IONATOR User Manual

Version 1.0

J. T. Yoskowitz

Last Updated August 22, 2022

Contents

1 Introduction

What a splendid fellow you are to realize you need IONATOR!

IONATOR stands for **ION**ization **A**nd **T**racking **O**f **R**esidual gas. It is a C++ custom element created for General Particle Tracer (GPT), a particle tracking code. The original purpose of IONATOR is to allow the user to simulate electron impact ionization of residual gas within an electron accelerator. But the code can be used in any GPT simulation involving electron impact ionization.

While other custom elements for GPT have been made for the simulation of electron impact ionization [1], the defining feature of IONATOR is its Monte Carlo routines for ion production and secondary electron energy selection. These routines make each simulation more realistic, as ionization is probabilistic in nature. IONATOR also has the ability to implement a user-defined 1-D gas density map, which can be useful when simulating electron impact ionization within a particle accelerator, as the gas density may vary along the beamline. Finally, IONATOR simulates ion production in “real-time", meaning that the user can visualize the time-dependent interaction of ions with other simulation particles (via space charge effects, e.g.), as opposed to plots of particle trajectories where it is often difficult to discern how particles interact with each other.

This user manual is broken up into several sections. Section 3 describes how to install and implement the custom element (provided GPT is already installed). Section 4 describes the theoretical equations used by IONATOR. Section 5 describes the code in detail. Section 6 describes several features of IONATOR. Section 7 shows several real-world example simulations that demonstrate how IONATOR is used. Finally, section 8 shows how to troubleshoot typical errors that may occur when using IONATOR.

NOTE: Before using IONATOR, it is highly recommended that you become familiar with using GPT and have successfully run several simulations on your own. Chapter 2 of the GPT User Manual contains several tutorials to help you learn the basics of GPT.

2 License and Citation Info

IONATOR, writeIonInfo, colorcoding, and this user manual are licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

If using IONATOR in your published work, please use the following citation:

J. T. Yoskowitz et al., “Simulating Electron Impact Ionization Using a General Particle Tracer (GPT) Custom Element", Proc. 12th In. Particle Accelerator Conf. (IPAC ’21), Campinas, SP, Brazil (Virtual Conference), May 2021.

3 Installing IONATOR

IONATOR contains two files: `ionator.c` and `ionator_infostructures_10.h`. Use the following instructions to install IONATOR for use in GPT depending on the operating system:

1. For Windows machines:

   1.1 Copy both files into the GPT `elems` folder which is usually located at `C:\Program Files\General Particle Tracer\elems\`. Note that you may need administrator permission to copy these files into this folder.

   1.2 In the GPT `kernel` folder, edit the file `elemlist` and add the following text on a new line at the end of the file: `ionator ionator`.

   1.3 Close GPT if open, and reopen GPT in administrator mode.

   1.4 Under the “Elements" menu, click “Rebuild interface code". If successful, “`ionator`" will appear in the list of GPT Elements on the left-hand side of the GUI window.

   1.5 Under the “Elements" menu, click “Compile GPT-Elems" and then “Compile GPT-Progs". If both run error-free, then the installation is complete.

2. For Linux machines:

   2.1 Copy both files into the GPT `elems/` folder, which is usually located at `~/gpt/elems/`.

   2.2 In the `elems` folder, edit the file `elemlist` and add the following text on a new line at the end of the file: `ionator ionator`.

   2.3 Run `make` twice. If the second `make` command runs error-free, then the installation is complete.

4 Ionization Theory

This section summarizes the theory of electron impact ionization and the theoretical equations used by the code. For a more detailed description, see Chapter 2 of Ref. [2]

4.1 Electron Impact Ionization

Electron impact ionization occurs when an electron interacts with an atom or molecule resulting in the ejection of an electron. The incident electron is called the primary electron and the atom or molecule is called the target gas particle. After ionization, the target gas particle becomes a positive ion and the incident electron scatters away and is called a scattered electron. The ejected electron is called the secondary electron. Whichever electron has *lesser* energy is called the secondary electron and the other is the scattered electron. Figure 1 shows a diagram of the electron impact ionization process.

Ionization can occur provided the primary electron overcomes the minimum energy, the ionization or binding energy, required to remove an electron from the target gas particle. Both the secondary and scattered electrons may continue to ionize provided they have sufficient kinetic energy to do so.

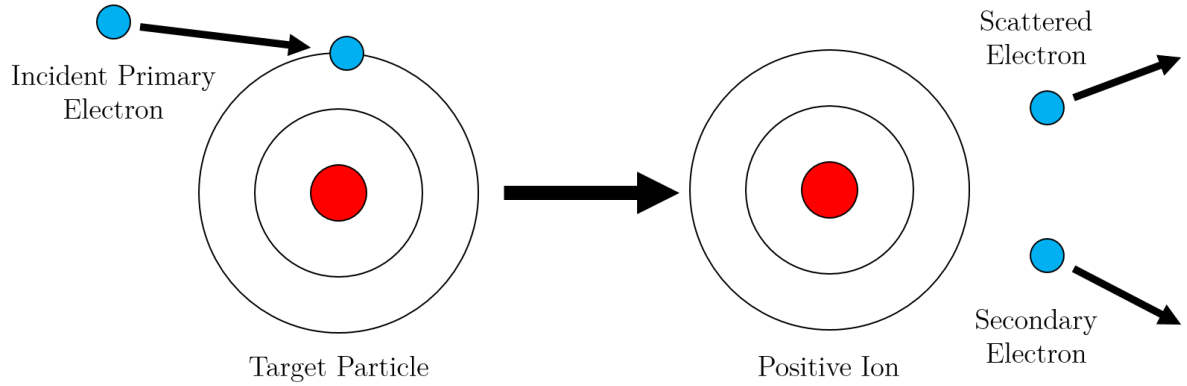

Figure 1: Diagram of electron impact ionization.

4.2 Ion Production Rate

The number of ions produced, N_{ion}, by a number of electrons, N_e, in a given timestep is calculated using the formula:

$$N_{ion} = \rho \sigma d N_e \tag{1}$$

where ρ is the gas density in ions/m^3, σ is the ionization cross section in m^2, and d is the distance travelled in meters by the electron in the timestep. The ionization cross section may be calculated using Reiser's formula [3], originally derived by Bethe [4]:

$$\sigma\left[\mathrm{m}^2\right] = \frac{1.872 \times 10^{-24} A_1}{\beta_e^2} f(T)\left[\ln\left(7.515 \times 10^4 A_2 \beta_e^2 \gamma^2\right) - \beta_e^2\right] \tag{2}$$
$$f(T) = \frac{B}{T}\left(\frac{T}{B} - 1\right)$$

where β_e and γ are the relativistic factors for the primary electron, A_1 and A_2 are derived from empirical constants given by Rieke and Prepejchal [5] that depend on the gas species, and $f(T)$ is a function for fitting σ at low energies, i.e., when the primary electron kinetic energy T is close to the ionization energy B of the target gas particle. Figure 2 shows a log-log plot of the ionization cross section for H_2, CO, and CH_4. Low energy electrons are orders of magnitude more likely to ionize than high energy electrons regardless of the target gas particle. The rise in ionization cross section at electron kinetic energies higher than 1 GeV is due to the γ^2 factor being high at these energies.

Dividing Eq. 1 by the timestep Δt and the distance travelled d gives the ion production rate per unit length:

$$\frac{N_{ion}}{\Delta t \cdot d} = \rho\sigma\frac{N_e}{\Delta t} \tag{3}$$

Note that Eq. 3 denotes the *average* ion production rate per unit length and assumes that the gas density is relatively uniform throughout the distance travelled d. In the limit of small timesteps, the instantaneous ion production rate per unit length is given by:

$$\frac{dN_{ion}}{dt}\left(\frac{1}{d}\right) = \rho\sigma\frac{I}{e} \tag{4}$$

where I is the electron current and e is the elementary charge. Figure 3 shows plots of the ion production rate per unit length for various gas species. The ion production rate curves are dominated by the ionization cross section. Table 1 shows values for the ionization cross section and ion production rate per unit length for H_2 gas for various primary electron energies, assuming a gas density of $\rho = 3.56 \times 10^{10}$ m^{-3} and a current of $I = 100$ µA ($\approx 6.24 \times 10^{14}$ electrons/s).

