UNIVERSITY OF CALIFORNIA

Los Angeles

Electrospray Plume Evolution and Divergence

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Aerospace Engineering

by

McKenna Davis

2024

ABSTRACT OF THE DISSERTATION

Electrospray Plume Evolution and Divergence

by

McKenna Davis
Doctor of Philosophy in Aerospace Engineering
University of California, Los Angeles, 2024
Professor Richard E. Wirz, Chair

Electrospray thrusters require significant improvements in operational lifetime for use in multi-year spacecraft propulsion missions. The primary thruster lifetime-limiting mechanism is propellant overspray, in which wide-angle particles impinge on and saturate downstream electrodes instead of exiting through the electrode aperture and contributing to produced thrust. Electrospray particles are emitted within a small radial range, but diverge as they move downstream from emission to form a 3D plume, the edges of which contribute to overspray. In order to improve electrospray thruster designs towards minimizing overspray and optimizing operational lifetime, we need to understand what causes electrospray plume divergence.

This dissertation investigates electrospray plume divergence using the Discrete Electrospray Lagrangian Interaction (DELI) Model to simulate electrospray particle dynamics. The governing equation for particle propagation includes the applied electrostatic force from the potential difference between the emitter and downstream electrode, the Coulomb forces between particles (including image charges), and the drag force. Each of these forces is investigated theoretically and computationally to determine its influence on plume divergence.

None of the forces introduce radial divergence into a set of particles emitted straight down the axis of emission with no range in radial coordinate. However, electrospray particles are always emitted with some small range in radial coordinate due to hydrodynamic instabilities and minute asymmetries in the emitter. All three forces exacerbate existing radial divergence among a set of particles: the applied electric field has a radial component due to jet curvature and the electrode aperture; there is a radial component to Coulomb forces between particles with a difference in radial coordinate; and drag counters particle motion, keeping particles in a clustered state in which Coulomb forces are magnified.

Simulations compare the radial divergence of groups of particles with equal velocities and with an upstream velocity gradient, in which upstream particles are moving faster than their forward neighbors. In the upstream velocity gradient case, faster particles catch up to their forward neighbors, magnifying the Coulomb interaction between the two in response to their increased proximity. We term this interaction a ‘traffic jam’ and correlate it with increased plume divergence through Coulomb interactions. We present two novel means of characterizing plume divergence: 1) a metric for positional divergence based on three standards of a Gaussian or Super-Gaussian fit to particle mass density distribution as a function of radial coordinate, and 2) emittance as a metric for positional and velocity divergence. We further describe how emittance can be used to identify when an electrospray plume has reached the steady state.

Machine learning is applied for the first time to electrospray particle dynamics data, produced by the DELI Model. Results demonstrate predictive abilities for downstream particle dynamic properties given particle properties at emission. Furthermore, a novel method is proposed for combining experimental electrospray particle data, computational plume evolution models, and machine learning algorithms to optimize diagnostic design.

In summary, this dissertation presents a comprehensive consideration of electrospray plume divergence using computational and analytical models supported by experimental data. The origins and sources of growth of electrospray plume divergence are identified, new metrics for electrospray plume divergence are presented, and machine learning algorithms are developed to predict electrospray plume divergence.

The dissertation of McKenna Davis is approved.

University of California, Los Angeles

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Nomenclature

Physical Constants

g	Gravitational constant
k_B	Boltzmann constant

Superscripts

n	Order
t	Time step index

Subscripts

inf	Free-stream
A	Applied
B	Beam
C	Coulomb
Ch	Cheng
Cu	Cunningham
D	Drag
d	Drift
dom	Domain
el	Electrode
em	Emission
f	Final
fl	Background fluid
G	Gravity
I	Image
i	Initial
j	Jet
m	Mass
n	Number
$norm$	Normalized
p	Particle
q	Charge
R	Rayleigh
red	Reduced
ref	Reference
rel	Relative
RPA	Retarding potential analyzer
t	Tilt
TN	Tangential momentum
v	Viscous relaxation
w	Wall
x	Lateral x position coordinate
y	Lateral y position coordinate
z	Axial position coordinate

Variables

$\dot{m}$	Mass flowrate
ϵ_0	Permittivity of vacuum
γ	Ratio of specific heats
γ_{ST}	Surface tension
λ	Mean free path
$\mathbf{a}$	Acceleration
$\mathbf{c}$	Exit velocity
$\mathbf{E}$	Electric field

$\mathbf{F}$	Force	Kn	Knudsen number
$\mathbf{I_{sp}}$	Specific impulse	L	Characteristic length
$\mathbf{K}$	Mobility	l	Jet tip to reference point distance
$\mathbf{R}$	Position	LOS	Line-of-sight angle
$\mathbf{T}$	Thrust	m	Mass
$\mathbf{v}$	Velocity	Ma	Mach Number
μ	dynamic viscosity	n	Number of particles/samples
ν	kinematic viscosity	P	Pressure
ϕ	Azimuthal angle/Potential	p	Momentum/ Predicted Variable
ρ	Density	q	Charge
σ	Cross-section	R	Correlation
θ	Angle	r	Radius/Radial Position
ε	Emittance	Re	Reynold's number
a	Speed of sound	T	Temperature
b	Impact Parameter	t	Time
C	Coefficient	x'	Momentum Angle, x component
d	Diameter	x	Lateral x position coordinate
I	Current	y	Lateral y position coordinate
j	Mass flux	z	Axial position coordinate
k	Thermal conductivity		

ACKNOWLEDGMENTS

I would first like to thank my advisor, Professor Richard Wirz, for recognizing my potential and supporting my development as a researcher. Through both foreseen and unforeseen challenges in my graduate studies, you have been an understanding mentor. I would also like to express my thanks to my graduate committee members: Prof. Jeff Eldredge, Prof. Kunihiko Taira, Prof. Jacob Bortink, and Dr. David Bilyeu. My interest in and understanding for my research grew through your lessons, and beyond the classroom and the laboratory, you have been kind to and inspired me.

I would also like to thank those who mentored me at the Air Force Research Laboratories during my NDSEG Fellowship: Dr. Justin Koo, Dr. Daniel Eckhardt, and Dr. Rob Martin. Under your mentorship, my dissertation research was bettered by peer review long before it was submitted for publication. I am further grateful to my mentors in electrospray research at NASA JPL, Dr. John Ziemer and Dr. Colleen Marrese-Reading. Your passion inspires me even more than your incredible research, and I am thankful for your encouragement of my contribution to our field. To the fellow women in electric propulsion, it is an honor to count myself among you. Thank you to Dr. Elaine Petro, Dr. Deborah Levin, Dr. Colleen Marrese-Reading, and Dr. Kristina Lemmer for being visible beacons of success in our field. I have been thankful for your presence every time we've been in the same room, and your words of encouragement have carried their weight tenfold.

I am thankful to have completed this PhD in a laboratory full of people I can not only call incredible scientific researchers, but also dear friends. Mary Konopliv, Peter Wright, Ani Thuppul, Henry Huh, Nolan Uchizono, Adam Collins, Gary Li, Stephen Samples, Patrick Crandall, Graeme Sabiston, Shehan Parmar, Gary Wan, Rich Obenchain, Blake Haist, and Luke Franz – I'm so thankful this experience was one shared with you guys. To the electrospray team members – when I think of this PhD, I will think of our row in the cubicles, complete with kitchen stall and custom calendar. I am thankful to have experienced the free-dom of graduate school before Covid with you, and to have relied on each other amidst the difficulties of the pandemic. I'm also thankful that the Plasma, Energy & Space Propulsion Laboratory has grown to include members at Oregon State University, because this growth has brought friendships and new ideas which have enriched this dissertation.

Finally, I express my deepest thanks to my family. Thank you to my husband, Jeremy Breddan, for supporting me through all the highs and lows of this PhD. The joy that you bring into my life makes doing the hard things possible. Thank you to my mother, Laurie Davis. I am who I am for your efforts. You have seen every paper, presentation, and draft of this dissertation, supported me through every frustrated phone call, and celebrated every accomplishment. In all I do, I hope to make you proud. To my brothers, the responsibility of being your older sister has always inspired me to be my best. To Conor, your tenacity for defining and pursuing success for yourself inspires me to do the same. To Evan, your kindness has lifted me up when I need it most. To my father, Malcolm Davis, I am thankful for my earliest inspirations to pursue aerospace engineering. Camping trips looking at the stars of the Arizona desert are my earliest memory of love for the cosmos. Finally, to my grandmothers, Joy Ott and Laura Davis, I am thankful for your love and nurturing. I have always felt myself to be part of a lineage of incredible women, and I am thankful to you for establishing that.

I dedicate this dissertation to my grandfathers, Dr. Donald Eugene Ott and Dr. Frederick Gerald Davis. Your love of learning, encouragement of my education from an early age, and support for my graduate pursuits have been my inspiration to continue through the most challenging parts of earning my PhD. Through all my successes in life, I will think of you.

This dissertation research was supported by the National Defense Science and Engineering Graduate Fellowship (No. 7275700478), NASA Jet Propulsion Laboratory (Award No. 1580267), and the Air Force Office of Scientific Research (Award No. FA9550-21-1-0067).

VITA

2015	Research Intern, Physics Department, Rhodes College
2016	Research Assistant, Astrophysics Department, University of Birmingham
2016	Budapest Semesters in Mathematics Program
2017	Engineering Intern, American Pan
2017	Goldwater Scholarship
2018	B.S. (Physics, Mathematics), Rhodes College
2018 - 2019	UCLA Mechanical and Aerospace Engineering First-Year Research Fellow
2019	NASA Space Grant
2019 - 2021	National Defense Science and Engineering Graduate Fellow
2020 - 2024	Graduate Writing Consultant, UCLA
2021	M.S. (Aerospace Engineering), UCLA
2021 - 2024	Graduate Research Student, Plasma, Energy & Space Propulsion Laboratory, UCLA

PUBLICATIONS

Electrospray Plume Evolution: Background Pressure Influence. M. J. D. Breddan and R. E. Wirz. Journal of Aerosol Science, (2024), Accepted with Revisions.

Machine Learning Electrospray Plume Dynamics. M. J. D. Breddan and R. E. Wirz. Engineering Applications of Artificial Intelligence 133 D, 108095 (2024).

Electrospray Plume Evolution: Influence of Drag. M. J. D. Breddan and R. E. Wirz. Journal of Aerosol Science 167, 106079 (2023).

Multi-spatial-mode effects in squeezed-light-enhanced interferometric gravitational wave detectors. Daniel Töyrä, Daniel D. Brown, McKenna Davis, Shicong Song, Alex Wormald, Jan Harms, Haixing Miao, and Andreas Freise, Physical Review D 96, 022006 (2017).

Exact and Approximate Capacitance and Force Expressions for the Electrostatic Interaction Between Two Equal-Sized Charged Conducting Spheres. Shubho Banerjee, Mason Levy, McKenna Davis, and Blake Wilkerson, IEEE Transactions on Industry Applications 53 (3), 2455 - 2460 (2017).

CHAPTER 1

Introduction

This section begins with a brief introduction to spacecraft propulsion before presenting the governing physics and current state in spacecraft propulsion of the electrospray. This section concludes with the motivation, objective, hypothesis, and approach of this dissertation, and an outline of the dissertation manuscript.

1.1 Spacecraft Propulsion

In keeping with Newton's First Law of Motion, spacecraft at rest will remain at rest unless an unbalanced force is exerted upon them. Space is full of unbalanced forces, from gravitational and electromagnetic fields, to thermal and solar radiation pressures. Many missions utilize these existing forces to propel spacecraft towards a designated location, but the set of trajectories traversable under only these forces is limited. Therefore, many missions require additional onboard propulsion systems to augment or counter existing forces and access a much wider range of space.

Moving through space requires changes in spacecraft momentum. To generalize trajectory planning for any mass of spacecraft, aerospace engineers utilize the metric of change in velocity, or mass specific change in moment:

$$\Delta \mathbf{v} = g\mathbf{I_{sp}} \ln\left(\frac{m_i}{m_f}\right), \tag{1.1}$$

where g is the gravitational acceleration constant on Earth, m_i is the spacecraft initial mass with all propellant, m_f is the spacecraft final mass after expelling all propellant, and $\mathbf{I_{sp}}$ is the specific impulse of the spacecraft, defined as

$$\mathbf{I_{sp}} = \frac{\mathbf{T}}{g\dot{m}}, \tag{1.2}$$

where $\dot{m}$ is propellant mass flowrate and thrust is

$$\mathbf{T} = \mathbf{c}\dot{m}, \tag{1.3}$$

where $\mathbf{c}$ is the propellant exit velocity.

Propulsion systems which control spacecraft orientation by providing changes in angular momentum are termed attitude control systems. Examples include momentum wheels, control moment gyroscopes, and Vernier thrusters. Attitude control systems allow spacecraft to rotate and face targets for long periods of time, for purposes including weather tracking, satellite communication, military reconnaissance, and cosmological studies. Other propulsion systems provide changes in translation momentum, allowing spacecraft to move through space. There are discrete classes of such spacecraft propulsive devices: thermal, chemical and electrical.

Thermal propulsion systems create momentum by releasing heated or pressurized propellants. Examples include cold gas thrusters, steam rockets, and nuclear thermal rockets. In these systems, propellant is pressurized or heated prior to entering the thruster system, unlike in chemical propulsion in which propellant is heated and gains momentum in a chemical reaction such as fuel oxidation. Chemical propulsion systems can utilize solid fuel, as exemplified by the earliest rockets, 13th century Chinese gunpowder rockets. They can also utilize liquid propellants, as demonstrated by the Space Shuttle used to place humans in orbit. Liquid-propellant rockets are further divided into monopropellant, bi-propellant, and tripropellant systems, which use one, two, and three types of propellant, respectively, in the same system. Hybrid rockets combine solid-fuel and liquid-propellant propulsion systems, using a solid propellant in the combustion chamber and adding a liquid or gas oxidizer for combustion. Chemical propulsion systems are heralded for the high thrust to mass ratios they achieve by accelerating heavy particles (yielding high thrust through high mass flowrate following Eq. 1.3). However, such systems are specific impulse limited by the terminal velocity the heavy particles can reach in the chemical reaction, typically with $I_{sp} < 450\,\mathrm{s}$.

Electric propulsion systems create momentum by using electrostatic or electromagnetic fields to accelerate particles to higher velocities than is possible with chemical propulsion. These fields can be increased in strength as spacecraft power capabilities allow to exert more force on a particle than a chemical reaction can impart. Furthermore, the particles accelerated in electric propulsion systems can be as low-mass as single charges, such that they reach higher terminal velocities under the same applied force than heavy fuel particles in chemical propulsion systems. Electric propulsion systems include ion engines (famously featured in Star Wars) such as gridded ion engines and Hall effect thrusters, pulsed plasma thrusters, magneto plasma dynamic thrusters, resistojets, arcjets, air breathing electric propulsion systems, field emission electric propulsion thrusters, and electrospray thrusters. Electric propulsion systems are capable of delivering specific impulse on order of $10.000\,\mathrm{s}$. However, such systems are thrust-limited by the electric power available on the spacecraft. Low-mass propellant corresponds to a lower mass flowrate, and propellant exit velocity depends on the field strength created with available electric power, so thrust also depends on available power following Eq. 1.3. The thrust provided by electric propulsion systems is often lower than in chemical propulsion systems where heavy fuel provides a higher mass flowrate. The thrust and specific impulse capabilities of various thermal, chemical, and electric propulsion systems are shown in Fig. 1.1.

The choice of propulsion systems depends on mission objectives. Chemical propulsion systems are the default choice for liftoff systems because they provide sufficient thrust to exit the Earth's atmosphere. Electric propulsion systems utilize charged gaseous propellants much lighter than the fuel and oxidizers required for chemical propulsion, enabling a range of propellant-limited missions from multi-year deep space missions, which must preserve propellant to ensure long operational lifetimes, to Lower Earth Orbit (LEO) CubeSat endeavors,

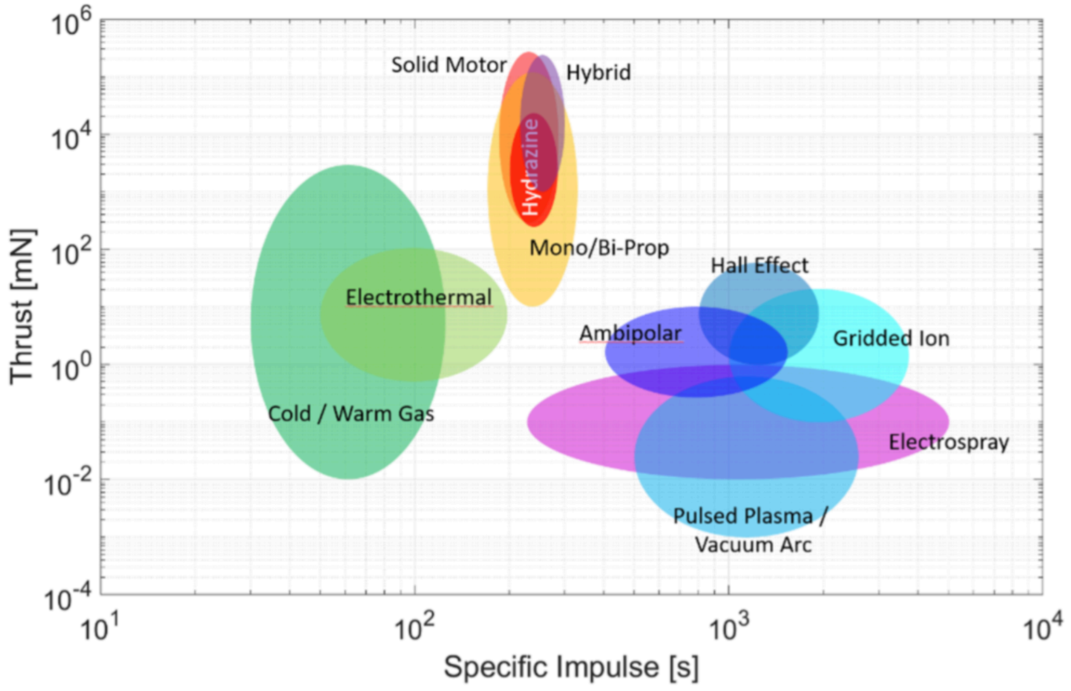


Figure 1.1: Thrust vs. specific impulse for several common spacecraft propulsion systems[1].

which are size-limited in the propellant quantity they can carry. The number of launched CubeSats has increased exponentially since the technology's inception as shown in Fig. 1.2, signaling an increased need for electric propulsion technologies in the future.

1.2 Electrospray Plume Physics

The term electrospray is used to refer to both a) the physical phenomenon of an electrified jet forming and releasing a fine aerosol in response to a strong electric field and b) the apparatus which deliberately applies an electric field to a fluid meniscus to prompt this physical phenomenon. This dissertation will utilize the second definition by default, although many statements herein will be true for both definitions. Electrospray will also be used as an adjective, such as in the cases of 'electrospray thruster' and 'electrospray ionization.'

Electrosprays create an aerosol from a liquid meniscus through the application of a

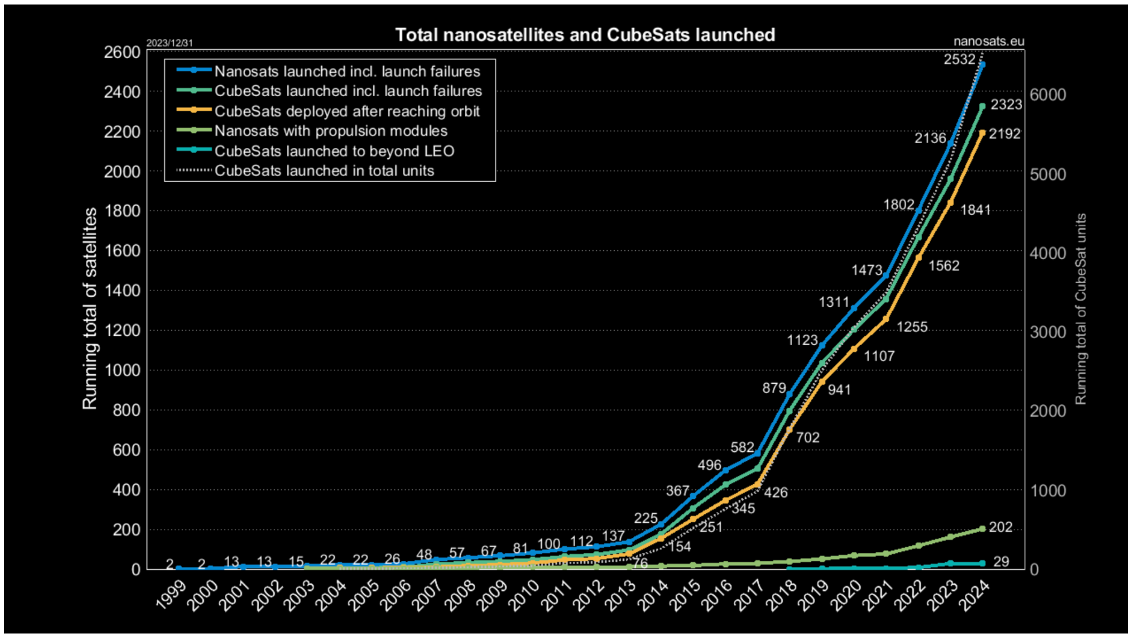


Figure 1.2: Total nanosatellites and CubeSats launched in recent years demonstrates exponential increase trend in CubeSat launches.[2].

strong electric field which electrohydrodynamically deforms the meniscus into one or multiple cone-jets which emit charged particles (droplets, ions, and/or ionic clusters). These like-charged emitted species repel one another through Coulomb interactions, evolving into a 3-dimensional plume over time. The means of supplying liquid to be sprayed can be active, such as applying pressure to move fluid through a capillary needle, or passive, such as through the innate capillary forces in a porous mesh. The electrostatic potential difference which induces fluid motion is created between the emitter and a downstream electrode. In many electrospray applications, such as mass spectrometry[25, 26] the single downstream electrode is a solid collector plate. In thrust-producing electrospray applications, there are one or more downstream electrodes commonly termed ‘grids’ which contain thrust-releasing apertures. An electrospray geometry with two downstream electrodes with thrust-releasing apertures is visualized in Fig. 1.3.

Electrospray plume particle populations have been observed to vary from nearly homo-

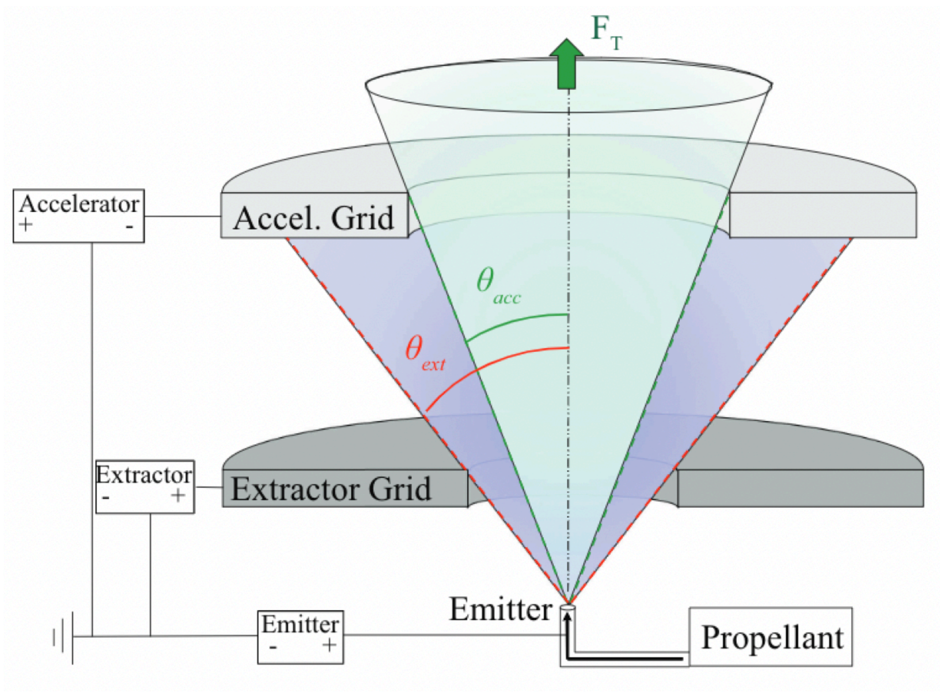


Figure 1.3: Electrospray geometry for a thrust-producing application. The electric potential difference is generated between the emitter and downstream electrode, which has an aperture to allow for thrust release. [3]

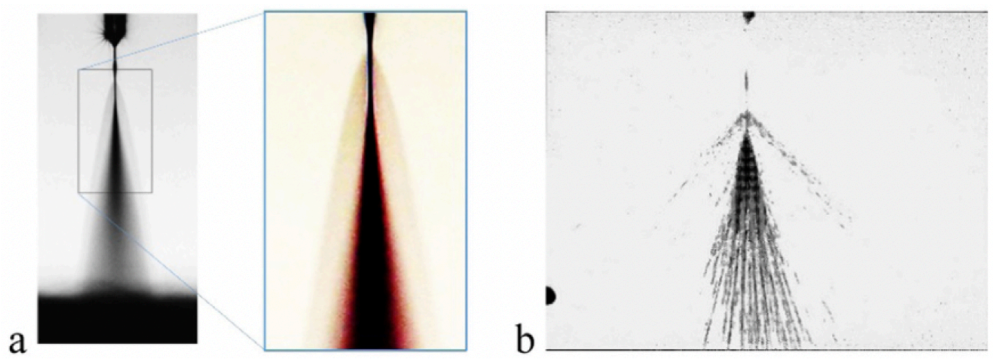


Figure 1.4: Cloud plumes of small secondary particles are separated from central primary particle plumes in bi-modal inhomogeneous plumes.[4]

geneous in species to highly inhomogeneous, containing particle with a wide range of size, mass, and charge. In plumes with pronounced bi-modality in the emitted particle population, there is marked species separation in the plume as shown in Fig. 1.4. At lower angles the plume consists of larger ‘primary’ particles, at higher plume angles there is a ‘cloud’ plume of smaller ‘secondary’ particles, and at middle angles there is sometimes a ‘dark zone’ with low particle density.

The mechanisms of secondary particle production which yield inhomogeneous plumes occur both during and after particle emission from the electrified jet. These means of secondary production, depicted in Fig. 1.5, are:

- Emission of minute ‘satellite’ particles from the jet between larger primary particles.
- Electric field-induced ion emission the fluid meniscus when the surface electric field magnitude surpasses a fluid-dependent threshold value on the order of $E = 1\,\mathrm{V\,nm^{-1}}$, typically at the ‘neck’ of the cone jet where curvature and thereby electric field strength are maximized[27].
- Electric field-induced ion emission from the surface of primary particles after emission if the surface electric field magnitude surpasses a fluid-dependent threshold value on the order of $E = 1\,\mathrm{V\,nm^{-1}}$ [27].

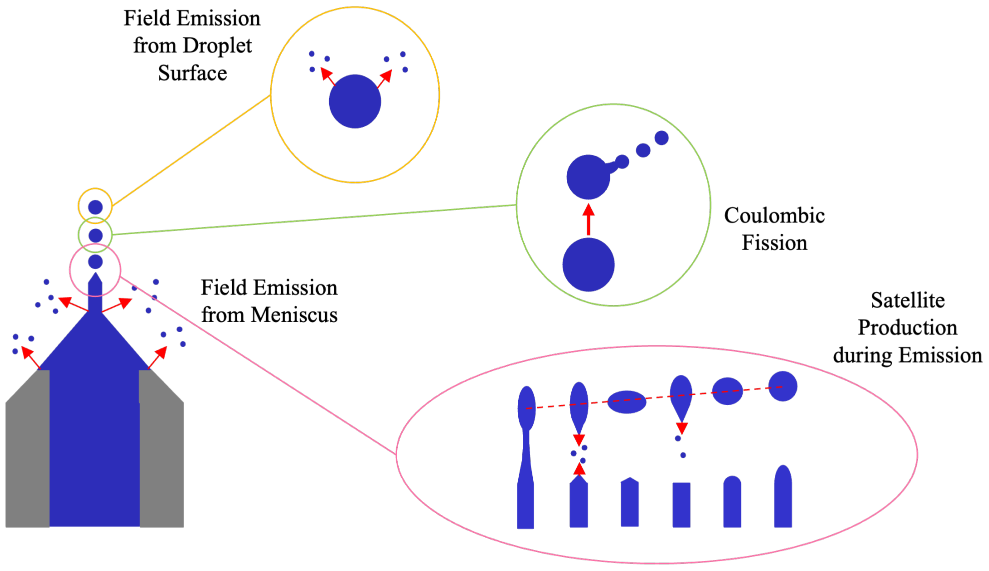

Field Emission from Droplet Surface
Coulombic Fission
Field Emission from Meniscus
Satellite Production during Emission

Figure 1.5: Sources of secondary particle production.

- Coulomb fission (also called Coulomb explosion) of a primary particle emitted with charge higher than its Rayleigh limit[28]:

$$q_R = \pi\sqrt{8\gamma_{ST}\epsilon_0 d^3}, \tag{1.4}$$

  where γ_{ST} is surface tension, d is particle diameter, and ϵ_0 is the permittivity of free space. Electrospray particles have been reported to fission with charges as low as 60% of the Rayleigh limit because their shapes deform from the spherical ideal.

- Electric field-induced ion emission or Coulomb fission of a particle created by the coalescence of multiple primary particles following emission which meets the above thresholds for secondary particle production.

The time required for ion emission is shorter than that for a Coulomb fission such that particles preferentially field-emit ions. Field-emission of ions can suppress Coulomb fission in a particle emitted over its Rayleigh limit by rapidly charge-reducing the particle to below its Rayleigh limit.

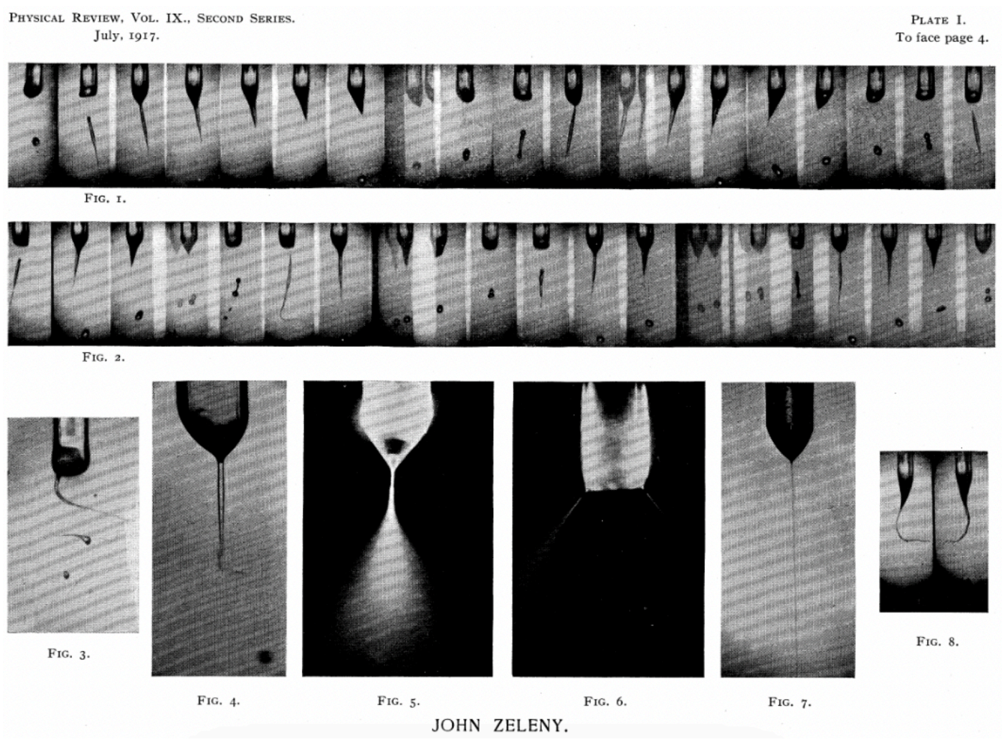


Figure 1.6: The first published images of the electrospray [5].

Electrosprays (the physical phenomenon) were first theorized by Lord Rayleigh when he derived the Rayleigh limit, who hypothesized that spherical fluid particles could reach such a charge threshold before throwing out fine jets of liquid. The first images of an electrospray were published by John Zeleny in 1917[5], presented in Fig. 1.6, beginning over a century of research into the many behavioral/operational modes of electrosprays. Primary modes which have been identified include spindle, pulsating, whipping, multi-jet, and the optimal cone-jet mode[4]. Electrospray modes have been experimentally discretized as functions of flowrate and applied voltage as displayed in Fig. 1.7. In the cone-jet mode, a single jet emits particles within a small displacement range of the axis of emission. The cone formed by the fluid meniscus was studied by Sir Geoffrey Ingram Taylor in the 1960s, who found that it has a half angle of 49.3° in the optimal cone-jet mode, a geometry now know as the Taylor cone[29].

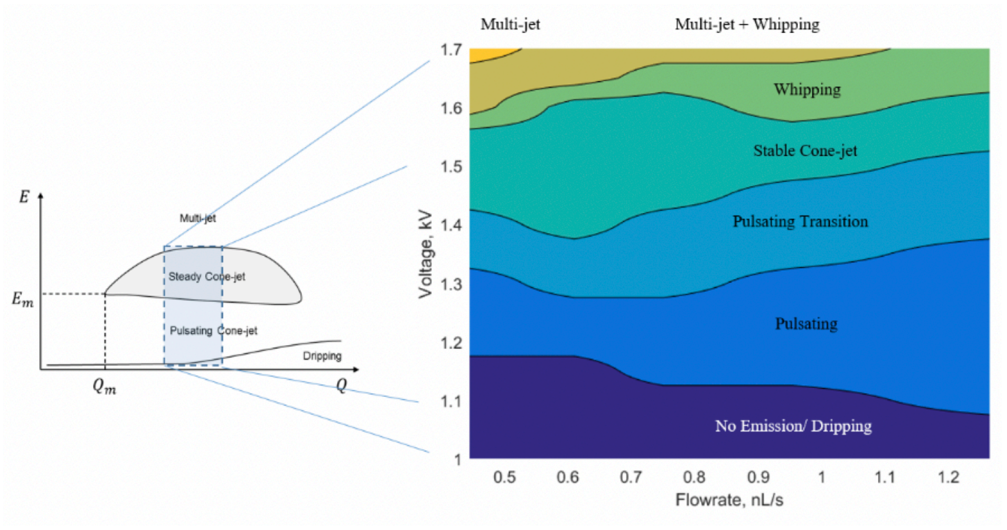


Figure 1.7: Electrospray behavioral modes experimentally mapped over a range of flowrate and voltage [6].

1.3 Electrospray Spacecraft Propulsion

Applications of the electrospray have developed simultaneously alongside efforts to visualize and describe the electrospray. Once a niche scientific phenomenon, electrosprays have found use in ink-jet[30, 31] and 3D printers[32, 33]; drug delivery systems[34]; automotive[35], agricultural[36, 37], cleaning[38, 39, 40], and fire-fighting sprays[41, 42, 43]; and mass spectrometry[25], for which John Fenn and Koichi Tanaka were awarded the Nobel Prize in Chemistry in 2002. Electrosprays are also applicable for spacecraft propulsion, having demonstrated successful in-space operation in 2015 on the Space Technology 7 Disturbance Reduction System (ST7-DRS), a National Aeronautics and Space Administration (NASA) technology demonstration payload of the European Space Agency (ESA) Laser Interferometer Space Antenna (LISA) Pathfinder (LPF) Mission [44].

The LISA Mission aims to observe the gravitational waves predicted by Einstein's Theory of General Relativity without noise from vibrations in the crust of the Earth, and the LISA Pathfinder spacecraft demonstrated the several key technology components for the future

LISA Mission. Figure 1.8 shows an expanded view of the LISA Pathfinder spacecraft components. This mission used a bi-propellant chemical propulsion system to reach the Lagrange 1 point in space, which then detached ('Propulsion Module' in Fig. 1.8). Once stationed, the spacecraft used electrospray propulsion systems ('Electric propulsion' in Fig. 1.8) for high-resolution thrust stabilization during gravitational-wave observation. These electrospray thruster devices were developed by Busek and are named the Colloid MicroNewton Thrusters (CMNTs) after their micro-Newton thrust precision capabilities. When the CMNT Disturbance Reduction Systems (DRS) were active, noise was comparable to the diameter of a DNA Helix (2 nm). One of the of the CMNTs is displayed in the inset in Fig. 1.8, showing four arms extruding from a main body containing the power source, among other components. Each arm ends with a gridded array, and each hole in the grid is the thrust-releasing aperture for a single electrospray. This gridded electrospray thruster structure is highly scalable because the grid size can be expanded to meet mission requirements.

1.3.1 Electrospray Divergence Considerations

The LISA mission requires operational thruster lifetime an order of magnitude greater than the 2,400 hours demonstrated by the LISA Pathfinder DRS thrusters. While performance enhancements are always desirable, operational lifetime is the metric limiting the use of electrospray thrusters for propulsion in current and upcoming spacecraft missions. Figure 1.9 presents a hierarchical failure tree of electrospray life-limiting mechanisms. Each first-tier failure mechanism is inherent to electrospray operation in a thrust-releasing system: overspray describes plume constituents which reach too wide of angles to pass through the thrust-releasing aperture, backstreaming electrons are generated through overspray and then guided by the applied electric field, and electrochemical interactions are induced between such backstreaming electrons and emitted particles. From this failure tree, overspray has been identified as the primary life-limiting mechanism, presented in Fig. 1.10. Particles which reach too wide of angles to exit through the thrust-releasing aperture instead impinge on

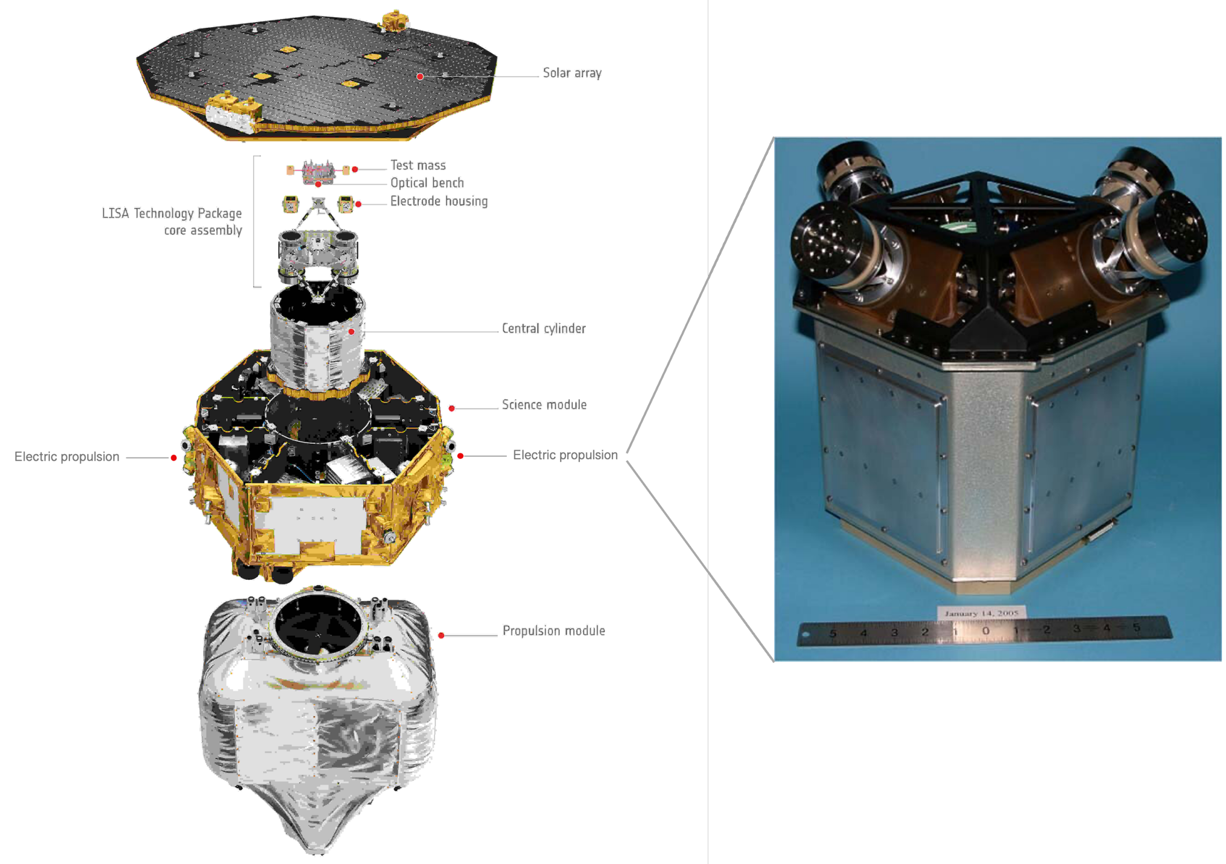


Figure 1.8: An expanded view of the LISA Pathfinder spacecraft, with ‘Electric propulsion’ labels added to the original figure from [7]. The insert figure shows one of the Colloid MicroNewton Thrusters providing electrospray propulsion [8].

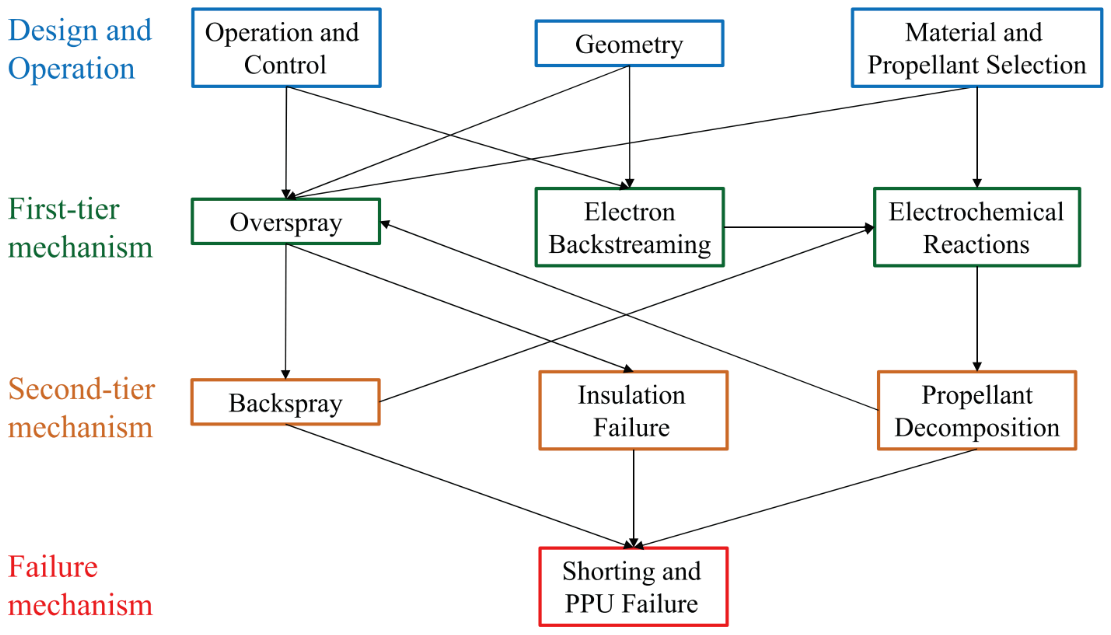


Figure 1.9: A hierarchical tree of life-limiting/failure mechanisms for electrospray thrusters.[3]

the downstream electrodes critical in electrostatic thrust generation. These fluid particles are absorbed by the electrode until it exceeds its volumetric absorption threshold, triggering propellant backspray towards the emitter and electrical shorting. Over multi-year periods of thruster operation, even a minute percentage of emitted flux incident on downstream electrodes will cause thruster failure through these overspray mechanisms.

Following the overspray life-limiting mechanism, the widest angles of the plume with the lowest particle density are most critical to electrospray thruster lifetime extension efforts. This key criteria is dissimilar to many other electrospray applications, such as printing and drug delivery, which are concerned with the central majority of the plume rather than its edges. Experimental measurements at wide plume angles are difficult to obtain and prone to high uncertainty due to the relative lack of particle flux to such angles. Therefore, electrospray plume modeling is needed to provide further insight towards electrospray plume structure at wide angles. Experimental and computational studies work with synergy on

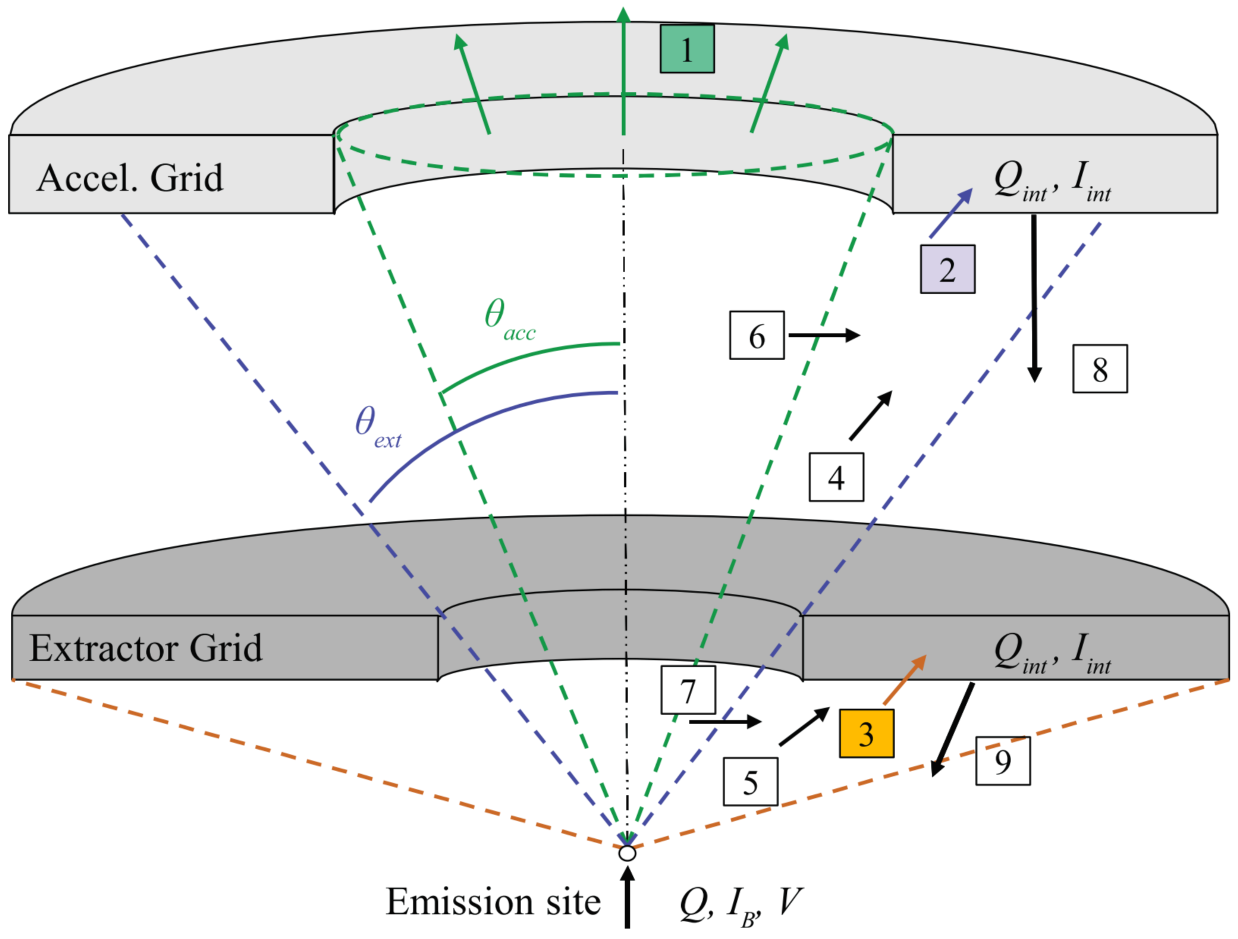


Figure 1.10: Particles optimally exit the thruster and produce thrust (1); however, overspray to the first ‘extractor’ grid (3) and a secondary downstream ‘accelerator’ grid (2) occurs from particles displaced to wide plume angles, (5) and (4) respectively, by Coulomb interactions, represented by (7) and (6) respectively. The two grids will eventually become saturated and backspray upstream, (9) and (8) respectively[3].

electrospray thruster designs to optimize thruster lifetimes for future of missions of interest.

The operational lifetime of an electrospray thruster before overspray fully saturates a downstream electrode can be analytically estimated given a functional form of the mass flux, j, in the plume, such as a Gaussian distribution of mass flux over plume angles θ:

$$j(\theta) = \exp\left(-\frac{\theta^2}{2\sigma_m^2} \right), \tag{1.5}$$

where σ_m is the standard deviation of the mass flux distribution[3]. Non-Gaussian mass flux distributions have also been reported, such as Super-Gaussian distributions[45, 23]:

$$j(\theta) = A(I_B) \exp\left(-\left(\frac{(\theta - \theta_t)^2}{2\sigma_m(I_B)^2} \right)^n \right), \tag{1.6}$$

where A is a scale factor which is a function of beam current I_B, θ_t is the tilt angle of the plume (0 in an ideal case), and n is the order of the super-Gaussian. The time for propellant overspray to saturate an electrode depends on many factors, such as the electrode aperture radius, the applied electric field, the emitter-to-electrode distance, and the open volume of the porous electrode material. The emitter aperture radius and emitter-to-electrode distance can be collectively represented using a line-of-sight angle, shown in Fig.1.11 and calculated by

$$LOS = \arctan\left(\frac{r_{el}}{z_{el}} \right), \tag{1.7}$$

where r_{el} is the radius of the electrode aperture and z_{el} is the axial coordinate of the electrode with the origin coordinate at the center of the emitter tip. Assuming Gaussian mass flux distributions, the impact of varying LOS and electrode open volume on time to saturate a downstream electrode for different standard deviations σ_m of mass flux distribution is illustrated in Fig. 1.12, reproduced from [3]. The best method for improving lifetime is beam confinement, moving left along the horizontal axis. However, changes to the electrode geometry dictating LOS and changes to the electrode material dictating porous capacity can also improve lifetime. The influence of changes to electrode geometry on electrospray plume divergence is further discussed in Chapter 3.

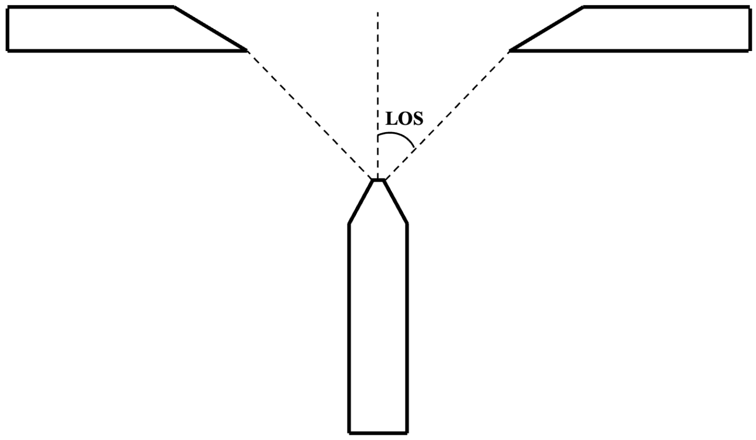


Figure 1.11: The line-of-sight angle between the emitter and a the aperture edged of a downstream electrode.

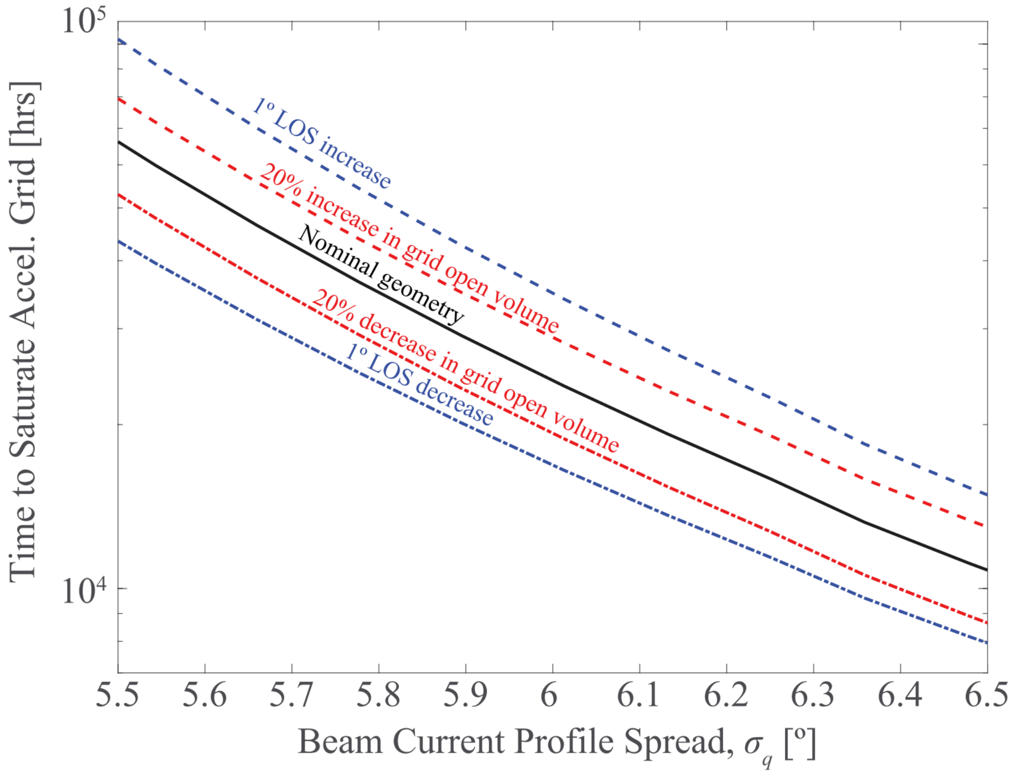


Figure 1.12: The effect of changes to electrode geometry (LOS), porous electrode capacity (open volume), and beam shape (standard deviation σ_m of Gaussian mass flux distribution) on electrode (in this case, an 'accelerator' grid) saturation time.[3]

In order to increase thruster lifetimes by confining the electrospray beam, or limiting the plume divergence, we must understand the sources of plume divergence and the mechanisms which grow divergence as the plume evolves. Furthermore, we must develop industry standards for characterizing plume divergence in order to measure the success of proposed means of beam confinement. These motivating factors and the objective of this dissertation research towards these needs are presented in the following section.

1.4 This Dissertation

The motivation for this dissertation research is to improve the lifetime of electrospray thrusters for spacecraft propulsion. In Sec. 1.3.1, we discussed that electrospray thruster lifetimes are primarily limited by propellant backspray which occurs as a result of propellant flux to downstream electrodes. The amount of propellant flux to downstream electrodes depends on the divergence of the electrospray plume. Therefore, the objective of this dissertation is to characterize the physics governing electrospray plume divergence and the resulting plume shape.

The electrospray plume is governed by three major forces: the electrostatic force from the potential difference between the emitter and downstream electrodes, the Coulomb force between particles, and the counter-motion drag force. Based on high speed video of electrospray particle dynamics presented in Sec. 4.1 which shows particles clustering prior to diverging, we hypothesize that Coulomb forces are the dominant force in electrospray plume expansion. Our approach to investigating this hypothesis is to computationally study the influence of each of these forces on electrospray plume evolution using a Lagrangian model of electrospray plume particle dynamics.

The outline of the dissertation is as follows. In chapter 1, we present introductions to spacecraft propulsion in general, electrospray physics, and electrospray spacecraft propulsion. This chapter concludes with the present outline of the dissertation manuscript. Chapter 2 presents the Discrete Electrospray Lagrangian Interaction (DELI) Model for simulating electrospray plume evolution, including validation of the model published in [46]. Chapters 3-6 discuss electrospray plume divergence physics: Chapter 3 focused on the applied electrostatic force from potential difference between electrodes, Chapter 4 on the Coulomb force between charged particles, Chapter 5 on the drag force, and Chapter 6 on the other forces not included in the DELI Model. Chapter 7 presents a new means of electrospray plume divergence analysis. Chapter 8 applies machine learning algorithms to particle evolution data produced by the DELI Model. Finally, Chapter 9 summarizes the major conclusions of this dissertation and presents future work which expands upon this dissertation research. The dissertation appendices present: empirical coefficient of drag terms used in DELI Model simulations (App. A), variable histograms for the machine learning studies (App. B), hyperparameter tuning information for the machine learning studies (App. C), plume evolution parameter studies (App. D), unpublished analytical efforts of dissertation research including nondimensionalization of the governing equation (App. E), and the code for the DELI Model in C++ (App. F).

CHAPTER 2

Electrospray Plume Modeling

This chapter presents the computational approach of this dissertation research to investigating electrospray plume evolution. It begins with a literature review of electrospray plume modeling efforts. Next, the end-to-end electrospray evolution model developed by the UCLA Plasma, Energy & Space Propulsion Laboratory is presented, followed by the introduction of the Discrete Electrospray Lagrangian Interaction (DELI) model developed during this dissertation, which is part of the end-to-end model. Finally, canonical verification and experimental validation of the DELI Model are presented.

2.1 Literature Review

The first simulation of electrospray plume evolution was developed by Ganan-Calvo et al. in 1994[9]. The authors utilized a Lagrangian approach and a governing equation with applied electrostatic, Coulomb, drag, and image charge forces to simulate the evolution of a single electrospray plume. This seminal work serves as the foundation for many other publications: Tang and Liu propagated the plume in the vacuum domain relevant to colloid thrusters[47]; Gamero-Castaño also simulated in vacuum, but used a line-of-charge approximation for the Coulomb term[48]; Deng and Gomez also used a line-of-charge approximation for space charge and also extended their simulations from single electrosprays to multi-plexed electrospray systems[49]; Yang et al. also simulated multi-plexed electrosprays, but on a Personal Super Computer with theoretical computational power of 10 TFlops[50]; Oh et al.[51] and Jung et al. studied dual or "twin" nozzle systems[52]; and Wilhelm et al. introduced a solvent evaporation module[10]. Grifoll and Rosell-Llompart have published extensively on simulating plume evolution[53, 11, 54, 55, 26]. They along with Arumugham-Achari investigated simulation timestep[55], Coulomb explosions[26], induced flow in the background gas, vapor concentration, gas temperature, and residual charges[54]. Some non-Lagrangian approaches to electrospray plume modeling have also been proposed: Higuera utilized an Eulerian approach to plume propagation[56], and Cui and Weng utilized a PIC approach to plume simulation, assuming the plume (in this case an ion beam) becomes Coulombically collisionless soon after emission[57].

All charged particles in an electrospray interact Coulombically, creating a computationally expensive n-body problem, with n^2 Coulomb force calculations in each time step, where n is the number of particles. Therefore, methods of approximating the Coulomb term have been proposed to increase computational efficiency. Early efforts to curtail the Coulomb term were made by Rietveld, who neglected all Coulomb forces[58], and Hartman et al., who restricted the the Coulomb term to consider only the closest 120 particles[59]. The line-of-charge approximation to the plume used by Gamero-Castaño[48] and Deng and Gomex[49] is another simple means of approximating the Coulomb term. This method was shown to accurately resolve the evolved plume outline, but misrepresent interior plume structure[49]. More sophisticated and accurate means of approximating space charge were developed by Grifoll and Rosell-Llompart, such as the such as the Lumped Space Charge[11] and Continuous Charge[53] methods.

In recent years, methods have been proposed which discretize the electrospray plume and treat independent regions differently. Gamero-Castaño[13] defined a radial threshold around emission, within which a fully discrete Coulomb term was used and outside of which Poisson's equation was solved in a manner similar to Grifoll and Rosell-Llompart's Continuous Charge method[53]. Similarly, Petro et al. defined two radial thresholds around emission: within the first 5 µm radially, the Coulomb force was calculated fully discretely in every timestep; between the first and second radial thresholds ($5\,\mu\text{m} < r \;\; 250\,\mu\text{m}$), the Coulomb force was calculated fully discretely but only updated every 10 timesteps; outside the second radial threshold ($r > 250\,\mu m$), Coulomb interactions were neglected [14]. Grifoll and Rosell-Llompart also proposed discretizing the electrospray plumes, but for the purpose of using different simulation timesteps in the different regions instead of different Coulomb force terms, termed the Zonal Time Stepping method [11]. Specifically, a smaller timestep was used closer to emission, where Coulomb interactions occur most frequently, than was used further downstream where particle density is lower.

2.2 End-to-End Model System

The electrospray is comprised of multiple scales of physics and is governed by balances of different forces in different regions. Therefore, it is best approached computationally with a multi-model system in which it is discretized into regions with different governing equations. These regions, each with a unique associated model, are displayed in Fig. 2.1. The Extraction Region, in which an ideally axisymmetric cone-jet forms and emits particles, is simulated with an electrohydrodynamic computational fluid dynamics (CFD) model[60]; analytical equations have also been developed for emitted particle properties based on jet properties[4]. In the UCLA PESPL, the model for this region was developed by Henry Huh during his dissertation research[61] and is named the Plasma & Space Propulsion Laboratory Electrohydrodynamic Model (PSPL-EHD). In the Transition Region, particle break-up and coalescence occurs following emission on very small scales (nanometer to micron order), necessitating molecular dynamics (MD) models. Research in this region was completed in the UCLA PESPL by Shehan Parmar during his thesis research[62]. In the Interaction Region, the plume has high particle density in the region following emission, and Coulomb interactions between particles dominantly govern plume dynamics. This region of the plume is the focus of this dissertation, and the model for this region is described in detail in the following Sec. 2.3. In the Plume Region, the plume has expanded enough that Coulomb interactions

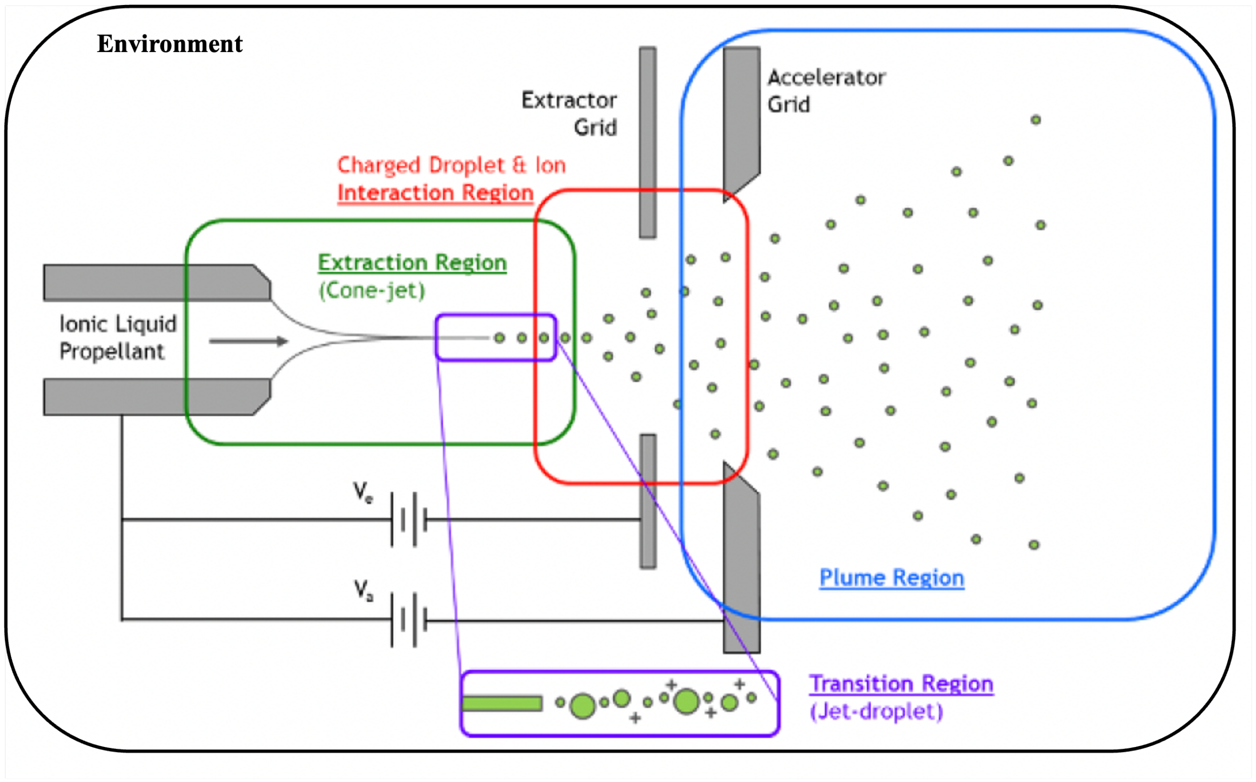


Figure 2.1: End-to-end model system for electrospray plume evolution consists of discretized regions based on governing physics.

no longer dominate particle dynamics, and the plume can be simulated with solely applied electrostatic forces. This process is completed in the PESPL using the commercial software COMSOL Multiphysics 5.1. Computational and analytical research in this region was completed by Shehan Parmar during his thesis research [63, 64]. The threshold between the Interaction Region and the Plume Region is discussed further in Sec. 4.4. Additional facility effects such as electron backstreaming were investigated through environmental models by Nolan Uchizono during his dissertation research [65, 66] and Jared Magnusson during his thesis research [67].

Information is passed between models to create an end-to-end model of the electrospray system, from jet formation, to particle emission, to plume evolution. The CFD model in the Emission Region PSPL-EHD model provides emitted particle data (charge, radius, mass, 3D location, 3D velocity) to the Transition Region MD model. The Interaction Region particle tracking model takes input particle data directly from the Emission Region model if breakup and coalescence effects are being excluded; otherwise, the Interaction Region model can take input particle data from the output of the Transition Region MD model. The particle data output by the Interaction Region particle tracking model is input to the Plume Region model, which in turn outputs particle dynamics data at any given downstream location. Information from environmental models, such as electron backstreaming populations, can be incorporated into each of the primary plume propagation models. For example, a backstreaming electron population can be introduced into the Interaction Region as a negative charge species emitted at the downstream end of the domain[66, 65]. All models are capable of operating independently of the others to study their specific plume regime given appropriate input data from any source, such as from the literature.

2.3 DELI Model

The Discrete Electrospray Lagrangian Interaction (DELI) Model was developed during this dissertation in order to simulate electrospray plume evolution. This model is part of the end-to-end model system introduced in Sec. 2.2 and covers the Interaction Region where Coulomb forces dominate plume evolution. While the electrospray plume modeling literature includes examples of both Lagrangian[9, 11, 68] and Eulerian[56] models, the DELI model was chosen to be Lagrangian in order to divulge the dynamics of individual particles which reach wide plume angles and detract from thruster lifetime. Furthermore, Eulerian methods can misrepresent the influence of inter-particle Coulomb forces in high charge density areas such as the Interaction Region. A fully discrete approach to Coulomb forces was chosen for the DELI model to optimize simulation accuracy despite this approach being computationally intensive. Study into the regions of the plume, potentially time-evolving, in which space charge can be approximated while preserving simulation accuracy is an active area of electrospray plume modeling research and is further discussion in Ch. 4.

This section will present the governing equations of the DELI model, the model algorithm for plume evolution including a flowchart representation, details of the time-stepping algorithm and time step size, and a description of the particle emission module. Following the introduction of the model, verification of the applied electrostatic module, and Coulomb module, and the time-stepping algorithm are presented. Finally, validation with results from the literature is provided for full electrospray plume evolution.

