figure 3.2: Potential structure predicted by Hirsch (Hirsch, 2004)

CONFINEMENT OF IONIZED FUSION GASES 45

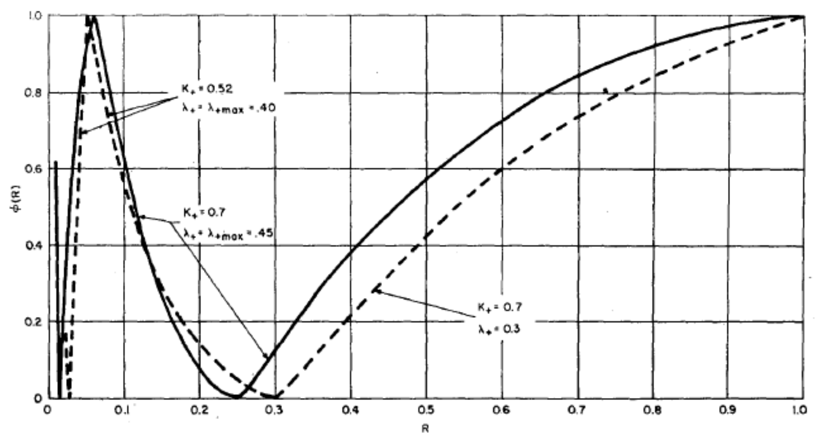


FIG. 5. Comparison of normalized potential distributions for $K_+=0.7$ at λ_{+max} and at 0.67 λ_{+max}.

Possible reasons for this could be that the current is too small so that the ion density is not high enough for such fusion reactions to occur. Also a large potential well near the center causes the ions to slow down, with the effect that the ion energy is small where the ion density is largest. Converged core and counter-streaming-ion fusion reactions can be expected to be seen by including electrons in the simulation model as the trapped electrons would decrease the well depth as explained before. The fusion reaction rates indicate higher neutron and proton production rates ($\sim 10^7/s$) than those obtained by other numerical methods (J F Santarius, 2002). The neutron and proton production rates are also closer to the experimental production rates (J F Santarius, 2002).