4.3 Secondary Electron Energy Distribution

The distribution of possible secondary electron energies W for a given primary electron energy T is given by the differential cross section derived in the Binary Encounter Dipole Model [6]:

Primary Electron Energy (keV)	$\sigma_{H_2^+}\left(\mathrm{m}^2\right)$	$\frac{dN_{\mathrm{H}_2^+}}{dt}\left(\frac{1}{d}\right)\left(\frac{\mathrm{H}_2^+}{\mathrm{s}\cdot\mathrm{m}}\right)$
0.1	1.1×10^{-20}	2.4×10^5
1	2.0×10^{-21}	4.4×10^4
10	2.9×10^{-22}	6.3×10^3
100	4.6×10^{-23}	1.0×10^3

Table 1: Calculated values of the ionization cross section and ion production rate per unit length for H_2 gas.

$$\begin{aligned}\frac{d\sigma(W,T)}{dW} &= \frac{S}{B(t+u+1)}\left[\frac{\left(\frac{N_i}{N}\right)-2}{t+1}\left(\frac{1}{w+1}+\frac{1}{t-w}\right)\right.\\ &\quad\left.+\left(2-\frac{N_i}{N}\right)\left(\frac{1}{(w+1)^2}+\frac{1}{(t-w)^2}\right)+\frac{\ln t}{N(w+1)}\frac{df(w)}{dw}\right] \qquad (5)\\ S &= 4\pi a_0^2 N R^2/B^2\\ t &= T/B\\ u &= U/B\\ w &= W/B\\ N_i &= \int_0^\infty \frac{df}{dw}dw\end{aligned}$$

where a_0 is the Bohr radius, R is the Rydberg energy, N is the number of electrons in the subshell of the gas molecule prior to ionization, U is the average kinetic energy of electrons in the subshell, and df/dw is the differential oscillator strength. Because the differential oscillator strengths for CO and CH_4 are not well-known, Kim and Rudd give an approximation for the differential cross section [6]:

$$\begin{aligned}\frac{d\sigma(W,T)}{dW} &= \frac{S}{t+u+1}\left[\frac{1}{(t-w)^2}+\frac{1}{(w+1)^2}-\frac{1}{t+1}\left(\frac{1}{t-w}+\frac{1}{w+1}\right)\right.\\ &\quad\left.+\ln t\left(\frac{1}{(t-w)^3}+\frac{1}{(w+1)^3}\right)\right] \qquad (6)\end{aligned}$$

Figure 4 shows differential cross-section curves of each supported gas species assuming a primary electron energy of 130 keV. The majority of secondary electrons produced are likely to have kinetic energies substantially smaller than primary electron kinetic energies.

4.4 Ion Energy Distribution

Because the primary electron mass is orders of magnitude less than any target gas particle, the kinetic energy of the ion depends largely on the energy of the target gas particle. Provided the target gas particle follows the ideal gas law, the distribution of ion energies can be modelled as a Maxwellian distribution, which is often defined in terms of speed v:

$$F(v)dv = \sqrt{\frac{2}{\pi}} \frac{v^2 \exp -\frac{v^2}{2a^2}}{a^3} dv \tag{7}$$
$$a = \left(\frac{kT}{m}\right)^{\frac{1}{2}}$$

where k is the Boltzmann constant, T is the gas temperature, and m is the mass of the ion. Figure 5 shows the Maxwellian distributions of target gas particle speeds at room temperature.

Once the energy of the primary electron, target gas particle, and secondary electron are known, the combined energy of the ion and scattered electron, E_{comb} can be calculated using energy conservation:

$$E_{comb} = E_{ion} + E_{scat} = T + E_{gas} - W - B \tag{8}$$

At the present moment, it is unknown what exactly the distribution of ion kinetic energies should be following electron-impact ionization. Several references suggest that the energy of the target gas particle is virtually unchanged provided the particle does not dissociate [4, 7, 8], but they do not provide any empirical evidence for this assertion. Until experiments that measure the energy distribution of ions and scattered electrons following ionization are performed and a theoretical distribution is derived, it is left to the user to determine what fraction of E_{comb} is given to the ion, with the remaining energy given to the scattered electron.

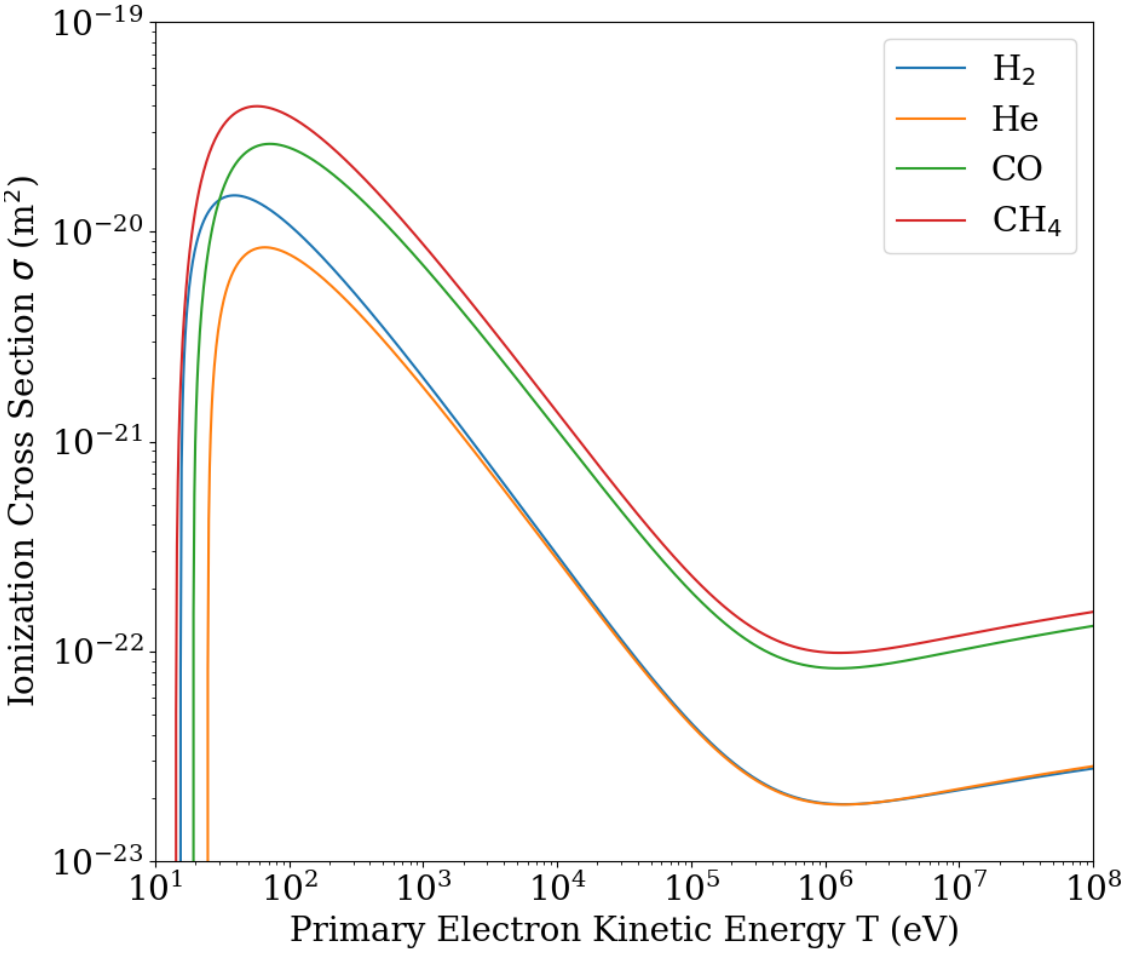

Figure 2: Log-log plot of the ionization cross section of H_2, CO, and CH_4.

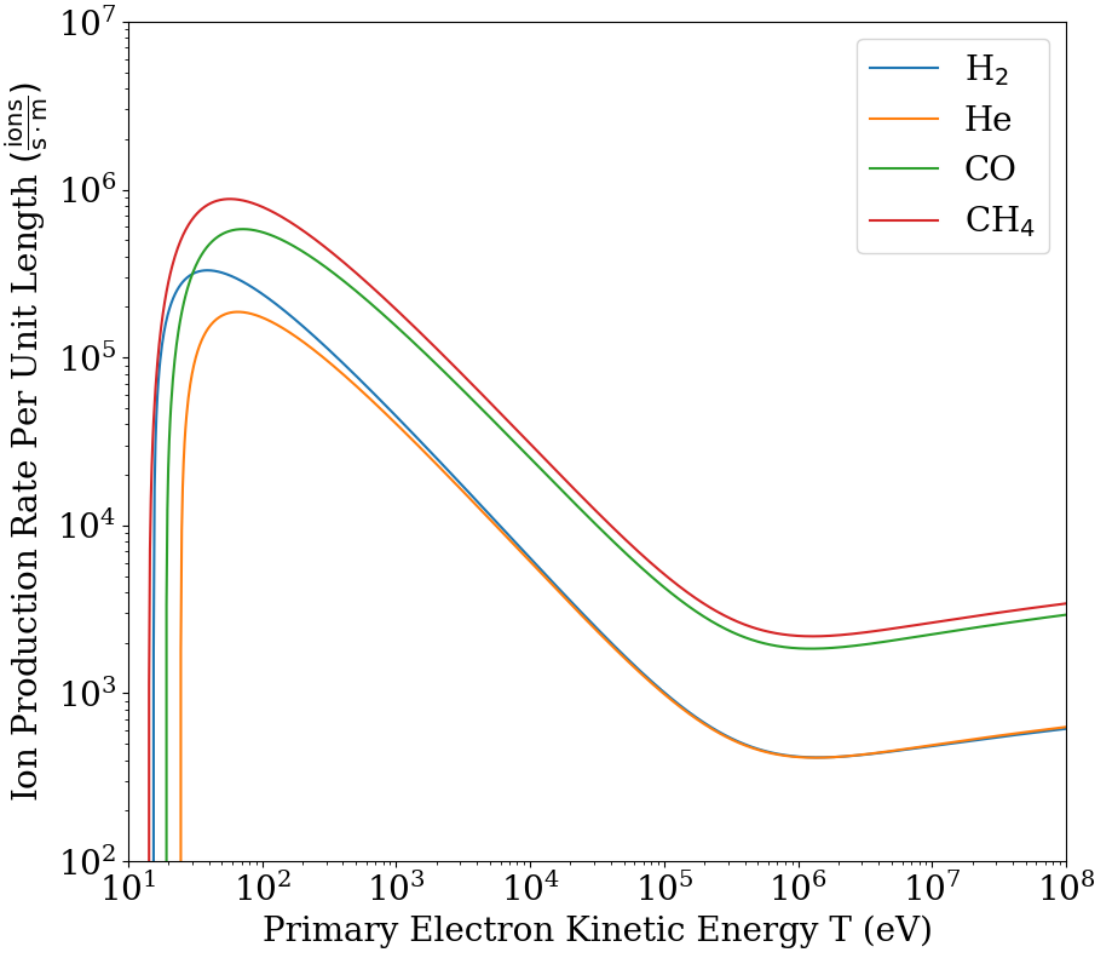

Figure 3: Log-log plot of the ion production rate per unit length of H_2, CO, and CH_4 gas.