2.3.1 Governing Equations

The DELI Model utilizes the following governing force equation to simulate particle motion in the vacuum of space:

$$\begin{aligned} m\mathbf{a} &= q(\mathbf{E_A} + \mathbf{E_C}) \\ &= q\mathbf{E_A} + \frac{q}{4\pi\epsilon_0}\sum_{j=1}^{n}\frac{q_i\mathbf{r_i}}{|\mathbf{r_i}|^3} \end{aligned} \tag{2.1}$$

where m is particle mass, $\mathbf{a}$ is particle acceleration, q is particle charge, $\mathbf{E_A}$ is the electric field applied by electrodes, $\mathbf{E_C}$ is the the Coulomb field induced by other particles (space charge), ϵ_0 is the permittivity of vacuum, n is the total number of particles, and $\mathbf{r_i}$ is the separation vector from particle i. The 2D-axisymmetric static electric field applied by the electrodes $\mathbf{E_A}$ is obtained from COMSOL Multiphysics 5.1 by fitting a fine, adaptive mesh to the relevant emitter-electrode geometry and applying the relevant electric potentials. A Taylor cone with semi-angle 49.3° is staked to the emitter radial edge (outer radial edge in the case of a tapered emitter), and a jet with length 13.5 times the mean particle diameter[9, 53] and radius 1.89 times the mean particle radius is emitted from the tip of the cone. This jet-to-mean-particle radius ratio was first derived by Rayleigh for atmospheric, uncharged jet breakup[69, 28] and holds for steady cone-jet mode electrosprays of lower-conductivity propellants[70, 71, 72], although it has been challenged for electrosprays of high-conductivity propellants[73, 74, 75]. Both the Taylor cone and jet are held at the same voltage as the emission electrode in the

COMSOL model unless otherwise specified. By setting the emitted cone and jet to the high potential of the emitter, the repulsive force exerted on emitted particles from the cone and jet is included in the electric field force applied by the electrodes, $\mathbf{E_A}$, in Eq. 5.19 governing particle motion. The 2D-axisymmetric electrostatic field is imported from COMSOL to the DELI Model and interpolated onto 3D particle coordinates assuming axisymmetry. The interpolation utilizes the nearest-neighbor method, in which particles experience the electric field associated with the closest point on the imported $\mathbf{E_A}$ data mesh, to obtain the electric field acting on each particle in the plume.

When simulating atmospheric validation cases, a drag term is included in the particle governing equation. This drag term is empirical and dependent on emitted particle characteristics and properties of the background fluid. For incompressible, continuum flow, Stokes found the drag force on a sphere to be directly proportional to its diameter:

$$\mathbf{F_D} = -3\pi d \nu_{fl} \mathbf{v}, \tag{2.2}$$

where ν_{fl} is the kinematic viscosity of the surrounding fluid[17]. In more compressible or rarefied flow environments, the drag force on the particle departs from the Stokes solution:

$$\mathbf{F_D} = -\frac{\pi}{8} d^2 \nu_{fl} C_D \mathbf{v}|\mathbf{v}|, \tag{2.3}$$

where C_D is the coefficient of drag. Stokes analytically determined the coefficient of drag when the Reynolds number is $Re << 1$, where

$$Re = \frac{|\mathbf{v}| d}{\nu_{fl}}, \tag{2.4}$$

and the Knudsen number $Kn << 1$, where

$$Kn = l/d, \tag{2.5}$$

and

$$l = \frac{1}{\rho_n \sigma} \tag{2.6}$$

is the particle mean free path, ρ_n is the number density of the surrounding fluid particles, and σ is the cross-section of the surrounding particles, to be

$$C_D = 24/Re. \tag{2.7}$$

Otherwise, the coefficient of drag is determined empirically[16, 19, 76, 77, 78] over specific range of flow parameters Re, Kn, and Ma, where Ma is Mach Number:

$$Ma = \frac{|\mathbf{v}|}{a}. \tag{2.8}$$

Note that Knudsen, Mach, and Reynolds number are related by

$$Kn = \frac{Ma}{Re}\sqrt{\frac{\gamma\pi}{2}}, \tag{2.9}$$

where γ is the ratio of specific heats, such that knowing two flow parameters determines the third. The collection of empirical coefficients of drag utilized during dissertation research studies is given in Appendix A. With the addition of the drag term from Eq. 5.17, the governing equation becomes

$$\begin{aligned} m\mathbf{a} &= q(\mathbf{E_A} + \mathbf{E_C}) - \mathbf{F_D} \\ &= q\mathbf{E_A} + \frac{q}{4\pi\epsilon_0}\sum_{j=1}^{n}\frac{q_i\mathbf{r_i}}{|\mathbf{r_i}|^3} - C_D\frac{\pi}{8}\rho_{fl}d^2\mathbf{v}|\mathbf{v}|. \end{aligned} \tag{2.10}$$

When simulating non-thruster geometries with solid collector plate electrodes, rather than electrodes with thrust-releasing apertures, Eq. 2.10 is modified to include an image charge term which accelerates particles as they approach the collector plate:

$$\begin{aligned} m\mathbf{a} &= q(\mathbf{E_A} + \mathbf{E_C} + \mathbf{E_I}) - \mathbf{F_D} \\ &= q\mathbf{E_A} + \frac{q}{4\pi\epsilon_0}\sum_{i}^{n} q_i\left(\frac{\mathbf{r_i}}{|\mathbf{r_i}|^3} - \frac{\mathbf{r_{Ii}}}{|\mathbf{r_{Ii}}|^3}\right) - C_D\frac{\pi}{8}\rho_{fl}d^2\mathbf{v}|\mathbf{v}|, \end{aligned} \tag{2.11}$$

where $\mathbf{E_I}$ is the field due to image charge in the collector plate, and $\mathbf{r_I}_i$ is the separation vector from each image charge, which has axial component $2H - z_i$, where H is the collector plate height and z_i is the axial coordinate of the particle yielding the image. In the case that there is a downstream electrode with a thrust-releasing aperture, there will still be an image charge attraction induced by charged particles in the electrode, but it is more geometrically complex than for the parallel collector plate case and is currently simplfied to Eq.2.11 in the model. A diagram of the forces in Eq. 2.11 is presented in Fig. 2.2.

Particles are assumed to be perfectly spherical in drag force calculations. In Coulomb force calculations, they are treated as uniform spherical charges, with the high-order effects of motion of charge on the surface of fluid droplets excluded from the current model. Charge motion in fluid droplets and its influence on their Coulomb interactions has been researched in the literature [21, 79] and is discussed in Sec. 6.3. Brownian motion, gravitational, Boussinesq–Basset, and phoretic forces are many orders of magnitude weaker than the forces included in Eq. 2.11 are thus neglected from the governing equation of motion[9, 4]. The forces which are neglected from the governing equation of the DELI model are disucssed in Ch. 6. Secondary emission, whether through the field emission of ions from the surface of particles downstream of emission[80, 27], Coulomb repulsion[81, 26], or particle impacts on surfaces within the domain[67, 65, 82, 83], is not currently considered in the model and has potential for future work.

2.3.2 Numerical Algorithm

A flowchart of electrospray plume simulation in the DELI model is presented in Fig. 2.3. The model begins by importing a 2D-axisymmetric electric field solution from COMSOL, and creating a corresponding 3D simulation domain. In order to emit particles under flow constraints, the model inputs the density of the sprayed fluid, particle mass and charge distributions, particle emission location and velocity distributions, and current and/or mass flow rate constraints. The particle emission module is discussed further in discussed further in Sec. 2.3.4. After particle emission is constrained to meet flow inputs, all particles are advanced according to Eq. 2.11 as described in Sec. 2.3.1. Any particles which exit the simulation domain or strike an electrode are removed from the simulation following the

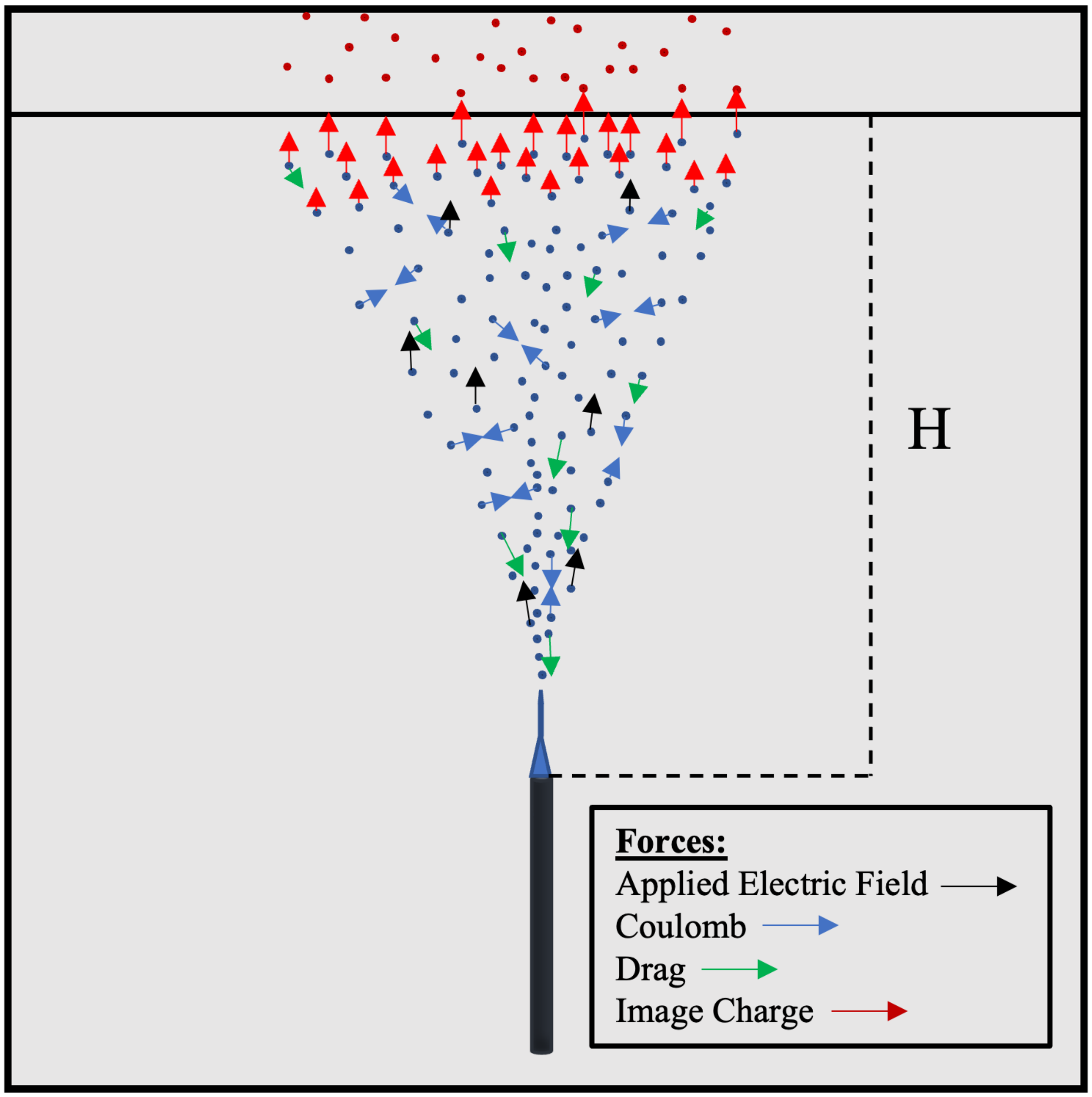


Figure 2.2: Electrospray domain with a solid collector plate, displaying examples of the applied electric field, Coulomb, drag, and image charge forces on emitted particles. Only the portion of image charges nearest the collector plate are shown.

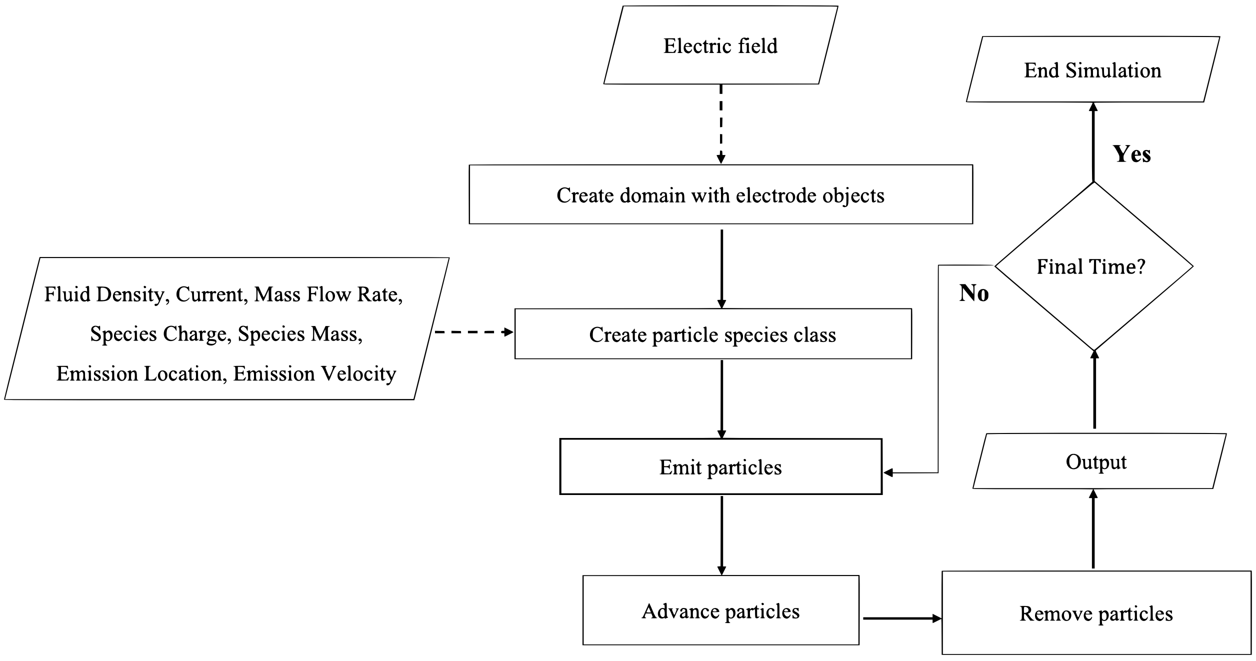

Electric field
End Simulation
Yes
Create domain with electrode objects
Final Time?
No
Fluid Density, Current, Mass Flow Rate, Species Charge, Species Mass, Emission Location, Emission Velocity
Create particle species class
Emit particles
Output
Advance particles
Remove particles

Figure 2.3: Algorithm for evolving electrospray plume in the Discrete Electrospray Lagrangian Interaction Model.

advancement step. Particle properties are output at a user-declared frequency. This iterative process of emitting particles to meet flow constraints, propagating them according to the governing Eq. 2.11, and removing them as they exit the simulation domain is repeated for a user-designated number of steps, after which the simulation is terminated.

The DELI model utilizes the first-order Velocity Verlet time-stepping algorithm[84] to advance particles:

$$\begin{aligned} \mathbf{R}^{t+1} &= \mathbf{R}^t + \mathbf{v}^t \Delta t + \frac{1}{2}\mathbf{a}^t \Delta t^2, \\ \mathbf{v}^{t+1} &= \mathbf{v}^t + \frac{1}{2}\Delta t(\mathbf{a}^t + \mathbf{a}^{t+1}), \end{aligned} \tag{2.12}$$

where $\mathbf{R}$ is the particle position vector, t is the time step index, and Δt is the simulation time step, discussed further in Sec. 2.3.3. The velocity used in Eq. 5.17 to calculate the drag force in each iteration is

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \mathbf{a}^t \Delta t. \tag{2.13}$$

This algorithm is common in the molecular dynamics community and has been shown to preserve energy in electrospray plume simulations[55]. This order of this time-stepping algorithm is verified in Sec. 2.5

2.3.3 Simulation Time Step

Simulated plume evolution accuracy is sensitive to simulation time step[55, 14]. Grifoll et al. reported that overly large time steps cause false 'physical collisions' between particles, in which the distance between particle centers is less than their summed radii[55]. In the absence of a coalescence module, these physical collisions are interpreted as high-magnitude Coulomb collisions which propel 'outlier particles' to inaccurately wide plume angles. Grifoll et al. therefore advised plume simulations utilize a time step for which the number of physical collisions in the plume has plateaued to a minimum, such that further decreasing the time step does not further decrease the number of collisions, in order to ensure any remaining collisions are genuine particle collision events in the plume rather than simulation artefacts.

The motivation in selecting a simulation time step for the DELI model is to capture high-frequency particle emission and interaction events. A sufficiently small time step is chosen to allow each emitted particle to move downstream for multiple timesteps before the next particle is emitted. The DELI simulation time step is a defined to be one hundredth of the mean particle emission period:

$$\Delta t \leq \frac{\overline{t_{em}}}{100}, \tag{2.14}$$

where the average particle emission period, $\overline{t_{em}}$, is the ratio of average mass, $\overline{m}$, to commanded mass flow rate, $\dot{m}$, or average charge, $\overline{q}$, to commanded current, I:

$$\overline{t_{em}} = \frac{\bar{m}}{\dot{m}} = \frac{\bar{q}}{I}. \tag{2.15}$$

The time step Δt in Eq. 2.14 is set to be one on the order of $\frac{\overline{t_{em}}}{100}$. For example, if $100\,\mu s \leq \overline{t_{em}} < 1000\,\mu s$, then $\Delta t = 1\,\mu s$. Once a time step is selected, it is ensured not to cause false physical collisions between electrospray particles. If physical collisions are observed during the simulation, a lower timestep is tested to see if the number of physical collisions can be reduced. The simulation time step is finalized once it is verified to create the lowest number of physical collisions.

2.3.4 Particle Emission

The DELI model emits particles given input distributions for diameter and charge - $P(d)$ and $P(q)$, respectively. The distributions may be discrete, or a single joint probability distribution $P(d, q)$, and can be varied to match different experimental or theoretical conditions. The emission module further requires an inputs for emission position $P(r_i, \theta_i, z_i)$, where $r_i, \theta_i,$ and z_i are cylindrical emission coordinates, or $P(x_i, y_i, z_i)$, where $x_i, y_i,$ and z_i are Cartesian emission coordinates; the distributions for the three emission coordinates may be discrete or joint. Previous evolution studies note that the resulting plume structure is not sensitive to the magnitude of the applied radial perturbation, Δr_i, so long as the magnitude range is less than or equal to the jet radius, r_j, corresponding to varicose instabilities inherent in the jet[59]. When the radial perturbation magnitude exceeds the jet radius, the perturbations correspond to a jet experiencing both kink and varicose instabilities, such that the simulated plume is no longer operating in the desired steady cone-jet mode[59]. Simulation results are sensitive to $P(\Delta z_i)$: too large a range in axial emitted coordinate allows particles to be emitted on top of other existing particles, creating a physical collision in the emission condition.

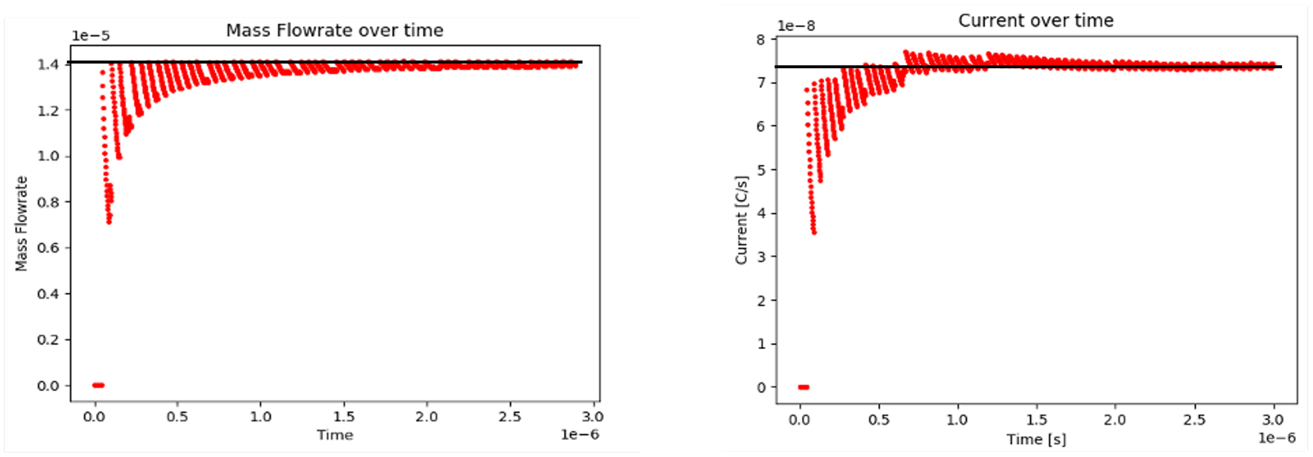
Electrospray particle emission location is determined by the electrohydrodynamic cone-jet from which the particles are released. Furthermore, the electrostatic force which propels particle downstream after emission is influenced by the changing cone-jet structure. Dynamically updating the applied electric field to reflects changes in cone-jet structure is computationally intensive and therefore has only been implemented in the Emission Region by the PSPL-EHD Model to simulate particle emission. The DELI model uses the electric field input from COMSOL as discussed in Sec. 2.3.1, and the particle axial emission coordinate is fixed at the jet tip.

Emitted mass flow rate and current are constrained in alternating emission steps, such that neither constraint is neglected in favor of the other. In each step, the emission module compares the last-emitted particle mass, m_{-1}, or last-emitted particle charge, q_{-1}, to the appropriate flow rate constraint - mass flowrate or charge, respectively - to calculate the associated particle emission period, t_{em}:

$$t_{em} = \begin{cases} \frac{m_{-1}}{\dot{m}}, & n \text{ is even} \\ \frac{q_{-1}}{I}, & n \text{ is odd} \end{cases}, \tag{2.16}$$

where n is the total number of emitted particles. When only mass flowrate is known, the particle emission period is a function of particle mass for every emission; when only current is known, the emission period is always a function of particle charge. If more simulation time has passed since the last particle emission than the particle emission period in Eq. 2.16, a new particle is emitted with properties from particle property input distributions. Figure 2.4 displays the DELI Model particle emission module matching flow constraints following startup of the flow.

Because the DELI model utilizes a constant time step, the total simulation time is always equal to a positive integer multiple of the discrete simulation time step $C\Delta t$, $C \in \mathbb{Z}$. In contrast, particle emission periods can vary over a continuous range. Therefore, t_{em} can be offset from the closest discrete simulation time step Δt by some small $\delta_t = |t - t_{em}| < \Delta t$. In this case, there is some small difference between the emitted flow and the targeted flow with mass $\delta_m = \delta t \dot{m}$ and charge $\delta_q = \delta t I$. Utilizing a time step that is a small fraction of t_{em}, such that $\delta t << t_{em}$ as described in Eq. 2.14, prevents this difference from being significant and compromising the efficacy of the emission module to conform to commanded flow constraints. Alternatively, the charge or mass difference between emitted flow and commanded flow following an emission event may be added on to the next-emitted particle, as done by Grifoll and Rosell-Llompart [11]. This method precludes any difference between



(a) (b)

Figure 2.4: The DELI model matches mass flowrate and current constraints over time following startup.

simulated and commanded flow emission, but assigns additional mass or charge to emitted particles beyond particle property input distributions.

2.4 Verification

The first verification tests performed on the Coulomb force component of particle propagation in the DELI model were canonical tests of symmetry, several of which are presented in Fig. 2.5. Coulomb forces between two interacting particles were observed to be symmetric, with differences between the force on two particles within the range of machine error, as shown in Fig. 2.6.

Following these canonical tests, the Coulomb force component of the DELI Model was further verified by comparing simulated angles of deflection in a canonical two-particle interaction with no applied electric field or drag to the analytical solutions for these deflection angles. The analytical solution is derived from the conservation of energy in the center-of-

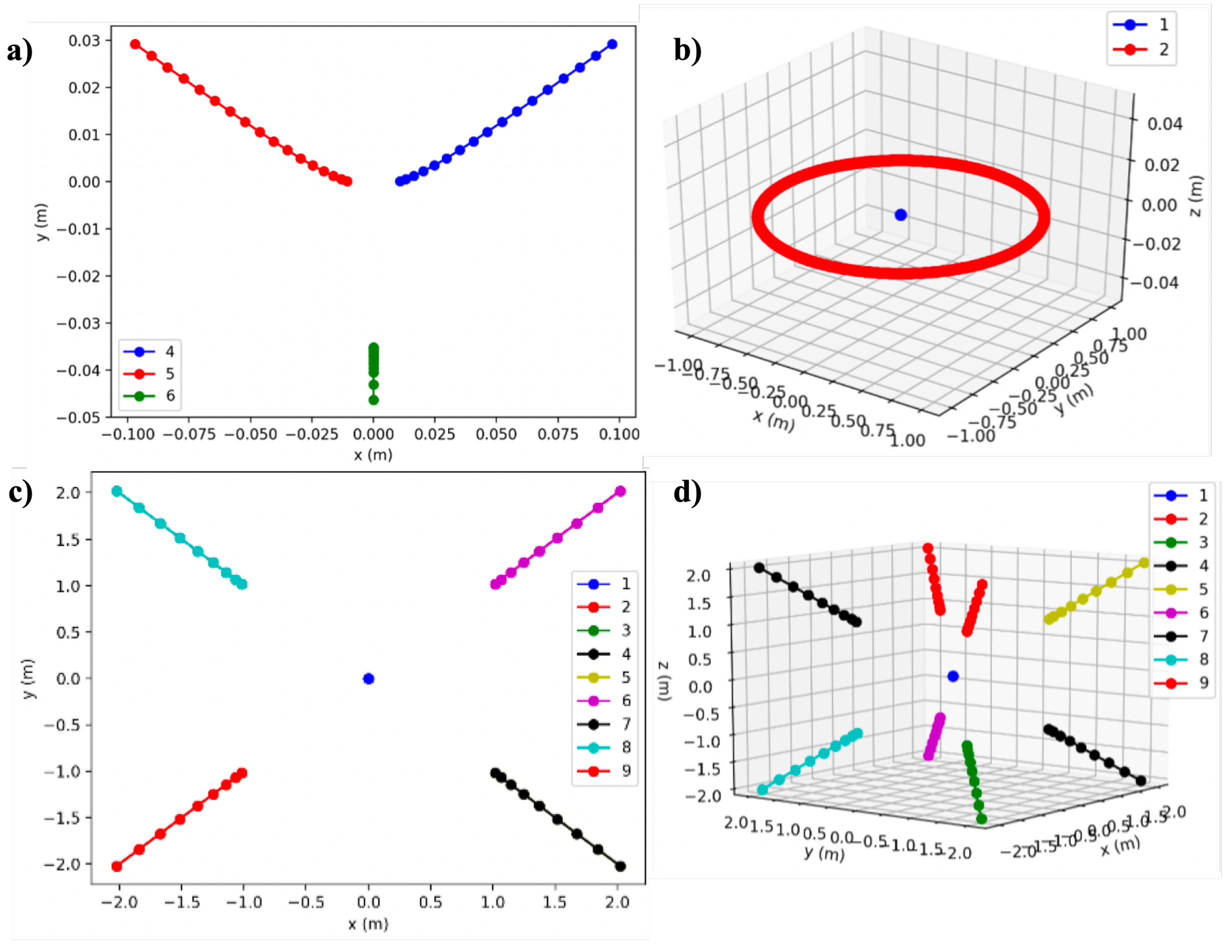


Figure 2.5: All particles have $q = 1\,\mathrm{C}$ and $m = 1\,\mathrm{kg}$ unless otherwise specified. a) Particle 6 approaches stationary particles 4 and 5, positioned 0.02m apart. b) Particle 5 orbits stationary particle 4 ($q = 1 \times 10^{11}\,\mathrm{C}$ and $m = 1 \times 10^{11}\,\mathrm{kg}$) with a velocity of $1\,\mathrm{m\,s^{-1}}$. c) 8 stationary particles initially at rest at the corners of a cube with side length 2 m and 1 particle in the center. d) 8 stationary particles are initially at rest at the corners of a cube with side length 2 m and 1 particle in the center.

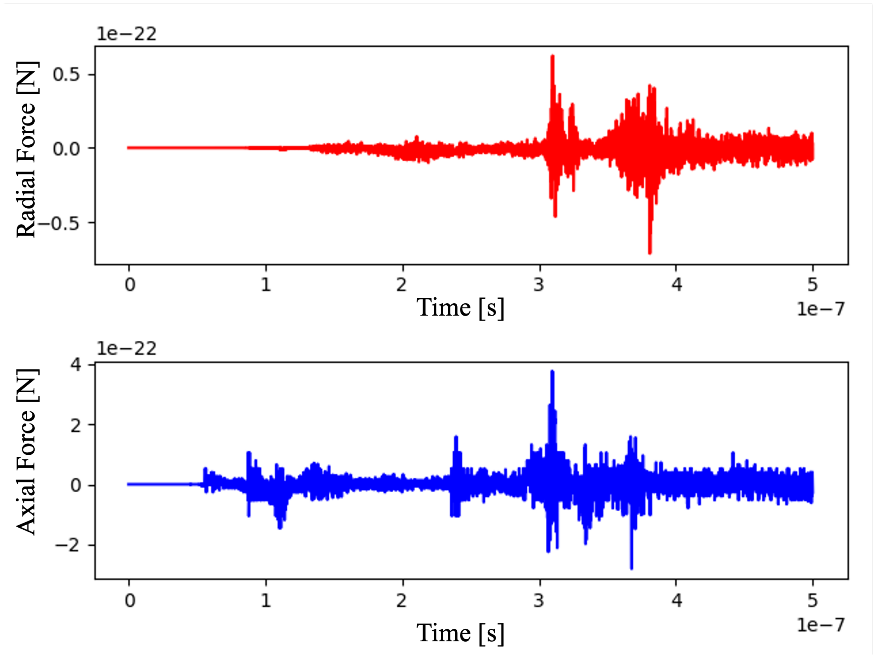


Figure 2.6: Difference in the radial and axial components of the Coulomb force on two interactions particles. The Coulomb force is exerted symmetrically, with differences in force on the two particles in the range of machine error.

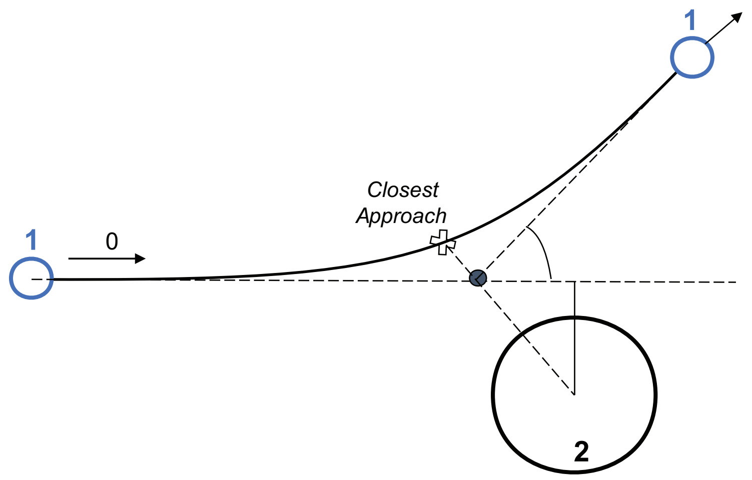


Figure 2.7: Particle 1 approaches particle 2 with relative velocity v_0 and impact parameter b. The deflection is shown from the frame of the reference of particle 2. The deflection angle θ of particle 1 from particle 2 is measured from the intersection of the line connecting the point of closest approach of particle 1 to particle 2 and the impact parameter axis line.

mass frame:

$$\tan\left(\frac{\theta}{2}\right) = \frac{q_1 q_2 b}{4\pi\epsilon_0 |\mathbf{v_{rel}}^2| m_{red}}, \tag{2.17}$$

where θ is the deflection angle in the center-of-mass frame, q_1 and q_2 are the charges of the two particles, b is the impact parameter (minimum separation between the particles if they passed without interacting), $\mathbf{v_{rel}} = \mathbf{v_1} - \mathbf{v_2}$ is the relative velocity (where $\mathbf{v_1}$ and $\mathbf{v_2}$ are the velocities of the two particles), and $m_{red} = \frac{m_1 m_2}{m_1+m_2}$ is the reduced mass of the system (where m_1 and m_2 are the masses of the two particles). This two particle deflection scenario is displayed in Fig. 2.7; in the presented case, $m_2 >> m_1$ such that the lab frame of view is very close to that of nearly stationary particle 2.

This deflection angle verification study was conducted with radius $r_1 = 1\,\mu\text{m}$ and $r_2 = 100\,\mu\text{m}$ particles, which are in the range of experimentally observed electrospray particle sizes[4, 9]. The study included three relative velocities: $v_0 = v^*, 2v^*,$ and $4v^*$, where $v^* = 5\,\text{m/s}$, such that all three velocities are on order of experimentally observed electrospray particle emission

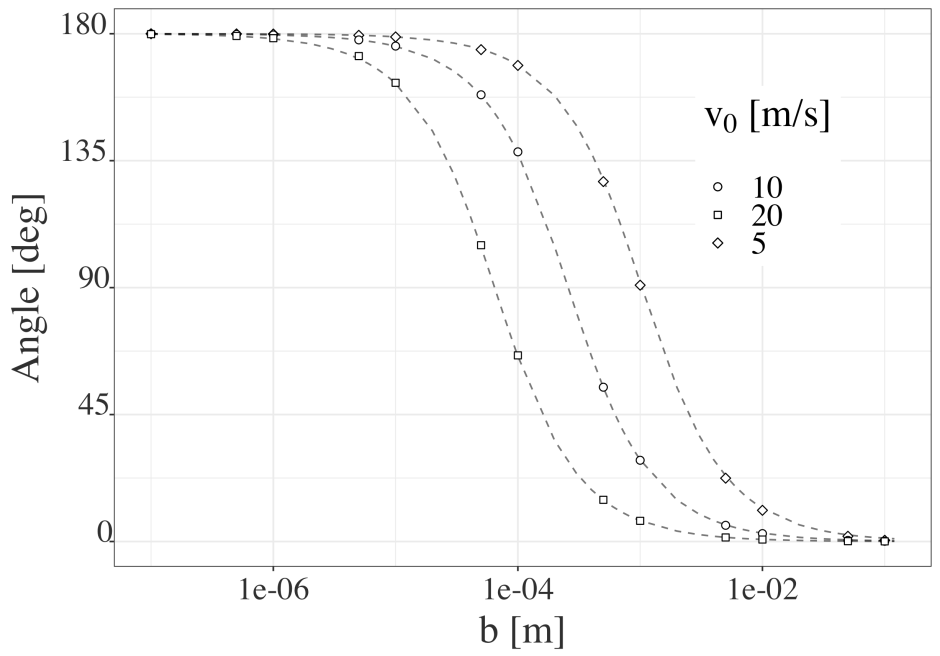


Figure 2.8: DELI deflection angle results for 2-particle Coulomb interactions are compared to analytical solutions (dashed lines).

velocities[9]. The verification study included impact parameters ranging from $b = 0.1\,\mu\text{m}$ to $b = 0.1\,\text{m}$ to verify Coulomb deflection angles from near 0° to almost 180°. DELI simulations with size-diverse plumes[68] include instances in which small particles are 'bounced' back and forth between larger particles, such that large-angle deflections near 180 degrees are anticipated. particles are eventually deflected through Coulomb interactions off the axis of emission at smaller plume angles ($< 40°$), such that lower angles of deflection are also anticipated[85, 23]. The results of the verification study are displayed in Fig. 2.8.

The DELI deflection angle results in Fig. 2.8 strong display agreement with the analytical solutions for all angles. In all cases, the absolute error of the DELI result is below 1° and the relative error is below 0.6 %. This verification study demonstrates the trustworthiness of the DELI model in simulating Coulomb interactions.

Two particle Coulomb interactions were also used to verify the order of the first-order Velocity Verlet time-stepping algorithm used to advance particles. The order of magnitude

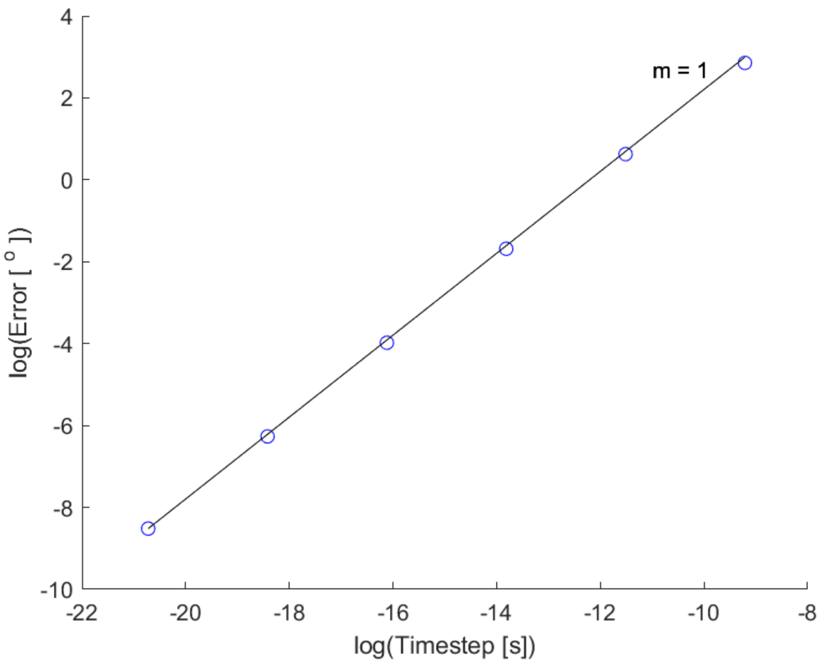


Figure 2.9: The error in deflection angle from a two particle Coulomb interaction is given for different orders of magnitude of simulation timestep. A line with slope 1 is shown for reference to confirm that the time-stepping algorithm is first-order.

of the time step was varied and the resulting error in deflection angle from the analytical solution was observed. The results are presented in Fig. 2.9. The slope of the error line is 0.996, very near 1, verifying that this scheme moves particles forward in time with first-order error, $O(\Delta t)$.

2.5 Validation

The DELI model has been validated against experimental measurements and simulation results from Gañán-Calvo et al. [9], as well as subsequent publications by Wilhelm et al.[10] and Grifoll and Rosell-Llompart [11] in which the authors validate their models against the same source case. While the DELI model is primarily motivated by electrosprays for spacecraft propulsion, which occurs in a vacuum environment, an atmospheric validation case was chosen for several reasons. Simulation in the atmospheric regime is less computationally intensive and includes less uncertainty in emitted particle data than does simulation in the vacuum regime; there is a wide variety of atmospheric particle data available in the literature, and atmospheric plumes generally contain fewer, larger, and slower particles than those in vacuum[4, 73]. Such atmospheric particles can be directly visualized optically through methods such as flash photography and Phase Doppler Anemometer (PDA), which cannot be applied to vacuum plumes comprised of particles with radii smaller than the diffraction limit[4]. Although atmospheric validation introduces the drag force to DELI simulations, it also includes all governing forces relevant to the vacuum regime.

The electrode geometry, applied voltage, flow rate constraints, and emitted species data for the validation case are taken from Sec. 3.4 *Experiment no. 2* of Gañán-Calvo et al. [9]: the needle radius is 0.5 mm, needle-to-collector-plate distance is 30 mm, and voltage applied to needle is $\Phi_0 = 5.2\,\mathrm{kV}$; volumetric flow rate is $Q = 2.4\times10^{-9}\,\mathrm{m^3/s}$ and current is $I = 45\,\mathrm{nA}$; sprayed fluid is liquid heptane with 0.4% of STADIS 450 with density $\rho = 685\,\mathrm{kg/m^3}$; constant emission velocity is $v_0 = 7.8\,\mathrm{m/s}$; the particle diameter distribution is Gaussian with mean $\bar{d} = 38\,\mathrm{\mu m}$ and standard deviation $\sigma = 0.04\bar{d}$; and the number of charges in a particle is assumed to hold the following proportionality to its size:

$$\frac{q}{\bar{q}} = \left(\frac{d}{\bar{d}}\right)^3. \tag{2.18}$$

Particle radial emission location follow a Gaussian distributions following earlier published simulations[9, 10, 11]. In some publications, the distribution parameters are not specified[9, 10], while later authors[11] specify the radial perturbation mean, $\overline{\Delta r_{em}}$, to be two average particle diameters, $\bar{d}$. For the presented validation case, the DELI model uses

$$\overline{r_{em}} = 2\bar{d} \tag{2.19}$$

in line with [11]. For the axial emission location coordinate, Gañán-Calvo et al. and Wilhelm et al. perturbed emission randomly within 60-100 times the jet radius r_j above jet emission from the conical meniscus[9, 10]. When the axial emission range $P(z_{em}) = (60r_j, 100r_j)$ was implemented in DELI Model simulations, the range proved to be so wide that it occasionally prompted spurious 'physical collisions' between particles by emitting a particle on top of an existing particle, as discussed in Sec. 2.3.4. Grifoll and Rosell-Llompart elected to prevent such collisions by altering the emission axial range to be in an interval equal to jet speed (used also as particle emission speed) times default time step[11]. In the absence of a sub-module which adjusts the near-emission electric field as axial emission coordinate varies, as discussed in Sec. 2.3.4, the DELI model prevents non-physical collisions in this validation case by holding the axial emission coordinate constant at the most-probable axial breakup position for a cone-jet electrospray, one Taylor cone height and 3 mean varicose breakup wavelengths, or 13.5 $\bar{d}$, downstream of the emitter electrode[9, 55].

To ensure an accurate comparison with previous results, simulations used an analytical form of the electric field applied by electrodes instead of importing an electric field generated by COMSOL. The analytical eletric potential takes the form

$$\phi(r, z) = \frac{K_v}{\log(4H/R_a)} \log\left(\frac{[r^2 + (1-z)^2]^(1/2) + (1-z)}{[r^2 + (1+z)^2]^(1/2) + (1+z)}\right), \tag{2.20}$$

where r and z are cylindrical coordinates normalized with emitter-to-extractor distance H, R_a is the needle radius, and K_v is a constant set to 1 for the presented simulations for the given geometry. The coefficient of drag utilized for this validation case is

$$C_D = \frac{24}{Re}(1 + 0.15 Re^{0.687}), \tag{2.21}$$

valid for $Re < 800$ [86], which was used by Wilhelm et al. for the same validation case[10]. The coefficient of drag used in the original simulation of this source data is not specified[9], and the coefficient of drag term used by Grifoll and Rosell-Llompart[11] differs from the Wilhelm et al. term by less than 1 % across all Reynolds numbers for which both equations are valid.

The average particle emission period for the presented electrospray is

$$\overline{t_{em}} = \frac{\bar{m}}{\dot{m}} = 11.97\mu\text{s} \tag{2.22}$$

such that the DELI default time step according to Sec. 2.3.3 is $\Delta t = 0.1\,\mu\text{s}$, such that

$$\Delta t < \frac{\overline{t_{em}}}{100}. \tag{2.23}$$

This time step is one tenth that utilized by Grifoll and Rosell-Llompart, which they obtained by calculating the minimum electric variation time, or time needed to experience 100% change in the total electric force applied to a particle (from electrodes, other particle, and image charges), of any particle in the plume.

The DELI results for plume shape at steady state are compared to the steady-state plume outline results of previous authors in Fig. 2.10. Simulated plumes are defined to have reached steady state when the number of particles in the domain becomes asymptotic[9, 26]. Both the lateral x and axial z coordinates have been normalized with collector plate height H. The outline for the Gañán-Calvo et al. result was obtained by symmetrically mirroring the published result for radial outline[9] across the emission axis. The DELI plume result is consistent with previous published results, featuring higher particle density near the upstream emission region and a more dispersed plume further downstream. The DELI result for the validation case plume at steady state contains 502 ± 4 particles, in the range of the 400 ± 2 particles reported by Gañán-Calvo et al.[9], 552 particle average reported by Wilhelm et al.[10], and 498.2 particle average reported by Grifoll and Rosell-Llompart[11].

In Fig. 2.11, a comparison is provided between DELI results and those of previously published studies for normalized mean axial particle velocity over normalized distance from the collector plate. The distance from the collector plate is normalized with collector plate height H, and the velocity is normalized following Gañán-Calvo et al. [9] with $v_{norm} = H/t_v$, where $t_v = \bar{d}/18\alpha\nu_{fl}$ is the particle viscous relaxation time, $\alpha = \rho_{fl}/\rho_p$, and ρ_p is emitted fluid density. The DELI result matches closely with that of Wilhelm et al.[10], which is also very near the Grifoll and Rosell-Llompart result[11]. The trends in velocity seen in all three previous studies are clearly preserved in the DELI results: particles undergo electrostatic acceleration near the high-voltage emitter (normalized distance to plate ≈ 1), before being drag-decelerated as they move downstream, and finally slightly accelerated again by image

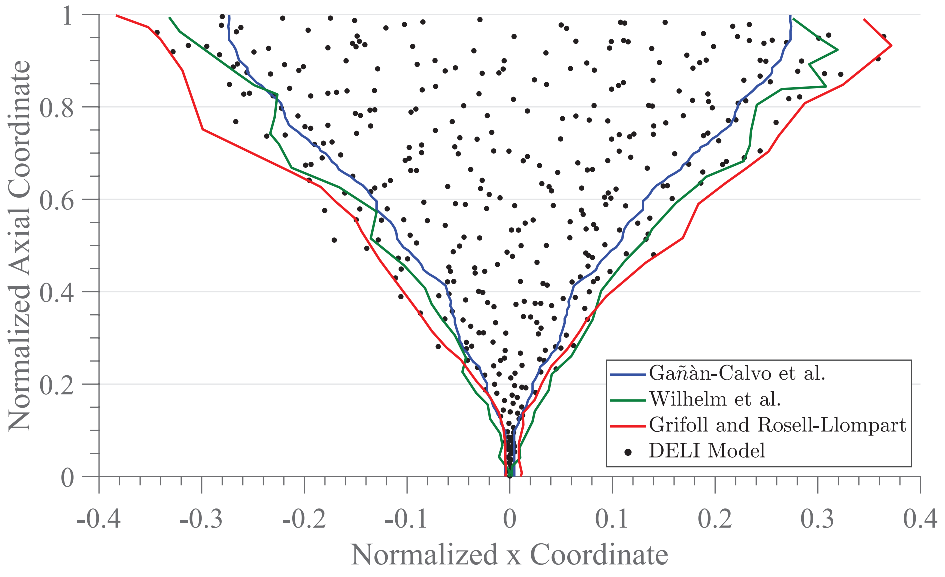


Figure 2.10: DELI results for plume shape evolved to steady state are compared with previously published steady state results.

charges as they approach the collector plate (normalized distance to plate $\rightarrow 0$). Wilhelm et al.[10] demonstrates that a fit can be obtained to the Gañán-Calvo et al. mean axial velocity result[9], which deviates from that of other authors, by altering a constant in the analytical electric potential equation. Furthermore, Wilhelm et al.[10] notes that the Gañán-Calvo et al. result[9] includes an erroneously strong image charge effect responsible for the exceedingly strong near-plate accelerations in the Gañán-Calvo solution in Fig. 2.11 and the overall narrower plume in the Gañán-Calvo et al. result in Fig. 2.10. This assertion about the image charge is supported by Grifoll and Rosell-Llompart[11] results and further corroborated by DELI results for mean axial velocity trends and evolved plume shape. It is notable that Wilhelm et al.[10], Grifoll and Rosell-Llompart[11], and DELI results still contain a slight visible increase in particle velocity very near the collector plate from the induced image charge, but not so strong as in the Gañán-Calvo et al. result[9].

A final means of validation with experimental results[9] and the simulation results of previous authors[9, 10, 11] is presented in Fig. 2.12. This figure displays particle axial velocity

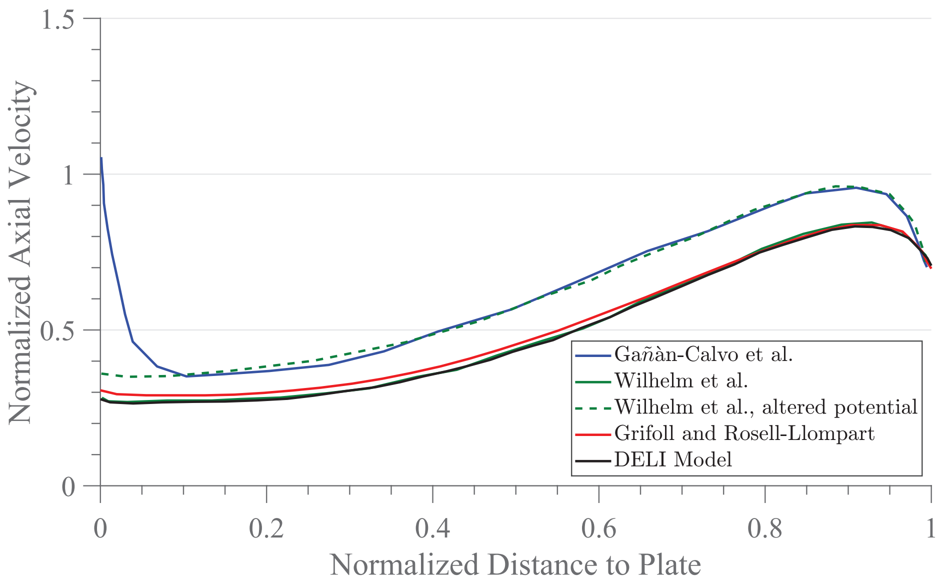


Figure 2.11: DELI results for mean axial velocity approaching the collector plate are compared with previously published results.

trends, normalized with $v_{norm} = H/t_v$ as in Gañán-Calvo et al.[9], over radial coordinate, normalized with collector plate height H, at several constant axial positions in the steady-state plume. Three clear trends can be identified from the DELI simulation results with drag presented in Fig. 2.12: 1) the plume widens with increasing axial coordinate, 2) the plume slows with increasing axial coordinate at a given radial coordinate, 3) the plume slows with increasing radial coordinate at a given axial coordinate, with some local variation. All of these trends match the simulation results presented by Gañán-Calvo et al.[9], Wilhelm et al.[10], and Grifoll and Rosell-Llompart[11], and experimental results presented by Gañán-Calvo et al.[9], validating the DELI Model for simulating electrospray plume evolution.

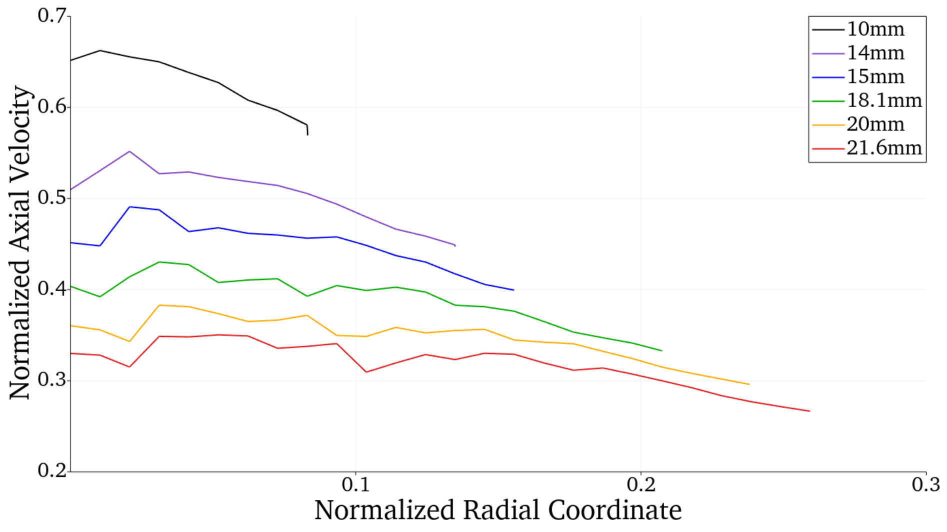


Figure 2.12: DELI results for normalized particle velocity magnitude over radial position are shown for several axial heights. The axial positions 14 mm, 18.1 mm, and 21.6 mm correspond to the heights for which Gañán-Calvo et al. presented simulation and experimental results[9]. The axial positions 10 mm, 15 mm, and 20 mm correspond to the heights for which Wilhelm et al.[10] and Grifoll and Rosell-Llompart[11] presented simulation results.

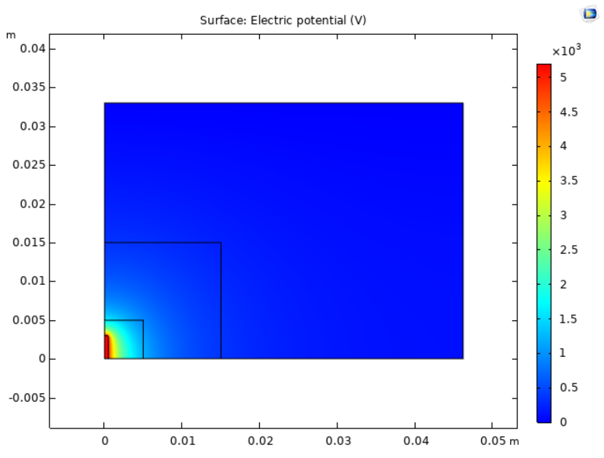
CHAPTER 3

Electrostatic Plume Divergence

The electric field in an electrospray thruster is generated by applying a potential difference between an emitter and one or more downstream electrodes. Such a potential difference between an emitter and one downstream electrode with a thrust-releasing aperture, generated in COMSOL, is presented in Fig. 3.1. Fig. 3.1b presents a magnified view of the potential field surrounding the emitter, which is held at 5 kV. In this case, the emitter is at high voltage and the downstream electrode is grounded, but these potentials can be changed to tune the resulting electrostatic field. The direction of the electric field can be reversed in order to draw out particles of opposite charge, a technique which is employed in bipolar electrospray thrusters[87].

The thrust-releasing aperture is centered above the emitter such that the resulting electric field is axisymmetric. On the axis of emission, the electric field is strictly axial. Therefore, the electric field will not cause radial divergence in a line of particles emitted straight down the axis of emission. Off of the axis of emission, there is a radial component to the electric field due to the thrust-releasing aperture in the downstream electrode. This radial electric field component contributes radial electrostatic acceleration to particles emitted off of the emission axis, thereby exacerbating particle displacement from the axis. The axial and radial components of the electric field for the region surrounding the emitter are shown in Fig. 3.2b and Fig. 3.2a, respectively.

The electric field is strongest on the corners of the tapered emitter where 1) the emitter curvature is the high and 2) the downstream electrode is closest due to the thrust-releasing


(a)

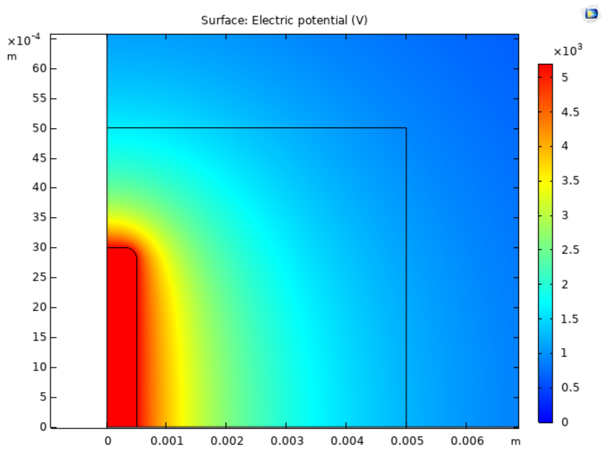
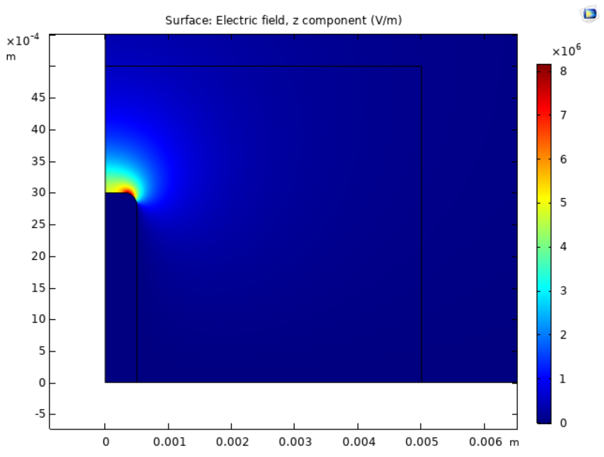

(b)

Figure 3.1: A 2D slice showing half of the axisymmetric electric potential field (a) in the full domain between the emitter and one downstream electrode with a thrust-releasing aperture and (b) only in the region surrounding the high-voltage emitter.


(a)

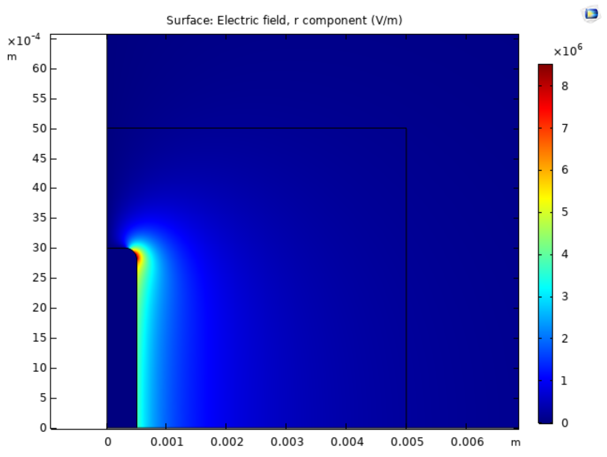

(b)

Figure 3.2: The (a) axial and (b) radial components of the electric field surrounding the high-voltage emitter.

aperture, as shown in Fig. 3.2. Even in electrospray geometries with a solid collector plate with no thrust-releasing aperture, the curvature of the emitter (and the fluid cone-jet) still contributes a radial component to the electric field. One could argue that in an ideal geometry, particles are emitted in the center of an elongated emitter tip, far enough from any upstream emitter tapering to be effected by electric field concentration due to emitter curvature. Furthermore, in the ideal case of a steady cone-jet, the electric field is symmetric such that particles which are emitted from the exact center of the jet tip will be not radially electrostatically accelerated. However, in reality, electrospray particles are always emitted with some non-zero divergence range around the axis of emission due to hydrodynamic instabilities and micro-scale roughness on the emitter surface. Therefore, electrostatic plume divergence is inherent to electrosprays due to the curvature of jet tip, and it cannot be precluded with geometric design changes to the emitter or the downstream electrode aperture. Electrostatic plume divergence can be mitigated to some degree by scaling the distance between downstream electrodes and the aperture width of the electrodes. Such geometric changes are demonstrated in Fig. 3.3 for an electrospray with two downstream electrodes, an ‘extractor grid’ and further downstream ‘accelerator grid.’

Decreasing the distance between these electrodes while holding their potentials constant increases the strength of the electric field generated between them, thereby increasing the axial component of the electric field which generates thrust and the radial component of the electric field which contributes to plume divergence. Decreased aperture width corresponds to less electrostatic plume divergence because there is a lesser radial component to the electric field. However, it also corresponds to decreased thruster lifetime because particles at wide angles strike the downstream electrode instead of passing through the aperture. Particles which impinge on downstream electrodes saturate the electrodes over time, continually detracting from thruster performance and eventually leading to propellant backspray and thruster failure[3]. Conversely, wider apertures increase contamination in the thruster domain from the external space environment. Figure 3.4, reproduced from [3], shows the

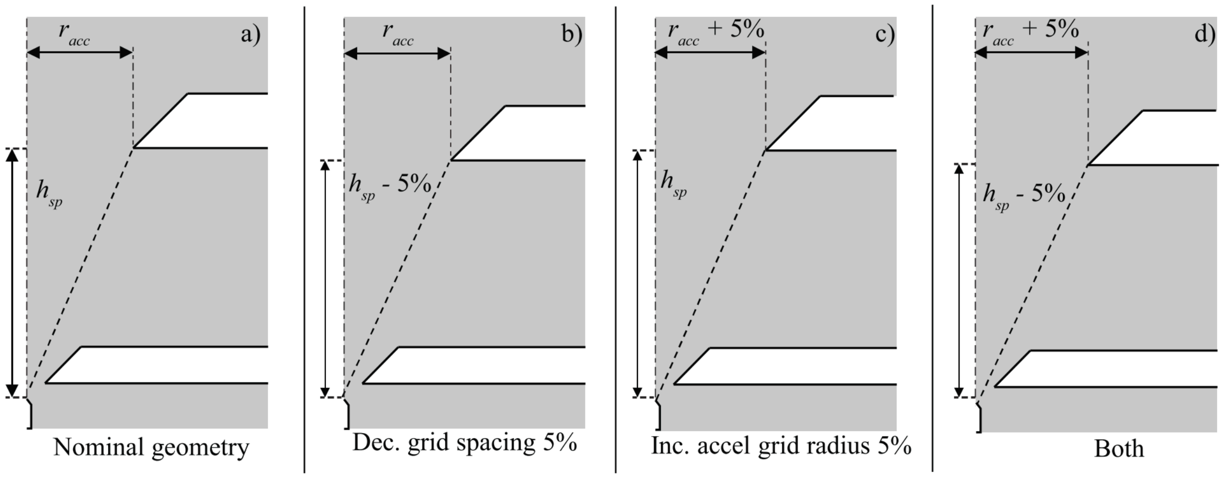


Figure 3.3: Varying accelerator grid geometries demonstrating methods to increase the angle from the emitter to the accelerator grid (marked with dashed line) from a) nominal configuration by b) decreasing grid spacing, c) increasing accelerator grid radius, and d) a combination of spacing and radius changes. [3].

influence on thruster lifetime (based on time until electrode saturation) of changes to the distance between electrodes and the accelerator electrode aperture width.

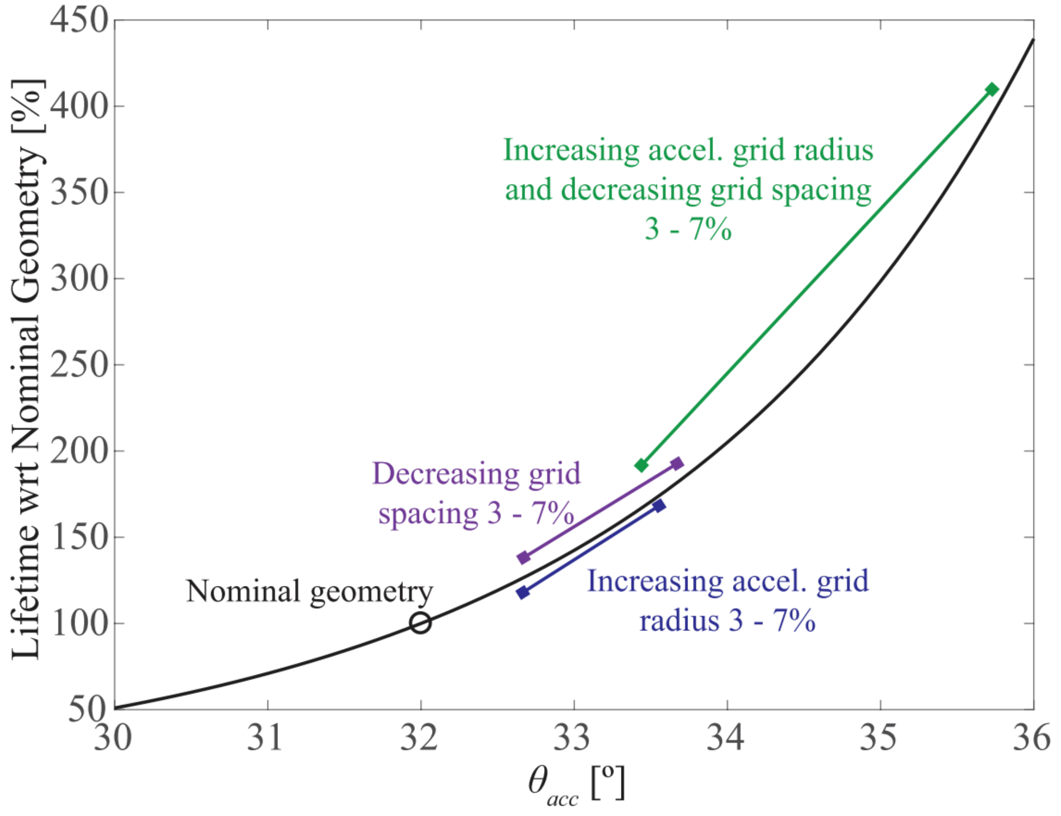


Figure 3.4: The effects of changing the accelerator grid's aperture radius and the spacing between the accelerator and extractor grids on time to saturate the accelerator grid. [3].

CHAPTER 4

Coulomb Plume Divergence

The Coulomb force is the electrostatic force generated between spherically symmetric charged particles which are stationary in a non-accelerating frame of reference. The Coulomb force between two particles is:

$$\mathbf{F_C} = \frac{1}{4\pi\epsilon_0}\frac{q_1 q_2 \hat{\mathbf{r_{12}}}}{|\mathbf{r_{12}}|^2}, \tag{4.1}$$

where q_1 and q_2 are the particle charges, $\mathbf{r_{12}}$ is the displacement vector between the charges, and ε_0 is the permittivity of vacuum. In an ensemble of particles, the sum Coulomb force on a particle of charge q is

$$\mathbf{F_C} = \frac{q}{4\pi\varepsilon_0}\sum_{i=1}^{n}\frac{q_i \mathbf{r_i}}{|\mathbf{r_i}|^3}, \tag{4.2}$$

where n is this number of particles in the ensemble. Eq. 4.2 is the Coulomb force equation used within Eq. 5.19 for particle propagation in the DELI model. As noted in 2.3.1, the DELI model approximates fluid electrospray particles as spheres of uniform charge density in order to apply this analytically simple Coulomb force equation. The literature presents analyses of the complex electromagnetic forces generated between non-symmetrical fluid particles with moving charges[88, 89, 22]; however, these analyses currently study individual particle pairs and are too computationally expensive to extend to the full electrospray plume under existing computing capabilities. When interacting charges are moving or the frame of reference is accelerating, Eq.s 4.1 and 4.2 oversimplify the resulting force because they do not account for the magnetic fields generated by moving charges. These magnetic fields become non-negligible as particles approach relativistic speeds, such as in particle accelerators[22, 89]. However, electrospray particles do not reach sufficient velocities for the simplified Coulomb force expressions to introduce substantive error in electrospray plume simulations. Eq. 4.2 is the literature standard for Coulomb force in Lagrangian plume simulations. How the Coulomb force in electrospray plumes can be further approximated for computational efficiency was introduced in Sec. 2.1 and is discussed further in Sec. 4.4.

The Coulomb force repels charged particles away from one another. If particle are emitted in a perfectly straight line, the Coulomb forces between them will be directed along that line and Coulomb interactions will not displace particles away from the line. Therefore, if electrospray particles are emitted down the axis of emission with no radial displacement, Coulomb forces will not introduce radial displacement. However, as discussed in Ch. 3, electrospray particles are always emitted with some range in radial position due to hydrodynamic instabilities and micro-scale emitter asymmetries. Therefore, Coulomb interactions are inherent to all electrosprays regardless of whether the geometry is thrust-releasing or contains a solid collector plate. The Coulomb force is an origin of plume divergence in all cases of electrospray operation.

