

# Brief Summary and Tutorial of the FEM Plasma Code: Zapdos

Corey DeChant

October 2019

# Summary

The purpose of this tutorial document is to give the reader an overview of Zapdos and its capabilities. Within this tutorial document, there is a review of the finite element method (FEM) technique of transforming the strong form of an equation into its weak form. With this technique, we obtain the weak form of the multi-fluid approach to plasma modeling and the available physics within Zapdos. This document also reviews the object-oriented approach of MOOSE/Zapdos, the function of each object block, and the structure of these object blocks within an input file. There is a quick overview of MOOSE's GUI, Peacock, which is helpful for first time users of MOOSE and those with little/no experience with C++. With this knowledge, the tutorial document ends with a few examples of input files and a description of the plasma discharges that they incorporate.

# Acknowledgements

We would like to acknowledge the following as collaborators with the support and development of Zapdos:

- University of Illinois at Urbana-Champaign: Laboratory of Computational Plasma Physics
- University of Notre Dame: Go Research Group
- University of California, Berkeley: Graves Lab
- Oak Ridge National Laboratory: Fusion Energy Division

We would also like to acknowledge the following as funding sources:

- United States Department of Energy through the Office of Science Graduate Student Research (SCGSR) Program
- National Science Foundation through the SI2-SSE Award 1740300
- Idaho National Laboratory through the INL Graduate Fellowship Program

# Contents

<b>Summary</b>	<b>1</b>
<b>Acknowledgements</b>	<b>2</b>
<b>1 Introduction</b>	<b>5</b>
<b>2 Brief Introduction to FEM and MOOSE Objects</b>	<b>6</b>
2.1 FEM: Strong to Weak Form . . . . .	6
2.2 MOOSE/Zadpos Objects . . . . .	7
2.2.1 GlobalParams . . . . .	7
2.2.2 Mesh and MeshModifiers . . . . .	7
2.2.3 Problem . . . . .	7
2.2.4 Variables and Kernels . . . . .	7
2.2.5 AuxVariables and AuxKernels . . . . .	7
2.2.6 InterfaceKernels . . . . .	8
2.2.7 BCs . . . . .	8
2.2.8 ICs . . . . .	8
2.2.9 Functions . . . . .	8
2.2.10 Materials . . . . .	8
2.2.11 Postprocessors . . . . .	8
2.2.12 Preconditioning . . . . .	8
2.2.13 Executioner . . . . .	8
2.2.14 TimeStepper . . . . .	9
2.2.15 Outputs . . . . .	9
<b>3 Physics of Zapdos</b>	<b>10</b>
3.1 Weak Form of Multi-Fluid Plasmas . . . . .	10
3.2 Boundary Conditions . . . . .	12
3.2.1 Lymberopoulos BCs . . . . .	12
3.2.2 Sakiyama BCs . . . . .	13
3.2.3 Hagelaar BCs . . . . .	13
3.2.4 Dirichlet Density BCs . . . . .	14
3.2.5 Potential BCs . . . . .	14
3.3 Treatment of Variables and Scaling Methods . . . . .	15
<b>4 GEC Reference Cell Tutorial Examples</b>	<b>16</b>
4.1 Zapdos Input Files and Peacock . . . . .	16
4.2 1D: CCP Discharge . . . . .	18
4.3 2D: GEC Reference Cell . . . . .	18

4.4	Hands-on Work . . . . .	18
4.4.1	GEC at Higher Pressures . . . . .	18
4.4.2	The Plasma-Liquid Interface . . . . .	18
<b>5</b>	<b>Appendix</b>	<b>20</b>
	<b>References</b>	<b>44</b>

# Chapter 1

## Introduction

Zapdos is an open source plasma fluid code developed by Dr. Alexander Lindsay at North Carolina State University (NCSSU). It is built as an application in the MOOSE (Multiphysics Object Oriented Simulation Environment) framework. The MOOSE framework is an object-oriented C++ finite element method (FEM) framework developed at Idaho National Laboratory for the purpose of solving highly coupled multiphysics problems. The MOOSE framework has several applications that are dedicated to particular areas of physics and that can be easily coupled together, Zapdos being the application for plasma simulation.

Zapdos uses the multi-fluid approach (i.e. that each particle is treated with its own continuity equation). The electric field is treated with the electrostatic approximation and currently there is no magnetic field treatment in Zapdos. It is worth noting that there is currently an electromagnetic solver in development by NCSSU's Casey Icenhour that can be coupled to Zapdos in the future. The plasma chemistry within Zapdos is treated by MOOSE's general chemistry application, CRANE. CRANE was developed by Shane Keniley at University of Illinois at Urbana-Champaign. The inner working of CRANE will not be discussed in this document, but the process of using CRANE within Zapdos will be discussed. For a more in-depth overview of CRANE, please visit <https://github.com/lcpp-org/crane>.

The installation instructions for Zapdos can be found at <https://shannon-lab.github.io/zapdos/getting-started/>. By using these instructions, CRANE will be downloaded as a submodule of Zapdos. CRANE is then built as a library whose objects are accessible within Zapdos, enabling direct coupling of the two codes.

## Chapter 2

# Brief Introduction to FEM and MOOSE Objects

### 2.1 FEM: Strong to Weak Form

The complete process of the FEM will not be discussed in this tutorial document, but the main concept of transforming the strong form of an equation to the weak form will be reviewed. The equations in MOOSE/Zapdos are coded in the weak form, so understanding this concept is useful when adding physics to Zapdos. The strong form refers to the differential form of an equation. For an example, lets look at the classic diffusion problem of:

$$\nabla \cdot -D\nabla u = s(\mathbf{r}) \quad (2.1)$$

Where  $D$  is the diffusion coefficient,  $u$  is the species of diffusion, and  $s(\mathbf{r})$  is the source term. Equation 2.1 would be classified as the strong form of the diffusion problem. To convert this into the weak form for FEM, one must multiply the equation by a test function  $\psi_i$  and integrate over the whole domain of interest. All the terms will also be moved over to the left-hand side (LHS), resulting in the following:

$$\int_{\Omega} \psi_i \nabla \cdot -D\nabla u dV - \int_{\Omega} s(x) dV = 0 \quad (2.2)$$

Where  $\Omega$  is the extent of the domain. Now the first term can be treated by using integration by parts, which gives:

$$\int_{\Omega} \nabla \psi_i \cdot D\nabla u dV - \int_{\partial\Omega} \psi_i D\nabla u \cdot \mathbf{n} dS - \int_{\Omega} s(x) dV = 0 \quad (2.3)$$

where  $\partial\Omega$  is the domain boundaries and  $\mathbf{n}$  is the normal vector to the boundary. Equation 2.3 contains the “kernels” (e.g. the first and third term) and boundary conditions (e.g. the second term) that are entered into MOOSE/Zapdos. The rest of the FEM involves the discretization of the equation. For the purpose of this tutorial document, that process will be overlooked, but the fine details of the discretization process can

be found in the third chapter of [1]. In the example problems in this tutorial, the continuous Galerkin finite element method is utilized with linear Lagrange test functions. Solution of the resulting system of equations is obtained via Newton's Method.

## 2.2 MOOSE/Zadpos Objects

The rest of this chapter will consist of summaries for MOOSE/Zadpos object blocks. The object blocks are how input files are broken up into different significant parameters. These blocks can fall into one of the following categories: GlobalParams, Mesh, MeshModifiers, Problem, Variables, Kernels, AuxVariables, AuxKernels, Interfacekernels, BCs, ICs, Functions, Materials, Postprocessors, Preconditioning, Executioner, TimeStepper, and Outputs.

### 2.2.1 GlobalParams

The GlobalParams block is for defining parameters that appear in multiple blocks in a given input file. This is useful in particular for Zapdos, since Zapdos has scaling options for voltage and distance to better help with the conditioning of problems (e.g. sometimes it is computationally easier to define voltage in kV instead of V).

### 2.2.2 Mesh and MeshModifiers

The Mesh block is for creating a mesh or reading existing mesh files. MOOSE has its own mesh building feature – more information can be found here: <https://mooseframework.org/source/mesh/GeneratedMesh>. MOOSE/Zadpos can also read a variety of mesh files. For the examples in this tutorial document, the meshes are created using the finite-element mesh generator, Gmsh [6]. Naming of boundaries and domains can be defined in the mesh file itself, or in the MeshModifiers block. As the name suggest, the MeshModifier block modifies the mesh defined in the Mesh block. Such modifications can include stretching low dimension meshes into higher dimension and defining boundaries. For the following 1D example, a MeshModifier is used to defined to end points of the 1D mesh as boundaries.

### 2.2.3 Problem

The Problem block is where the type of problem to be solved is defined. By default, this is set to be FEProblem (containing one Nonlinear System to be solved), but other options (such as that for Eigenvalue problems) are available. Also, the coordinate system of the problems can be set in the Problem block. For the 2D example, an axial symmetric cylindrical system is used.

### 2.2.4 Variables and Kernels

The Variables block is where the variables of the problem are defined. As mentioned in Section 2.1, the default setting is a linear Lagrange basis function. The selection of these functions can highly depend on the types of problems and are beyond the scope of this tutorial. The Kernels block contains the physics of the problems, and in particular, the Kernel block contains the volume integrals found in the weak form of the set of equations (e.g. the first and third term of Equation 2.3).

### 2.2.5 AuxVariables and AuxKernels

The AuxVariables and AuxKernels blocks are similar to the Variables and Kernels blocks, but they contain for variables and terms not essential for solving the problem, but that still provide useful information. For an example, the electron density and potential would be considered variables, since they are essential for solving the multi-fluid method of plasma. The value of the electron current would be considered an AuxVariable (or auxiliary variable), since the current can be defined in terms of the electron density and potential.



### 2.2.6 InterfaceKernels

The InterfaceKernels block contains constraints or relationships between variables at the boundary of two domains. In order to use InterfaceKernels, interface boundaries must be defined to set the location of the constraint or relationship. In Zapdos, InterfaceKernels are used for equating the electron flux at the plasma-liquid interface.

### 2.2.7 BCs

The BCs block is for defining the boundary conditions of the problem. In FEM, there are two main types of boundary conditions, Dirichlet and Integrated BCs. The Dirichlet BC is an "NodalBC" in MOOSE (which is not integrated over the boundary) and directly defines the value of the variable at the boundary. For Zapdos, Dirichlet BCs are used to set the potential at boundaries to a sin function for RF discharges or zero for grounded walls. Integrated BCs are integrated over the boundary and are used for flux BCs in Zapdos.