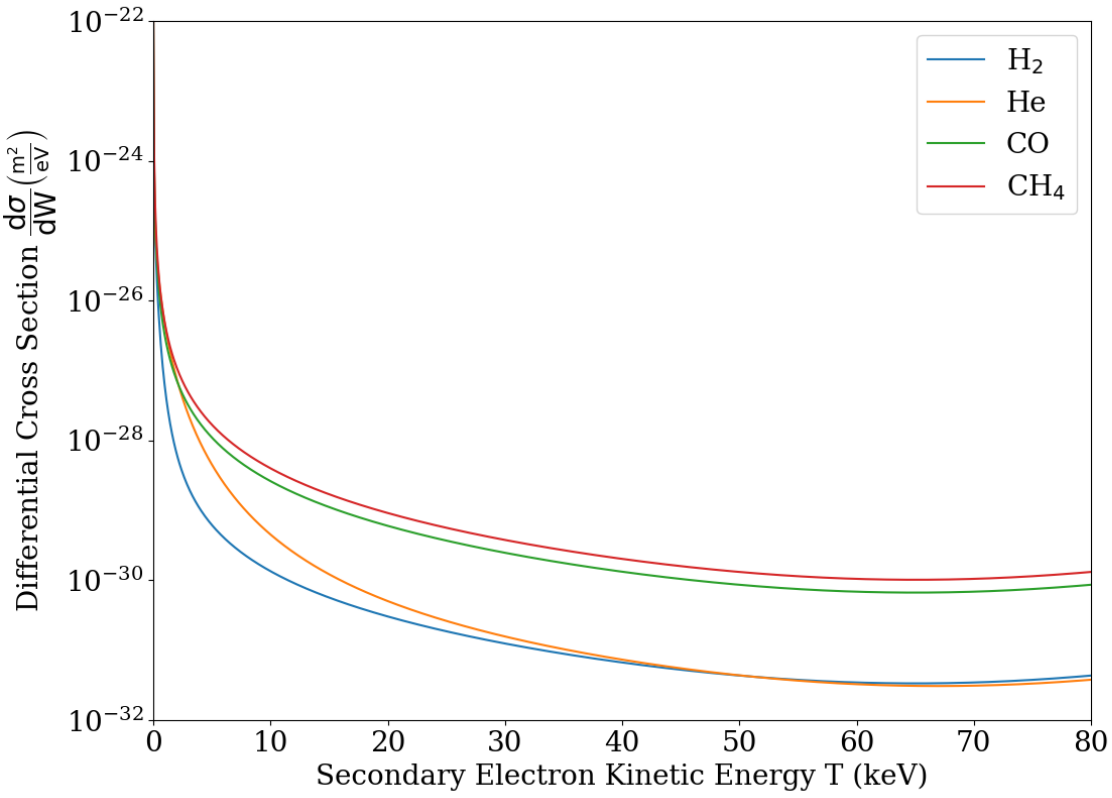

Figure 4: Log plots of the differential cross section for secondary electron energies of several target gas particles assuming a primary electron energy of 130 keV.

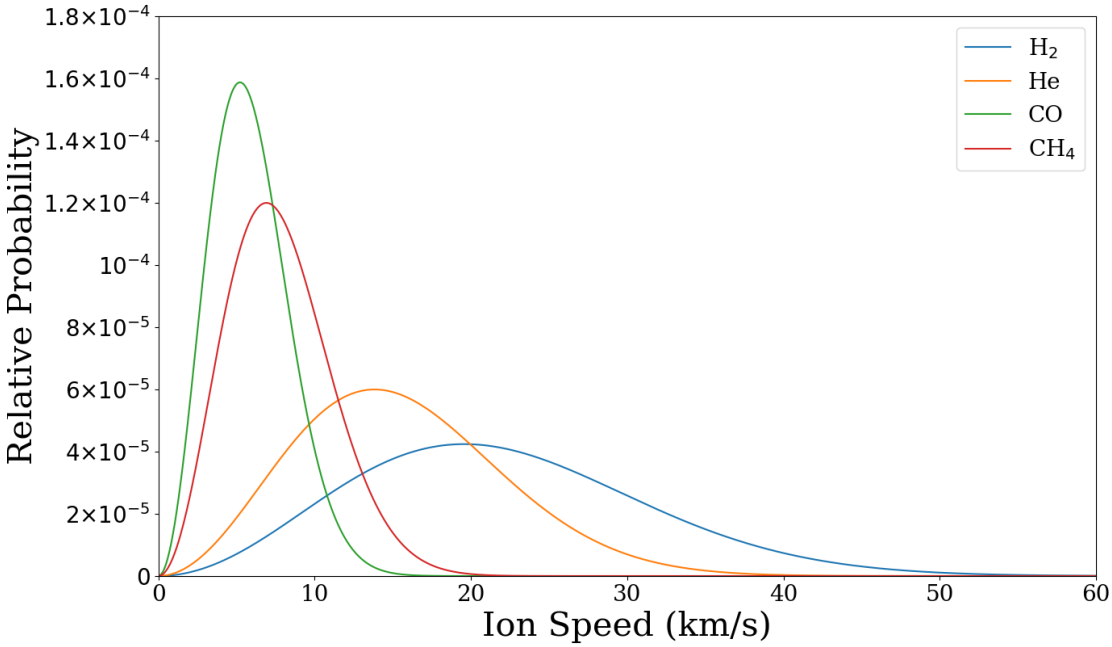

Figure 5: Maxwellian distribution for H_2, CO, and CH_4 gas.

5 IONATOR Code

This section describes the IONATOR code, which contains several algorithms to determine ionization eligibility and to calculate kinematic parameters for all particles involved in ionization. A flow chart of the main algorithm is shown in Figure 6.

When the IONATOR function is present in the input file, GPT will call the function at the end of every simulation timestep. It is possible for multiple IONATOR functions to be present in the input file (e.g. when simulating multiple gas species), in which case all instances of IONATOR will be called simultaneously at the end of every timestep. Using the input parameters given by the user, IONATOR will loop through all particles present in the simulation and determine if they are eligible to ionize. A Monte Carlo algorithm determines whether an eligible particle will ionize based on the ionization cross section. If a particle ionizes, a secondary electron is created and a separate Monte Carlo algorithm determines its kinetic energy. The remaining kinematic parameters for all particles are calculated and the secondary electron, scattered electron, and ion are added to the simulation.

The details of the main algorithm are described in the following sub sections.

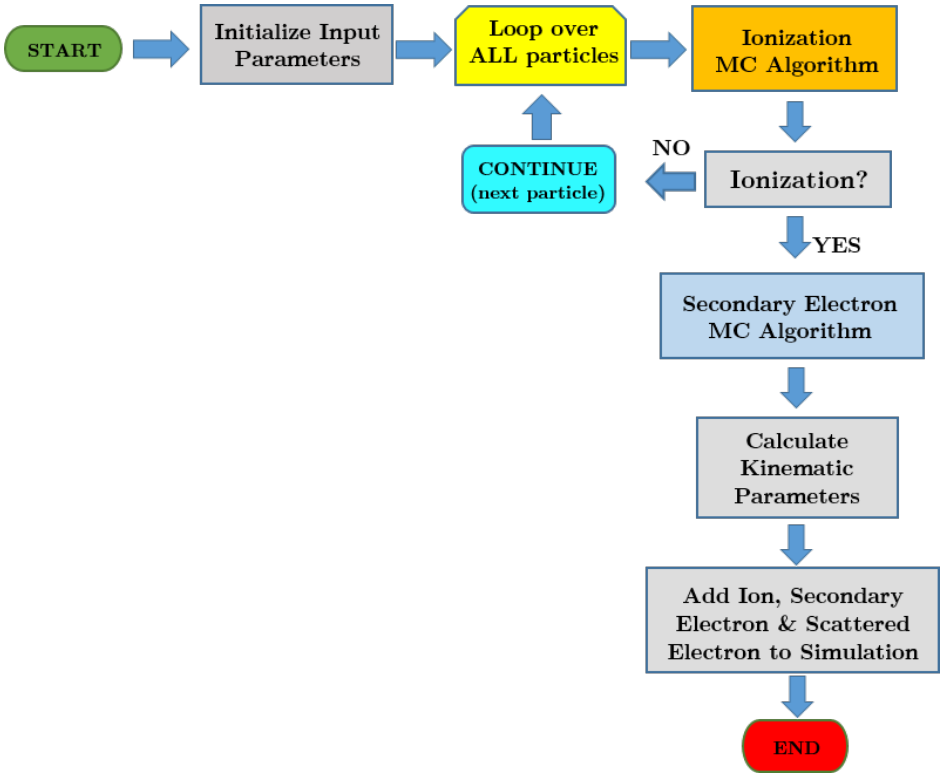

Figure 6: Flow chart of the main algorithm.