4.1 Experimental Motivation

Early in this dissertation research, we hypothesized that Coulomb interactions served as the primary means of plume divergence based on experimental high-speed video (HSV) of atmospheric ethanol plumes obtained by the UCLA PESPL. Figure 4.1 displays HSV frames showing a set of particles which is initially near-linear before several particles cluster together and are displaced radially. Frame-by-frame particle tracking shows that the green particle moves at a faster velocity than the blue particle preceding the displacement of these particles. This velocity difference causes the distance between the blue and green particles to decrease over time such that the particles are clustered together. Following this clustering, the blue and green particles are displaced radially in opposite directions. This radial displacement spreads both upstream and downstream in the set of particles, yielding

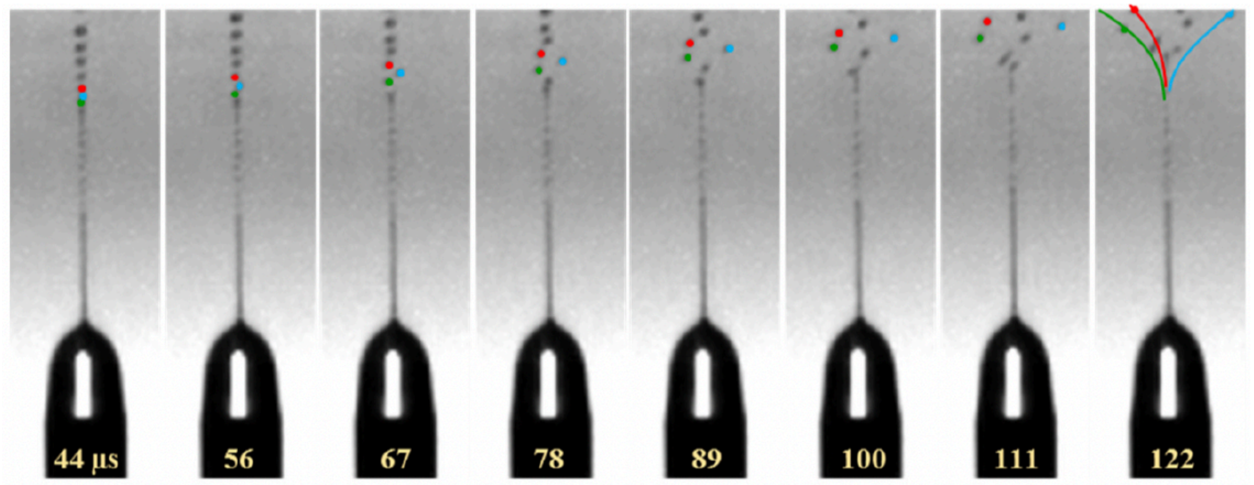


Figure 4.1: Successive high speed video frames obtained on the UCLA PESPL Atmospheric Pressure Electrospray eXperiment (APEX) system show electrospray particle clustering preceding plume divergence[12].

overall plume divergence.

From this video analysis, we hypothesized that the forces driving plume divergence correlate inversely with the distance between plume particles. Following this line of thought, we hypothesized that Coulomb forces are responsible for plume divergence given the inverse-squared relationship between Coulomb force and the distance between particles, stated in Eq. 4.2. Given the relative velocity difference between the green and blue particles prior to their radial displacement, we further hypothesized that differences in particle speed - specifically upstream velocity gradients, in which upstream particles move faster than their downstream neighbors - cause particles to cluster. Inspired by the UCLA PESPL surrounding of Los Angeles, we colloquially termed this phenomena a 'traffic jam.' The increased proximity of the particles in this clustered state magnifies the Coulomb forces between particles, yielding plume divergence [90, 68].

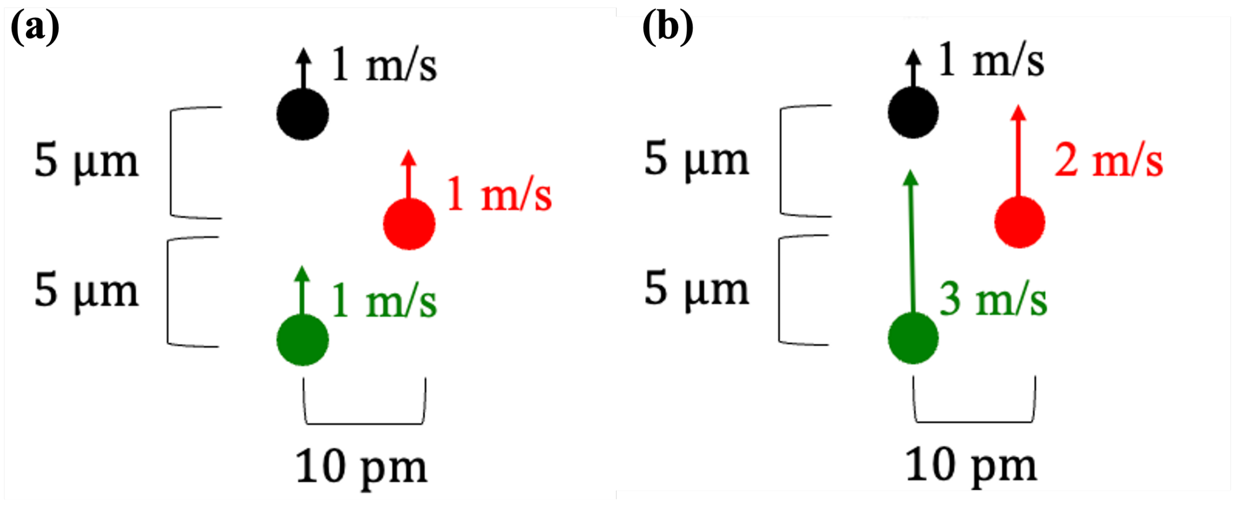


Figure 4.2: Three ethanol particles with axial spacing Δz = 0 µm and the middle particle displaced to the right by Δx =10 pm. The particles have (a) equal initial velocities v =1 m/s and (b) an upstream initial velocity gradient with initial velocities v_1 = 1 m/s, v_2 = 2 m/s, v_3 = 3 m/s from downstream to upstream.

4.2 Three Particle Demonstration

We tested our hypothesis of Coulomb plume divergence for the case of three particles. We compared two simulation scenarios to observe the effect of relative particle velocity on resulting particle divergence: one in which the particles have equal velocities, and the other in which there is an upstream particle velocity gradient. In both cases, three identical ethanol particles are evenly distributed over a distance Δz = 10 µm and the middle particle is laterally displaced to the right of the others by Δx =10 pm. These initial conditions are portrayed in Fig. 4.2. To isolate the particle dynamics to only inertia and Coulomb interactions, drag is neglected and there is no applied electric field. The simulation results are presented in Fig. 4.3.

In case a), in which the particles have the same initial velocity (v = 1 m/s), particles maintain their axial separation and are only slightly perturbed laterally via Coulombic in-

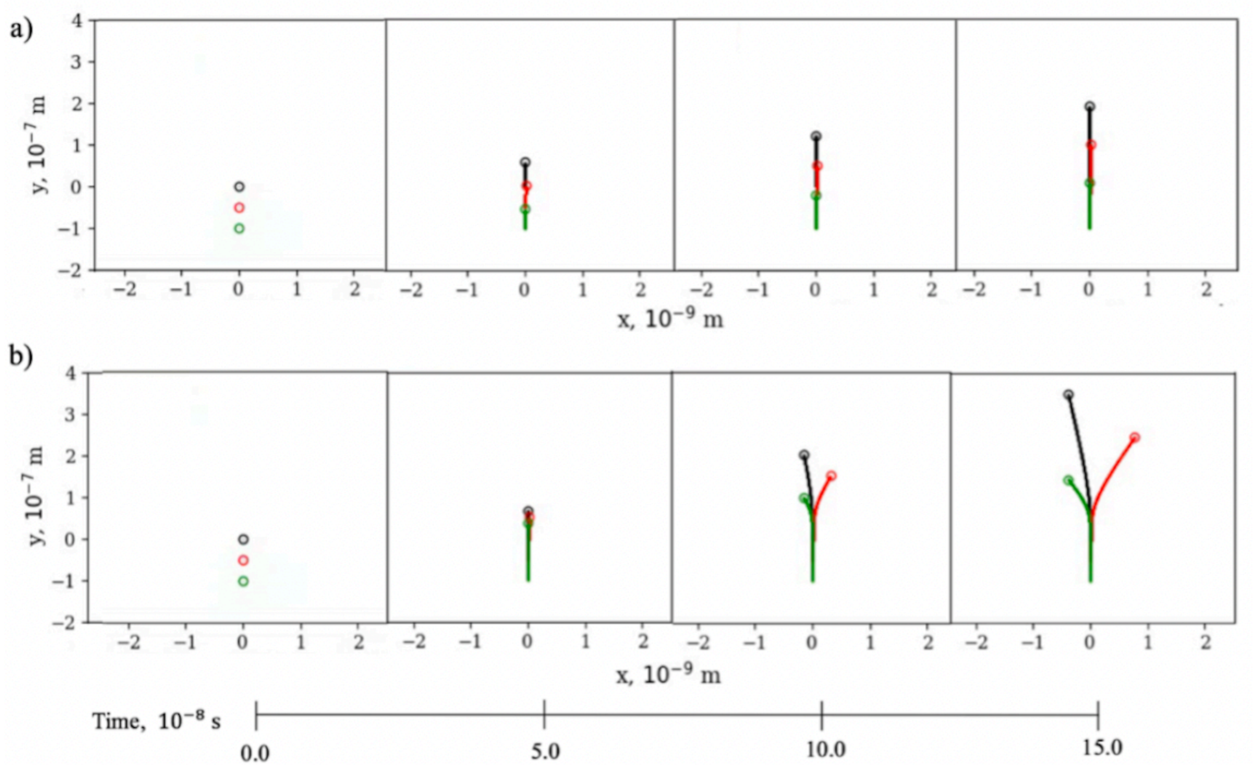


Figure 4.3: Displacement from Coulomb interaction between particles with a) equal initial velocities $v = 1$ m/s and b) an upstream initial velocity gradient, with initial velocities $v_1 = 1$ m/s, $v_2 = 2$ m/s, $v_3 = 3$ m/s from downstream to upstream.

teractions. However, in case b), there is an upstream initial velocity gradient ($v_1 = 1\,\mathrm{m/s}$, $v_2 = 2\,\mathrm{m/s}$, $v_3 = 3\,\mathrm{m/s}$ from downstream/front to upstream/back) which causes particles to cluster and have Coulomb interactions magnified by their close proximity. This 'traffic jam,' coupled with the initial lateral offset of the middle particle, causes all three particles to be laterally displaced by the inter-particle Coulomb forces. These simulations demonstrate that the relative velocities between particles determine the degree to which a small displacement in their initial positions is expanded into larger divergence. Therefore, these simulations corroborate our experimentally-motivated hypothesis that upstream particle velocity gradients enhance plume divergence through Coulomb interactions.

4.3 Theory of Coulomb Plume Divergence

We have provided experimental and computational evidence supporting our initial hypothesis of Coulomb plume divergence, such that it has developed from a hypothesis into a theory. A summary of our theory of Coulomb plume divergence is as follows. Some emitted particles move forward faster than their downstream neighbors, such that they becomes close to the forward particles over time ('traffic jams'). The Coulomb force exerted on these particles increases in response to their increased proximity, such that any difference in their radial coordinates is exacerbated by the Coulomb interaction. Through such differences in particle axial velocity, a collection of particles with a small range in radial coordinate will experience localized particle clustering events which further displace particles radially through Coulomb interactions, thereby resulting in plume divergence. Without such differences in axial velocity, particles emitted in a near-linear formation cannot cluster and increase their Coulomb influence on one another beyond their initial state. Coulomb interactions still act over longer distances to expand differences in particle radial coordinates, but this Coulomb plume divergence process occurs much more slowly than in plumes with particle clustering events.

This theory for plume divergence supports experimental observations and simulation replications of particle size-segregation in electrospray plumes[55, 59, 9, 47]. In an interaction between polydisperse particles, the Coulomb force is exerted identically on the two particles, but the lower mass (and therefore smaller, assuming constant density) particle is more accelerated by the interaction, and is thereby repelled to a wider plume angle. Therefore, smaller particles will be generally found at wider angles in polydisperse plumes than larger particles as a result of Coulomb interactions. If plume particles are not only size-diverse, but also have a range of specific charge, then the plume will be segregated by specific-charge in addition to mass as a result of Coulomb interactions. This specific charge segregation trend can be inferred by combining Newton's Second Law of Motion with the Coulomb force Eq. 4.2 to yield an equation for Coulomb acceleration of a particle with charge q and mass m in a charge ensemble:

$$\begin{aligned} \mathbf{a_C} &= \frac{q}{4\pi\varepsilon_0 m}\sum_{i=1}^{n}\frac{q_i\mathbf{r_i}}{|\mathbf{r_i}|^3} \\ &= \frac{q}{m}\mathbf{E_C}. \end{aligned} \tag{4.3}$$

The acceleration imparted to a particle from a Coulomb interaction is therefore a function of specific charge, such that high specific charge particles such as ions are Coulombically repelled to wider plume angles than low specific charge particles such as heavy droplets.

Specific charge inhomogeneity in an emitted particle population creates differences in particle velocity due to differences in electrostatic acceleration. These differences in velocity lead to plume divergence through Coulomb interactions, as displayed in Sec. 4.2. Particle electrostatic acceleration from the applied electric field correlates linearly specific charge, following Lorentz Force Law and Newton's Second Law of Motion:

$$\mathbf{a_E} = \frac{q}{m}\mathbf{E_A}. \tag{4.4}$$

A high specific charge particle emitted behind a lower specific charge particle will be electrostatically accelerated forward into the larger particle, causing a high-magnitude Coulomb interaction between the clustered particles which will displace both particles, but predominantly the smaller particle. The drag force is also capable of introducing velocity differences in particle velocity in a polydisperse plume. Drag decelerates each particle as a function of diameter as given in Eq. 5.17, such that larger particles are slowed more by the drag force than smaller particles. In summary, if a plume is comprised of specific charge inhomogeneous or polydisperse particles, the applied electrostatic force or the drag force, respectively, will introduce differences in particle velocity even if the particles are emitted with the same velocity. These differences in particle velocity will lead to particle clustering in localized areas with upstream particle velocity gradients, thereby yielding plume divergence through Coulomb interactions.

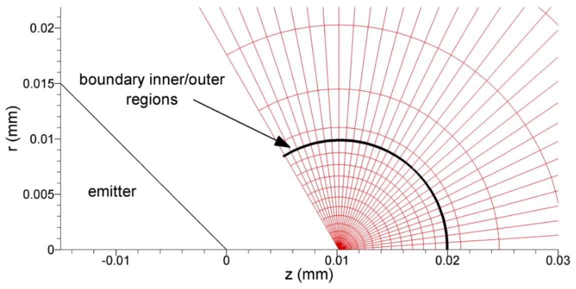
4.4 Defining the Interaction Region

We have identified that Coulomb interactions are an intrinsic source of divergence in electrospray plumes, regardless of system design. We have also presented the computational challenge of simulating the N-body problem created by these Coulomb interactions in Sec. 2.1. Simulations with the fully discrete Coulomb term given in Eq. 4.2 have been limited by current computational capabilities to simulating fully evolved electrospray plumes with on order of 10^5 micron-radius particles. Simulations of smaller particles (e.g. ions) which use a fully discrete Coulomb term are limited to curtailed domain lengths giving only a portion of the full experimental plume. Such plumes have been analytically estimated to contain 10^8 particles[91], which is computationally overwhelming with a fully discrete Coulomb term.

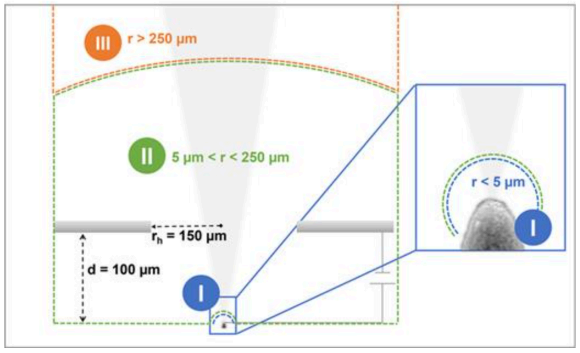
Due to the computational limitations of a fully discrete Coulomb term, significant attention has been given in the literature to means of approximating the Coulomb term while preserving simulation accuracy. Several such methods have been presented in Sec. 2.1. Initially, approximations for the Coulomb term were applied indiscriminately across the full plume domain. However, this assumption that all regions of the plume should be treated equally with regards to the Coulomb term used in simulation is being challenged. In recent years, methods have been proposed which discretize the electrospray plume and treat independent regions differently, as presented in Fig. 4.4. Since the first photos published by Zeleny[5], electrosprays have been recognized to be most charge-dense near emission, with charge-density decreasing downstream as the plume expands. Force analysis of Lagrangian simulations of such plumes identifies that the Coulomb force dominates other governing forces in the particle-dense region near emission. As particles move downstream and are displaced radially into less particle-dense regions, the Coulomb force dominance fades into applied electrostatic force dominance. Approximations to the Coulomb term will introduce less error to simulated particle trajectories when applied in regions where Coulomb forces are not dominant. Therefore, establishing a threshold in the plume structure beyond which the Coulomb force does not dominate particle trajectories is critical towards determining where the Coulomb term can be approximated without sacrificing simulation accuracy. While such a threshold bounding the interaction region is of great computational value, the literature has yet to converge on such a definition.

This section aims to determine a threshold for the 'interaction region' in which Coulomb interactions dominate particle dynamics. Our first discussions of this concept occurred at the Fall 2018 Technology Interchange Meeting at NASA Jet Propulsion Laboratory. In the following years, two models have been presented in the literature which apply radial threshold(s)[13, 14] to determine whether a fully discrete or approximated Coulomb term is used to simulate particle motion, introduced in Sec. 2.1. Gamero-Castaño and Galobardes-Esteban defined a radial threshold around emission within which a fully discrete Coulomb term was used, and outside of which Poisson's equation was solved for the electric potential field[13]. Petro et al. defined two radial thresholds around emission: within the first 5 µm radially, the Coulomb force was calculated fully discretely in every timestep; between the first and second radial thresholds (5 µm < r 250 µm), the Coulomb force was calculated fully discretely but only updated every 10 timesteps; outside the second radial threshold

($r \; > 250\,\mu m$), Coulomb interactions were neglected [14]. These thresholding schemes are displayed in Fig. 4.4 for reference. In both these approaches to approximating the Coulomb term, all emitted particles are bounded by the same radial threshold(s)[13, 14]; however, different species in polydisperse plumes diverge differently under balancing of applied electrostatic and Coulombic forces. The objective of this section is to propose a definition for the ‘interaction region’ threshold for approximating the Coulomb term which accounts for species-dependent physics in polydisperse plumes.



(a)



(b)

Figure 4.4: Radial thresholds for approximating the Coulomb force utilized by (a) Gamero-Castaño and Galobardes-Esteban [13] and (b) Petro et al. [14].

Plume simulation results presented in this section were obtained by the Plasma, Energy & Space Propulsion Laboratory (PESPL) at the University of California, Los Angeles (UCLA) using the Discrete Electrospray Lagrangian Interaction (DELI) Model. The electrospray geometry, 1.31nl/s flowrate constraint, and emitted EMI-Im particle data for the presented plume divergence results are from Miller et al[24]; three species of particles were utilized to observe differences in species divergence. A snapshot of the simulated plume at steady state is presented in Fig. 4.5 with color corresponding to particle radius to identify separate particle species.

The purpose of defining a threshold for the interaction region is to separate the region in

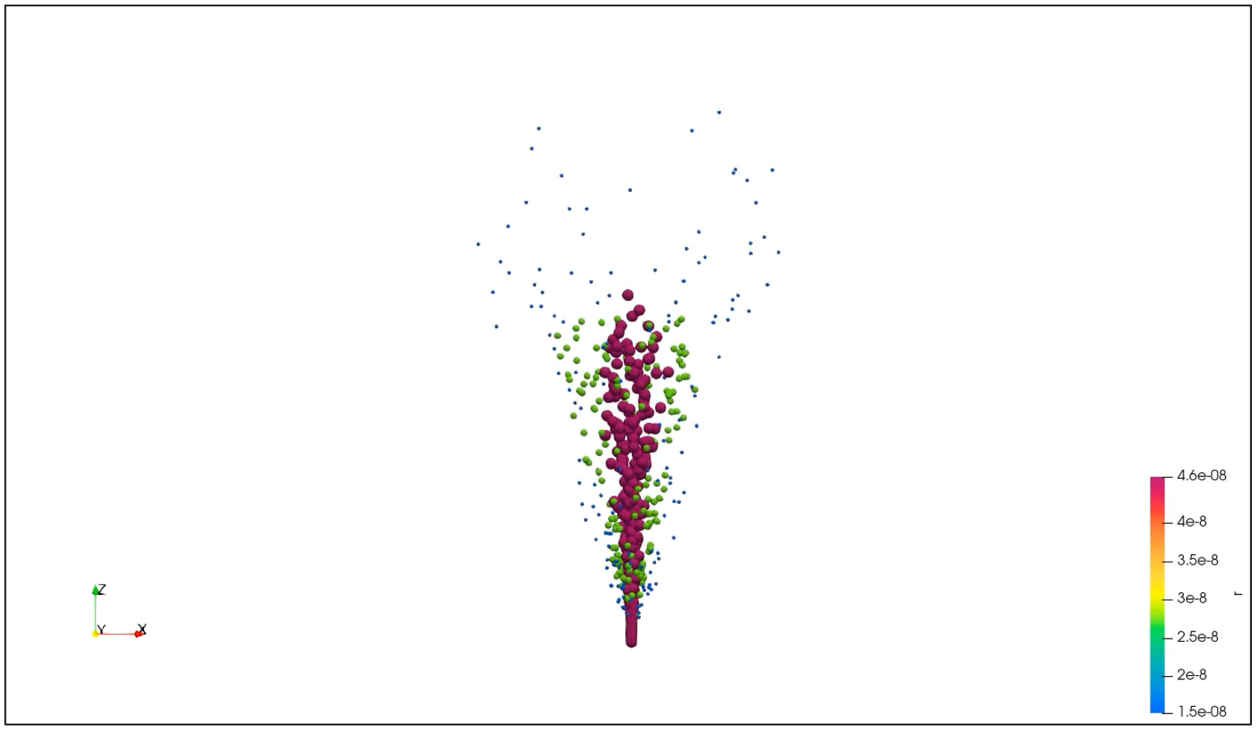


Figure 4.5: Snapshot of DELI simulation of electrospray plume with color corresponding to the radius of each of the three particle species.

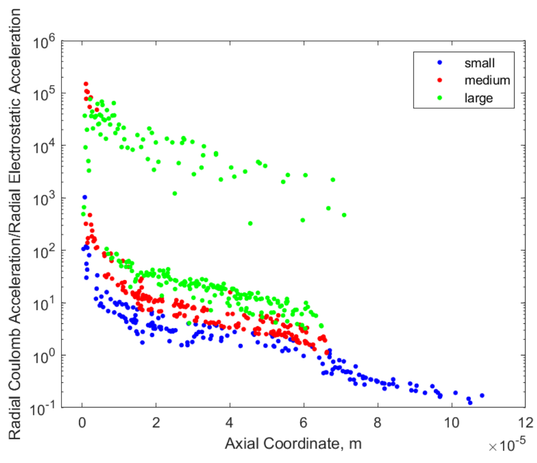
which the Coulomb force dominates plume divergence (and therefore should be calculated exactly) from the region in which it is not dominant and may be approximated. Acknowledging that plume divergence stems from the radial component of particle velocity, we define the interaction region for each particle to be where the radial component of acceleration from the Coulomb force, $\mathbf{a_{Cr}}$, is larger than radial acceleration from the applied electrostatic force, $\mathbf{a_{Er}}$. In mathematical form, a particle is in the interaction region for the range of positions, $\mathbf{R}$, for which

$$\frac{\mathbf{a_{Cr}}}{\mathbf{a_{Er}}}(\mathbf{R}) \geq 1. \tag{4.5}$$

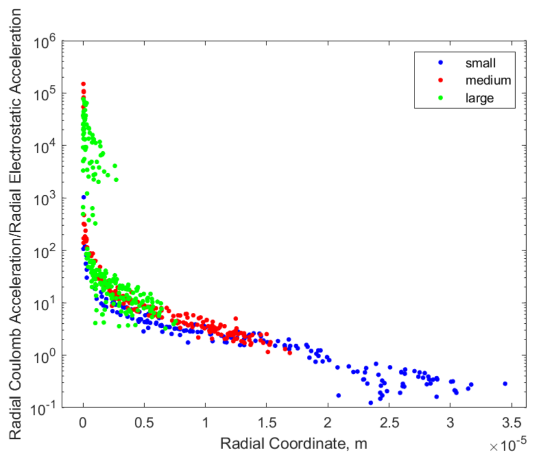
The interaction region threshold can be extended to further minimize error from approximating Coulomb interactions by decreasing the value to the right of the inequality in Eq. 4.6. For example, if 0.5 is used in place of 1, the interaction region threshold would extend to where the acceleration from the applied electric field is double the acceleration from Coulomb interactions, thereby extending the portion of the plume in which the discrete Coulomb term is used for particle propagation.

The radial acceleration ratios are shown over axial and radial coordinates for the tri-species EMI-Im plume in Fig. 4.6 for the region following emission. The interaction region threshold where the radial acceleration ratio reaches unity is seen to be species-dependent: the mass-mobile small species (blue) are quickly repelled from the particle-dense plume center and exit the interaction region after less axial propagation than the medium (red) and large (green) species which remain trapped further in the plume center. Particle species segregation is evident for the radial acceleration ratio in the same manner as has been observed for angular particle divergence. The interaction region threshold is also shown not to be one radial threshold for all species and rather is a species-dependent function of both axial and radial coordinate. There is a notable division among large species particles which diverge to wide angles and those which maintain low radial coordinates as they move downstream; particles which maintain low plume angles do not exit the interaction region defined by Eq. 4.6 even far downstream of emission due to continued Coulomb interactions near the emission axis. Therefore, Coulomb interactions cannot be approximated for such on-axis particles even beyond a threshold for which other, wider-angled particles have exited their interaction regions.

We propose that the interaction region threshold given by Eq. 4.6 should be applied on a particle-by-particle basis in electrospray plume simulations to determine when Coulomb interactions can be approximated for individual particles. It is not beneficial in simulation to define and apply a location-specific threshold for the interactions region, even for individual particle species, because stochastic Coulomb interactions change when individual particles exit the interaction region. Furthermore, it is possible for electrospray particles to exit and then re-enter regions of Coulomb dominance, such that the interaction region should be defined based on acceleration ratios per Eq. 4.6, instead of physical location with no force analysis. A spatial boundary for the interaction region of a given particle species



(a)



(b)

Figure 4.6: The ratio of radial acceleration from the Coulomb force and the applied electrostatic force are plotted over a) axial coordinate and b) radial coordinate with color corresponding to species.

can be obtained in the case that it is desirable for visualization and discussion purposes. The acceleration of a particle by electrostatic forces is a function charge and mass, such that same-species particles experience the same dynamics on a given trajectory, with the exception of stochastic near-neighbor Coulomb interactions. To consider such stochasticity, the interaction region threshold for a particle species can be defined by geometrically averaging the interaction region thresholds of all simulated particles in that species. Again, such a physical interaction region threshold should be used only for visualization purposes; in simulation, the interaction region threshold in Eq. 4.6 should be applied on a particle-by-particle basis.

In conclusion, Coulomb interactions create a computationally burdensome n-body problem in Lagrangian electrospray plume simulations and other charged particle simulations[92, 93]. The computational challenge associated with a fully discrete Coulomb term motivates the approximation of this term for computational efficiency. However, Coulomb interactions are the dominant source of radial acceleration on particles in a region near emission, and thereby are the dominant source of plume divergence in that region. Approximating Coulomb interactions in regions where they dominate plume divergence introduces error into electrospray plume simulations. Therefore, it is necessary to threshold the ‘interaction region’ in which Coulomb interactions dominate electrospray plume divergence and utilize a fully discrete Coulomb term within that region. In this section we have proposed this interaction region threshold definition to be

$$\frac{\mathbf{a_{Cr}}}{\mathbf{a_{Er}}}(\mathbf{R}) \geq 1, \tag{4.6}$$

such that particles are in the interaction region when Coulomb interactions dominate their radial acceleration. We advise that Lagrangian electrospray plume simulations apply this definition on a particle-by-particle basis such that Coulomb interactions are only approximated when they no longer dominate a particle’s divergence. In this manner, Coulomb interactions are approximated for computational efficiency only when they do not dominantly govern plume divergence.

CHAPTER 5

Drag Plume Divergence

The drag force is generated on a moving body by surrounding fluid which is resistant to being displaced. The drag force is similar to frictional forces in that it acts counter to motion; however, unlike friction, the drag force is proportional to the velocity of the body relative to the fluid. No drag force exists on bodies which are stationary with respect to the surrounding fluid. The drag force depends on the velocity, size, and shape of the moving body, as well as properties of the background fluid.

As given in Eq. 5.17 drag force acting on a spherical particle with diameter d and velocity $\mathbf{v}$, moving through a fluid of density ρ_{fl}, is

$$\mathbf{F_D} = -\frac{\pi}{8} d^2 \rho_{fl} C_D \mathbf{v}|\mathbf{v}|, \tag{5.1}$$

where C_D is the coefficient of drag. When Reynolds number,

$$Re = \frac{|\mathbf{v}|d}{\nu_{fl}}, \tag{5.2}$$

where ν_{fl} is the kinematic viscosity of the background fluid, is $Re << 1$, and Knudsen number,

$$Kn = l/d, \tag{5.3}$$

where particle mean free path is

$$l = \frac{1}{\rho_n \sigma} \tag{5.4}$$

ρ_n is the number density of the background fluid molecules, and σ is the cross-section of the fluid molecules, is $Kn < 0.001$, and Mach number,

$$M = \frac{|\mathbf{v}|}{a}, \tag{5.5}$$

where a is the local speed of sound, is subsonic (not transitional) $Ma < 0.8$, Stokes[17] found the coefficient of drag to be

$$C_D = 24/Re. \tag{5.6}$$

As Reynolds number increases, causing convective effects such as flow separation from the particle surface [16, 19]; or Knudsen number increases, causing rarefaction effects associated with decreasing air density; or Mach number increases, causing compressibility effects associated with high-speed flows, the coefficient of drag is empirically determined as a function of Reynolds, Knudsen, and Mach numbers[16, 94, 95, 96, 86, 97]. These three flow parameters are analytically related through

$$Kn = \frac{Ma}{Re}\sqrt{\frac{\gamma\pi}{2}}, \tag{5.7}$$

where γ is the ratio of specific heats.

Drag acts directly opposite to particle motion, such that it will not impart a radial component to particles with solely axial motion. Therefore, drag will not act radially on electrospray particles on a strictly axial trajectory. In the realistic electrospray plume case in which particles are emitted in a non-zero radial range around the axis of emission, the applied electric field and Coulomb interactions radially accelerate particles as discussed in Ch. 3 and Ch. 4, respectively. Those forces impart particles with non-zero radial velocity components, such that the drag force then acts counter to particle radially divergent motion. In this manner, the drag force causes radially displaced particles to remain more clustered near the central majority of the plume than they would in a dragless environment.

We discussed in Sec. 4.3 that ‘traffic jams’ between particles lead to plume divergence through Coulomb forces between close-range particles. The drag force creates velocity differences between particles of different sizes following the dependence on particle diameter given in Eq. 5.17. The drag force also exacerbates existing velocity differences between particles because it depends on particle velocity. The acceleration resulting from the drag force with

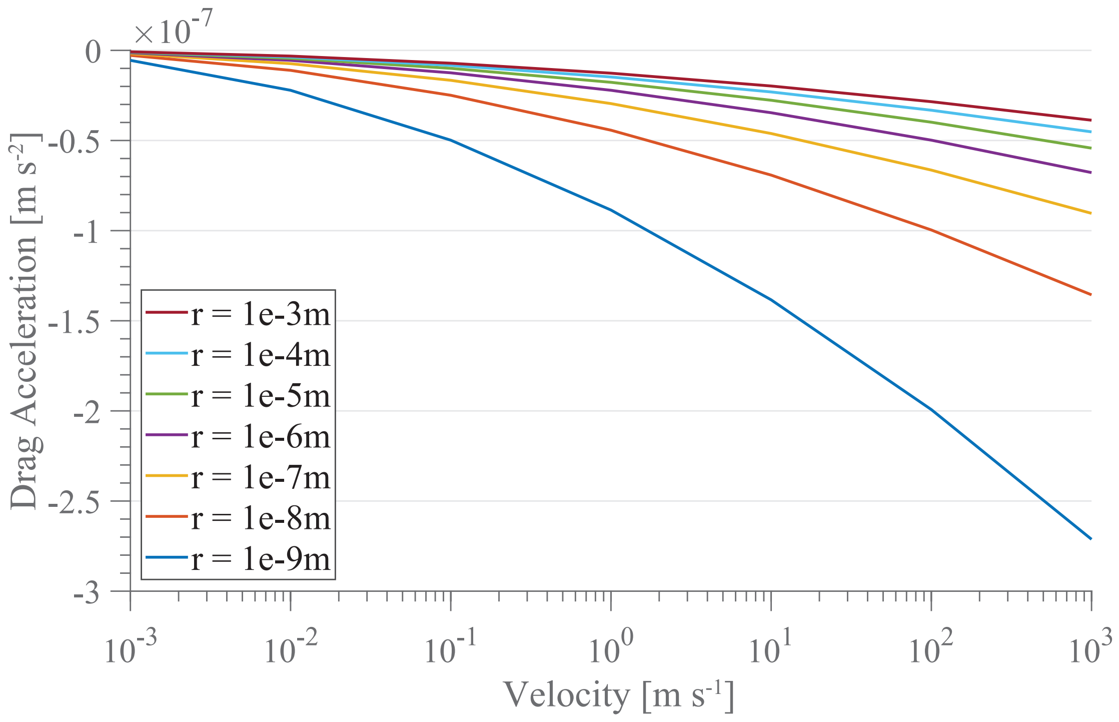


Figure 5.1: The acceleration resulting from the drag force at different velocities is shown for a range of particle sizes.

coefficient of drag[19] used in the validation case in Sec. 2.5,

$$C_D = \left(0.127 + \frac{3.957}{0.140 + Re^{0.983}}\right) \tag{5.8}$$

is presented in Fig. 5.1 for particles of different size and velocity moving through air at atmospheric pressure. Results display the difference in the drag deceleration of polydisperse particles; specifically, smaller particles are more drag decelerated. In this manner, the drag force creates velocity differences between polydisperse particles and exacerbates existing velocity differences between particles, both of which foster particle clustering events which lead to Coulomb plume divergence, as discussed in Sec. 4.3.

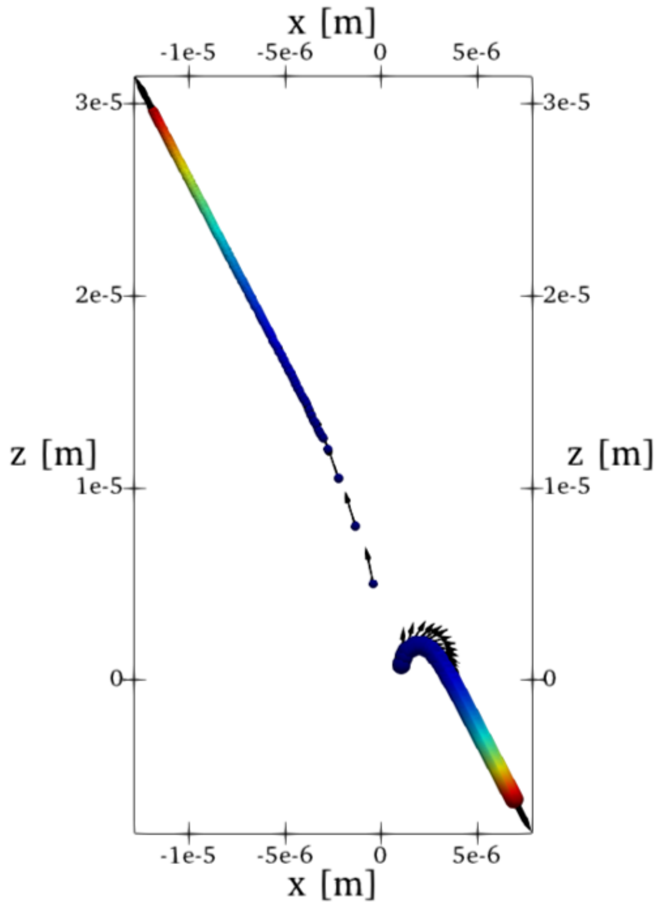
5.1 Two Particle Study

Particle displacement in a two-particle Coulomb interaction is compared in Fig. 5.2 for the case with atmospheric, full drag force (Fig. 5.2a) and the case with no drag force(Fig. 5.2b). No external electric field is applied in either case. A particle with radius $r = 1\,\mu\text{m}$ is initially positioned $3r$ behind and $1r$ to the right of a particle with radius $0.5r$. Both particles have specific charge $q/m = 1\,\text{C/kg}$, initial axial velocity $v_z = 1\,\text{m/s}$, and no initial radial velocity. Particle density is that of the Sec. 2.5 validation case particles. These size, specific charge, velocity, and density values are on order with experimentally observed electrospray particle properties [9, 98]. The simulation time step is $\Delta t = 1\,\mu\text{s}$, which captures the interaction over many time steps.

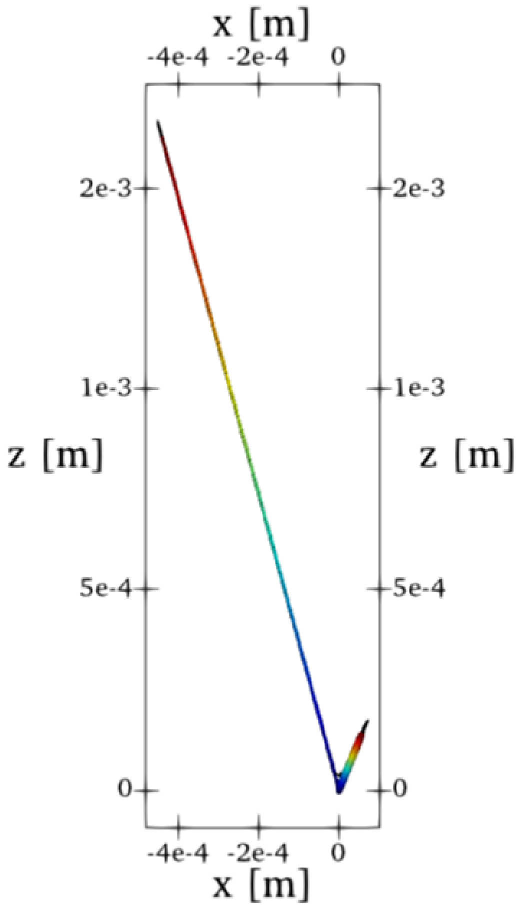
In both the cases, the small particle is more displaced from its initial position than the large particle due to its higher mass mobility. In the dragless case, both particles have linear trajectories following the Coulomb interaction because they maintain their inertia in the absence of other forces. In the case with drag, both particles are less displaced than in the dragless case (O(-5)m vs. O(-4)m displacements) because their kinetic energy is lost to drag. The small particle experiences more drag deceleration than the large particle as shown in Fig. 5.1, such that the trajectory of the smaller particle is more significantly by drag than that of the large particle. Drag has hereby been demonstrated to significantly affect the Coulomb interaction between two particles, and to influence polydisperse particles differently according to their size.

5.2 Full Plume Study

This section extends the study of drag divergence from a two-particle interaction to a full electrospray plume. Atmospheric electrospray plumes have been experimentally observed to be more divergent than those produced under the same propellant, flowrate, and voltage conditions in vacuum, as shown in Fig. 5.3. There are many differences in the physics



(a)



(b)

Figure 5.2: The two particle Coulomb interaction (a) under the influence of the drag force and (b) in a dragless environment. Arrows display particle velocity vectors.

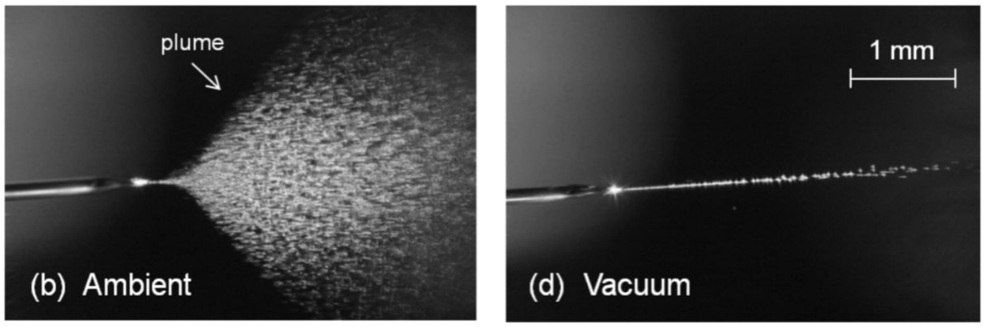


Figure 5.3: Experimental comparison of atmospheric and vacuum electrospray plumes [15].

of electrospray formation and evolution in atmospheric and vacuum environments, including the amount of drag force exerted on emitted particles, induced secondary flow in the background fluid [54], fluid evaporation and Coulomb fission of particles which have been critically size-reduced by evaporation [28, 10, 26], corona discharges[99, 4], and the properties of the generated cone-jet (stability, length, radius, velocity, etc.) and emitted particles (mass, charge, velocity, etc.)[4]. Many liquids which create stable cone-jet electrosprays in atmosphere do not have sufficiently low vapor pressure, high surface tension, and high conductivity to create a stable cone-jet electrospray in the vacuum environment. This materials science challenge precluded electrospray spacecraft propulsion from being realized for many decades following its theoretical conception and initial technology tests[100], until ionic liquids and other propellants with desired properties for vacuum electrospraying were developed[101]. Such propellants utilized in vacuum electrosprays, with higher conductivity and lower vapor pressure than those utilized atmospherically, produce particles with orders of magnitude smaller radii moving orders of magnitude faster than those in atmospheric electrosprays[4]. Therefore, differences between vacuum and atmospheric electrosprays often stem from differences in utilized propellants as well as environment-dependent physics.

This section is divided into two studies of electrospray plume evolution which seek to isolate the influence of drag divergence. The first study treats the drag force, F_D, as the independent variable, varying the fraction of atmospheric drag force applied to the plume and observing plume divergence in response. While this study provides theoretical insight towards the influence of drag on plume divergence, the drag force cannot be scaled directly in practice. The second study treats the background pressure, P, as the independent variable, because this variable can be directly controlled in an experimental setting and it is related to the drag force as explained in Sec. 5.2.2. The domain and emitted species utilized in both studies are those of the validation case described in Sec. 2.5, which were experimentally observed at atmospheric pressure. The influence of drag on emitted particle properties is beyond the scope of this dissertation, which focuses on evolution of electrospray plume following particle emission.

5.2.1 Influence of Drag Force

In this subsection, different fractions of atmospheric drag force are applied to simulated plumes and their evolution is compared. The drag force is calculated given in Eq. 5.17 with the coefficient of drag from Eq. 5.8. In this study, drag is not modified as a function of Reynolds, Mach, and Knudsen numbers based on background pressure; rather, a scaling coefficient is multiplied by the drag force at atmospheric pressure. The scaling coefficients used in this study are 1, $\frac{3}{4}$, $\frac{1}{2}$, $\frac{1}{4}$, and 0. In this manner, the relationship trend between drag force and plume divergence can be obtained without yet considering how empirical coefficients of drag vary with flow parameters. In all simulations, the axial emission location of particles is held constant at the most probable jet breakup location as in the validation case in Sec. 2.5. The radial emission location range is that presented in Sec. 2.3.4, such that simulated emission is in the steady cone-jet mode. The electric field applied by the electrodes, E_A in Eq. 2.11, is determined using COMSOL as described in Sec. 2.3.2 for the Sec. 2.5 validation case. The flowchart for particle propagation presented in Fig. 5.7 is representative of the algorithm for this study.

Plume structures evolved to steady state under the different fractions of applied drag force are compared in Fig. 5.4, in which both the lateral x and axial z coordinates are

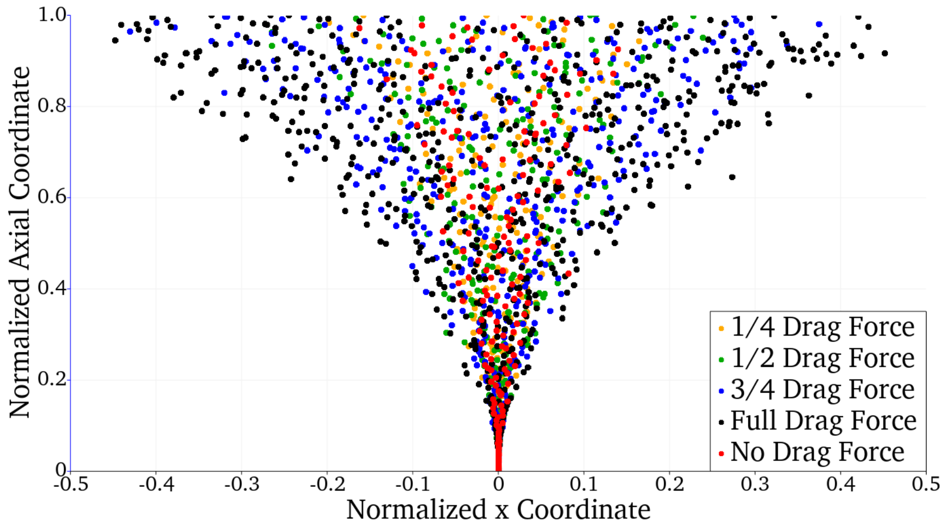


Figure 5.4: A comparison between plumes evolved to steady state with different levels of applied drag force.

normalized with emitter-to-collector-plate-length H. Drag positively contributes to plume divergence, with plumes expanding wider with increasing fraction of drag force. At the collector plate, the widest particle in the dragless plume reaches less than half the angle of the widest particle in the full atmospheric drag plume.

Figure 5.5 provides comparisons of the mean axial velocity over normalized distance from plate for the plumes evolved with different fractions of the drag force. Without drag, particles are electrostatically accelerated away from the high voltage emitter and maintain the resulting high velocities as they move downstream. With increasing fraction of drag force applied, particles are increasingly drag decelerated following their electrostatic acceleration, slowing them as they move downstream. There is an order of magnitude difference between mean terminal velocity at the collector plate in the dragless and the full drag cases.

Drag deceleration causes particles to remain in the simulation domain for longer residence times, such that the number of particles in a plume evolved to steady state increases with increased drag force. Fig. 5.6 compares the number of particles at steady state in plumes evolved with different fractions of the atmospheric drag force, with steady state defined when

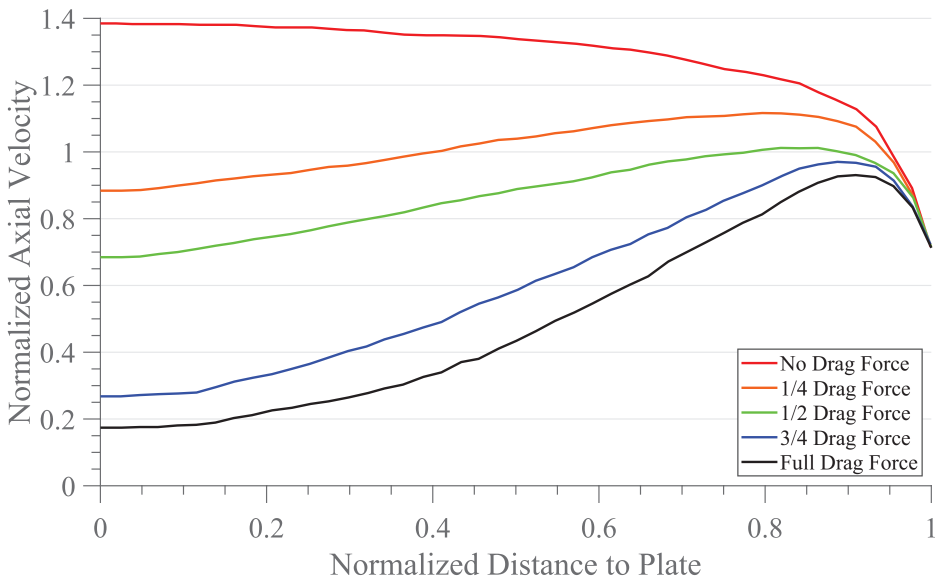


Figure 5.5: The 3D plume evolved to steady-state with different fractions of drag force applied.

the number of particles in the simulation domain plateaus (with little variance)[26, 9]. For the dragless case, the steady state plume contains 169 ± 5 particles, with $\frac{1}{4}$ drag force it has 212 ± 3 particles, 251 ± 4 particles with $\frac{1}{2}$ drag force, 403 ± 6 particles with $\frac{3}{4}$ drag force, and 607 ± 6 particles with the full drag force applied. This increase in steady state particle number with increased drag force fits with results in Fig. 5.4 of plumes widening with increasing drag force. Inter-particle Coulomb forces grow in magnitude with increased particle density, such that plumes with higher number particles experience greater Coulomb plume divergence as described in Sec. 4.3.

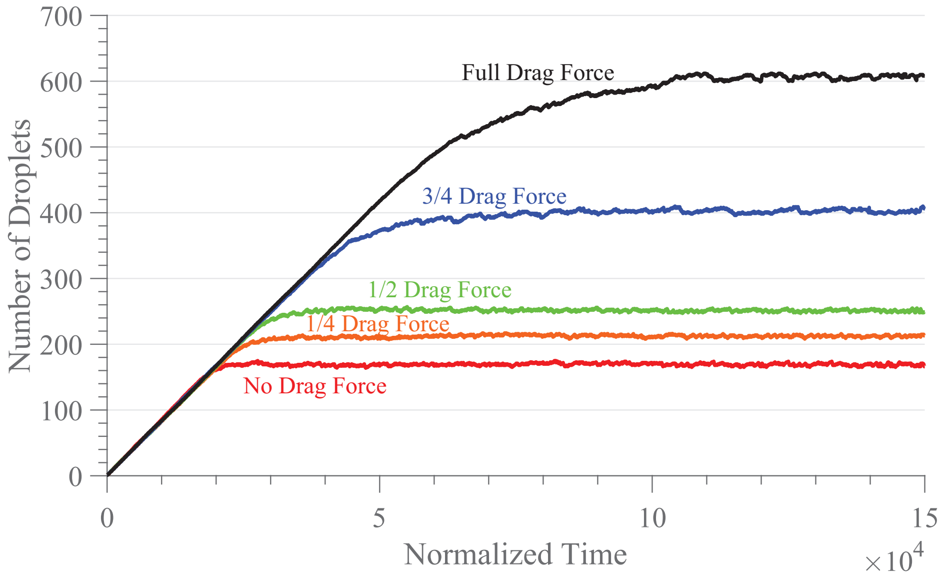


Figure 5.6: The number of particles in the simulation domain over progressing time steps (time normalized with the default time step Δt) for plumes evolved with different levels of applied drag force.

5.2.2 Influence of Background Pressure

In this subsection, the evolution of electrospray plume in different background pressures is compared. The drag force on each particle is calculated based on background pressure using empirical coefficients of drag which are functions of particle Reynolds, Mach, and Knudsen number. Background pressure is an experimentally controllable variable, such that this study is experimentally replicable, unlike the previous study which scaled drag force directly. As in the previous study, the only change in response to different background pressures included in this study is changes in the drag force; further differences in particle emission and subsequent breakup are not included in this study. The domain specifications and emitted particle properties are again that of the validation case in Se. 2.5. The background fluid molecules are air, with radius $r_{fl} = 1.8 \times 10^{-10}\,\mathrm{m}$, molar mass $M_{fl} = 28.97\,\mathrm{g}$, dynamic viscosity $\mu_{fl} = 1.89\,\mathrm{Pa \cdot s}$, and ratio of specific heats $\gamma_{fl} = 1.4$. The set of pressures examined is shown in Table 5.1.

The background pressure determines the density of the background fluid molecules in the simulated electrospray environment,

$$\rho_n = \frac{P}{kT}, \tag{5.9}$$

where k is the Boltzmann constant and T is temperature, which is room temperature in this publication. The number density of the background fluid molecules determines the mean free path of a charged electrospray particle,

$$\lambda = \frac{1}{\rho_n \sigma_{fl}}, \tag{5.10}$$

where σ is the interaction cross-section of the background fluid molecules. For neutral molecules such as the air in the presented simulations, the interaction cross-section is the physical cross-sectional area of the molecule,

$$\sigma_{fl} = \pi {r_{fl}}^2, \tag{5.11}$$

where r_{fl} is the background fluid molecule radius.

P [Atm]	P [Torr]
1	760
0.75	550
0.5	380
0.25	190
0.1	76
0.05	38
0.02	15.2
0.01	7.6
0.005	3.8
0.002	1.52
0.001	0.76
0.0005	0.38
0.0002	0.152
0.0001	0.076

Table 5.1: The range of background pressures utilized for the presented DELI simulation.

As background pressure decreases in an environment, the number density of the background gas decreases according to Eq. 5.9, a process termed rarefaction. In response to this density change, the mean free path of a particle passing through the environment increases following Eq. 5.10. Such rarefaction effects on particle dynamics are represented through a rarefaction parameter called the Knudsen number. The Knudsen number expresses the rarefaction of a flow as the ratio of mean free path to particle diameter:

$$Kn = \frac{\lambda}{d}, \tag{5.12}$$

thereby representing how far, in terms of multiples of its own diameter, a particle will progress on average in an environment before a collision [102, 18, 19, 20, 103]. The literature has established discrete rarefaction regimes for discussion and theory purposes, presented in Table. 5.2 [104, 105, 106, 107].

Knudsen Range	Regime
$Kn < 0.001$	Continuum Flow
$0.001 \leq Kn < 0.1$	Slip Flow
$0.1 \leq Kn < 10$	Transitional Flow
$10 \leq Kn$	Free Molecular Flow

Table 5.2: Collisionality regimes determined based on the rarefaction parameter, Knudsen number.

Following Table 5.2, at high Knudsen numbers, particle trajectories are very rarely altered through collisions with background gas particles, such that they have 'free molecular flow' through the environment. At lower Knudsen numbers in the 'transitional' regime, the background gas particle density is higher and collisions with such background particles have a non-negligible effect on the dynamics of particles passing through the environment. At even lower Knudsen numbers, collisions with background particles have a significant effect, such that passing particles are 'slipping' on the background particles instead of moving through them freely. At the lowest Knudsen numbers, background gas density is so high that background particles fill the environment as a 'continuum' instead of as discrete particles, and collisions with background particles are critical in passing particle dynamics.

The numerical influence of Knudsen number on particle motion is less clear-cut than the theoretical discussion, with the literature presenting a plethora of empirical equations for coefficient of drag[18, 19, 20, 103, 16, 97] as functions of Knudsen number, as well as Mach number,

$$Ma = \frac{|\mathbf{v}|}{a}, \tag{5.13}$$

where a is the speed of sound in the environment, and Reynolds number, which expresses the ratio of inertial to viscous forces:

$$Re = \frac{|\mathbf{v}|d}{\nu_{fl}}, \tag{5.14}$$

where $\mathbf{v}$ is particle velocity, d is particle diameter, and ν_{fl} is the kinematic viscosity of the background gas,

$$\nu_{fl} = \frac{\mu_{fl} A}{\rho_n M_{fl}}, \tag{5.15}$$

where M_{fl} is the molar mass of the background gas molecules, μ_{fl} is the dynamic viscosity, and A is Avogadro's number. Note that Mach number, Reynolds number, and Knudsen number are analytically related via

$$Kn = \frac{Ma}{Re}\sqrt{\frac{\gamma\pi}{2}}, \tag{5.16}$$

where γ is the ratio of specific heats, such that range limits on two flow parameters limits the third flow parameter. Empirical coefficient of drag equations are applicable only within the range of flow parameters over which they were developed. Appendix A presents the empirical drag terms utilized in the presented DELI simulations, their flow parameter ranges, and their source publications. Two sets of DELI simulations were run: one using the [16] coefficient of drag for all particles, and one using the coefficients of drag from [17], [18], [19], and [20] depending on particle Re and Kn. Below $P = 0.007692$ Atm, mean particle Knudsen number exceeds one and the only applicable coefficient of drag in Table A.1 is from [16]. Therefore, only the [16] coefficient of drag term was used to simulate the $P = 0.005, 0.002, 0.001, 0.0005, 0.0002$ and 0.0001 Atm cases.

For the high Kn Transitional Flow range ($Kn > 1$) and the Free Molecular Flow range ($Kn \geq 10$) there are few empirical coefficient of drag terms in the literature because the collisions between particles and background fluid molecules are infrequent, stochastic exchanges of momentum range than a continuous loss of particle momentum to drag. When simulating this environment in the absence of an empirical drag term, background fluid molecules could be randomly dispersed in the simulation domain with the number density set as a function of background pressure following Eq. 5.9, and momentum exchanges from collisions between particles and background fluid molecules could be included in particle propagation. This effort is beyond the scope of this publication, but could extend the present study to lower pressures in the future.

The coefficient of drag is utilized to calculate the drag force exerted on each particle,

$$\mathbf{F_D} = -\frac{\pi}{8} d^2 \rho_f C_D \mathbf{v} |\mathbf{v}|, \tag{5.17}$$

where the density of the surrounding gas can be calculated using the number density given by Eq. 5.9,

$$\rho_{fl} = \rho_n \left(\frac{M_{fl}}{A} \right). \tag{5.18}$$

Electrospray particles in the DELI Model are propagated downstream from emission under electrostatic, Coulomb, image charge, and drag forces:

$$\begin{aligned} m\frac{d^2\mathbf{x}}{dt^2} &= q(\mathbf{E_A} + \mathbf{E_C} + \mathbf{E_I}) - \mathbf{F_D} \\ &= q\mathbf{E_A} + \frac{q}{4\pi\epsilon_0} \sum_i^n q_i \left(\frac{\mathbf{r_i}}{|\mathbf{r_i}|^3} - \frac{\mathbf{r_{Ii}}}{|\mathbf{r_{Ii}}|^3} \right) - C_D \frac{\pi}{8} \rho_{fl} d^2 \mathbf{v} |\mathbf{v}|, \end{aligned} \tag{5.19}$$

where $\mathbf{E_A}$ is the applied electric field, $\mathbf{E_C}$ is the Coulomb electric field generated by the charged plume particles, $\mathbf{E_I}$ is the electric field created by image charges induced in the collector plate, ϵ_0 is the permittivity of vacuum, n is the total number of particles in the plume, and $\mathbf{r_i}$ is the separation vector between particles ($\mathbf{r_I}_i$ is the separation vector from each image charge, which has axial component $2H - z_i$, where H is the collector plate height and z_i is the axial coordinate of the particle yielding the image). Image charges are also generated in the emitter and the fluid meniscus, but these are geometrically complex and not included in the presented simulations. The flowchart for the DELI Model, including the selection of an empirical drag term in each timestep based on the Kn, Re, and Ma of each particle, is presented in Fig. 5.7.

Simulation results successfully replicate the experimentally established trend of decreasing plume divergence with decreasing background pressure. Fig. 5.8 presents the outlines

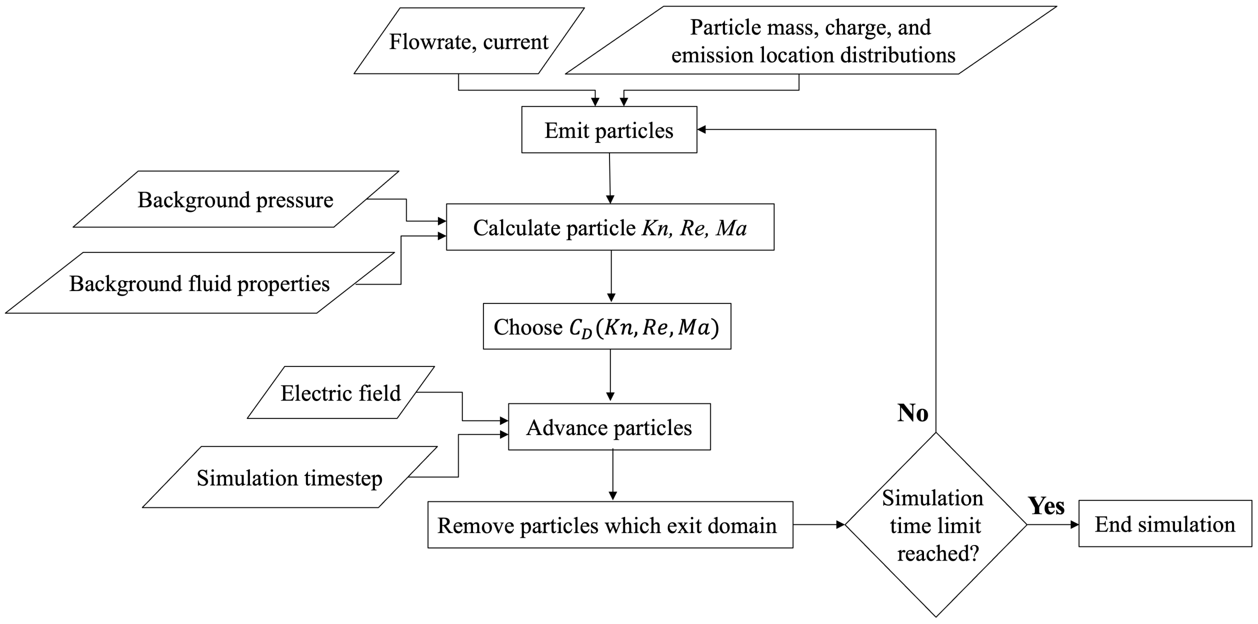

Flowrate, current
Particle mass, charge, and emission location distributions
Emit particles
Background pressure
Calculate particle Kn, Re, Ma
Background fluid properties
Choose $C_D(Kn, Re, Ma)$
Electric field
Advance particles
No
Simulation timestep
Remove particles which exit domain
Simulation time limit reached?
Yes
End simulation

Figure 5.7: The DELI Model algorithm for electrospray plume evolution with drag calculated as a function of particle Reynolds, Knudsen, and Mach number.

of plumes simulated over the range of background pressures given in Table 5.1. Square root functional forms were fit to each plume outline and the fit coefficients are plotted against pressure in the inset plot of Fig. 5.8. There is a clear decrease in the coefficient of the outline functional fit with background pressure, demonstrating the narrowing of the plume outlines as pressure is decreased.

The correlation between background pressure and plume divergence is further analyzed in Fig. 5.9, in which the terminal angles for one standard deviation (containing 68% of particles), three standard deviation (containing 99.7%), and outline (containing 100%) measurements of particle number density are plotted against background pressure. Results are shown for simulations using a single coefficient of drag from [16] at pressures, and for simulations using multiple coefficients of drag from [17], [18], [19], and [20] depending on particle Re and Kn at the given pressure. The standard deviation measurements were obtained by dividing the plumes axially into sections and radially fitting a Gaussian profile to particle number density distributions in each section. The terminal angle is defined as the polar

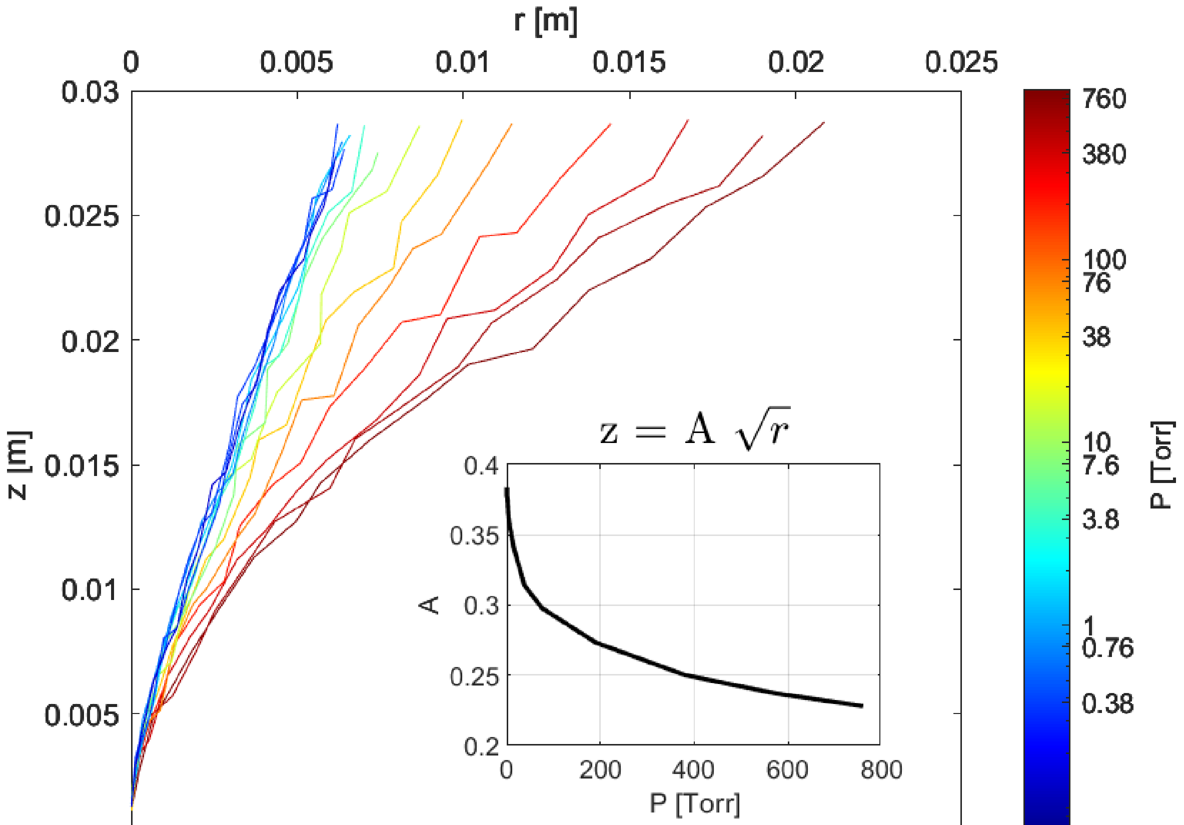


Figure 5.8: The outline of plumes simulated with different background pressures with color according to background pressure on a logarithmic scale. The inset plot shows the coefficient of a square root functional fit to the plume outline at each background pressure.

angle measurement,

$$\theta = \tan\left(\frac{\sqrt{x^2+y^2}}{z}\right), \tag{5.20}$$

where x, y, and z are the 3D positional coordinates upon reaching the collector plate at the downstream end of the simulation domain. The outline terminal angle in the atmospheric case is consistent with previous publications [10, 11, 46]. In all cases, the angles of particle velocity vectors at the collector plate was wider than their positional angles, implying that the plumes would expand further in wider domains.

In Fig. 5.9, there is a significant angular difference between the 3σ and outline measurements, especially at high pressure levels. The terminal 3σ measurement displays the angle within which all but outlier particles are incident on the collector plate, so very few outlier particles account for a large range of wide incidence angles; 0.3% of particles account for the widest 12° of incidence angles for the simulated plumes at near-atmospheric background pressures. Therefore, the 3σ measurement may be a more accurate measurement of evolved plume shape than the outline measurement, which is dictated by outliers.

The single and multiple coefficient of drag results agree within 3° for the one standard deviation, three standard deviation, and outline particle number density terminal angles for all pressures in the range of both coefficient of drag approaches. In both cases, plume divergence continuously decreases with decreasing background pressure for all three angular measurements in Fig. 5.9, although the changes are more significant for the outline and 3σ measurements than for the 1σ measurements. Hence, the trajectories of particles on the fringe or edge of the plume are more impacted by changes in background pressure than those of particles in the central majority of the plume. This is physically reasonable, as particles on the plume edges experience outwardly radial, expansion-inducing Coulomb forces from particles in the plume center with only the drag force countering divergence. On the other hand, particles in the plume center experience inwardly radial, counter-expansion forces from both the drag force and from Coulomb forces from particles wider in the plume. The only case in which particles do not exert radial Coulomb forces on one another is if they are emitted in a perfectly straight line; electrospray particles are always emitted within some non-zero radial position range due to hydrodynamic instabilities and microscopic asymmetries in emitter geometry. Therefore, the divergence of particles on the plume edge is more impacted by the drag force, and thereby background pressure, than that of particles in the plume center.

Including reference thresholds for Knudsen number of the mean particle at emission in Fig. 5.9 shows that the simulations span the Slip, Transitional, and Free Molecular Flow regimes. When $P < 5.87\,\text{Torr}$, the average-sized particle ($d = 38\,\mu\text{m}$) has $Kn > 1$ and is approaching the Free Molecular Flow regime where dynamics are not longer affected by collisions with background air molecules. In the simulation cases below this pressure, the outlines in Fig. 5.8 and the angular measurements in Fig. 5.9 display that the plume structure plateaus to a near-constant shape. For $P \leq 0.76\,\text{Torr}$, corresponding to $Kn \geq 7.7299$ for the mean particle, the plume is not any further narrowed by additional reductions in background pressure. Collisions between electrospray particles and background air molecules happen so infrequently at these pressures that they have negligible impact on resulting particle trajectories. At pressures below $P = 0.76\,\text{Torr}$, the divergence of the plume is solely a result of the applied electrostatic force and inter-particle Coulomb forces rather than the drag force.