### 2.2.8 ICs

The ICs block defines the initial conditions of the variables of the problem. Variables initial states can be setup as a uniform constant throughout the mesh, or be defined as a function of position.

### 2.2.9 Functions

The Functions block defines any functions that are used in the other blocks. Functions can be defined by position and time. Custom, parsed functions (via the ParsedFunction object) also allows for direct usage of common functions, such as trigonometric and logarithmic functions.

### 2.2.10 Materials

The Material block is used to define material properties within the mesh. In Zapdos, these properties include mass, mobility, diffusion coefficients for charged and neutral species, common physical constants, and reaction rates. It is worth noting that the reaction rates and electron mobility/diffusion coefficients are tabulated in a table provided by the user in terms of mean electron energy (usually solved from a Boltzmann solver). All rates and electron mobility/diffusion coefficients used in the tutorial examples comes from the work done by Lymberopoulos in [3] or by BOLSIG+.

### 2.2.11 Postprocessors

The Postprocessors block defines a variety of postprocessing operations. In the following examples, the post-processor of InversePlasmaFreq is used to find the inverse of the peak plasma frequency to help define the time steps.

### 2.2.12 Preconditioning

The Preconditioning block is to define any preconditioning processes needed to solve the problems. The following examples in this tutorial uses SMP (Single Matrix Preconditioner), which builds one preconditioning matrix based on the Jacobians in the Kernels. This setting requires more advanced understanding of solving nonlinear systems of equations and is beyond the scope of this tutorial.

### 2.2.13 Executioner

The Executioner block defines whether the problem to be solved is transient or steady-state, as well as various solver settings. For the problems in Zapdos, this setting is usually set to transient. For a transient

problem, the end time, max time step and min time step can be defined in this block. For more advanced users, convergence tolerance and PETSc solver options are also defined in the Executioner block.

### **2.2.14 TimeStepper**

The TimeStepper block determines the definition of the time steps. The two main time steppers used in Zapdos are IterationAdaptiveDT and PostprocessorDT. IterationAdaptiveDT will increase or decrease the time step depending of the convergence rate of the problem. PostprocessorDT uses a post-processor output to determine the time step (e.g. PostprocessorDT with the InversePlasmaFreq post-processor defines the time step as the inverse of the peak plasma frequency).

### **2.2.15 Outputs**

The Output block defines the type of file used for the output file. The default output file type is an exodus file, which can be read with visualization application (such as ParaView). The name of the output file can also be defined in this block.

## Chapter 3

# Physics of Zapdos

### 3.1 Weak Form of Multi-Fluid Plasmas

The physics of Zapdos are based on the multi-fluid approach to plasma simulation. With this approach, each particle is simulated with its own continuity equation in the form of:

$$\frac{\partial n_j}{\partial t} + \nabla \cdot \mathbf{\Gamma}_j = \sum \mathbf{S}_{iz} \quad (3.1)$$

Where  $n$  is the density,  $j$  is the subscript for either electrons ( $e$ ), ions ( $i$ ) or metastables ( $*$ ),  $\mathbf{\Gamma}$  is the flux of the species, and  $\mathbf{S}_{iz}$  is the source terms. The source terms are treated by CRANE and for the purpose of this tutorial, CRANE can handle one, two and three body reaction rates. It should be noted that Zapdos and CRANE can also treat the source terms in the form of Townsend coefficients. Since the tutorial example problems use reaction rates, that is the form discussed in this document. The form of these sources terms are:

One Body:

$$\mathbf{S}_{iz} = vk n_1$$

Two Body:

$$\mathbf{S}_{iz} = vk n_1 n_2 \quad (3.2)$$

Three Body:

$$\mathbf{S}_{iz} = vk n_1 n_2 n_3$$

Where  $v$  is the stoichiometric coefficient and  $k$  is the rate coefficient. The flux of the particles are in the form of:

$$\mathbf{\Gamma}_j = \mu_j \mathbf{E} n_j - D_j \nabla n_j \quad (3.3)$$

Where  $\mu$  is the mobility coefficient,  $\mathbf{E}$  is the electric field, and  $D$  is the diffusivity coefficient. For the use of metastables, the flux would be the same as equation 3.3, except there would not be an electric field advection term. In Zapdos, the electrostatic approximation is used, which results in  $\mathbf{E} = -\nabla V$  ( $V$  being

the electrostatic potential). Now, transforming the strong form of the Equations 3.1 and 3.2 into the weak from, we can get the kernel equations used in Zapdos.

$$\underbrace{\int_{\Omega} \psi_i \frac{\partial n_j}{\partial t} dV}_{\text{ElectronTimeDerivative for e,i,*}} - \underbrace{\int_{\Omega} \nabla \psi_i \cdot \mu_j \nabla V n_j dV}_{\text{CoeffDiffusionElectrons for e, CoeffDiffusion for i,*}} + \underbrace{\int_{\Omega} \nabla \psi_i \cdot D_j \nabla n_j dV}_{\text{CoeffDiffusionElectrons for e, CoeffDiffusion for i,*}} + \underbrace{\int_{\partial\Omega} \psi_i \nabla \Gamma_j \cdot \mathbf{n} dS}_{\text{BCs}} - \underbrace{\int_{\Omega} \psi_i \sum \mathbf{S}_{i\mathbf{z}} dV}_{\text{CRANE Kernels}} = 0 \quad (3.4)$$

$$\begin{aligned}
 & \text{CRANE Kernels} \\
 & \underbrace{\int_{\Omega} \psi_i v k n_1 dV}_{\text{'Product/Reactant'FirstOrderLog}} \\
 & \text{Two Body:} \\
 & \underbrace{\int_{\Omega} \psi_i v k n_1 n_2 dV}_{\text{Electron'Product/Reactant'SecondOrderLog for e and 'Product/Reactant'SecondOrderLog for i,*}} \\
 & \text{Three Body:} \\
 & \underbrace{\int_{\Omega} \psi_i v k n_1 n_2 dV}_{\text{'Product/Reactant'ThirdOrderLog}}
 \end{aligned} \quad (3.5)$$

Where the under and over braces are the names of the kernels for each species. For CRANE, what determines the uses of the 'Product' or 'Reactant' version of a kernels is whether the variable being solved in that kernel is a product or reactant (more information in the example files). The reason that electrons have different kernels from the ions and metastables is due to the fact the the mobility and diffusion coefficients are a function of the electron mean energy and they require a more complex Jacobian definition. The electron mean energy is also treated as a separate variable and requires its own continuity equation, in the form of:

$$\frac{\partial n_e \varepsilon}{\partial t} + \nabla \cdot \Gamma_{\varepsilon} = e \Gamma_e \cdot \nabla V - 3 \frac{m_e}{m_g} n_e n_g k_{elastic} T_e - \sum_j E_j K_j \quad (3.6)$$

Where  $\varepsilon$  is the mean electron energy,  $e$  is the elementary charge,  $m_e$  is the electron mass,  $m_g$  is the mass of the background gas,  $n_g$  is the background gas density,  $k_{elastic}$  is the elastic rate coefficient,  $T_e$  is the electron temperature (i.e.  $T_e = \frac{2}{3} \varepsilon$ ),  $E_j$  is the threshold energy of inelastic collisions, and  $K_j$  is the inelastic rate coefficient. It should be noted that Zapdos can also treat the elastic and inelastic mean energy losses in the form of Townsend coefficients. Since the tutorial example problems use reaction rates, that will the form discussed in this document. The flux of the mean electron energy is in the form of:

$$\Gamma_{\varepsilon} = \frac{5}{3} \varepsilon \Gamma_e - \frac{5}{3} D_e n_e \nabla \varepsilon \quad (3.7)$$

When using the transformation of the strong form of the Equation 3.6 into the weak from, the electron mean energy kernel equations in Zapdos become:

$$\begin{aligned}
& \underbrace{\int_{\Omega} \psi_i \frac{\partial n_e \varepsilon}{\partial t} dV}_{\text{ElectronTimeDerivative}} - \underbrace{\int_{\Omega} \nabla \psi_i \cdot \mu_{\varepsilon} \nabla V n_e \varepsilon dV}_{\text{EFieldAdvectionEnergy}} + \underbrace{\int_{\Omega} \nabla \psi_i \cdot D_{\varepsilon} \nabla n_e \varepsilon dV}_{\text{CoeffDiffusionEnergy}} + \underbrace{\int_{\partial\Omega} \psi_i \nabla \Gamma_{\varepsilon} \cdot \mathbf{n} dS}_{\text{BCs}} - \underbrace{\int_{\Omega} \psi_i e \Gamma_{\mathbf{e}} \cdot \nabla V dV}_{\text{JouleHeating}} \\
& + \underbrace{\int_{\Omega} \psi_i 3 \frac{m_e}{m_g} n_e n_g k_{elastic} T_e dV}_{\text{ElectronEnergyTermElasticRate}} + \underbrace{\int_{\Omega} \psi_i \sum_j E_j K_j dV}_{\text{ElectronEnergyTermRate}} = 0
\end{aligned} \tag{3.8}$$

To close the set of equations, the electrostatic approximation version of the Poisson's equation is used in the form of:

$$-\nabla^2 V = \frac{e(n_i - n_e)}{\varepsilon_0} \tag{3.9}$$

Where  $\varepsilon_0$  is the permittivity of free space. The weak form the the Poisson's equation and potential kernel equations are:

$$\begin{aligned}
& \underbrace{\int_{\Omega} \nabla \psi_i \varepsilon_0 \nabla V dV}_{\text{CoeffDiffusionLin}} - \underbrace{\int_{\partial\Omega} \psi_i \varepsilon_0 \nabla V \cdot \mathbf{n} dS}_{\text{BCs}} - \underbrace{\int_{\Omega} \psi_i e (n_i - n_e) dV}_{\text{ChargeSourceMoles.KV}}
\end{aligned} \tag{3.10}$$

## 3.2 Boundary Conditions

When looking at the weak form of an equation, one might notice the boundary condition in the form of the surface integral. For Zapdos, the boundary conditions for the density species are either treated as a Dirichlet BC with a set density value at the boundary or they are defined by the flux normal to the boundary. Currently there are three sets of flux boundary conditions: Lymberopoulos BCs, Sakiyama BCs, and Hagelaar BCs (which are based on the following papers: [3, 2, 7, 8]).