Figure 3.3: Plot of electrostatic potential inside the IEC device

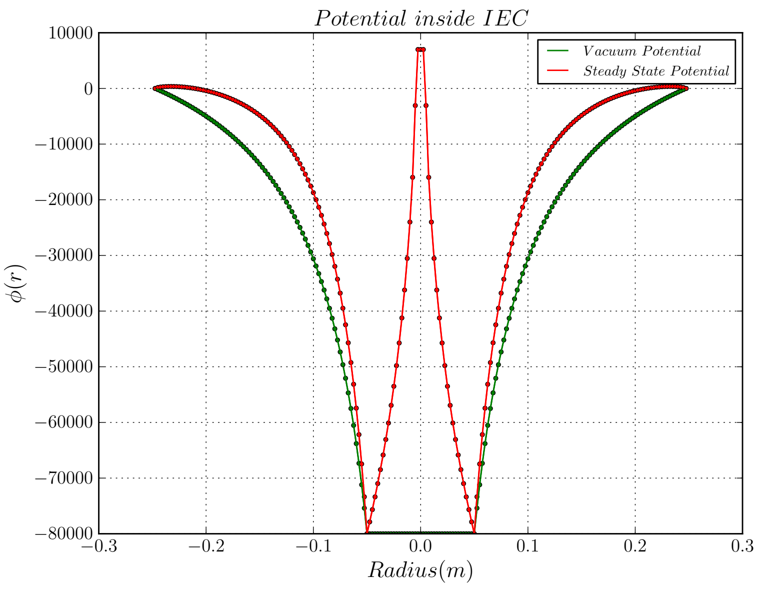


Figure 3.7: Rate of beam background type fusion reaction

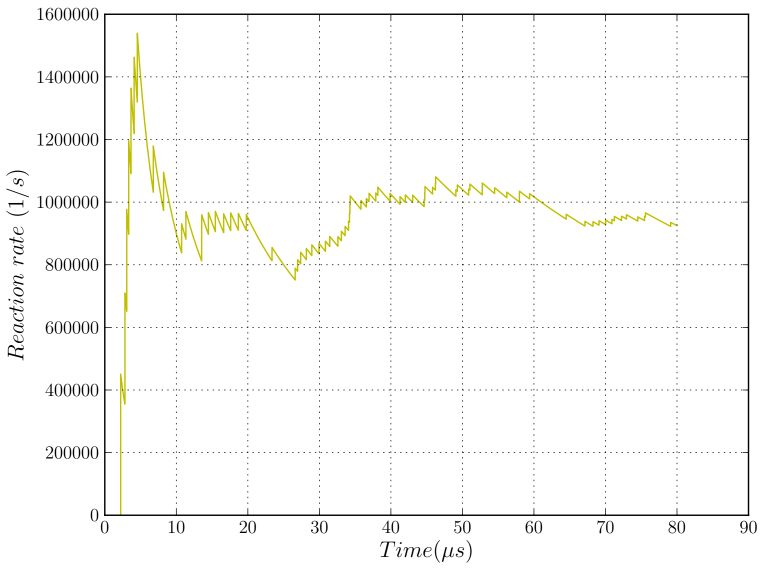


Figure 3.4: Radial distribution of the Deuterium ions

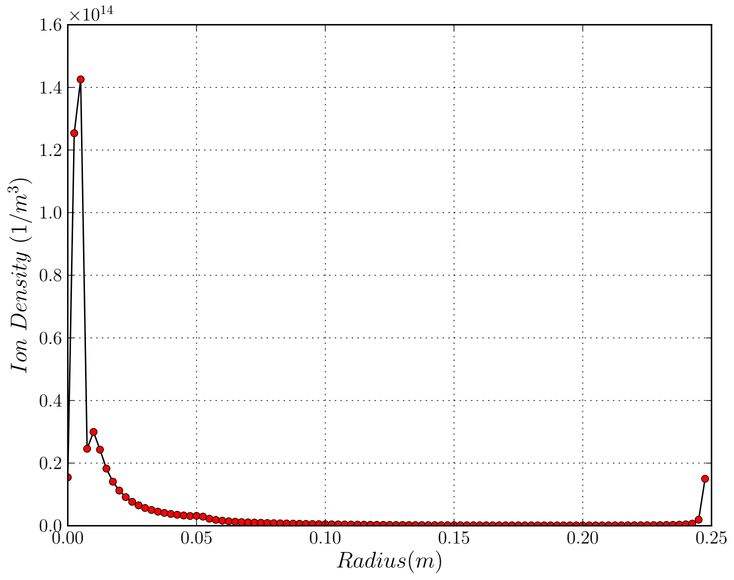


Figure 3.8: Rate of volume source type fusion reaction

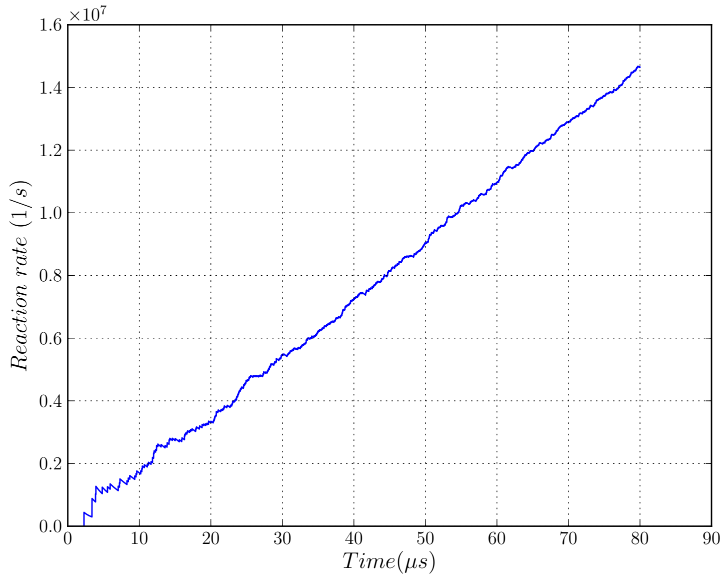


Figure 3.5: Total number of Deuterium ions as a function of time

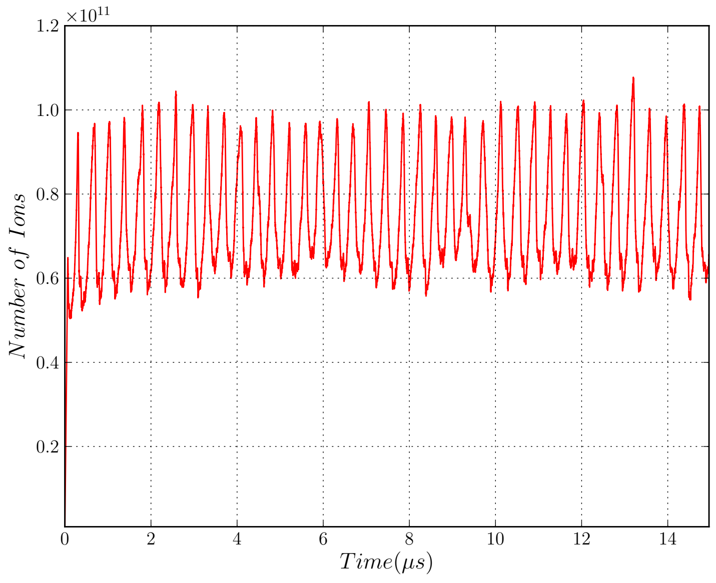
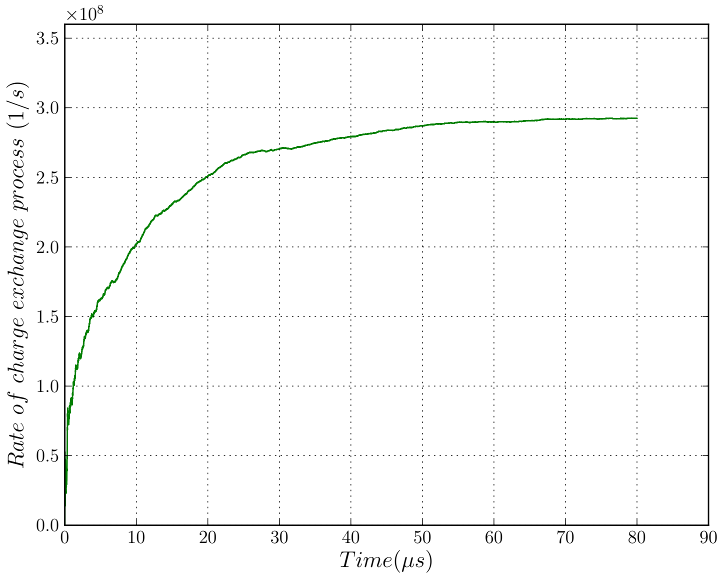
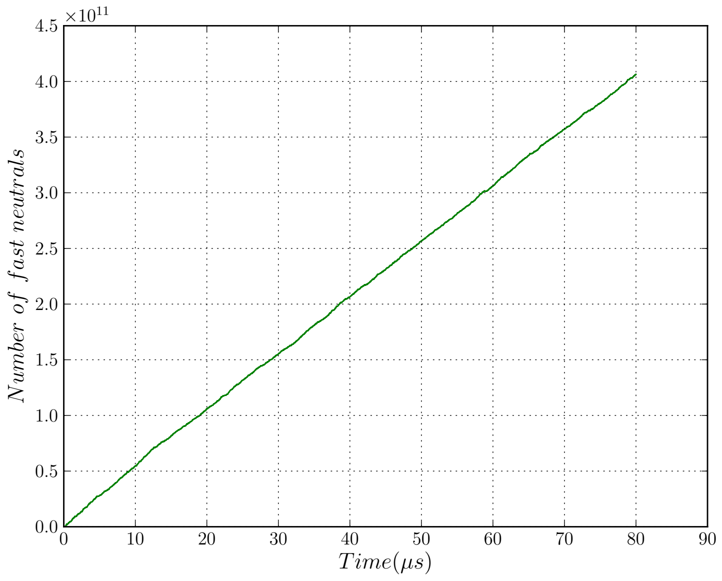


Figure 3.9: Rate of charge exchange process



The plot of time-averaged cathode current is shown as a function of time in figure 3.10. The figure shows a steady state value of $\sim 1.2\ mA$. Typical values of the cathode current are higher than this steady state value by an order of magnitude.

Figure 3.6: Total number of fast neutrals as a function of time



3.2 Conclusion and Improvements

Table 3.2: Comparison of results with experiments and previous simulations

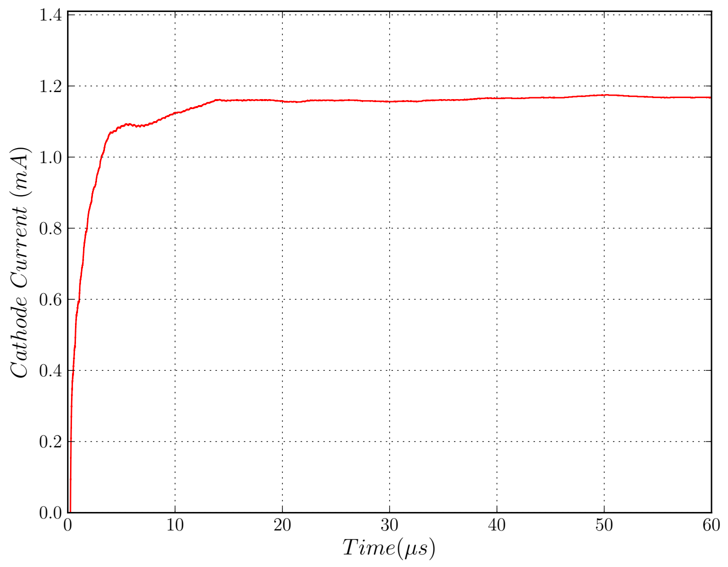
Quantity	Experimental	Tomiyasu	Present work
Potential Well/Bias	$\sim 1/3$	$\ll 1$	> 1
Ions	—	Steady	Steady oscillations
Cathode current	—	Steady state behavior matches	
Fast neutrals	—	Steady state behavior matches	
Fusion reaction rate	1.3×10^7[a]	8.8×10^5[b]	$\sim 10^7$[c]

Table 3.2 shows how the model presented here compares with the experiments and the PIC simulation of Tomiyasu (Tomiyasu *et al.*, 2003). The following is a list of modifications and/or additions that need to be made to improve the model presented here. The suggestions are made in the decreasing order of the impact they have on the model.