A final metric for observing changes to plume structure corresponding to changes in background pressure is the number of particles in the plume at steady state. This metric is defined as the number of particles in the plume when this population asymptotes after increasing during plume startup. A similar trend is observed for number of particles at steady state with background pressure as for plume divergence: the number of particles at steady state decreases as background pressure decreases until $P = 0.76\,\text{Torr}$, at which point the number of particles at steady state remains constant even with further drops in pressure. This pressure threshold for the plateau in number of particles at steady state is the same as for the plateau in plume divergence. At high background pressures, a high density of stationary background air molecules collide with moving ES particles, slowing their progress through

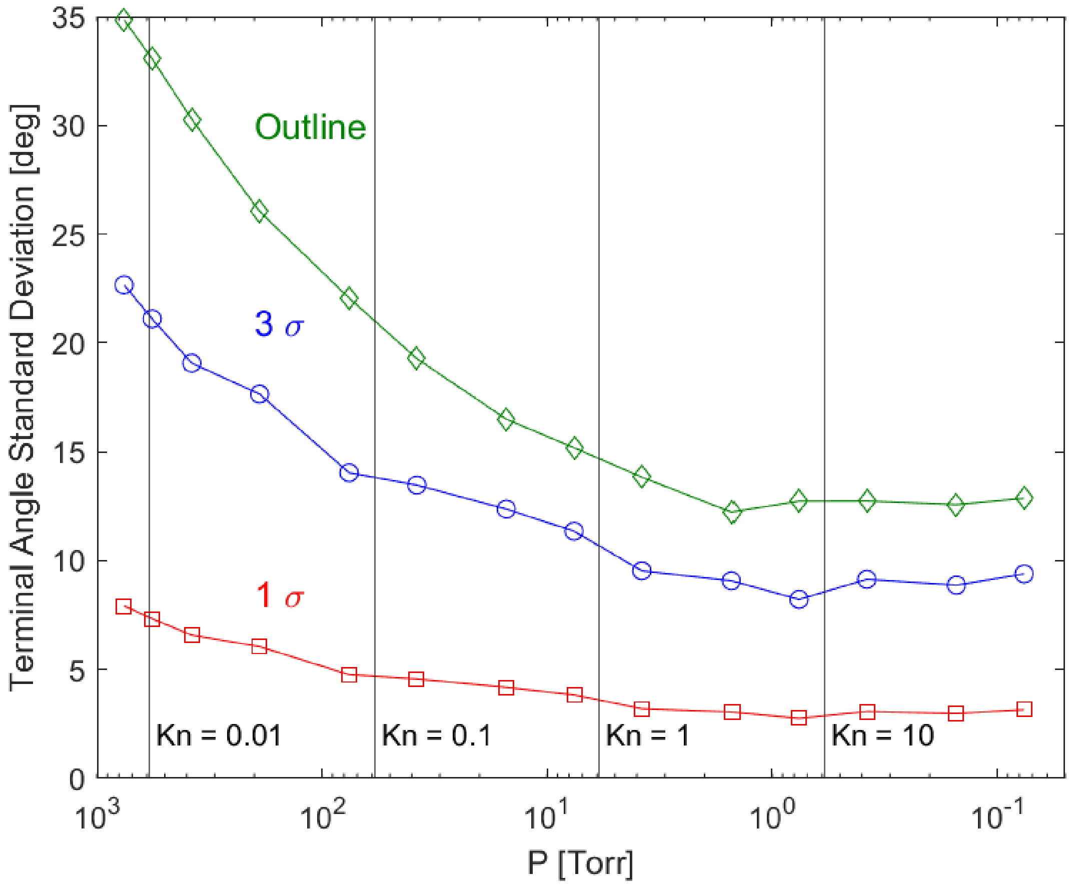


Figure 5.9: Terminal angle measurements at the collector plate location are given for 1 and 3 standard deviations of particle number density and the outline of plumes simulated at different background pressures using a single coefficient of drag from [16] (solid lines) and multiple coefficients of drag from [17], [18], [19], and [20] depending on particle Re and Kn (dashed lines).

the domain with drag force. This slowed motion towards the collector plate causes a higher number density of particles in the plume domain which repel one another Coulombically. In this manner, the same physical process which correlates increased background pressure with an increased number of particles in the plume at steady state also correlates increased background pressure with increased plume divergence. The number of particles in the plume at steady state for pressures at and below $P = 0.76\,\text{Torr}$ is equal to the result obtained for the plume simulated with no drag force ($F_D = 0$) [46, 108], confirming that drag is negligible at these pressure and no changes will occur to plume shape through further reductions in background pressure.

Having investigated the relationship between background pressure and plume divergence computationally with DELI Model simulations, we now present an analytical determination of the background pressure for minimized electrospray plume divergence. For many electrospray technologies, optimizing performance requires minimizing plume divergence. For example, the operational lifetime of electrospray thrusters for spacecraft propulsion depends on avoiding propellant deposition on downstream electrodes at wide plume angles, and thrust control also is maximized by a minimally divergent plume [3, 85, 23, 45]. When operating electrospray technologies desiring minimal plume divergence, the background pressure needs to be sufficiently low that drag no longer significantly contributes to plume divergence. This pressure threshold for no drag divergence for the electrospray particles occurs by conceptual definition in the Free Molecular regime with $Kn = 10$. Combining equations 5.9 - 5.12, the analytical form of this pressure threshold is

$$P = \frac{k_B T}{10\sigma_{fl} d}. \tag{5.21}$$

The simulated plume presented in Fig.s 5.8 and 5.9 was observed to exhibit minimal divergence for $P \leq 0.76\,\text{Torr}$, where the mean emitted particle is in the Transitional Flow regime with $Kn \geq 7.7299$. Note that this Kn threshold is very near the analytical Free Molecular threshold. Based on our simulation results, we phenomenologically define a pressure threshold for minimum electrospray plume divergence to be

$$P_{th} \equiv \frac{k_B T}{7.7299 \sigma_{fl} d}. \tag{5.22}$$

Eq. 5.22 provides a pressure limit for an environment shown computationally to sufficiently resemble full vacuum for electrospraying the presented plume. The Knudsen number threshold of $Kn = 7.7299$ is based on a specific size of mean particle and the equation is therefore plume-dependent. Equation 5.21 provides a stricter pressure limit for plume confinement for plumes of any particle size; this threshold is not plume-dependent because it is analytically determined based on the definition of the Free Molecular regime. Fig. 5.10 plots both pressure limits for drag-free electrospray operation for different particle sizes spanning the range used in electrospraying.

The cross-section of the background fluid molecule in Eq. 5.21 is a general cross-section term which can be specified to charged or neutral molecules for any background fluid. Electrosprays generally operate in domains with neutral background fluid molecules, such as air. However, when neutral molecules are placed in an electric field, they become polarized, with the positive and negative charges pushed to opposite sides of the molecule. Charged electrospray particles create local electric fields, such that they induce dipoles in neutral background molecules and are then attracted to the oppositely-charge end of the dipoles [109, 110, 111]. This ion-induced dipole attraction increases the cross-section of the background fluid molecule and introduces a charge-dependence to the interaction between the electrospray particle and the background fluid molecule.

Thus far in this section, drag has been discussed as the effect of physical collisions between electrospray particles and background fluid molecules. Depending on Knudsen number, particles move through a Continuum Flow of background molecules, or have sparse collisions in an otherwise Free Molecular Flow environment. In this view of drag, electrospray particles only exchange momentum with background molecules with which they physically collide. However, when long-range polarization forces are considered, electrospray particles have electrostatic interactions with the induced dipoles in all background molecules [109, 110, 111].

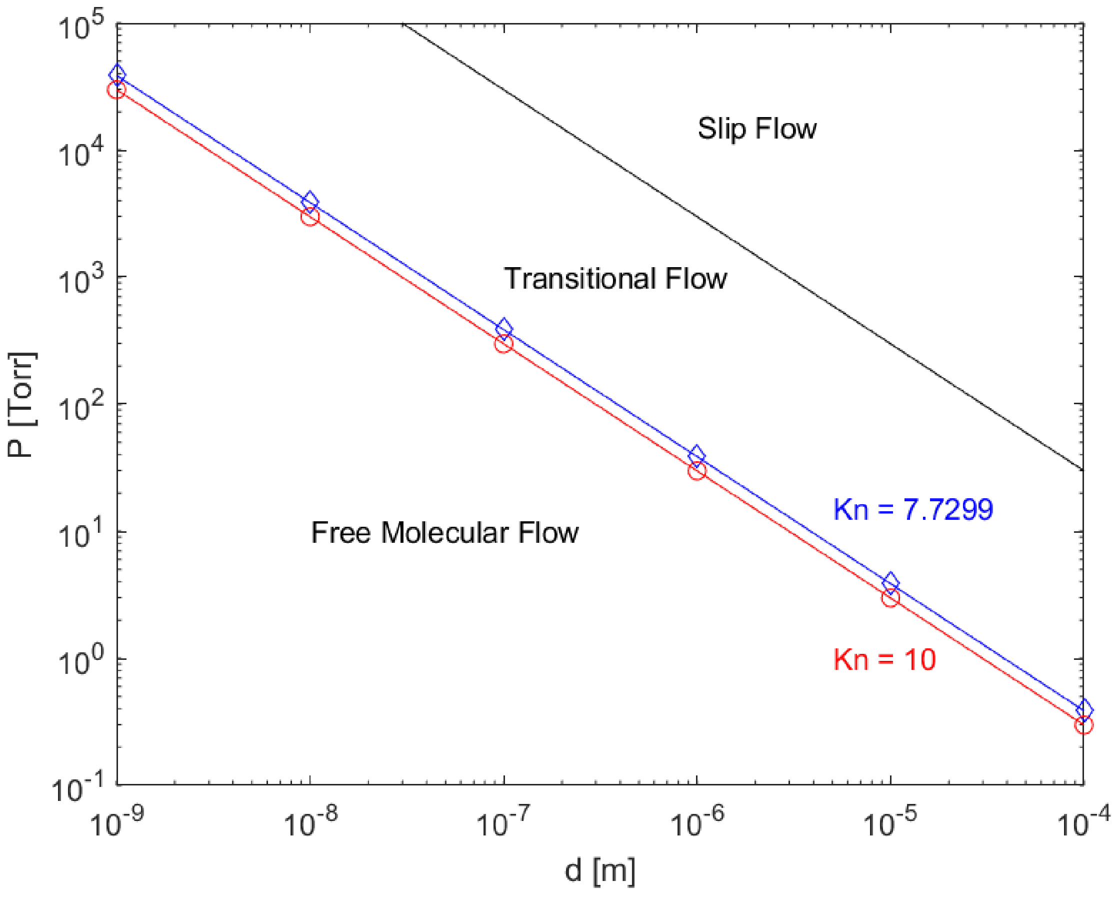


Figure 5.10: The pressure limit, P, for drag-free plume evolution for a range of particle sizes as bounded by the Free Molecular flow definition from Eq. 5.21 and the phenomenological pressure threshold from Eq. 5.22.

Electrospray particles exchange momentum with all the background molecules, not only those with which they physically collide, as they do with all the other charged electrospray particles.

Molecular dynamics simulations have observed polarization forces to significantly impact the ion mobility,

$$K = \frac{v_d}{E}, \tag{5.23}$$

where v_d is drift velocity and E is electric filed strength, of singly-charged ions in air with diameters less than 1.3 nm [110]. Electrospray particles can be much higher in charge; for example, large droplets can contain thousands of single charges. Therefore, while long-range polarization forces are too computationally intensive to include in current electrospray plume simulations, they should be anticipated to occur in the background fluids of electrosprays. These attractive forces between electrospray particles and neutral background gas molecules increase the overall force which the molecules exert on the particles, thereby increasing drag, slowing particle motion, and increasing particle density in the domain. This increase in charge density causes increased plume divergence through Coulomb interactions, such that gas polarization increases plume divergence.

The Free Molecular pressure threshold for eliminating drag effects presented in Eq. 5.21 and plotted in Fig. 5.10 is applicable beyond electrosprays to any particulate flow system, including particle accelerators [112, 113], aerosol drug delivery systems [114, 115, 34], ablative beams [116, 117], and other electric propulsion systems like Hall thrusters [118, 119, 120, 121]. Performing this pressure threshold analysis allows for efficient electrospray or other experimentation in vacuum with minimal investment in high-vacuum systems as it analytically determines the background pressure required for negligible drag influence prior to experiment construction.

5.3 Drag Paper Collision Analysis

An analysis of collisions between charged electrospray plume particles and the background neutral air molecules present in the electrospray domain is presented in this section. Secondary induced flow of the background gas is not considered in this analysis; the neutral air molecules are considered stationary in the laboratory frame, while the charged electrospray particles are moving. Given the pressure, P, and temperature, T, in an electrospray domain, the number density of air molecules in the domain can be calculated through the ideal gas law:

$$\rho_n = \frac{P}{k_B T}, \tag{5.24}$$

where k_B is the Boltzmann constant. The mean free path, l, or average distance traveled by a charged electrospray particle before a colliding with a stationary background neutral air molecule, through a domain with number density ρ_n of such stationary particles, is given by Eq. 2.6. The cross section of the neutral air molecules is given by Eq. 5.11. Given a path length of interest, L, such as the emitter-to-collector-plate or emitter-to-extractor-electrode distance in an electrospray geometry, the domain Knudsen number can be calculated following Eq. 5.12 as the ratio of the mean free path to the domain length:

$$Kn_{dom} = \frac{l}{L} = \frac{1}{L \rho_n \pi r_{fl}^2}. \tag{5.25}$$

Note that this Knudsen number is defined with a different characteristic length than was done for coefficient of drag purposes, which utilized the particle diameter as the scaling length L instead of the domain length. The objective of present collision analysis is to find the pressure at which the mean free path of an electrospray particle exceeds the particle path length in a given geometry, such that collisions with air molecules can be neglected from the analysis of particle dynamics in the full domain.

Figure 5.11 displays the relationship between the domain Knudsen number and background pressure for multiple path lengths, L. The path lengths considered range from $L = 1\,\mu\text{m}$, smaller than electrospray thruster emitter-to-extractor-electrode distances, to

$L = 10\,\mathrm{m}$, the size of a large vacuum chamber, such that the presented results are applicable to a wide range of technology geometries. A dashed line is shown representing the emitter-to-collector-plate distance, H, from Sec. 2.5, with a star marking the atmospheric case presented in that section. The Knudsen number for the atmospheric validation case falls well below unity, meaning that collisions with background air molecules have a significant effect on electrospray particle dynamics; this conclusion aligns with the notable effects of the drag force on particle dynamics observed in Sec. 2.5. The pressures considered range from atmospheric pressure to ultra-high vacuum, thereby including pressures relevant to many industry and scientific applications. Using this figure, given the path length, L, of an electrospray application, the pressure at which the mean free path of an electrospray particle exceeds the relevant domain length can be determined, as given by the unity Knudsen number threshold. Alternatively, given the pressure level, P, that can be obtained in a given electrospray apparatus, the domain length which yields a particle mean free path greater than that length can be determined, such that the electrospray geometry may be scaled to avoid collisions between charged electrospray particles and background neutral air molecules.

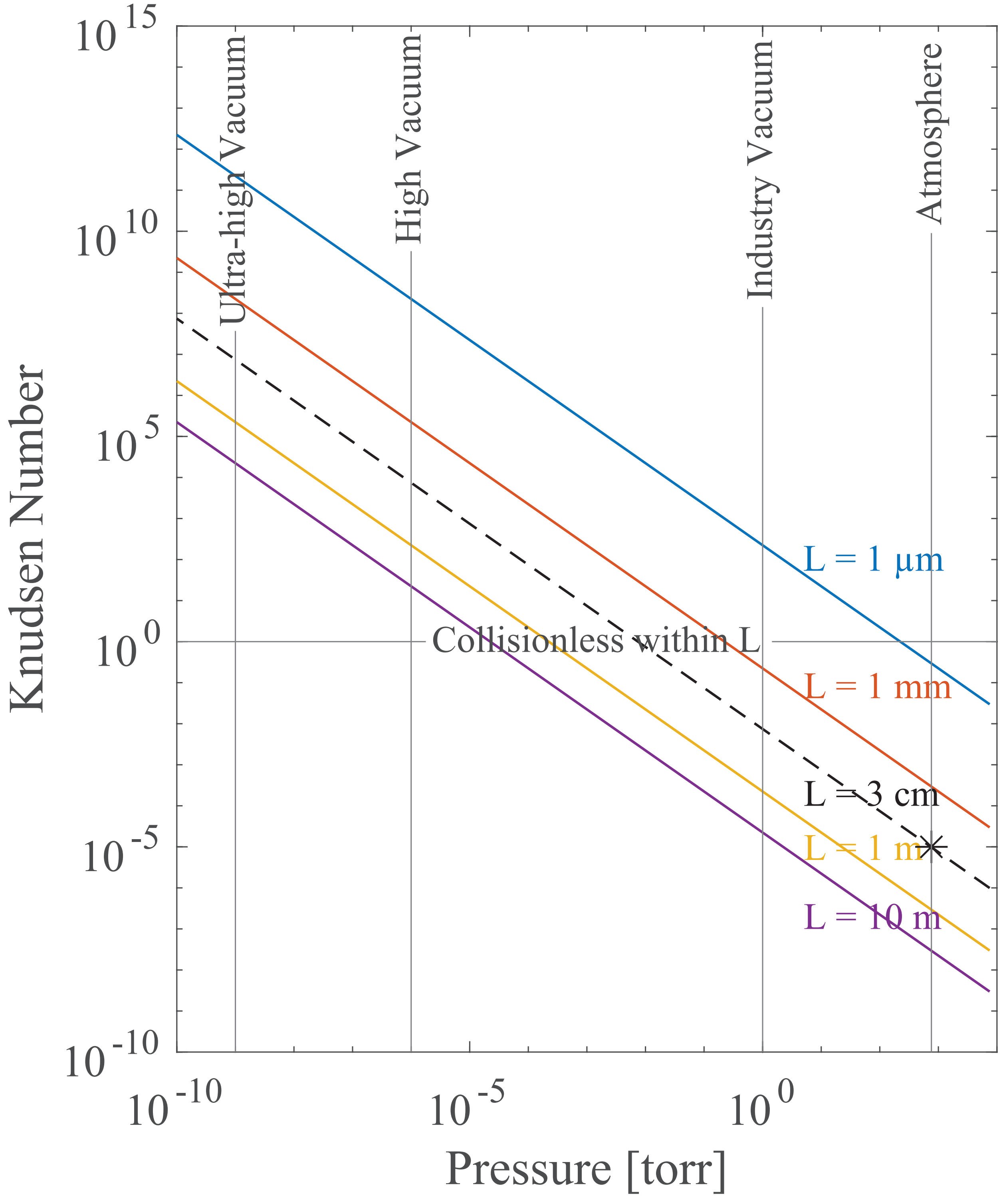


Figure 5.11: The relationship between pressure and Knudsen Number is presented for different path lengths.

CHAPTER 6

Other Sources of Plume Divergence

The previous three chapters have been devoted to each of the three dominant forces in electrospray plume dynamics: the applied electrostatic force, the inter-particle Coulomb forces, and the drag force. These forces are represented in the governing equation for the particle propagation DELI Model, Eq. 2.11. This chapter is devoted to forces and phenomena which have a lesser impact on plume evolution and are therefore neglected from DELI simulations. The objective of discussing these higher-order plume evolution terms is to provide a comprehensive view of plume divergence beyond current DELI Model simulation capabilities.

6.1 Gravity

The gravitational forces acts on all electrospray particles according to their mass,

$$\mathbf{F_G} = m\mathbf{g}, \tag{6.1}$$

where **g** is the Earth's gravitational acceleration. Order of magnitude analysis revealed the gravitational force to be less than 1% than the applied electrostatic, Coulomb, and drag forces for the validation case in Sec. 2.5, such that it can be neglected without introducing significant error to simulated plume evolution. Gravity acts downward, such that its influence on electrospray particles depends on the orientation of the electrospray setup. The effect of the gravitational forces on an electrospray plume in various orientations is displayed in Fig. 6.1. If the electrospray faces upward, gravity works counter to the axial forward motion of the electrospray particles. This causes slowed axial velocities and thereby increased particle

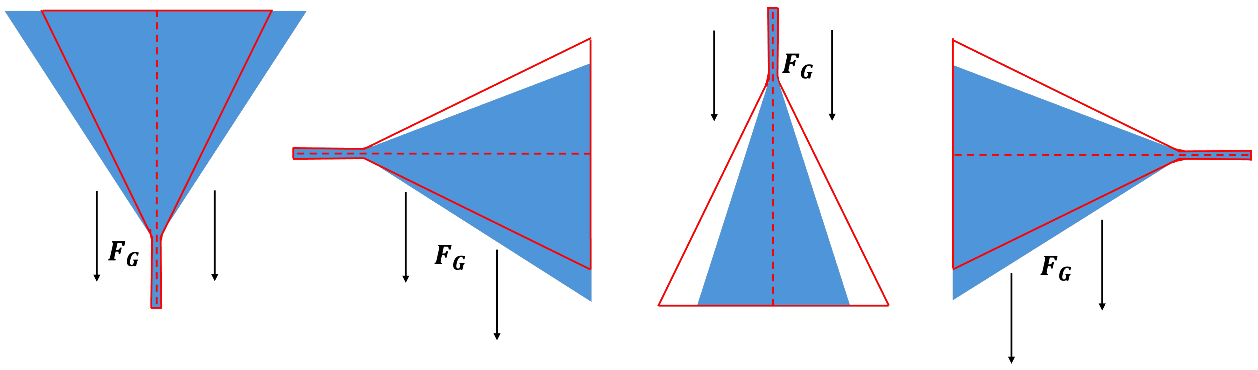


Figure 6.1: The influence of gravity on electrospray plumes in different orientations. The plume outline without gravity is shown in red, the gravitational force is denoted with black arrows, and the modified plume shape under the influence of gravity is shown in blue.

residence time in the domain, resulting in a higher particle number density within the plume. As discussed in Sec. 5.2, increased particle number density causes increased plume divergence through Coulomb interactions. Therefore, orienting the electrospray to emit against gravity causes gravity to contribute to plume divergence. On the other hand, if the electrospray is facing down such that particles are emitted in the direction of the gravitational force, then gravity accelerates particles in the direction of their inertial axial motion. In this case, particles move more quickly and spend less time in the domain, allowing for less plume divergence through Coulomb interactions as described in Sec. 4.3. In the case in which electrospray plumes face to the side, at a 90° angle to the gravitational force, the plume will experiences 'skewing' or 'tilting' as particles are pulled to one side. The plume will not maintain axisymmetry around the axis of emission. Plume emission at an angle which is not fully parallel or perpendicular to the gravitational force will yield a combination of the effects described here for the cases with emission parallel and perpendicular to gravity.

This 'tilting' of the electrospray plume when oriented perpendicular to gravity has been experimentally observed[122, 85, 123]. In order to include the gravitational force in DELI simulations to reproduce such behavior, the governing Eq. 2.11 would need to be modified to include the gravitational force:

$$\begin{aligned} m\mathbf{a} &= q(\mathbf{E_A} + \mathbf{E_C} + \mathbf{E_I}) - \mathbf{F_D} + \mathbf{F_G} \\ &= q\mathbf{E_A} + \frac{q}{4\pi\varepsilon_0}\sum_i^n q_i\left(\frac{\mathbf{r_i}}{|\mathbf{r_i}|^3} - \frac{\mathbf{r_I}}{|\mathbf{r_I}|^3}\right) - C_D\frac{\pi}{8}\rho_g d^2\mathbf{v}|\mathbf{v}| - m\mathbf{g}. \end{aligned} \tag{6.2}$$

6.2 Thermal Gradient

The thermophoretic force acts on particles due to temperature gradients in the background fluid. Background gas molecules in warmer areas have higher kinetic energies than those in colder areas, such that they transfer more energy in collisions with particles. This causes a net thermophoretic force on the particles towards colder areas. In continuum flows with $Kn \leq 0.001$, the thermophoretic force is

$$\mathbf{F_T} = -\frac{6\pi d\mu_{fl}^2 C\Lambda\nabla T}{\rho_{fl}(2\Lambda+1)T}, \tag{6.3}$$

where T is the local background fluid temperature, ∇T is the local temperature gradient in the background fluid, $C = 1.17$ is a dimensionless constant, μ is the dynamic viscosity of the background fluid, ρ is the density of the background fluid, and

$$\Lambda = \frac{k_{fl}}{k_p}, \tag{6.4}$$

where k_{fl} is the thermal conductivity of the background fluid and k_p is the thermal conductivity of the particle. To include the thermophoretic force in DELI simulations, the governing Eq. 2.11 would be:

$$\begin{aligned} m\mathbf{a} &= q(\mathbf{E_A} + \mathbf{E_C} + \mathbf{E_I}) - \mathbf{F_D} + \mathbf{F_T} \\ &= q\mathbf{E_A} + \frac{q}{4\pi\varepsilon_0}\sum_i^n q_i\left(\frac{\mathbf{r_i}}{|\mathbf{r_i}|^3} - \frac{\mathbf{r_I}}{|\mathbf{r_I}|^3}\right) - C_D\frac{\pi}{8}\rho_g d^2\mathbf{v}|\mathbf{v}| - \frac{6\pi d\mu^2 C\Lambda\nabla T}{\rho(2\Lambda+1)T}. \end{aligned} \tag{6.5}$$

To include rarefaction effects on the thermophoretic force when the flow is not in the continuum regime, empirical formulas have been developed[124].

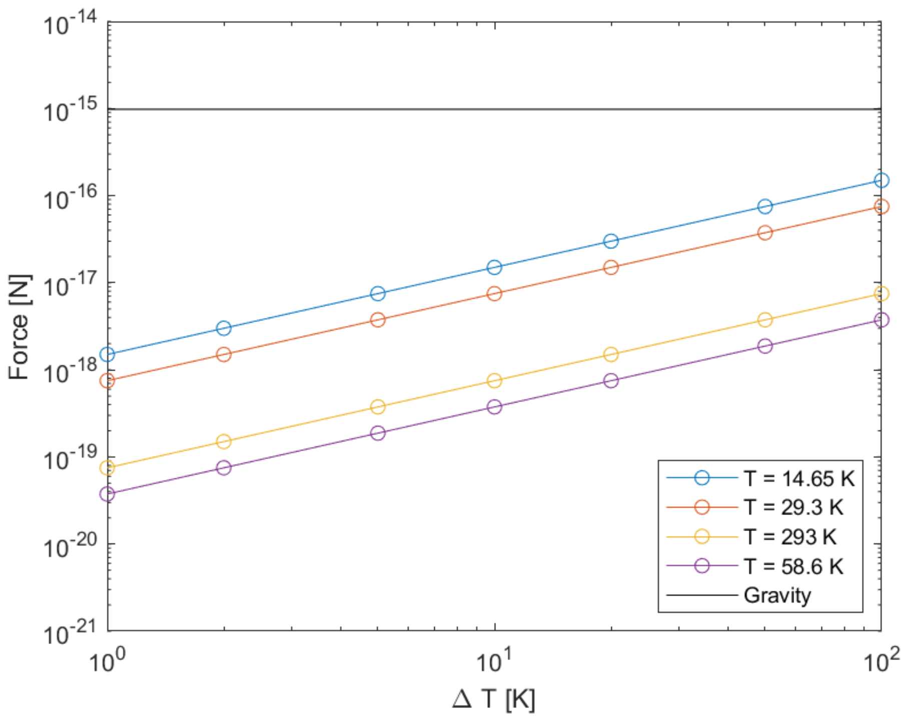


Figure 6.2: The thermophoretic force on a micron diameter particle for different local background fluid temperatures T and temperature gradients ∇T. The gravitational force on a $m = 1 \times 10^{-16}$ kg particle is included as for reference.

Fig. 6.2 displays the thermophoretic force on a micron diameter particle in different background fluid temperature gradients and local background fluid temperatures. A line showing the gravitational force on a particle with $m = 1 \times 10^{-16}$ kg (the same order as the mean particle in the validation case in Sev. 2.5) is included for reference. The thermophoretic force is less than the gravitational force in all cases, which has already been noted in Sec. 6.1 to be negligible compared to the applied electrostatic, Coulomb, and drag force.

Thermal gradients can exist inside of the electrospray jet and fluid electrospray droplets in addition to throughout the thruster domain. Thermal gradients within the electrospray jet have been reported in simulations of jet formation and particle emission[125], with the jet tip being hotter than the length of the jet. This thermal gradient provides additional kinetic energy to particles emitted from the tip. Thermal gradients within particles affect charge motion within the particle, which is discussed in the following Sec. 6.3.

6.3 Fluid Mechanics

The DELI Model approximates electrospray particles to be perfectly spherical and uniformly charged, but in reality they are fluid collections of mobile charges. Interactions between charges within the particle, and interactions with mobile charges between particles cause charges to move around and deform the fluid surface which contains them. Charge mobility causes the Coulomb interactions between particles to vary from the theoretical ideal for perfect spheres. Polarization alters the Coulomb interaction between aqueous droplets by inducing multipoles[21]. As a result, charges of one sign within a particle move to be closer to charges of the opposite sign within neighboring particle, increasing their electrostatic attraction or reducing their electrostatic repulsion. This process is displayed in Fig. 6.3, reproduced from [21].

Deforming from the spherical ideal also causes 'primary' particles emitted from the jet to emit 'secondary' progeny particles through Coulomb fission when the primary particle charge is below the Rayleigh limit for spherical particles of the same size. Fluid electrospray particles have been observed to undergo Coulomb fission at less than 70% of the Rayleigh charge limit for spherical particles of the same size[126, 127, 128]. Furthermore, primary particles can emit secondaries through field-emission of ions where the local surface electric field exceeds $1\,\mathrm{V\,nm^{-1}}$[27]. Electric field strength is magnified in areas of high curvature, such that particles which deform in a highly curved manner may emit ions when they would not have done so in a non-deformed, spherical state. The mass, charge, and number of progeny produced during particle breakup, through Coulomb fission or field-emission of ions, is highly variable and an area of active research[129]. Coulomb fission has been included in Lagrangian simulations with the ranges for mass, charge, and number of secondaries based on experi-

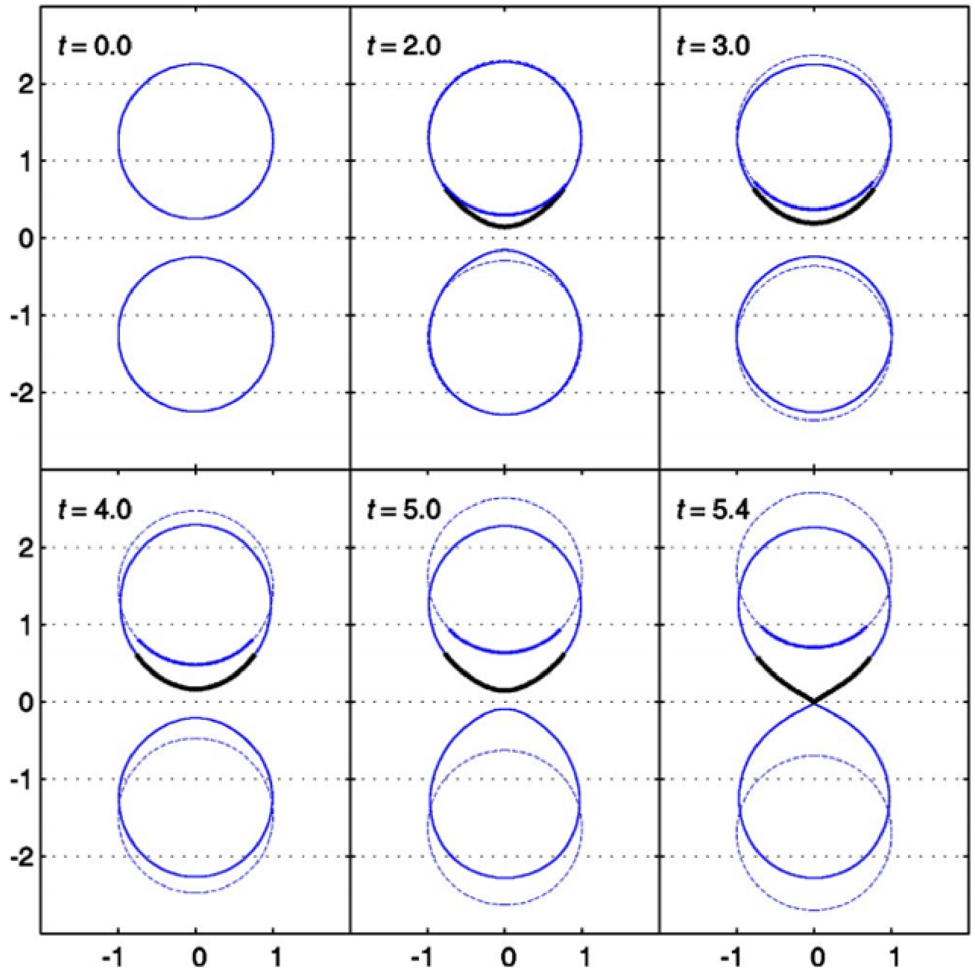


Figure 6.3: Spherical fluid particles have initial separation with 2.5 particle radii experience local attraction to one another and coalesce. The black line on the bottom of the lower particle represents negative charge concentration. The dashed lines show the motion of rigid particles for comparison [21]

mental observations[26, 10]. Particle breakup is currently restricted to molecular dynamics simulations of individual of small numbers of particles; such calculations are currently too computationally intensive for full electrospray plume simulations.

In addition to particle breakup, fluid electrospray particles are capable of coalescing. Coalescence events include full particles merging together, and partial transfers of groups of charges from one particle to another. Similar to particle breakup, particle fluid coalescence is currently too computationally expensive to include in full plume simulations. Molecular dynamics simulations of coalescing particles can provide statistical outputs on the particle properties resulting from particle coalescence which can be input to full plume simulations for reference during simulated particle collisions.

6.4 Gas Polarization

Electrosprays generally operate in domains with neutral background fluid molecules, such as air. However, when neutral molecules are placed in an electric field, they become polarized, with the positive and negative charges pushed to opposite sides of the molecule. Charged electrospray particles approach create local electric fields, such that they induce a dipole in the neutral background molecules and are then attracted to the oppositely-charge end of the dipole. This ion-induced dipole attraction introduces a charge-dependence to the interaction between the electrospray particle and the background fluid molecules which is not represented in conventional drag force equations such as that presented in Ch. 5 [109, 110, 111].

In Ch. 5, drag is discussed as the effect of physical collisions between electrospray particles and background fluid molecules. Depending on Knudsen number, electrospray particles can move through a Continuum Flow of background molecules, or have sparse collisions in an otherwise Free Molecular Flow environment. Electrospray particles only exchange momentum with background molecules with which they physically collide in this view of drag. However, when long-range polarization forces are considered, electrospray particles have electrostatic interactions with the induced dipoles in all background molecules [109, 110, 111]. Electrospray particles exchange momentum with all the background molecules, not only those with which they physically collide, as they do with all the other charged electrospray particles.

Molecular dynamics simulations have observed polarization forces to significantly impact the ion mobility,

$$K = \frac{v_d}{E}, \tag{6.6}$$

where v_d is drift velocity, of singly-charged ions in air with diameters less than 1.3 nm [110]. Electrospray particles can be much higher in charge; for example, large droplets can contain thousands of single charges. Therefore, while long-range polarization forces are too computationally intensive to include in current electrospray plume simulations, they should be anticipated to occur in the background fluids of electrosprays. These attractive forces between electrospray particles and neutral background gas molecules increase the overall force which the molecules exert on the particles, thereby increasing the effect of drag. As described in Ch. 5, the counter-motion effect of drag slows particles motion through the domain, thereby increasing particle density in the domain. This increase in charge density causes increased plume divergence through Coulomb interactions as described in Ch. 4. Therefore, gas polarization contributes positively to plume divergence.

CHAPTER 7

Novel Methods for Characterizing Plume Divergence

While divergence is a key metric for electrospray lifetime and also performance, the community has yet to converge on a metric for electrospray plume divergence. Means of presenting simulated or experimentally observed electrospray plume divergence in the literature include plume outlines and 2-dimensional snapshots of the plume, which may be instantaneous or time-averaged. The electrospray plume outline is dictated by outlier particles, which can reach much wider angles than the majority of plume particles. Figure 5.9 presents the terminal angles on the downstream collector plate reached by one standard deviation (containing 68% of particles), three standard deviations (containing 99.7%), and the outline (containing 100%) of particle number density for various background pressures. The standard deviation measurements were obtained by dividing the plumes axially into sections and radially fitting a Gaussian profile to particle number density distributions in each section. A Super-Gaussian fit could also be used in cases for which it fits better to the particle mass density distribution[23]. The terminal angle is defined as the polar angle measurement,

$$\theta = \tan\left(\frac{\sqrt{x^2 + y^2}}{z}\right), \tag{7.1}$$

where x, y, and z are the 3D positional coordinates upon reaching the collector plate at the downstream end of the simulation domain. In Fig. 5.9, there is a significant angular difference between the 3σ and outline measurements, especially at high pressure levels. The terminal 3σ measurement displays the angle within which all but outlier particles are incident on the collector plate, so very few outlier particles account for a large range of wide incidence angles; 0.3% of particles account for the widest 12° of incidence angles for the simulated plumes at near-atmospheric background pressures. Therefore, utilizing the outline to represent plume divergence overstates the divergence of the bulk of the plume.

Snapshots of the plume, such as that presented in Fig. 2.10, are not determined solely by outlier particles, but rather represent every particle in the plume. Such snapshots can be misleading in that particles which appear near one another in a 2-D view may not actually be near each other in 3-D space. Furthermore, snapshots are not quantifiable. One can utilize plume outlines along with plume snapshots to visually compare plume divergence, as presented in Fig. 2.10, but it is not feasible to quantitatively compare the plume divergence using snapshots. Therefore, while snapshots are a means of visualizing plume divergence, they are not a plume divergence metric. The objective of this section is to propose a definition for electrospray plume divergence which is quantifiable and is representative of the majority of the plume, rather than outlier particles. Furthermore, recognizing that electrospray plumes can evolve in momentum in addition to position, we propose a secondary plume divergence metric which accounts for momentum evolution in addition to positional evolution.

Fundamentally, electrospray plumes and the plumes or beams produced by other electric propulsion systems[130, 131, 132, 133] are collections of moving particles, similar to a laser beam. In a focused beam like a laser, there is a 'sharp' edge to the particle stream such that angular beam divergence from the emission axis can be defined as

$$\theta = \arctan\left(\frac{\Delta r_b}{l}\right) \tag{7.2}$$

where Δr_b is the beam radius (measure from the beam center to the outline containing 100% of particles) over distance l along the primary axis of beam motion. However, electrospray plumes are diffuse and have no sharp edge. As presented in Fig. 5.9, the plume outline set by the widest particle trajectories is much broader than most particle trajectories, such that outlines misrepresent divergence of the bulk of the plume. We propose that the effective plume 'edge' for electrospray thrusters can be defined based on three standard deviations, 3σ, of a Gaussian fit to the particle mass density distribution. Three standard deviations represents the statistical bulk (99.7%) of the plume without being skewed by outlier particle trajectories. Particle mass density is the key distribution for thruster divergence analysis because electrospray thruster lifetime is limited by the mass of propellant deposited on downstream grids [3]. Plume divergence angle at distance z downstream of emission is thereby obtained from Eq. 7.2 by substituting the radius of three of standard deviations of particle mass density, $\Delta r_{3\sigma}$ for Δr_b, and distance z for the path length l:

$$\theta_{3\sigma} = \arctan\left(\frac{\Delta r_{3\sigma}}{z}\right). \tag{7.3}$$

Utilizing Fig. 5.9 as an example, under our definition, the electrospray plume has divergence of 23.5° at atmospheric pressure ($P = 760\,\mathrm{Torr}$).

In addition to defining plume divergence using three standard deviations of the mass density distribution, we propose the use of emittance, ϵ, as a new metric for characterizing electrospray plume divergence. Emittance is a beam parallelism measure from the particle accelerator community[88, 89, 22]. Accelerator beams such as the Large Hadron Collider (LHC) at CERN propagate within confining systems for kilometers[134], orders of magnitude longer distances than the beams and plumes produced by electric propulsion systems which may propagate only centimeters before exiting the thruster. In pursuit of keeping charged particle beams steady and confined for such long distances, the particle accelerator research community has developed a rich literature of beam quality analysis and diagnostic metrics[88, 89, 22]. Emittance is one such beam quality metric, which displays the influence of non-Hamiltonian forces on the beam. Louiville's Theorem of emittance conservation states that emittance is conserved in a non-accelerating beam with no Hamiltonian forces. If the beam is being accelerated, the emittance normalized with Lorentz parameters β and γ is conserved:

$$\epsilon_n = \epsilon\beta\gamma \tag{7.4}$$

where

$$\beta = \frac{v}{c}, \tag{7.5}$$

v is particle velocity, c is the speed of light, and

$$\gamma = \frac{1}{1-\beta^2}. \tag{7.6}$$

Emittance decreases in response to non-Hamiltonian forces like cooling, and increases in response to non-Hamiltonian forces such as heating, friction, scattering phenomena[135], or close-range Coulomb 'collisions' termed 'intra-beam scattering' in the CERN literature[88, 89]. The overall space charge effect of the charged electrospray plume creates an electric potential field which exerts Hamiltonian forces on particles and does not increase emittance; however, the stochastic, close-range Coulomb 'collisions' between neighboring particles do not create a consistent Hamiltonian force potential field and therefore do increase emittance. Such Coulomb collisions violate Louiville's Theorem of emittance conservation in 6-dimension phase space by initiating Markov processes separate from the Hamiltonian influence of the space charge of the overall plume[89]. Close-range Coulomb interactions between particles which have clustered in a 'traffic jam' are critical to electrospray plume divergence[68, 46], such that emittance can be hypothesized to increase in electrospray plumes due to intra-beam scattering.

Emittance plots convey transverse particle position component against transverse angle component, where angle is defined as the ratio of momentum in the transverse direction to momentum in the axial direction of beam propagation,

$$x' = \frac{p_x}{p_z}. \tag{7.7}$$

A quantitative emittance metric can be calculated using the area of the particle collection in the position-angle trace space for each transverse component direction:

$$\varepsilon_x = \frac{1}{\pi} \int \int dx dx'. \tag{7.8}$$

Emittance evolution in a beam is observed by taking emittance measurements of cross-sectional beam slices as shown in Fig. 7.1.

The beam emittance plots in Fig. 7.1 show changes in position as the beam converges and diverges, but no changes of momentum. The elliptical shape of the particle distributions on the emittance plots is conventional for propagating beams, in which particle position and

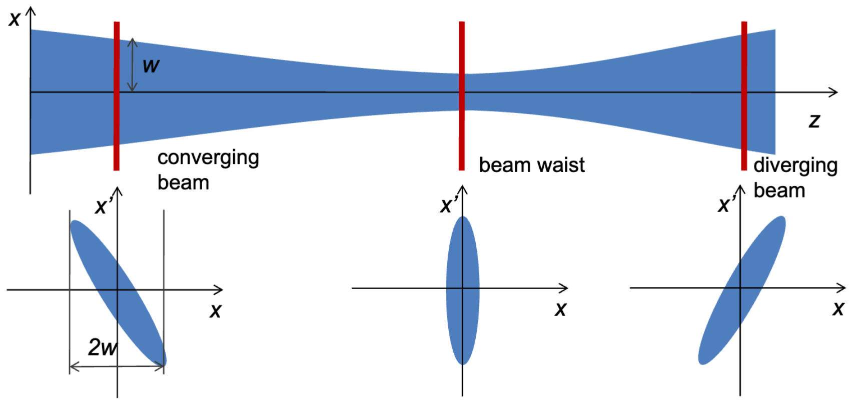


Figure 7.1: Cross sections of the plume used to observe trends in emittance as the plume moves downstream[22].

momentum in a beam cross-section have a physics-driven relationship. The area of the emittance plot ellipses in Fig. 7.1 remains constant throughout the propagation of the beam, conveying that his beam is not influenced by non-Hamiltonian forces such as intra-beam scattering.

7.1 Results and Discussion

Plume simulation results presented in this publication were obtained by the Plasma, Energy & Space Propulsion Laboratory (PESPL) at the University of California, Los Angeles (UCLA) using the Discrete Electrospray Lagrangian Interaction (DELI) Model. This model has been previously introduced and validated[46]. The electrospray geometry, 1.13nl/s flowrate constraint, and emitted EMI-Im particle data for the presented plume divergence results are from Miller et. al[24]. Three species of EMI-Im particles were utilized with the properties presented in Table 7.1.

A 2-D $x - z$ snapshot of the tri-species electrospray plume after 80 ms of simulation time using 1 µs timesteps is presented in Fig. 7.2. Lines overlayed on the plume show the location of cross-sectional beam slices at which emittance measurements were taken. The emittance

Table 7.1: Tri-species EMI-Im electrospray mass and integer charge number properties[24]. All particles are negatively charged.

Attribute	Small Species	Medium Species	Large Species
Mass [kg]	9.99e6	5.99.e7	2.78e8
Charge Number	90	252	650

plots of the particle collections at each of these cross-sections are presented in Fig. 7.3.

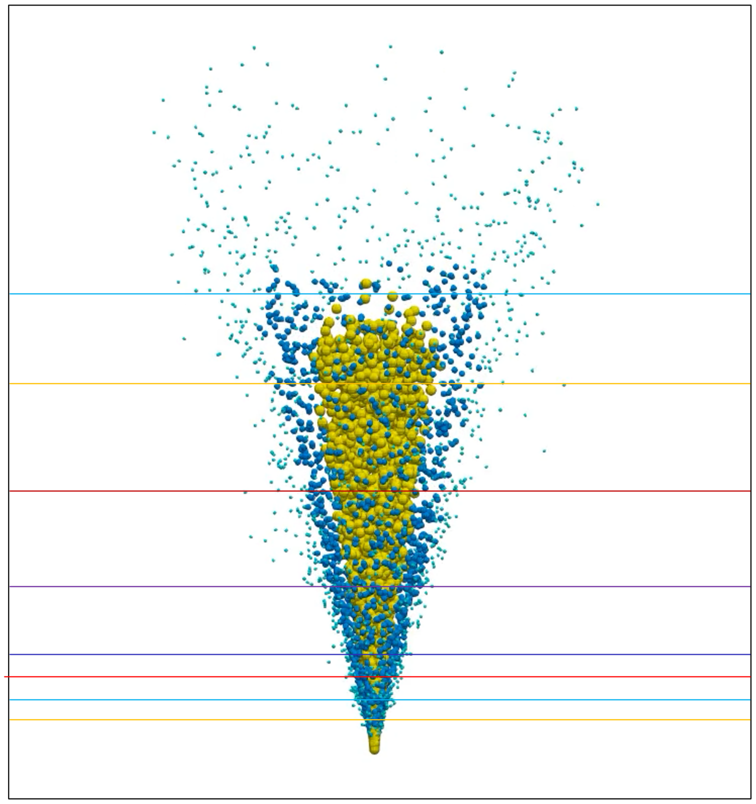


Figure 7.2: Cross sections of the tri-species EMI-Im plume at which emittance measurements were taken.

The emittance plots in Fig. 7.3 show that as the electrospray plume propagates downstream increasing axial coordinate z, the positional spread of the droplets across lateral

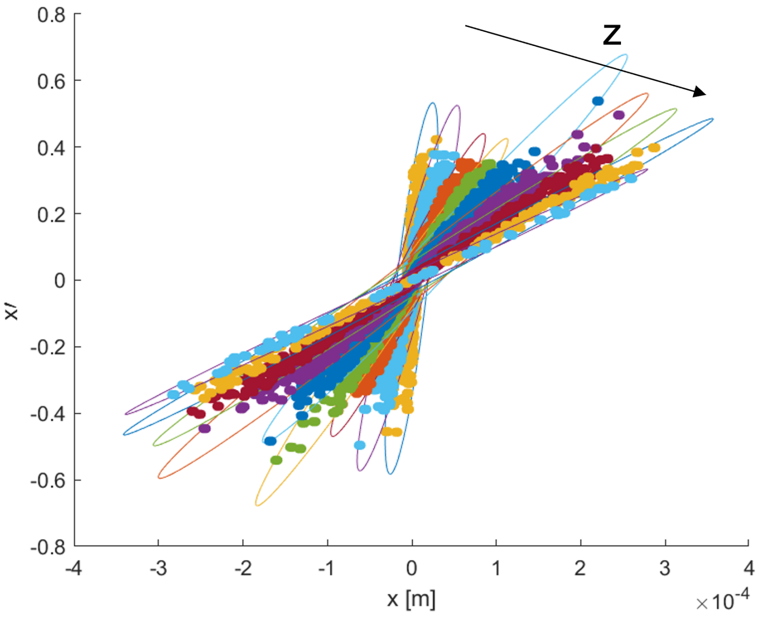


Figure 7.3: Emittance plots of the particles from each cross-sectional beam slice in Fig. 7.2.

coordinate x increases, meaning that the plume is diverging. The plume is also observed to generally decrease in x momentum angle, with a few outlier droplets that increase their momentum angle from the initial emission state. The momentum angle x' can be reduced by particle mass to velocity angle,

$$v_x' = \frac{v_x}{v_z}. \tag{7.9}$$

Therefore, most particles in this plume are observed to decrease in velocity angle as they move downstream. This observation fits with the understanding that Coulomb interactions cause significant divergence near the emission region, but as particles move downstream and spread out from one another, their trajectories are less divergent and guided by electric field lines. The area of the minimum enclosing ellipse for each cross-sectional collection of particles in Fig. 7.3 was calculated and used in Eq. 7.8 to yield emittance measurements for each cross-section. The emittance measurements are plotted against axial coordinate in Fig. 7.4 to show the evolution of emittance as the electrospray plume propagates downstream.

Cross-sectional emittance measurements in the direction of plume propagation allows for

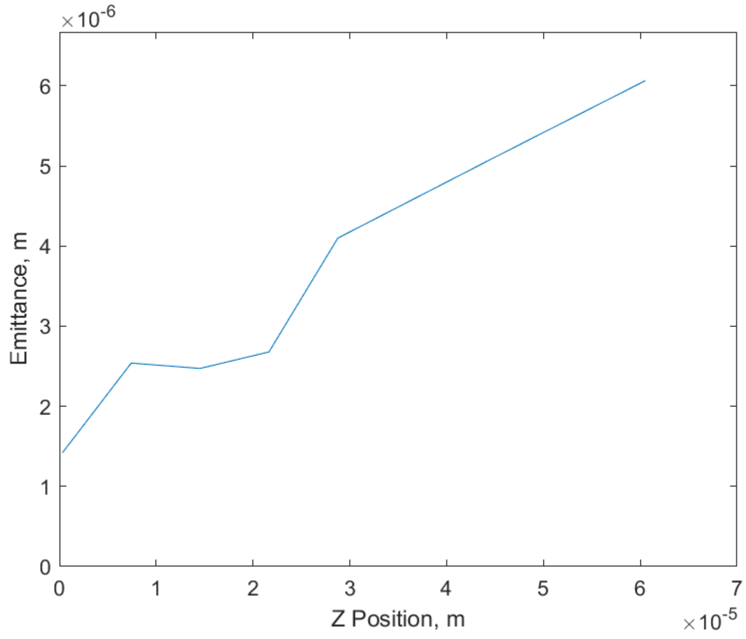


Figure 7.4: Emittance in the x component direction against axial coordinate for cross-sectional plume slices

divergence analysis considering both position and velocity. Figure 7.4 shows that emittance increases as the electrospray beam propagates. This result fits our hypothesis, because electrospray plume divergence is driven by stochastic Coulomb collisions between neighboring charged particles, which are a source of emittance growth. While the emittance trend presented in Fig. 7.4 is generally increasing, the exact shape of the emittance growth curve was found to be sensitive to the spacing between cross-sectional beam slices in the region near emission ($z < 3 \times 10^{-5}$ m). The frequency of near-neighbor Coulomb collisions decreases as particles move downstream of the charge-dense emission region, such that the emittance evolution slope can be expected to decrease as the plume moves further downstream. Bounding the 'interaction region' in which particles are dominated by Coulomb interactions is an active area of electrospray research[13, 14, 136]. Emittance can be useful towards this effort, because when stochastic, near-neighbor Coulomb collisions are no longer contributing to plume divergence, normalized emittance will plateau.

Finally, we propose that emittance diagrams of the full plume, as opposed to cross-sectional plume slices, can be useful for identifying when plumes have reached steady state. An emittance plot for the x-component of the full tri-species EMI-Im plume is presented in Fig. 7.5. As the plume evolves, so does the emittance representation of the plume. During plume startup, the full-plume emittance evolves with plume position and velocity distributions. When the plume is steady in both position and velocity, the emittance diagram will show no further changes in structure. We propose that a simulated plume has reached steady state when its full-plume emittance diagram reaches a constant shape.

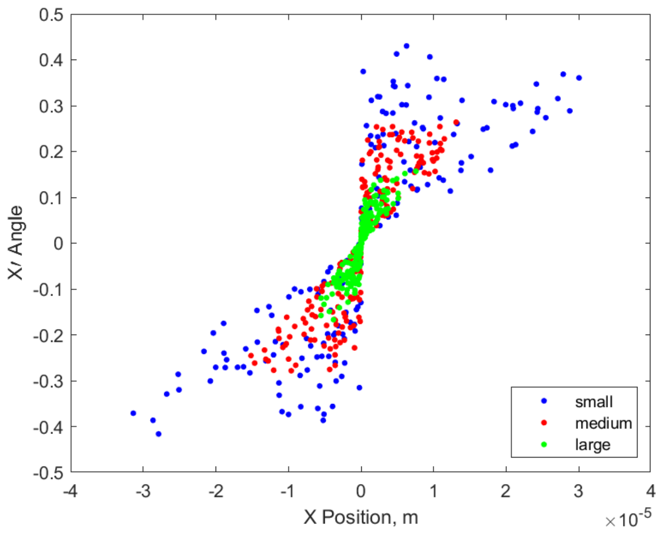


Figure 7.5: Emittance in the x component direction for the full plume with color corresponding to species.

CHAPTER 8

Predicting Plume Divergence

Although both experimental and computational electrospray research have produced a surplus of valuable data, machine learning (ML) has previously only been applied to experimental data: to predict mean emitted particle size [137]; to assist with medical diagnoses [87, 138, 139, 140] and biological classifications [141, 142]; and to estimate ionization efficiency [143]. This chapter demonstrates the utility of applying machine learning, specifically regression algorithms, to simulated electrospray particle data.

Electrospray plume simulations are computationally expensive because the many charged particles in a plume interact Coulombically. Atmospheric electrospray plumes have been observed to contain over 10^4 particles [11]. In the vacuum regime, electrospray plumes generate significantly higher numbers of particles with diameters beneath the diffraction limit of optical microscopy [48, 122, 4]. Analysis of the flow rates and times to steady state from the University of California, Los Angeles (UCLA) Plasma, Energy & Space Propulsion Laboratory (PESPL) plume measurements made in vacuum [23] assuming 10 nm diameter particles implies that vacuum plumes can contain over 10^8 particles in typical experimental domain sizes on the order of 0.5 m [46]. Simulations of such electrospray plumes create large amounts of particle dynamics data to which ML methods can be applied for both classification and regression purposes. Each unique particle trajectory from a Lagrangian simulation can be used as a sample in the training set for ML models. Once a Lagrangian model has generated sufficient particle dynamics data for a validated case, regression algorithms trained on that particle data can predict the dynamics of particles outside the training set. In this manner, ML regression models can form meta-models of the Lagrangian particle tracking model for the given case of interest. A benefit of regressing on the known particle data set is saving the computational expense of additional runs of the full Lagrangian model.

A second benefit of applying ML to electrospray dynamics data is that certain algorithms, such as the Random Forest, yield feature rankings which are not inherently produced by Lagrangian particle tracking models, thereby elucidating correlations between discrete particle properties. No closed-form relation is known between electrospray particle properties at emission and final particle properties downstream of emission. While feature rankings cannot provide such an explicit relationship, they can provide insight towards the correlation strength between given emission and downstream particle properties.

Thirdly, it is beneficial to understand correlations between different final particle properties. Significant experimental time, effort, and resource is focused on building, aligning, and utilizing many types of electrospray measurement devices with discrete and specific purposes. For example, Faraday probes obtain current measurements, quartz crystal microbalances (QCM) obtain mass flux measurements; inductive charge detectors (ICD) measure charge flux; retarding potential analyzers (RPA) measure particle retarding potential; and time of flight (TOF) sensor measurements are used to determine particle velocity distributions. Probes can be moved throughout the plume to map parameters of interest, but may run into challenges with measurement speed, resolution, and geometric constraints. Challenges are compounded in efforts to utilize multiple probes in a single diagnostic chamber. Therefore, it is valuable to predict one type of measurement reliably from another with the assistance of a validated Lagrangian particle tracking code and ML models. This publication demonstrates how regression algorithms trained on validated particle tracking simulation data can predict unknown final particle properties given a known final property. Furthermore, analyzing which final particle properties help most in predicting other final properties allows ML models to optimize diagnostic design by determining which probes to include in a diagnostic suite to maximize knowledge of final particle properties.

This study serves as an initial investigation into applying machine learning to electrospray simulation results. Results reveal three primary benefits: the creation of a surrogate model for a Lagrangian particle tracking model, the provision of feature relations between emission and final particle properties, and the prediction of unknown final particle properties given known final properties. To demonstrate the utility of applying ML regression to electrospray simulation results, this investigation uses data produced by the UCLA PESPL Discrete Electrospray Lagrangian Interaction (DELI) Model [46], but this process can also be applied to other Lagrangian particle tracking models and data in the literature [9, 48, 13, 11, 14].

8.1 Methods

8.1.1 Data Acquisition

The Lagrangian particle tracking data used in this investigation were obtained from DELI Model simulations of emitted particle data and experimental conditions from Gañan-Calvo et al. [9], for which case the DELI Model was validated in Sec. 2.5. Particle data sampling was conducted during the steady-state of the simulated plume, when total particle number in the simulation domain had become asymptotic [9, 26]. Each data sample includes the following feature combination of initial (emission) and final (at collector plate) particle properties of every particle which left the domain during simulation: mass m, charge q, 3D emission coordinates (x_i, y_i, z_i), 3D emission velocity (v_{xi}, v_{y_i}, v_{zi}), final positional plume angle θ_f, 3D final velocity (v_{xf}, v_{yf}, v_{zf}), and final potential ϕ_{RPAf}. Eight of these particle characteristics (m, q, x_i, y_i, z_i, v_{xi}, v_{y_i}, v_{zi}) are assigned at emission, while the remaining five (θ_f, v_{xf}, v_{yf}, v_{zf}, ϕ_{RPAf}) are obtained when the particle reaches the collector plate. The DELI Model simulations did not include particle breakup after emission such that the mass and charge of each particle are the same at the collector plate as they are at emission. As in Eq. 2.11, particles in the DELI Model are propagated downstream from emission under electrostatic, Coulomb, image charge, and drag forces:

$$m\frac{d^2\mathbf{x}}{dt^2} = q(\mathbf{E_A} + \mathbf{E_C} - \mathbf{E_I}) - \mathbf{F_D} = q\mathbf{E_A} + \frac{q}{4\pi\epsilon_0}\sum_{i=1}^{n}\frac{q_i\mathbf{r_i}}{|\mathbf{r_i}|^3} - \frac{q^2}{4\pi\epsilon_0}\frac{\mathbf{r_I}}{|\mathbf{r_I}|^3} - C_D\frac{\pi}{8}\rho_{fl}d^2\mathbf{v}|\mathbf{v}|, \quad (8.1)$$

where $\mathbf{x}$ is the particle position, $\mathbf{E_A}$ is the applied electric field, $\mathbf{E_C}$ is the Coulomb electric field generated by the charged plume particles, $\mathbf{E_I}$ is the electric field created by image charges induced in the collector plate, and $\mathbf{F_D}$ is the drag force. Furthermore, ϵ_0 is the permittivity of vacuum, n is the total number of particles in the plume, $\mathbf{r_i}$ is the separation vector between particles ($\mathbf{r_I}$ is the separation vector between a particle and its image charge in the collector plate), C_D is the coefficient of drag, ρ_{fl} is the density of the surrounding fluid, d is particle diameter, and $\mathbf{v}$ is particle velocity.

Axial emission coordinate z_i does not vary between particles for the presented simulations such that it can be removed from the feature set. Two cases of particle data results are analyzed in this publication:

Case I: particles have strictly axial emission velocity

Case II: particles have non-zero radial emission velocity.

The emission velocity magnitude was held constant in both cases. Experimental observations of the cone-jet and particles [123], or just the cone-jet structure when particles are too small to be visualized [122, 85], show that particle emission can occur at a tilted angle from the vertical axis, contributing radial components to the velocity vectors of emitted particles. Off-axis particle emission can occur due to a tilted cone-jet structure [122, 85] or instabilities in a jet emitted from a nominal cone-jet structure [123]. Therefore, for case II, particle emission velocity angle is set to correspond to particle emission angle:

$$\tan\left(\frac{v_{ri}}{v_{zi}}\right) = \tan\left(\frac{r_i}{l}\right), \quad (8.2)$$

where r_i is the emission radial coordinate, v_{ri} is the emission radial velocity component, and l is the distance between the jet tip from which particles are emitted and the upstream reference point for calculating velocity emission angle. This distance is varied from one mean particle radius $l = \overline{d}/2$ to two mean jet breakup wavelengths $l = 2\overline{\lambda}$, where $\overline{\lambda} = 4.5\overline{d}$ [9, 55].

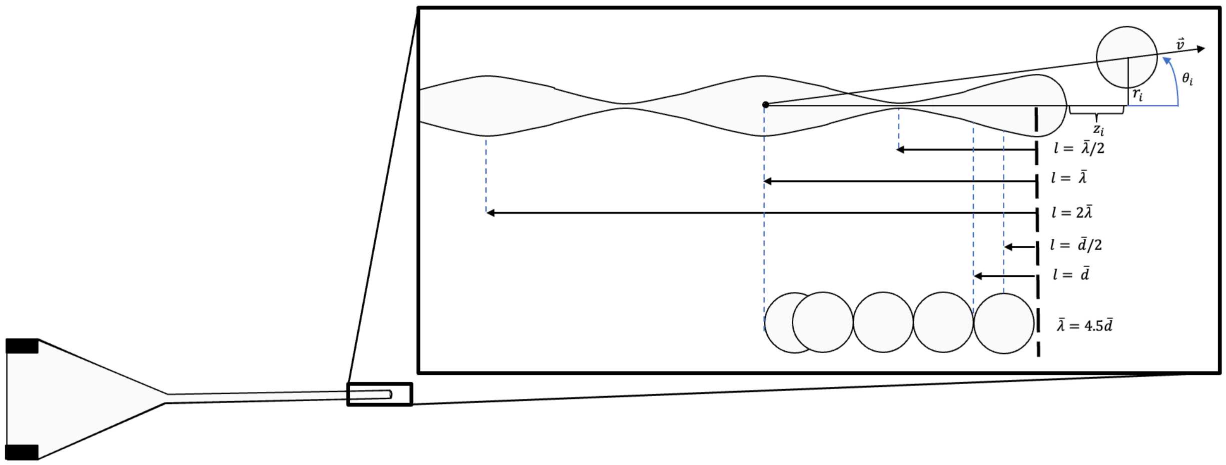


Figure 8.1: Emission velocity vector direction is the angle between particle emission position and a reference point in the jet distance l upstream from the jet tip, where l varies from one mean particle radius to one mean breakup wavelength. An example emission velocity vector is shown for $l = \overline{\lambda}$.

Figure 8.1 provides a diagram of this method of determining emission velocity angle. The nearer the reference point to the emission point, the larger the radial component to the particle emission velocity vector. Sec. 8.2.2 presents results for the shortest l case, $l = \overline{d}/2$, because it has the most variance in emission velocity component data and thereby gives the most stark comparison with the solely axial velocity case presented in Sec. 8.2.1.

Table 8.1 provides descriptions of each feature variable and formulas for cases where calculations were made to obtain the variable. In the table formulas, θ_i is the emission polar angle, ϕ_i is the emission azimuthal angle, $\boldsymbol{v_f}$ is exit velocity, r_f is the final radial position, and z_f is the final axial position, in this case the constant collector plate position. The formulas given for the emission velocity vector components are for the case in which emission velocity is not strictly axial; in that case there are no lateral components to velocity and the axial component is constant according to Gañan-Calvo et al. [9].

DELI simulation datasets including 35,000 unique particle samples were used for each of the emission velocity cases presented. Histograms of the emission (m, q, x_i, y_i, v_{xi}, v_{y_i},

Table 8.1: Feature Variables

Variable	**Definition**	**Formula**
m	mass	
q	charge	
x_i	emission x position	
y_i	emission y position	
z_i	emission z position	
r_i	emission radial position	$\sqrt{{x_i}^2 + {y_i}^2}$
v_{xi}	emission velocity x component	$\arctan\left(\frac{r_i}{l}\right)\cos\theta_i\cos\phi_i$
v_{y_i}	emission velocity y component	$\arctan\left(\frac{r_i}{l}\right)\cos\theta_i\sin\phi_i$
v_{zi}	emission velocity z component	$\arctan\left(\frac{r_i}{l}\right)\sin\theta_i$
r_f	final radial position	$\sqrt{{x_f}^2 + {y_f}^2}$
θ_f	final positional plume angle	$\tan\left(\frac{r_f}{z_f}\right)$
v_{xf}	final velocity x component	
v_{y_f}	final velocity y component	
v_{zf}	final velocity z component	
ϕ_{RPAf}	final potential	$\frac{\frac{1}{2}m\lvert\boldsymbol{v_f}\rvert^2}{q}$

v_{zi}) and final (θ_f, v_{xf}, v_{y_f}, v_{zf}, ϕ_{RPAf}) particle property data are presented in Appendix B for each emission velocity case; for the strictly axial emission velocity case, the constant emission velocity components are not visualized.