### 3.2.1 Lymberopoulos BCs

The Lymberopoulos BCs consist of only BCs for electrons and charged particles. The form of the normal electron flux is:

$$\begin{aligned}
& \text{LymberopoulosElectronBC:} \\
& \Gamma_{\mathbf{e}} \cdot \mathbf{n} = \mp k_s n_e - \gamma \Gamma_{\mathbf{i}} \cdot \mathbf{n}
\end{aligned} \tag{3.11}$$

Where  $k_s$  is the electron surface recombination coefficient assuming an electron sticking coefficient of unity and  $\gamma$  is the secondary electron coefficient. As can be seen in equation 3.11, the Lymberopoulos BCs for electrons consists of a thermal term that assumes a constant electron mean energy at the walls (the first term on the LHS) and a secondary electron term defined by a emission coefficient multiplied by the ion flux (the second term on the LHS). The normal ion flux is only defined by the advection term, such that:

$$\begin{aligned} &\text{LymberopoulosIonBC:} \\ &\mathbf{\Gamma}_i \cdot \mathbf{n} = -\mu_i n_i \nabla V \cdot \mathbf{n} \end{aligned} \quad (3.12)$$

### 3.2.2 Sakiyama BCs

The Sakiyama BCs for the normal fluxes are similar to the Lymberopoulos BCs, except they add some more complexity and there is a normal flux for the mean electron energy. The normal electron flux now does not assume a constant electron mean energy at the walls and is determined by:

$$\begin{aligned} &\text{SakiyamaElectronDiffusionBC \& SakiyamaSecondaryElectronBC:} \\ &\mathbf{\Gamma}_e \cdot \mathbf{n} = \frac{1}{4} \sqrt{\frac{8k_b T_e}{\pi m_e}} n_e - \gamma \mathbf{\Gamma}_i \cdot \mathbf{n} \end{aligned} \quad (3.13)$$

Where  $k_b$  is Boltzmann's constant. The normal ion flux is the same as the LymberopoulosIonBC, except now the ion flux is only present when the electric field at the wall is positive (or when the gradient of the potential is negative).

$$\begin{aligned} &\text{LymberopoulosIonBC:} \\ &\mathbf{\Gamma}_i \cdot \mathbf{n} = -\mu_i n_i \nabla V_{eff} \cdot \mathbf{n}, \text{ if } \nabla V_{eff} \cdot \mathbf{n} \geq 0 \text{ else } \mathbf{\Gamma}_i \cdot \mathbf{n} = 0 \end{aligned} \quad (3.14)$$

The normal flux for the mean electron energy contains the same terms as the electrons and is defined as:

$$\begin{aligned} &\text{SakiyamaEnergyDiffusionBC \& SakiyamaEnergySecondaryElectronBC:} \\ &\mathbf{\Gamma}_\varepsilon \cdot \mathbf{n} = \left(\frac{5}{3}\varepsilon\right) \frac{1}{4} \sqrt{\frac{8k_b T_e}{\pi m_e}} n_e - \left(\frac{5}{3}\varepsilon\right) \gamma \mathbf{\Gamma}_i \cdot \mathbf{n} \end{aligned} \quad (3.15)$$

### 3.2.3 Hagelaar BCs

The Hagelaar BCs are currently the most complete definitions for the normal fluxes. All of the Hagelaar BCs contain a boundary reflection coefficient. Along with this, the normal fluxes for electron and mean electron energy now have an advection term and the normal ion flux has a thermal velocity term. The Hagelaar BCs are:

$$\begin{aligned} &\text{HagelaarElectronBC \& SecondaryElectronBC:} \\ &\mathbf{\Gamma}_e \cdot \mathbf{n} = \frac{1 - r_{dens}}{1 + r_{dens}} \left( -(2a_e - 1) \mu_e \nabla V \cdot \mathbf{n} (n_e - n_\gamma) + \frac{1}{2} v_{th,e} (n_e - n_\gamma) \right) - (1 - a_e) \gamma_p \mathbf{\Gamma}_p \cdot \mathbf{n} \end{aligned} \quad (3.16)$$

$$\begin{aligned} &\text{HagelaarIonAdvectionBC \& HagelaarIonDiffusionBC:} \\ &\mathbf{\Gamma}_\varepsilon \cdot \mathbf{n} = \frac{1 - r_i}{1 + r_i} \left( (2a_i - 1) \mu_i \nabla V \cdot \mathbf{n} n_i + \frac{1}{2} v_{th,i} n_i \right) \end{aligned} \quad (3.17)$$

$$\begin{aligned} &\text{HagelaarEnergyBC:} \\ &\mathbf{\Gamma}_\varepsilon \cdot \mathbf{n} = \frac{1 - r_{ens}}{1 + r_{ens}} \left( -(2a_e - 1) \frac{5}{3} \mu_e \nabla V \cdot \mathbf{n} (n_e \varepsilon - n_\gamma \varepsilon) + \frac{5}{6} v_{th,e} (n_e \varepsilon - n_\gamma \varepsilon) \right) - \frac{5}{3} \varepsilon_\gamma (1 - a_e) \gamma_p \mathbf{\Gamma}_p \cdot \mathbf{n} \end{aligned} \quad (3.18)$$

Where  $r_i$ ,  $r_{dens}$  and  $r_{en}$  are the boundary reflection coefficients for ions, electrons, and electron mean energy respectively,  $\varepsilon_\gamma$  is the energy of the secondary electrons, and:

$$\begin{aligned} a_k &= 1, \text{sgn}_k \mu_k \mathbf{E} \cdot \mathbf{n} > 0 \\ &0, \text{sgn}_k \mu_k \mathbf{E} \cdot \mathbf{n} \geq 0 \end{aligned} \quad (3.19)$$

$$v_{th,k} = \sqrt{\frac{8T_k}{\pi m_k}} \quad (3.20)$$

$$n_\gamma = (1 - a_e) \frac{\gamma_p \Gamma_p \cdot \mathbf{n}}{\mu_e \mathbf{E} \cdot \mathbf{n}} \quad (3.21)$$

Where  $v_{th,k}$  is the thermal velocity of species  $k$  and  $n_\gamma$  is the density of secondary electrons.

### 3.2.4 Dirichlet Density BCs

The other form of BCs for the densities and mean electron energy is Dirichlet BC, which are:

$$\begin{aligned} &\text{LogDensityDirichletBC:} \\ &n_j = \text{value} \end{aligned} \quad (3.22)$$

$$\begin{aligned} &\text{ElectronTemperatureDirichletBC:} \\ &\varepsilon n_e = \frac{2}{3} \frac{\varepsilon}{n_e} \end{aligned} \quad (3.23)$$

The reason for the electron temperature is in terms of  $n_e$  is because Zapdos solves the mean electron energy as mean electron energy density (this will be explained more in the next section).

### 3.2.5 Potential BCs

The last sets of BCs are for the potential. Zapdos uses a functional Dirichlet BC for simulating RF discharges, in the form of:

$$\begin{aligned} &\text{FunctionDirichletBC:} \\ &V = V_{amp} \sin(ft) \end{aligned} \quad (3.24)$$

Where  $V_{amp}$  is the peak amplitude of the voltage,  $f$  is the frequency, and  $t$  is the time. Along with modeling RF discharges, Zapdos can modeling DC discharges using Kirchoff's voltage law in the form of:

$$\begin{aligned} &\text{NeumannCircuitVoltageMoles_KV:} \\ &V_{source} + V_{cathode} = (e\Gamma_i - e\Gamma_e)AR \end{aligned} \quad (3.25)$$

Where  $A$  is the cross-sectional area of the plasma and  $R$  is the ballast resistance. It should be noted that this BC uses an integrated BC by solving for the gradient of the potential in the fluxes. The newest potential BC in Zapdos is a dielectric BC (based on [2]) and in the form of:

EconomouDielectricBC:

$$\frac{\partial V}{\partial t} = \frac{d_i}{\varepsilon_i} \left( e\Gamma_i - e\Gamma_e + \varepsilon_0 \frac{\partial \nabla V}{\partial t} \right) \quad (3.26)$$

Where  $\varepsilon_i$  and  $d_i$  are the permittivity and effective thickness of the insulator.

### 3.3 Treatment of Variables and Scaling Methods

To better help with convergence, the densities are treated in log form and the mean electron energy is treated as a mean electron energy density (i.e. as the product of electron density and mean electron energy). This means Zapdos tries to solve the density, mean energy, and potential in the plasma in the form of:

$$N_j = \ln(n_j) \quad (3.27)$$

$$E_n = \ln(n_e \varepsilon) \quad (3.28)$$

$$V = V \quad (3.29)$$

Along with transforming the density variables into log form, Zapdos also has scaling methods in place. These include having the option to solve for the potential in terms of volts or kilovolts, setting the units of the density to moles/m<sup>3</sup>, and creating a unity mesh then scaling the mesh. The idea behind these scaling methods is to reduce the variation in magnitudes of the variable (and thus the entries in the Jacobian matrix) which should better help solve convergence.



## Chapter 4

# GEC Reference Cell Tutorial Examples

### 4.1 Zapdos Input Files and Peacock

The following tutorial file can either be viewed in a text editor (such as Atom) or in MOOSE's GUI, Peacock. The input file is broken up into different blocks and within these blocks, the user defines different kernels, BCs, materials, etc. Below is an example of the structure of a block within the input.

```
[Kernels] #Beginning/Name of Object Block (This example is the Kernel Block)
#Time Derivative of electrons
[./em_time_deriv] #Beginning/Custom Name of kernel (User Defined)
    type = ElectronTimeDerivative #Type of kernel
    variable = em #Variable effected by kernel
[./] #End of first kernel
#Advection term of electron
[./em_advection]
    type = EFieldAdvectionElectrons
    variable = em
    potential = potential #Coupled variable
    mean_en = mean_en #Coupled variable
    position_units = ${dom0Scale} #Scaling Option
[./]
#Diffusion term of electrons
[./em_diffusion]
    type = CoeffDiffusionElectrons
    variable = em
    mean_en = mean_en
    position_units = ${dom0Scale}
[./]
[] #End of Object Block
```

Coding an input file can be tedious, and it is possible to often miss required inputs for new users. To

make the creation of an input file easy, there is Peacock. Peacock is MOOSE's GUI and it can create input files using drag down menus. More information about Peacock can be found at <https://mooseframework.inl.gov/old/wiki/Peacock/>. Figures 4.1 and 4.2 shows the Peacock interface.