- Electrons have to be included in the model as already mentioned in this chapter.

[a] $100kV, 30mA$
[b] $100kV, 40mA$
[c] $80kV, 1.2mA$

Figure 3.10: Time averaged cathode current



Electrons are generated by electron-emitting filaments placed outside the anode, by ionization of background gas, and also by emission from the cathode when struck by fast ions.

- Unlike collisions with the background gas, where only one of the reactant species is tracked, ion-ion collisions are paired events, and modelling them as binary collisions would be highly inefficient as that would require pairing of particles in every time step and colliding the pairs. An efficient method for modelling ion-ion collisions has to be developed. The present work does not take ion-ion collisions into consideration.

- The model has to be extended to the region outside the anode to simulate impact ionization of the background gas by the electrons generated by the elecrton-emitters.

- In the present model, Euler technique is used for integration of the equations of motion. Euler intergration introduces a slight drift in the energy of the particles. Leapfrog integration can be used instead to overcome this defect and obtain more accurate particle trajectories.

- In some experiments, a thrid intermediate grid that is given RF excitation is used to support ionization of the fuel gas. The grid is covered with a fine mesh to confine electrons around the it, and as the electrons oscillate around the grid they produce ionization of the background gas. The model has to be extended to study the stability of the plasma under various configurations of RF excitation.

APPENDIX A

Simulation Code

The `#pragma` directives are used for parallelizing the code.

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include<time.h>

#include<string.h>

#include<gsl/gsl_integration.h>

#include <gsl/gsl_rng.h>

#include <gsl/gsl_randist.h>

#include<stdarg.h>


#define HALF_PI


#define Nmax 100000

#define Nmax_CX 100000

#define T 800000


static double K = 1.3806488 * pow(10,-23) ;

// conversion factor from mtorr to pascal

static double FACTOR = 0.133322368 ;

static double EPSILON =  8.854187 * pow ( 10 , -12 ) ;

static double E = 1.6 * pow(10,-19) ;

static double PI = 3.14159265 ;


double f (double x, void * params) {

  double energy = *(double *) params ;

  double m = 8 *PI* EPSILON * energy / ( E*E) ;

  double f = pow( HALF_PI - atan(m*x) , 2 ) ;

  return f;

}
```

```
int main()
{
clock_t start=clock();
//============================================================================//
//                            Begin Declarations
//============================================================================//

double r = 0.0025 ; // Delta r or grid size
double R_and = 0.25 ; // Anode radius
double R_cat = 0.05 ; // Cathode radius
int   N ; // No. of space elements or
N = (int)(R_and/r) ; // No. of elements in grid
double Vcat = -80000.0 ; // Cathode potential
double Vand = 0 ; // Anode potential
int   i, j, k ; // Iteration variables
int   l ; // Indexing variable
l = (int)(R_cat/r) ;

int   Niter = 7500 ; // Total no.of iterations for finding potential
double Vl[N+1] ; // Potential array
double Vl_old[N+1] ; // Potential arrays to store copy
double residual[Niter] ; // Array to store residual
double max ; // max is used to find maximum residual
double V[(2*N)+1] ; // Potential array to store potential
// on right and left sides
double rad[(2*N)+1] ; // Radius array to plot the solution

//============================================================================//
//                             Initialization
//============================================================================//

for(i=0;i<=N;i++)
{
    Vl[i] = 0 ;
    Vl_old[i] = 0 ;
}

Vl[l] = Vcat ;
```

```
Vl[N] = Vand ;

//==============================================================================//
//  Begin Iteration loop to find the Vacuum potential
//==============================================================================//

for(i=0 ; i<Niter ; i++)
{
    // Copy old arrays
    for(k=0 ; k<=N ; k++)
    {
        Vl_old[k] = Vl[k];
    }

    for(j=1 ; j<=N-1 ; j++)
    {
     if( j != l )
        // Difference form of Laplace’s equation
        Vl[j] = (Vl[j+1]+Vl[j-1]) * 0.5 +
             (Vl[j+1]-Vl[j-1]) * 0.5 * (j+1) / ( pow((j+1),2) + 0.25 ) ;

    }

    // Difference equation at r=0
    Vl[0] = Vl[1];

// Assert boundary conditions
Vl[l] = Vcat;
Vl[N] = Vand;

// Find residual
max = 0;
for(k=0 ; k<=N ; k++)
{
    if(fabs ( Vl[k] - Vl_old[k] ) > max)
     max = fabs ( Vl[k] - Vl_old[k] );

}
```

```
residual[i] = max;

// Termination criteria on residual
if(max < pow(10,-5))  break;
}
//============================================================================//
//  End iteration loop to find the Vacuum potential
//============================================================================//

//============================================================================//
//    Write vacuum potential data to the file vacuum_potential.txt
//============================================================================//

FILE *fp ; // File pointer
char *ptr ;                  // Pointer to store name of data folder
char actualpath[1000] ;          // Array to store name of file

system( "mkdir -p data" ) ;      // Create data folder
ptr = "data";
realpath ( ptr , actualpath ) ; // Store path of the data folder in actualpath
strcat ( actualpath , "/vacuum_potential.txt" ) ;
fp = fopen ( actualpath , "w" ) ;
if( fp == NULL )
{
    printf( "fp is NULL\n" ) ;
    return(0) ;
}

// Combine Vl and reversed Vl and print to file
for(i = 0 ; i <= (2*N) ; i++)
{
    V[i] = (i<N) * Vl[ (N-i)*(i<N) ] +
                   (i==N) * Vl[N-i] +
        (i>N) * Vl[ (i-N)*(i>N) ] ;

    rad[i] = -(i<N)*r*(N-i) +
       (i>N)*r*(i-N) ;  // When i=N, R=0
    fprintf( fp , "%f\t%f\n" , V[i] , rad[i] ) ;
}

fclose(fp) ;

//------------------------------------------------------------------------------//
//==============================================================================//
//    The following code simulates the motion of ions
//==============================================================================//
//------------------------------------------------------------------------------//

//==============================================================================//
// Begin Declarations
//==============================================================================//