8.1.2 Data Preparation

The data in this study are strictly numerical and there are no missing data entries. A flowchart for the process of data preparation is presented in Fig. 8.2

Data were first filtered for uniqueness by eliminating duplicate samples. Then, the order

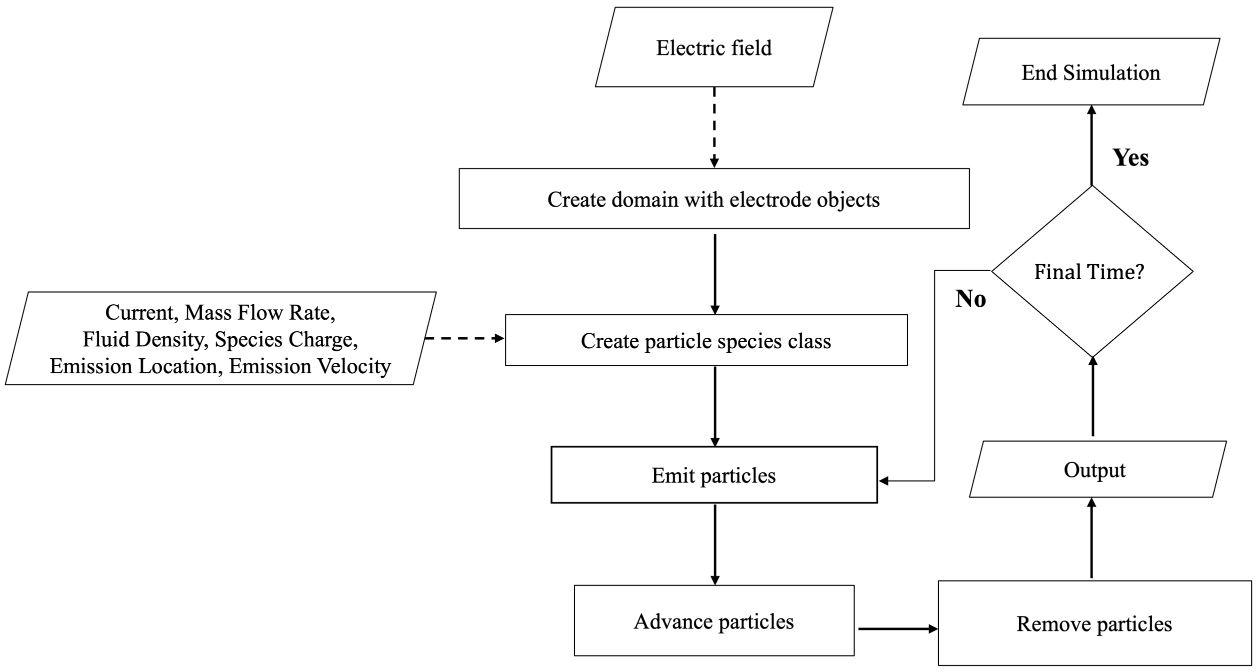

Electric field
End Simulation
Yes
Create domain with electrode objects
Final Time?
No
Current, Mass Flow Rate, Fluid Density, Species Charge, Emission Location, Emission Velocity
Create particle species class
Emit particles
Output
Advance particles
Remove particles

Figure 8.2: Process of preparing particle tracking data for input to machine learning models.

of data samples was shuffled randomly using the shuffle function from the scikit-learn library (v1.1.2) Python package [144]. Finally, all data were scaled before regression using the StandardScaler function in the scikit-learn library (v1.1.2)[144]. A standard scaler was fit to the training feature data and then applied to the testing feature data. The scikit-learn library train_test_split function was used to split the data into training and testing sets for the ML models, with 80% of the particle dynamics data reserved for training, and the remaining 20% for testing.

8.1.3 Model Construction

Six different ML models are applied to electrospray simulation results in this study: Random Forest (RF), Support Vector Regression (SVR), k-Nearest Neighbor(kNN), Multilayer Perceptron (MLP), Extreme Gradient Boosting (XGBoost), and Light Gradient Boosting Machine (LGBM) models. The scikit-learn library (v1.1.2)[144] was used for all models besides the XGboost model from the py-xgboost library (v1.6.2)[145] and the LGBM model from the lightgbm library (v3.3.2)[146].

The performance metrics used in this study are the coefficient of determination (R^2) and the absolute-normalized root mean squared error ($ANRMSE$), defined as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (\bar{p_i} - \hat{p_i})^2}{\sum_{i=1}^{N} (\bar{p_i} - p_i)^2}, \tag{8.3}$$

$$ANRMSE = \frac{\sqrt{\frac{\sum_{i=1}^{N} (p_i - \hat{p_i})^2}{N}}}{\overline{|p_i|}}, \tag{8.4}$$

where N is the number of samples, p_i is the true value of the variable being predicted, $\hat{p_i}$ is the value predicted by an ML model, and $\overline{p_i}$ is the mean of the true values.

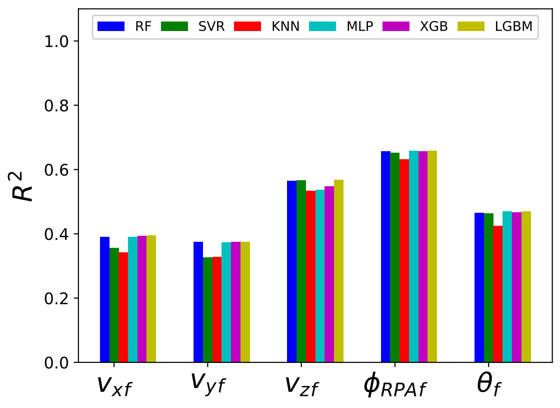
8.1.4 Model Optimization

Hyperparameter tuning was conducted through 5-fold cross validation on all ML models to obtain each presented result. The goal of this tuning was to optimize model R^2 scores. The hyperparameters for each model, their tuning ranges, and the tuned parameters for the strictly axial emission velocity case and the non-zero radial emission velocity case where $l = \overline{d}/2$ are presented in Appendix C.

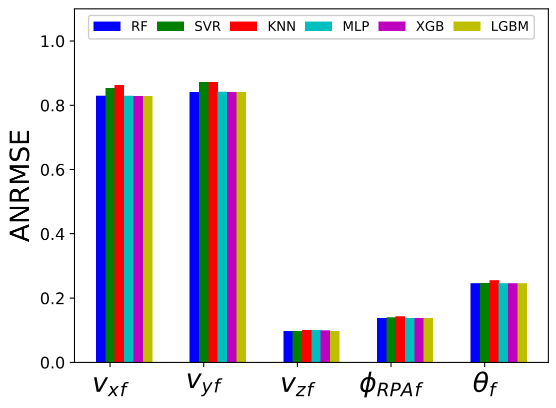
8.2 Results and Discussion

8.2.1 Case I: Strictly Axial Emission Velocity

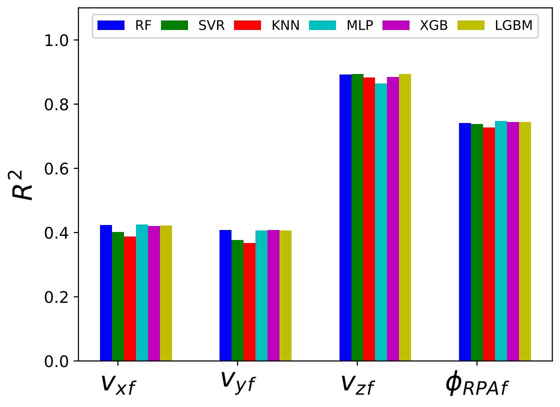
We first present analysis of the case in which particles were emitted with constant, strictly axial velocity. R^2 scores and $ANRMSE$ results are shown in Fig.s 8.3a and 8.3b, respectively, for the prediction of each final particle property (θ_f, v_{xf}, v_{yf}, v_{zf}, ϕ_{RPAf}) using non-constant emission particle properties (m, q, x_i, y_i) as input features. The performance metric results are presented with bar color corresponding to predictive model, such that the range of performance can be seen among the six ML models.



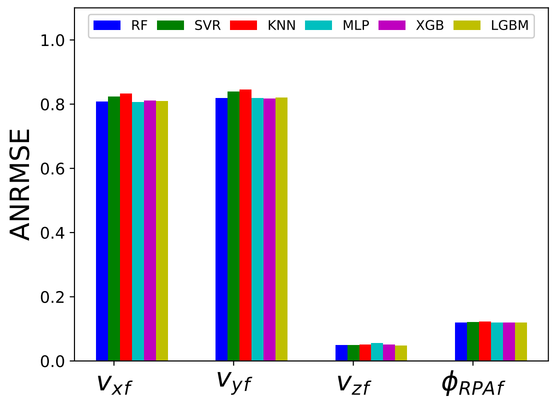
(a)



(b)



(c)

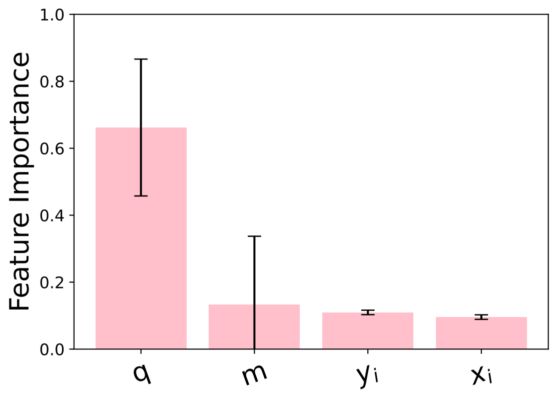


(d)

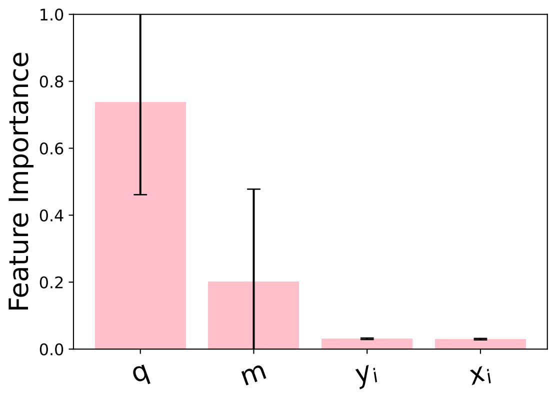
Figure 8.3: Performance metrics R^2 (a,c) and $ANRMSE$ (b,d) for all regression algorithms with final angle withheld from the feature set (a, b) and final angle included (c,d).

We observe that no ML models are able to reliably predict any final particle properties from solely emission properties, as all R^2 scores are below 0.7, signaling low correlation between the input features and predicted variable. Axial emission coordinate and all components of emission velocity are constant in this case. Furthermore, as shown in the mass and charge histograms in Appendix B, this plume is nearly monodisperse. Therefore, there is little ability to differentiate emitted particles from one another in order to predict final particle dynamic properties from solely emission properties. However, when final particle angle θ_f is added to the input feature list alongside emission particle properties, stronger performance is achieved in predicting other final particle properties (specifically v_{zf} and ϕ_{RPAf}), as shown in the increased R^2 scores in Fig. 8.3c and the decreased $ANRMSE$ scores in Fig. 8.3d. This predictive ability response to knowing final angle reveals a strong correlation between final angle and other final particle dynamic properties for this emission velocity case. Knowing final angle alongside emission properties allows ML models to reliably regress final axial velocity ($R^2 > 0.88$ for all models) and final potential ($R^2 > 0.76$ for all models) from the known particle data. Additionally, the $ANRMSE$ with final angle in the feature set in Fig. 8.3d has decreased for final axial velocity and final potential from the case in which final angle was not included in Fig. 8.3b. Therefore, a limited meta-model for the full Lagrangian model can be created with ML models to obtain final axial velocity and final potential given emission properties and final angle.

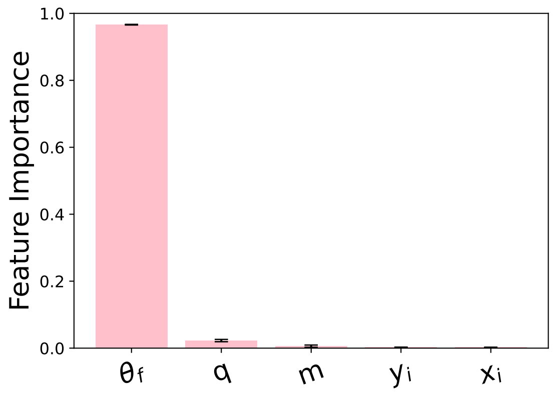
Feature correlations are further investigated for final axial velocity and final potential, which the ML models predict with high accuracy when final angle is included with emission properties in the known feature set. Figure 8.4 displays the feature relation results produced by the Random Forest model for each of these two variables, for the case in which final angle was not included in the input feature list and the case in which it was. Error bars display the range of performance of all individual estimators, or trees, in the RF algorithm. When final angle is not included in the feature list, particle charge has dominant feature importance for the prediction of both final axial velocity and final angle. This charge-dominated result



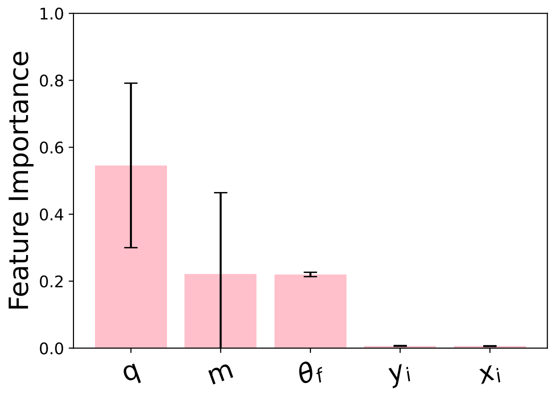
(a)



(b)



(c)

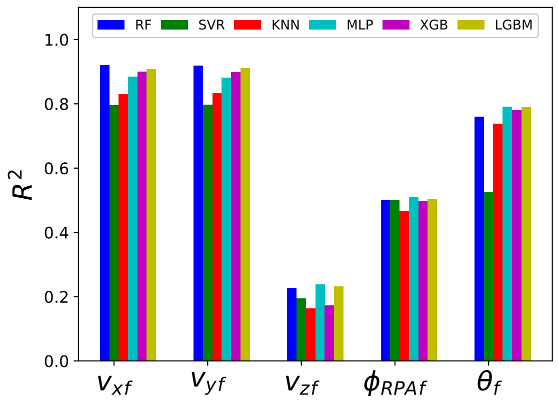


(d)

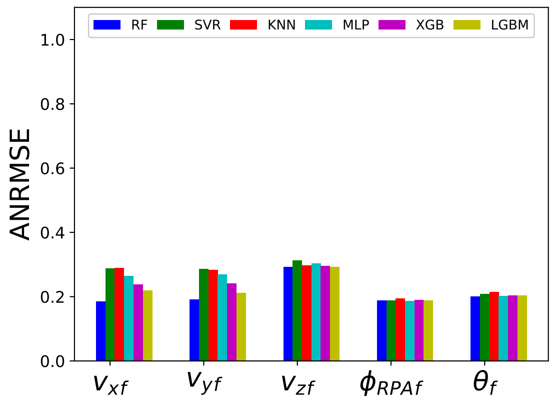
Figure 8.4: Random Forest feature rankings for final axial velocity (a,c) and potential (b,d) for the case in which final angle is excluded from the feature set (a,b) and the case in which it is included (c,d). Error bars show feature importance range of all individual estimators in the RF algorithm.

physically stems from the direct relationship between particle charge and the electrostatic forces in Eq. 8.1. When final angle is included in the feature list, it is significantly dominant over all other features for the prediction of final axial velocity, displaying that final angle and final axial velocity are highly correlated for this simulation case. For the prediction of final potential energy, particle charge remains the dominant feature when final angle is included in the feature list. This is expected as final potential energy is indirectly related to particle charge by definition, as given in Table 8.1. Similarly, mass has secondary feature dominance as potential energy is directly related to particle mass by the definition given in Table 8.1. However, final angle has a non-negligible feature importance for final potential energy in every decision tree in the random forest, with average feature importance across the forest nearly equal to that of mass. Final particle angle offers insight towards how much a particle has been displaced from its emission state under the forces in Eq. 8.1, such that knowing this angle provides insight toward other final properties which have been displaced from their emission state by the same forces, such as velocity and potential.

In the future, terminal particle properties may be more reliably predicted from solely emission particle properties for the strictly axial emission velocity case with 1) an increased number of unique training data samples; 2) increased variance among particle emission property feature data sets, such as increased mass polydispersity or charge inhomogeneity, so that emitted particles are more distinguishable from one another; 3) the identification of additional particle emission properties of relevance to expansion which can be added to the feature set; and 4) the improvement of Lagrangian particle tracking models, such as including particle coalescence and breakup[10, 26, 83, 62], so that simulated results more closely match the physical reality of electrospray evolution. As an example of the effect of increasing data samples on the study, changing from 3,000 samples for initial research to 35,000 samples for the presented case improved the mean R^2 score for predicting final axial velocity component by 0.09 in the case without θ_f in the known feature set, and by 0.08 in the case with θ_f in the known feature set.



(a)



(b)

Figure 8.5: Performance metrics R^2 (a) and $ANRMSE$ (b) for all regression algorithm for the case with non-zero radial emission velocity.

8.2.2 Case II: Non-zero Radial Emission Velocity

We now extend our analysis to the simulation case in which particles were emitted with non-zero radial velocity. The results presented in this section are for the shortest l length tested ($l = \overline{d}/2$) because this case has the largest variance among emitted velocity vector components of all simulated cases. Results were also obtained for $l = \overline{d}, \overline{\lambda}/2, \overline{\lambda}$, and $2\overline{\lambda}$, where the mean jet breakup wavelength $\overline{\lambda} = 4.5\overline{d}$; the influence of l on ML model results is discussed in Sec. 8.2.3. R^2 scores and $ANRMSE$ results are shown in Fig. 8.5 for the prediction of final particle properties using solely emitted particle properties as input features.

Unlike in the previous case with constant, strictly axial emission velocity, there is significant correlation between emission and final particle properties in this case with non-zero radial emission velocity. The final lateral velocity components can be predicted with $R^2 \geq 0.79$ and $ANRMSE \leq 0.3$ by all models. Therefore, a surrogate model for the larger Lagrangian model can be created for the prediction of final lateral velocity components from solely emission properties in this case. Additionally, all models but SVR predict final angle with $R^2 \geq 0.77$ and $ANRMSE \leq 0.25$. Therefore, the surrogate model created by these ML models can also predict final angle from solely emission properties.

R^2 scores for predicting all final properties improve when final angle is added to the known features set because all particle properties are still displaced from their emission state through the same dynamic process described by Eq. 8.1, yielding an angular dependence of final properties as discussed in Sec. 8.2.1. This improvement is most significant for predicting final axial velocity, for which mean R^2 increases by 0.19 when final angle is added to the known feature set. The differences between predictions which can be reliably made in the two emission velocity cases and the governing physics underlying such predictive abilities are discussed in Sec. 8.2.3. Prediction accuracy for this non-zero radial emission velocity case may be improved through the same mechanisms as for the strictly axial emission velocity case: increased training data samples, increased variance in emission property feature data sets, expansion of the feature set to include newly identified particle emission properties, and improvement of the Lagrangian particle tracking models used to obtain training data.

8.2.3 Comparison of Case I and II

We now compare and contrast results from simulation Case I, with solely axial emission velocity, and Case II, for which emission velocity angle corresponds to emission position angle. There are significant differences between the predictive meta-models which can be created using ML in each case. While the applied machine learning models all perform different methods of physics-blind, statistical regression, the correlations between particle properties that they reveal stem from underlying electrospray particle dynamic physics. Fig. 8.6 presents snapshots in the x-z plane of the pathlines of 30 particles for the axial emission velocity (red pathlines) and non-zero radial emission velocity (blue pathlines) cases. The snapshots were taken every 10 µs over a 30 µs period and pathlines are shown for every other emitted particle out of 60 particles to prevent the density of rendered pathlines from obscuring figure clarity. As seen in blue in Fig. 8.6, charged particles emitted with non-zero radial emission velocity disperse quickly from one another due to inertia, and therefore spend less time in close proximity to one another experiencing nonlinear, inter-particle Coulomb forces. In this case, particle expansion dynamics have a significant linear component from inertia which allows ML models to correlate some final particle properties with solely emission properties. In contrast, as shown in red in Fig. 8.6, charged particles emitted with strictly axial emission velocity remain in a nearly linear formation for much longer, until Coulomb forces exacerbate the minute perturbation applied to particle emission position to yield plume expansion. In this case, particle dynamics are governed by nonlinear Coulomb forces, which preclude ML models from predicting final particle properties from exclusively emission properties.

Particle positional angle exiting an electrospray thruster is a key metric for thruster lifetime because there is a finite line-of-sight angle at which particles can exit the thruster; particles beyond this angular limit strike downstream thruster electrodes and saturate them over time until the thruster fails due to propellant backspray [44, 3, 12, 45]. Predictions for final particle angle from all six ML models using solely emission particle properties are compared to the actual final angle values in the test data set in Fig. 8.7. Feature rankings produced by the RF models for both emission velocity cases are also presented in Fig. 8.7. Note that the feature rankings for the case with radial emission velocity include velocity components v_{xi}, v_{y_i}, and v_{zi} because they differ between emitted particles.

Fig. 8.7 shows the R^2 score for predicting final positional angle from strictly emission properties is higher in the non-zero radial emission velocity case as a result of stronger influence of linear forces on particle dynamics. For the solely axial emission velocity case, although final angle cannot be reliably predicted ($R^2 < 0.7$) from solely emission properties due to the non-linear nature of particle dynamics, RF feature rankings reveal charge to be the dominant feature for predicting final angle. This result matches the physical understanding of plume expansion in this case as dominated by the charge-dependent Coulomb collisions [9, 11,

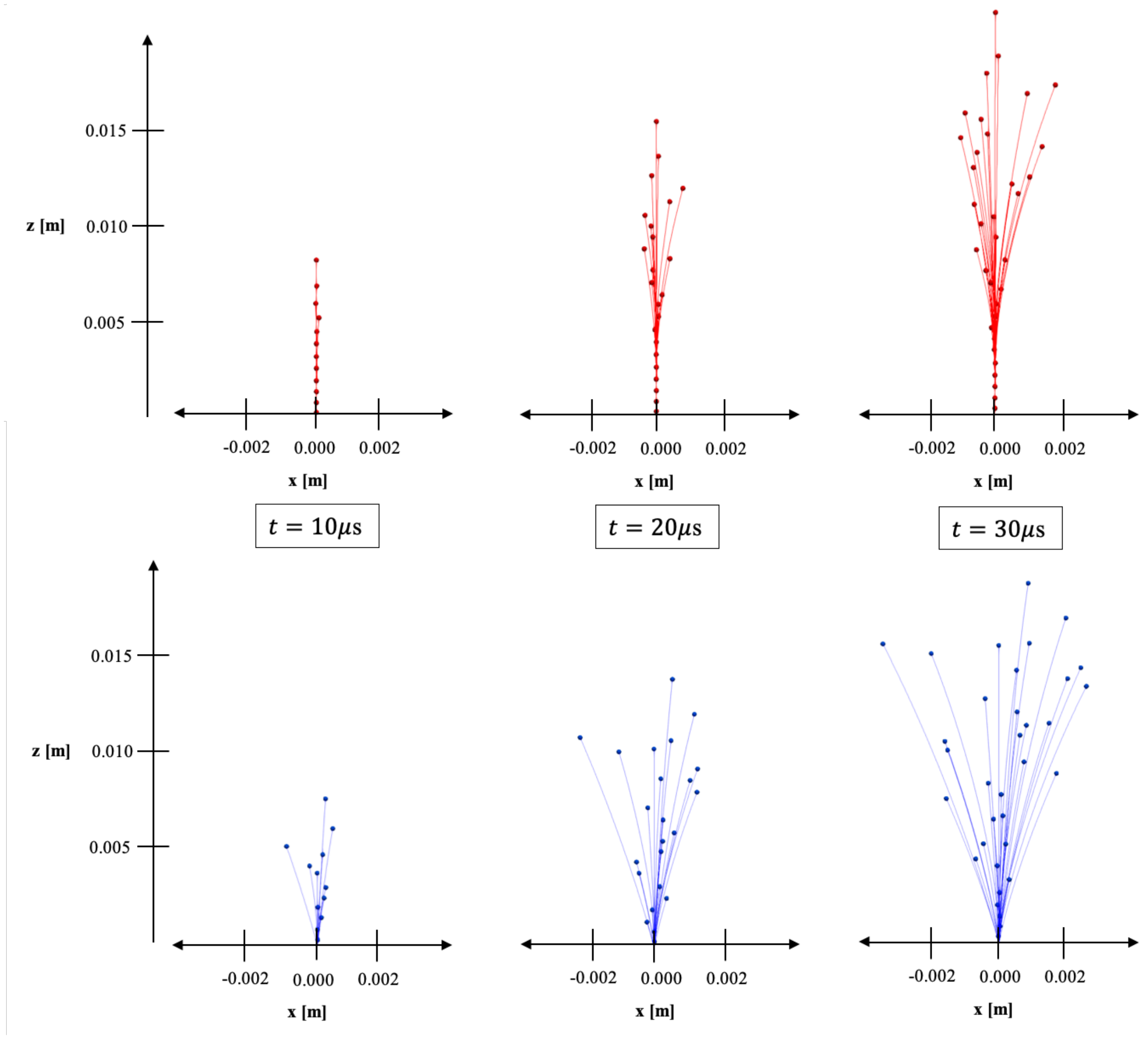
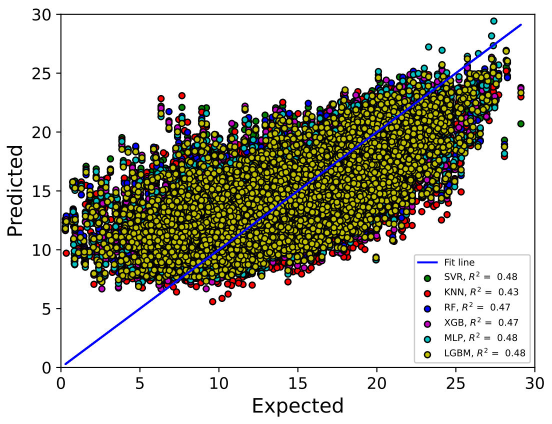
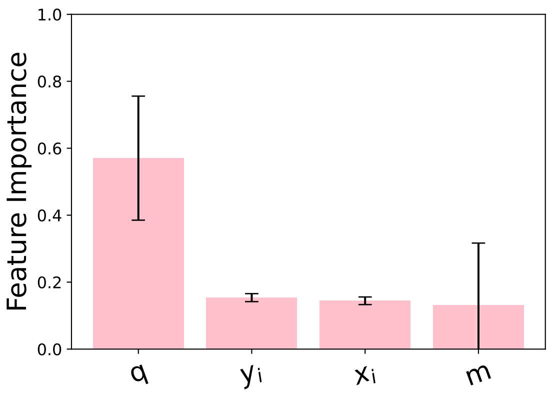


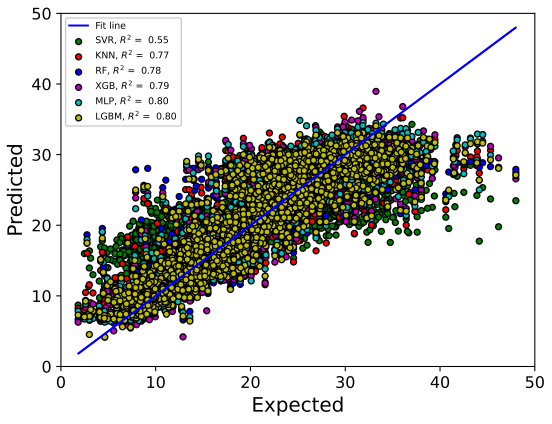
Figure 8.6: Pathlines are shown for 30 particles in the solely axial emission velocity Case I (red) and the non-zero radial emission velocity Case II (blue).



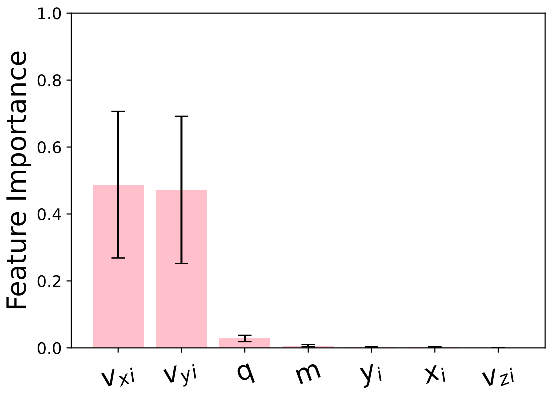
(a)



(b)



(c)



(d)

Figure 8.7: Predicted vs. actual values for final angle (a,c) and the Random Forest feature rankings for predicting final angle (b,d) for the case in which emission velocity is strictly axial (a,b) and the case in which there is a radial component to emission velocity (c,d). Feature ranking error bars show the range among estimators in the RF algorithm.

68, 46] in the absence of lateral emission velocity components. For the case including radial emission velocity, final angle is reliably predicted ($R^2 \geq 0.8$) by all ML models but SVR. An R^2 score exceeding 0.75 is comparable with the high end of R^2 scores in other machine learning studies of electrosprays [137, 143]. In the non-zero radial emission velocity case, RF feature rankings reveal that final positional angle has a dominant feature dependence on emission lateral velocity components, followed by a lesser feature dependence on charge. These results fit with the physical understanding of particle trajectories in this case as dominated by inertial motion from emission with additional influence from charge-dependent electrostatic forces in Eq. 8.1. In both cases, there is a sparsity of data with high final particle positional angles, leading to higher error in the predictions of ML models for high-angle particles due to a lack of sufficient model training data in that range. This data sparsity at high angles is not a simulation artefact and is also seen in experimental measurements due to the relative lack of particle flux at high positional angles compared to the particle-dense center of the plume [45, 147, 23]. Model performance in this range can be improved in the future with increased data sampling at high angles.

ML models applied to particle dynamics in the DELI Model simulation region, in which particles are propagated according to Eq. 8.1, have determined that emission lateral velocity components, and therefore emission velocity angle, are strongly correlated with final positional angle as shown in Fig. 8.7. Sobol Index analysis of the region downstream of that of the DELI Model, in which particles are propagated under applied electrostatic forces with no Coulomb interactions, found that particle entry velocity angle into that region has the strongest correlation with final position angle of any particle entry property [63]. Thus, there is a continuous narrative of emission velocity angle dominantly influencing final positional angle in all electrospray plume simulation regions. Given the importance of final particle position angle to thruster lifetime, the correlation between final positional angle and emission velocity angle motivates continued study into the angle at which particles are emitted from the fluid electrospray jet. Existing literature on electrospray jet behavior identifies several operational modes which emit particles with off-axis velocity vectors, such as whipping [85, 148, 6, 123], multi-jet [85, 149, 6, 150], and transitional modes [4, 85], and furthermore suggests whipping can occur at the jet tip even when the overall cone-jet structure is in the desired steady operational mode [123]. Increased study of modes which emit particles with off-axis velocity will improve the understanding of deviations from desired thruster lifetime during mission-relevant electrospray thruster operation, which includes periods of non-optimal transitional operation such as start-up and shut-down modes [44, 3, 12, 45].

The thrust force produced by an electrospray device is given by:

$$\mathbf{T} = \int \dot{m}(\theta)\mathbf{v}(\theta)d\theta, \tag{8.5}$$

where $\dot{m}(\theta)$ is the mass flux at a given angle and $\mathbf{v}(\theta)$ is the velocity at that angle. Therefore, final particle velocity exiting the thruster not only relates to thruster lifetime as discussed, but is alo directly related to produced thrust, a key performance metric. For predicting axial final velocity from strictly emission properties, comparing R^2 scores in Sections 8.2.1 and 8.2.2 shows that such scores are higher in the strictly axial emission velocity case. In both emission velocity cases, the applied electrostatic field ($\mathbf{E_A}$ in Eq. 8.1) provides the dominant axial propagation force to emitted particles [68, 4]. The acceleration $\mathbf{a_{E_A}}$ imparted to a particle from this field is directly related to charge and indirectly related to mass:

$$\mathbf{a_{E_A}} = \frac{q}{m}\mathbf{E_A}. \tag{8.6}$$

Under this electrostatic means of axial propagation in a constant applied electrostatic field, the final axial velocity of particles emitted at the same axial velocity has a linear relationship with charge and mass. Therefore, ML models can strongly correlate final axial velocity with emission properties for the strictly axial emission velocity case. However, for the non-zero radial velocity emission case, this linear relationship between final axial velocity and emission properties is significantly weakened because variance in particles' final axial velocity does not stem solely from electrostatic acceleration, but also from variance in emission axial velocity. Randomness in emission conditions, rather than a change in the dominant physical force guiding particle motion, causes the axial motion of particles to be less predictable in the non-zero radial velocity emission case than in the strictly axial emission case. In both cases there is a relative sparsity of data at low final axial velocities compared to higher final axial velocities displayed in Appendix B; ML model prediction strength for final axial velocity can be improved through increased data sampling.

On the other hand, for predicting lateral final velocity components from strictly emission properties, comparing R^2 scores in Sections 8.2.1 and 8.2.2 shows that such scores are higher in the non-zero radial emission velocity case. Coulomb interactions between charged particles, which are highly nonlinear, provide the dominant radial expansion force to electrospray particle trajectories [9, 11, 68, 46]. Therefore, in the absence of a radial component to emitted particle velocity, plume expansion is highly nonlinear. As for final positional angle, final lateral velocity emission components can be more reliably predicted from emission properties in the non-zero radial emission velocity case due to the dominance of linear inertia in radial particle motion, in comparison to the dominance of nonlinear Coulomb interactions in particle radial motion in the strictly axial emission velocity case. In agreement with the dominance of emission inertia, RF feature rankings reveal that emission lateral velocity components share dominant feature importance for final lateral velocity components in the case with non-zero radial emission velocity.

The results presented for the non-zero radial emission velocity case in Sec. 8.2.2 are for the shortest l length tested ($l = \overline{d/2}$) because this case has the largest variance among emitted velocity vector components of all simulated cases. Results were also obtained for $l = \overline{d}, \overline{\lambda}/2, \overline{\lambda},$ and $2\overline{\lambda}$, where the mean jet breakup wavelength $\overline{\lambda} = 4.5\overline{d}$. The R^2 scores for predicting final angle θ_f, final lateral velocity component v_{f_x}, final axial velocity component v_{f_z}, and final potential ϕ_{RPA_f} are displayed in Fig. 8.8 for all emission velocity cases. Point markers represent the mean performance of all six ML models and error bars display the range of performance among the six models. The performance metrics for the strictly axial velocity case are shown as constant, dashed lines for comparison with the non-zero radial

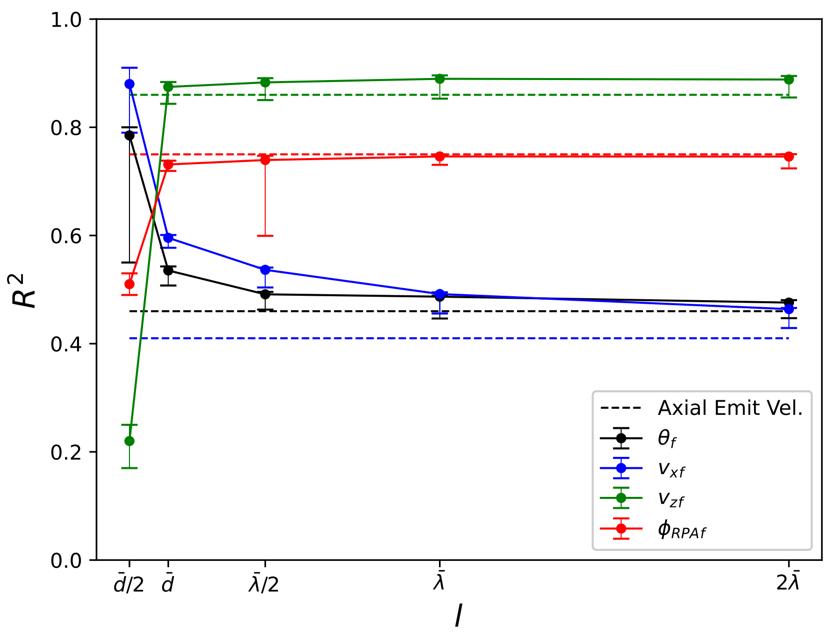


Figure 8.8: Mean R^2 scores for different distances l between the jet tip and the reference point for determining velocity emission angle for the prediction of: final angle given emission properties (black), final lateral velocity (v_{f_x}) given emission properties (blue), final axial velocity given emission properties (green), and final particle potential given emission properties and final angle (red). Dashed lines display R^2 results for the case with no radial emission velocity. Error bars display the performance range of the 6 ML models.

emission velocity cases.

Fig. 8.8 displays that for predicting final angle given emission properties, R^2 varies directly with l. R^2 also varies directly with l for predicting final axial velocity given emission properties. On the other hand, for predicting final lateral x velocity component given emission properties, R^2 varies inversely with l. The same is true for predicting the final lateral y velocity component. R^2 also varies indirectly with l for the predicting final potential given emission properties and final angle. These results show that ML model performance becomes increasingly similar to the strictly axial emission case as l increases. The ML results for $l = 2\overline{\lambda}$ are nearly equivalent to those from the zero radial emission velocity case represented by the dashed lines. The convergence of ML results to the strictly axial case with increasing l is physically logical because increasing the distance l between the jet tip and upstream reference point for velocity emission angle decreases the resulting range of emission velocity angles. The lower the variance in emission velocity angle, the less particles disperse from inertia directly following emission, such that particle pathlines more closely resemble those in the strictly axial emission velocity case shown in red in Fig. 8.6. When particle velocity emission conditions approach the strictly axial case as l increases, resulting particle dynamics and the ability to predict final particle properties from emission properties also approach those of the strictly axial velocity emission case.

8.2.4 Discussion of Application to Experimental Data

We will now discuss the diagnostic design benefits of using ML models trained on electrospray particle tracking simulation data in conjunction with experimental results. Our discussion is focused on the coupling of experimental particle property measurements with ML models applied to simulated particle tracking data to predict unknown particle properties which can be validated experimentally. Suppose an angular plume profile has been obtained using one type of measurement device. Further suppose that a particle tracking model such as the UCLA PESPL DELI Model has been validated for the case of interest, and the ML models described in this publication have been trained on a significant amount of particle dynamics data produced by the particle tracking model for this case. Let the varying particle emission properties (m, q, x_i, y_i) be established by optical tools [151, 72, 122, 85], validated electrospray particle emission models [60, 152, 153, 154], or estimated using theoretical limits accepted in the field [155, 156, 98, 157, 4]. The Lagrangian model, and thereby ML models which utilize Lagrangian simulation data, will more faithfully replicate physical reality when inputs for particle emission properties are able to be discretely measured for each particle - such as experimentally with Phase Doppler Anemometers (PDA) [9], flash shadowgraph

[158], or high speed videography (HSV) [6], or computationally with a validated electrospray emission model [60, 152, 153, 154] - than if they must be estimated with some uncertainty using an analytical approach [155, 98, 4] or experimental measurements which cannot resolve single particles [122], as in electrosprays comprised of particles beneath the diffraction limit [23, 4, 48]. In the case that particle emission property data contains uncertainty introduced by theoretical approximations of unmeasured properties, experimental noise, or inherent uncertainty in experimental measurements, machine learning models can be applied to separate datasets with feature values spanning the ranges of emission properties to establish prediction ranges for final particle properties. Uncertainty quantification within machine learning is an active research field [159, 160, 161], and a field-standard means of quantifying the influence of uncertainty in feature set values on resulting ML predictions has yet to be established.

A testing set of particle data for the trained ML models can be created by choosing an angle and experimentally measuring final particle properties at that angle. The emission properties of particles which terminated at this angle can be a) selected based on discrete particle trajectory results from validated Lagrangian simulations or b) randomly selected from emission property ranges determined experimentally, analytically, or computationally. This particle data generation process can be repeated at each measured angle to generate a data set encompassing the full angular range of the measured plume. The particle data samples can then be fed to trained ML models to predict other final particle properties which the ML models have been shown to predict accurately. In cases where final angle can be accurately regressed from emission properties, ML models can first predict the final angle of a test set of emitted particles, and then the coupled final angle and emission property particle data can be input to the ML models again to predict other final properties.

For example, in the strictly axial emission velocity case in Sec. 8.2.1, measured final angle and emission property data can be used to predict final axial velocity and particle potential. Therefore, for this case, an angular profile measurement, such as that provided by a Faraday probe [23, 162], could be used to estimate the velocity measurements obtained by angularly sweeping a TOF probe, or predict the potential measurements generated by angularly sweeping a RPA device. In this manner, the limited meta-model for the full Lagrangian model created using ML models can be used in conjunction with experimental measurements to predict additional measureable properties. Furthermore, the ML feature ranking results for this case which reveal that final positional angle is of significant value to predicting final axial velocity and final potential are valuable for optimizing diagnostic designs. Experimentalists can infer from these results that a simple Faraday probe can be used in conjunction with an ML meta-model to predict measurements which would be obtained by more complex measurement devices, such as RPA and TOF devices.

In conclusion, the research in this chapter is the first application of ML models to electrospray simulation data, published in [91]. Lagrangian particle tracking data produced by the UCLA PESPL DELI Model for validated simulation cases are used in order to demonstrate the utility of regressing on simulated electrospray particle data. This chapter has identified and exemplified three discrete benefits of applying ML models to electrospray simulation results: (1) the creation of limited meta-models for the full Lagrangian particle tracking model, (2) feature relations between emission and final particle dynamics properties, and (3) the prediction of unknown final particle properties with the assistance of known final properties. We also presented a method to couple ML models trained on validated simulated particle tracking data with experimental measurements to predict unmeasured properties which could be validated experimentally. Our presented processes for creating ML models from Lagrangian particle tracking data can be replicated for any electrospray particle dynamics data set - simulated or measured - therefore, we welcome researchers involved in all approaches to electrospray research, development, and characterization to utilize the presented approach and findings.

The primary obstacle to correlating emission and final particle properties in the context of electrospray particle dynamics stems from the non-linearity of particle trajectories, primarily due to the non-linear Coulomb force between charged particles. Machine learning for non-linear charged particle trajectories is an active area of research exploration, including at the Joint Institute of Nuclear Research [163], the Large Hadron Collider [164, 165], and in medical biophysical research settings [166]. Machine learning predictions of electrospray particle dynamics will improve as the overarching application of ML to nonlinear particle trajectories matures, as electrospray plume evolution models become more representative of experimental realities, and as additional electrospray particle dynamics data is obtained through further plume measurement and simulation efforts.

CHAPTER 9

Conclusion

9.1 Conclusion

We conclude from the overview of the current state of electrospray spacecraft propulsion presented in Ch. 1 that operational lifetime is the primary area in which electrospray thrusters must improve to meet future mission requirements. Propellant overspray to downstream electrodes is the primary life-limiting mechanism in electrospray thrusters. We demonstrated the geometric changes to thruster design can reduce overspray and positively contribute to thruster lifetime; however, we concluded that mitigating mitigating electrospray plume divergence is the most effective means of extending thruster lifetime. To investigate electrospray plume divergence, the Discrete Electrospray Lagrangian Interaction (DELI) Model was developed, verified, and validated as described in Ch. 2.

Chapters 3-6 of this dissertation were devoted to understanding the origins and sources of growth of electrospray plume divergence. Neither applied electrostatic, Coulomb, nor drag forces are found to induce plume divergence in linear sets of droplets down the axis of emission; rather, electrospray droplets are emitted with a small range of radial divergence due to hydrodynamic instabilities and microscopic emitter asymmetries. Chapter 3 discussed plume divergence in response to the electrostatic forces from the potential difference applied between the emitter and downstream electrode(s), concluding that there is a radial component to the electric field due to curvature in the jet and the thrust-aperture in the downstream electrode(s) which magnifies existing divergence in the plume. Chapter 4 investigated plume divergence in response to Coulomb forces between droplets, which have a radial component that grows existing radial divergence in the plume. We compared divergence between sets of particles with equal velocity and with an upstream particle velocity gradient in which upstream particles are moving faster than their forward neighbors and concluded that the latter condition creates a ‘traffic jam’ in which droplet cluster and have magnified Coulomb interactions, resulting in increased plume divergence. We proposed a means of thresholding the ‘interaction region’ in which Coulomb forces are dominant with a ratio of Coulomb to applied electrostatic forces, and demonstrated that such a threshold is species-dependent and a function of axial and radial position.

Chapter 5 investigated plume divergence in response to the drag force, concluding that drag grows existing plume divergence by slowing particle progress and thereby causing increased charge density in the domain. This increase in charge density causes magnified Coulomb interactions, which grows existing plume divergence. Decreasing the background pressure decreases the density of background fluid molecules and thereby the drag force, confining the plume in response. Pressure thresholds beyond which drag no longer contributes to plume divergence were proposed based on simulation results and the theoretical definition of Free Molecular Flow. Similarly, the pressure at which particle mean free path is greater than the domain length was presented for several domain lengths, such that electrospray particles can move through the full domain without colliding with background molecules. Chapter 6 discusses the influence of forces and phenomena (gravity, thermal gradients, fluid mechanics, and gas polarization) on plume divergence which are not included in the DELI Model. The neglected forces are concluded to have a much smaller influence on plume evolution and divergence than those included in the DELI Model, although they could be included in the future for completeness as discussed in Sec. 9.2.

Chapter 7 presented new means of characterizing plume divergence: a positional divergence definition based on three standard deviations of a Gaussian or Super-Gaussian fit to particle mass distribution over radial coordinate, and a 2D position and velocity divergence measurement called emittance which is frequently used in particle accelerators[88, 89, 22]. We concluded that steadiness in an emittance diagram is a means of determining that a simulated plume has reached steady state.

Finally, chapter 8 presented the first application of machine learning to simulated particle dynamics data, produced by the DELI Model. We concluded three primary benefits of applying machine learning to simulated particle dynamics data: the creation of a surrogate model for a Lagrangian particle tracking model, the provision of feature relations between emission and final particle properties, and the prediction of unknown final particle properties given known final properties. We proposed a novel method for combining experimental data, Lagrangian plume evolution models, and machine learning algorithms to optimize diagnostic design.

Overall, this dissertation has identified the origins and sources of growth of electrospray plume divergence, proposed novel means of characterizing plume divergence, and applied machine learning to predict plume divergence. The DELI Model developed during this dissertation to simulate electrospray plume evolution is validated against published experimental data, and the code is presented in App. F for future plume evolution and divergence research, some of which is proposed in the following Sec. 9.2.

9.2 Future Work

I have no shortage of ideas to continue this research, but I do have a shortage of time remaining in my degree. In this section, I propose future work which builds upon this dissertation research.

9.2.1 Chapter 6 Considerations

Chapter 6 was devoted to forces and phenomena which occur in physical electrosprays but are not included in the DELI Model: gravity, thermophoretic forces, fluid mechanical effects, and gas polarization. Including these phenomena would improve the accuracy of the DELI model. The governing equation of the DELI Model can be modified to include gravity and thermophoretic forces as shown in Eq. 6.2 and Eq. 6.5, respectively. Fluid mechanical effects and gas polarization have thus far only been incorporated into machine learning models of individual or small groups of particles due to their computational intensity. However, as computational capabilities improve, they may be included in full electrospray plume models such as the DELI Model in the future. Additional considerations of atmospheric effects, such as induced flow in the background gas, which have been included in atmospheric electrospray models such as those for mass spectrometry[54], could also be incorporated into the DELI Model in the future.

9.2.2 Plume Evolution Studies

Appendix D presents studies of plume evolution in response to changing mass flowrate, mean specific charge, and specific charge inhomogeneity. Chapter 5 also presents studies of plume evolution in response to changing the drag force (directly and via the background pressure). Many more variable studies on plume evolution could be conducted in the DELI Model - any variable which is input to the model as an independent variable (emitted particle properties, flowrate constraints, electrode geometry and resulting electric field, background pressure, background fluid properties, etc.) could be varied, and the result on plume evolution observed. Such variable studies have served as excellent high school and undergraduate research projects throughout this dissertation.

9.2.3 Include Secondaries

Electrospray particles impinge on surfaces inside the thruster, such as downstream electrodes, and generate secondary particles [67, 65, 66]. These secondary particles can be charged, such that they have Coulomb interactions with electrospray particles. They can also be oppositely charged to primary emitted electrospray particles, such that they are accelerated in the opposite direction by the applied electric field. In this manner, negatively charged secondaries generated from the downstream electrode are electrostatically drawn upstream towards the emitter. The presence of charged secondary particles in the electrospray thruster domain changes the dynamics of primary emitted electrospray particles. Therefore, including such secondaries would be a useful future addition to the DELI Model. An electron population of secondaries generated from the downstream electrode could be included in the DELI model by calling the same function used to emit primary particles, but with different emission properties, such as emission location set to downstream electrode location. The emission mass, charge, and velocity distributions could be based on experimental secondary distributions or analytical relationships given the properties of primary particles which strike the downstream electrode[66].

9.2.4 Transient Modes

This dissertation has focused on simulating electrospray plume evolution in the optimal steady cone-jet mode. Simulations have included the start-up mode, from no flow to steady state, such as in Ch. 5. Simulations were also conducted investigating the shut-down electrospray mode from steady state to no flow. The results for number of particles in the simulation domain from start-up, to steady state, to shut-down are shown in Fig. 9.1. DELI Model simulations of start-up and shut-down have thus far instantaneously turned the flow on and off. In the future, more realistic flowrates which are functions of time could be instituted. Pulsating mode electrosprays in which the emitted flowrate is inconsistent over time could also be simulated. Correlating the frequency and magnitude of flowrate pulsation with maximum divergence angle obtained by the plume would be a very interesting future study with potentially useful thrust applications.

In addition to electrospray operational modes which are temporally unstable, the DELI Model could also simulate electrospray modes which change spatially, such as the whipping

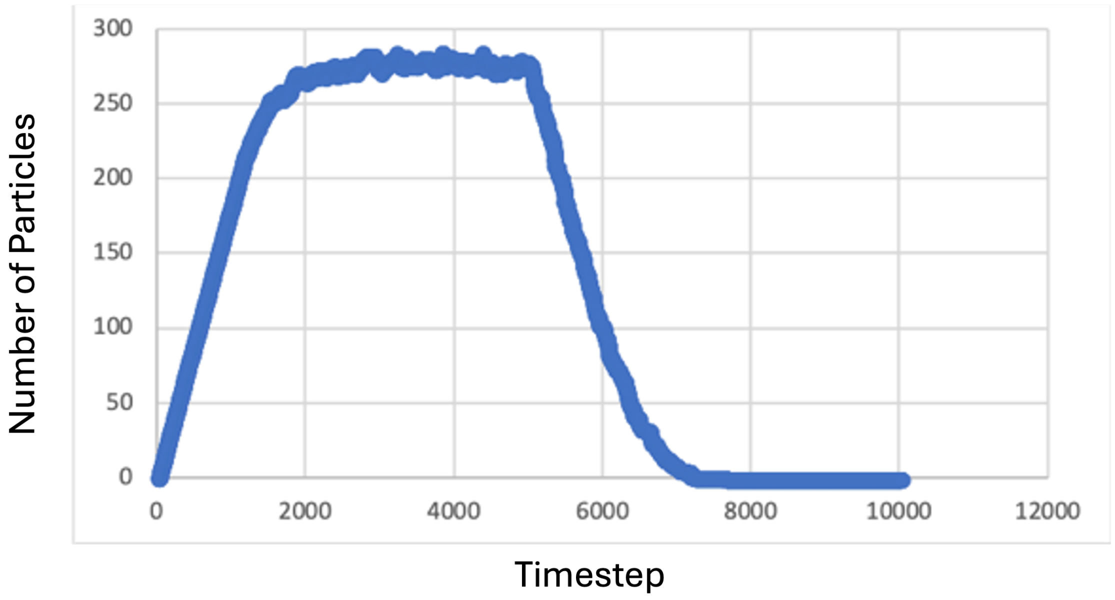


Figure 9.1: The number of particles in an electrospray plume simulated in the DELI Model from the start-up for flow, to steady state, to flow shut-down.

jet mode shown in Fig. 9.2. This mode could be simulated by changing the position and velocity angle at which particles are emitted over time. The DELI Model could also simulate operational modes which are spatially stable, but not operationally ideal, such as the multi-jet mode shown in Fig. 9.3. In a mission setting, electrospray thrusters will go through many operational modes, including start-up, shut-down, and steady cone-jet at different flowrates. Furthermore, the pressure and temperature conditions in which the thruster operates may change. The DELI Model is capable of being applied in the future to mission-relevant simulations with time-varying emission and environment conditions.

9.2.5 Parallelization

The DELI Model is currently fully sequential. Particle tracking models have a legacy of parallelization, which greatly increases simulation efficiency[167, 168, 169]. Parallelization has been employed in in particle-in-cell (PIC) electrospray plume models [170]. Paralleliza-

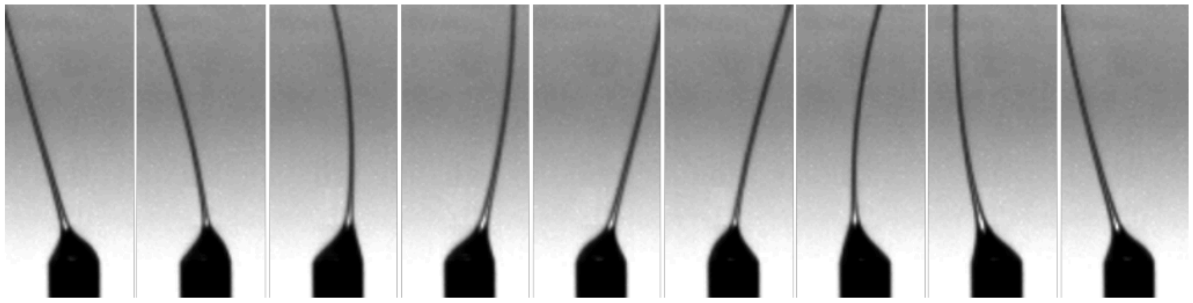


Figure 9.2: Whipping mode of electrospray jet operation[6].

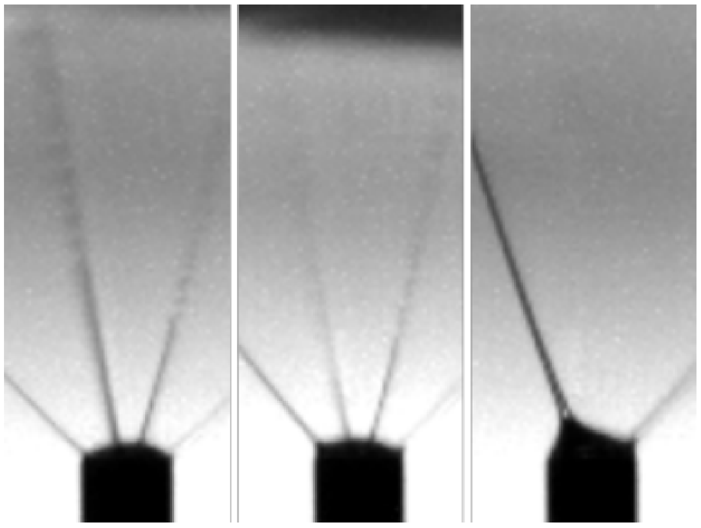


Figure 9.3: Examples of the multi-jet mode of electrospray jet operation[6].

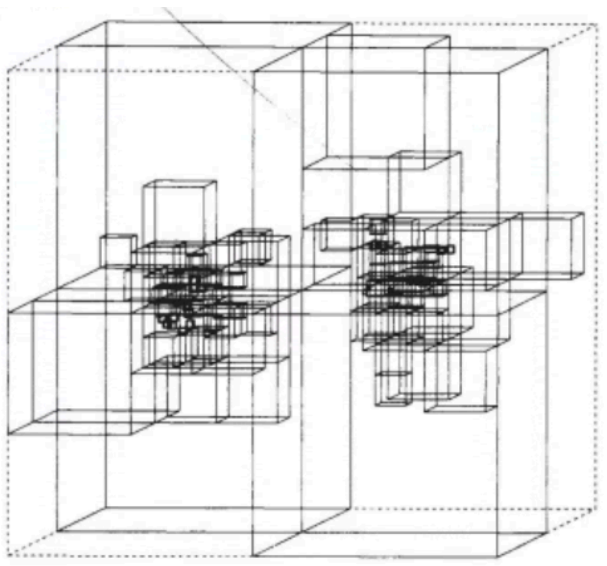


Figure 9.4: Particles are divided into cells with an octree algorithm to parallelize the simulation of an n-body problem.

tion of the DELI Model with OpenMP and MPI was considered and discussed with AFRL mentors during the course of this dissertation research, but not implemented due to time constraints and the prioritization of other research efforts. In the future, implementing parallelization into the DELI Model would decrease simulation runtime, allowing for a faster pace of research. One specific area of the model in which parallelization could be implemented is in approximating the Coulomb term outside the interaction region bounded in Sec, 4.4. Octree methods can reduce the order of n-body problems such as Coulomb interactions by dividing particles into cells, treating particles in neighboring cells individually, and treating particles in distant cells as single large particles as the cell center-of-mass, as shown in Fig. 9.4. Such methods can reduce the order of calculation in an n-body problem from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ for the Barnes-Hut algorithm[167], and from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ for the Fast Multiple Method[168]. Parallelization would also be useful in the case that the DELI Model is extended to simulate multi-plexed electrosprays[49, 50].

9.2.6 Machine Learning

Chapter 8 presented the first application of machine learning to simulated electrospray dynamics data; there is so much more which can be done. Regression on DELI Model particle dynamics data could be applied in the reverse direction as in Ch. 8, predicting particle emission properties from 'final' properties at the collector plate. Machine learning algorithms could be trained on data sets of different input mass flowrates, applied electric fields, or particle properties, and the resulting DELI Model plume structures (quantified through mass flux and charge density profiles, such as those presented in Sec. D.3) in order to predict the electrospray plume structure which results from given input conditions. Classification algorithms could also be applied to DELI Model data in useful ways, such as to identify and group particles which reach wide plume angles and contribute to overspray, or which strike the downstream electrode with enough kinetic energy to produce secondaries. The possibilities for applying machine learning to DELI Model data are as wide as the simulation possibilities for the model.

Appendix A

Empirical Drag Terms

Empirical coefficient of drag terms were selected from the literature for use in the DELI simulations over different background pressures in Sec. 5.2.2. The empirical terms, domain of flow parameters, and references from which the terms were selected are given in Table A.1.

C_D	Re Domain	Kn Domain	Reference
$\frac{24}{Re}$	$Re < 1$	$Kn < 0.001$	[17]
$\frac{24}{Re}\left(\frac{1+2\frac{2-C_{TM}}{C_{TM}}Kn}{1+3\frac{2-C_{TM}}{CTM}Kn}\right)$	$Re < 1$	$0.01 \leq Kn < 0.1$	[18]
$0.127 + \frac{3.957}{0.150+Re^{0.983}}\left(\frac{7.407Kn+2.293}{1.688Kn+0.292}\right)$	$1 \leq Re \leq 50$	$0.001 \leq Kn < 0.1$	[19]
$\frac{24(1+0.15Re^{0.687})e^{a_1+a_2Re^{a_3}+a_4Kn^{a_5}+a_6ReKn}}{Re[1+2.0Kn(1.142+0.558e^{-0.4995/Kn})]}$	$1 \leq Re \leq 3.5$	$0.1 \leq Kn < 1$	[20]

Table A.1: The empirical coefficient of drag terms used in DELI Model simulations.

In the table, $\sigma_T = 1$ is the tangential momentum accommodation coefficient[18], and $a_1 = 1.40671$, $a_2 = -1.50268$, $a_3 = 0.01803$, $a_4 = 0.00118$, $a_5 = -0.79338$, and $a_6 = 0.09905$ [20]. In addition to these empirical terms with discrete flow parameter ranges, the coefficient of drag described in [16] was also used in Sec. 5.2.2, which is valid for all Knudsen numbers. Below $P = 0.007692$ Atm, mean particle Knudsen number exceeds one and the only applicable coefficient of drag in Table A.1 is from Loth$_0$8.$Therefore, only the Loth_0 8 coefficient of drag term was used to$ $= 0.005, 0.002, 0.001, 0.0005, 0.0002$ *and* 0.0001 *Atmcases*.

Appendix B

Machine Learning Histograms

Histograms of the feature variables for the six emission velocity cases, ranging from strictly axial emission velocity to increasingly radial emission velocity vectors. The feature variables are mass m, charge q, 3D emission coordinates (x_i, y_i, z_i), 3D emission velocity (v_{xi}, v_{y_i}, v_{zi}), final positional plume angle θ_f, 3D final velocity (v_{xf}, v_{y_f}, v_{zf}), and final potential ϕ_{RPAf}.

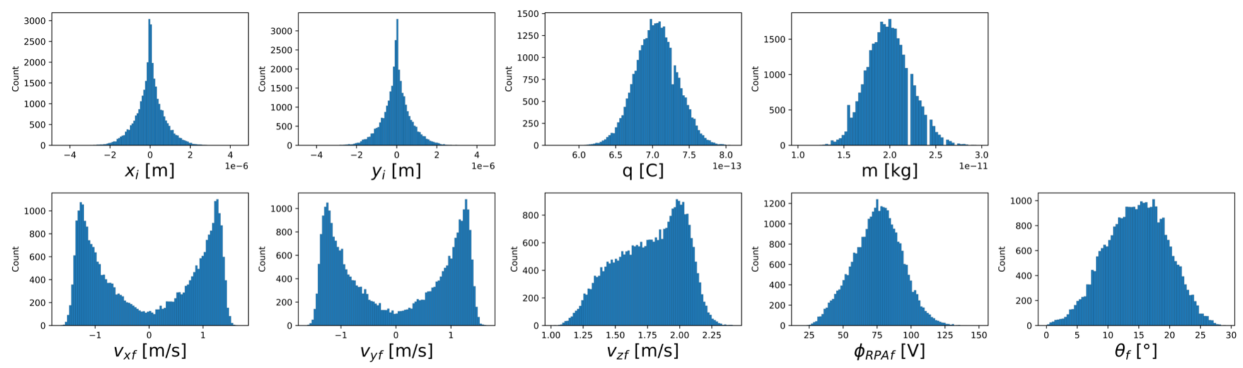


Figure B.1: Histograms of each particle data feature for the case where emission velocity is strictly axial.

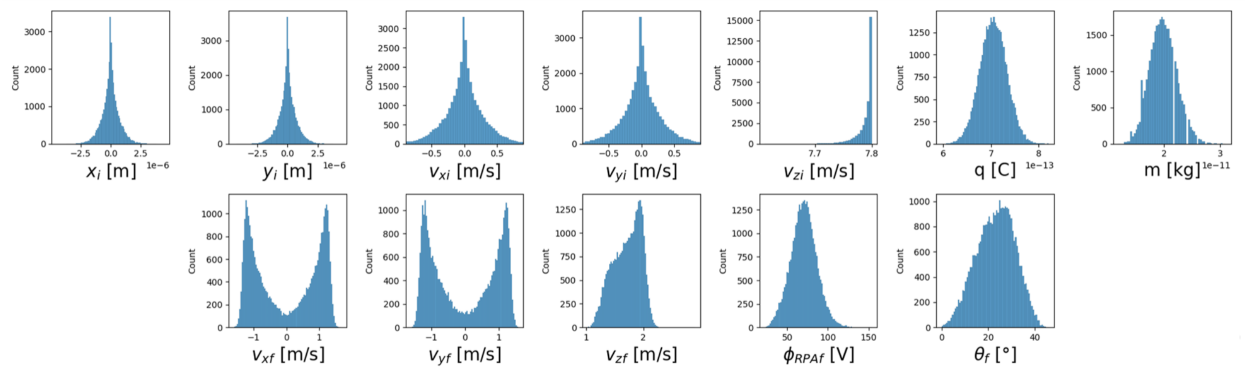


Figure B.2: Histograms of each particle data feature for the case where the distance between the jet tip and the reference point for emission velocity angle is equal to the mean particle radius: $l = d/2$.

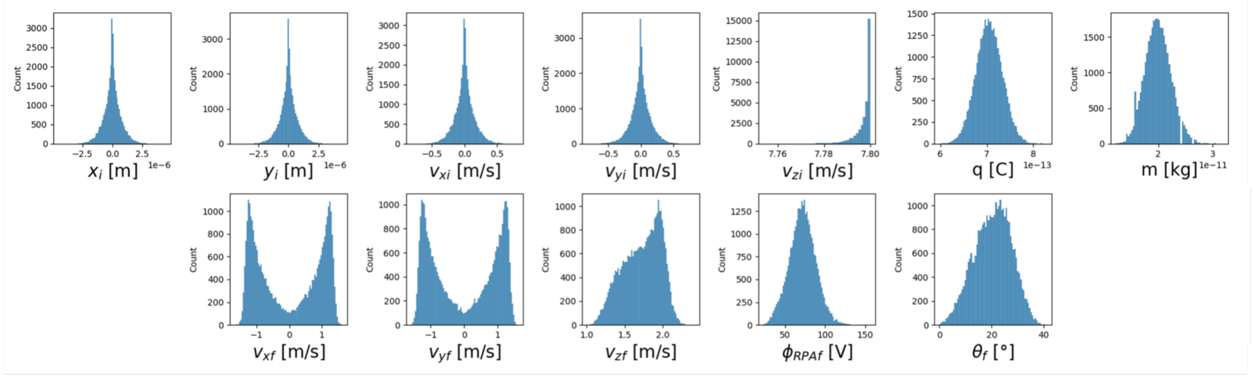


Figure B.3: Histograms of each particle data feature for the case where the distance between the jet tip and the reference point for emission velocity angle is equal to the mean particle diameter: $l = d$.

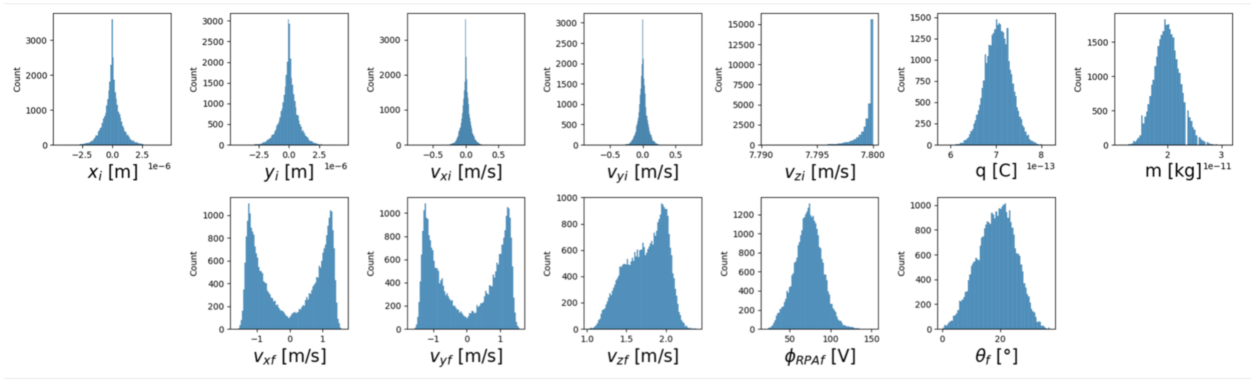


Figure B.4: Histograms of each particle data feature for the case where the distance between the jet tip and the reference point for emission velocity angle is equal to half the mean jet breakup wavelength: $l = \lambda/2$.

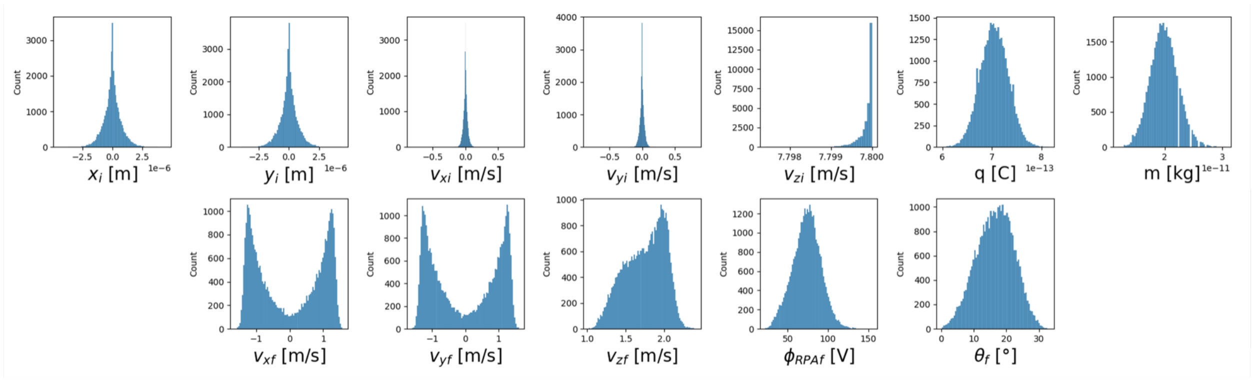


Figure B.5: Histograms of each particle data feature for the case where the distance between the jet tip and the reference point for emission velocity angle is equal to the mean jet breakup wavelength: $l = \lambda$.