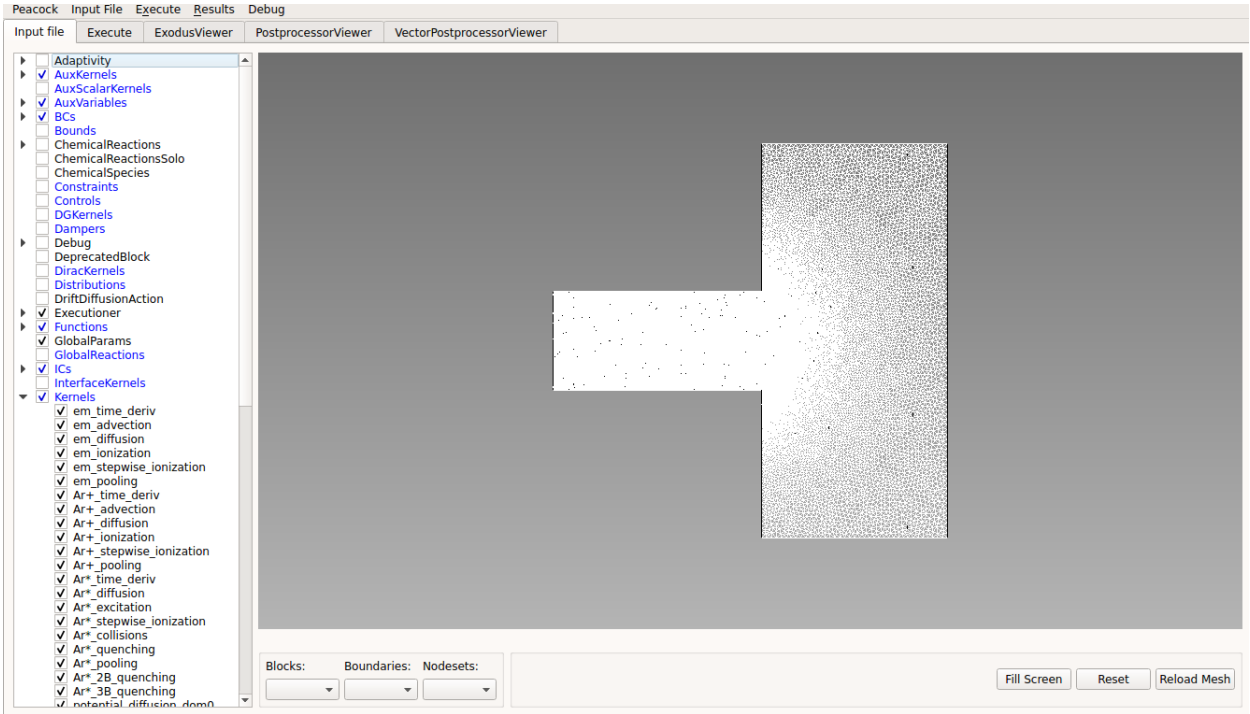


Figure 4.1: Example of a Zapdos input file in Peacock

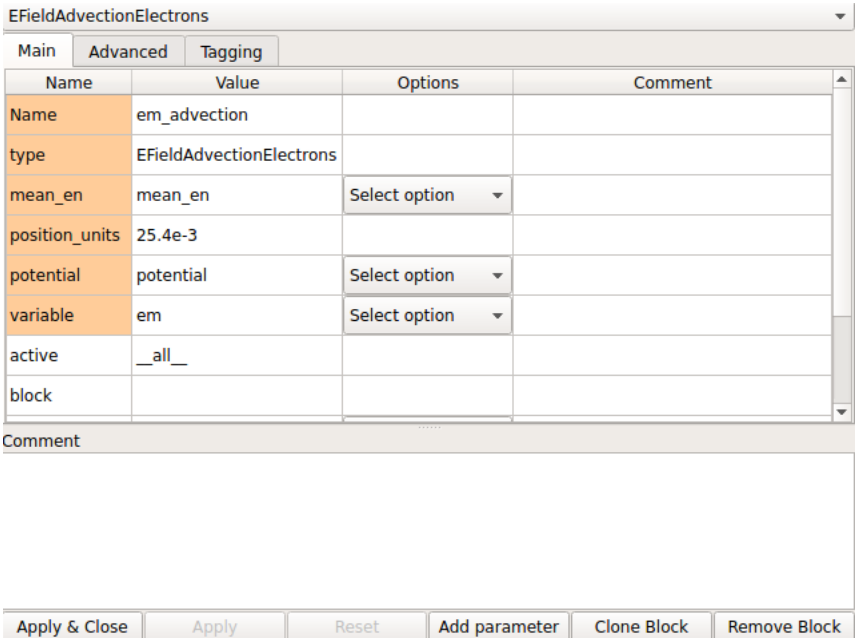


Figure 4.2: Example for a kernel option menu in Peacock

## 4.2 1D: CCP Discharge

This tutorial is based on the work by Lymberopoulos [3] and the input file can be found as `zapdos/problems/Lymberopoulos_with_argon_metastables.i`. This input file simulates a 1Torr parallel plate argon plasma with one electrode powered at 100V 13.56Hz RF and the other electrode grounded. This simulation also includes argon metastables. The contents of the input can be found in Appendix A1.

## 4.3 2D: GEC Reference Cell

This tutorial is based on the work by Lymberopoulos [2] and the input file can be found as `zapdos/problems/Lymberopoulos_with_argon_metastables_2D_At100mTorr.i`. This input file simulates a 100 mTorr argon plasma within the GEC reference cell. The top and bottom electrodes are powered in such a way that the peak-to-peak voltage is 100V at 13.56Hz and the walls of the chamber is grounded. There is also a thin piece of insulator that acts as a dielectric boundary and this simulation includes argon metastables. The contents of the input can be found in Appendix A2.

## 4.4 Hands-on Work

### 4.4.1 GEC at Higher Pressures

In this exercise, the goal is to replicate the discharge parameters found in the work due by Lymberopoulos in [4]. One may notice this discharge is very similar to the one the 100m Torr case in the previous example, expect that the pressure is 1Torr and the peak-to-peak voltage is 60V. For help with this exercise, the completed input file can be found at `zapdos/problems/Lymberopoulos_with_argon_metastables_2D_At1Torr.i`.

### 4.4.2 The Plasma-Liquid Interface

A good next step after understanding simulating only plasma is to try to replicate the works by Lindsay in [5]. This work includes `InterfaceKernels` and two different domains. In order to use `InterfaceKernels`, one must setup interface boundaries with the `MeshModifier` block. Examples code for the `MeshModifier` and `InterfaceKernels` blocks are below. Note: in order to get atmospheric runs working, Zapdos currently needs to use the `MoleStabilizing Kernel` with a value of 20. For help with this exercise, the completed input file can be found at `zapdos/problems/mean_en.i`.

```
[MeshModifiers]
  #Setting the path from plasma to water
  [./interface]
    type = SideSetsBetweenSubdomains
    master_block = '0' #plasma
    paired_block = '1' #water
    new_boundary = 'master0_interface'
  [../]
[]

[InterfaceKernels]
  #Defining election advection to the water
  [./em_advection]
    type = InterfaceAdvection
    mean_en_neighbor = mean_en
```

```

    potential_neighbor = potential
    neighbor_var = em    #electrons in plasma
    variable = emliq    #electrons in water
    boundary = master1_interface
    position_units = ${dom1Scale}
    neighbor_position_units = ${dom0Scale}
[../]
#Defining election diffusion to the water
[/em_diffusion]
    type = InterfaceLogDiffusionElectrons
    mean_en_neighbor = mean_en
    neighbor_var = em
    variable = emliq
    boundary = master1_interface
    position_units = ${dom1Scale}
    neighbor_position_units = ${dom0Scale}
[../]
[]

```

## Chapter 5

# Appendix

### A1: Input Code: Lymberopoulos\_with\_argon\_metastables.i

```
dom0Scale=25.4e-3

[GlobalParams]
  potential_units = kV
  use_moles = true
[]

[Mesh]
  type = FileMesh
  file = 'Lymberopoulos.msh'
[]

[MeshModifiers]
  [./left]
    type = SideSetsFromNormals
    normals = '-1 0 0'
    new_boundary = 'left'
  [../]
  [./right]
    type = SideSetsFromNormals
    normals = '1 0 0'
    new_boundary = 'right'
  [../]
[]

[Problem]
  type = FEProblem
[]

[Variables]
  [./em]
  [../]

  [./Ar+]
  [../]

  [./Ar*]
  [../]

  [./mean-en]
  [../]

  [./potential]
  [../]
[]

[Kernels]
  #Electron Equations (Same as in paper)
```

```

#Time Derivative term of electron
[/em.time-deriv]
  type = ElectronTimeDerivative
  variable = em
[../]
#Advection term of electron
[/em.advection]
  type = EFieldAdvectionElectrons
  variable = em
  potential = potential
  mean_en = mean_en
  position_units = ${dom0Scale}
[../]
#Diffusion term of electrons
[/em.diffusion]
  type = CoeffDiffusionElectrons
  variable = em
  mean_en = mean_en
  position_units = ${dom0Scale}
[../]
#Net electron production from ionization
[/em.ionization]
  type = ElectronReactantSecondOrderLog
  variable = em
  v = Ar
  energy = mean_en
  reaction = 'em + Ar -> em + em + Ar+'
  coefficient = 1
[../]
#Net electron production from step-wise ionization
[/em.stepwise.ionization]
  type = ElectronReactantSecondOrderLog
  variable = em
  v = Ar*
  energy = mean_en
  reaction = 'em + Ar* -> em + em + Ar+'
  coefficient = 1
[../]
#Net electron production from metastable pooling
[/em.pooling]
  type = ProductSecondOrderLog
  variable = em
  v = Ar*
  w = Ar*
  reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
  coefficient = 1
[../]

#Argon Ion Equations (Same as in paper)
#Time Derivative term of the ions
[/Ar+_time-deriv]
  type = ElectronTimeDerivative
  variable = Ar+
[../]
#Advection term of ions
[/Ar+_advection]
  type = EFieldAdvection
  variable = Ar+
  potential = potential
  position_units = ${dom0Scale}
[../]
[/Ar+_diffusion]
  type = CoeffDiffusion
  variable = Ar+
  position_units = ${dom0Scale}
[../]
#Net ion production from ionization
[/Ar+_ionization]
  type = ElectronProductSecondOrderLog
  variable = Ar+
  electron = em
  target = Ar
  energy = mean_en
  reaction = 'em + Ar -> em + em + Ar+'
  coefficient = 1
[../]
#Net ion production from step-wise ionization