double cathode_transparency ;   // Cathode transparency
static double x[Nmax] ; // Radii positions of ions
static double  Nx[Nmax] ; // Nx[j] stores no. of ions at x[j]
static double dx[Nmax] ;   // Radial Displacement of ions
static double v[Nmax] ; // Radial Velocity of ions
static double v_az[Nmax] ; // Azimuthal velocity of ions
double v_azi ; // Initial azimuthal velocity
static double x_CX[Nmax_CX] ; // Radii positions of CX neutrals
static long   Nx_CX[Nmax_CX] ; // Nx_CX[j] stores no. of ions at x_CX[j]
static double v_CX[Nmax_CX] ;  // Total Velocity of CX neutrals
static double v_r_CX[Nmax_CX] ; // Radial velocity of CX neutrals
double dt = pow(10 , -10) ; // Time step
double acc ; // Acceleration
int    ion_no ; // Number of ions that are added
int    mean_ion = 1 ; // Mean no. of ions added in each time step
double r_add = R_and ; // Radius where to add ions
double phi[ N + 1 ] ; // Potential array
double phi_old[ N + 1 ] ; // Potential array to store old values
double r_kill = 0.25 ;  // Radius below which ions are killed
static double tin[T] ; // Total ion no.
static double tcxn[T] ; // Total no. of charge exchange neutrals
static double time[T] ; // Array to store time
double q_ion ;  // Ion charge
double m_ion ;  // Mass of ion
double rho[ N + 1 ] ; // Array to store density of ions
double vol ; // Volume
double ion_den[ N + 1 ] ; // Array to store Ion density
double CX_den[ N + 1 ] ;  // Array to store CX neutral density
double v_avg[ N + 1 ] ; // Array to store Average velocity of ions
double v_avg_CX[ N + 1 ] ; // Array to store Average velocity of CX neutrals
double v_az_avg[ N + 1 ] ; // Array to store Average azimuthal velocity
double ion_den_avgt1[ N + 1 ]; // Ion density average over time
double ion_den_avgt2[ N + 1 ]; // Stores copy of the average ion density
double phi_avg[ N + 1 ] ;  // Potential average over time
double sigma_DD ;                // D+ D+ Reaction Cross-section
double sigma_DDe ;  // D+ D Elastic Scattering Cross-section
double sigma_CX ; // D+ D Charge Exchange Cross-section
double sigma_CXDD ;  // CX neutral- D Fusion Cross-section
double sigma_DDb ; // D+ D Fusion Reaction Cross-section
int    n_DD ; // No. of D+ D+ fusion reactions
int    n_CX ; // No. of charge exchange reactions
int    n_CXDD ;   // No. of CX neutral - D reactions
int    n_DDb ; // No. of D+ D fusion reactions
double DD_avg ; // Time averaged no. of D+ D+ fusion reactions
double CX_avg ; // Time averaged no. of charge exchange reactions
double CXDD_avg ; // Time averaged no. of CX neutral - D reactions
double DDb_avg ; // Time averaged no. of D+ D fusion reactions
double n_gas ;  // Number density of neutral gas
double p_gas = 2 ;  // Pressure of neutral gas in mtorr
double a_0 = 5.29 * pow(10,-11);// Bohr radius
double N_inst = pow(10,8) ;     // No. of instances of the 1-d simulation
double nc ; // A variable for counting Cathode Current
double nc_avg =0 ; // Average cathode current
double na ; // A variable for counting Cathode Current
double na_avg = 0 ; // Average anode current

double energy = 0 ; // A variable to store the energy in context
double theta ; // Stores the value of an angle in context
double theta_d ; // Stores deviation in an angle
double vel ; // Stores velocity in context
double temp ; // Temporary variable
double temp2 ;  // Temporary variable
int    len = 0 ;  // Temporary variable
int    tempint = 0 ; // Temporary integer variable
int    flag = 0; // A flag
double factor_1 ;  // A constant factor


long   jmax = 0 ;  // Maximum index of an existing ion

q_ion = 1.6 * pow( 10 , -19 ) ;

m_ion = 2 * 1.66053892 * pow( 10 , -27 ) ;

v_azi = sqrt ( 4 * pow( 10 , 1 ) * 1.6 * pow( 10 , -19 ) * 2 / m_ion ) ;

n_gas = p_gas * FACTOR / ( K * 300 ) ;

factor_1 = 8 * PI * EPSILON / ( E * E ) ;

sigma_DD = pow ( 10 , -40 ) ;

sigma_CX = pow ( 10 , -30 ) ;

sigma_DDb = sigma_DD ;

sigma_CXDD = sigma_DD ;

sigma_DDe = pow ( 10 , -30 ) ;

cathode_transparency = 0.95 ;

// Setup a gsl random no. generator  called rng
gsl_integration_workspace * w = gsl_integration_workspace_alloc( 1000 ) ;
gsl_rng_env_setup() ;

const gsl_rng_type * Z ;
Z = gsl_rng_default ;
```

```
gsl_rng * rng ;
rng = gsl_rng_alloc (Z) ;

double result, error_1 ;

gsl_function F;
F.function = &f;
F.params = &energy;

//============================================================================//
// Initialization
//============================================================================//
#pragma omp parallel for
for(i=0 ; i<Nmax ; i++)
{
x[i] = 0 ;
dx[i] = 0 ;
v[i] = 0 ;
v_az[i] = 0 ;
Nx[i] = 0 ;

x_CX[i] = 0 ;
v_CX[i] = 0 ;
Nx_CX[i] = 0 ;
}

// Initialize potential and average ion density
#pragma omp parallel for
for(i=0 ; i<=N ; i++)
{
    phi[i] = Vl[ (i-1) * (i!=0) ];
    ion_den_avgt1[i] = 0;
    ion_den_avgt2[i] = 0;
}

//============================================================================//
//     Open file ion_no.txt for writing total ion no. and time data
//      and file ion_den2.txt for writing time averaged ion density
```

```
//==============================================================================//

// Open ion_no.txt
ptr = "data";
realpath ( ptr , actualpath ) ; // Store path of the data folder in actualpath
strcat ( actualpath , "/ion_no.txt" ) ;
fp=fopen ( actualpath , "w" ) ;
if(fp == NULL)
{
    printf("fp is NULL\n") ;
    return(0);
}

// Open ion_den2.txt
FILE *id ; // File pointer
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/ion_den2.txt" ) ;
id = fopen ( actualpath , "w" ) ;
if(id == NULL)
{
    printf("File pointer is NULL\n");
    return(0);
}

// Open Convergence.txt
FILE *fp_convergence ;
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/Convergence.txt" ) ;
fp_convergence = fopen ( actualpath , "w" ) ;
if(fp_convergence == NULL)
{
    printf("Could not open Convergence.txt\n");
    return(0);
}

// Open reactions.txt
FILE *fp_reactions ;
```

```
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/reactions.txt" ) ;
fp_reactions = fopen ( actualpath , "w" ) ;
if(fp_reactions == NULL)
{
    printf("Could not open reactions.txt\n");
    return(0);
}

// Open cathode_current.txt
FILE * fp_current ;
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/Current.txt" ) ;
fp_current = fopen ( actualpath , "w" ) ;
if(fp_current == NULL)
{
    printf("Could not open Current.txt\n");
    return(0);
}

// Open Average_potential.txt
FILE * fp_avg_pot ;
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/Average_potential.txt" ) ;
fp_avg_pot = fopen ( actualpath , "w" ) ;
if(fp_avg_pot == NULL)
{
    printf("Could not open Average_potential.txt\n");
    return(0);
}

//============================================================================//
// Begin iterating over time steps
//============================================================================//
for( i=0 ; i<=T ; i++)
```

```
{

// Add ions
len = mean_ion;    // len is temporary variable
for( j=0 ; j<Nmax ; j++)
{
if( Nx[j]==0 && len!=0 )
{
x[j] = r_add -
   r * rand() / RAND_MAX ; // Need not type cast here
temp = (float) rand() / RAND_MAX ;
v[j] = -v_azi * cos(temp * PI / 2)    ;
v_az[j] = v_azi * sin(temp * PI / 2) ;
dx[j] = 0 ;
Nx[j] = (int) N_inst ;
len-- ;
}
if( len == 0 ) break;
if( (len!=0) && ( j == (Nmax-1) )  )
{
printf("FILLED!\n") ;
return(0) ;
}