5.1 Input Parameters

IONATOR must be called within the input file with all input parameters specified either on the same line, or on individual lines as shown below in Listing 1. All parameters must be present and in the order shown.

```
ionator(
<ECS>,
<L>,
<"name of gas species">,
<"name of orbital" or "default">,
<"name of electron set">,
<"name of ion set">,
<0 or "name of secondary electron set">,
<0 or "name of scattered electron set">,
<gas density or "name of density file">,
<gas temperature T>
<ion fraction k>,
<integration precision (nsteps)>);
```

Listing 1: IONATOR function with input parameter definitions.

The required input parameters are as follows:

1. <ECS> : The element coordinate system (ECS) tells GPT the position of the *center* of the ionization region in the simulation. Please note that the <ECS> specification may constitute either 2, 3, or 10 arguments depending on how it is defined. See section 1.4 in the GPT User Manual for more info.

   Example: `"wcs","z",0.5` sets the center of the ionization region at $z = 0.5\,\mathrm{m}$ in the world coordinate system (WCS).

2. <L> : With the ECS specified, L denotes the total length of the ionization region.

   Example: If $z = 0.5\,\mathrm{m}$ is the center of the ionization region in the WCS, then an L specification of `0.5` would define the boundaries of the ionization region to be between $z = 0.25\,\mathrm{m}$ and $z = 0.75\,\mathrm{m}$ (with no boundary in the radial direction).

3. <"name of gas species"> and <"name of orbital"> : These two specifications denote the species of the target gas particle and the orbital from which the secondary electron is ejected. Both arguments must be specified as strings (i.e.

within quotes). Using “default" for the orbital specification automatically chooses the orbital with the lowest ionization energy (i.e. the most likely orbital for the secondary electron). See Table 3 for a list of all possible gas species and orbital specifications.

Example: `"CH4","default"` sets CH_4 as the target gas species with the default orbital ($1t_2$) for secondary electrons.

4. `<"name of electron set">` and `<"name of ion set">` : The user must specify a name for the *primary* electron and ion particle sets. Note that while primary electron set must be defined using the `setparticles()` keyword, the ion particle set does not. The primary electron particle set must be defined *before* the IONATOR function in the input file and the set names must match. Please do not alter any parameters of the ion particle set other than the set name – they will be defined automatically by IONATOR. See section 4.2.1.1 in the GPT user manual for info on using `setparticles()`.

   Example: `"electrons","ions"`.

5. `<"name of secondary electron set">`, `<"name of scattered electron set">` : The user *may* specify a name for the secondary and scattered electron particle sets. If a non-string (such as `0`) is specified for either set name, then that particle set is turned off, meaning that these particles will not be added to the simulation. However, IONATOR will continue to make ionization calculations *as if* they were present. Turning off secondary or scattered electrons may help to speed up a simulation and reduce memory usage, as there are less particles produced, or to help clean up a simulation when only primary electrons and ions are of importance. Please note that only primary, secondary, and scattered electrons that are present within the simulation are eligible to ionize. Turning off secondary or scattered electrons may decrease the number of ions produced in the simulations. However, since the ionization cross section for a given electron is very small, it is unlikely that turning off secondary or scattered electrons will have a significant impact on the number of ions produced.

   Example: `"secondaryelectrons",0` will turn on secondary electrons and turn off scattered electrons.

6. `<gas density or "name of density file">` The user must specify the density of the target gas particle, either as a numerical value (in m^{-3}) or as a 1-D density map, in the specified ionization region. A non-string assumes a constant density throughout the ionization region. If a density map is used, then the filename is specified as a string (e.g. "densitymap.gdf"). The density map must be a GDF file and must either be in the same directory as the GPT batch file, or the explicit

path to the file must be included in the filename. See section 6.2 for more info on creating a density map.

Example: `3.5e11` ($\sim 10^{-11}$ Torr), or `"H2density.gdf"`.

7. `<gas temperature T>` The user specifies the temperature of the target gas particles, in Kelvin. The temperature must be specified as a positive real number. The temperature is used in the Maxwellian distribution function.

   Example: `293.15` (Room temperature).

8. `<ion fraction k>` The user specifies the fraction of energy `k` that goes to the ion, with the remaining energy going to the scattered electron. This fraction must be a real number between 0 and 1 (non-inclusive). See section 5.4 for more info.

   Example: `0.01` (1%).

9. `<integration precision (nsteps)>` The user specifies an integer number of steps, `nsteps`, used by the integration functions to numerically integrate the secondary electron differential cross section and the Maxwellian distribution function. Higher values of `nsteps` provide higher accuracy for the integration functions at the cost of higher CPU time. See section 5.2 for more info.

   Example: `200`.

5.1.1 Eligibility

A given simulation particle is eligible to ionize if it meets the following conditions:

1. The particle is present in the simulation and will not be removed during this timestep. If the particle is removed by any GPT element (such as `zminmax` or `rmax`) the particle is not eligible for ionization.

2. The particle is an electron macro-particle. The electron must be in the primary, secondary, or scattered electron particle sets. If the particle is in any other particle set, it is not eligible for ionization.

3. The coordinate system of the particle must match the ECS defined in the input parameters.

4. The particle will not leave the ionization region during the timestep. To test this, a displacement vector is calculated from the position of the particle at the start and end of the timestep. The particle is eligible to ionize if the *center* of the displacement vector is within the ionization region. **NOTE**: It is possible for a particle to enter and exit the ionization region during a single timestep, especially

if the timestep is large, or if the general motion of the particle is complex. Using the center of the displacement vector introduces uncertainty in the simulation. However, it is unlikely that this uncertainty will be significant for most applications.

5. The kinetic energy of the particle T_e is above the ionization energy of the target gas particle. The kinetic energy is calculated using the Lorentz factor at the end of the timestep.

5.1.2 Ion Production and Monte Carlo

For each eligible electron macro-particle, a Monte Carlo approach is used to determine whether it ionizes in this timestep. The number of ions, N_{ion}, created by the macro-particle is calculated using Eq. 1 with the gas density defined in the input parameters. If N_{ion} is greater than one for *any* eligible electron macro-particle, then the timestep fails and is retried with smaller timesteps until N_{ion} is less than one for *all* particles. While it is unlikely that N_{ion} can be greater than one, since the ionization cross section for a single electron is tiny, N_{ion} can become greater than one if the electron macro-particle represents a large number of electrons (i.e., N_e is large), the timestep is long, or if the gas density is high.

Once N_{ion} is less than one for all particles, a random number generator selects a number between 0 and 1. If the random number is greater than N_{ion}, then the particle does not ionize and the simulation continues to the next particle. Figure 7 shows a flow chart of the Monte Carlo algorithm [2].

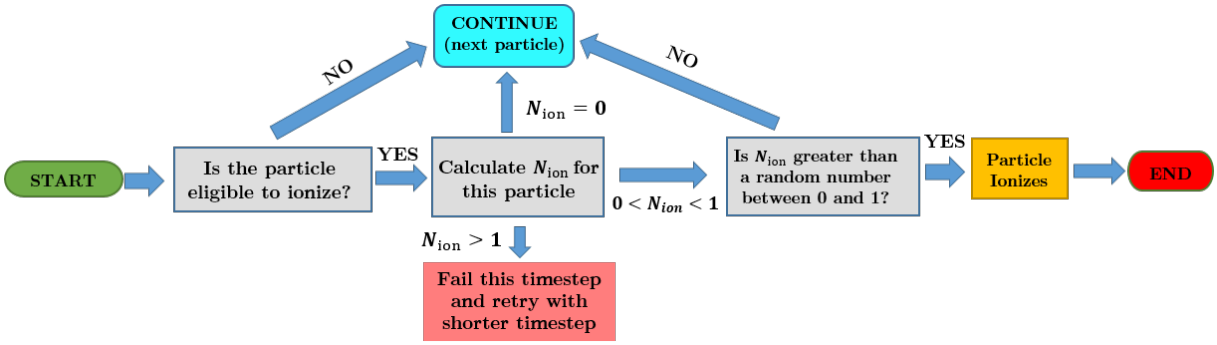

Figure 7: Monte Carlo algorithm for ion production.

5.2 Secondary Electron Energy Algorithm

Once it is determined that an electron macro-particle ionizes, the distribution of possible energies W for the secondary electron is calculated using the differential cross section (equation 5 or 6) based on the user-defined gas species and orbital. Equation 5

is used if the gas species is H_2 or He, otherwise equation 6 is used. See Table 3 for a list of all supported gas species and orbitals.

The probability distribution given by the differential cross section is clearly non-uniform, as Figure 4 shows, while the distribution of numbers between 0 and 1 generated by the random number generator *is* uniform. To select the secondary electron energy, the following algorithm is used:

1. The maximum secondary electron energy is based on the primary electron energy and is defined as $E_{max} = 0.75(T - B)$, as the differential cross section becomes unphysical past this energy.
2. We define a function $F(N)$ that approximates the integral of the differential cross section between $E = E_0 = 0$ and $E = E_N$ using the trapezoidal rule with `nsteps` partitions (defined in the input parameters).

$$F(N) = \sum_{i=1}^{N} 0.5 \frac{E_{max}}{\texttt{nsteps}} (f(E_N) + f(E_{N-1}))$$

3. We define a second function $g(E)$ as the ratio:

$$g(E) = \frac{f(E_N)}{f(E_{max})}$$

   Clearly $g(E) \in [0, 1]$.