```

```

[./Ar+_stepwise_ionization]
type = ElectronProductSecondOrderLog
variable = Ar+
electron = em
target = Ar*
energy = mean_en
reaction = 'em + Ar* -> em + em + Ar+'
coefficient = 1
[../]
#Net ion production from metastable pooling
[./Ar+_pooling]
type = ProductSecondOrderLog
variable = Ar+
v = Ar*
w = Ar*
reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
coefficient = 1
[../]

#Argon Excited Equations (Same as in paper)
#Time Derivative term of excited Argon
[./Ar*_time_deriv]
type = ElectronTimeDerivative
variable = Ar*
[../]
#Diffusion term of excited Argon
[./Ar*_diffusion]
type = CoeffDiffusion
variable = Ar*
position_units = ${dom0Scale}
[../]
#Net excited Argon production from excitation
[./Ar*_excitation]
type = ElectronProductSecondOrderLog
variable = Ar*
electron = em
target = Ar
energy = mean_en
reaction = 'em + Ar -> em + Ar*'
coefficient = 1
[../]
#Net excited Argon loss from step-wise ionization
[./Ar*_stepwise_ionization]
type = ElectronProductSecondOrderLog
variable = Ar*
electron = em
target = Ar*
energy = mean_en
reaction = 'em + Ar* -> em + em + Ar+'
coefficient = -1
_target_eq_u = true
[../]
#Net excited Argon loss from superelastic collisions
[./Ar*_collisions]
type = ElectronProductSecondOrderLog
variable = Ar*
electron = em
target = Ar*
energy = mean_en
reaction = 'em + Ar* -> em + Ar'
coefficient = -1
_target_eq_u = true
[../]
#Net excited Argon loss from quenching to resonant
[./Ar*_quenching]
type = ElectronProductSecondOrderLog
variable = Ar*
electron = em
target = Ar*
energy = mean_en
reaction = 'em + Ar* -> em + Ar_r'
coefficient = -1
_target_eq_u = true
[../]
#Net excited Argon loss from metastable pooling
[./Ar*_pooling]
type = ReactantSecondOrderLog

```

```

        variable = Ar*
        v = Ar*
        reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
        coefficient = -2
        -v.eq-u = true
    [../]
#Net excited Argon loss from two-body quenching
[./Ar*_2B_quenching]
    type = ReactantSecondOrderLog
    variable = Ar*
    v = Ar
    reaction = 'Ar* + Ar -> Ar + Ar'
    coefficient = -1
    [../]
#Net excited Argon loss from three-body quenching
[./Ar*_3B_quenching]
    type = ReactantThirdOrderLog
    variable = Ar*
    v = Ar
    w = Ar
    reaction = 'Ar* + Ar + Ar -> Ar_2 + Ar'
    coefficient = -1
    [../]

#Voltage Equations (Same as in paper)
#Voltage term in Poissons Equation
[./potential_diffusion_dom0]
    type = CoeffDiffusionLin
    variable = potential
    position_units = ${dom0Scale}
    [../]
#Ion term in Poissons Equation
[./Ar+_charge_source]
    type = ChargeSourceMoles_KV
    variable = potential
    charged = Ar+
    [../]
#Electron term in Poissons Equation
[./em_charge_source]
    type = ChargeSourceMoles_KV
    variable = potential
    charged = em
    [../]

#Since the paper uses electron temperature as a variable, the energy equation is in
#a different form but should be the same physics
#Time Derivative term of electron energy
[./mean_en_time_deriv]
    type = ElectronTimeDerivative
    variable = mean_en
    [../]
#Advection term of electron energy
[./mean_en_advection]
    type = EFieldAdvectionEnergy
    variable = mean_en
    potential = potential
    em = em
    position_units = ${dom0Scale}
    [../]
#Diffusion term of electrons energy
[./mean_en_diffusion]
    type = CoeffDiffusionEnergy
    variable = mean_en
    em = em
    position_units = ${dom0Scale}
    [../]
#Joule Heating term
[./mean_en_joule_heating]
    type = JouleHeating
    variable = mean_en
    potential = potential
    em = em
    position_units = ${dom0Scale}
    [../]
#Energy loss from ionization
[./Ionization_Loss]

```



```

    type = ElectronEnergyTermRate
    variable = mean_en
    em = em
    v = Ar
    reaction = 'em + Ar -> em + em + Ar+'
    threshold_energy = -15.7
    position_units = ${dom0Scale}
[../]
#Energy loss from excitation
[/Excitation_Loss]
    type = ElectronEnergyTermRate
    variable = mean_en
    em = em
    v = Ar
    reaction = 'em + Ar -> em + Ar*'
    threshold_energy = -11.56
    position_units = ${dom0Scale}
[../]
#Energy loss from step-wise ionization
[/Stepwise_Ionization_Loss]
    type = ElectronEnergyTermRate
    variable = mean_en
    em = em
    v = Ar*
    reaction = 'em + Ar* -> em + em + Ar+'
    threshold_energy = -4.14
    position_units = ${dom0Scale}
[../]
#Energy gain from superelastic collisions
[/Collisions_Loss]
    type = ElectronEnergyTermRate
    variable = mean_en
    em = em
    v = Ar*
    reaction = 'em + Ar* -> em + Ar'
    threshold_energy = 11.56
    position_units = ${dom0Scale}
[../]
[]

[AuxVariables]
[/emDeBug]
[../]
[/Ar+_DeBug]
[../]
[/Ar*_DeBug]
[../]
[/mean_enDeBug]
[../]

[/Te]
    order = CONSTANT
    family = MONOMIAL
[../]

[/x]
    order = CONSTANT
    family = MONOMIAL
[../]

[/x_node]
[../]

[/rho]
    order = CONSTANT
    family = MONOMIAL
[../]

[/em_lin]
    order = CONSTANT
    family = MONOMIAL
[../]

[/Ar+_lin]
    order = CONSTANT
    family = MONOMIAL

```

```

[../]
[./Ar*_lin]
  order = CONSTANT
  family = MONOMIAL
[../]

[./Ar]
[../]

[./Efield]
  order = CONSTANT
  family = MONOMIAL
[../]

[./Current_em]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./Current_Ar]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./emRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./exRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./swRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./deexRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./quRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./poolRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./TwoBRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./ThreeBRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[ ]

[AuxKernels]
[./emDeBug]
  type = DebugResidualAux
  variable = emDeBug
  debug.variable = em
  #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]
[./Ar+_DeBug]
  type = DebugResidualAux
  variable = Ar+_DeBug

```

```

    debug_variable = Ar+
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]
[./mean_enDeBug]
    type = DebugResidualAux
    variable = mean_enDeBug
    debug_variable = mean_en
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]
[./Ar*_DeBug]
    type = DebugResidualAux
    variable = Ar*_DeBug
    debug_variable = Ar*
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]

[./emRate]
    type = ProcRateForRateCoeff
    variable = emRate
    v = em
    w = Ar
    reaction = 'em + Ar -> em + em + Ar+'
[../]
[./exRate]
    type = ProcRateForRateCoeff
    variable = exRate
    v = em
    w = Ar*
    reaction = 'em + Ar -> em + Ar*'
[../]
[./swRate]
    type = ProcRateForRateCoeff
    variable = swRate
    v = em
    w = Ar*
    reaction = 'em + Ar* -> em + em + Ar+'
[../]
[./deexRate]
    type = ProcRateForRateCoeff
    variable = deexRate
    v = em
    w = Ar*
    reaction = 'em + Ar* -> em + Ar'
[../]
[./quRate]
    type = ProcRateForRateCoeff
    variable = quRate
    v = em
    w = Ar*
    reaction = 'em + Ar* -> em + Ar_r'
[../]
[./poolRate]
    type = ProcRateForRateCoeff
    variable = poolRate
    v = Ar*
    w = Ar*
    reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
[../]
[./TwoBRate]
    type = ProcRateForRateCoeff
    variable = TwoBRate
    v = Ar*
    w = Ar
    reaction = 'Ar* + Ar -> Ar + Ar'
[../]
[./ThreeBRate]
    type = ProcRateForRateCoeffThreeBody
    variable = ThreeBRate
    v = Ar*
    w = Ar
    vv = Ar
    reaction = 'Ar* + Ar + Ar -> Ar-2 + Ar'
[../]
[./Te]
    type = ElectronTemperature
    variable = Te
    electron_density = em

```

```

    mean_en = mean_en
[../]
[./x-g]
    type = Position
    variable = x
    position_units = ${dom0Scale}
[../]

[./x-ng]
    type = Position
    variable = x_node
    position_units = ${dom0Scale}
[../]

[./em_lin]
    type = DensityMoles
    convert_moles = true
    variable = em_lin
    density_log = em
[../]
[./Ar+_lin]
    type = DensityMoles
    convert_moles = true
    variable = Ar+_lin
    density_log = Ar+
[../]
[./Ar*_lin]
    type = DensityMoles
    convert_moles = true
    variable = Ar*_lin
    density_log = Ar*
[../]

[./Ar_val]
    type = ConstantAux
    variable = Ar
    # value = 3.22e22
    value = -2.928623
    execute_on = INITIAL
[../]

[./Efield_calc]
    type = Efield
    component = 0
    potential = potential
    variable = Efield
    position_units = ${dom0Scale}
[../]
[./Current_em]
    type = Current
    potential = potential
    density_log = em
    variable = Current_em
    art_diff = false
    block = 0
    position_units = ${dom0Scale}
[../]
[./Current_Ar]
    type = Current
    potential = potential
    density_log = Ar+
    variable = Current_Ar
    art_diff = false
    block = 0
    position_units = ${dom0Scale}
[../]
[]

[BCs]
#Voltage Boundary Condition, same as in paper
[./potential_left]
    type = FunctionDirichletBC
    variable = potential
    boundary = 'left'
    function = potential_bc_func
[../]