}

nc = 0 ;
na = 0 ;
jmax = 0 ;

// Initialization
    #pragma omp parallel for
for ( j=0 ; j<=N ; j++ )
{
v_avg[j] = 0 ;
v_az_avg[j] = 0 ;
ion_den[j] = 0 ;
```

```
v_avg_CX[j] = 0 ;
}

// Move every ion in one time step
#pragma omp parallel for private(len,acc,temp,tempint)
for( k=0 ; k<Nmax ; k++ )
{

if( Nx[k] != 0 )
{
#pragma omp critical
if ( k > jmax)
jmax = k ;

len = (int)(x[k] / r) ; // len is a temporary variable
// used for indexing

// Calculate acceleration
acc = ( phi[ len - 1 ] - phi[ len ] ) * q_ion / ( m_ion * r ) +
                               pow( v_az[k] , 2 ) / x[k]  ;

// Calculate displacement
dx[k] = v[k] * dt +
    acc * dt * dt / 2 ;

// Stop if displacement in a time
// step is greater than grid size
if( fabs( dx[k] ) > r )
{
printf( "dx is %f\n" , dx[k] ) ;
printf( "x is %f\n" , x[k] ) ;
//return(0) ;
}

// Check if ion is crossing cathode
if( (x[k]-R_cat)*(x[k]+dx[k]-R_cat) < 0 )
{
temp = (double)rand() / RAND_MAX ;
```

```
//tempint = gsl_ran_binomial( rng , temp , Nx[k] ) ;

if ( temp < (1-cathode_transparency) )
{
Nx[k] = (int) 0 ;
#pragma omp atomic
nc += Nx[k] ;
}
}

temp = v_az[k] * x[k] ; // temp is a temporary variable
// that stores old angular momentum
// Update positon of ion
x[k] += dx[k] ;

// Update velocity of ion
v[k] += acc * dt ;

// Update azimuthal velocity
v_az[k] = temp / x[k] ;



if( x[k] < 0 )
{
    x[k] = -x[k] ;
v[k] = -v[k] ;
}

// Kill the ions that touch the anode
if( x[k] >= R_and )
{
#pragma omp atomic
na += Nx[k] ;
Nx[k] = (int) 0 ;
}

if ( Nx[k] != 0 )
{
```

```
len = (int)( ( x[k] + r/2 ) / r ) ;

#pragma omp atomic
ion_den[len] += Nx[k] ;

                // Uncomment this block for non-parallel code
//#pragma omp atomic
//v_avg[len] = ( v_avg[len] * (ion_den[len]-Nx[k]) + fabs(v[k])*Nx[k] )
//                / ion_den[len]  ;

                // Comment the following three lines for non-parallel code
#pragma omp atomic
v_avg[len] *= ion_den[len] - Nx[k] ;
#pragma omp atomic
v_avg[len] += fabs(v[k]) * Nx[k] ;
#pragma omp atomic
v_avg[len] /= ion_den[len] ;

            // Uncomment this block for non-parallel code
  //#pragma omp atomic
//v_az_avg[len] = ( v_az_avg[len] * (ion_den[len]-Nx[k]) + fabs(v_az[k])*Nx[k] )
//                / ion_den[len]  ;

    // Comment the following three lines for non-parallel code
#pragma omp atomic
v_az_avg[len] *= ion_den[len] - Nx[k] ;
#pragma omp atomic
v_az_avg[len] += fabs(v_az[k]) * Nx[k] ;
#pragma omp atomic
v_az_avg[len] /= ion_den[len] ;
}
}

if( Nx_CX[k] != 0 )
{
#pragma omp critical
if( k > jmax )
jmax = k ;
```

```
// Check if particle is crossing cathode
if( (x_CX[k]-R_cat)*( x_CX[k] + v_r_CX[k]*dt - R_cat ) < 0 )
{
temp = (double)rand() / RAND_MAX ;

if( temp < (1-cathode_transparency) )
{
Nx_CX[k] = (int) 0 ;
}
}

x_CX[k] = x_CX[k] + v_r_CX[k] * dt ;



if( x_CX[k] < 0 )
{
    x_CX[k] = -x_CX[k] ;
v_CX[k] = -v_CX[k] ;
}

if( x_CX[k] >= R_and )
Nx_CX[k] = (int) 0 ;

if( Nx_CX[k] != 0)
{
len = (int)( ( x_CX[k] + r/2 ) / r ) ;

#pragma omp atomic
CX_den[len] += Nx_CX[k] ;


}
}
}

// Calculate ion density at each radius
#pragma omp parallel for private(vol)
for ( j=0 ; j<=N ; j++ )
{
```

```
vol = 4 * PI * pow( ( j + 1 ) * r , 2 ) * r ;

ion_den[j] = ion_den[j] / vol ;

}

// Initialize no. of reactions to zero
n_DD = 0 ;
n_CX = 0 ;
n_CXDD = 0 ;
n_DDb = 0 ;

// Simulate ion-ion fusion, ion-Background gas fusion, elastic scattering
// and charge exchange
#pragma omp parallel for private(len,temp,vel,tempint,energy,temp2,theta,theta_d)
for ( j=0 ; j<jmax ; j++ )
{
// Ion- Ion Fusion
if ( Nx[j] != 0)
{
len = (int)( ( x[j] + r/2 ) / r ) ;

    vel = sqrt( pow(fabs(v_avg[len]-v[j]) , 2 )
               + pow( fabs(v_az_avg[len]-v_az[j]) , 2 ) ) ;

temp = ion_den[len] * vel * sigma_DD ;

tempint = gsl_ran_binomial( rng , temp , Nx[j] ) ;

if(tempint != 0)
{
if (tempint > Nx[j])
tempint = Nx[j] ;

Nx[j] = (int) (Nx[j] - tempint) ;
#pragma omp atomic
n_DD += tempint ;
printf( "DD:%d\n" , tempint ) ;
}

}


// Ion -Background gas

if (Nx[j] != 0)

{

vel = sqrt( pow(v[j],2) + pow(v_az[j],2) ) ;


temp = n_gas * vel * sigma_DDb ;


tempint = gsl_ran_binomial( rng , temp , Nx[j] ) ;


if(tempint != 0)

{

if (tempint > Nx[j])

tempint = Nx[j] ;


Nx[j] = (int) (Nx[j] - tempint) ;

#pragma omp atomic

n_DDb += tempint ;

printf( "DDb:%d\n" , tempint ) ;

}

}


/*

// Elastic scattering for large no. of scatterings

if( x[j] != 100 )

{

vel = sqrt( v_az[j] * v_az[j] + v[j] * v[j] ) ;


energy = 0.5 * m_ion * pow(v[j],2) ;


gsl_integration_qag (&F, 0, a_0, 0, 1e-6, 1000, 6, w, &result, &error_1);


            result = result / ( n_gas*fabs(v[j])*sigma_DDe ) ;


            result = sqrt(result *( 4 / a_0 )) ;
            theta_d = n_gas*vel*sigma_DDe * gsl_ran_gaussian (rng, result) ;

theta = atan2( v_az[j] , v[j] ) ;
//printf("Theta % 0.8f\n", theta) ;
//printf("Theta_d % 0.8f\n", theta_d) ;
//printf("Sin % 0.8f\n", sin(theta+theta_d)) ;
//printf("Cos % 0.8f\n", cos(theta+theta_d)) ;

v_az[j] = vel * sin(theta + theta_d) ;

v[j] = vel * cos(theta + theta_d) ;
}
*/

/*
// Elastic scattering as a binomial process
if( Nx[j] != 0 )
{
vel = sqrt( v_az[j] * v_az[j] + v[j] * v[j] ) ;

temp = n_gas*vel*sigma_DDe ;

tempint = gsl_ran_binomial( rng , temp , N_inst ) ;

if ( Nx[j] < tempint )
tempint = Nx[j] ;

if( tempint != 0 )
{
energy = 0.5 * m_ion * pow(v[j],2) ;

temp = factor_1 *energy ;   // factor_1 = 8*pi*epsilon/(e*e)