4. A random number between 0 and 1 is chosen using a random number generator. Then $g(E)$ is calculated for successive N until $g(E)$ is greater than the random number, at which point E is chosen to be the secondary electron energy.

Once the secondary electron energy is chosen, its momentum can be calculated as:

$$\vec{p}_{sec} = \left|\vec{p}_{sec}\right| \hat{p}_{sec} \tag{9}$$

$$\left|\vec{p}_{sec}\right| = m_{sec} c(\gamma_{sec}^2 - 1.0) \tag{10}$$

$$\gamma_{sec} = \frac{W}{m_e c^2} + 1$$

The direction of the secondary electron, $\hat{p}_{sec}$ is assumed to be random. To produce a random unit vector for $\hat{p}_{sec}$, the components are written in spherical coordinates:

$$\begin{aligned} x &= r \sin(\theta) \cos(\phi) \\ y &= r \sin(\theta) \sin(\phi) \\ z &= r \cos(\theta) \end{aligned} \tag{11}$$

where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ (θ and ϕ are analogous to latitude and longitude respectively). A random unit vector can be generated by letting $r = 1$ and choosing $\theta = n_1\pi$ and $\phi = 2n_2\pi$ where n_1 and n_2 are random numbers between 0 and 1.

5.3 Target Gas Particle

Assuming the target gas particle follows the ideal gas law, the energy of the target gas particle is based on a Maxwell-Boltzmann velocity distribution (Eq. 7) with the gas temperature defined in the input parameters. The target gas velocity is chosen using a similar algorithm to the secondary electron energy algorithm in the previous subsection. Once the velocity is chosen, the energy of the target gas particle is given by:

$$E_{gas} = (\gamma - 1)mc^2 \tag{12}$$
$$\gamma = \left(1 - \frac{v^2}{c^2}\right)^{-\frac{1}{2}}$$

The momentum of the gas particle is then calculated as:

$$\vec{p}_{gas} = \left|\vec{p}_{gas}\right| \hat{p}_{gas} \tag{13}$$
$$\left|\vec{p}_{gas}\right| = m_{gas}c(\gamma_{gas}^2 - 1.0) \tag{14}$$

where $\hat{p}_{gas}$ is assumed to be random and is calculated using Eqs. 11.

NOTE: IONATOR does not produce a target gas particle in the simulation. Instead, the parameters of the target gas particle are calculated solely for the purposes of calculating the parameters of the ion and scattered electron, as described in the following subsection.

5.4 Ion and Scattered Electron

The energy of the ion is calculated using the formula:

$$E_{ion} = E_{comb}k \tag{15}$$

where k is a number between 0 and 1 (non-inclusive) and is set by the user in the input parameters. E_{comb} is the combined energy of the ion and scattered electron defined in Eq. 8. The momentum of the ion is calculated in a similar way as the momentum of the target gas particle:

$$\vec{p}_{ion} = \left|\vec{p}_{ion}\right| \hat{p}_{ion} \tag{16}$$
$$\left|\vec{p}_{ion}\right| = m_{ion}c(\gamma_{ion}^2 - 1.0) \tag{17}$$

with its direction assumed to be random. The energy of the scattered electron is:

$$E_{scat} = E_{comb}(1 - k) \tag{18}$$

Its momentum is calculated from momentum conservation:

$$\vec{p}_{scat} = \vec{p}_e + \vec{p}_{gas} - \vec{p}_{ion} - \vec{p}_{sec} \tag{19}$$

$$\hat{p} = \frac{\vec{p}_{scat}}{|\vec{p}_{scat}|} \tag{20}$$

After all kinematic parameters are calculated, the ion, secondary electron, and scattered electron are added to the simulation. The ion is placed in a random location along the trajectory of the primary electron during this timestep, defined by the primary electron's initial position, $\vec{r}_{start}$ and final position, $\vec{r}_{end}$:

$$\vec{r}_{ion} = \vec{r}_{start} + \lambda\Delta\vec{r} + (1.0 - \lambda)\Delta t\vec{\beta_{ion}}c \tag{21}$$
$$\Delta r = \vec{r}_{end} - \vec{r}_{start}$$

where λ is a random number between 0 and 1. The secondary electron and scattered electron are then placed at the location of the ion. Finally, the number of macro-particles the primary electron represents is reduced by 1.

5.5 Summary of Kinematic Parameters

Table 2 summarizes the kinematic parameters of all particles involved in ionization.

Particle	Kinematic Parameter	Symbol	Calculated From
Primary Electron	Energy	T	Known/Given
	Momentum Magnitude	$\lvert\vec{p}_e\rvert$	
	Momentum Direction	$\hat{p}_e$	
Ion	Energy	T_{ion}	Eq. 15
	Momentum Magnitude	$\lvert\vec{p}_{ion}\rvert$	Eq. 17
	Momentum Direction	$\hat{p}_{ion}$	Eq. 11
Secondary Electron	Energy	W	MC routine (Eq. 5 or 6)
	Momentum Magnitude	$\lvert\vec{p}_{sec}\rvert$	Eq. 10
	Momentum Direction	$\hat{p}_{sec}$	Eq. 11
Target Gas Particle	Energy	T_{gas}	MC routine (Eq. 7)
	Momentum Magnitude	$\lvert\vec{p}_{gas}\rvert$	Eq. 14
	Momentum Direction	$\hat{p}_{gas}$	Eq. 11
Scattered Electron	Energy	T_{scat}	Eq. 18
	Momentum Magnitude	$\lvert\vec{p}_{scat}\rvert$	Eq. 19
	Momentum Direction	$\hat{p}_{scat}$	Eq. 20

Table 2: Table of kinematic parameters for each particle involved in ionization.

6 IONATOR Features

6.1 Gas Species and Orbitals

Secondary electrons can originate from any orbital in a target gas particle. The differential cross section for the energy of a secondary electron from a given orbital depends on the gas species. Though the secondary electron is more likely to originate in an outer (valence) shell than an inner shell due to outer shells having lower ionization energies.

Table 3 below lists the parameters used in the differential cross section formulas (Eqs. 5 and 6) for each orbital in each supported gas species. Both the gas species and the orbital must be specified in the input parameters as strings (see section 5.1). The parameters originate in Binary Encounter Dipole Model original article [6], or the NIST Electron-Impact Cross Sections database [9]. They are stored in the IONATOR infostructures file. Future updates will provide more supported gas species.

Gas Species	Gas Spec.	Orbital	Orbital Spec.	B (eV)	U (eV)	N (eV)
H_2	“H2"	$1\sigma_g$	“default"	15.43	25.68	2.0
He	“He"	$1s$	“default"	24.59	39.51	2.0
CO	“CO"	5σ	“5sigma" or “default"	14.01	42.26	2.0
		1π	“1pi"	17.66	54.30	4.0
		4σ	“4sigma"	21.92	73.18	2.0
		3σ	“3sigma"	41.92	79.63	2.0
CH_4	“CH4"	$1t_2$	“1t2" or “default"	14.25	25.96	6.0
		$2a1$	“2a1"	25.73	33.05	2.0
CO_2	“CO2"	$1\pi_g$	“1pig" or “default"	13.77	64.43	4.0
		$1\pi_u$	“1piu"	19.70	49.97	4.0
		$3\sigma_{2u}$	“3sigma2u"	20.27	71.56	2.0
		$4\sigma_{1g}$	“4sigma1g"	21.62	74.66	2.0
		$2\sigma_{2u}$	“2sigma2u"	40.60	78.38	2.0
		$3\sigma_{1g}$	“3sigma1g"	42.04	75.72	2.0

Table 3: Differential cross section parameters for each orbital in each supported gas species.

6.2 Gas Density

The gas density is used in the calculation of the ion production rate in Eq. 1. It can be specified as either a constant (in m^{-3}) or as a 1-D density map, assuming cylindrical symmetry about the *z*-axis. For the latter case, the filename for the density map must be specified as a string: “`<filename>.gdf`".

The density map must be a GDF file with exactly two columns: `z` and `density`. While the `z` values must in ascending order, they do not have to be equidistant. If the 1-D density is an ASCII (text) file of the form:

z	density
0	0.0
0.1	4e-11
0.2	5e-11
0.5	5e-11
...	...

then the file can be converted to a GDF file by running ASCI2GDF:

```
asci2gdf -o <filename>.gdf <filename>.txt
```

IONATOR will read the GDF file and interpolate the 1-D data using a natural cubic spline (in a similar manner to the built-in `map1D_B` and `map1D_E` GPT elements). Of course, the accuracy of the interpolation will depend on how “smooth" the density data is. Adding more data points near erratic density values will increase the accuracy of the cubic spline.

WARNING: The *z* values in the density file will be shifted to the *center* of the ionization region, defined by the ECS input parameter. If the ionization region is between $0 \leq z \leq 1$, then the first few input parameters `"wcs","z",0.5,1.0, ...` will define the desired ionization region, but shift all *z* values in the density file by 0.5. To adjust for this, you must manually shift the *z* values in the density file accordingly. A future update will (hopefully) make this shift unnecessary.