```

```

[./ potential_dirichlet_right]
  type = DirichletBC
  variable = potential
  boundary = 'right'
  value = 0
[../]

#New Boundary conditions for electrons, same as in paper
[./ em_physical_right]
  type = LymberopoulosElectronBC
  variable = em
  boundary = 'right'
  gamma = 0.01
  #gamma = 1
  ks = 1.19e5
  #ks = 0.0
  ion = Ar+
  potential = potential
  position_units = ${dom0Scale}
[../]
[./ em_physical_left]
  type = LymberopoulosElectronBC
  variable = em
  boundary = 'left'
  gamma = 0.01
  #gamma = 1
  ks = 1.19e5
  #ks = 0.0
  ion = Ar+
  potential = potential
  position_units = ${dom0Scale}
[../]

#New Boundary conditions for ions, should be the same as in paper
[./ Ar+_physical_right_advection]
  type = LymberopoulosIonBC
  variable = Ar+
  potential = potential
  boundary = 'right'
  position_units = ${dom0Scale}
[../]
[./ Ar+_physical_left_advection]
  type = LymberopoulosIonBC
  variable = Ar+
  potential = potential
  boundary = 'left'
  position_units = ${dom0Scale}
[../]

#New Boundary conditions for ions, should be the same as in paper
#(except the metastables are not set to zero, since Zapdos uses log form)
[./ Ar*_physical_right_diffusion]
  type = LogDensityDirichletBC
  variable = Ar*
  boundary = 'right'
  value = 100
[../]
[./ Ar*_physical_left_diffusion]
  type = LogDensityDirichletBC
  variable = Ar*
  boundary = 'left'
  value = 100
[../]

#New Boundary conditions for mean energy, should be the same as in paper
[./ mean_en_physical_right]
  type = ElectronTemperatureDirichletBC
  variable = mean_en
  em = em
  value = 0.5
  boundary = 'right'
[../]
[./ mean_en_physical_left]
  type = ElectronTemperatureDirichletBC
  variable = mean_en
  em = em
  value = 0.5

```

```

    boundary = 'left'
[../]

[]

[ICs]
[./em_ic]
    type = FunctionIC
    variable = em
    function = density_ic_func
[../]
[./Ar+_ic]
    type = FunctionIC
    variable = Ar+
    function = density_ic_func
[../]
[./Ar*_ic]
    type = FunctionIC
    variable = Ar*
    function = density_ic_func
[../]
[./mean_en_ic]
    type = FunctionIC
    variable = mean-en
    function = energy_density_ic_func
[../]

[./potential_ic]
    type = FunctionIC
    variable = potential
    function = potential_ic_func
[../]

[]

[Functions]
[./potential_bc_func]
    type = ParsedFunction
    value = '0.100*sin(2*3.1415926*13.56e6*t)'
[../]
[./potential_ic_func]
    type = ParsedFunction
    value = '0.100 * (25.4e-3 - x)'
[../]
[./density_ic_func]
    type = ParsedFunction
    value = 'log((1e13 + 1e15 * (1-x/1)^2 * (x/1)^2)/6.022e23)'
[../]
[./energy_density_ic_func]
    type = ParsedFunction
    value = 'log(3./2.) + log((1e13 + 1e15 * (1-x/1)^2 * (x/1)^2)/6.022e23)'
[../]

[]

[Materials]
[./GasBasics]
    type = GasElectronMoments
    interp_trans_coeffs = false
    interp_elastic_coeff = false
    ramp_trans_coeffs = false
    user_p_gas = 133.322
    em = em
    potential = potential
    mean-en = mean-en
    user_electron_mobility = 30.0
    user_electron_diffusion_coeff = 119.8757763975
    property_tables_file = Argon_reactions_paper_RateCoefficients/electron_moments.txt
    position_units = ${dom0Scale}
[../]
[./gas_species_0]
    type = HeavySpeciesMaterial
    heavy_species_name = Ar+
    heavy_species_mass = 6.64e-26
    heavy_species_charge = 1.0
    mobility = 0.144409938
    diffusivity = 6.428571e-3
[../]

```

```

[./ gas-species_1]
type = HeavySpeciesMaterial
heavy-species_name = Ar*
heavy-species_mass = 6.64e-26
heavy-species_charge = 0.0
diffusivity = 7.515528e-3
[../]
[./ gas-species_2]
type = HeavySpeciesMaterial
heavy-species_name = Ar
heavy-species_mass = 6.64e-26
heavy-species_charge = 0.0
[../]
[./ reaction_0]
type = ZapdosEEDFRateConstant
mean_en = mean_en
sampling_format = electron_energy
property_file = 'Argon_reactions_paper.RateCoefficients/reaction_em + Ar -> em + Ar*.txt'
reaction = 'em + Ar -> em + Ar*'
position_units = ${dom0Scale}
file_location = ''
em = em
[../]
[./ reaction_1]
type = ZapdosEEDFRateConstant
mean_en = mean_en
sampling_format = electron_energy
property_file = 'Argon_reactions_paper.RateCoefficients/reaction_em + Ar -> em + em + Ar+.txt'
reaction = 'em + Ar -> em + em + Ar+'
position_units = ${dom0Scale}
file_location = ''
em = em
[../]
[./ reaction_2]
type = ZapdosEEDFRateConstant
mean_en = mean_en
sampling_format = electron_energy
property_file = 'Argon_reactions_paper.RateCoefficients/reaction_em + Ar* -> em + Ar.txt'
reaction = 'em + Ar* -> em + Ar'
position_units = ${dom0Scale}
file_location = ''
em = em
[../]
[./ reaction_3]
type = ZapdosEEDFRateConstant
mean_en = mean_en
sampling_format = electron_energy
property_file = 'Argon_reactions_paper.RateCoefficients/reaction_em + Ar* -> em + em + Ar+.txt'
reaction = 'em + Ar* -> em + em + Ar+'
position_units = ${dom0Scale}
file_location = ''
em = em
[../]
[./ reaction_4]
type = GenericRateConstant
reaction = 'em + Ar* -> em + Ar.r'
#reaction_rate_value = 2e-13
reaction_rate_value = 1.2044e11
[../]
[./ reaction_5]
type = GenericRateConstant
reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
#reaction_rate_value = 6.2e-16
reaction_rate_value = 373364000
[../]
[./ reaction_6]
type = GenericRateConstant
reaction = 'Ar* + Ar -> Ar + Ar'
#reaction_rate_value = 3e-21
reaction_rate_value = 1806.6
[../]
[./ reaction_7]
type = GenericRateConstant
reaction = 'Ar* + Ar + Ar -> Ar.2 + Ar'
#reaction_rate_value = 1.1e-42
reaction_rate_value = 398909.324
[../]

```

```

[]

#New postprocessor that calculates the inverse of the plasma frequency
[Postprocessors]
  [./InversePlasmaFreq]
    type = PlasmaFrequencyInverse
    variable = em
    use_moles = true
    execute_on = 'INITIAL TIMESTEP_BEGIN'
  [../]
[]

[Preconditioning]
  active = 'smp'
  [./smp]
    type = SMP
    full = true
  [../]

  [./fdp]
    type = FDP
    full = true
  [../]
[]

[Executioner]
  type = Transient
  end_time = 0.00737463126
  #end_time = 3e-7
  petsc_options = '-snes_converged_reason -snes_linesearch_monitor '
  solve_type = NEWTON
  petsc_options_iname = '-pc_type -pc_factor_shift_type -pc_factor_shift_amount -ksp_type -snes_linesearch_minlam'
  petsc_options_value = 'lu NONZERO 1.e-10 fgmres 1e-3'
  nl_rel_tol = 1e-08
  #nl_abs_tol = 7.6e-5 #Commit out do to test failure on Mac
  dtmin = 1e-14
  l_max_its = 20

  #Time steps based on the inverse of the plasma frequency
  [./TimeStepper]
    type = PostprocessorDT
    postprocessor = InversePlasmaFreq
  [../]
[]

[Outputs]
  print_perf_log = true
  [./out]
    type = Exodus
  [../]
[]

```

## A2: Input Code:

### Lymberopoulos\_with\_argon\_metastables\_2D\_At100mTorr.i

dom0Scale=25.4e-3

```

[GlobalParams]
  potential_units = kV
  use_moles = true
[]

[Mesh]
  type = FileMesh
  file = 'Lymberopoulos.msh'
[]

[MeshModifiers]
  [./left]
    type = SideSetsFromNormals
    normals = '-1 0 0'
    new_boundary = 'left '

```



```

[../]
[/right]
  type = SideSetsFromNormals
  normals = '1 0 0'
  new_boundary = 'right'
[../]
[]

[Problem]
  type = FEProblem
[]

[Variables]
[./em]
[../]

[./Ar+]
[../]

[./Ar*]
[../]

[./mean_en]
[../]

[./potential]
[../]
[]

[Kernels]
#Electron Equations (Same as in paper)
#Time Derivative term of electron
[./em.time.deriv]
  type = ElectronTimeDerivative
  variable = em
[../]
#Advection term of electron
[./em.advection]
  type = EFieldAdvectionElectrons
  variable = em
  potential = potential
  mean_en = mean_en
  position_units = ${dom0Scale}
[../]
#Diffusion term of electrons
[./em.diffusion]
  type = CoeffDiffusionElectrons
  variable = em
  mean_en = mean_en
  position_units = ${dom0Scale}
[../]
#Net electron production from ionization
[./em.ionization]
  type = ElectronReactantSecondOrderLog
  variable = em
  v = Ar
  energy = mean_en
  reaction = 'em + Ar -> em + em + Ar+'
  coefficient = 1
[../]
#Net electron production from step-wise ionization
[./em.stepwise.ionization]
  type = ElectronReactantSecondOrderLog
  variable = em
  v = Ar*
  energy = mean_en
  reaction = 'em + Ar* -> em + em + Ar+'
  coefficient = 1
[../]
#Net electron production from metastable pooling
[./em.pooling]
  type = ProductSecondOrderLog
  variable = em
  v = Ar*
  w = Ar*
  reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
  coefficient = 1