// Generate an RV  uniformly distriuted in (-a0,a0)
temp2 = 2 * ( gsl_rng_uniform_pos(rng) - 0.5 ) * a_0 ;

temp2 = temp2 * temp ;
```

```
if(temp2 != 0)

theta_d = atan( 1 / temp2 ) ;

else

theta_d = PI ;


theta = atan2( v_az[j] , v[j]) ;


for( k=0 ; (k <= Nmax) && (tempint != 0) ; k++ )

{

if( Nx[k] == 0 )

{

x[k] = x[j] ;

v[k] = vel * cos(theta + theta_d) ;

v_az[j] = vel * sin(theta + theta_d) ;

Nx[k] = (int) tempint ;

tempint = 0 ;

Nx[j] = (int)(Nx[j] - Nx[k]) ;

}

}

}

}*/


// Simulate elastic scattering as a Bernoulli process

if( Nx[j] != 0 )

{

temp = (double)rand() / RAND_MAX ;


if ( temp < n_gas*vel*sigma_DDe )

{

  vel = sqrt( v_az[j] * v_az[j] + v[j] * v[j] ) ;


energy = 0.5 * m_ion * pow(v[j],2) ;


temp = factor_1 *energy ;   // factor_1 = 8*pi*epsilon/(e*e)


// Generate an RV  uniformly distriuted in (-a0,a0)

temp2 = 2 * ( gsl_rng_uniform_pos(rng) - 0.5 ) * a_0 ;
```

```
temp2 = temp2 * temp ;

if(temp2 != 0)
theta_d = atan( 1 / temp2 ) ;
else
theta_d = PI ;

theta = atan2( v_az[j] , v[j] ) ;

v_az[j] = vel * sin(theta + theta_d) ;

v[j] = vel * cos(theta + theta_d) ;
}
}

/*
// Simulate charge exchange as a binomial process
if( Nx[j] != 0 )
{
vel = sqrt( pow(v[j],2) + pow(v_az[j],2) ) ;

temp = sigma_CX * vel * n_gas ;

tempint = gsl_ran_binomial( rng , temp , N_inst ) ;

if ( Nx[j] < tempint )
tempint = Nx[j] ;

if ( tempint != 0 )
{
len = (double)rand() / RAND_MAX ;

flag = 0 ;    // Set flag to 1 after finding one CX neutral

theta = atan2( v_az[j] , v[j] ) ;

for( k=0 ; k< Nmax_CX ; k++)
{
```

```
if ( (Nx_CX[k] == 0) && ( flag !=1 ) )
{
flag = 1 ;        // Flag set
x_CX[k] = x[j] ;
v_CX[k] = ( 1 -len ) * vel ;
v_r_CX[k] = v_CX[k] * cos(theta) ;
Nx_CX[k] = (int) tempint ;
}

if ( (Nx[k] == 0) && ( len != 4 )  )
{
x[k] = x[j] ;
v[k] = len * vel * cos(theta) ;
v_az[k] = len * vel * sin(theta) ;
Nx[k] = (int) tempint ;
Nx[j] = (int) (Nx[j] - Nx[k]) ;
len = 4 ;       // Now use len as a flag
}

if( (len == 4) && (flag == 1) )
break ;
}

n_CX += tempint ;
printf( "CX:%d\n" , tempint ) ;
}


}
*/

// Simulate charge exchange as a Bernoulli process
if( Nx[j] != 0 )
{
temp = (double)rand() / RAND_MAX ;

vel = sqrt( pow(v[j],2) + pow(v_az[j],2) ) ;

if ( temp < sigma_CX * vel * n_gas )
```

```
{
len = (double)rand() / RAND_MAX ;

flag = 0 ; // Set flag to 1 after finding one CX neutral

theta = atan2( v_az[j] , v[j] ) ;



for( k=0 ; k< Nmax ; k++)
{
#pragma omp critical
if (Nx_CX[k] == 0)
{
if ( k > jmax )
jmax = k ;
Nx_CX[k] = Nx[j] ;
flag = 1 ;  // Flag set
}

if(flag == 1)
{
x_CX[k] = x[j] ;
v_CX[k] = ( 1 -len ) * vel ;
v_r_CX[k] = v_CX[k] * cos(theta) ;
break ;
}
}

v_az[j] = len * vel * sin(theta) ;

     v[j]    = len * vel * cos(theta) ;

#pragma omp atomic
n_CX++ ;
}


}
```

```
// Simulate CX neutral - Background gas fusion
if(Nx_CX[j] != 0)
{
temp = n_gas * fabs(v_CX[j]) * sigma_CXDD ;

tempint = gsl_ran_binomial( rng , temp , Nx_CX[j]  ) ;

if(tempint != 0)
{
if (tempint > Nx_CX[j])
tempint = Nx_CX[j] ;

Nx_CX[j] = (int)(Nx_CX[j] - tempint) ;
#pragma omp atomic
n_CXDD += tempint ;
printf( "CXDD:%d\n" , tempint ) ;
}
}


}

// Count number of existing ions
tin[i] = 0 ;
tcxn[i] = 0 ;

#pragma omp parallel for
for ( j=0 ; j<jmax ; j++ )
{
if ( Nx[j] != 0 )
#pragma omp atomic
     tin[i] += Nx[j] ;

if( Nx_CX[j] != 0 )
#pragma omp atomic
tcxn[i] += Nx_CX[j] ;
}

time[i] = (i+1) * dt * pow ( 10 , 10 ) ;
```

```
fprintf ( fp , "%lf\t%lf\t%lf\n" , time[i] , tin[i], tcxn[i] ) ;
if ( ( i % 1000 ) == 0 )
printf ( "%d\n" , i ) ;

// Recalculate potential using new charge distribution

// Initialize
#pragma omp parallel for
for ( j=0 ; j<=N ; j++ ) rho[j] = 0 ;

// Calculate charge density in units of Coulomb * epsilon
#pragma omp parallel for private(len,vol)
for ( j=0 ; j<jmax ; j++ )
{
if( Nx[j] != 0 )
{
len = (int)( ( x[j]+ r / 2 ) / r ) ;    // len is temporary variable

vol =4 * PI * len * len * pow ( r, 3 ) * ( len !=  0 ) +
     4 * PI * pow (r , 3 ) * ( len == 0 );

#pragma omp atomic
rho[len] += q_ion * Nx[j] / (EPSILON * vol) ;
}
}

// Solve Poisson's equation
for ( j=0 ; j<Niter ; j++ )
{
// Copy old array
for ( k=0 ; k<=N ; k++ ) phi_old[k] = phi[k] ;