6.3 Output Ion Statistics Using writeIonInfo

Often it is useful to get statistics on the locations and production rates of ions produced in ionization simulations. Instead of sifting through the simulation file and extracting ion data at each timestep, the custom element `writeIonInfo.c` (separate from IONATOR) can be used to write parameters to an external GDF file (separate from the simulation file) whenever a new ion is produced in the simulation.

To use writeIonInfo, it must be installed in a similar manner to IONATOR (see section 3). It can be called in the input file using two arguments: the ECS and the filename. For the ECS specification, use `"wcs", "I"`, unless using a custom coordinate system (CCS).

```
writeIonInfo(ECS,<filename>.gdf)
```

GPT calls the writeIonInfo element at the *end* of every successful timestep. writeIonInfo will scan through all particles and determine if it is a “new" ion. To determine if a given simulation particle is a new ion, three conditions must be met:

1. The particle must have a positive charge.
2. The particle must be present in the simulation and not be removed at the end of the timestep.
3. The particle must have an ID # that is not in writeIonInfo’s list of ion ID #’s.

If all three conditions are met, then the ion parameters are stored C++ vectors, one vector for each parameter. Then the ion ID # is added to an unordered list for fast ID lookup. At the end of the simulation, the vectors are written to the GDF file. Table 4 lists the names and descriptions of the parameters written.

Parameter	Name	Description
ID	`ionID`	ID # for this ion
t_{start}	`tstart`	Timestep begin (s)
t_{end}	`tend`	Timestep end (s)
Δt (= $t_{end} - t_{start}$)	`dt`	Timestep length (s)
x	`ionWr_x`	Ion x position (m)
y	`ionWr_y`	Ion y position (m)
z	`ionWr_z`	Ion z position (m)
γ_{ion}	`ionG`	Ion Lorentz factor
$\gamma\beta_x$	`ionGBr_x`	Ion x $\gamma\beta$ momentum
$\gamma\beta_y$	`ionGBr_y`	Ion y $\gamma\beta$ momentum
$\gamma\beta_z$	`ionGBr_z`	Ion z $\gamma\beta$ momentum
E_{ion}	`ionEnergy`	Ion energy (eV)
m_{ion}	`ionMass`	Ion mass (kg)

Table 4: Ion parameters written to the writeIonInfo GDF file.

6.4 Color Coding Particles Using colorcoding

When viewing a simulation, GPT allows the user to color code the particles by built-in parameters, such as mass, charge, and position. Particles that have similar parameters, such as primary, secondary, and scattered electrons cannot easily be distinguished by built-in parameters. Thus, the GPT custom element `colorcoding.c` was created for this purpose. colorcoding allows the user to color code simulation particles by particle set name. The syntax is:

```
colorcoding(
"<Set 1>", <color value 1>,
"<Set 2>", <color value 2>,
...
"<Set 5>", <color value 5>)
```

The colorcoding function takes an even number of input parameters and supports up to five different particle types. Each particle set to be colored must have its set name and color value set in the input parameters. In addition, the particle set must be defined earlier in the input file, either with the GPT `setparticles()` function or with IONATOR. colorcoding gives each particle a new parameter called `color` that can be used as the color coding parameter instead of one of the built-in particle parameters. Color values are real numbers (positive or negative) that are mapped onto a color scheme from blue to red. That is, the lowest color value is colored blue and the highest color value is colored red. While any color values can be used, it is convenient to use the color values in Table 5 below to assign specific colors to particle sets.

Note that because color values are mapped onto a blue-red color scheme regardless of their value, one particle set will always be colored blue and another set red. If a particle set should have a specific color other than blue or red, then one work-around is to assign one or more "dummy" particle sets using the `setparticles()` function in the input file, then immediately removing all particles in the dummy sets using the `setreduce()` function. Note that if all particles have the same color value, then all particles are colored green.

Color Value	Color
1.0	Blue
2.0	Light Blue
3.0	Cyan
4.0	Turquoise
5.0	Green
6.0	Yellow-Green
7.0	Yellow
8.0	Yellow-Orange
9.0	Orange
10.0	Red

Table 5: Table of color values.

To color code a given simulation, use the following steps:

1. Open a GPT simulation.

2. Open plot settings, either by hitting the Enter-key, right clicking on the simulation and clicking "Settings", or clicking "Settings" in the Plot menu in the top menu bar.
3. Under the "Scatter" tab, check the box next to "Color coded".
4. Select the array to color code by. When using colorcoding, select the `color` array.
5. Select either automatic scaling (using the lowest and highest values of the selected array) or manual scaling.
6. Click "OK".

7 IONATOR Examples

The following examples demonstrate how to use IONATOR in a variety of applications and how to interpret and analyze the resulting simulations.

7.1 Example 1: Basic Ionization Simulation

In this example, we analyze two different simulations: one involving a single electron macro-particle and another involving an electron macro-particle bunch. Both the single macro-particle and macro-particle bunch represent the same amount of charge, 1 µC. The electrons are given an initial kinetic energy of 1 keV and travel in the +z-direction for 0.5 m, ionizing H_2 gas along its trajectory. The density of the H_2 gas is 3×10^{10} m^{-3}.

Listing 2 shows the input file for the single particle simulation. To simulate an electron bunch, replace the `setstartpar` command with the commands in Listing 3. To run the simulation, copy the commands into a GPT input file `input.in` and run the following command:

```
gpt -o result.gdf input.in
```

Figure 8 shows a snapshot of the resulting simulations at t = 20 ns, as viewed in the yz-plane. Several observations can be made:

1. The locations of the ions follow the trajectory of the primary electron macro-particle (or bunch).

2. No ions are created by the secondary and scattered electrons. While possible, it is unlikely for them to do so, as the ionization probability of a single electron is much smaller than combined probability of all primary electrons represented by the electron macro-particle or electron bunch.

3. The ion production rate is independent of the number of primary electron macro-particles used in a simulation, provided the total number of primary electrons represented remains constant. This observation is expected, as the number of ions created, N_{ion}, is independent of position provided there is uniform gas density (see Eq. 1). To check this, rerun the simulation with the writeIonInfo routine and count the number of entries in the writeIonInfo file. (Note that because IONATOR employs a Monte Carlo routine, the number of ions produced in the two simulations will likely be different, though they should be similar to within 1%. If rerun many times using different seed values with the GPT `randomize` command, the average ion production rate will be same).

4. The ion production rate is also independent of the simulation timesteps selected in the `tout` command. To check this, see what happens if you choose different timestep lengths (the third argument in the `tout` command)

5. The secondary and scattered electrons do in fact travel in random directions, despite the unequal scales of the two axes in the snapshots making it seem like they travel more vertically (along the *y*-axis).