```

```

[../]
#Argon Ion Equations (Same as in paper)
#Time Derivative term of the ions
[./Ar+_time_deriv]
  type = ElectronTimeDerivative
  variable = Ar+
[../]
#Advection term of ions
[./Ar+_advection]
  type = EFieldAdvection
  variable = Ar+
  potential = potential
  position_units = ${dom0Scale}
[../]
[./Ar+_diffusion]
  type = CoeffDiffusion
  variable = Ar+
  position_units = ${dom0Scale}
[../]
#Net ion production from ionization
[./Ar+_ionization]
  type = ElectronProductSecondOrderLog
  variable = Ar+
  electron = em
  target = Ar
  energy = mean_en
  reaction = 'em + Ar -> em + em + Ar+'
  coefficient = 1
[../]
#Net ion production from step-wise ionization
[./Ar+_stepwise_ionization]
  type = ElectronProductSecondOrderLog
  variable = Ar+
  electron = em
  target = Ar*
  energy = mean_en
  reaction = 'em + Ar* -> em + em + Ar+'
  coefficient = 1
[../]
#Net ion production from metastable pooling
[./Ar+_pooling]
  type = ProductSecondOrderLog
  variable = Ar+
  v = Ar*
  w = Ar*
  reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
  coefficient = 1
[../]

#Argon Excited Equations (Same as in paper)
#Time Derivative term of excited Argon
[./Ar*_time_deriv]
  type = ElectronTimeDerivative
  variable = Ar*
[../]
#Diffusion term of excited Argon
[./Ar*_diffusion]
  type = CoeffDiffusion
  variable = Ar*
  position_units = ${dom0Scale}
[../]
#Net excited Argon production from excitation
[./Ar*_excitation]
  type = ElectronProductSecondOrderLog
  variable = Ar*
  electron = em
  target = Ar
  energy = mean_en
  reaction = 'em + Ar -> em + Ar*'
  coefficient = 1
[../]
#Net excited Argon loss from step-wise ionization
[./Ar*_stepwise_ionization]
  type = ElectronProductSecondOrderLog
  variable = Ar*
  electron = em

```

```

    target = Ar*
    energy = mean_en
    reaction = 'em + Ar* -> em + em + Ar+'
    coefficient = -1
    _target_eq_u = true
[../]
#Net excited Argon loss from superelastic collisions
[/Ar*_collisions]
    type = ElectronProductSecondOrderLog
    variable = Ar*
    electron = em
    target = Ar*
    energy = mean_en
    reaction = 'em + Ar* -> em + Ar'
    coefficient = -1
    _target_eq_u = true
[../]
#Net excited Argon loss from quenching to resonant
[/Ar*_quenching]
    type = ElectronProductSecondOrderLog
    variable = Ar*
    electron = em
    target = Ar*
    energy = mean_en
    reaction = 'em + Ar* -> em + Ar_r'
    coefficient = -1
    _target_eq_u = true
[../]
#Net excited Argon loss from metastable pooling
[/Ar*_pooling]
    type = ReactantSecondOrderLog
    variable = Ar*
    v = Ar*
    reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
    coefficient = -2
    _v_eq_u = true
[../]
#Net excited Argon loss from two-body quenching
[/Ar*_2B_quenching]
    type = ReactantSecondOrderLog
    variable = Ar*
    v = Ar
    reaction = 'Ar* + Ar -> Ar + Ar'
    coefficient = -1
[../]
#Net excited Argon loss from three-body quenching
[/Ar*_3B_quenching]
    type = ReactantThirdOrderLog
    variable = Ar*
    v = Ar
    w = Ar
    reaction = 'Ar* + Ar + Ar -> Ar_2 + Ar'
    coefficient = -1
[../]

#Voltage Equations (Same as in paper)
#Voltage term in Poissons Equation
[/potential_diffusion_dom0]
    type = CoeffDiffusionLin
    variable = potential
    position_units = ${dom0Scale}
[../]
#Ion term in Poissons Equation
[/Ar+_charge_source]
    type = ChargeSourceMoles_KV
    variable = potential
    charged = Ar+
[../]
#Electron term in Poissons Equation
[/em.charge_source]
    type = ChargeSourceMoles_KV
    variable = potential
    charged = em
[../]

#Since the paper uses electron temperature as a variable, the energy equation is in

```

```

#a different form but should be the same physics
#Time Derivative term of electron energy
[/mean_en_time_deriv]
type = ElectronTimeDerivative
variable = mean_en
[../]
#Advection term of electron energy
[/mean_en_advection]
type = EFieldAdvectionEnergy
variable = mean_en
potential = potential
em = em
position_units = ${dom0Scale}
[../]
#Diffusion term of electrons energy
[/mean_en_diffusion]
type = CoeffDiffusionEnergy
variable = mean_en
em = em
position_units = ${dom0Scale}
[../]
#Joule Heating term
[/mean_en_joule_heating]
type = JouleHeating
variable = mean_en
potential = potential
em = em
position_units = ${dom0Scale}
[../]
#Energy loss from ionization
[/Ionization_Loss]
type = ElectronEnergyTermRate
variable = mean_en
em = em
v = Ar
reaction = 'em + Ar -> em + em + Ar+'
threshold_energy = -15.7
position_units = ${dom0Scale}
[../]
#Energy loss from excitation
[/Excitation_Loss]
type = ElectronEnergyTermRate
variable = mean_en
em = em
v = Ar
reaction = 'em + Ar -> em + Ar*'
threshold_energy = -11.56
position_units = ${dom0Scale}
[../]
#Energy loss from step-wise ionization
[/Stepwise_Ionization_Loss]
type = ElectronEnergyTermRate
variable = mean_en
em = em
v = Ar*
reaction = 'em + Ar* -> em + em + Ar+'
threshold_energy = -4.14
position_units = ${dom0Scale}
[../]
#Energy gain from superelastic collisions
[/Collisions_Loss]
type = ElectronEnergyTermRate
variable = mean_en
em = em
v = Ar*
reaction = 'em + Ar* -> em + Ar'
threshold_energy = 11.56
position_units = ${dom0Scale}
[../]
[]

[AuxVariables]
[/emDeBug]
[../]
[/Ar+-DeBug]
[../]

```

```

[./Ar*_DeBug]
[../]
[./mean_enDeBug]
[../]

[./Te]
  order = CONSTANT
  family = MONOMIAL
[../]

[./x]
  order = CONSTANT
  family = MONOMIAL
[../]

[./x_node]
[../]

[./rho]
  order = CONSTANT
  family = MONOMIAL
[../]

[./em_lin]
  order = CONSTANT
  family = MONOMIAL
[../]

[./Ar+_lin]
  order = CONSTANT
  family = MONOMIAL
[../]

[./Ar*_lin]
  order = CONSTANT
  family = MONOMIAL
[../]

[./Ar]
[../]

[./Efield]
  order = CONSTANT
  family = MONOMIAL
[../]

[./Current_em]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./Current_Ar]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./emRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./exRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./swRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./deexRate]
  order = CONSTANT
  family = MONOMIAL
  block = 0
[../]
[./quRate]

```

```

    order = CONSTANT
    family = MONOMIAL
    block = 0
[../]
[/poolRate]
    order = CONSTANT
    family = MONOMIAL
    block = 0
[../]
[/TwoBRate]
    order = CONSTANT
    family = MONOMIAL
    block = 0
[../]
[/ThreeBRate]
    order = CONSTANT
    family = MONOMIAL
    block = 0
[../]
[]

[AuxKernels]
[/emDeBug]
    type = DebugResidualAux
    variable = emDeBug
    debug.variable = em
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]
[/Ar+_DeBug]
    type = DebugResidualAux
    variable = Ar+_DeBug
    debug.variable = Ar+
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]
[/mean_enDeBug]
    type = DebugResidualAux
    variable = mean_enDeBug
    debug.variable = mean_en
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]
[/Ar*_DeBug]
    type = DebugResidualAux
    variable = Ar*_DeBug
    debug.variable = Ar*
    #execute_on = 'LINEAR NONLINEAR TIMESTEP_BEGIN'
[../]

[/emRate]
    type = ProcRateForRateCoeff
    variable = emRate
    v = em
    w = Ar
    reaction = 'em + Ar -> em + em + Ar+'
[../]
[/exRate]
    type = ProcRateForRateCoeff
    variable = exRate
    v = em
    w = Ar*
    reaction = 'em + Ar -> em + Ar*'
[../]
[/swRate]
    type = ProcRateForRateCoeff
    variable = swRate
    v = em
    w = Ar*
    reaction = 'em + Ar* -> em + em + Ar+'
[../]
[/deexRate]
    type = ProcRateForRateCoeff
    variable = deexRate
    v = em
    w = Ar*
    reaction = 'em + Ar* -> em + Ar'
[../]
[/quRate]
    type = ProcRateForRateCoeff

```

```

variable = quRate
v = em
w = Ar*
reaction = 'em + Ar* -> em + Ar_r'
[../]
[./poolRate]
type = ProcRateForRateCoeff
variable = poolRate
v = Ar*
w = Ar*
reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
[../]
[./TwoBRate]
type = ProcRateForRateCoeff
variable = TwoBRate
v = Ar*
w = Ar
reaction = 'Ar* + Ar -> Ar + Ar'
[../]
[./ThreeBRate]
type = ProcRateForRateCoeffThreeBody
variable = ThreeBRate
v = Ar*
w = Ar
vv = Ar
reaction = 'Ar* + Ar + Ar -> Ar_2 + Ar'
[../]
[./Te]
type = ElectronTemperature
variable = Te
electron_density = em
mean_en = mean_en
[../]
[./x-g]
type = Position
variable = x
position.units = ${dom0Scale}
[../]
[./x-ng]
type = Position
variable = x_node
position.units = ${dom0Scale}
[../]
[./em_lin]
type = DensityMoles
convert_moles = true
variable = em_lin
density_log = em
[../]
[./Ar+_lin]
type = DensityMoles
convert_moles = true
variable = Ar+_lin
density_log = Ar+
[../]
[./Ar*_lin]
type = DensityMoles
convert_moles = true
variable = Ar*_lin
density_log = Ar*
[../]
[./Ar_val]
type = ConstantAux
variable = Ar
# value = 3.22e22
value = -2.928623
execute_on = INITIAL
[../]
[./Efield_calc]
type = Efield
component = 0
potential = potential
variable = Efield