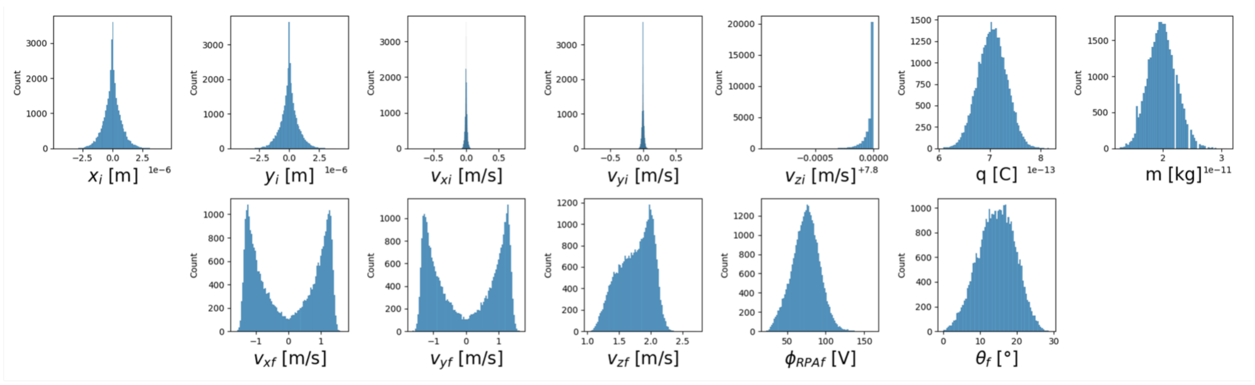
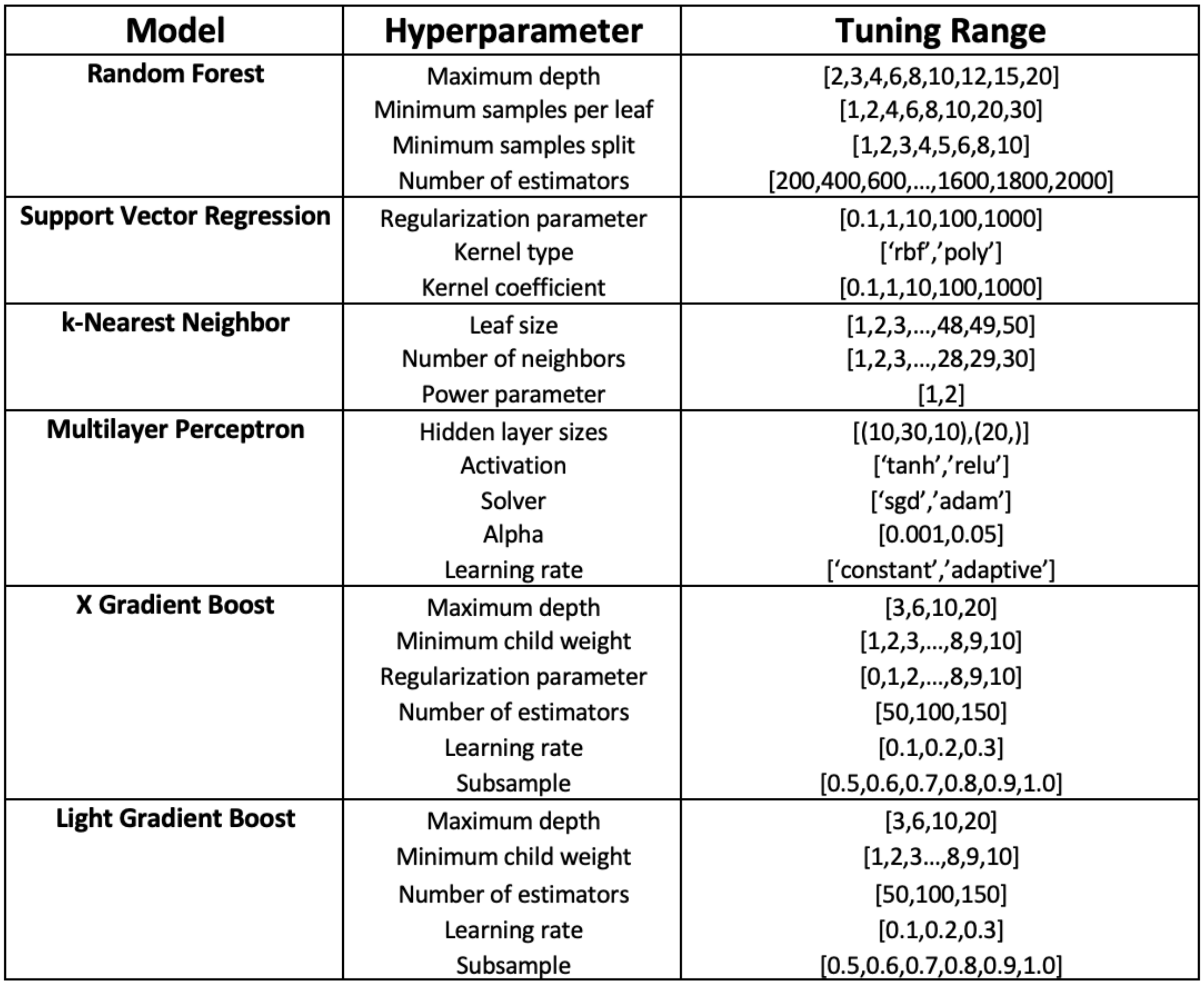


Figure B.6: Histograms of each particle data feature for the case where the distance between the jet tip and the reference point for emission velocity angle is equal to twice the mean jet breakup wavelength: $l = 2\lambda$.

Appendix C

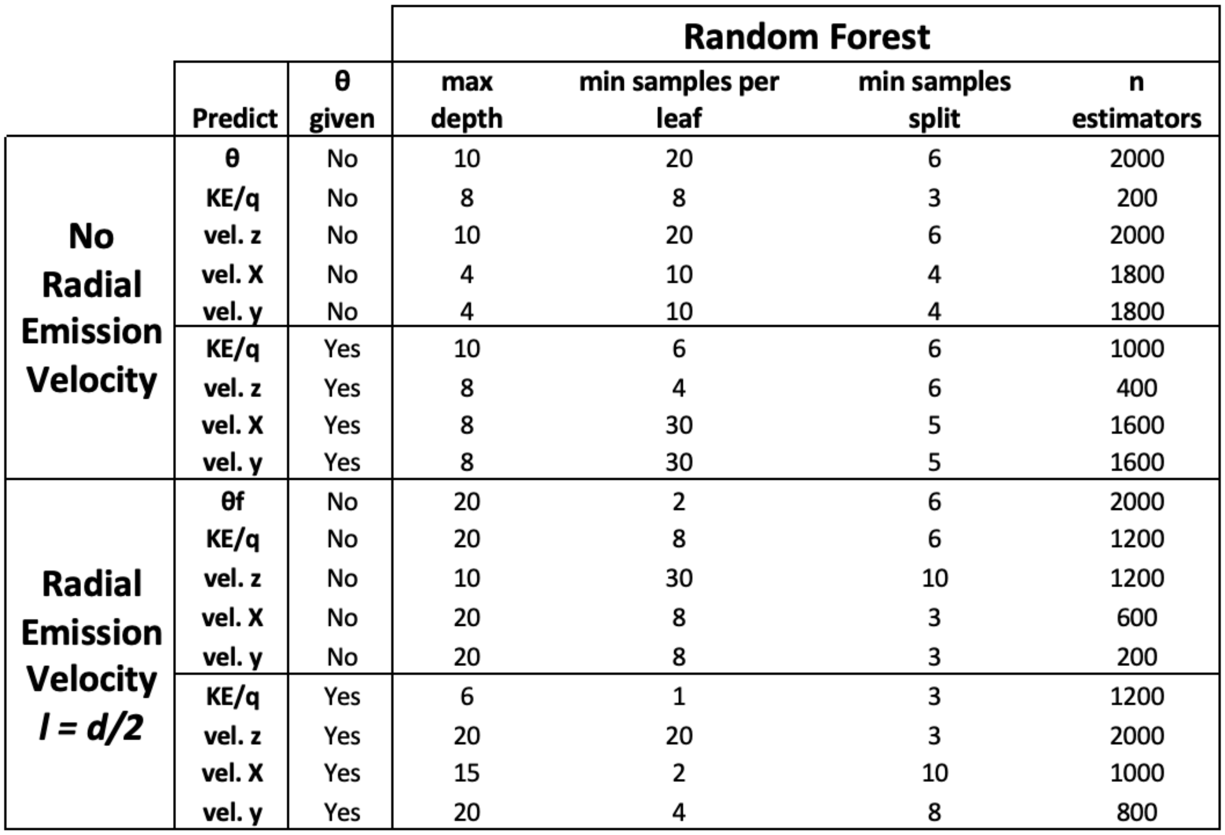
Machine Learning Hyperparameter Tuning

The algorithm hyperparameters, their tuning ranges, and their optimal values for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases are given for the six machine learning algorithms used in this study: Random Forest, Support Vector Regression, K-Nearest Neighbors, Multi-Layer Perceptron, Extreme Gradient Boosting Method, and Light Gradient Boosting Method.



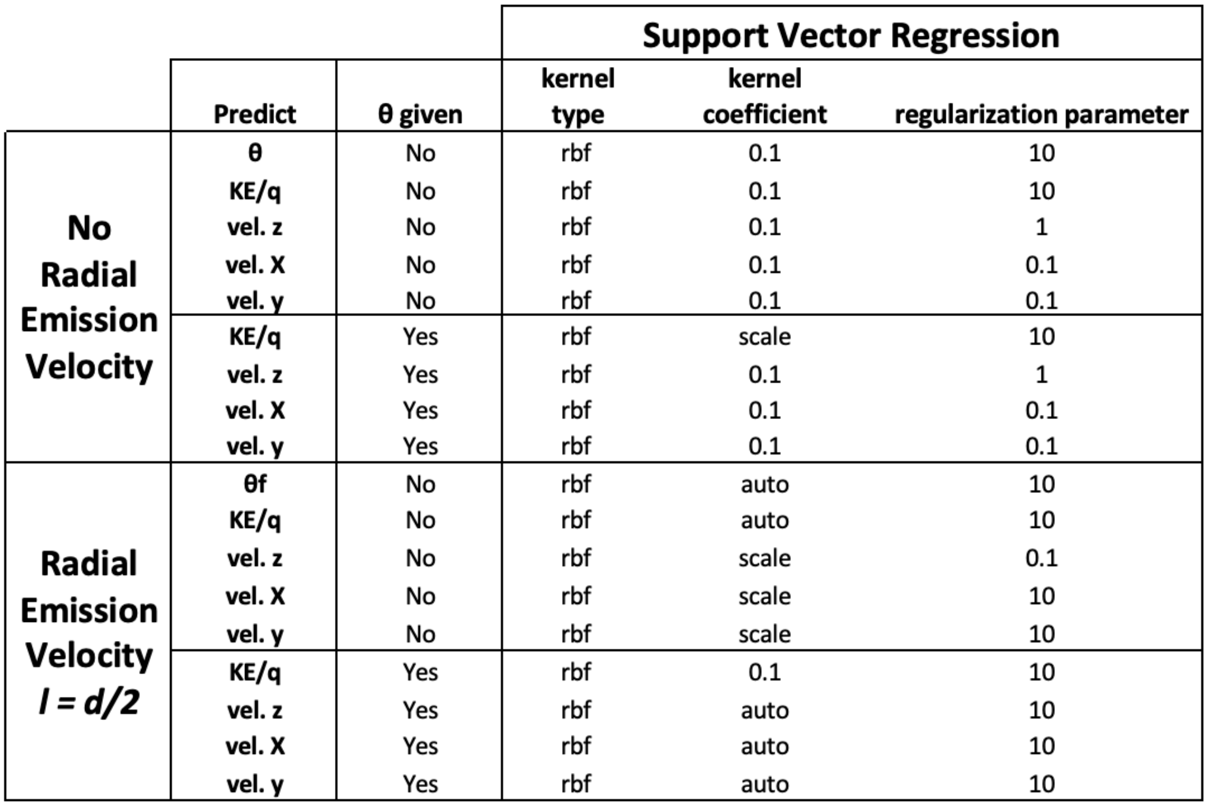
Model	Hyperparameter	Tuning Range
Random Forest	Maximum depth	[2,3,4,6,8,10,12,15,20]
	Minimum samples per leaf	[1,2,4,6,8,10,20,30]
	Minimum samples split	[1,2,3,4,5,6,8,10]
	Number of estimators	[200,400,600,...,1600,1800,2000]
Support Vector Regression	Regularization parameter	[0.1,1,10,100,1000]
	Kernel type	['rbf','poly']
	Kernel coefficient	[0.1,1,10,100,1000]
k-Nearest Neighbor	Leaf size	[1,2,3,...,48,49,50]
	Number of neighbors	[1,2,3,...,28,29,30]
	Power parameter	[1,2]
Multilayer Perceptron	Hidden layer sizes	[(10,30,10),(20,)]
	Activation	['tanh','relu']
	Solver	['sgd','adam']
	Alpha	[0.001,0.05]
	Learning rate	['constant','adaptive']
X Gradient Boost	Maximum depth	[3,6,10,20]
	Minimum child weight	[1,2,3,...,8,9,10]
	Regularization parameter	[0,1,2,...,8,9,10]
	Number of estimators	[50,100,150]
	Learning rate	[0.1,0.2,0.3]
	Subsample	[0.5,0.6,0.7,0.8,0.9,1.0]
Light Gradient Boost	Maximum depth	[3,6,10,20]
	Minimum child weight	[1,2,3...,8,9,10]
	Number of estimators	[50,100,150]
	Learning rate	[0.1,0.2,0.3]
	Subsample	[0.5,0.6,0.7,0.8,0.9,1.0]

Figure C.1: Hyperparameters and their tuning ranges for each of the six utilized ML models.



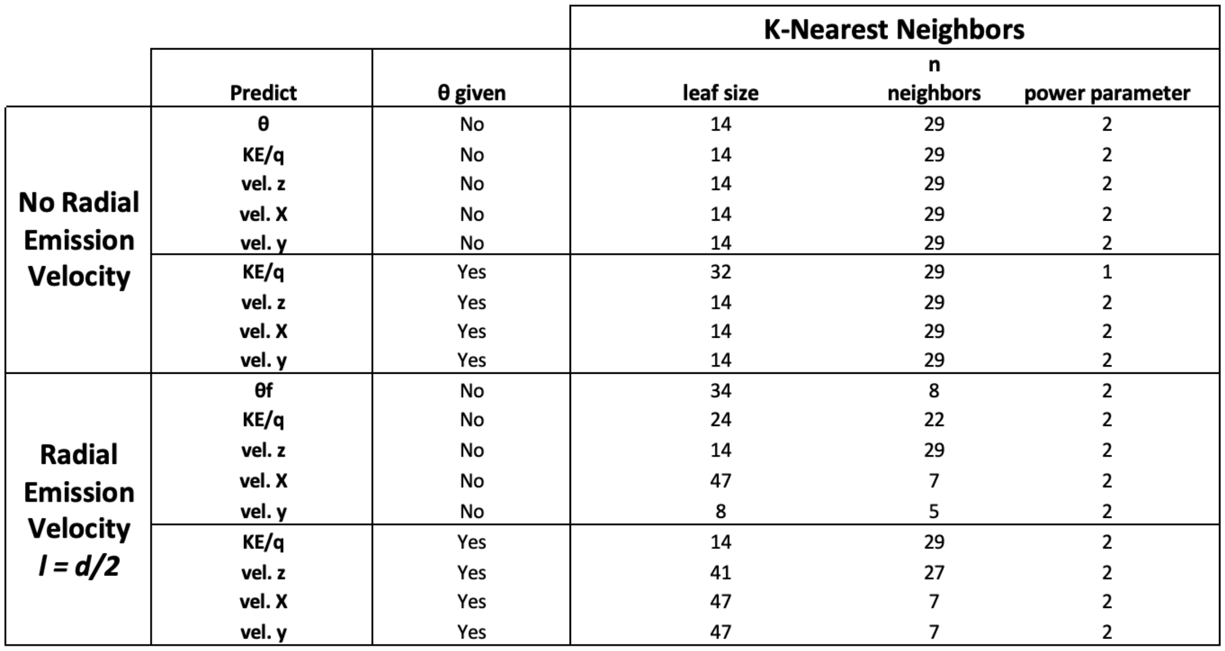
			Random Forest			
	Predict	θ given	max depth	min samples per leaf	min samples split	n estimators
No Radial Emission Velocity	θ	No	10	20	6	2000
	KE/q	No	8	8	3	200
	vel. z	No	10	20	6	2000
	vel. X	No	4	10	4	1800
	vel. y	No	4	10	4	1800
	KE/q	Yes	10	6	6	1000
	vel. z	Yes	8	4	6	400
	vel. X	Yes	8	30	5	1600
	vel. y	Yes	8	30	5	1600
Radial Emission Velocity *l = d/2*	θf	No	20	2	6	2000
	KE/q	No	20	8	6	1200
	vel. z	No	10	30	10	1200
	vel. X	No	20	8	3	600
	vel. y	No	20	8	3	200
	KE/q	Yes	6	1	3	1200
	vel. z	Yes	20	20	3	2000
	vel. X	Yes	15	2	10	1000
	vel. y	Yes	20	4	8	800

Figure C.2: Optimal hyperparameter settings for the Random Forest algorithm for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases, when final angle is given and when it is not. All variables in the prediction column are final state variables.



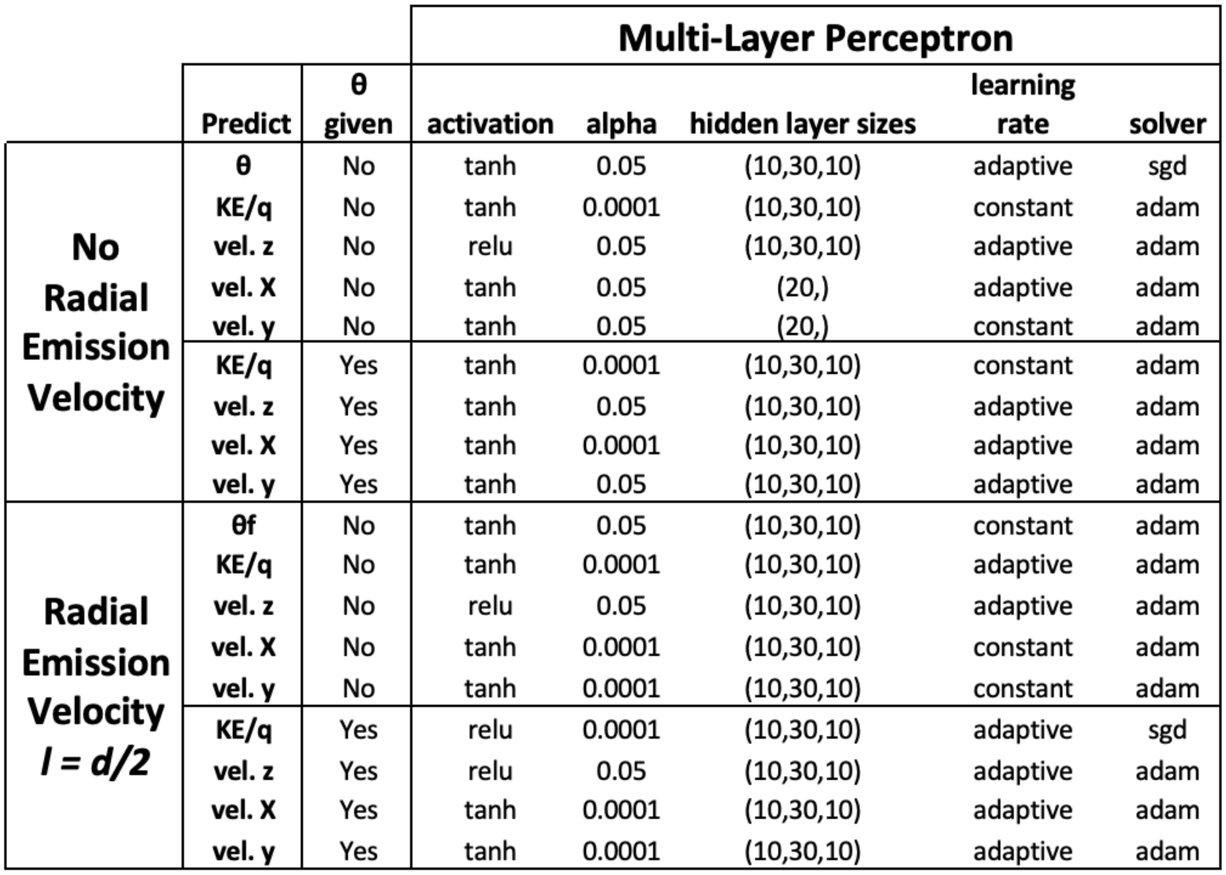
			Support Vector Regression		
	Predict	**θ given**	**kernel type**	**kernel coefficient**	**regularization parameter**
No Radial Emission Velocity	**θ**	No	rbf	0.1	10
	KE/q	No	rbf	0.1	10
	vel. z	No	rbf	0.1	1
	vel. X	No	rbf	0.1	0.1
	vel. y	No	rbf	0.1	0.1
	KE/q	Yes	rbf	scale	10
	vel. z	Yes	rbf	0.1	1
	vel. X	Yes	rbf	0.1	0.1
	vel. y	Yes	rbf	0.1	0.1
Radial Emission Velocity *l = d/2*	**θf**	No	rbf	auto	10
	KE/q	No	rbf	auto	10
	vel. z	No	rbf	scale	0.1
	vel. X	No	rbf	scale	10
	vel. y	No	rbf	scale	10
	KE/q	Yes	rbf	0.1	10
	vel. z	Yes	rbf	auto	10
	vel. X	Yes	rbf	auto	10
	vel. y	Yes	rbf	auto	10

Figure C.3: Optimal hyperparameter settings for the Support Vector Regression algorithm for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases, when final angle is given and when it is not.



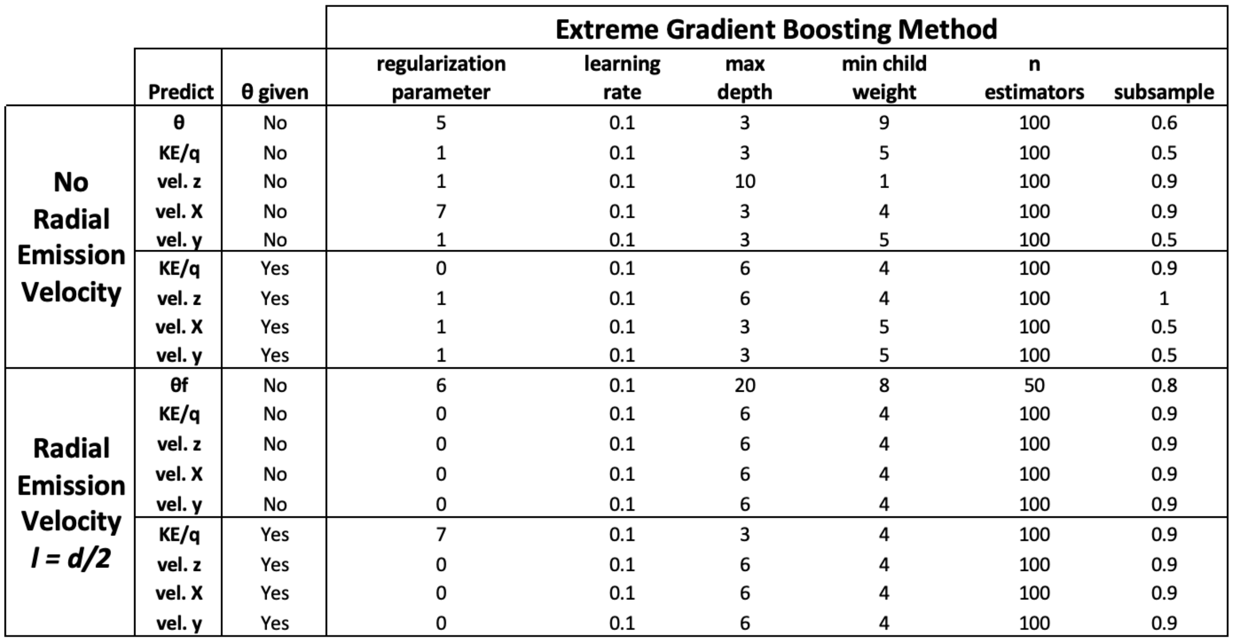
			K-Nearest Neighbors		
	Predict	θ given	leaf size	n neighbors	power parameter
No Radial Emission Velocity	θ	No	14	29	2
	KE/q	No	14	29	2
	vel. z	No	14	29	2
	vel. X	No	14	29	2
	vel. y	No	14	29	2
	KE/q	Yes	32	29	1
	vel. z	Yes	14	29	2
	vel. X	Yes	14	29	2
	vel. y	Yes	14	29	2
Radial Emission Velocity *l = d/2*	θf	No	34	8	2
	KE/q	No	24	22	2
	vel. z	No	14	29	2
	vel. X	No	47	7	2
	vel. y	No	8	5	2
	KE/q	Yes	14	29	2
	vel. z	Yes	41	27	2
	vel. X	Yes	47	7	2
	vel. y	Yes	47	7	2

Figure C.4: Optimal hyperparameter settings for the K-Nearest Neighbors algorithm for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases, when final angle is given and when it is not.



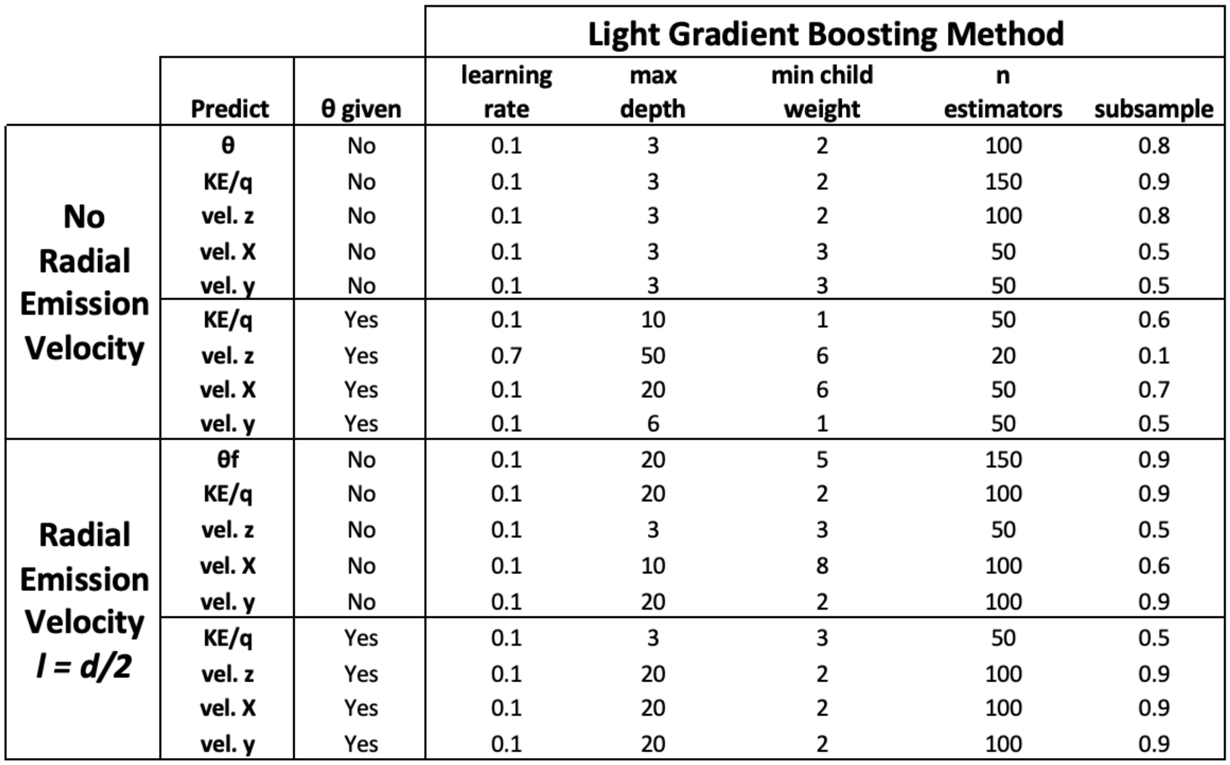
			Multi-Layer Perceptron				
	Predict	θ given	activation	alpha	hidden layer sizes	learning rate	solver
No Radial Emission Velocity	θ	No	tanh	0.05	(10,30,10)	adaptive	sgd
	KE/q	No	tanh	0.0001	(10,30,10)	constant	adam
	vel. z	No	relu	0.05	(10,30,10)	adaptive	adam
	vel. X	No	tanh	0.05	(20,)	adaptive	adam
	vel. y	No	tanh	0.05	(20,)	constant	adam
	KE/q	Yes	tanh	0.0001	(10,30,10)	constant	adam
	vel. z	Yes	tanh	0.05	(10,30,10)	adaptive	adam
	vel. X	Yes	tanh	0.0001	(10,30,10)	adaptive	adam
	vel. y	Yes	tanh	0.05	(10,30,10)	adaptive	adam
Radial Emission Velocity *l = d/2*	θf	No	tanh	0.05	(10,30,10)	constant	adam
	KE/q	No	tanh	0.0001	(10,30,10)	adaptive	adam
	vel. z	No	relu	0.05	(10,30,10)	adaptive	adam
	vel. X	No	tanh	0.0001	(10,30,10)	constant	adam
	vel. y	No	tanh	0.0001	(10,30,10)	constant	adam
	KE/q	Yes	relu	0.0001	(10,30,10)	adaptive	sgd
	vel. z	Yes	relu	0.05	(10,30,10)	adaptive	adam
	vel. X	Yes	tanh	0.0001	(10,30,10)	adaptive	adam
	vel. y	Yes	tanh	0.0001	(10,30,10)	adaptive	adam

Figure C.5: Optimal hyperparameter settings for the Multi-Layer Perceptron algorithm for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases, when final angle is given and when it is not.



	Predict	θ given	regularization parameter	learning rate	max depth	min child weight	n estimators	subsample
			Extreme Gradient Boosting Method					
No Radial Emission Velocity	**θ**	No	5	0.1	3	9	100	0.6
	KE/q	No	1	0.1	3	5	100	0.5
	vel. z	No	1	0.1	10	1	100	0.9
	vel. X	No	7	0.1	3	4	100	0.9
	vel. y	No	1	0.1	3	5	100	0.5
	KE/q	Yes	0	0.1	6	4	100	0.9
	vel. z	Yes	1	0.1	6	4	100	1
	vel. X	Yes	1	0.1	3	5	100	0.5
	vel. y	Yes	1	0.1	3	5	100	0.5
Radial Emission Velocity *l = d/2*	**θf**	No	6	0.1	20	8	50	0.8
	KE/q	No	0	0.1	6	4	100	0.9
	vel. z	No	0	0.1	6	4	100	0.9
	vel. X	No	0	0.1	6	4	100	0.9
	vel. y	No	0	0.1	6	4	100	0.9
	KE/q	Yes	7	0.1	3	4	100	0.9
	vel. z	Yes	0	0.1	6	4	100	0.9
	vel. X	Yes	0	0.1	6	4	100	0.9
	vel. y	Yes	0	0.1	6	4	100	0.9

Figure C.6: Optimal hyperparameter settings for the Extreme Gradient Boosting Method algorithm for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases, when final angle is given and when it is not.



			Light Gradient Boosting Method				
	Predict	θ given	learning rate	max depth	min child weight	n estimators	subsample
No Radial Emission Velocity	θ	No	0.1	3	2	100	0.8
	KE/q	No	0.1	3	2	150	0.9
	vel. z	No	0.1	3	2	100	0.8
	vel. X	No	0.1	3	3	50	0.5
	vel. y	No	0.1	3	3	50	0.5
	KE/q	Yes	0.1	10	1	50	0.6
	vel. z	Yes	0.7	50	6	20	0.1
	vel. X	Yes	0.1	20	6	50	0.7
	vel. y	Yes	0.1	6	1	50	0.5
Radial Emission Velocity *l = d/2*	θf	No	0.1	20	5	150	0.9
	KE/q	No	0.1	20	2	100	0.9
	vel. z	No	0.1	3	3	50	0.5
	vel. X	No	0.1	10	8	100	0.6
	vel. y	No	0.1	20	2	100	0.9
	KE/q	Yes	0.1	3	3	50	0.5
	vel. z	Yes	0.1	20	2	100	0.9
	vel. X	Yes	0.1	20	2	100	0.9
	vel. y	Yes	0.1	20	2	100	0.9

Figure C.7: Optimal hyperparameter settings for the Light Gradient Boosting Method algorithm for the solely axial emission velocity and the $l = d/2$ radial emission velocity cases, when final angle is given and when it is not.

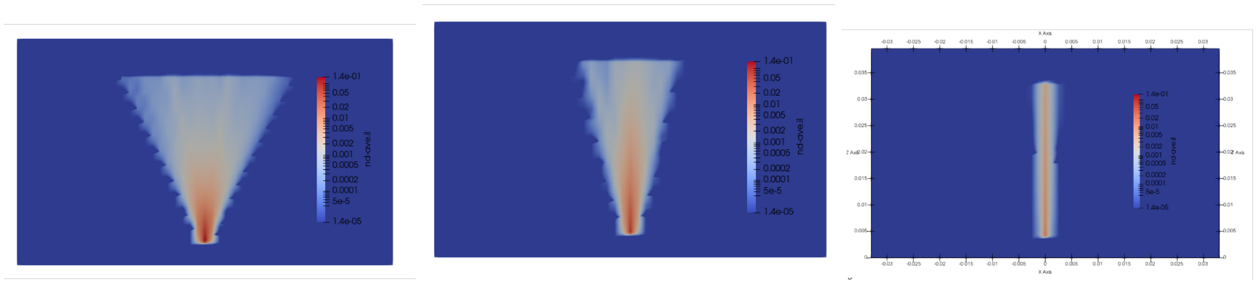
Appendix D

Plume Evolution Studies

Plume evolution in response to changes in flowrate and emitted particle properties were conducted with the DELI Model throughout the course of this dissertation research. The goal of these studies was to gain a better understanding of how these properties affect electrospray plume evolution. The properties studies presented in this section are on mass flowrate of the emitted fluid, mean specific charge of the emitted fluid, and specific charge inhomogeneity in the emitted particle population. Targeted attention was given to specific charge because inhomogeneity in specific charge has been identified to contribute to plume divergence through 'traffic jams' and Coulomb interactions, as discussed in Sec. 4.3.

D.1 Mass Flowrate

A study on the influence of mass flowrate on plume evolution was conducted with the DELI model using the atmospheric validation case in Sec. 2.5 as the control case. Emitted species properties are held constant despite changes in mass flowrate, as the electrohydrodynamic process of particle emission is beyond the scope of this dissertation. Fig. D.1 presents mass density contours, time-averaged from when the plume reaches steady-state, of plumes with 100%, 50%, and 10% of the mass flowrate from the validation Sec. 2.5. Plume divergence is seen to increase with increasing mass flowrate. This follows the theory of plume divergence through Coulomb interactions presented in Sec. 4.3, as increased mass flowrate causes increased particle number density in the interaction region near emission.



(a) (b) (c)

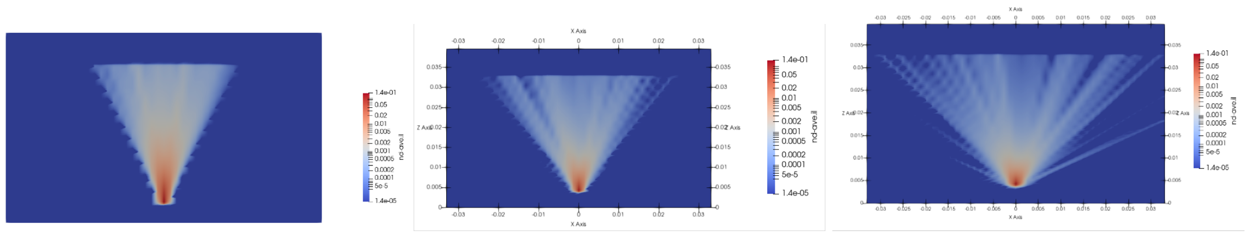
Figure D.1: Mass density contours of electrospray plumes with a) 100%, b) 50%, and c) 10% of the mass flowrate from the validation Sec. 2.5.

D.2 Mean Specific Charge

A study on the influence of the mean specific charge of emitted particles on evolved plume evolution was conducted with the DELI model using the atmospheric validation case in Sec. 2.5 as the control case. All other emitted particle properties beyond the mean specific charge were not changed from the validation case. Fig. D.2 presents mass density contours, time-averaged from when the plume reaches steady-state, of plumes with the same, double, and triple the mean specific charge of emitted particle from the validation Sec. 2.5. Plume divergence is seen to increase with increasing mean specific charge. This follows the theory of plume divergence through Coulomb interactions presented in Sec. 4.3, as increasing mean specific charge of emitted particles magnifies the strength of their Coulomb interactions. Because the particles have increased in charge but not mass, they are more displaced in response to such increased Coulomb forces.

D.3 Specific Charge Inhomogeneity

In Sec. 4.3, we discussed how specific charge inhomogeneity in an electrospray plume population contributes positively to plume divergence because applied electrostatic forces, and drag forces if present, accelerate particles differently according to their specific charge. Therefore,



(a) (b) (c)

Figure D.2: Mass density contours of electrospray plumes with a) the same, b) double, and c) triple mean specific charge of emitted particle from the validation Sec. 2.5.

velocity differences are introduced into specific charge inhomogeneous plumes, which cause particles to cluster and displace one another through proximity-magnified Coulomb interactions. In agreement with this theory, experimental mass flux profiles have been observed to take on different shapes than charge density profiles of the same plume as presented in Fig. D.3. This difference in profile shapes suggests that 1) the plume is specific charge inhomogeneous and 2) particles are displaced differently according to their specific charge. To simulate charge inhomogeneous plume evolution, the DELI simulation utilized the Gaussian particle size distribution from Sec. 2.5 and a separate Gaussian particle charge distribution with the mean charge from Sec. 2.5. A snapshot of the resulting plume is presented in Fig. D.4, with two lines displaying the cross sections at which profiles measurements were taken. The mass flux density and current density profiles taken at those locations are presented in Fig. D.5. The two profiles can be seen to diverge in shape as the plume moves downstream, matching the trend demonstrated by the experimental results.

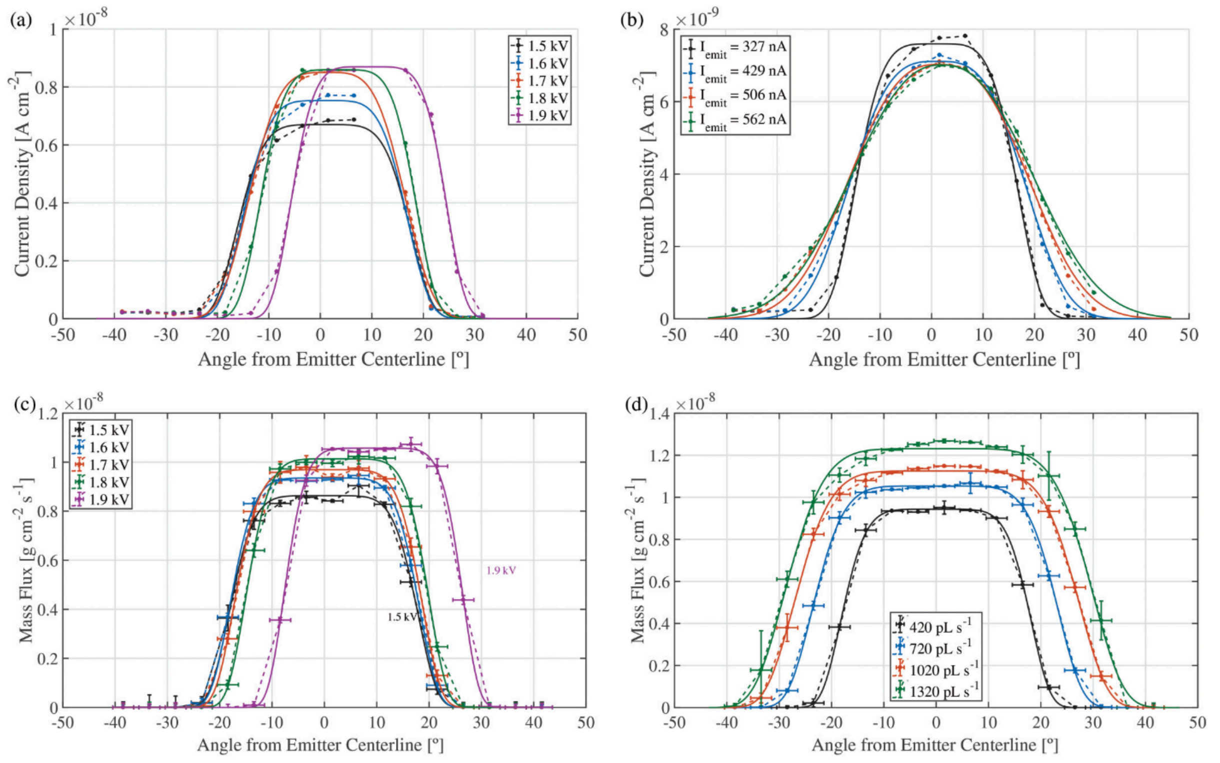


Figure D.3: Current density as a function of half angle for varying (a) extraction voltages (fixed flow rate of 420 pL s^{-1}) and (b) flow rates (constant voltage of 1.6 kV). Mass flux as a function of half angle for varying (c) extraction voltages (fixed flow rate of 420 pL s^{-1}) and (d) flow rates (constant voltage of 1.6 kV). All profiles shown with super-Gaussian fits.[23]

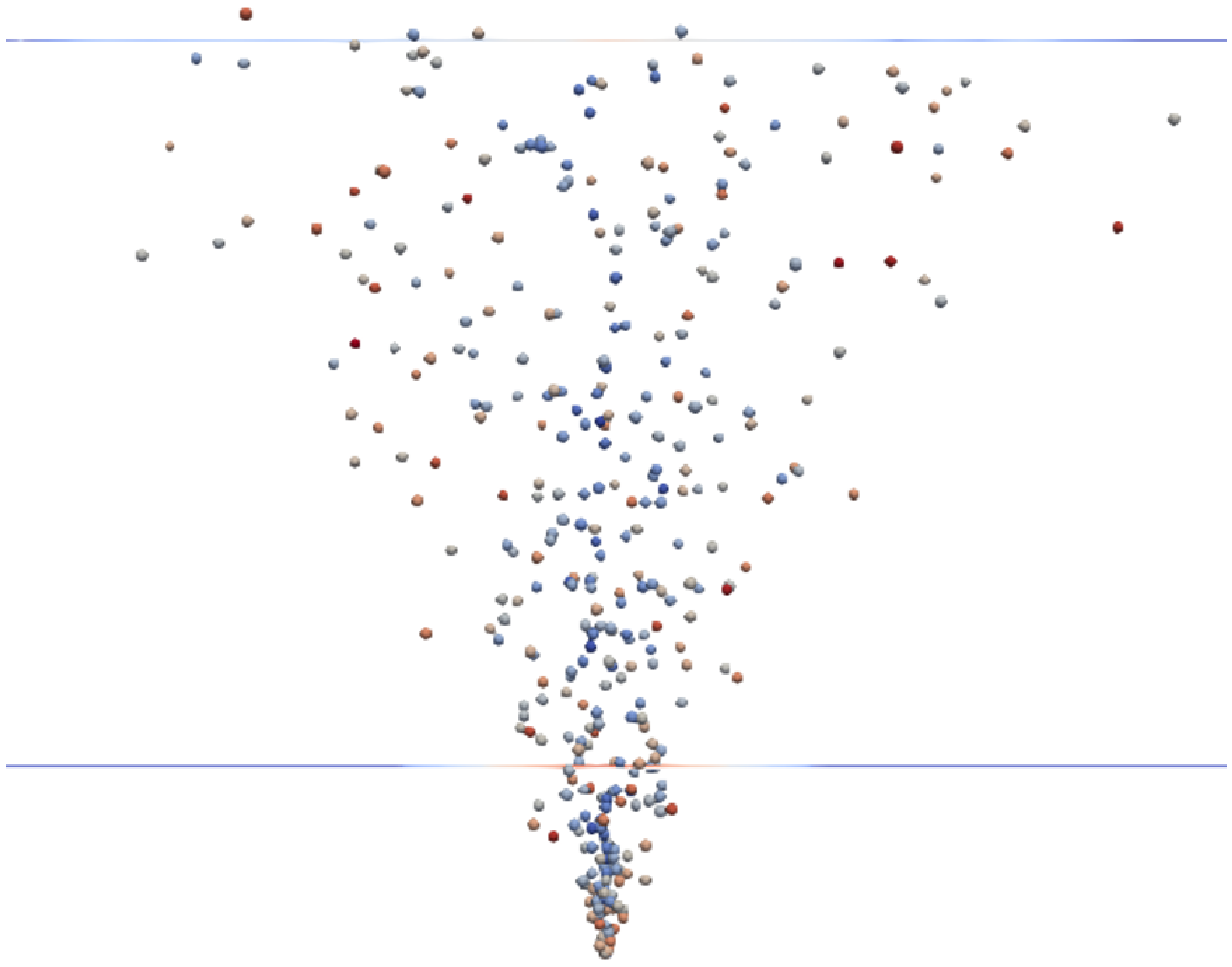
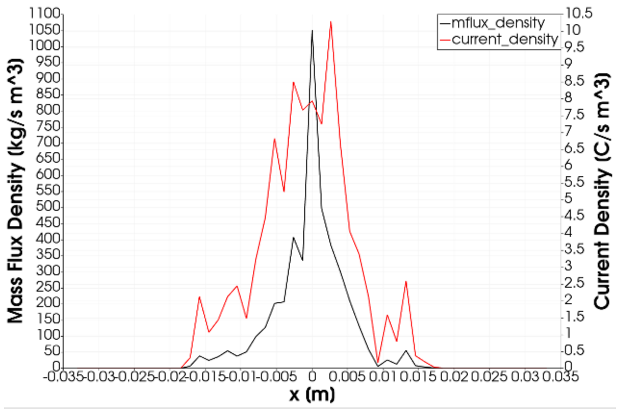
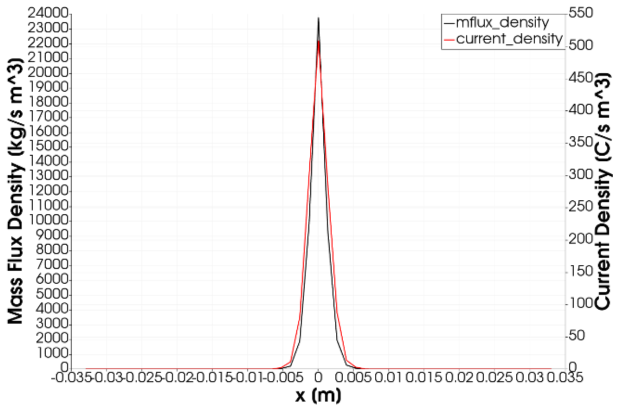


Figure D.4: An *x*-*z* snapshot of a specific charge inhomogeneous plume simulated in the DELI Model, with particles colored according to specific charge.



(a)



(b)

Figure D.5: Mass flux and current density profiles of an specific charge inhomogeneous electrospray plume a) near emission b) further downstream.

Appendix E

Analytical Efforts

During this dissertation research, analytical effort was put forth towards 1) a nondimensional representation of the governing equation for particle propagation in the DELI Model (Eq. 5.19, without drag or image charges) and 2) an expression for the 'traffic jam' described in Sec. 4.3. While these efforts were not included in published or presented efforts, they enriched our understanding of electrospray plume evolution, and contributed to the formation of the Coulomb Plume Divergence Theory presented in Sec. 4.3.

E.1 Nondimensionalize Governing Equation

The governing equation for particle propagation in the DELI Model is:

$$m\mathbf{a} = \frac{1}{4\pi\epsilon_0}\sum_{i}^{n}\frac{qq_i\mathbf{r}}{|\mathbf{r_i}^3}| + q\mathbf{E} \tag{E.1}$$

To non-dimenzionalize this equation, we need to identify characteristic parameters. The choices of variable to non-dimensionalize m and q (and q_i) are simple: mean particle mass $\overline{m}$ and mean particle charge $\overline{q}$, respectively. The choice to non-dimensionalize $\mathbf{a}$ is less clear. Unlike mass and charge, particle acceleration changes as the particle moves. We choose to focus our analysis directly following emission, where plume divergence and evolution begins. Therefore, we choice the mean acceleration directly following emission, $\overline{\mathbf{a_{emit}}}$. Similarly, we nondimensionalize $\mathbf{E}$ with the electric field magnitude at the jet tip, $\overline{\mathbf{E_{emit}}}$, and $\mathbf{r}$ (and $\mathbf{r_i}$) with the mean distance vector a particle travels in the first time step after emission from the jet tip, defined as

$$\overline{\mathbf{r_{emit}}} = T\overline{\mathbf{v_0}}, \tag{E.2}$$

where $\overline{\mathbf{v_0}}$ is the mean emission velocity and mean droplet emission period is

$$T = \frac{\overline{m}}{\dot{m}}, \tag{E.3}$$

where $\dot{m}$ is mass flowrate. Using * to represent a quantity which has been nondimensionalized via division by these characteristic parameters, the non-dimensionalized governing equation for particle propagation in the DELI Model is:

$$\overline{m\mathbf{a_{emit}}} m^* \mathbf{a}^* = \frac{1}{4\pi\epsilon_0} \frac{\overline{q}^2}{\overline{\overline{r_{emit}^2}}} \sum_i^n \frac{q^* q_i^* \mathbf{r}^*}{|\mathbf{r_i^*}^3|} + \overline{q}\overline{E_{emit}} q^* \mathbf{E}^* \tag{E.4}$$

We can define mean force magnitude on a particle following emission:

$$\overline{\mathbf{F_{emit}}} = \overline{m\mathbf{a_{emit}}}, \tag{E.5}$$

the Coulomb force magnitude between two mean-charge particles separated by one emission timestep:

$$\overline{\mathbf{F_{C_{emit}}}} = \frac{1}{4\pi\epsilon_0} \frac{\overline{q}^2}{\overline{\overline{r_{emit}^2}}}, \tag{E.6}$$

and the mean electrostatic applied force at the jet tip where particles are emitted:

$$\overline{\mathbf{F_{E_{emit}}}} = \overline{q}\overline{\mathbf{E_{emit}}} \tag{E.7}$$

such that the non-dimensionalized governing equation can be written as

$$\overline{\mathbf{F_{emit}}} m^* \mathbf{a}^* = \overline{\mathbf{F_{C_{emit}}}} \sum_i^n \frac{q^* q_i^* \mathbf{r}^*}{|\mathbf{r_i^*}^3|} + \overline{\mathbf{F_{E_{emit}}}} q^* \mathbf{E}^*. \tag{E.8}$$

Therefore, we are able to see that the force which is exerted on a particle following emission is a determined by 2 scaling factors: the force of a mean Coulomb interaction $\overline{\mathbf{F_{C_{emit}}}}$, and the mean electrostatic force $\overline{\mathbf{F_{E_{emit}}}}$. The first scaling factor is directly related to mean charge and mass flowrate, and inversely related to mean charge. The second scaling factor is directly related to mean charge and to mean electric field strength, which can be tuned by changing the electric potential, or through geometric modifications as discussed in Ch. 3.

E.2 Traffic Jam Expression

The traffic jam nondimensionalization effort was approached from a two particle perspective with the understanding that local clustering events between individual particles spread throughout the plume, as discussed in Ch. 4.

We understand a 'traffic jam' to occur when neighboring particles have different velocities near the same position. We define this 'bottleneck' position x_b to occur some distance downstream emission which can be defined as some constant $A \in \mathbb{R}$ multiplied by the characteristic length parameter $\overline{r_{emit}} = |\overline{\mathbf{r_{emit}}}|$ defined in Sec. E.1,

$$x_b = A\overline{r_{emit}} = \overline{T}|\overline{\mathbf{v_0}}|. \tag{E.9}$$

We define the first droplet to reach this position after some some t_1 which can be expressed as some constant $B \in \mathbb{R}$ multiplied by the characterise time parameter $\overline{T}$

$$t_1 = B\overline{T}. \tag{E.10}$$

The second droplet is emitted later by some finite emission period determined by scaling particle mass (or charge) against mass flowrate (or current) as described in Sec. 2.3.4:

$$T_2 = \frac{m_2}{\dot{m}}. \tag{E.11}$$

In order for the second particle to reach the bottleneck position at the same time as the first particle the second droplet must take time after emission

$$t_2 = t_1 - T_2 = B\overline{T} - \frac{m_2}{\dot{m}}. \tag{E.12}$$

Recall that traffic jams have been found to occur in Sec. 4.3 when there is an upstream velocity gradient, where upstream particles are moving faster that those downstream. In this two particle case, that would mean that the second particle has a faster velocity, v_2, than that of the first particle, v_1. The velocity of the particles at the bottleneck position depends on their emission velocity, v_0, and acceleration a they have experienced over the time t approaching that position:

$$v = v_0 + \int_0^t a(t)dt. \tag{E.13}$$

It is difficult to move forward in the nondimensional analysis with temporal integrals, so we average the acceleration experienced by the particle approaching the bottleneck position, $\overline{a}$ such that the velocity of the particle at that position can be written as

$$v = v_0 + \overline{a}t. \tag{E.14}$$

Therefore, the velocities of the two particles at the bottleneck position are

$$v_1 = v_{0_1} + \overline{a_1}B\overline{T} \tag{E.15}$$

and

$$v_2 = v_{0_2} + \overline{a_2}(B\overline{T} - T_2). \tag{E.16}$$

Thus the second particle has difference in velocity from the first particle of

$$\Delta v = v_2 - v_1 = \Bigg(v_{0_2} + \overline{a_2}(B\overline{T} - T_2)\Bigg) - \Bigg(v_{0_1} + \overline{a_1}B\overline{T}\Bigg). \tag{E.17}$$

Defining difference in the initial velocity of the second particle from the first particle of

$$\Delta v_0 = \Delta v_{0_2} - \Delta v_{0_1}, \tag{E.18}$$

and the difference in the mean acceleration experienced by the particles from emission to the bottleneck position of

$$\Delta\overline{a} = \overline{a_2} - \overline{a_1}, \tag{E.19}$$

we can re-write the difference in velocity equation as

$$\Delta v = \Delta v_0 + B\overline{T}(\Delta\overline{a}) - \overline{a_2}T_2. \tag{E.20}$$

When this difference in velocity is negative, we have an upstream velocity gradient which can cause particles to cluster and have a magnified Coulomb interaction.

To establish a frequency term with Hz units, we define a traffic jam based on the velocity difference Δv which particles achieve traveling the distance x_b:

$$TJ = \frac{\Delta v}{x_b} \tag{E.21}$$

Substituting Eq. E.9 and E.20, this equation takes the form

$$TJ = \frac{1}{A\overline{\overline{T}}\overline{v_0}}\left(\Delta v_0 + B\overline{\overline{T}}(\Delta\overline{a}) - \overline{a_2}T_2\right). \tag{E.22}$$

With some restructuring, E.11,

$$TJ = \frac{1}{A\overline{\overline{T}}}\left(\frac{\Delta v_0}{\overline{v_0}}\right) + \frac{B}{A}\left(\frac{\Delta\overline{a}}{\overline{v_0}}\right) - \frac{1}{A\overline{\overline{T}}}\left(\frac{\overline{a_2}T_2}{\overline{v_0}}\right). \tag{E.23}$$

Defining

$$\overline{v_2} = \overline{a_2}T_2 \tag{E.24}$$

which represents the additional speed particle two would have gained had it been born at the same time as particle one, our traffic jam equation becomes

$$TJ = \frac{1}{A\overline{\overline{T}}}\left(\frac{\Delta v_0}{\overline{v_0}}\right) + \frac{B}{A}\left(\frac{\Delta\overline{a}}{\overline{v_0}}\right) - \frac{1}{A\overline{\overline{T}}}\left(\frac{\overline{v_2}}{\overline{v_0}}\right). \tag{E.25}$$

Defining coefficients $C_1 = 1/A$ and $C_2 = B/A$,

$$TJ = \frac{C_1}{\overline{\overline{T}}}\left(\frac{\Delta v_0 - \overline{v_2}}{\overline{v_0}}\right) + C_2\left(\frac{\Delta\overline{a}}{\overline{v_0}}\right). \tag{E.26}$$

Recall that a traffic jam causing Coulomb plume divergence must have a upstream velocity gradient, such that Δv defined in Eq. E.20 is positive. In this case we have a positive traffic jam frequency, $TJ > 0$, which means

$$\frac{C_1}{\overline{\overline{T}}}\left(\frac{\Delta v_0 - \overline{v_2}}{\overline{v_0}}\right) > -C_2\left(\frac{\Delta\overline{a}}{\overline{v_0}}\right). \tag{E.27}$$

Note that $C_2/C_1 = B$, such that this inequality can be simplified to

$$\Delta v_0 - \overline{v_2} > -B\Delta\overline{a}\overline{\overline{T}}. \tag{E.28}$$

Defining the average velocity a particle has at the bottleneck position to be

$$\overline{v} = B\overline{\Delta a T}, \tag{E.29}$$

our inequality becomes

$$\Delta v_0 - \overline{v_2} > -\overline{v}, \tag{E.30}$$

or

$$\overline{v_2} < \Delta v_0 + \overline{v}. \tag{E.31}$$

In order to cause a traffic jam, the velocity which particle two would gain if emitted at the same time as particle one must be greater than the sum of the difference in particle initial velocity and the average velocity of a particle reaching the bottleneck position. Expanding this equation using the definitions for each term,

$$\overline{a_2}T_2 < v_{0_2} - v_{0_1} + B\overline{\Delta a T}, \tag{E.32}$$

from which substituting t_1 gives

$$\overline{a_2}T_2 < v_{0_2} - v_{0_1} + t_1\overline{\Delta a}, \tag{E.33}$$

and expanding $\overline{\Delta a}$,

$$\overline{a_2}T_2 < v_{0_2} - v_{0_1} + t_1(\overline{a_2} - \overline{a_1}). \tag{E.34}$$

Rearranging,

$$\overline{a_2}(T_2 - t_1) - v_{0_2} < -t_1\overline{a_1} - v_{0_1}, \tag{E.35}$$

and multiplying through by -1,

$$\overline{a_2}(t_1 - T_2) + v_{0_2} > t_1\overline{a_1} + v_{0_1}, \tag{E.36}$$

or

$$v_2 > v_1, \tag{E.37}$$

Therefore, we see that our traffic jam frequency term successfully describes the situation in which a second upstream particle reaches the same bottleneck location with greater speed as a downstream first particle.