     // Difference form of Poisson's equaiton
    for ( k=1 ; k<=N ; k++ )
    {
     if ( k != 1 )
     phi[k]=( phi[k+1] + phi[k-1] ) * 0.5 +
            ( phi[k+1] - phi[k-1] ) * 0.5 * k / ( k * k + 0.25 ) +
                    rho[k] * k * k * 0.5 * r / ( k * k + 0.25) ;
    }

    // Difference equation at r=0
    phi[0] = phi[1] ;

    // Assert boundary conditions
    phi[N] = Vand ;
    phi[1] = Vcat ;

    // Find residual
max = 0 ;

for( k=0 ; k<=N ; k++ )
{
if( ( phi[k] - phi_old[k] ) > max )
{
    max = fabs( phi[k] - phi_old[k] ) ;
}
}

// Break if desired accuracy is reached
if( max < pow(10,-5) ) break;
}

// Compute time averaged ion density and phi
// temp = 0 ;
#pragma omp parallel for
for( j=0 ; j<=N ; j++ )
{
    // Update phi_avg
    phi_avg[j] = ( phi_avg[j] * i + phi[j] ) / ( i + 1 ) ;

    // Copy old ion density
    ion_den_avgt2[j] = ion_den_avgt1[j] ;

    // Update ion density average
    ion_den_avgt1[j] = ( ion_den_avgt1[j] * i + ion_den[j] ) / ( i + 1 ) ;
    if( fabs( ion_den_avgt2[j] - ion_den_avgt1[j] ) > temp )

     temp = fabs( ion_den_avgt2[j] - ion_den_avgt1[j] ) ;

}

// Compute time averaged quantities
DD_avg   = ( DD_avg * i * dt + n_DD )     / ( (i+1) * dt ) ;
CX_avg   = ( CX_avg * i * dt + n_CX )     / ( (i+1) * dt ) ;
CXDD_avg = ( CXDD_avg * i * dt + n_CXDD ) / ( (i+1) * dt ) ;
DDb_avg  = ( DDb_avg * i * dt + n_DDb )   / ( (i+1) * dt ) ;
nc_avg   = ( nc_avg * i * dt + nc * q_ion ) / ( (i+1) * dt ) ;
na_avg   = ( na_avg * i * dt + na * q_ion ) / ( (i+1) * dt ) ;
//nc_avg = nc*q_ion/dt ;
/*if( temp < 75 && flag == 0 )
{
flag = i ;
    for( j=0 ; j<=N ; j++ )
    ion_den_avgt1[j] = ion_den[j] ;
}
*/
// Cathode current
fprintf( fp_current , "%lf\t%lf\n" , nc_avg , na_avg ) ;

// Print time averaged ion density to a file
//for( j=0 ; j<=N ; j++ )
// fprintf( id , "%f\n" , ion_den_avgt1[j] ) ;

// Print time averaged potential to a file
//for( j=0 ; j<=N ; j++ )
// fprintf( fp_avg_pot , "%f\n" , phi_avg[j] ) ;

// Print error in ion density to Convergence.txt
fprintf ( fp_convergence , "%d\t%f\n" , i , temp ) ;

// Print time averaged no. of reactions to reactions.txt
fprintf ( fp_reactions, "%f\t%f\t%f\t%f\n" , DD_avg , CX_avg , CXDD_avg , DDb_avg ) ;
```

```
}

//============================================================================//
// End iterating over time steps
//============================================================================//

// Close files
fclose(fp);
fclose(id);
fclose(fp_convergence) ;
fclose(fp_reactions) ;
fclose(fp_current) ;
fclose(fp_avg_pot) ;

// Write steady_state potential data to the file potential_steady_state.txt
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/potential_steady_state.txt" ) ;
fp=fopen ( actualpath , "w" ) ;
if( fp == NULL )
{
    printf( "fp is NULL\n" ) ;
    return(0) ;
}

for( j=0 ; j<=N ;j++ )
    fprintf( fp , "%f\t%f\n" , phi_avg[j] , r * j ) ;

fclose(fp);

// Write ion flux data to the file ion_flux.txt
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/ion_flux.txt" ) ;
fp = fopen ( actualpath , "w" ) ;

if( fp == NULL )
```

```
{
    printf( "fp is NULL\n" ) ;
    return(0) ;
}

for( j=0 ; j<=N ; j++ )
fprintf( fp , "%f\t%d\n" , 1 / ( 4*PI*pow( (j+1)*r , 2 )*v_avg[j] ) , j ) ;

fclose(fp);

// Write final ion density data to the file ion_density.txt
ptr = "data" ;
realpath ( ptr , actualpath ) ;
strcat ( actualpath , "/ion_density.txt" ) ;
fp = fopen ( actualpath , "w" ) ;

if( fp == NULL )
{
printf( "fp is NULL\n" ) ;
return(0) ;
}

for( j=0 ; j<=N ; j++ )
fprintf( fp , "%f\n" , ion_den_avgt1[j] ) ;

fclose(fp);

// Print output
printf( "j\trho[j]\n" ) ;
for( j=0 ; j<=N ; j++ ) printf( "%d\t%f\n" , j , rho[j] ) ;

clock_t end=clock();
double comptime=(end-start)/(double)CLOCKS_PER_SEC;
printf("elapsed time:\t%f\n",comptime);

// Plot using python script
system( "python plot.py" ) ;
```

```
}
```

APPENDIX B

Code used for plotting

```
from matplotlib import rc
rc('font',**{'family':'serif', 'size':'13', 'serif':'serif'})
rc('axes', **{'grid':'False', 'labelsize':'16', 'titlesize':'16'})
rc('text', usetex=True)

from pylab import*
import os

V    = []
r    = []
TPN  = []
TCXN = []
time = []
phi  = []
r2   = []
DD   = []
CX   = []
CXDD = []
DDb  = []

f = open ( os.path.realpath ( '../data/vacuum_potential.txt' ) , "r" )
lines = f.readlines()
f.close()
for line in lines:
    l = line.split()
    V.append( l[0] )
    r.append( l[1] )

f = open ( os.path.realpath ( '../data/potential_steady_state.txt' ) , "r" )
lines = f.readlines()
f.close()
```

```
for line in lines:
        l = line.split()
        phi.append( l[0] )
        r2.append ( l[1] )

phi_ss = []
phi_ss = phi[::-1]
phi_ss = phi_ss[0:-1]
phi_ss = phi_ss + phi



figure(0)
title( "$Potential\ inside\ IEC$" )
xlabel( "$Radius(m)$" )
ylabel( "$\phi(r)$" )
plot( r , V , '-g' )
plot( r , phi_ss , '-r' )
plot( r , V , 'go' , markersize = 3 )
plot( r, phi_ss , 'ro' , markersize = 3 )
plot( r , V , '-g' )
plot( r , phi_ss , '-r' )
legend( ( '$Vacuum\ Potential$' , '$Steady\ State\ Potential$' ) , prop={'size':10})
grid()



f = open ( os.path.realpath ( '../data/ion_no.txt' ) , "r" )
lines = f.readlines()
f.close()
for line in lines:
        l = line.split()
        time.append( l[0] )
        TPN.append ( l[1] )
        TCXN.append( l[2] )

time = [ float(i)/10000 for i in time]
figure(1)
#title( "$Plot\ of\ Total\ Number\ of\ Particles\ Vs\ Time$" )
xlabel( "$Time(\mu s)$" )
```