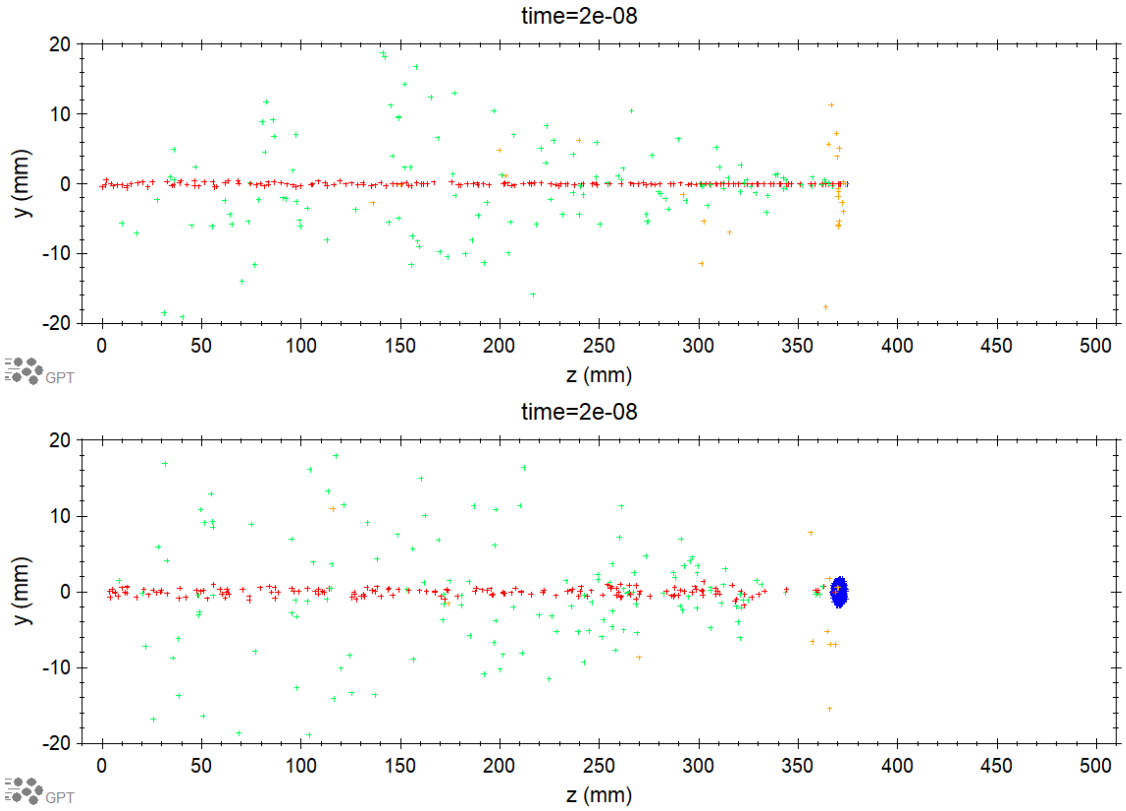

Figure 8: Snapshots of the single particle (top) and electron bunch simulations at t = 20 ns. The simulations are color coded as follows: primary electron (blue), secondary electron (green), scattered electron (orange), H_2^+ ion (red).

```
#Electron bunch parameters
eKE       = 1000;                       #Kinetic energy (in eV)
erestmass = me*c^2/(-qe);               #Electron rest mass in eV
egamma    = (eKE/erestmass)+1;          #Lorentz factor
ebeta     = sqrt(egamma^2-1.0)/egamma;  #Beta factor
Qtot      = -1e-6;                      #Total bunch charge in C

#Set electron macro-particle set
setstartpar("electrons",0,0,0,0,0,egamma*ebeta,me,qe,Qtot/qe);

#Particle removal
Zmin = -0.001;                          #Minimum z (in meters)
Zmax = 0.5;                             #Maximum z (in meters)
R = 0.02;                               #Maximum r (in meters)
zminmax("wcs","I",Zmin,Zmax);           #Remove if z>Zmax or z<Zmin
rmax("wcs","I",);                       #Remove if r>R

#IONATOR
length = 0.5;                           #Length of element (in z)
center = 0.25;                          #Center of element (in z)
gasSetName = "H2";                      #H2 set name
ionSetName = "ions";                    #H2 ion set name
seSetName = "secE";                     #Secondary electron set name
scatSetName = "scatE";                  #Scattered electron set name
H2density = 3e10;                       #H2 density
H2Temp = 293.15;                        #H2 Temperature (in K)
ionK = 0.01;                            #Ion energy fraction
precision = 200;                        #Precision (nsteps)

ionator("wcs", "z", center, length, gasSetName, "default", "electrons",
ionSetName, seSetName, scatSetName, H2density, H2Temp, ionK, precision);

#Color Coding
colorcoding("electrons", 1.0, "scatE", 9.0, "secE", 5.0, "ions", 10.0);

#Output
tout(0, 2.8e-8, 2e-10);
```

Listing 2: Input File for single electron macro-particle simulation.

```
#Set electron macro-particle bunch
setparticles("electrons",1e4,me,qe,Qtot);

#Set bunch parameters
len = 50e-12;                                    #RMS bunch length (s)
sigma = 0.0005;                                  #Sigma (m)
cutoff = 4;                                      #Cutoff at 4*sigma.
setrxydist("electrons","g",0,sigma,0,cutoff);    #Gaussian radial distribution
setphidist("electrons","u",0,2*pi);              #Uniform angular distribution
settdist("electrons","g",len*cutoff,len,cutoff,cutoff); #Gaussian temp. dist.
setGdist("electrons","u",egamma,0);              #Uniform energy distribution
```

Listing 3: Input file commands for the electron bunch simulation. Remove the `setstartpar` command in Listing 2 and add these commands to simulate an electron bunch instead of a single macro-particle.

7.2 Example 2: Ionization simulation with linearly increasing density

In this example, we use the user-defined gas density feature (described in section 6.2) to create an ionization simulation with a CH_4 gas density that linearly increases along z for 1 m

To create the density file, copy the lines in Listing 7.2 below into an ASCII (text) file. Note that because the density data is interpolated, it does not matter that there are no 0.05 increments in density data past $z = 0.25$. We will verify this in the simulation analysis. Because the ionization region is between $0 \leq z \leq 1$, the z values have been shifted accordingly.

Save the file as `density.txt`, then convert it into a GDF file using the command:

```
asci2gdf -o density.gdf density.txt
```

Run the simulation using the input file in Listing 7.2. To analyze the simulation, the writeIonInfo command is included in the input file to output ionization statistics. A snapshot of the simulation output is shown in Figure 9.

We can visually see the density of ions increasing along the z-axis. To prove that the ion production rate scales linearly with z, we can analyze the ion info file. Run the `gdf2his` program on this file using the command line:

```
gdf2his -b -o Ex2_IonInfo_Histograms.gdf Ex2_IonInfo.gdf ionWr_z 0.05
```

```
 z       density
-0.5     0.0
-0.45    5E9
-0.4     1E10
-0.35    1.5E10
-0.3     2E10
-0.25    2.5E10
-0.2     3E10
-0.1     4E10
 0.0     5E10
 0.1     6E10
 0.2     7E10
 0.3     8E10
 0.4     9E10
 0.5     1E11
```

Listing 4: Density file for linearly increasing gas density. Note the shifted z-values.

The resulting file is a copy of the original writeIonInfo file with four new arrays: `ionWr_z_cnt`, `ionWr_z_cntbar`, `ionWr_z_his`, and `ionWr_z_hisbar`. Open the file and plot `ionWr_z_cntbar` vs. `ionWr_z_hisbar` as a line plot. The result is shown in Figure 10. We see that the number of ions produced scales roughly linearly with z. The slight variation from a linear histogram is due to poor statistics. If we increase each density point by a factor of 10 and rerun the simulation, we see that the histogram follows a more linear trend (Figure 11).

```
#Electron bunch parameters
eKE       = 1000;                       #Kinetic energy (in eV)
erestmass = me*c^2/(-qe);               #Electron rest mass in eV
egamma    = (eKE/erestmass)+1;          #Lorentz factor
ebeta     = sqrt(egamma^2-1.0)/egamma;  #Beta factor
Qtot      = -1e-6;                      #Total bunch charge in C

#Set electron macro-particle bunch
setparticles("electrons",1e4,me,qe,Qtot);

#Set initial distribution parameters
len = 50e-12;                                   #RMS bunch length (s)
sigma = 0.0005;                                 #Sigma (m)
cutoff = 4;                                     #Cutoff at 4*sigma.
setrxydist("electrons","g",0,sigma,0,cutoff);   #Gaussian radial distribution
setphidist("electrons","u",0,2*pi);             #Uniform angular distribution
settdist("electrons","g",len*cutoff,len,cutoff,cutoff); #Gaussian temp. dist.
setGdist("electrons","u",egamma,0);             #Uniform energy distribution

#Particle removal
Zmin = -0.001;                  #Minimum z (in meters)
Zmax = 0.5;                     #Maximum z (in meters)
R = 0.02;                       #Maximum r (in meters)
zminmax("wcs","I",Zmin,Zmax);   #Remove particle if z>Zmax or z<Zmin
rmax("wcs","I",R);              #Remove particle if r>R

#IONATOR
ionator(
"wcs", "z", 0.5, 1.0, "CH4", "default", "electrons", "ions",
"secE", "scatE", "density.gdf", 293.15, 0.01, 200);

#writeIonInfo and colorcoding
writeIonInfo("wcs","I","Ex2_IonInfo.gdf");
colorcoding("electrons", 1.0, "scatE", 9.0, "secE", 5.0, "ions", 10.0);

#Output
tout(0, 2.8e-8, 2e-10);
```

Listing 5: Input file for Example 2.

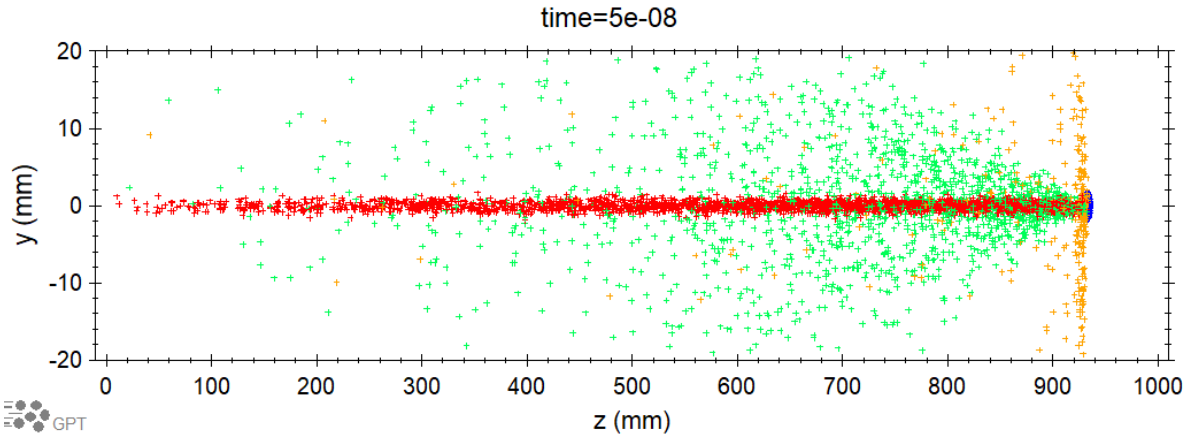

Figure 9: Snapshot of the simulation for Example 2 at t = 50 ns.