Appendix F

DELI Model Code

The DELI Model code in C++ is presented below. The code is comprised of a Main.cpp file which is executed to run the simulation, World.h and World.cpp files which contain the independent variables related to particle property and simulation domain, PotentialSolver.h and PotentialSolver.cpp files which can apply a homogeneous mesh to solve simple electric field when it is not necessary to import more complex electric fields from COMSOL, Source.h and Source.cpp files related to particle emission, Droplet.h and Droplet.cpp files related to particle propagation, Output.h and Output.cpp files related to outputting data for processing and visualization, and Fields.h which contains 3D vector definitions. As noted in the comments at the top of Main.cpp, this C++ version of the code was built using particle tracking code from Ludos Brieda's textbook[171] and adapted to be specific to electrospray simulation. A prototype version of the DELI Model exists in Python, which was developed during the first two years of this dissertation before computational runtimes of large electrospray plume simulations necessitated a move to C++. This version of the code is not included herein, but can be made available upon request to future researchers.

```cpp
/* Based on Ch.4 of Brieda, "Plasma Simulations by Example"
 *
 * Demo particle-particle code
 *
 * World grid used to solve potential and obtain vacuum field
 * Part_grid is a coarser grid used to sort particles
 * Particles in cells more than some threshold away act as a single point charge
 * otherwise, Coulomb force is used directly
 *
 * Only the Coulomb force is included, see Droplets::advance
 *
 * Droplets are injected in DropletSource::sample using some arbitrary, likely
 * very non-physical model. Here you can change size range, surface charge density,
 * and wheter to make negative ions
 *
 * Code uses custom "vec3" data objects that are defined in Field.h (see the book
 * for more info on templates/operator overloading if not familiar)
 *
 * To compile and run (assuming gcc on Linux, on Windows use Visual Studio or Eclipse):
 * $ g++ -O2 -I DELI/ DELI/*.cpp -std=c++11 -o il-pic
 * $ mkdir results
 * $ ./il-pic
 *
 * To visualize electrodes, you can use Paraview "threshold filter" to threshold by
 * object id, uncheck "All scalars"
 *
 * Open parts_il_* group, add "glyphs", change to spheres
 */
#include <math.h>
#include <iostream>
#include <iomanip>
#include <vector>
#include <chrono>
#include <memory>
#include <stdio.h>
#include "World.h"
#include "PotentialSolver.h"
#include "Droplets.h"
#include "Output.h"
#include "Source.h"

using namespace std;        //to avoid having to write std::cout
using namespace Const;      //to avoid having to write Const::ME
```

```cpp

/*program execution starts here*/
int main(int argc, char *args[])
{
    /*Time Keeping Variables*/
    double make_world_time; double add_objects_time; double solve_potential_time; double solve_electric_field; double create_species_time;
    double emit_time; double sort_cells_time; double advance_time; double compute_densities_time; double compute_cumulatives_time; double output_time;

    /*initialize domain*/
    World world(Const::x_n,Const::y_n,Const::z_n);
    world.setExtents({ -Const::electrode_h, -Const::electrode_h, 0.0}, {Const::electrode_h, Const::electrode_h, Const::electrode_h * 1.01});
    std::cout << "xn: " << Const::x_n << std::endl;
    std::cout << "yn: " << Const::y_n << std::endl;
    std::cout << "zn: " << Const::z_n << std::endl;
    std::cout << "x: " << (2 * Const::electrode_h) / Const::x_n << std::endl;
    std::cout << "y: " << (2 * Const::electrode_h) / Const::y_n << std::endl;
    std::cout << "height z: " << (1.01*Const::electrode_h) / Const::z_n <<  std::endl;
    std::cout << "area x*y: " << ((2 * Const::electrode_h) / Const::x_n )* ((2 * Const::electrode_h) / Const::y_n )<< std::endl;
    //world.setExtents({ 0.0,0.0,0.0 }, { Const::electrode_h*2,Const::electrode_h*2,Const::electrode_h*1.1});
    std::cout << "EmHeight: " << Const::EmHeight << " , Jet Length: " << Const::EmHeight - Const::emitter_h - Const::TC_h << std::endl;

    /*Read Electric field from external .csv file*/
    std::vector<std::pair<std::string, std::vector<double>>> result = world.read_Efield();
    cout << "read Efield" << endl;

    int num_steps = 100000;
    double default_dt = 1e-6;
    world.setTime(default_dt, num_steps);   // time step and number of steps

    make_world_time = world.getWallTime();

    /*set objects*/
    //world.addObject(new RingZ({0,0,Const::Em_to_grid},Const::grid_thick,Const::grid_rad),0.0); //
```

```
        Grounded extractor plate w/ orifice
    world.addObject(new PlateZ({ 0,0,Const::electrode_h },
      Const::grid_thick), 0.0); //Grounded extractor plate (no orifice)
    //world.addObject(new ConeZ
      ({ 0,0,Const::emitter_h },Const::TC_h,Const::emitter_rad),
      Const::voltage); //Taylor Cone (high voltage)
    world.addObject(new CylinderZ
      ({0,0,0},Const::emitter_h,Const::emitter_rad), Const::voltage); //
      Emitter (high voltage)

    //TWO PLATE SET-UP//
    //world.addObject(new PlateZ({0,0,0.03}, 1e-3), 0.0); //Grounded plate
       (no orifice)
    //world.addObject(new PlateZ({0,0,0}, 1e-6), 1000.0); //High Voltage
      plate (no orifice)

    add_objects_time = world.getWallTime();

    /*initialize potential solver and solve initial potential*/
   // PotentialSolver solver(world, SolverType::PCG, 10000, 1e-6);
    //solver.setReferenceValues(0, 0, 0);  // n0=0 (3rd entry) gives us
      linear solver
    //std::cout << "Solving potential" << std::endl;
    //solver.solve();
    //Output::PSOutput(world);

    solve_potential_time = world.getWallTime();

    /*obtain initial electric field*/
    //solver.computeEF();
    //std::cout << "Potential solution took " << world.getWallTime() << "
      seconds" << std::endl;

    solve_electric_field = world.getWallTime();

    // create a second grid for sorting particles
    int3 part_grid_dims(100,100,100);

    /*set up particle species*/
    Droplets droplets("il", world, part_grid_dims);

    /*setup injection sources*/
    DropletSource source(droplets,world,1);     //source
    int emit_count = 0;
    //Two particle simulation
    //source.sample(world.time, {1e-1, 0, 0.01 }, { 0.0,0.0,10.0 }, 1); //
      back droplet
    //source.sample(world.time, {1e-6, 0, 0.01}, { 0.0,0.0,1.0 }, 1); //
      back droplet
```

```cpp
    //emit_count += 1;
    //source.sample(world.time, {0, 0, 0.01 + 3e-6}, {0.0,0.0,1.0}, 0.5); //forward droplet
    //emit_count += 1;

    create_species_time = world.getWallTime();
    double3 last_emitted = { 0.0, 0.0, 0.0 }; //time, mass, charge

    std::cout << "Don't forget to create a 'results/' folder if does not exist" << std::endl;

    //Track recent history of number of emitted droplets
    vector <int> n_hist(500);

    /* main loop*/
    while (world.advanceTime())
    {
        size_t n = droplets.getNp();
        emit_time = 0.0;



        if (n == 0) {
            source.sample(world.time, n); //initial droplet
            n = droplets.getNp();
            last_emitted = { world.time, droplets.particles[n - 1].mass, droplets.particles[n - 1].charge }; //time, mass, charge
            //std::cout << "last emitted: " << last_emitted << std::endl;
            emit_count += 1;
        }

        else {
            //int parity = emit_count % 2;
            //if (parity != 0) {
                if (world.time - last_emitted[0] >= last_emitted[1] / Const::mflowrate) {
                    source.sample(world.time, n);
                    n = droplets.getNp();
                    //std::cout << " mass emitted: " << emit_count << std::endl;
                    emit_count += 1;
                    emit_time = world.getWallTime();
                    last_emitted = { world.time, droplets.particles[n - 1].mass, droplets.particles[n - 1].charge }; //time, mass, charge
                }
            //}
            //else {
            //    if (world.time - last_emitted[0] >= last_emitted[2] /
                    Const::current) {
            //        source.sample(world.time, n);
            //        n = droplets.getNp();
                    //std::cout << " charge emitted: " << emit_count << 
                  std::endl;
             //       emit_count += 1;
             //       emit_time = world.getWallTime();
              //      last_emitted = { world.time, droplets.particles[n - 
                1].mass, droplets.particles[n - 1].charge }; //time, mass, 
                 charge
              //  }
            //}
        }

        //Update n_hist array
        for (int i = 0; i < n_hist.size()-1; i++) {
            n_hist[i] = n_hist[i + 1];
        }
        n_hist[n_hist.size()-1] = n;

        //print history
       /* for (int i = 0; i < 9; i++) {
            std::cout << "i: " << i << std::endl;
            std::cout << "n_hist[i]: " << n_hist[i] << std::endl;
        }*/

        /*sort particles to cells for use in force calculation*/
        droplets.sortToCells();
        sort_cells_time = world.getWallTime();

        // advance velocity and position
        droplets.advance(result);
        advance_time = world.getWallTime();

        /*Droplet population information sampling and passing to grid*/
        droplets.computeInstDensities();
        compute_densities_time = world.getWallTime();

        //If steady-state, update averages
        if (world.getTs() > 500) {
            //std::cout << "steady? " << world.steady_state << std::endl;
            if (world.steady_state == false) {
                double up_count = 0.0; //n_hist increrasing
                double down_count = 0.0; //n_hist decreasing
                for (int i = 1; i < n_hist.size() - 1; i++) {
                    if (n_hist[i] > n_hist[i-1]) {
                        up_count += 1;
                    }
                    else if (n_hist[i] < n_hist[i - 1]) {
                        down_count += 1;
                    }
                }
                //std::cout << "up count: " << up_count << std::endl;
                //std::cout << "down count: " << down_count << std::endl;
                double d_u = down_count/up_count;
                //std::cout << "ratio: " << d_u << std::endl;
                if (d_u >= 0.8) {
                    world.steady_state = true;
                }
            }

            else if (world.steady_state == true) {
                world.t_steady += 1.0;
                droplets.computeAveDensities();
            }
        }


        //droplets.computeCumulativeContours();
        compute_cumulatives_time = world.getWallTime();

        output_time = 0.0;
        Output::nOutput(droplets.getNp(), world.getTs(), world.time);
        if (world.getTs() % 1000 == 0) {
            /*screen output*/
            Output::screenOutput(world, droplets);
            Output::diagOutput(world, droplets);

            /*file output*/
            droplets.sampleMoments();
            droplets.computeSpaceCharge();
            Output::particles(world, droplets);
            Output::fields(world, droplets);

            /*flowrate & emission counter screen output*/
            droplets.computeFlow();
            std::cout << "emitted mass flowrate: " << world.mflow << 
              std::endl;
            std::cout << "emitted charge flowrate: " << world.qflow << 
              std::endl;
            std::cout << "total emitted: " << emit_count << std::endl; // 
              includes droplets no longer in domain
            cout << "collisions: " << world.collision_count << endl;
            //cout<<"TJs: " << world.TJ_count <<endl;
            output_time = world.getWallTime();
        }
        Output::runtimesOutput(make_world_time, add_objects_time, 
          solve_potential_time, solve_electric_field, create_species_time, 
           emit_time, sort_cells_time, advance_time,
          compute_densities_time, compute_cumulatives_time, output_time);
        make_world_time = 0.0;
        add_objects_time = 0.0;
        solve_potential_time = 0.0;
        solve_electric_field = 0.0;
        create_species_time = 0.0;
        output_time = 0.0;
    }
    /*screen output*/
    Output::screenOutput(world, droplets);
    Output::diagOutput(world, droplets);

    /*file output*/
    Output::particles(world, droplets);
    Output::fields(world, droplets);
    Output::exitparticlesOutput(droplets);

    /* grab starting time*/
    std::cout<<"Simulation took "<<world.getWallTime()<<" seconds"<<
      std::endl;
    system("pause");
    return 0;       //indicate normal exit
}

```

```
#ifndef _SOLVER_H
#define _SOLVER_H

#include <assert.h>
#include "World.h"

//structure to hold data for a single row
template <int S>
struct Row {
    Row() {for (int i=0;i<S;i++) {a[i]=0;col[i]=-1;}}
    void operator= (const Row &o) {for (int i=0;i<S;i++) {a[i] = o.a
      [i];col[i]=o.col[i];}}
    double a[S];        //coefficients
    int col[S];
};

/*matrix with up to seven non zero diagonals*/
class Matrix
{
public:
    Matrix(int nr):nu{nr} {rows=new Row<nvals>[nr];}
    Matrix(const Matrix &o):Matrix(o.nu) {
        for (int r=0;r<nu;r++) rows[r] = o.rows[r];
    };  //copy constructor
    ~Matrix() {if (rows) delete[] rows;}
    dvector operator*(dvector &v);  //matrix-vector multiplication

    double& operator() (int r, int c); //reference to A[r,c] value in a
      full matrix
    void clearRow(int r) {rows[r]=Row<nvals>();} //reinitializes a row
    Matrix diagSubtract(dvector &P);    //subtracts a vector from the
      diagonal
    Matrix invDiagonal();       //returns a matrix containing inverse of
      our diagonal
    double multRow(int r, dvector &x);  //multiplies row r with vector x

    static constexpr int nvals = 7; //maximum 7 non-zero values
    const int nu;           //number of rows (unknowns)

protected:
    Row<nvals> *rows;   //row data
};

enum SolverType {GS, PCG, QN};

class PotentialSolver
{
public:
    /*constructor*/
    PotentialSolver(World &world, SolverType type, int max_it, double
      tol):
        world(world), solver_type(type), A(world.ni*world.nj*world.nk),
        max_solver_it(max_it), tolerance(tol) {
        //std::cout << "In Solver constructor, n0: " << n0 << std::endl;
            buildMatrix();
        }

    /*sets reference values*/
    void setReferenceValues(double phi0, double Te0, double n0) {
        this->phi0 = phi0;
        this->Te0 = Te0;
        this->n0 = n0;
        //std::cout << "Set n0: " << n0 << std::endl;
    }

    double getTol() const { return tolerance; }

    /*computes electric field = -gradient(phi)*/
    void computeEF();

    /*builds the "A" matrix for linear potential solver*/
    void buildMatrix();

    //calls the appropriate potential solver
    bool solve()
    {
        switch(solver_type)
        {
            std::cout << "solver type: " << solver_type << std::endl;
            case GS: return solveGS();
            case PCG: return solveNRPCG();
            case QN: return solveQN();
            default: return false;
        }
    }
protected:
    World &world;
    SolverType solver_type;
    Matrix A;               //system matrix for the linear equation

    enum NodeType {REG,NEUMANN,DIRICHLET};
    std::vector<NodeType> node_type;    //flag for different node types

    unsigned max_solver_it; //maximum number of solver iterations
    double tolerance;       //solver tolerance
    double phi0 = 0;        //reference plasma potential
    //double n0 = 1e12;     //reference electron density
    double n0 = 0;      //reference electron density
    //double Te0 = 1.5;     //reference electron temperature in eV
    double Te0 = 0;     //reference electron temperature in eV


    /*computes potential in vacuum (all phi0 besides objects)*/
    bool solveV();

    /*computes potential using quasineutral boltzmann model*/
    bool solveQN();

    /*solves non-linear potential using Gauss-Seidel*/
    bool solveGS();

    /*linear PCG solver for Ax=b system*/
    bool solvePCGLinear(Matrix &A, dvector &x, dvector &b);

    /*linear GS solver for Ax=b system*/
    bool solveGSLinear(Matrix &A, dvector &x, dvector &b);

    /*Newton Raphson solver for a nonlinear system, uses PCG for the
      linear solve*/
    bool solveNRPCG();
};
#endif

```

```cpp
/*defines the simulation domain*/
#include <random>
#include <math.h>
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <fstream>
#include "World.h"
#include "Field.h"

//make an instance of the Rnd class
Rnd rnd;

using namespace std;

/*constructor*/
World::World(int ni, int nj, int nk):
    ni{ni}, nj{nj}, nk{nk}, nn{ni,nj,nk},
    phi(nn), rho(nn),
    node_vol(nn), dh3(nn),
    ef(nn), Pos(nn),
    Time(nn), object_id(nn) {
        time_start =  chrono::high_resolution_clock::now(); //save
          starting time point
    }

/*sets domain bounding box and computes mesh spacing*/
void World::setExtents(double3 _x0, double3 _xm) {
    /*set origin and the opposite corner*/
    x0 = _x0;
    xm = _xm;

    /*compute spacing by dividing length by the number of cells*/
    for (int i=0;i<3;i++)
        dh[i] = (xm[i]-x0[i])/(nn[i]-1);

    //compute centroid
    xc = 0.5*(x0+xm);

    /*recompute node volumes*/
    computeNodeVolumes();

    for (int i = 0; i < ni; i++) {
        for (int j = 0; j < nj; j++) {
            for (int k = 0; k < nk; k++) {
                double3 dh = getDh();
                dh3[i][j][k] = dh[2];
            }
        }
    }
}

/*returns elapsed wall time in seconds*/
double World::getWallTime() {
  auto time_now = chrono::high_resolution_clock::now();
  chrono::duration<double> time_delta = time_now-time_start;
  return time_delta.count();
}

/*computes node volumes, dx*dy*dz on internal nodes and fractional
 * values on domain boundary faces*/
void World::computeNodeVolumes() {
    for (int i=0;i<ni;i++)
        for (int j=0;j<nj;j++)
            for (int k=0;k<nk;k++)
            {
                double V = dh[0]*dh[1]*dh[2];   //default volume
                if (i==0 || i==ni-1) V*=0.5;    //reduce by two for each 
                  boundary index
                if (j==0 || j==nj-1) V*=0.5;
                if (k==0 || k==nk-1) V*=0.5;
                node_vol[i][j][k] = V;
            }
}


/*ads a new object and fixes dirichlet node*/
void World::addObject(Object *object, double phi_obj)
{
    objects.emplace_back(object);
    for (int i=0;i<ni;i++)
        for (int j=0;j<nj;j++)
            for (int k=0;k<nk;k++)
            {
                /*compute node position*/
                double3 x = pos(i,j,k);
                if (object->isInside(x))
                {
                    object_id[i][j][k] = 1;
                    phi[i][j][k] = phi_obj;
                }
            }
}

vector<pair<string, vector<double>>> World::read_Efield()
{
    std::ifstream fin; //go to cwd
    fin.open("Em_Ex_2DAxi.csv", ios::in);//open .csv file

    vector<pair<string, vector<double>>> result; //multi-dim structure to 
      store input: <column name, data>
    string line, colname; //helper strings

    for (int row = 0; row < 8; row++) {
        getline(fin, line); //skip first 9 lines with COMSOL model details
    }

    if (fin.good()) { //Read the column names
        getline(fin, line); //reads first line with column labels

        //Manual entry column names
        result.push_back({ "r", vector<double>{} });
        result.push_back({ "z", vector<double>{} });
        result.push_back({ "Er", vector<double>{} });
        result.push_back({ "Ez", vector<double>{} });

        //Read column names from .csv
        //std::stringstream ss(line); //make stream from line string
        //while (std::getline(ss, colname, ',')) { //break input up at 
          commas
        //  result.push_back({ colname, vector<double>{} }); //add column 
          names to results
        //  cout << "colname:" << colname << endl;
        //}
    }

    double val; //Double reference
    int ColId = 0; //initialize
    int RowId = 0; //initialize
    while (getline(fin, line)) { //read remaining lines one at a time
        std::stringstream ss(line); //make stream from line string

        ColId = 0; //re-initiate for each new line
        while (ss >> val) { //reads only doubles
            result.at(ColId).second.push_back(val); //adds data to 
              appropraite column name section of results
            if (ss.peek() == ',') ss.ignore(); //Ignore commas
            ColId++; //update column tracker
        }
        RowId++;//update row tracker
    }
    fin.close();//close .csv file
    //cout << "Col: " << ColId << endl;
    //cout << "Row: " << RowId << endl;

    //vector<double> r; vector<double> z; vector<double> Er; 
```

```
        vector<double> Ez;
    //for (int j = 0; j < RowId; j++) {
    //  r.push_back(result.at(0).second[j]);
    //  z.push_back(result.at(1).second[j]);
    //  Er.push_back(result.at(2).second[j]);
    //  Ez.push_back(result.at(3).second[j]);
    //}
    //for (int j = 0; j < RowId; j++) {
    //  cout << "point " << j << ": (";
    //  cout << r[j] << ", " << z[j] << ")" << endl;
    //  }
    //}
    //cout << "z:" << endl;
    //for (int j = 0; j < RowId; j++) {
    //  cout << z[j] << endl;
    //}
    //cout << "Er:" << endl;
    //for (int j = 0; j < RowId; j++) {
    //  cout << Er[j] << endl;
    //}
    //cout << "Ez:" << endl;
    //for (int j = 0; j < RowId; j++) {
    //  cout << Ez[j] << endl;
    //}

    pair<pair<int,int>, vector<pair<string, vector<double>>>> Efield;
    Efield.first.first = RowId;
    Efield.first.second = ColId;
    Efield.second = result;


    return result;
}
```

```
#ifndef _SOLVER_H
#define _SOLVER_H

#include <assert.h>
#include "World.h"

//structure to hold data for a single row
template <int S>
struct Row {
    Row() {for (int i=0;i<S;i++) {a[i]=0;col[i]=-1;}}
    void operator= (const Row &o) {for (int i=0;i<S;i++) {a[i] = o.a
      [i];col[i]=o.col[i];}}
    double a[S];        //coefficients
    int col[S];
};

/*matrix with up to seven non zero diagonals*/
class Matrix
{
public:
    Matrix(int nr):nu{nr} {rows=new Row<nvals>[nr];}
    Matrix(const Matrix &o):Matrix(o.nu) {
        for (int r=0;r<nu;r++) rows[r] = o.rows[r];
    };  //copy constructor
    ~Matrix() {if (rows) delete[] rows;}
    dvector operator*(dvector &v);  //matrix-vector multiplication

    double& operator() (int r, int c); //reference to A[r,c] value in a
      full matrix
    void clearRow(int r) {rows[r]=Row<nvals>();} //reinitializes a row
    Matrix diagSubtract(dvector &P);    //subtracts a vector from the
      diagonal
    Matrix invDiagonal();       //returns a matrix containing inverse of
      our diagonal
    double multRow(int r, dvector &x);  //multiplies row r with vector x

    static constexpr int nvals = 7; //maximum 7 non-zero values
    const int nu;           //number of rows (unknowns)

protected:
    Row<nvals> *rows;   //row data
};

enum SolverType {GS, PCG, QN};

class PotentialSolver
{
public:
    /*constructor*/
    PotentialSolver(World &world, SolverType type, int max_it, double
      tol):
        world(world), solver_type(type), A(world.ni*world.nj*world.nk),
        max_solver_it(max_it), tolerance(tol) {
        //std::cout << "In Solver constructor, n0: " << n0 << std::endl;
            buildMatrix();
        }

    /*sets reference values*/
    void setReferenceValues(double phi0, double Te0, double n0) {
        this->phi0 = phi0;
        this->Te0 = Te0;
        this->n0 = n0;
        //std::cout << "Set n0: " << n0 << std::endl;
    }

    double getTol() const { return tolerance; }

    /*computes electric field = -gradient(phi)*/
    void computeEF();

    /*builds the "A" matrix for linear potential solver*/
    void buildMatrix();

    //calls the appropriate potential solver
    bool solve()
    {
        switch(solver_type)
        {
            std::cout << "solver type: " << solver_type << std::endl;
            case GS: return solveGS();
            case PCG: return solveNRPCG();
            case QN: return solveQN();
            default: return false;
        }
    }
protected:
    World &world;
    SolverType solver_type;
    Matrix A;                //system matrix for the linear equation

    enum NodeType {REG,NEUMANN,DIRICHLET};
    std::vector<NodeType> node_type;    //flag for different node types

    unsigned max_solver_it; //maximum number of solver iterations
    double tolerance;       //solver tolerance
    double phi0 = 0;        //reference plasma potential
    //double n0 = 1e12;     //reference electron density
    double n0 = 0;      //reference electron density
    //double Te0 = 1.5;     //reference electron temperature in eV
    double Te0 = 0;     //reference electron temperature in eV


    /*computes potential in vacuum (all phi0 besides objects)*/
    bool solveV();

    /*computes potential using quasineutral boltzmann model*/
    bool solveQN();

    /*solves non-linear potential using Gauss-Seidel*/
    bool solveGS();

    /*linear PCG solver for Ax=b system*/
    bool solvePCGLinear(Matrix &A, dvector &x, dvector &b);

    /*linear GS solver for Ax=b system*/
    bool solveGSLinear(Matrix &A, dvector &x, dvector &b);

    /*Newton Raphson solver for a nonlinear system, uses PCG for the
      linear solve*/
    bool solveNRPCG();
};
#endif

```

```cpp
#include <math.h>
#include <iostream>
#include "World.h"
#include "PotentialSolver.h"
#include "Field.h"
#include "Output.h"

using namespace std;
using namespace Const;

/*solves poisson equation with Boltzmann electrons using the Gauss-Seidel 
  scheme*/

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

using namespace std;
using dvector = vector<double>;

//matrix-vector multiplication
dvector Matrix::operator*(dvector &v) {
    dvector r(nu);
    for (int u=0;u<nu;u++) {
        auto &row = rows[u];
        r[u] = 0;
        for (int i=0;i<nvals;i++){
            if (row.col[i]>=0) r[u]+=row.a[i]*v[row.col[i]];
            else break; //end at the first -1
        }
    }
    return r;
}

//returns reference to A[r,c] element in the full matrix
double& Matrix::operator()(int r, int c){
    //find this entry
    auto &row = rows[r]; int v;
    for (v=0;v<nvals;v++)
    {
        if (row.col[v]==c) break;   //if found
        if (row.col[v]<0) {row.col[v]=c;   //set
                           break;}
    }
    assert(v!=nvals);   //check for overflow
    return row.a[v];
}

```

```
/*returns inverse of a diagonal preconditioner*/
Matrix Matrix::invDiagonal()
{
    Matrix M(nu);
    for (int r=0;r<nu;r++)  M(r,r) = 1.0/(*this)(r,r);
   return M;
}

/*subtracts diagonal matrix diag from A*/
Matrix Matrix::diagSubtract(dvector &P) {
    Matrix M(*this);    //make a copy
    for (int u=0;u<nu;u++) M(u,u)=(*this)(u,u)-P[u];
    return M;
}

//multiplies row r with vector x
double Matrix::multRow(int r, dvector &x){
    auto &row = rows[r];
    double sum=0;
    for (int i=0;i<nvals;i++)
    {
        if (row.col[i]>=0) sum+=row.a[i]*x[row.col[i]];
        else break;
    }
    return sum;
}


dvector operator-(const dvector &a, const dvector &b) {
    size_t nu = a.size();
    dvector r(nu);
    for (size_t u=0;u<nu;u++) r[u] = a[u]-b[u];
    return r;
}

dvector operator+(const dvector &a, const dvector &b) {
    size_t nu = a.size();
    dvector r(nu);
    for (size_t u=0;u<nu;u++) r[u] = a[u]+b[u];
    return r;
}

dvector operator*(const double s, const dvector &a) {
    size_t nu = a.size();
    dvector r(nu);
    for (size_t u=0;u<nu;u++) r[u] = s*a[u];
    return r;
}

```

```
/*vector math helper functions*/
namespace vec
{
    /*returns sum of v1[i]*v2[i]*/
    double dot(dvector v1, dvector v2)
    {
        double dot = 0;
        size_t nu = v1.size();
        for (size_t j=0;j<nu;j++)
            dot+=v1[j]*v2[j];
        return dot;
    }

    /*returns l2 norm*/
    double norm(dvector v)
    {
        double sum = 0;
        int nu = v.size();
        for (int j=0;j<nu;j++)
            sum+=v[j]*v[j];
        return sqrt(sum/nu);
    }

    /** converts 3D field to a 1D vector*/
    dvector deflate(Field &f3)
    {
        dvector r(f3.ni*f3.nj*f3.nk);
        for (int i=0;i<f3.ni;i++)
              for (int j=0;j<f3.nj;j++)
                    for (int k=0;k<f3.nk;k++)
                         r[f3.U(i,j,k)] = f3[i][j][k];
        return r;
    }

    /** converts 1D vector to 3D field*/
    void inflate(dvector &d1, Field& f3)
    {
        for (int i=0;i<f3.ni;i++)
            for (int j=0;j<f3.nj;j++)
                for (int k=0;k<f3.nk;k++)
                    f3[i][j][k] = d1[f3.U(i,j,k)];
    }

};

//constructs the coefficient matrix
void PotentialSolver::buildMatrix()
{
    //std::cout << "In Build Matrix, n0: " << n0 << std::endl;
    double3 dh = world.getDh();
    double idx = 1.0/dh[0];
    double idy = 1.0/dh[1];
    double idz = 1.0/dh[2];
    double idx2 = idx*idx;  /*1/(dx*dx)*/
    double idy2 = idy*idy;
    double idz2 = idz*idz;
    int ni = world.ni;
    int nj = world.nj;
    int nk = world.nk;
    int nu = ni*nj*nk;

    /*reserve space for node types*/
    node_type.resize(nu);

    /*solve potential*/
    for (int k=0;k<nk;k++)
        for (int j=0;j<nj;j++)
            for (int i=0;i<ni;i++)
            {
                int u = world.U(i,j,k);
                A.clearRow(u);
                //dirichlet node
                if (world.object_id[i][j][k]>0)
                {
                    //std::cout << "Dirichlet: " << i << j << k << 
                   std::endl;
                    A(u,u)=1;   //set 1 on the diagonal
                    node_type[u] = DIRICHLET;
                    continue;
                }

                //Neumann boundaries
                node_type[u] = NEUMANN;     //set default
                if (i==0) {A(u,u)=idx;A(u,u+1)=-idx;}
                else if (i==ni-1) {A(u,u)=idx;A(u,u-1)=-idx;}
                else if (j==0) {A(u,u)=idy;A(u,u+ni)=-idy;}
                else if (j==nj-1) {A(u,u)=idy;A(u,u-ni)=-idy;}
                else if (k==0) {A(u,u)=idz;A(u,u+ni*nj)=-idz;}
                else if (k==nk-1) {
                    A(u,u)=idz;
                    A(u,u-ni*nj)=-idz;}
                else {
                    //standard internal stencil
                    A(u,u-ni*nj) = idz2;
                    A(u,u-ni) = idy2;
                    A(u,u-1) = idx2;
                    A(u,u) = -2.0*(idx2+idy2+idz2);
                    A(u,u+1) = idx2;
                    A(u,u+ni) = idy2;
                    A(u,u+ni*nj) = idz2;
                    node_type[u] = REG; //regular internal node
                }
            }
    //solveQN();
    solveV();
}

/*vacuum potential solver*/
bool PotentialSolver::solveV()
{
    std::cout << "In V Solver, n0: " << n0 << std::endl;
    Field& phi = world.phi;
    Field& rhoi = world.rho;
    double rho0 = n0 * QE;
    double rho_ratio_min = 1e-6;


    for (int i = 0; i < world.ni; i++)
        for (int j = 0; j < world.nj; j++)
            for (int k = 0; k < world.nk; k++)
            {
                if (world.object_id[i][j][k] > 0) continue; /*skip Dirichlet nodes*/

                phi[i][j][k] = phi0;
            }
    return true;
}

/*quasi-neutral potential solver*/
bool PotentialSolver::solveQN()
{
    std::cout << "In QN Solver, n0: " << n0 << std::endl;
    Field& phi = world.phi;
    Field& rhoi = world.rho;
    double rho0 = n0*QE;
    double rho_ratio_min = 1e-6;


    for (int i=0;i<world.ni;i++)
        for (int j=0;j<world.nj;j++)
            for (int k=0;k<world.nk;k++)
            {
                if (world.object_id[i][j][k] > 0) {
                    continue; /*skip Dirichlet nodes*/
                }

```

```cpp
                double rho_ratio = rhoi[i][j][k]/rho0;
                if (rho_ratio<rho_ratio_min) rho_ratio=rho_ratio_min;
                phi[i][j][k] = phi0 + Te0*log(rho_ratio);
            }
    return true;
}

/*Newton Raphson solver for a nonlinear system, using PCG for the linear solve   */
bool PotentialSolver::solveNRPCG()
{
    std::cout << "In PCG Nonlinear" << std::endl;

    /*main NR iteration loop*/
    const int NR_MAX_IT=20;     /*maximum number of NR iterations*/
    const double NR_TOL = 1e-3;
    int nu = A.nu;

    Matrix J(nu);
    dvector P(nu);
    dvector y(nu);
    dvector x = vec::deflate(world.phi);
    dvector b = vec::deflate(world.rho);

    /*set RHS to zero on boundary nodes (zero electric field)
      and to existing potential on fixed nodes */
    for (int u=0;u<nu;u++)
    {
        if (node_type[u]==NEUMANN) b[u] = 0;            /*neumann boundary*/
        else if (node_type[u]==DIRICHLET) b[u] = x[u];  /*dirichlet boundary*/
        else b[u] = -b[u]/EPS_0;            /*regular node*/
    }

    // use linear solver if n0 non-zero
    std::cout << "e density n0: " << n0 << std::endl;
    if (n0<=0) {
        bool converged = solvePCGLinear(A,x,b);
        //bool converged = solveGSLinear(A, x, b);
        vec::inflate(x,world.phi);
        return converged;
    }


    double norm;
    bool converged=false;
    for(int it=0;it<NR_MAX_IT;it++)
    {
        /*compute F by first subtracting the linear term */
        dvector F = A*x-b;

        /*subtract b(x) on regular nodes*/
        for (int n=0;n<nu;n++)
            if (node_type[n]==REG)  /*regular nodes*/
                F[n] -= QE*n0*exp((x[n]-phi0)/Te0)/EPS_0;

        /*Compute P, diagonal of d(bx)/dphi*/
        for (int n=0;n<nu;n++)
        {
            if (node_type[n]==REG)
                P[n] = n0*QE/(EPS_0*Te0)*exp((x[n]-phi0)/Te0);
        }

        /*Compute J = A-diag(P)*/
        Matrix J = A.diagSubtract(P);

        /*solve Jy=F*/
        if (!solvePCGLinear(J,y,F))
            solveGSLinear(J,y,F);

        /*clear any numerical noise on Dirichlet nodes*/
        for (int u=0;u<nu;u++)
            if (node_type[u]==DIRICHLET) y[u]=0;

        /*x=x-y*/
        x = x-y;

        norm=vec::norm(y);
        //cout<<"NR norm: "<<norm<<endl;

        Output::convOutput(it, NR_TOL, norm);
        if (norm<NR_TOL)
        {
            converged=true;
            break;
        }
    }

    if (!converged)
        cout<<"NR+PCG failed to converge, norm = "<<norm<<endl;

    /*convert to 3d data*/
    vec::inflate(x,world.phi);
    return converged;
}

/*PCG solver for a linear system Ax=b*/
```

```
bool PotentialSolver::solvePCGLinear(Matrix &A, dvector &x, dvector &b)
{
    std::cout << "In PCG Linear" << std::endl;
    bool converged= false;

    double l2 = 0;
    Matrix M = A.invDiagonal(); //inverse of Jacobi preconditioner

    /*initialization*/
    dvector g = A*x-b;
    dvector s = M*g;
    dvector d = -1*s;

    for (unsigned it=0;it<max_solver_it;it++)
    {
        dvector z = A*d;
        double alpha = vec::dot(g,s);
        double beta = vec::dot(d,z);

        x = x+(alpha/beta)*d;
        g = g+(alpha/beta)*z;
        s = M*g;

        beta = alpha;
        alpha = vec::dot(g,s);

        d = (alpha/beta)*d-s;
        l2 = vec::norm(g);
        Output::convOutput(it, tolerance, l2);
        if (l2<tolerance) {converged=true;break;}
    }

    if (!converged) cerr<<"PCG failed to converge, norm(g) = "<<l2<<endl;
    return converged;
}

/*solves non-linear Poisson equation using Gauss-Seidel*/
bool PotentialSolver::solveGS()
{
    std::cout << "In GS Nonlinear" << std::endl;

    //references to avoid having to write world.phi
    Field &phi = world.phi;
    Field &rho = world.rho;     //rho contains only ion contribution

    //precompute 1/(dx^2)
    double3 dh = world.getDh();
    double idx2 = 1.0/(dh[0]*dh[0]);
    double idy2 = 1.0/(dh[1]*dh[1]);
    double idz2 = 1.0/(dh[2]*dh[2]);

    double L2=0;            //norm
    bool converged= false;

    /*solve potential*/
    for (unsigned it=0;it<max_solver_it;it++)
    {
        for (int i=0;i<world.ni;i++)
            for (int j=0;j<world.nj;j++)
                for (int k=0;k<world.nk;k++)
                {
                    /*skip over solid (fixed) nodes = Dirichlet boundaries*/
                    if (world.object_id[i][j][k]>0) continue;

                    if (i==0)
                        phi[i][j][k] = phi[i+1][j][k];
                    else if (i==world.ni-1)
                        phi[i][j][k] = phi[i-1][j][k];
                    else if (j==0)
                        phi[i][j][k] = phi[i][j+1][k];
                    else if (j==world.nj-1)
                        phi[i][j][k] = phi[i][j-1][k];
                    else if (k==0)
                        phi[i][j][k] = phi[i][j][k+1];
                    else if (k==world.nk-1)
                        phi[i][j][k] = phi[i][j][k-1];
                    else {  //standard internal open node

                        //evaluate electron density from the Boltzmann relationshp
                        double ne = n0 * exp((phi[i][j][k]-phi0)/Te0);

                        double phi_new = ((rho[i][j][k]-Const::QE*ne)/Const::EPS_0 +
                                        idx2*(phi[i-1][j][k] + phi[i+1][j][k]) +
                                        idy2*(phi[i][j-1][k]+phi[i][j+1][k]) +
                                        idz2*(phi[i][j][k-1]+phi[i][j][k+1]))/(2*idx2+2*idy2+2*idz2);

                        /*SOR*/
                        phi[i][j][k] = phi[i][j][k] + 1.4*(phi_new-phi[i][j][k]);
                    }
                }

        /*check for convergence*/
        if (it%25==0)
        {
            double sum = 0;
            for (int i=0;i<world.ni;i++)
                for (int j=0;j<world.nj;j++)
                    for (int k=0;k<world.nk;k++)
                    {
                        /*skip over solid (fixed) nodes*/
                        if (world.object_id[i][j][k]>0) continue;

                        double R = 0;
                        if (i==0)
                            R = phi[i][j][k] - phi[i+1][j][k];
                        else if (i==world.ni-1)
                            R = phi[i][j][k] - phi[i-1][j][k];
                        else if (j==0)
                            R = phi[i][j][k] - phi[i][j+1][k];
                        else if (j==world.nj-1)
                            R = phi[i][j][k] - phi[i][j-1][k];
                        else if (k==0)
                            R = phi[i][j][k] - phi[i][j][k+1];
                        else if (k==world.nk-1)
                            R = phi[i][j][k] - phi[i][j][k-1];
                        else {
                                //evaluate electron density from the Boltzmann relationshp
                                double ne = n0 * exp((phi[i][j][k]-phi0)/Te0);
                                R = -phi[i][j][k]*(2*idx2+2*idy2+2*idz2) +
                                    (rho[i][j][k]-Const::QE*ne)/Const::EPS_0 +
                                    idx2*(phi[i-1][j][k] + phi[i+1][j][k]) +
                                    idy2*(phi[i][j-1][k]+phi[i][j+1][k]) +
                                    idz2*(phi[i][j][k-1]+phi[i][j][k+1]);
                        }

                        sum += R*R;
                    }

            L2 = sqrt(sum/(world.ni*world.nj*world.nk));
            Output::convOutput(it, tolerance, L2);
            if (L2<tolerance) {converged=true;break;}
        }
    }

    if (!converged) cerr<<"GS failed to converge, L2="<<L2<<endl;
    return converged;
```

```
}

/*solves non-linear Poisson equation using Gauss-Seidel*/
bool PotentialSolver::solveGSLinear(Matrix &A, dvector &x, dvector &b)
{
    std::cout << "In GS Linear" << std::endl;

    double L2=0;            //norm
    bool converged= false;

    /*solve potential*/
    for (unsigned it=0;it<max_solver_it;it++)
    {
        for (int u=0;u<A.nu;u++)
        {
            double S = A.multRow(u,x)-A(u,u)*x[u]; //multiplication of 
              non-diagonal terms
            double phi_new = (b[u]- S)/A(u,u);

            /*SOR*/
            x[u] = x[u] + 1.*(phi_new-x[u]);
        }

         /*check for convergence*/
         if (it%25==0)
         {
             dvector R = A*x-b;
             L2 = vec::norm(R);
             Output::convOutput(it, tolerance, L2);
             if (L2<tolerance) {converged=true;break;}
        }
    }

    if (!converged) cerr<<"GS failed to converge, L2="<<L2<<endl;
    return converged;
}


/*computes electric field = -gradient(phi) using 2nd order differencing*/
void PotentialSolver::computeEF()
{
    //grab references to data
    Field &phi = world.phi;
    Field3 &ef = world.ef;

    double3 dh = world.getDh();
    double dx = dh[0];
    double dy = dh[1];
    double dz = dh[2];
```

```cpp

    for (int i=0;i<world.ni;i++)
        for (int j=0;j<world.nj;j++)
            for (int k=0;k<world.nk;k++)
            {
                /*x component*/
                if (i==0)
                    ef[i][j][k][0] = -(-3*phi[i][j][k]+4*phi[i+1][j][k]-phi[i+2][j][k])/(2*dx); /*forward*/
                else if (i==world.ni-1)
                    ef[i][j][k][0] = -(phi[i-2][j][k]-4*phi[i-1][j][k]+3*phi[i][j][k])/(2*dx);    /*backward*/
                else
                    ef[i][j][k][0] = -(phi[i+1][j][k] - phi[i-1][j][k])/(2*dx); /*central*/

                /*y component*/
                if (j==0)
                    ef[i][j][k][1] = -(-3*phi[i][j][k] + 4*phi[i][j+1][k]-phi[i][j+2][k])/(2*dy);
                else if (j==world.nj-1)
                    ef[i][j][k][1] = -(phi[i][j-2][k] - 4*phi[i][j-1][k] + 3*phi[i][j][k])/(2*dy);
                else
                    ef[i][j][k][1] = -(phi[i][j+1][k] - phi[i][j-1][k])/(2*dy);

                /*z component*/
                if (k==0)
                    ef[i][j][k][2] = -(-3*phi[i][j][k] + 4*phi[i][j][k+1]-phi[i][j][k+2])/(2*dz);
                else if (k==world.nk-1)
                    ef[i][j][k][2] = -(phi[i][j][k-2] - 4*phi[i][j][k-1]+3*phi[i][j][k])/(2*dz);
                else
                    ef[i][j][k][2] = -(phi[i][j][k+1] - phi[i][j][k-1])/(2*dz);
            }
}

```

```cpp
#ifndef SOURCE_H_
#define SOURCE_H_

#include <iostream>
#include "World.h"
#include "Droplets.h"


//simple monoenergetic source
class DropletSource {
public:
    DropletSource(Droplets &droplets, World &world, int num_droplets) :
    sp{droplets}, world{world}, num_droplets{num_droplets} {}

    //generates particles
    void sample(double addtime, double n);
    //void sample(double addtime, double3 x, double3 v, double r_ratio,
      double n);


protected:
    Droplets &sp;   //reference to the injected species
    World &world;       //reference to world
    double3 x0;
    int num_droplets;


};


#endif /* SOURCE_H_ */

```

```cpp
#include <iostream>
#include "Source.h"

//samples particles with finite thermal and drift velocity
void DropletSource::sample(double addtime, double n)
//void DropletSource::sample(double addtime, double3 x, double3 v, double r_ratio, n)
{
    std::default_random_engine generator;

    // random device class instance, source of 'true' randomness for initializing random seed
    std::random_device rd;

    // Mersenne twister PRNG, initialized with seed from previous random device instance
    std::mt19937 gen(rd());

    //Gaussian distributions
    //radius
    std::normal_distribution<double> ddistribution(Const::dmean,Const::dRMS);
    //specific charge
    //std::normal_distribution<double> qmdistribution(qmmean,qmsigma);
    //charge
    //std::normal_distribution<double> qdistribution(Const::qmean, 0.1*Const::qmean);
    //radius
    //std::lognormal_distribution<> ddistribution(mean order, std. dev. order);


    for (int p=0;p<num_droplets;p++) {
        //Radius
        //double r = (1e-6)*r_ratio;
        double d = ddistribution(gen);
        double r = d/ 2;
        double m = Const::density * (4 / 3) * Const::PI * pow(r, 3);
        //double d = 2*r;

        //Charge
        double q = Const::qmean * (d/ (Const::dmean));
        //double q = qdistribution(gen);

        //Specific Charge
        //double qm = qmdistribution(gen);
        double qm = q / m;

        // pos and velocity
        double rscale = 0.1*(0.5/1.87)*Const::dmean; //0.1r_jet, r_jet = 1.87r_drop
        //double rscale = 2 * Const::dmean; //From Grifoll & Rosell-Llompart 2012
        double3 pos = sp.emitPerturbed(rscale);
        //double3 pos = x;
        double3 vel = { 0,0, Const::v0 };
        //double3 vel = v;

        //acceleration
        double3 acc = {0.0,0.0,0.0};

        //force
        double3 Cforce = {0.0,0.0,0.0};
        double3 Dforce = {0.0,0.0,0.0};
        double3 Eforce = {0.0,0.0,0.0}; //fix this
        double3 Iforce = { 0.0, 0.0 ,0.0 };

        double birthID = n; //will be updated at first motion

        //make negative ions
        //if (rnd()>=0.5) qden*=-1.0;

        sp.addParticle(birthID, pos, vel, acc, Cforce, Eforce, Dforce, Iforce, r, qm, addtime);


    }


}

```

```cpp
/*Defines flying material data*/

#ifndef _DROPLETS_H
#define _DROPLETS_H

#include <iostream>
#include <vector>
#include "Field.h"
#include "World.h"

/** Data structures for particle storage **/
struct Particle
{
    int birthID;            // ID in particles vector
    double birthmass;       // mass at birth [kg]
    double3 pos;            // position
    double3 vel;            // velocity
    double3 vel_for_drag;   // for 1st order Velocity Verlet Scheme with drag (Grifoll, AKA, & RL 2011)
    double3 acc;            // acceleration
    double3 Cforce;          // Coulombic force
    double3 Eforce;          // Electric field (from electrodes) force
    double3 Dforce;          // Drag force
    double3 Iforce;         //Image charge force
    double3 force;           // force
    double3 frat;           // frat
    double r;               // radius
    double charge;          // charge
    double mass;            // droplet mass
    double addtime;         // time born
    double rss;             // impact parameter for 90 degree self scatter
    double rsep;            // distance to nearest neighbor (surface-to-surface)
    double angle;           // plume angle

    Particle(int ID, double3 x, double3 v, double3 acc, double3 Cforce, double3 Eforce, double3 Dforce, double3 Iforce, double r, double qm, double addtime):
        birthID{ ID }, pos{ x }, vel{ v }, acc{ acc }, Cforce{ Cforce }, Eforce{ Eforce }, Dforce{ Dforce }, Iforce{ Iforce }, r{ r }, addtime{ addtime } {
        double A = 4.0*Const::PI*r*r;
        double V = A*r/3.0;

        vel_for_drag = v;
        mass = V*Const::density;
        charge = mass*qm;
        force = Cforce + Eforce + Dforce;
        frat = Cforce/Eforce;
        angle = pow((pow(pos[0], 2) + pow(pos[1], 2)), 0.5) / pos[2];
        birthmass = mass;

        rss = (1 / (4.0 * Const::PI * Const::EPS_0)) * charge * charge * 
          2 / (mass * pow(2.0, 2.0));
        rsep = 1.0;
    }
};

struct Cell {
    std::vector<Particle *> parts;
    double3 xc;     // cell center position
    double3 mass_xc; // mass centroid
    double charge;      // total charge
};

class PartGrid {
public:
    PartGrid(const int3 &dims, const double3 &x0, const double3 &xm):
        dims{dims},x0{x0}, xm{xm} {
        size_t num_cells = dims[0]*dims[1]*dims[2];
        cells.resize(num_cells);

        // set cell sizes, assuming dims is the number of cells
        for (int i=0;i<3;i++)
            dh[i] = (xm[i]-x0[i])/dims[i];

        // set centers
        size_t c = 0;
        //std::vector<Cell>::iterator it = cells.begin();

        for (int k=0;k<dims[2];k++)
            for (int j=0;j<dims[1];j++)
                for (int i=0;i<dims[0];i++) {
                    cells[c].xc = {(i+0.5)*dh[0], (j+0.5)*dh[1], (k+0.5)
                  *dh[2]};
                    //it->xc = ...
                    c++;
                }
    }

    void clear() {
        for (Cell &c:cells) {c.parts.clear();c.charge=0;}
    }

    // return cell index, or -1 if out of bounds
    int XtoC(const double3 &pos) {
        int3 l;
        // convert and check for bounds
        for (int d=0;d<3;d++) {
            l[d] = (int)((pos[d]-x0[d])/dh[d]);
            if (l[d]<0 || l[d]>dims[d]) return -1;
        }

        return l[2]*dims[0]*dims[1] + l[1]*dims[0] + l[0];
    }


    int3 dims;
    double3 x0, xm;     // bounding box extents
    std::vector<Cell> cells;

    double3 dh;
};

/*species container*/
class Droplets
{
public:
    Droplets(std::string name, World &world, int3 part_grid_dims) :
        name(name),
        inst_den(world.nn), inst_mden(world.nn), inst_qden(world.nn),
        ave_den(world.nn), ave_mden(world.nn), ave_qden(world.nn), Cf
          (world.nn),
        mass_cm(world.nn), charge_cm(world.nn),
        vel(world.nn), force(world.nn), //den_ave(world.nn),
        radfrat(world.nn),
        n_sum(world.nn),nv_sum(world.nn),
        world(world), part_grid(part_grid_dims, world.getX0(), world.getXm
          ()) {   }

    /*return a pointer to part_grid*/
    PartGrid& getPG() { return part_grid; }

    /*return a pointer to the particles address*/
    std::vector<Particle>& getparticlesadd() { return particles; }

    /*return the particles*/
    std::vector<Particle> getparticles() { return particles; }

    /*returns the number of simulation particles*/
    size_t getNp()  {return particles.size();}

    /*returns the number of real particles*/
    int getRealCount();

    /*computes simulation timestep*/
    double adaptTimeStep(double default_dt);

    /*Computer Efield using nearest-neighbor*/
    double3 compute_Efield_nn(std::vector<std::pair<std::string, 
      std::vector<double>>>, Particle& part);

    /*Computing Efield using barycentric interpolation*/
    double3 compute_Efield_bc(std::vector<std::pair<std::string, 
      std::vector<double>>>, Particle& part);

    /*Computing Efield using line-of-charge (LOC) approach from Ganan-
      Calvo et al. 1994*/
    double3 compute_Efield_LOC(Particle& part);

    /*Computing Efield using analytical approach from Ganan-Calvo et al. 
      1994*/
    double3 compute_Efield_ANLTC(Particle& part);

    /*returns the species momentum*/
    double3 getMomentum();

    /*returns the species kinetic energy*/
    double getKE();

    /*moves all particles using electric field ef[]*/
    void advance(std::vector<std::pair<std::string, std::vector<double>>> 
      result);

    /*compute instataneous number, mass, and charge densities*/
    void computeInstDensities();

    /*compute steady-state averaged number, mass, and charge densities*/
    void computeAveDensities();

    /*compute space charge*/
    void computeSpaceCharge();

    /*compute mass contour*/
    void computeCumulativeContours();

    /*samples velocity moments*/
    void sampleMoments();

    /*uses sampled data to compute velocity and temperature*/
    void computeGasProperties();

    /*clears sampled moment data*/
    void clearSamples();

    /*computes emitted mass flowrate and current*/
    void computeFlow();

    // sorts particles to cells
    void sortToCells();

    /*adds a new particle of radius r*/
    void addParticle(double birthID, double3 pos, double3 vel, double3 
      acc, double3 Cforce, double3 Eforce, double3 Dforce, double3 Iforce, 
       double r, double qm, double addtime);

    /*updates number density*/
    //void updateAverages() {den_ave.updateAverage(den);}

    /*returns random thermal velocity*/
    double sampleVth(double T);

    /*samples random isotropic velocity*/
    double3 emitPerturbed(double diameter);

    const std::string name;         /*species name*/

    std::vector<Particle> particles;    /*contiguous array for storing 
      particles*/
    std::vector<std::vector<double>> exitparticles;
    Field inst_den;         /*instantaneous number density*/
    Field inst_mden;            /*instantaneous mass density*/
    Field inst_qden;            /*instantaneous charge density*/
    Field ave_den;          /*steady-state average number density*/
    Field ave_mden;         /*steady-state average mass density*/
    Field ave_qden;         /*steady-state average charge density*/
    Field3 Cf;          //Coulombic force (space charge) field 
      components*/
    Field mass_cm;   /*cumulative mass*/
    Field charge_cm;   /*cumulative charge*/
    Field radfrat;      /*radial F_C/F_E*/
    Field3 vel;         /*stream velocity*/
    Field3 force;       /*Total force*/
    //Field den_ave;        /*averaged number density*/
    PartGrid part_grid;
    World& world;


protected:

    Field n_sum;
    Field3 nv_sum;
};


```

```
#endif

```

```cpp
/*definitions for species functions*/
#include <math.h>
#include <iostream>
#include "Droplets.h"
#include "Field.h"

/*updates velocities and positions of all particles of this species*/
void Droplets::advance(std::vector<std::pair<std::string, std::vector<double>>> results)
{

    // average particle grid cell dimension, using this to scale forces
    double pg_size = mag(part_grid.dh);
    int p_index = 0;
    // update particle velocities
    for (Particle& part : particles)
    {
        /*increment particle's dt by world dt*/
        double dt = world.getDt();

        /*reset nearest neighbor separation*/
        part.rsep = 1.0;

        /*Drag Calculations*/
        double P = 0.50*Const::P0;  /*Pressure, Pa*/
        double rho_n = P / (Const::K * Const::T0); /*Number Density, 1/m^3*/
        double sigma = Const::PI * pow(Const::r_air, 2);    /*m^2, Neutral air particle cross-sectional area*/
        double MFP = 1 / (rho_n * sigma); /*[m], Mean Free Path*/
        /*double MFP = 2.937e-7; [m], Mean Free Path through air particles at atmospheric pressure*/
        double Kn = MFP / (2 * part.r); /*Knudsen Number*/

        double nu_air = Const::mu_air / (rho_n * Const::M_air / Const::A); /*m^2/s, kinematic viscosity of air*/
        double Re = mag(part.vel) * 2 * part.r / nu_air; /*Reynolds number = v*d/nu */

        double Ma = (Re * Kn) / pow(Const::gamma_air * Const::PI / 2, 0.5);

        double Cd = 0; //initialize coefficient of drag

        double C_m;
        double G_m;
        if (Re > 45) {
            if (Ma <= 1.45) {
                C_m = 5 / 3 + (2 / 3) * tanh(3 * log(Ma + 1)); // Ma <=
                  1.45
            }
            else {
                C_m = 2.044 + 0.2 * exp(-1.8 * pow(log(Ma / 1.5), 2)); // Ma > 1.45
            }
            if (Ma <= 0.89) {
                G_m = 1 - 1.525 * pow(Ma, 4); // Ma <= 0.89
            }
            else {
                G_m = 0.0002 + 0.0008 * tanh(12.77 * (Ma - 2.02)); // Ma > 0.89
            }

            double H_m = 1 - (0.258 * C_m) / (1 + 514 * G_m);

            Cd = (24 / Re) * (1 + 0.15 * pow(Re, 0.687)) * H_m + (0.42 * C_m) / (1 + (42500 * G_m) / pow(Re, 1.16)); // Re > 45
        }

        else {
            double f_Kn = 1 / (1 + Kn * (2.514 + 0.8 * exp(-0.55 / Kn)));

            double CD_Kn_Re = (24 / Re) * (1 + 0.15 * pow(Re, 0.687)) * f_Kn;

            double s = Ma * sqrt(Const::gamma_air / 2);
            double Tratio = 1.0;
            double CD_fm = (1 + 2 * pow(s, 2)) * exp(-pow(s, 2)) / ((pow(s, 3)) * sqrt(Const::PI)) + (4 * pow(s, 4) + 4 * pow(s, 2) - 1) * erf(s) / (2 * pow(s, 4)) + (2 / (3 * s)) * sqrt(Const::PI * Tratio);

            double CD_fm_prime = (1 + 2 * pow(s, 2)) * exp(-pow(s, 2)) / ((pow(s, 3)) * sqrt(Const::PI)) + (4 * pow(s, 4) + 4 * pow(s, 2) - 1) * erf(s) / (2 * pow(s, 4));

            double CD_fm_Re = CD_fm / (1 + ((CD_fm_prime / 1.63) - 1) * sqrt(Re / 45));

            Cd = (CD_Kn_Re) / (1 + pow(Ma, 4)) + (pow(Ma, 4) * CD_fm_Re) / (1 + pow(Ma, 4)); // Re <= 45
        }

        //if (Kn >= 1e-3 && Kn < 1e-1) {
        //  if (Re > 1 && Re < 50) {
        //      if (Ma < 3.37) {
        //          Cd = (0.127 + 3.957 / (0.140 + pow(Re, 0.983))) * ((7.407 * Kn + 2.293) / (1.688 * Kn + 0.292)); //Niazmand and
```

```
     Anbarsooz (2012)
        //     //std::cout << "Low Kn Cd" << std::endl;
        //   }
        //   else {
        //     std::cout << "No Cd - Ma too high for low Kn term." << std::endl;
        //   }
        // }
        // else {
        //   std::cout << "No Cd - Re outside range for low Kn term." << std::endl;
        // }
        //}
        //else if (Kn >= 1e-1) {
        // if (Kn <= 1) {
        //   if (Re > 0.1 && Re < 3.5) {
        //     if (Ma > 0.0067 && Ma < 2.36) {
        //       Cd = 1; //Tao, Zhang, and Guo (2017)
        //       //std::cout << "Mid Kn Cd" << std::endl;
        //     }
        //     else {
        //       std::cout << "No Cd - Ma outside range for middle Kn term." << std::endl;
        //     }
        //   }
        //   else {
        //     std::cout << "No Cd - Re outside range for middle Kn term." << std::endl;
        //   }
        // }
        // else if (Kn <= 5.44) {
        //   if (Re > 2.99 && Re < 1378.81) {
        //     if (Ma > 10.96 && Ma < 25.29) {
        //       Cd = 1; //Singh and Schwartzenruber (2016)
        //       //std::cout << "High Kn Cd" << std::endl;
        //     }
        //     else {
        //       std::cout << "No Cd - Ma outside range for high Kn term." << std::endl;
        //     }
        //   }
        //   else {
        //     std::cout << "No Cd - Re outside range for high Kn term." << std::endl;
        //   }
        // }
        // else {
        //   std::cout << "No Cd - Kn too high" << std::endl;
        // }
        //}
        //std::cout << "Cd : " << Cd << std::endl;

        //F_D = 6 Pi Cd r v eta (eta = kinematic viscosity = rho nu, where nu = dynamic viscosity)
        for (int i = 0; i < 3; i += 1) {
            part.Dforce[i] = -Const::drag_bin * Cd * (Const::PI / 8) * (rho_n * Const::M_air / Const::A) *
                pow((2 * part.r), 2) * part.vel_for_drag[i] * abs(part.vel_for_drag[i]);
        }

        //Nearest-neighbor lookup on uniform mesh
        part.Eforce = compute_Efield_nn(results, part) * part.charge; //Ef = E*q

        //Hardline Efield such that 2D stays 2D
        for (int i = 0; i < 3; i += 1) {
            part.Cforce[i] = 0.0;
            part.force[i] = 0.0;
            if (part.pos[i] == 0.0) {
                part.Eforce[i] = 0.0;       // initialize to vacuum Lorentz force
            }
            else {
                continue;
            }
        }


        double3 r_im = { 0, 0, Const::electrode_h + Const::grid_thick + (Const::electrode_h - part.pos[2]) }; //Self Image Charge separation vector
        part.Cforce -= (part.charge * part.charge / Const::k) * r_im / (pow(mag(r_im), 3)); //Force from Self Image Charge

        //now add contribution from all other particles
        // F = q1*q2/(4*pi*eps0)*unit(r12)/(r12*r12)
        for (Cell& cell : part_grid.cells) {
            // skip empty cells
            if (cell.parts.empty()) continue;

            // compute distance to the mass centroid
            //double dist = mag(cell.mass_xc-part.pos);
            //if (dist>10*pg_size) {
                //std::cout << "PIC" << std::endl;
                //double3 r12 = part.pos - cell.mass_xc;
                //part.Cforce += part.charge*cell.charge/Const::k*unit(r12)/(dot(r12,r12));
```

```cpp
            //}
            //else {
                // loop over all particles and add individual contributions
            for (Particle* pp2 : cell.parts) {
                Particle& part2 = *pp2;     // dereference


                if (&part2 == &part) continue;  // ignore self, but would also be taken care of below

                double3 r12 = part.pos - part2.pos; //physical separation vector
                double3 p2_im = { part2.pos[0],  part2.pos[1], Const::electrode_h + Const::grid_thick + (Const::electrode_h - part2.pos[2]) };

                double3 r12_im = part.pos - p2_im; //separation vector from image charge
                part.Cforce += (part.charge * part2.charge / Const::k) * r12 / (pow(mag(r12), 3)); //Physical Plume Force
                part.Cforce -= (part.charge * part2.charge / Const::k) * r12_im / (pow(mag(r12_im), 3)); //Image Charge Plume Force

                //Collision Tracking
                double separation = mag(r12) - part.r - part2.r;
                if (separation < part.rsep) {
                    part.rsep = separation;
                    if (separation <= 0) { //overlapping particles (physical collision)
                        world.collision_count += 0.5; //only add 0.5 because both particles will count collision
                    }
                    else if (separation <= 0.1 * (part.r + part2.r)) { // near-collision particles, 10% from overlap
                        world.TJ_count += 0.5; //"Traffic Jam" count
                    }
                }
            }
        }


        part.force = part.Eforce + part.Cforce + part.Dforce;

        for (int i = 0; i < 3; i += 1) {
            if (part.Eforce[i] == 0.0) {
                part.frat[i] = 0.0;
            }
```

```
                else {
                    part.frat[i] = abs(part.Cforce[i])/abs(part.Eforce[i]);
                }
            }
        }

        // update positions
        for (Particle &part: particles)
        {
            /*increment particle's dt by world dt*/
            double dt = world.getDt();
            p_index += 1;

            /*keep iterating while time remains and the particle is alive*/
            part.pos += part.vel*dt + 0.5*part.acc*pow(dt,2); //using v and a 
              from last time step (not updated yet)

            /*calculate new acceleration from a = F/m */
            double3 new_acc = part.force / part.mass; //update particle 
              acceleration to new timestep

            /*update velocity to new timstep with dv/dt = (a + a_old)/2, 
              Velocity Verlet first order algorithm*/
            part.vel += dt * 0.5 * (part.acc + new_acc);

            part.vel_for_drag = part.vel +  dt * part.acc;

            //update particle acceleration to new timestep
            part.acc = new_acc;

            /*For 2 Particle*/
            //std::cout << "p_index: " << p_index << ", r: " << part.r << 
              std::endl;
            //if (p_index == 2) {//big droplet held fixed
                //std::cout << "Fixed particle: " << p_index << ", new 
                  position: " << part.pos << std::endl;
            //  continue;
            //}
            //else {
            //  part.pos += part.vel * dt; //small droplet moves
                //std::cout << "Mobile particle: " << p_index << ", new 
                  position: " << part.pos << std::endl;
            //}

            part.angle = atan(pow((pow(part.pos[0], 2) + pow(part.pos[1], 2)), 
               0.5) / part.pos[2]) * (180.0/Const::PI);


            /*did this particle leave the domain?*/
            if (!world.inBounds(part.pos) ||
                 world.inObject(part.pos)) {
                //if (!world.inBounds(part.pos)){std::cout << "Left domain: " 
                  << p_index << ", new position: " << part.pos << std::endl; }
                //if (world.inObject(part.pos)) { std::cout << "Hit object: " 
                  << p_index << ", new position: " << part.pos << std::endl; }
                part.mass = 0;  //kill the particle
            }

        }

        //remove dead particles
        size_t np = particles.size();

        for (size_t p=0;p<np;p++)
        {
            particles[p].birthID = p; //update birthID
            if (particles[p].mass>0) continue;  //ignore live particles
            //std::cout << "particle death, " << world.time << std::endl;
            size_t nep = exitparticles.size();
            for (size_t ep = 0; ep < nep; ep++)
                if (exitparticles[ep][0] == p * 1.0) {
                    double velx = particles[p].vel[0];
                    double vely = particles[p].vel[1];
                    double velz = particles[p].vel[2];
                    double thetaf = tan(pow(pow(particles[p].pos[0], 2) + pow 
                      (particles[p].pos[1], 2), 0.5) / particles[p].pos[2]) * 
                      180 / Const::PI;
                    double KE_q = 0.5 * particles[p].birthmass*pow(mag 
                      (particles[p].vel),2)/particles[p].charge;
                    exitparticles[ep][10] = thetaf;
                    exitparticles[ep][11] = velx;
                    exitparticles[ep][12] = vely;
                    exitparticles[ep][13] = velz;
                    exitparticles[ep][14] = KE_q;
                }
            particles[p] = particles[np-1]; //overwrite dead particle with 
              particle from the end the particles vector
            //for (size_t s = p; s < np - 1; s++) { //overwrite dead particle 
              by shuffling remainder of array to the left
            //  particles[s] = particles[s + 1]; //At this point, we've 
              written over dead particle. There is a duplicate last entry 
              which we will delete later
            //}
            np--;   //reduce count of valid elements
            p--;    //decrement p so this position gets checked again
        }

        //now delete particles[np:end]
        particles.erase(particles.begin()+np,particles.end());
    }


    /*adds a new particle, rewinding velocity by half dt*/
    void Droplets::addParticle(double birthID, double3 pos, double3 vel, 
      double3 acc, double3 Cforce, double3 Eforce, double3 Dforce, double3 
      Iforce, double r, double qm, double addtime)
    {
        //don't do anything (return) if pos outside domain bounds
        if (!world.inBounds(pos)){
            std::cout<<"emitted out of bounds"<<std::endl;
            return;}

        // TODO: add contribution from other particles

        //get particle logical coordinate
        // double3 lc = world.XtoL(pos);

        //evaluate electric field at particle position
       //  double3 ef_part = world.ef.gather(lc);

        //rewind velocity back by 0.5*dt*ef

        //add to list
        particles.emplace_back(birthID, pos,vel,acc, Cforce, Eforce, Dforce, 
          Iforce, r, qm, addtime);
        double thetaf = 0.0;
        double vfx = 0.0;
        double vfy = 0.0;
        double vfz = 0.0;
        double KE_q = 0.0;
        double V = 4.0 /3.0 * Const::PI * pow(r,3);
        double mass = V * Const::density;
        double charge = mass * qm;
        exitparticles.push_back({ birthID, pos[0], pos[1], pos[2], vel[0], vel
          [1], vel[2], r, mass, charge, thetaf, vfx, vfy, vfz, KE_q });
        //std::cout << "particle born, " << addtime << std::endl;
        //std::cout <<"emission location:" << pos <<std::endl;
        //std::cout <<"emitted r:" << r <<std::endl;
        //std::cout <<"emitted qm:" << qm <<std::endl;
    }

    /*returns perturbed droplet emission*/
    double3 Droplets::emitPerturbed(double rscale) {
        // random device class instance, source of 'true' randomness for 
          initializing random seed
        std::random_device rd;

        // Mersenne twister PRNG, initialized with seed from previous random 
          device instance
        std::mt19937 gen(rd());
        //Gaussian distribution
        std::normal_distribution<double> pert_distribution(0,rscale);


        //Gaussian radial perturbation and random from set range axial 
          perturbation
        double r_pert = pert_distribution(gen);
        double theta = 2 * Const::PI * rnd();

        double x_pert = cos(theta)*r_pert;
        double y_pert = sin(theta)*r_pert;

        double3 pos;
        pos[0] = x_pert;
        pos[1] = y_pert;
        pos[2] = Const::EmHeight;

        return pos;
    }

    /* computes the adaptive timestep*/
    double Droplets::adaptTimeStep(double default_dt)
    {
        double ts;
        double mints = default_dt;

        for (Particle &part:particles){
            double minr = 1;
            for (Particle &part2:particles) {
                if (&part2 == &part) continue;    // ignore self
                double3 r12 = part.pos - part2.pos;
                double r_mag = mag(r12);
                if (r_mag < minr){
                    minr = r_mag;
                }
            }
            ts = (-mag(part.vel) + sqrt(pow(mag(part.vel),2) + 2*mag(part.acc) 
              *minr))/mag(part.acc);
            if (ts < mints){
                mints = ts;
            }
        }
        //std::cout << "adapted ts: " << mints << std::endl;
        return mints;
    }