```
ylabel( "$Number\ of\ Ions$" )
#plot( time , TPN  , '-r' )
#plot( time , TCXN , '-g' )
#plot( time , TPN  , 'ro' , markersize = 2 )
#plot( time , TCXN , 'go' , markersize = 2 )
plot( time , TPN  , '-r' )
#plot( time , TCXN , '-g' )
#legend( ( '$Ions$','$CX\ Neutrals$' ) )
grid()
xlim( 0 , 20 )

figure(11)
xlabel( "$Time(\mu s)$" )
ylabel( "$Number\ of\ fast\ neutrals$" )
#plot( time , TCXN , 'go' , markersize = 2 )
plot( time , TCXN , '-g' )
grid()

f = open ( os.path.realpath ( '../data/ion_density.txt' ) , "r" )
ion_den = f.readlines()
f.close()
figure(2)
plot( r2 , ion_den , '-k' )
plot( r2 , ion_den , 'ro' , markersize = 5 )
#title( "$Plot\ of\ Ion\ Density\ at\ steady\ state$" )
xlabel( "$Radius(m)$" )
ylabel( "$Ion\ Density\ (1/m^3)$" )
grid()

f = open ( os.path.realpath ( '../data/reactions.txt' ) , "r" )
lines = f.readlines()
f.close()
for line in lines:
        l = line.split()
        DD.append( l[0] )
        CX.append( l[1] )
        CXDD.append ( l[2] )
        DDb.append  ( l[3] )
```

```
figure(3)
xlabel( "$Time(\mu s)$" )
ylabel( "$Reaction\ rate\ (1/s)$" )
#plot( time , DD , '-r' )
#plot( time , DD , 'ro' , markersize = 2 )
plot( time , DD , '-r' )
#title( '$DD$' )
grid()

figure(4)
xlabel( "$Time(\mu s)$" )
ylabel( "$Rate\ of\ charge\ exchange\ process\ (1/s)$" )
#plot( time , CX , '-g' )
#plot( time , CX , 'go' , markersize = 2 )
plot( time , CX , '-g' )
ylim( 0 , 3.6*10**8 )
#title( '$CX$' )
grid()

figure(5)
xlabel( "$Time(\mu s)$" )
ylabel( "$Reaction\ rate\ (1/s)$" )
#plot( time , CXDD , 'bo' , markersize = 2)
plot( time , CXDD , '-b' )
#title( '$CXDD$' )
grid()

figure(6)
xlabel( "$Time(\mu s)$" )
ylabel( "$Reaction\ rate\ (1/s)$" )
#plot( time , DDb , 'yo' , markersize = 2 )
plot( time , DDb , '-y' )
#title( '$DDb$' )
grid()

# Uncomment this block for plotting ion density as a function of time
'''
```

```
ion()
ion_den = []
f= open ( os.path.realpath( '../data/ion_den2.txt' ) , "r" )
lines = f.readlines()
f.close()

figure(7)
for i in range( len(time) ) :
        if( i%1000 == 0 ) :
                clf()
                title( '$Plot\ of\ ion\ density$' )
                xlabel( '$Radius(m)$' )
                ylabel( "$Ion density(1/m^{3})$" )

                ylim( 0 , 3*10**12 )
                plot( r2 , lines[i*len(r2):(i+1)*len(r2)] , 'ro' , markersize = 4 )
                draw()
                plot( r2 , lines[i*len(r2):(i+1)*len(r2)] , '-k' )
                draw()

ioff()
'''
cathode_current = []
anode_current   = []
f = open ( os.path.realpath ( '../data/Current.txt' ) , "r" )
lines = f.readlines()
f.close()

for line in lines:
l = line.split()
cathode_current.append( 1000*float( l[0] ) )
anode_current.append  ( 1000*float( l[1] ) )

figure(8)

#plot ( time , cathode_current , '-r' )
#plot ( time , anode_current   , '-g' )
#plot ( time , cathode_current , 'ro' , markersize=2 )
```

```
#plot ( time , anode_current   , 'go' , markersize=2 )
plot ( time , cathode_current , '-r' )
#plot ( time , anode_current   , '-g' )
ylim( 0 , 1.41 )
xlim( 0 , 60 )
grid()

#legend ( ( "$ Cathode\ Current\ (mA) $" , "$ Anode\ Current\ (mA) $" ) )
#title( "$Cathode\ Current\ Vs\ Time$" )
xlabel( '$Time(\mu s)$' )
ylabel( "$Cathode\ Current\ (mA)$" )

figure(9)
#plot ( time , anode_current   , 'go' , markersize=2 )
plot ( time , anode_current   , '-g' )
xlabel( '$Time(\mu s)$' )
ylabel( "$Anode\ Current\ (mA)$" )
ylim( 0 , 200 )
xlim( 0 , 20 )
grid()

flux = []
j    = []

f = open ( os.path.realpath ( '../data/ion_flux.txt' ) , "r" )
lines = f.readlines()
f.close()
for line in lines:
        l = line.split()
        flux.append( l[0] )
        j.append   ( l[1] )

#figure(10)
#plot(j,flux,'r+')
#plot(j,flux,'-k')
#grid()
show()
```

REFERENCES

1. **Farnsworth, P. T.** (1966). Electric discharge device for producing interactions between nuclei. US Patent 3,258,402.

2. **Farnsworth, P. T.** (1968). Method and apparatus for producing nuclear-fusion reactions. US Patent 3,386,883.

3. **Hirsch, R. L.** (2004). Inertial-electrostatic confinement of ionized fusion gases. *Journal of Applied Physics*, **38**(11), 4522–4534.

4. **J F Santarius, G. L. K. B. B. C. S. K. M. G. R. P. R. F. R. J. W. W., R P Ashley**, Modeling d-d operation of the uw iec experiment. *In 5th US-Japan Workshop on IEC Fusion*. 2002.

5. **Krupakar Murali, S.** (2004). *Diagnostic study of steady state advanced fuel (D–D and D–3 He) fusion in an IEC device*. Ph.D. thesis, Ph. D. dissertation, University of Wisconsin, Madison.

6. **Lavrentev, O. A.** (1975). Electrostatic and electromagnetic high-temperature plasma traps. *Annals of the New York Academy of Sciences*, **251**, 152–178.

7. **Meyer, R.** (2007). *Inertial electrostatic confinement: theoretical and experimental studies of spherical devices*. Ph.D. thesis, University of Missouri–Columbia.

8. **Paar, H.**, *An Introduction to Advanced Quantum Physics*. Wiley, 2010. ISBN 9780470665091.

9. **Santarius, J. F.** and **G. A. Emmert** (2012). Iec device core physics explorations.

10. **Thorson, T. A.**, **R. D. Durst**, **R. J. Fonck**, and **L. P. Wainwright** (1997). Convergence, electrostatic potential, and density measurements in a spherically convergent ion focus. *Physics of Plasmas (1994-present)*, **4**(1), 4–15.

11. **Tomiyasu, K.**, **J. Santarius**, and **G. Kulcinski**, Numerical simulation for uw-iec device. *In 6th US-Japan Workshop on IEC Fusion*. 2003.