```

```

    position.units = ${dom0Scale}
[../]
[/Current_em]
    type = Current
    potential = potential
    density_log = em
    variable = Current_em
    art_diff = false
    block = 0
    position.units = ${dom0Scale}
[../]
[/Current_Ar]
    type = Current
    potential = potential
    density_log = Ar+
    variable = Current_Ar
    art_diff = false
    block = 0
    position.units = ${dom0Scale}
[../]
[]

[BCs]
#Voltage Boundary Condition, same as in paper
[/potential_left]
    type = FunctionDirichletBC
    variable = potential
    boundary = 'left'
    function = potential_bc_func
[../]
[/potential_dirichlet_right]
    type = DirichletBC
    variable = potential
    boundary = 'right'
    value = 0
[../]

#New Boundary conditions for electrons, same as in paper
[/em_physical_right]
    type = LymberopoulosElectronBC
    variable = em
    boundary = 'right'
    gamma = 0.01
    #gamma = 1
    ks = 1.19e5
    #ks = 0.0
    ion = Ar+
    potential = potential
    position.units = ${dom0Scale}
[../]
[/em_physical_left]
    type = LymberopoulosElectronBC
    variable = em
    boundary = 'left'
    gamma = 0.01
    #gamma = 1
    ks = 1.19e5
    #ks = 0.0
    ion = Ar+
    potential = potential
    position.units = ${dom0Scale}
[../]

#New Boundary conditions for ions, should be the same as in paper
[/Ar+_physical_right_advection]
    type = LymberopoulosIonBC
    variable = Ar+
    potential = potential
    boundary = 'right'
    position.units = ${dom0Scale}
[../]
[/Ar+_physical_left_advection]
    type = LymberopoulosIonBC
    variable = Ar+
    potential = potential
    boundary = 'left'

```



```

    position.units = ${dom0Scale}
[../]

#New Boundary conditions for ions, should be the same as in paper
#(except the metastables are not set to zero, since Zapdos uses log form)
[./Ar*_physical_right_diffusion]
    type = LogDensityDirichletBC
    variable = Ar*
    boundary = 'right'
    value = 100
[../]
[./Ar*_physical_left_diffusion]
    type = LogDensityDirichletBC
    variable = Ar*
    boundary = 'left'
    value = 100
[../]

#New Boundary conditions for mean energy, should be the same as in paper
[./mean_en_physical_right]
    type = ElectronTemperatureDirichletBC
    variable = mean_en
    em = em
    value = 0.5
    boundary = 'right'
[../]
[./mean_en_physical_left]
    type = ElectronTemperatureDirichletBC
    variable = mean_en
    em = em
    value = 0.5
    boundary = 'left'
[../]

[]

[ICs]
[./em_ic]
    type = FunctionIC
    variable = em
    function = density_ic_func
[../]
[./Ar+_ic]
    type = FunctionIC
    variable = Ar+
    function = density_ic_func
[../]
[./Ar*_ic]
    type = FunctionIC
    variable = Ar*
    function = density_ic_func
[../]
[./mean_en_ic]
    type = FunctionIC
    variable = mean_en
    function = energy_density_ic_func
[../]

[./potential_ic]
    type = FunctionIC
    variable = potential
    function = potential_ic_func
[../]

[]

[Functions]
[./potential_bc_func]
    type = ParsedFunction
    value = '0.100*sin(2*3.1415926*13.56e6*t)'
[../]
[./potential_ic_func]
    type = ParsedFunction
    value = '0.100 * (25.4e-3 - x)'
[../]
[./density_ic_func]
    type = ParsedFunction

```

```

    value = 'log((1e13 + 1e15 * (1-x/1)^2 * (x/1)^2)/6.022e23)'
[../]
[./energy_density_ic_func]
    type = ParsedFunction
    value = 'log(3./2.) + log((1e13 + 1e15 * (1-x/1)^2 * (x/1)^2)/6.022e23)'
[../]
[]

[Materials]
[./GasBasics]
    type = GasElectronMoments
    interp_trans_coeffs = false
    interp_elastic_coeff = false
    ramp_trans_coeffs = false
    user_p_gas = 133.322
    em = em
    potential = potential
    mean_en = mean_en
    user_electron_mobility = 30.0
    user_electron_diffusion_coeff = 119.8757763975
    property_tables_file = Argon_reactions_paper_RateCoefficients/electron_moments.txt
    position_units = ${dom0Scale}
[../]
[./gas_species_0]
    type = HeavySpeciesMaterial
    heavy_species_name = Ar+
    heavy_species_mass = 6.64e-26
    heavy_species_charge = 1.0
    mobility = 0.144409938
    diffusivity = 6.428571e-3
[../]
[./gas_species_1]
    type = HeavySpeciesMaterial
    heavy_species_name = Ar*
    heavy_species_mass = 6.64e-26
    heavy_species_charge = 0.0
    diffusivity = 7.515528e-3
[../]
[./gas_species_2]
    type = HeavySpeciesMaterial
    heavy_species_name = Ar
    heavy_species_mass = 6.64e-26
    heavy_species_charge = 0.0
[../]
[./reaction_0]
    type = ZapdosEEDFRateConstant
    mean_en = mean_en
    sampling_format = electron_energy
    property_file = 'Argon_reactions_paper_RateCoefficients/reaction_em + Ar -> em + Ar*.txt'
    reaction = 'em + Ar -> em + Ar*'
    position_units = ${dom0Scale}
    file_location = ''
    em = em
[../]
[./reaction_1]
    type = ZapdosEEDFRateConstant
    mean_en = mean_en
    sampling_format = electron_energy
    property_file = 'Argon_reactions_paper_RateCoefficients/reaction_em + Ar -> em + em + Ar+.txt'
    reaction = 'em + Ar -> em + em + Ar+'
    position_units = ${dom0Scale}
    file_location = ''
    em = em
[../]
[./reaction_2]
    type = ZapdosEEDFRateConstant
    mean_en = mean_en
    sampling_format = electron_energy
    property_file = 'Argon_reactions_paper_RateCoefficients/reaction_em + Ar* -> em + Ar.txt'
    reaction = 'em + Ar* -> em + Ar'
    position_units = ${dom0Scale}
    file_location = ''
    em = em
[../]
[./reaction_3]
    type = ZapdosEEDFRateConstant
    mean_en = mean_en

```

```

sampling.format = electron.energy
property_file = 'Argon_reactions_paper.RateCoefficients/reaction.em + Ar* -> em + em + Ar+.txt'
reaction = 'em + Ar* -> em + em + Ar+'
position.units = ${dom0Scale}
file_location = ''
em = em
[../]
[./reaction_4]
type = GenericRateConstant
reaction = 'em + Ar* -> em + Ar_r'
#reaction_rate_value = 2e-13
reaction_rate_value = 1.2044e11
[../]
[./reaction_5]
type = GenericRateConstant
reaction = 'Ar* + Ar* -> Ar+ + Ar + em'
#reaction_rate_value = 6.2e-16
reaction_rate_value = 373364000
[../]
[./reaction_6]
type = GenericRateConstant
reaction = 'Ar* + Ar -> Ar + Ar'
#reaction_rate_value = 3e-21
reaction_rate_value = 1806.6
[../]
[./reaction_7]
type = GenericRateConstant
reaction = 'Ar* + Ar + Ar -> Ar_2 + Ar'
#reaction_rate_value = 1.1e-42
reaction_rate_value = 398909.324
[../]
[]

#New postprocessor that calculates the inverse of the plasma frequency
[Postprocessors]
[./InversePlasmaFreq]
type = PlasmaFrequencyInverse
variable = em
use_moles = true
execute_on = 'INITIAL TIMESTEP_BEGIN'
[../]
[]

[Preconditioning]
active = 'smp'
[./smp]
type = SMP
full = true
[../]

[./fdp]
type = FDP
full = true
[../]
[]

[Executioner]
type = Transient
end_time = 0.00737463126
#end_time = 3e-7
petsc.options = '-snes_converged_reason -snes_linesearch_monitor'
solve_type = NEWTON
petsc.options.iname = '-pc_type -pc_factor_shift_type -pc_factor_shift_amount -ksp_type -snes_linesearch_minlam'
petsc.options.value = 'lu NONZERO 1.e-10 fgmres 1e-3'
nl_rel_tol = 1e-08
#nl_abs_tol = 7.6e-5 #Commit out do to test failure on Mac
dtmin = 1e-14
l_max_its = 20

#Time steps based on the inverse of the plasma frequency
[./TimeStepper]
type = PostprocessorDT
postprocessor = InversePlasmaFreq
[../]
[]

```

```
[Outputs]
  print_perf_log = true
  [./out]
    type = Exodus
  [../]
[]
```

# References

- [1] Lindsay, Alexander David. "Coupling of Plasmas and Liquids." Thesis (Ph.D.)—North Carolina State University, 2016
- [2] Lymberopoulos, Dimitris P. and Economou, Demetre J. "Modeling and simulation of glow discharge plasma reactors." *J. Vac. Sci. Technol. A* **12** (1994) 1229; <https://doi.org/10.1116/1.579300>
- [3] Lymberopoulos, Dimitris P. and Economou, Demetre J. "Fluid simulations of glow discharges: Effect of metastable atoms in argon." *J. Appl. Phys.* **73** (1993) 3668; <https://doi.org/10.1063/1.352926>
- [4] Lymberopoulos, Dimitris P. and Economou, Demetre J. "Fluid simulations of radio frequency glow discharges: Two-dimensional argon discharge including metastables" *Appl. Phys. Lett.* **63** (1993) 18
- [5] Lindsay, Alexander David and *et al.* "Fully coupled simulation of the plasma liquid interface and interfacial coefficient effects" *J. Phys. D: Appl. Phys.* **49** (2016) 235204
- [6] <http://gmsh.info/doc/texinfo/gmsh.html>
- [7] Hagelaar G J M, De Hoog F J and Kroesen G M W. "Boundary conditions in fluid models of gas discharges" *Phys. Rev. E* **62** (2000) 1452–4
- [8] Sakiyama Y and Graves D B. "Nonthermal atmospheric RF plasma in one-dimensional spherical coordinates: asymmetric sheath structure and the discharge mechanism" *J. Appl. Phys.* **101** (2007) 073306