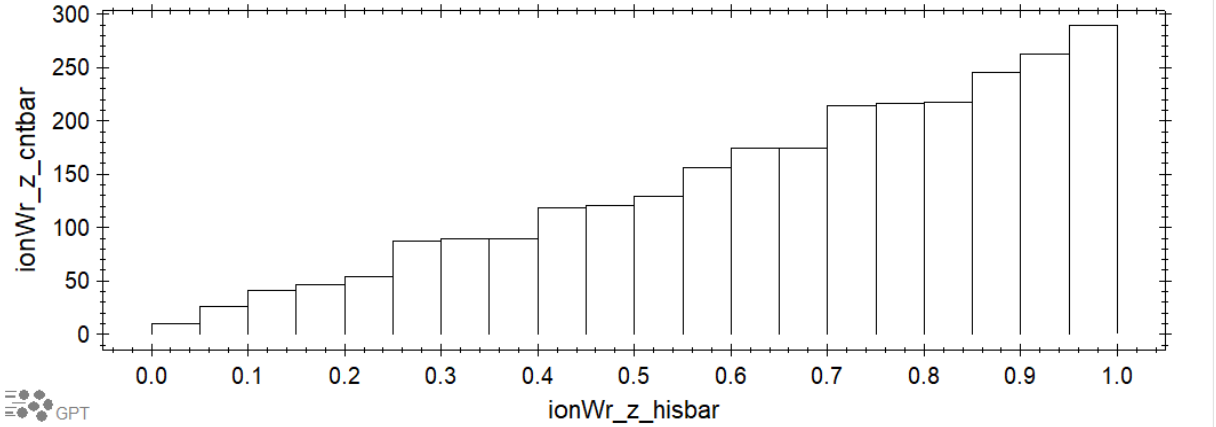

Figure 10: Histogram of the number of ions vs. z position with a bin width of 0.05 m.

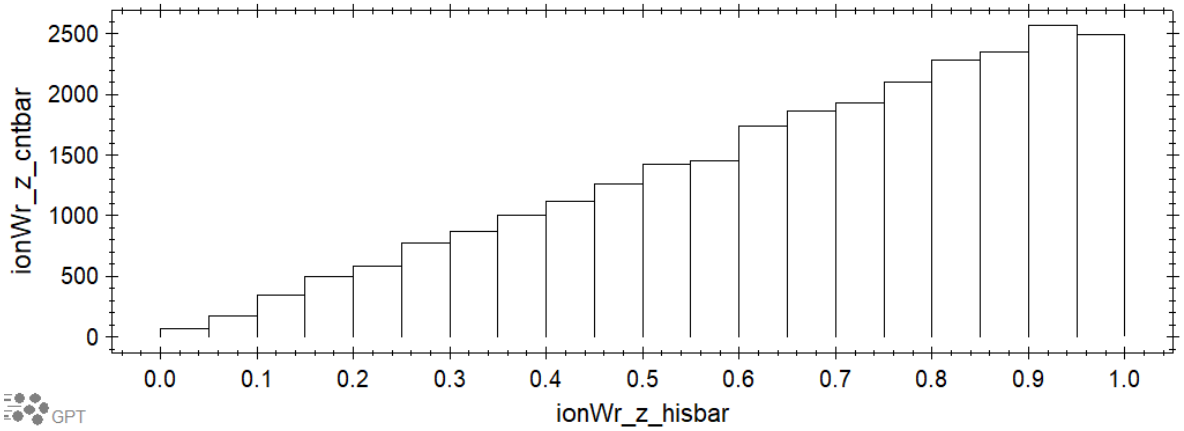

Figure 11: Histogram of the number of ions vs. z position with a bin width of 0.05 m with the density data increased by a factor 10.

8 Troubleshooting

The following table lists possible errors and troubleshooting tips when using IONATOR, writeIonInfo, and colorcoding. Errors in quotes are error messages written to cerr, which is usually displayed in the lower part of the batch file window when using GPTwin, or in the command-line interface when using Linux.

If you encounter an error not on this list, consult the GPT User Manual and GPT Programmers Reference manual, which can be accessed under the Help menu when using GPTwin, to help pinpoint the problem. If you still cannot figure out the problem despite your best efforts, or have found an inaccuracy in IONATOR, please email me at yoskowij@jlab.org and include as much detail about your problem as possible.

Error	Description
“Syntax: ionator(ECS, L, gasname, orbitalname, ..."	Syntax error. Ensure that: 1. All required arguments are present, 2.All arguments are spelled correctly, and 3. The ECS specification contains the correct number of arguments. See section 5.1 for more info.
“Invalid orbital name for <gasname>: <spec>. Choices are..."	Invalid orbital name. Ensure that the spelling of the orbital name matches one of the choices given.
“Unknown Gas Species: <gas spec>"	Invalid gas species. Ensure that the gas name matches one of the supported gas species in Table 3 and ensure that it is spelled correctly.
“String expected, found number <spec>" / “Number expected, found string <spec>" / “Value is not an integer: <spec>"	Invalid argument type. Ensure that the argument type is correct. Strings must be surrounded by quotes and integers cannot contain a decimal point.
“gasTemp must be above absolute zero! gasTemp = <spec>"	Invalid temperature. Unfortunately, according to the 3rd law of thermodynamics, we cannot reach absolute zero, nor go below it. Ensure that the gas temperature specification is a positive real number above zero.
“ionK must be between 0 and 1! ionK = <spec>"	Invalid ion fraction. The ionK specification must be a real number between 0 and 1. See section 5.4 for more info on this parameter.

“z array must be in ascending order"	*z* column in density file is not in ascending order. For fast reading of the density file, the density data in the density file must be sorted with the *z* column in ascending order. Also, ensure there are no duplicate data points.
“Error calculating spline"	Error calculating spline. GPT was unable to interpolate the density data in the density file. Ensure that the density data is readable and contains no errors/typos. If you still get this error, try copying the density data into a new text file and run `asci2gdf`.
“Could not calculate SE energy with sufficient accuracy. Try increasing nsteps" / “Could not calculate Gas energy with sufficient accuracy. Try increasing nsteps"	Insufficient precision in the integration function to calculate the secondary electron or target gas particle energy. Try increasing the `nsteps` input parameter.
“Syntax: writeIonInfo(ECS,filename)"	Syntax error. Ensure that: 1. All required arguments are present, 2.All arguments are spelled correctly, and 3. The ECS specification contains the correct number of arguments.
“Overflow in writeIonIDs: <number> > INT_MAX"	Overflow in vector array. The number of ions to be written to the writeIonInfo file exceeds INT_MAX, which is 2,147,483,647. Unfortunately, your simulation has produced too many ions for writeIonInfo to handle! Because the ion production rate scales with the number of primary electron charge (Eq. 1), you can try reducing the total number primary electrons and multiply the resulting number of ions produced by this number to get the true number of ions produced.
“Syntax: colorcoding([setname, color, [setname, color, ...]])"	Syntax error. Ensure that all required arguments are present and spelled correctly. Note that colorcoding takes an even number of arguments. Every setname argument must be followed by a color value argument.

“Invalid # colors calculation"	Invalid number of colors. The number of colors specified in colorcoding must be between one and five. No more than five colors can be specified, so colorcoding cannot have more than 10 arguments.
“Tried to assign color to particle in undefined particle set, <spec>"	Undefined particle set to be colored. Ensure all particle set names in the input parameters are defined somewhere in the input file. Also ensure that the set names are spelled correctly in the colorcoding input parameters.
“GPT Finished" appears abruptly without the simulation running to completion and without any error message	This usually indicates a division by zero error. When encountering this error, GPT will immediately terminate the program and return a “GPT Finished" message. Unfortunately, there is usually no accompanying error message to tell you where or when this error occurs in the simulation. Incrementally reducing the complexity and rerunning the simulation can usually help pinpoint the problem. You can also try using a different seed value using the GPT `randomize` function.

References

[1] J. Biswas and E. Wang, “New dynamic ionizer element to simulate ion back bombardment in DC gun", Zenodo, 2018, doi:10.5281/zenodo.1413642

[2] J. T. Yoskowitz, “Ion production and mitigation in DC high-voltage photo-guns", Ph.D. thesis, Phys. Dept. Old Dominion University, Norfolk, VA, 2021.

[3] M. Reiser, “Linear beam optics with space charge", in *Theory and Design of Charged Particle Beams* Weinheim, Germany: Wiley VCH Verlag GmbH, 2008, pp. 163–272

[4] H. Bethe, “Zur theorie des durchgangs schneller korpuskularstrahlen durch materie (On the theory of the passage of fast corpuscular rays through matter)", *Ann. Phys. (Leipzig)*, vol. 397, no. 3, pp. 325–400, 1930, doi:10.1002/andp.19303970303

[5] F. F. Rieke and W. Prepejchal, “Ionization cross sections of gaseous atoms and molecules for high-energy electrons and positrons", *Phys. Rev. A*, vol. 6, no. 4, pp. 1507-1519, Oct. 1972, doi:10.1103/PhysRevA.6.1507

[6] Y. Kim and M. E. Rudd, “Binary-encounter dipole model for electron-impact ionization", *Phys. Rev. A*, vol. 50, no. 5, pp. 3954–3967, Nov. 1994, doi:10.1103/PhysRevA.50.3954

[7] E. W. McDaniel, *Atomic Collisions : Electron and Photon Projectiles*, New York, NY, USA: Wiley, 1989.

[8] G. H. Dunn and L. J. Kieffer, “Dissociative ionization of H_2: A study of angular distributions and energy distributions of resultant fast protons", *Phys. Rev.*, vol. 132, no. 5, pp. 2109–2117, Dec. 1963, doi:10.1103/PhysRev.132.2109

[9] National Institute of Standards and Technology (NIST), “Electron-Impact Cross Sections for Ionization and Excitation", August 2005, https://www.nist.gov/pml/electron-impact-cross-sections-ionization-and-excitation-database