```

```

/* returns the species momentum*/
double3 Droplets::getMomentum() {
    double3 mom;
    for (Particle &part:particles)
        mom+=part.mass*part.vel;
    return mom;
}

/* returns the species kinetic energy*/
double Droplets::getKE() {
    double ke = 0;
    for (Particle &part:particles)
    {
        double v2 = mag(part.vel)*mag(part.vel);
        ke += 0.5*part.mass*v2;
    }
    return ke;
}

/*compute instantaneous densities*/
void Droplets::computeInstDensities()
{
    inst_den.clear();
    inst_mden.clear();
    inst_qden.clear();
    force.clear();
    for (Particle &part:particles)
    {
        double3 lc = world.XtoL(part.pos);
        inst_den.scatter(lc, 1);
        inst_mden.scatter(lc, part.mass);
        inst_qden.scatter(lc, part.charge);
        force.scatter(lc, part.force);
    }

    //divide by node surface area (in xy plane) for [units]/m^2 density
    inst_den / (world.node_vol*world.dh3); //node_vol *Idh3 = node_vol
      *cell_height = node_xy_surface_area
    inst_mden / (world.node_vol *world.dh3);
    inst_qden / (world.node_vol *world.dh3);
}

/*compute steady-state average densities*/
void Droplets::computeAveDensities()
{

    //std::cout << "t_steady: " << world.t_steady << std::endl;
    double t_last = world.t_steady - 1.0;
    //std::cout << "t_last: " << t_last << std::endl;
    ave_den *= t_last;
    ave_mden *= t_last;
    ave_qden *= t_last;
    //std::cout << "ave_den: " << ave_den << std::endl;
    ave_den += inst_den;
    ave_mden += inst_mden;
    ave_qden += inst_qden;
    //std::cout << "new_den: " << new_den << std::endl;
    ave_den /= world.t_steady;
    ave_mden /= world.t_steady;
    ave_qden /= world.t_steady;
    //std::cout << "new ave_den: " << ave_den << std::endl;
}

/*computes space charge at each mesh point using single avg. point 
  charge*/
void Droplets::computeSpaceCharge() {
    Cf.clear();
    for (int i = 0; i < Const::x_n; i++) {
        for (int j = 0; j < Const::y_n; j++) {
            for (int k = 0; k < Const::z_n; k++) {
                double3 C = { 0.0, 0.0, 0.0 };  //initialize space charge 
                  to 0 at each node
                double3 P = world.pos(i, j, k); //compute node position
                for (Particle& part : particles) {
                    double3 r12 = P - part.pos;
                    double r_mag = mag(r12);
                    C += Const::qmean * part.charge / Const::k * unit 
                  (r12) / (r_mag * r_mag);
                }
                Cf[i][j][k] = C;
            }
        }
    }
}

/*computes cumulative profiles*/
void Droplets::computeCumulativeContours()
{
    //radfrat.clear(); /*instantaneous, not cumulative*/
    for (Particle& part: particles)
    {
        double3 lc = world.XtoL(part.pos);
        mass_cm.scatter(lc, part.mass);
        charge_cm.scatter(lc, part.charge);
        radfrat.scatter(lc, pow(pow(part.frat[0], 2) + pow(part.frat[1], 
          2), 0.5));
    }
```

```
}

/*samples velocity moments*/
void Droplets::sampleMoments() {
    for (Particle &part:particles)
    {
        double3 lc = world.XtoL(part.pos);
        n_sum.scatter(lc, part.mass);
        nv_sum.scatter(lc,part.mass*part.vel);
    }
}

/*uses sampled data to compute velocity and temperature*/
void Droplets::computeGasProperties() {
    vel = nv_sum/n_sum; //stream velocity
}


/*clears sampled moment data*/
void Droplets::clearSamples() {
    n_sum = 0; nv_sum = 0;
}

/*computes emitted mass flowrate and current*/
void Droplets::computeFlow() {
    double m = 0;
    double q = 0;
    for (Particle& part : particles)
    {
        m += part.mass;
        q += part.charge;
    }
    world.mflow = m / world.time;
    world.qflow = q / world.time;
}

// creates a list of pointers to all particles in a cell,
// also computes total charge in the cell
void Droplets::sortToCells() {
    part_grid.clear();
    for (Particle &part:particles) {
        int c = part_grid.XtoC(part.pos);
        if (c<0) continue;

        part_grid.cells[c].parts.push_back(&part);
        part_grid.cells[c].charge += part.charge;
    }

    // set mass centroid in each cell
    for (Cell &cell:part_grid.cells) {
        double mass = 0;
        cell.mass_xc = 0;
        for (Particle *part:cell.parts) {
            cell.mass_xc += part->mass*part->pos;
            mass+=part->mass;
        }
        cell.mass_xc /= mass;
    }
}

double3 Droplets::compute_Efield_nn(std::vector<std::pair<std::string,
  std::vector<double>>> result,  Particle& part)
{
    double r = sqrt(pow(part.pos[0], 2) + pow(part.pos[1], 2)); //r^2 =
      x^2 + y^2
    //double theta = (180/3.14)*atan(part.pos[1] / part.pos[0]); //theta =
       atan(y/x)
    double theta = (180 / 3.14) * atan2(part.pos[1], part.pos[0]); //theta
       = atan(y/x)

    double rmin = 100.0;
    double3 Ef;

    double r_grid; double z_grid;
    int rmin_index;
    double Er; double Ez;


    for (int i = 0; i < result.at(0).second.size(); i++) {
        double dist = pow((pow((r - result.at(0).second[i]), 2) + pow
          ((part.pos[2] - result.at(1).second[i]), 2)), 0.5);
        if (dist < rmin) {
            rmin = dist;
            rmin_index = i;
        }
    }
    //std::cout << "rmin: " << rmin << ", id: " << rmin_index <<
      std::endl;
    Er = result.at(2).second[rmin_index];
    Ez = result.at(3).second[rmin_index];

    double Ex = Er * cos(theta * (3.14 / 180)); //x = r*cos(theta)
    double Ey = Er * sin(theta * (3.14 / 180)); //y = r*sin(theta)
    Ef = { Ex, Ey, Ez };
    r_grid = result.at(0).second[rmin_index];
    z_grid = result.at(1).second[rmin_index];

    //std::cout << "x: " << part.pos[0] << ", y: " << part.pos[1] << ", z:
        " << part.pos[2] << std::endl;
    //std::cout << "r: " << r << ", z: " << part.pos[2] << std::endl;
    //std::cout << "theta: " << theta << std::endl;
    //std::cout << "r_grid: " << r_grid << ", z_grid: " << z_grid << 
      std::endl;
    //std::cout << "Er: " << Er << std::endl;
    //std::cout << "Ex: " << Ef[0] << ", Ey: " << Ef[1] << ", Ez: " << Ef 
      [2] << std::endl;
    return Ef;
}

double3 Droplets::compute_Efield_bc(std::vector<std::pair<std::string, 
  std::vector<double>>> result, Particle& part)
{
    double r = sqrt(pow(part.pos[0], 2) + pow(part.pos[1], 2)); //r^2 = 
      x^2 + y^2
    //double theta = (180/3.14)*atan(part.pos[1] / part.pos[0]); //theta = 
       atan(y/x)
    double theta = (180 / 3.14) * atan2(part.pos[1], part.pos[0]); //theta 
       = atan(y/x)

    double rmin1 = 100.0; double rmin2 = 100.0; double rmin3 = 100.0; 
      double rmin4 = 100.0;
    int r1 = 1; int r2 = 1; int r3 = 1; int r4 = 1;
    double rmin4_proposed; int r4_proposed;
    //int *r1p = &r1; int* r2p = &r2; int* r3p = &r3;

    double3 Ef;
    double r_grid;
    double z_grid;
    double Er;

    for (int i = 0; i < result.at(0).second.size(); i++) {
        double dist = pow((pow((r - result.at(0).second[i]), 2) + pow
          ((part.pos[2] - result.at(1).second[i]), 2)), 0.5);
        if (dist < rmin3) {
            rmin3 = dist;
            //*r3p = i;
            r3 = i;
            if (dist < rmin2) {
                rmin3 = rmin2;
                //*r3p = *r2p;
                r3 = r2;
                r4_proposed = r3;//Propose P4 = old P3, check later that 
                  it's not collinear with P1 & P2
                rmin4_proposed = rmin3;//Propose P4 = old P3, check later 
                  that it's not collinear with P1 & P2
                rmin2 = dist;
                //*r2p = i;
```

```cpp
                r2 = i;
                if (dist < rmin1) {
                    rmin2 = rmin1;
                    //*r2p = *r1p;
                    r2 = r1;
                    rmin1 = dist;
                    //*r1p = i;
                    r1 = i;
                }
            }
            //Check that proposed P4 is not (very nearly) collinear with P1 & P2
            if (abs(((result.at(1).second[r2] - result.at(1).second[r1]) * (result.at(0).second[r4_proposed] - result.at(0).second[r1]) + (result.at(0).second[r1] - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at(1).second[r4_proposed]))) > 1e-15) {
                r4 = r4_proposed;
                rmin4 = rmin4_proposed;
                //std::cout << "Found new nonlinear close vertex" << std::endl;
            }
        }
    }
    std::cout << "|Denom 124|: " << abs(((result.at(1).second[r2] - result.at(1).second[r1]) * (result.at(0).second[r4] - result.at(0).second[r1]) + (result.at(0).second[r1] - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at(1).second[r4]))) << std::endl;

    //If 3 closest points are (very nearly) collinear, bump the third point for the third guaranteed non-collinear close point
    if (abs(((result.at(1).second[r2] - result.at(1).second[r3]) * (result.at(0).second[r1] - result.at(0).second[r3]) + (result.at(0).second[r3] - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at(1).second[r3]))) < 1e-15) {
        std::cout << "|Denom 123|: " << abs(((result.at(1).second[r2] - result.at(1).second[r3]) * (result.at(0).second[r1] - result.at(0).second[r3]) + (result.at(0).second[r3] - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at(1).second[r3]))) << std::endl;
        std::cout << "Replaced interpolation vertex due to collinearity" << std::endl;
        std::cout << "|Denom 123|: " << abs(((result.at(1).second[r2] - result.at(1).second[r3]) * (result.at(0).second[r1] - result.at(0).second[r3]) + (result.at(0).second[r3] - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at(1).second[r3]))) << std::endl;
        r3 = r4;
        rmin3 = rmin4_proposed;
    }

    //Check for collinearity among all 3 points
    //double slope_12 = (result.at(1).second[r2] - result.at(1).second[r1]) / (result.at(0).second[r2] - result.at(0).second[r1]);
    //double slope_23 = (result.at(1).second[r3] - result.at(1).second[r2]) / (result.at(0).second[r3] - result.at(0).second[r2]);
    //double slope_13 = (result.at(1).second[r3] - result.at(1).second[r1]) / (result.at(0).second[r3] - result.at(0).second[r1]);
    //if (slope_12 == slope_23 && slope_23 == slope_13 && slope_12 == slope_13) {//if collinear, find 4th point - nearest point not on line.
        //double rmin4 = 100;
        //int r4 = 1;
        //for (int i = 0; i < result.at(0).second.size(); i++) {
            //double dist = pow((pow((r - result.at(0).second[i]), 2) + pow((part.pos[2] - result.at(1).second[i]), 2)), 0.5);
            //if (dist <= rmin3) {//ignore the 3 closest points already found
                //continue;
            //}
            //else if (dist < rmin4) {
                //double slope_14 = (result.at(1).second[i] - result.at(1).second[r1]) / (result.at(0).second[i] - result.at(0).second[r1]);
                //if (slope_14 != slope_12) {//only accept 4th point not on line
                    //rmin4 = dist;
                    //r4 = i;
                //}
            //}
        //}//Once all points have been searched for to fine 4th point closest not collinear with closest 3, replace 3rd closest point with new 4th point
        //rmin3 = rmin4;
        //r3 = r4;
        //std::cout << "Replaced interpolation vertext due to collinearity" << std::endl;
    //}

    //std::cout << "rmin1: " << rmin1 << ", r1: " << r1 << std::endl;
    //std::cout << "rmin2: " << rmin2 << ", r2: " << r2 << std::endl;
    //std::cout << "rmin3: " << rmin3 << ", r3: " << r3 << std::endl;

    //Barycentric Coordinates:
    //Ef_p = w_1 * Ef_1 + w_2 + Ef_2 + w_3 + Ef_3
    //Weights are defined as follows:
    //x_p = w_1*x_1 + w_2*x_2 + w_3*x_3
    //y_p = w_1*y_1 + y_2*x_2 + w_3*y_3
    //w_1 + w_2 + w_3 = 1
    //Re-arranged:
    //w_1 = ((y_2 - y_3)(x_p - x_3) + (x_3 - x_2)(y_p - y_3))/((y_2 - y_3)
      (x_1 - x_3) + (x_3 - x_2)(y_1 - y_3))
    //w_2 = ((y_3 - y_1)(x_p - x_3) + (x_1 - x_3)(y_p - y_3))/((y_2 - y_3)
      (x_1 - x_3) + (x_3 - x_2)(y_1 - y_3))
    //w_3 = 1 - w_1 - w_2;


    double w_1 = ((result.at(1).second[r2] - result.at(1).second[r3]) * (r
       - result.at(0).second[r3]) + (result.at(0).second[r3] - result.at
      (0).second[r2]) * (part.pos[2] - result.at(1).second[r3])) /
      ((result.at(1).second[r2] - result.at(1).second[r3]) * (result.at
      (0).second[r1] - result.at(0).second[r3]) + (result.at(0).second[r3]
       - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at
      (1).second[r3]));
    double w_2 = ((result.at(1).second[r3] - result.at(1).second[r1]) * (r
       - result.at(0).second[r3]) + (result.at(0).second[r1] - result.at
      (0).second[r3]) * (part.pos[2] - result.at(1).second[r3])) /
      ((result.at(1).second[r2] - result.at(1).second[r3]) * (result.at
      (0).second[r1] - result.at(0).second[r3]) + (result.at(0).second[r3]
       - result.at(0).second[r2]) * (result.at(1).second[r1] - result.at
      (1).second[r3]));
    double w_3 = 1 - w_1 - w_2;
    //std::cout << "w1: " << w_1 << ", w2: " << w_2 << ", w3: " << w_3 <<
      std::endl;
    //std::cout << "Er_1: " << result.at(2).second[r1] << ", Er_2: " <<
      result.at(2).second[r2] << ", Er_3: " << result.at(2).second[r3] <<
      std::endl;

    Er = w_1 * result.at(2).second[r1] + w_2 * result.at(2).second[r2] +
      w_3 * result.at(2).second[r3];
    double Ez = w_1 * result.at(3).second[r1] + w_2 * result.at(3).second
      [r2] + w_3 * result.at(3).second[r3];

    double Ex = Er * cos(theta * (3.14 / 180)); //x = r*cos(theta)
    double Ey = Er * sin(theta * (3.14 / 180)); //y = r*sin(theta)
    Ef = { Ex, Ey, Ez };
    r_grid = result.at(0).second[r1];
    z_grid = result.at(1).second[r1];

    //std::cout << "x: " << part.pos[0] << ", y: " << part.pos[1] << ", z:
       " << part.pos[2] << std::endl;
    //std::cout << "r: " << r << ", z: " << part.pos[2] << std::endl;
    //std::cout << "theta: " << theta << std::endl;
    //std::cout << "r_grid: " << r_grid << ", z_grid: " << z_grid <<
      std::endl;
    //std::cout << "Er: " << Er << std::endl;
    //std::cout << "Ex: " << Ef[0] << ", Ey: " << Ef[1] << ", Ez: " << Ef 
      [2] << std::endl;
    return Ef;
}

double3 Droplets::compute_Efield_LOC(Particle& part) {
    //Analytical Solution form
    //R = nozzle OR = 0.5e-3m, H = emitter - to - plate distance = 0.03m
    //r * = r / H, z * =  1 - (z-h)/H
    //Wilhelm reports best fit with k = 0.7
    //phi = k / ln(4H / R) * ln(((r * ^2 + (1 - z*) ^ 2) ^ (1 / 2) + (1 - 
      z*)) / ((r * ^2 + (1 + z*) ^ 2) ^ (1 / 2) + (1 + z*)))

    int V0 = Const::voltage;
    double H = Const::Em_to_grid;
    double h = Const::emitter_h;
    double R = Const::emitter_rad;
    double kv = 0.7; //dimensionless constant

    double r = sqrt(pow(part.pos[0],2) + pow(part.pos[1],2)); // create r 
      array from xand y arrays
    double r_s = r / H;
    double z_s = 1 - (part.pos[2] - h) / H;
    double c = (kv / log(4 * H / R));

    double num = (pow((pow(r_s,2) + pow((1 - z_s),2)),(1 / 2))) + (1 - 
      z_s);
    double denom = (pow((pow(r_s,2) + pow((1 + z_s),2)), (1 / 2))) + (1 + 
      z_s);

    //num_dz = (1 / 2) * ((r_s ^ 2 + (1 - z_s) ^ 2) ^ (-1 / 2)) * 2 * (1 - 
       z_s) * (-1) - 1
    double num_dz = -(pow((pow(r_s, 2) + pow((1 - z_s),2)),(-1 / 2))) * (1 
       - z_s) - 1;
    //num_dr = (1 / 2) * ((r_s ^ 2 + (1 - z_s) ^ 2) ^ (-1 / 2)) * 2 * r_s
    double num_dr = (pow((pow(r_s,2) + pow((1 - z_s),2)),(-1 / 2))) * r_s;

    //denom_dz = (1 / 2) * ((r_s ^ 2 + (1 + z_s) ^ 2) ^ (-1 / 2)) * 2 * (1 
       + z_s) + 1
    double denom_dz = (pow((pow(r_s ,2) + pow((1 + z_s),2)),(-1 / 2))) * 
      (1 + z_s) + 1;
    //denom_dr = (1 / 2) * ((r_s ^ 2 + (1 + z_s) ^ 2) ^ (-1 / 2)) * 2 * 
      r_s
    double denom_dr = (pow((pow(r_s,2) + pow((1 + z_s),2)),(-1 / 2))) * 
      r_s;

    //Quotient rule : d(phi) / dz = (denom * d(num) / dz - num * d 
      (denom) / dz) / (denom) ^ 2
    double phi_dz = c* (denom * num_dz - num * denom_dz) / pow((denom), 
    2);
    double phi_dr = c* (denom * num_dr - num * denom_dr) / pow((denom), 
    2);

    double Ez = V0 / H * phi_dz;
    double Er = V0 / H * phi_dr;

    //Trig. conversion from 2D Axisymmetric to 3D
    double theta = (180 / 3.14) * atan2(part.pos[1], part.pos[0]); //theta 
     = atan(y/x)
    double Ex = Er * cos(theta * (3.14 / 180)); //x = r*cos(theta)
    double Ey = Er * sin(theta * (3.14 / 180)); //y = r*sin(theta)
    double3 Ef = { Ex, Ey, Ez };
    return Ef;
}

double3 Droplets::compute_Efield_ANLTC(Particle& part) {
    //Create Analytical Potential Solution Array From GC 94
    double H = Const::Em_to_grid; //Emitter - to - Extractor Height [m]
    double Ra = Const::emitter_rad; //Emitter radius [m]
    double Kv = 0.685; //Kv = fn(H/Ra) value for H/Ra = 60

    double za = ((part.pos[2] - Const::emitter_h - Const::TC_h) - 
    Const::Em_to_grid) / Const::Em_to_grid; // Set z coordinate origin 
    on top of emitter and TC, reverse for distance from plate [m], 
    nondimensionalize
    double ra = pow(pow(part.pos[0], 2) + pow(part.pos[1], 2), 0.5) / 
    Const::Em_to_grid; // Nondimensionalize r
    //std::cout << "(ra, za): " << ra << " , " << za << std::endl;;


    double A_V = Const::voltage * Kv / log(4 * H / Ra) * log((pow(pow(ra, 
    2) + pow(1 - za, 2), 0.5) + (1 - za)) / (pow(pow(ra, 2) + pow(1 + 
    za, 2), 0.5) + (1 + za))); // [V]
    double Er = -(Const::voltage * Kv * (ra / (pow(pow(za - 1, 2) + pow
    (ra, 2), 0.5) * (za + pow(pow(za + 1, 2) + pow(ra, 2), 0.5) + 1)) - 
    (ra * (pow(pow(za - 1, 2) + pow(ra, 2), 0.5) - za + 1)) / (pow(pow
    (za + 1, 2) + pow(ra, 2), 0.5) * pow((za + pow(pow(za + 1, 2) + pow
    (ra, 2), 0.5) + 1), 2))) * (za + pow(pow(za + 1, 2) + pow(ra, 2), 
    0.5) + 1)) / (H * log((4 * H) / Ra) * (pow(pow(za - 1, 2) + pow(ra, 
    2), 0.5) - za + 1)); // [V/m]
    double Ez = -(Const::voltage * Kv * (((2 * za - 2) / (2 * pow(pow(za - 
     1, 2) + pow(ra, 2), 0.5)) - 1) / (za + pow(pow(za + 1, 2) + pow(ra, 
     2), 0.5) + 1) - (((2 * za + 2) / (2 * pow(pow(za + 1, 2) + pow(ra, 
    2), 0.5)) + 1) * (pow(pow(za - 1, 2) + pow(ra, 2), 0.5) - za + 1)) / 
     pow(za + pow(pow(za + 1, 2) + pow(ra, 2), 0.5) + 1, 2)) * (za + pow
    (pow(za + 1, 2) + pow(ra, 2), 0.5) + 1)) / (H * log((4 * H) / Ra) * 
    (pow(pow(za - 1, 2) + pow(ra, 2), 0.5) - za + 1)); // [V/m]


    //Trig. conversion from 2D Axisymmetric to 3D
    double theta = (180 / 3.14) * atan2(part.pos[1], part.pos[0]); //theta
      = atan(y/x)
    double Ex = Er * cos(theta * (3.14 / 180)); //x = r*cos(theta)
    double Ey = Er * sin(theta * (3.14 / 180)); //y = r*sin(theta)
    double3 Ef = { Ex, Ey, Ez };
    return Ef;
}

```

```
#ifndef _OUTPUT_H
#define _OUTPUT_H

#include <vector>
#include <fstream>
#include "World.h"
#include "PotentialSolver.h"
#include "Droplets.h"

namespace Output {
    void fields(World &world, Droplets &droplets);
    void particles(World& world, Droplets& droplets);
    void screenOutput(World &world, Droplets &droplets);
    void diagOutput(World &world, Droplets &droplets);
    void exitparticlesOutput(Droplets& sp);
    void convOutput(int i, double tol, double err);
    void PSOutput(World& world);
    void runtimesOutput(double make_world_time, double add_objects_time, 
      double solve_potential_time, double solve_electric_field, double 
      create_species_time, double emit_time, double sort_cells_time, double 
       advance_time, double compute_densities_time, double 
      compute_cumulatives_time, double output_time);
    void nOutput(int n, int ts, double t);
}

#endif

```

```
#include <fstream>
#include <sstream>
#include <iostream>
#include <iomanip>
#include "Output.h"
#include "World.h"
#include "Droplets.h"

using namespace std;


//writes information to the screen
void Output::screenOutput(World& world, Droplets& sp)
{
    cout << "ts: " << world.getTs();
    cout << setprecision(3) << "\t " << sp.name << ":" << sp.getNp();
    cout << endl;
}

//file stream handle
namespace Output {
    std::ofstream f_diag;
    std::ofstream f_conv;
    std::ofstream f_PS;
    std::ofstream f_runtimes;
    std::ofstream f_ep;
}

/*save runtime diagnostics to a file*/
void Output::diagOutput(World& world, Droplets& sp)
{
    using namespace Output; //to get access to f_diag

    //is the file open?
    if (!f_diag.is_open())
    {
        f_diag.open("diagnostics.csv");
        f_diag << "ts,time,wall_time";
        f_diag << ",mp_count." << sp.name
            << ",px." << sp.name << ",py." << sp.name << ",pz." << sp.name
            << ",KE." << sp.name;
        f_diag << endl;
    }

    f_diag << world.getTs() << "," << world.getTime();
    f_diag << "," << world.getWallTime();

    double tot_KE = 0;
    double KE = sp.getKE(); //species kinetic energy
    tot_KE += KE;       //increment total energy
    double3 mom = sp.getMomentum();

    f_diag << "," << sp.getNp()
        << "," << mom[0] << "," << mom[1] << "," << mom[2] << "," << KE;

    //write out system potential and total energy

    f_diag << "\n"; //use \n to avoid flush to disc
    if (world.getTs() % 25 == 0) f_diag.flush();
}

void Output::exitparticlesOutput(Droplets& sp)
{
    using namespace Output; //to get access to f_diag

    //is the file open?
    if (!f_ep.is_open())
    {
        f_ep.open("exitparticles.csv");
        f_ep << "birthID, bpx, bpy, bpz, bvz, bvy, bvz, r, m, q, thetaf, 
          vfx, vfy, vfz, KE_q" << endl;
    }

    for (std::vector<double> ep : sp.exitparticles) {
        f_ep << ep[0] << "," << ep[1] << "," << ep[2] << "," << ep[3] << 
          "," << ep[4];
        f_ep << "," << ep[5] << "," << ep[6] << "," << ep[7] << "," << ep
          [8];
        f_ep << "," << ep[9] << "," << ep[10] << "," << ep[11];
        f_ep << "," << ep[12] << "," << ep[13] << "," << ep[14];
        f_ep << "\n";
    }
    f_ep.flush();
}

/*save potential solver convergence diagnostics to a file*/
void Output::convOutput(int i, double tol, double err)
{
    using namespace Output; //to get access to f_conv

    //check if file is open
    if (!f_conv.is_open())
    {
        f_conv.open("PS_convergence.csv"); //filename
        f_conv << "i, tolerance, error"; //labels
        f_conv << endl;
    }

```

```cpp
    f_conv << i << "," << tol; //iteration number and tolerance threshold
    f_conv << "," << err; //error in iteration step

    f_conv << "\n"; //use \n to avoid flush to disc
    if (i % 1 == 0) f_conv.flush(); //keep information from all iteration steps
}

/*save number of droplets at each (n) timestep to a file*/
void Output::nOutput(int n, int ts, double t)
{
    using namespace Output; //to get access to f_conv

    //check if file is open
    if (!f_conv.is_open())
    {
        f_conv.open("n_droplets.csv"); //filename
        f_conv << "n, timestep, time"; //labels
        f_conv << endl;
    }

    f_conv << n << "," << ts; //iteration number and tolerance threshold
    f_conv << "," << t; //error in iteration step

    f_conv << "\n"; //use \n to avoid flush to disc
    if (ts % 1 == 0) f_conv.flush(); //keep information from all iteration steps
}

/*save runtime diagnostics to a file*/
void Output::PSOutput(World& world)
{
    using namespace Output; //to get access to f_PS

    //is the file open?
    if (!f_PS.is_open())
    {
        f_PS.open("PS.csv");
        f_PS << "x, y, z, phi";
        f_PS << endl;
    }

    for (int i = 0; i < world.ni; i++) {
        for (int j = 0; j < world.nj; j++) {
            for (int k = 0; k < world.nk; k++) {
                f_PS << world.pos(i, j, k)[0] << "," << world.pos(i, j, k)[1] << "," << world.pos(i, j, k)[2]; //iteration number and tolerance threshold
                f_PS << "," << world.phi[i][j][k]; //error in iteration
```

```
                     step
                    f_PS << "\n";   //use \n to avoid flush to disc
                    f_PS.flush(); //store information
                }
            }
        }
}

void Output::runtimesOutput(double make_world_time, double
  add_objects_time, double solve_potential_time, double
  solve_electric_field, double create_species_time, double emit_time,
  double sort_cells_time, double advance_time, double
  compute_densities_time, double compute_cumulatives_time, double
  output_time) {

    using namespace Output; //to get access to f_runtimes

    //check if file is open
    if (!f_runtimes.is_open())
    {
        f_runtimes.open("runtimes.csv"); //filename
        f_runtimes << "make world, add objects, solve potential, solve
          electric field, create species, emit, sort cells, advance,
          compute densities, compute cumulatives, output";
        f_runtimes << endl;
    }

    f_runtimes << make_world_time << "," << add_objects_time << "," <<
      solve_potential_time << "," << solve_electric_field << "," <<
      create_species_time;
    f_runtimes << "," << emit_time << "," << sort_cells_time << "," <<
      advance_time << "," << compute_densities_time << "," <<
      compute_cumulatives_time << "," << output_time;

    f_runtimes << "\n"; //use \n to avoid flush to disc
    f_runtimes.flush(); //keep information from all iteration steps
}

/*saves fields in VTK format*/
void Output::fields(World& world, Droplets& sp)
{
    /*update gas macroscopic properties*/
    sp.computeGasProperties();

    stringstream name;
    name << "results/fields_" << setfill('0') << setw(5) << world.getTs()
      << ".vti";

    /*open output file*/
```

```cpp
    ofstream out(name.str());
    if (!out.is_open()) { cerr << "Could not open " << name.str() << endl; return; }

    /*ImageData is vtk format for structured Cartesian meshes*/
    out << "<VTKFile type=\"ImageData\">\n";
    double3 x0 = world.getX0();
    double3 dh = world.getDh();
    out << "<ImageData Origin=\"" << x0[0] << " " << x0[1] << " " << x0[2] << "\" ";
    out << "Spacing=\"" << dh[0] << " " << dh[1] << " " << dh[2] << "\" ";
    out << "WholeExtent=\"0 " << world.ni - 1 << " 0 " << world.nj - 1 << " 0 " << world.nk - 1 << "\">\n";

    /*output data stored on nodes (point data)*/
    out << "<PointData>\n";

    /*object id, scalar*/
    out << "<DataArray Name=\"object_id\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Int32\">\n";
    out << world.object_id;
    out << "</DataArray>\n";

    /*cell position, vector*/
    for (int i = 0; i < world.ni; i++) {
        for (int j = 0; j < world.nj; j++) {
            for (int k = 0; k < world.nk; k++) {
                world.Pos[i][j][k] = world.pos(i, j, k);
            }
        }
    }
    out << "<DataArray Name=\"Pos\" NumberOfComponents=\"3\" format=\"ascii\" type=\"Float64\">\n";
    out << world.Pos;
    out << "</DataArray>\n";

    /*node volumes, scalar*/
    Field f(world.node_vol);
    for (int i = 0; i < world.ni; i++)
        for (int j = 0; j < world.nj; j++)
            for (int k = 0; k < world.nk; k++)
                f[i][j][k] = world.node_vol[i][j][k];
    out << "<DataArray Name=\"NodeVol\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << world.node_vol;
    out << "</DataArray>\n";

    /*cell height, scalar*/
    out << "<DataArray Name=\"cell_height\" NumberOfComponents=\"1\"
```

```
        format=\"ascii\" type=\"Float64\">\n";
    out << world.dh3;
    out << "</DataArray>\n";


    /*potential, scalar*/
    out<<"<DataArray Name=\"phi\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out<<world.phi;
    out<<"</DataArray>\n";

    /*charge density, scalar*/
    //out << "<DataArray Name=\"rho\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    //out << world.rho;
    //out << "</DataArray>\n";

    /*instantaneous species number densities, scalar*/
    out<<"<DataArray Name=\"inst_den."<<sp.name<<"\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out<<sp.inst_den;
    out<<"</DataArray>\n";

    /*instantaneous species mass densities, scalar*/
    out << "<DataArray Name=\"inst_mden." << sp.name << "\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.inst_mden;
    out << "</DataArray>\n";

    /*instantaneous species charge densities, scalar*/
    out << "<DataArray Name=\"inst_qden." << sp.name << "\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.inst_qden;
    out << "</DataArray>\n";

    /*average species number densities, scalar*/
    out << "<DataArray Name=\"ave_den." << sp.name << "\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.ave_den;
    out << "</DataArray>\n";

    /*average species mass densities, scalar*/
    out << "<DataArray Name=\"ave_mden." << sp.name << "\" NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.ave_mden;
    out << "</DataArray>\n";

    /*average species charge densities, scalar*/
    out << "<DataArray Name=\"ave_qden." << sp.name << "\"
```

```
      NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.ave_qden;
    out << "</DataArray>\n";

    /*cumulative mass, scalar*/
    out << "<DataArray Name=\"mass_cm." << sp.name << "\" 
      NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.mass_cm;
    out << "</DataArray>\n";

    /*cumulative charge, scalar*/
    out << "<DataArray Name=\"charge_cm." << sp.name << "\" 
      NumberOfComponents=\"1\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.charge_cm;
    out << "</DataArray>\n";

    /*total force, vector*/
    out << "<DataArray Name=\"force." << sp.name << "\" 
      NumberOfComponents=\"3\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.force;
    out << "</DataArray>\n";

    /*time, scalar*/
    world.Time = world.time;
    out << "<DataArray Name=\"time\" NumberOfComponents=\"1\" format=
      \"ascii\" type=\"Float64\">\n";
    out << world.Time;
    out << "</DataArray>\n";

    /*species averaged number densities*/
    //out<<"<DataArray Name=\"nd-ave."<<sp.name<<"\" NumberOfComponents=
      \"1\" format=\"ascii\" type=\"Float64\">\n";
    //out<<sp.den_ave;
    //out<<"</DataArray>\n";

    /*species stream velocity, 3 component vector*/
    out<<"<DataArray Name=\"vel."<<sp.name<<"\" NumberOfComponents=\"3\" 
      format=\"ascii\" type=\"Float64\">\n";
    out<<sp.vel;
    out<<"</DataArray>\n";

    /*electric field, 3 component vector*/
    out<<"<DataArray Name=\"ef\" NumberOfComponents=\"3\" format=\"ascii\" 
       type=\"Float64\">\n";
    out<<world.ef;
    out<<"</DataArray>\n";

    /*Coulombic force, 3 component vector*/
    out << "<DataArray Name=\"Cf." << sp.name << "\" NumberOfComponents=
      \"3\" format=\"ascii\" type=\"Float64\">\n";
    out << sp.Cf;
    out << "</DataArray>\n";

    /*close out tags*/
    out<<"</PointData>\n";

    out<<"</ImageData>\n";
    out<<"</VTKFile>\n";
    out.close();

    /*clear samples if not at steady state*/
    if (!world.isSteadyState())
        sp.clearSamples();
}

/*saves particle data*/
void Output::particles(World &world, Droplets &sp) {
    /*loop over all species*/

    //open a phase_sp_it.vtp
    stringstream name;
    name<<"results/parts_"<<sp.name<<"_"<<setfill('0')<<setw(5)
      <<world.getTs()<<".vtp";

    /*open output file*/
    ofstream out(name.str());
    if (!out.is_open()) {cerr<<"Could not open "<<name.str()
      <<endl;return;}

    /*build a list of particles to output*/
    // here just outputting all but leaving this legacy code
    vector<Particle*> to_output;
    for (Particle &part : sp.particles) {
        to_output.emplace_back(&part);
    }

    /*header*/
    out<<"<?xml version=\"1.0\"?>\n";
    out<<"<VTKFile type=\"PolyData\" version=\"0.1\" byte_order=
      \"LittleEndian\">\n";
    out<<"<PolyData>\n";
    out<<"<Piece NumberOfPoints=\""<<to_output.size()<<"\" NumberOfVerts=
      \"0\" NumberOfLines=\"0\" ";
    out<<"NumberOfStrips=\"0\" NumberOfCells=\"0\">\n";

    /*points*/
    out<<"<Points>\n";
    out<<"<DataArray type=\"Float64\" NumberOfComponents=\"3\" format=
    \"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->pos<<"\n";
    out<<"</DataArray>\n";
    out<<"</Points>\n";

    /*data*/
    out<<"<PointData>\n";
    /*velocities*/
    out<<"<DataArray Name=\"vel\" type=\"Float64\" NumberOfComponents=\"3
      \" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->vel<<"\n";
    out<<"</DataArray>\n";

    /*accelerations*/
    out<<"<DataArray Name=\"acc\" type=\"Float64\" NumberOfComponents=\"3
      \" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->acc<<"\n";
    out<<"</DataArray>\n";

    /*Electric field forces*/
    out<<"<DataArray Name=\"Eforce\" type=\"Float64\" NumberOfComponents=
      \"3\" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->Eforce<<"\n";
    out<<"</DataArray>\n";

    /*Coulombic forces*/
    out<<"<DataArray Name=\"Cforce\" type=\"Float64\" NumberOfComponents=
      \"3\" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->Cforce<<"\n";
    out<<"</DataArray>\n";

    /*Drag forces*/
    out << "<DataArray Name=\"Dforce\" type=\"Float64\"
      NumberOfComponents=\"3\" format=\"ascii\">\n";
    for (Particle* part : to_output)
        out << part->Dforce << "\n";
    out << "</DataArray>\n";

    /*Total forces*/
    out<<"<DataArray Name=\"Force\" type=\"Float64\" NumberOfComponents=
      \"3\" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->force<<"\n";
    out<<"</DataArray>\n";

    /*Force ratios F_C/ F_E*/
    out << "<DataArray Name=\"frat\" type=\"Float64\" NumberOfComponents=\"3\" format=\"ascii\">\n";
    for (Particle* part : to_output)
        out << part->frat << "\n";
    out << "</DataArray>\n";

    /*r*/
    out<<"<DataArray Name=\"r\" type=\"Float64\" NumberOfComponents=\"1\" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->r<<"\n";
    out<<"</DataArray>\n";

    /*charge*/
    out<<"<DataArray Name=\"charge\" type=\"Float64\" NumberOfComponents=\"1\" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->charge<<"\n";
    out<<"</DataArray>\n";

    /*mass*/
    out<<"<DataArray Name=\"mass\" type=\"Float64\" NumberOfComponents=\"1\" format=\"ascii\">\n";
    for (Particle *part: to_output)
        out<<part->mass<<"\n";
    out<<"</DataArray>\n";

    /*radius f0r 180 degree self-scatter*/
    out << "<DataArray Name=\"rss\" type=\"Float64\" NumberOfComponents=\"1\" format=\"ascii\">\n";
    for (Particle* part : to_output)
        out << part->rss << "\n";
    out << "</DataArray>\n";

    /*nearest-neighbor separation*/
    out << "<DataArray Name=\"rsep\" type=\"Float64\" NumberOfComponents=\"1\" format=\"ascii\">\n";
    for (Particle* part : to_output)
        out << part->rsep << "\n";
    out << "</DataArray>\n";

    /*plume angle*/
    out << "<DataArray Name=\"angle\" type=\"Float64\" NumberOfComponents=\"1\" format=\"ascii\">\n";
    for (Particle* part : to_output)
        out << part->angle << "\n";
    out << "</DataArray>\n";

    /*close out tags*/
    out<<"</PointData>\n";

    out<<"</Piece>\n";
    out<<"</PolyData>\n";
    out<<"</VTKFile>\n";

    out.close();
}

```

```
/*Field is a container for mesh node data division by volume*/
#ifndef _FIELD_H
#define _FIELD_H

#include <ostream>

template <typename T>
struct vec3 {
    vec3 (const T u, const T v, const T w) : d{u,v,w} {}
    vec3 (const T a[3]) : d{a[0],a[1],a[2]} {}
    vec3 (): d{0,0,0} {}
    T& operator[](int i) {return d[i];}
    T operator[](int i) const {return d[i];}
    vec3<T>& operator=(double s) {d[0]=s;d[1]=s;d[2]=s;return (*this);}
    vec3<T>& operator+=(vec3<T> o) {d[0]+=o[0];d[1]+=o[1];d[2]+=o
      [2];return(*this);}
    vec3<T>& operator-=(vec3<T> o) {d[0]-=o[0];d[1]-=o[1];d[2]-=o
      [2];return(*this);}
    vec3<T> operator/(double s) {vec3<T>o; o[0]=d[0]/s;o[1]=d[1]/s;o[2]=d
      [2]/s;return o;}
    vec3<T> operator/=(double s) {d[0]/=s;d[1]/=s;d[2]/=s;return (*this);}

    //dot product of two vectors
    friend T dot(const vec3<T> &v1, const vec3<T> &v2) {
        T s=0;  for (int i=0;i<3;i++) s+=v1[i]*v2[i];
        return s;   }

    //vector magnitude
    friend T mag(const vec3<T> &v) {return sqrt(dot(v,v));}

    //unit vector
    friend vec3<T> unit(const vec3<T> &v) {return vec3(v)/mag(v);}

    //cross product
    friend vec3<T> cross(const vec3<T> &a, const vec3<T> &b) {
        return {a[1]*b[2]-a[2]*b[1], a[2]*b[0]-a[0]*b[2], a[0]*b[1]-a[1]*b
          [0]};
    }

protected:
    T d[3];
};

//vec3-vec3 operations
template<typename T>    //addition of two vec3s
vec3<T> operator+(const vec3<T>& a, const vec3<T>& b) {
    return vec3<T> (a[0]+b[0],a[1]+b[1],a[2]+b[2]); }
template<typename T>    //subtraction of two vec3s
vec3<T> operator-(const vec3<T>& a, const vec3<T>& b) {
    return vec3<T> (a[0]-b[0],a[1]-b[1],a[2]-b[2]); }
template<typename T>    //element-wise multiplication of two vec3s
vec3<T> operator*(const vec3<T>& a, const vec3<T>& b) {
    return vec3<T> (a[0]*b[0],a[1]*b[1],a[2]*b[2]); }
template<typename T>    //element wise division of two vec3s
vec3<T> operator/(const vec3<T>& a, const vec3<T>& b) {
    return vec3<T> (a[0]/b[0],a[1]/b[1],a[2]/b[2]); }

//vec3 - scalar operations
template<typename T>        //scalar multiplication
vec3<T> operator*(const vec3<T> &a, T s) {
    return vec3<T>(a[0]*s, a[1]*s, a[2]*s);}
template<typename T>        //scalar multiplication 2
vec3<T> operator*(T s,const vec3<T> &a) {
    return vec3<T>(a[0]*s, a[1]*s, a[2]*s);}

//output
template<typename T>    //ostream output
std::ostream& operator<<(std::ostream &out, vec3<T>& v) {
    out<<v[0]<<" "<<v[1]<<" "<<v[2];
    return out;
}

using double3 = vec3<double>;
using int3 = vec3<int>;

template <typename T>
class Field_
{
public:

    /*constructor*/
    Field_(int ni, int nj, int nk) :
    ni{ni}, nj{nj}, nk{nk}
    {
        //allocate memory for a 3D array
        data = new T**[ni];
        for (int i=0;i<ni;i++)
        {
            data[i] = new T*[nj];
            for (int j=0;j<nj;j++) data[i][j] = new T[nk];
        }

        clear();
    }

    //another constructor taking an int3
    Field_(int3 nn) : Field_(nn[0],nn[1],nn[2]) {};

    //copy constructor
    Field_(const Field_ &other):
    Field_{other.ni,other.nj,other.nk} {
        for (int i=0;i<ni;i++)
            for (int j=0;j<nj;j++)
                for (int k=0;k<nk;k++)
                    data[i][j][k] = other(i,j,k);
    }

    //move constructor
    Field_(Field_ &&other):
        ni{other.ni},nj{other.nj},nk{other.nk} {
            data = other.data;  //steal the data
            other.data = nullptr;   //invalidate
    }

    //move assignment operator
    Field_& operator = (Field_ &&f) {data=f.data;
                f.data=nullptr; return *this;}

    //destructor: release memory
    ~Field_() {
        //don't do anything if data is not allocated (or was moved away)
        if (data==nullptr) return;

        for (int i=0;i<ni;i++)
        {
            for (int j=0;j<nj;j++)
                delete[] data[i][j];

            delete[] data[i];
        }

        delete[] data;
    }

    //overloaded operator [] to allow direct access to data
    T** operator[] (int i) {return data[i];}

    /*returns data[i][j][k] marked as const to signal no data change*/
    T operator() (int i, int j, int k) const {return data[i][j][k];}

    /*sets all values to some scalar*/
    void operator =(double s) {
        for (int i=0;i<ni;i++)
          for (int j=0;j<nj;j++)
           for (int k=0;k<nk;k++)
            data[i][j][k] = s;
      }

    /*performs element by element division by another field*/
    void operator / (const Field_ &other) {
        for (int i=0;i<ni;i++)
            for (int j=0;j<nj;j++)
                for (int k=0;k<nk;k++) {
                    if (other.data[i][j][k]!=0)
                      data[i][j][k] /= other.data[i][j][k];
                else
                      data[i][j][k] = 0;
               }
    }

    /*increments values by data from another field*/
    Field_& operator += (const Field_ &other) {
        for (int i=0;i<ni;i++)
            for (int j=0;j<nj;j++)
                for (int k=0;k<nk;k++)
                    data[i][j][k]+=other(i,j,k);
        return (*this);
    }

    /*performs element by element multiplication by a double*/
    Field_& operator *= (double s) {
        for (int i=0;i<ni;i++)
            for (int j=0;j<nj;j++)
                for (int k=0;k<nk;k++)
                    data[i][j][k]*=s;
        return (*this);
    }

    /*performs element by element division by a double*/
    Field_& operator /= (double s) {
        for (int i = 0; i < ni; i++)
            for (int j = 0; j < nj; j++)
                for (int k = 0; k < nk; k++)
                    if (s != 0)
                        data[i][j][k] /= s;
                    else
                        data[i][j][k] = 0;
        return (*this);
    }

    //multiplication operator, returns f*s
    friend Field_<T> operator*(double s, const Field_<T>&f) {
        Field_<T> r(f);
        return r*=s;
    }

    //multiplication of a field by a field of doubles
    friend Field_<T> operator*(const Field_<T>&f1, const Field_<T>&f2) {
        Field_<T> r(f1);
        for (int i=0;i<f1.ni;i++)
            for (int j=0;j<f1.nj;j++)
                for (int k=0;k<f1.nk;k++)
                    r[i][j][k] = f1(i,j,k)*f2(i,j,k);
        return r;
    }

    //division of a field by a field of doubles
    friend Field_<T> operator/(const Field_<T>&f, const Field_<double>&d)
      {
        Field_<T> r(f);
        for (int i=0;i<f.ni;i++)
            for (int j=0;j<f.nj;j++)
                for (int k=0;k<f.nk;k++)
                {
                    if (d(i,j,k)!=0)    //check for div by zero
                        r[i][j][k] = f(i,j,k)/d(i,j,k);
                    else
                        r[i][j][k] = 0;
                }
        return r;
    }

    /*returns index for node (i,j,k)*/
    int U(int i, int j, int k) {return k*ni*nj+j*ni+i;}

    /*sets all data to zero*/
    void clear() {(*this)=0;}

    /* scatters scalar value onto a field at logical coordinate lc*/
    void scatter(double3 lc, T value)
    {
        int i = (int)lc[0];
        double di = lc[0]-i;

        int j = (int)lc[1];
        double dj = lc[1]-j;

        int k = (int)lc[2];
        double dk = lc[2]-k;

        data[i][j][k] += (T)value*(1-di)*(1-dj)*(1-dk);
        data[i+1][j][k] += (T)value*(di)*(1-dj)*(1-dk);
        data[i+1][j+1][k] += (T)value*(di)*(dj)*(1-dk);
        data[i][j+1][k] += (T)value*(1-di)*(dj)*(1-dk);
        data[i][j][k+1] += (T)value*(1-di)*(1-dj)*(dk);
        data[i+1][j][k+1] += (T)value*(di)*(1-dj)*(dk);
        data[i+1][j+1][k+1] += (T)value*(di)*(dj)*(dk);
        data[i][j+1][k+1] += (T)value*(1-di)*(dj)*(dk);
    }

    /* gathers field value at logical coordinate lc*/
    T gather(double3 lc)
    {
        int i = (int)lc[0];
        double di = lc[0]-i;

        int j = (int)lc[1];
        double dj = lc[1]-j;

        int k = (int)lc[2];
        double dk = lc[2]-k;

        /*gather electric field onto particle position*/
        T val = data[i][j][k]*(1-di)*(1-dj)*(1-dk)+
                data[i+1][j][k]*(di)*(1-dj)*(1-dk)+
                data[i+1][j+1][k]*(di)*(dj)*(1-dk)+
                data[i][j+1][k]*(1-di)*(dj)*(1-dk)+
                data[i][j][k+1]*(1-di)*(1-dj)*(dk)+
                data[i+1][j][k+1]*(di)*(1-dj)*(dk)+
                data[i+1][j+1][k+1]*(di)*(dj)*(dk)+
                data[i][j+1][k+1]*(1-di)*(dj)*(dk);

        return val;
    }

    //incorporates new instantaneous values into a running average
    void updateAverage(const Field_ &I) {
        for (int i=0;i<ni;i++)
            for (int j=0;j<nj;j++)
                for (int k=0;k<nk;k++)
                    data[i][j][k] = (I(i,j,k)+ave_samples*data[i][j][k])/
                  (ave_samples+1);
        ++ave_samples;  //increment number of samples
    }

    template<typename S>
    friend std::ostream& operator<<(std::ostream &out, Field_<S> &f);
    const int ni,nj,nk; //allocated dimensions

protected:
    T ***data;  /*data held by this field*/
    int ave_samples = 0;    //number of samples used for averaging
};
```

```
/*writes out data to a file stream*/
template<typename T>
std::ostream& operator<<(std::ostream &out, Field_<T> &f)
{
    for (int k=0;k<f.nk;k++,out<<"\n")
        for (int j=0;j<f.nj;j++)
            for (int i=0;i<f.ni;i++) out<<f.data[i][j][k]<<" ";
    return out;
}

//some typedefs
using Field = Field_<double>;
using FieldI = Field_<int>;
using Field3 = Field_<double3>;
using dvector = std::vector<double>;

#endif

```

REFERENCES

[1] Bowman A 2024 State-of-the-art of small spacecraft technology state of the art small spacecraft technology report

[2] Kulu E 2024 Running total

[3] Thuppul A, Wright P L, Collins A L, Ziemer J K and Wirz R E 2020 *Aerospace* 7 ISSN 2226-4310 URL https://www.mdpi.com/2226-4310/7/8/108

[4] Rosell-Llompart J, Grifoll J and Loscertales I G 2018 *Journal of Aerosol Science* 125 2–31 ISSN 0021-8502 from Electro-Hydro-Dynamics of liquids for the production of charged droplets by Electro-Spray to applications for tailored Materials (aerosols, powders, coatings) and Environment URL https://www.sciencedirect.com/science/article/pii/S0021850217304366

[5] Zeleny J 1917 *Physical Review* 10 1–6

[6] Wright P L, Thuppul A and Wirz R E 2018 Life-limiting emission modes for electrospray thrusters *Joint Propulsion Conference, AIAA* pp AIAA 2018–4726

[7] Lisa pathfinder overview accessed May 2024

[8] Lisa-pathfinder accessed May 2024

[9] Gañán-Calvo A, Lasheras J, Dávila J and Barrero A 1994 *Journal of Aerosol Science* 25 1121–1142 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/0021850294902054

[10] Wilhelm O, Mädler L and Pratsinis S 2003 *Journal of Aerosol Science* 34 815–836 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S002185020300034X

[11] Grifoll J and Rosell-Llompart J 2012 *Journal of Aerosol Science* 47 78–93 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850212000109

[12] Thuppul A, Wright P L and Wirz R E 2018 Lifetime considerations and estimation for electrospray thrusters *Joint Propulsion Conference, AIAA* pp AIAA 2018–4652

[13] Gamero-Castaño M and Galobardes-Esteban M 2022 *Journal of Applied Physics* 131 013307 (*Preprint* https://doi.org/10.1063/5.0073380) URL https://doi.org/10.1063/5.0073380

[14] Petro E M, Gallud X, Hampl S K, Schroeder M, Geiger C and Lozano P C 2022 *Journal of Applied Physics* 131 193301 (*Preprint* https://doi.org/10.1063/5.0065615) URL https://doi.org/10.1063/5.0065615

[15] Ninomiya S, Sakai Y, Chen L C and Hiraoka K 2014 *Journal of Surface Analysis* 20 171–176

[16] Loth E 2008 *AIAA Journal* 46 2219–2228 (*Preprint* https://doi.org/10.2514/1.28943) URL https://doi.org/10.2514/1.28943

[17] Stokes G G 2009 *On the Effect of the Internal Friction of Fluids on the Motion of Pendulums* (*Cambridge Library Collection - Mathematics* vol 3) (Cambridge University Press)

[18] Moshfegh A, Shams M, Ahmadi G and Ebrahimi R 2010 *Journal of Aerosol Science* 41 384–400 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850210000200

[19] Niazmand H and Anbarsooz M 2012 *Journal of Mechanical Science and Technology* 26 2741–2749

[20] Tao S, Zhang H and Guo Z 2017 *Journal of Aerosol Science* 103 105–116 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850216301884

[21] Tsao H K, Sheng Y J and Chen S B 2002 *Physical review. E, Statistical, nonlinear, and soft matter physics*

[22] Braun H 2008 Emittance diagnostics

[23] Thuppul A, Collins A L, Wright P L, Uchizono N M and Wirz R E 2021 *Journal of Applied Physics* 130 103301 (*Preprint* https://doi.org/10.1063/5.0056761) URL https://doi.org/10.1063/5.0056761

[24] Miller S, Ulibarri-Sanchez J, Prince B and Bemish R 2021 *Journal of Fluid Mechanics* 928 A12

[25] Fenn J B, Matthias M, Kai M C, Fu W S and Whitehouse C M 1989 *Science* 246 64–71

[26] Arumugham-Achari A K, Grifoll J and Rosell-Llompart J 2014 Numerical simulations of evaporating electrosprays with coulomb explosions *Aerosol Technologies*

[27] Gamero-Castaño M and de la Mora J F 2002 *Physical Review Letters* 89 147602

[28] FRS L R 1882 *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 14 184–186 (*Preprint* https://doi.org/10.1080/14786448208628425) URL https://doi.org/10.1080/14786448208628425

[29] Taylor G I 1964 *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 280 383–397

[30] Che F, Lin L, Zhang J, He Z, Uchiyama K and Lin J M 2016 *Analytical Chemistry* 88 4354–4360 pMID: 27015013 (*Preprint* https://doi.org/10.1021/acs.analchem.5b04749) URL https://doi.org/10.1021/acs.analchem.5b04749

[31] LibreTexts 2021 Applications of electrostatics URL

[32] qun Huang C, Jian G, Delisio J B, Wang H and Zachariah M R 2015 *Advanced Engineering Materials* 17

[33] Taylor A P and Velásquez-García L F 2015 *Nanotechnology* 26 505301

[34] Steipel R T, Gallovic M D, Batty C J, Bachelder E M and Ainslie K M 2019 *Materials Science & Engineering: C, Materials for biological applications* 105 1110070

[35] Anderson E, Carlucci A P, Risi A and Kyritsis D 2007 *International Journal of Vehicle Design* 45 61–79

[36] Law S E 2001 *Journal of Electrostatics* 51-52 25–42 ISSN 0304-3886 electrostatics 2001: 9th International Conference on Electrostatics URL https://www.sciencedirect.com/science/article/pii/S0304388601000407

[37] Jason Deveau S 2018 Electrostatic spraying in agriculture URL https://sprayers101.com/electrostatic/

[38] of America S B S 2021 Electrostatic spraying URL https://www.spectrumbsa.com/service/electrostatic-spraying/?ref=semgclid=CjwKCAjwruSHBhAtEiwA$_qCppl0BoQlpTOlY4H7LsrYl1fdOzToRLZ3k$

[39] Jaworek A, Balachandran W, Lackowski M, Kulon J and Krupa A 2006 *Journal of Electrostatics* 64 194–202 ISSN 0304-3886 URL https://www.sciencedirect.com/science/article/pii/S0304388605001580

[40] Cadnum J L, Jencson A L, Livingston S H, Li D F, Redmond S N, Pearlmutter B, Wilson B M and Donskey C J 2020 *American Journal of Infection Control* 951–954

[41] Hanlon J 2020 Fire department awarded grant for electrostatic sprayers URL https://oxfordleader.com/fire-department-awarded-grant-for-electrostatic-sprayers/

[42] Kinsey J and Pendleton F J 1985 Evaluation of charged fog for smoke clearing shipboard fires defense Technical Information Center, Accession number: ADA165551

[43] Okuda H and Kelly A J 1996 *Physics of Plasmas* 2191

[44] Ziemer J, Marrese-Reading C, Dunn C, Romero-Wolf A, Cutler C, Javidnia S, Le T, Vi I, Franklin G and Barela P 2017 Colloid microthruster flight performance results from space technology 7 disturbance reduction system *The 35th International Electric Propulsion Conference* p 20170010216

[45] Collins A L, Wright P L, Uchizono N M and Wirz R E 2022 *Journal of Electric Propulsion* 1 32 URL https://doi.org/10.1007/s44205-022-00031-w

[46] Breddan M J and Wirz R E 2023 *Journal of Aerosol Science* 167 106079 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S002185022200115X

[47] Tang H B, Qin C J and Liu Y 2011 *Journal of Aerosol Science* 42 114–126 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850210002387

[48] no M G 2008 *Journal of Fluid Mechanics* 604 339–368

[49] Deng W, Klemic J F, Li X, Reed M A and Gomez A 2006 *Journal of Aerosol Science* 37 696–714 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850205000959

[50] Yang W, Lojewski B, Wei Y and Deng W 2012 *Journal of Aerosol Science* 46 20–33 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850211001832

[51] Oh H, Kim K and Kim S 2008 *Journal of Aerosol Science* 801–813

[52] Jung J H, Oh H and Kim S 2010 *Powder Technology* 439–444

[53] Grifoll J and Rosell-Llompart J 2014 *Journal of Electrostatics* 72 357–364 ISSN 0304-3886 URL https://www.sciencedirect.com/science/article/pii/S0304388614000564

[54] Arumugham-Achari A K, Grifoll J and Rosell-Llompart J 2015 *Aerosol Science and Technology* 49 436–448 (*Preprint* https://doi.org/10.1080/02786826.2015.1039639) URL https://doi.org/10.1080/02786826.2015.1039639

[55] Grifoll J, Arumugham-Achari A K and Rosell-Llompart J 2011 Numerical simulation of electrospray droplets dynamics *V Reunion Esponola de Ciecia y Tecnologia de Aerosoles (RECTA)*

[56] Higuera F J 2013 *Journal of Fluid Mechanics* 734 363–386

[57] Cui C and Wang J 2020 Simulations of pure ionic electrospray thruster plume neutralization *AIAA Propulsion and Energy Forum*

[58] Rietveld I, Kobayashi K, Yamada H and Matsushige K 2006 *Journal of Physical Chemistry B* 23351–23364

[59] Hartman R, Borra J P, Brunner D, Marijnissen J and Scarlett B 1999 *Journal of Electrostatics* 47 143–170 ISSN 0304-3886 URL https://www.sciencedirect.com/science/article/pii/S0304388699000340

[60] Huh H and Wirz R E 2022 *Physics of Fluids* 34 112017 (*Preprint* https://doi.org/10.1063/5.0120737) URL https://doi.org/10.1063/5.0120737

[61] Huh H 2023 *Cone-Jet and Emission Behavior for Electrospray Thrusters via Computational Analysis* Ph.D. thesis University of California, Los Angeles

[62] Enomoto T, Parmar S M, Yamada R, Wirz R E and Takao Y 2022 *Journal of Electric Propulsion* 1 13

[63] Parmar S M, Collins A L and Wirz R E 2022 Electrospray plume modeling for rapid life and performance analysis *AIAA Science and Technology Forum and Exposition* pp AIAA 2022–1357

[64] Parmar S M, Collins A L and Wirz R E 2022 A bayesian data-driven model for quantifying electrospray lifetime *7th International Electric Propulsion Conference* pp IEPC–2022–230

[65] Uchizono N M, Collins A L, Marrese-Reading C, Arestie S M, Ziemer J K and Wirz R E 2021 *Journal of Applied Physics* 130 143301 (*Preprint* https://doi.org/10.1063/5.0063476) URL https://doi.org/10.1063/5.0063476

[66] Uchizono N 2023 *Secondary Species Emission and Behavior for Electrospray Thrusters* Ph.D. thesis University of California, Los Angeles

[67] Magnusson J M, Collins A L and Wirz R E 2020 *Aerospace* 7 ISSN 2226-4310 URL https://www.mdpi.com/2226-4310/7/11/153

[68] Davis M J, Collins A L and Wirz R E 2019 Electrospray plume evolution via discrete simulation *The 36th International Electric Propulsion Conference* pp IEPC–2019–590

[69] FRS L R 1879 *Proceedings of the London Mathematical Society* s1-11 57–72 (*Preprint* https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s1-11.1.57) URL https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s1-11.1.57

[70] Rosell-Llompart J and de la Mora J F 1994 *Journal of Aerosol Science* 25 1093–1119

[71] Cloupeau M and Prunet-Foch B 1989 *Journal of Electrostatics* 22 135–159 ISSN 0304-3886 URL https://www.sciencedirect.com/science/article/pii/0304388689900818

[72] Tang K and Gomez A 1994 *Physics of Fluids* 6 2317–2332 (*Preprint* https://doi.org/10.1063/1.868182) URL https://doi.org/10.1063/1.868182

[73] no M G C and Cisquella-Serra A 2021 *Phys. Rev. Fluids* 6(1) 013701 URL https://link.aps.org/doi/10.1103/PhysRevFluids.6.013701

[74] Miller S, Ulibarri-Sanchez J, Prince B and Bemish R 2021 *Journal of Fluid Mechanics* 928 A12

[75] Gamero-Castaño M 2002 *Phys. Rev. Lett.* 89(14) 147602 URL https://link.aps.org/doi/10.1103/PhysRevLett.89.147602

[76] Feng Z G 2010 *Journal of Dispersion Science and Technology* 31 968–974 (*Preprint* https://doi.org/10.1080/01932690903224110) URL https://doi.org/10.1080/01932690903224110

[77] Tao S, Zhang H and Guo Z 2017 *Journal of Aerosol Science* 103 105–116 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850216301884

[78] Singh N and Schwartzentruber T E 2016 Heat flux and drag correlations for high speed flight at any knudsen number *46th AIAA Thermophysics Conference* pp AIAA 2016–3841 (*Preprint* https://arc.aiaa.org/doi/pdf/10.2514/6.2016-3841) URL https://arc.aiaa.org/doi/abs/10.2514/6.2016-3841

[79] Banerjee S, Levy M, Davis M and Wilkerson B 2017 *IEEE Transactions on Industry Applications* 53 2455–2460

[80] Gamero-Castaño M and de la Mora J F 2000 *The Journal of Chemical Physics* 113 815–832 (*Preprint* https://doi.org/10.1063/1.481857) URL https://doi.org/10.1063/1.481857

[81] Gomez A and Tang K 1994 *Physics of Fluids* 6 404–414 (*Preprint* https://doi.org/10.1063/1.868037) URL https://doi.org/10.1063/1.868037

[82] Villanueva-Bonay E and Gamero-Castaño M 2019 *AIP Advances* 9 085204 (*Preprint* https://doi.org/10.1063/1.5100964) URL https://doi.org/10.1063/1.5100964

[83] Mehta N A and Levin D A 2019 *Phys. Rev. E* 99(3) 033302 URL https://link.aps.org/doi/10.1103/PhysRevE.99.033302

[84] Swope W C, Andersen H C, Berens P H and Wilson K R 1982 *The Journal of Chemical Physics* 76 637–649 (*Preprint* https://doi.org/10.1063/1.442716) URL https://doi.org/10.1063/1.442716

[85] Uchizono N M, Collins A L, Thuppul A, Wright P L, Eckhardt D Q, Ziemer J and Wirz R E 2020 *Aerospace* 7 ISSN 2226-4310 URL https://www.mdpi.com/2226-4310/7/10/141

[86] Clift R, Grace J and Weber M E 1978 *Bubbles, Drops and Particles* Dover Civil and Mechanical Engineering Series (Dover Publications, Incorporated) ISBN 048678892X, 9780486788920

[87] Wang Y, Chen Z, Shima K, Zhong D, Yang L, Wang Q, Jiang R, Dong J, Lei Y, Li X and Cao L 2022 *Journal of Mass Spectrometry* 57 e4831

[88] Sorensen A H 1986 *Conf. Proc. C* 860915 135 – 152

[89] Sørensen A 1989 URL https://cds.cern.ch/record/367259

[90] Wirz R E, Collins A L, Chen Z, Huerta C E, Li G Z, Samples S A, Thuppul A, Wright P L, Uchizono N M, Huh H, Davis M J and Ottaviano A 2019 Electric propulsion activities at the ucla plasma & space propulsion laboratory *The 36th International Electric Propulsion Conference* pp IEPC–2019–547

[91] Breddan M J and Wirz R E 2024 *Engineering Applications of Artificial Intelligence* 133 108095 ISSN 0952-1976 URL https://www.sciencedirect.com/science/article/pii/S0952197624002537

[92] Cretel C M and Wirz R E 2024 Ion thruster grid life and performance prediction via reduced order modeling *38th International Electric Propulsion Conference* pp IEPC–2024–758

[93] Ehsan Taghizadeh Richard A Obenchain L K F and Wirz R 2024 Electric propulsion facility optimization via reduced order modeling *38th International Electric Propulsion Conference* pp IEPC–2024–558

[94] Feng Z G, Michaelides E E and Mao S 2012 *Fluid Dynamics Research* 44 025502

[95] BAILEY A B and HIATT J 1972 *AIAA Journal* 10 1436–1440 (*Preprint* https://doi.org/10.2514/3.50387) URL https://doi.org/10.2514/3.50387

[96] Abraham F F 1970 *The Physics of Fluids* 13 2194–2195 (*Preprint* https://aip.scitation.org/doi/pdf/10.1063/1.1693218) URL https://aip.scitation.org/doi/abs/10.1063/1.1693218

[97] Singh N, Kroells M, Li C, Ching E, Ihme M, Hogan C J and Schwartzentruber T E 2022 *AIAA Journal* 60 587–597 (*Preprint* https://doi.org/10.2514/1.J060648) URL https://doi.org/10.2514/1.J060648

[98] no M G C and Hruby V 2002 *Journal of Fluid Mechanics* 459 245–276

[99] Jaworek A, Sobczyk A, Czech T and Krupa A 2014 *Journal of Electrostatics* 72 166–178 ISSN 0304-3886 URL https://www.sciencedirect.com/science/article/pii/S0304388614000114

[100] Kidd P W and Shelton H 1973 Life test (4350 hours) of an advanced colloid thruster module *AIAA 10th Electric Propulsion Conference* pp AA Paper 73–1078 (*Preprint* https://arc.aiaa.org/doi/pdf/10.2514/6.1973-1078) URL https://arc.aiaa.org/doi/abs/10.2514/6.1973-1078

[101] Prince B D, Fritz B A and Chiu Y H 2012 *Ionic Liquids in Electrospray Propulsion Systems* (American Chemical Society) chap 2, pp 27–49 ISBN 9780841227637

[102] Liu V, Pang S and Jew H 1965 *Physics of Fluids* 8 788–796

[103] Singh N and Schwartzentruber T E 2016 Heat flux and drag correlations for high speed flight at any knudsen number *46th AIAA Thermophysics Conference* p 55414 (*Preprint* https://arc.aiaa.org/doi/pdf/10.2514/6.2016-3841) URL https://arc.aiaa.org/doi/abs/10.2514/6.2016-3841

[104] Rapp B E 2017 Chapter 9 - fluids *Microfluidics: Modelling, Mechanics and Mathematics* Micro and Nano Technologies ed Rapp B E (Oxford: Elsevier) pp 243–263 ISBN 978-1-4557-3141-1 URL https://www.sciencedirect.com/science/article/pii/B9781455731411500095

[105] Karniadakis G, Beskok A and NR A 2005 *MicroFlows and Nanoflows - Fundamentals and Simulation* (Springer Science+Business Media)

[106] Shaoxian B and Shizhu W 2019 Chapter 3 - isothermal gas lubrication *Gas Thermohydrodynamic Lubrication and Seals* ed Shaoxian B and Shizhu W (Academic Press) pp 37–71 ISBN 978-0-12-816716-8 URL https://www.sciencedirect.com/science/article/pii/B9780128167168000036

[107] Duan Z and Ma H 2020 Chapter four - pressure drop and heat transfer in the entrance region of microchannels *Advances in Heat Transfer* vol 52 ed Abraham J, Gorman J and Minkowycz W (Elsevier) pp 249–333 URL https://www.sciencedirect.com/science/article/pii/S0065271720300022

[108] Breddan M J D, Curry D R, Sharma M R, Richmond M O, Collins A L and Wirz R E 2022 Electrospray plume modeling: Study on drag influence *AIAA Science and Technology Forum and Exposition* pp AIAA–2022–1358

[109] Ouyang H, Larriba-Andaluz C, Oberreit D R and Hogan C J J 2013 *Journal of the American Society for Mass Spectrometry* 24 1833–1847

[110] Larriba C and Hogan C J J 2013 *The Journal of Physical Chemistry A* 117 3887–3901

[111] Fernández-García J and Fernández de la Mora J 2013 *Journal of the American Society for Mass Spectrometry* 24 1872–1889

[112] Peach K, Wilson P and Jones B 2011 *The British Journal of Radiology* 84 Spec No 1 S4–S10

[113] Beacham J, Burrage C, Curtin D, Roeck A D, Evans J, Feng J L, Gatto C, Gninenko S, Hartin A, Irastorza I, Jaeckel J, Jungmann K, Kirch K, Kling F, Knapen S, Lamont M, Lanfranchi G, Lazzeroni C, Lindner A, Martinez-Vidal1 F, Moulson M, Neri N, Papucci M, Pedraza I, Petridis K, Pospelov M, Rozanov A, Ruoso G, Schuster P, Semertzidis Y, Spadaro T, Vallée C and Wilkinson G 2019 *Journal of Physics G: Nuclear and Particle Physics* 47 010501

[114] Dolovich M B and Dhand R 2010 *Lancet* 377 1032–1045

[115] Tang K and Gomez A 1994 *Journal of Aerosol Science* 25 1237–1249 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/0021850294902127

[116] Narayan K and Subramaniam S 2015 *Nature Methods* 12 1021–1031

[117] Ali M, Hung W and Yongqi F 2010 *International Journal of Precisions Engineering and Manufacturing* I 11 157–170

[118] Konopliv M F, Chaplin V H, Johnson L K and Wirz R E 2023 *Plasma Sources Science and Technology* 32 015009 URL https://dx.doi.org/10.1088/1361-6595/acb00b

[119] Walker M L R, Victor A L, Hofer R R and Gallimore A D 2005 *Journal of Propulsion and Power* 21 408–415 URL https://doi.org/10.2514/1.7713

[120] Boyd I D 2001 *Journal of Spacecraft and Rockets* 38 381–387

[121] Jacobson D, John J, Manzella D and Peterson P 2012 An overview of hall thruster development at nasa's john h. glenn research center *41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference &amp; Exhibit* pp 2005–4242 (*Preprint* https://arc.aiaa.org/doi/pdf/10.2514/6.2005-4242) URL https://arc.aiaa.org/doi/abs/10.2514/6.2005-4242

[122] Collins A L, Uchizono N M, Huh H and Wirz R E 2022 Three-dimensional microscopy and analysis of the emission cone meniscus for electrospray thrusters *37th International Electric Propulsion Conference* pp IEPC–2022–228

[123] Yang S, Wang Z, Kong Q and Li B 2022 *International Journal of Multiphase Flow* 146 103851 ISSN 0301-9322 URL https://www.sciencedirect.com/science/article/pii/S0301932221002731

[124] M A Gallis D J R and Torczynski J R 2004 *Aerosol Science and Technology* 38 692–706

[125] Magnani M and Gamero-Castaño M 2024 *Journal of Fluid Mechanics*

[126] Tang K and Smith R D 2001 *Journal of the American Society for Mass Spectrometry* 343–347

[127] Taflin D C, Ward T L and Davis E J 1989 *Langmuir* 376–384

[128] Davis E and Bridges A M 1994 *Journal of Aerosol Science* 1179–1199

[129] de la Mora J F 1996 *Journal of Colloid and Interface Science* 209–218

[130] Crandall P and Wirz R E 2024 Miniature rf gridded ion thruster for air-breathing ep *38th International Electric Propulsion Conference, Toulouse, France, IEPC-2024-259.*

[131] Konopliv M F, Johnson L K and Wirz R E 2024 Cathode species contributions to hall thruster plume dynamics *38th International Electric Propulsion Conference* pp IEPC–2024–488

[132] Cowan R W, Biswas S, Franz L K, Cretel C M, Obenchain R A and Wirz R E 2024 Hall thruster krypton sputtering effects on vacuum facility materials *38th International Electric Propulsion Conference* pp IEPC–2024–549

[133] Biswas S, Obenchain R A, Cowan R W and Wirz R E 2024 Review of pmi data for het-induced erosion of facility surfaces *38th International Electric Propulsion Conference* pp IEPC–2024–532

[134] Wenninger J 2015 *Comptes Rendus Physique* 16 347–355 ISSN 1631-0705 highlights of the LHC run 1 / Résultats marquants de la première période d'exploitation du GCH URL https://www.sciencedirect.com/science/article/pii/S1631070515000560

[135] Franz L and Wirz R E 2024 Xe-c scattering, implantation, and sputtering analysis for ep systems *38th International Electric Propulsion Conference* pp IEPC–2024–552

[136] Breddan M J D and Wirz R E 2024 Coulomb dominance in electrospray plume expansion *38th International Electric Propulsion Conference* pp IEPC–2024–514

[137] Wang F, Elbadawi M, Tsilova S L, Gaisford S, Basit A W and Parhizkar M 2022 *Materials & Design* 219 110735 ISSN 0264-1275 URL https://www.sciencedirect.com/science/article/pii/S0264127522003574

[138] Iwano T, Yoshimura K, Inoue S, Odate T, Ogata K, Funatsu S, Tanihata H, Kondo T, Ichikawa D and Takeda S 2020 *British Journal of Surgery* 107 632–635 ISSN 0007-1323

[139] Margulis K, Zhou Z, Fang Q, Sievers R E, Lee R J and Zare R N 2018 *Analytical Chemistry* 90 12198–12206 pMID: 30188683 (*Preprint* https://doi.org/10.1021/acs.analchem.8b03410) URL https://doi.org/10.1021/acs.analchem.8b03410

[140] Chung W Y, Correa E, Yoshimura K, Chang M C, Dennison A, Takeda S and Chang Y T 2020 *American Journal of Translational Research* 12 171–179 pMID: 32051746

[141] Goodacre R, York E V, Heald J K and Scott I M 2003 *Phytochemistry* 62 859–863 ISSN 0031-9422 plant Metabolomics URL https://www.sciencedirect.com/science/article/pii/S0031942202007185

[142] Zhou Z and Zare R N 2017 *Analytical Chemistry* 89 1369–1372 pMID: 28194988 (*Preprint* https://doi.org/10.1021/acs.analchem.6b04498) URL https://doi.org/10.1021/acs.analchem.6b04498

[143] Mayhew A W, Topping D O and Hamilton J F 2020 *ACS Omega* 5 9510–9516 pMID: 32363303 (*Preprint* https://doi.org/10.1021/acsomega.0c00732) URL https://doi.org/10.1021/acsomega.0c00732

[144] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M and Édouard Duchesnay 2011 *Journal of Machine Learning Research* 12 2825–2830 ISSN 1532-4435

[145] Chen T and Guestrin C 2016 Xgboost: A scalable tree boosting system *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '16 (New York, NY, USA: Association for Computing Machinery) p 785–794 ISBN 9781450342322 URL https://doi.org/10.1145/2939672.2939785

[146] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q and Liu T Y 2017 Lightgbm: A highly efficient gradient boosting decision tree *Advances in Neural Information Processing Systems* vol 30 ed Guyon I, Luxburg U V, Bengio S, Wallach H, Fergus R, Vishwanathan S and Garnett R (Curran Associates, Inc.) URL https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[147] Thuppul A, Collins A L, Wright P L, Uchizono N M and Wirz R E 2019 Spatially-resolved mass flux and current measurements of electrospray plumes *The 36th International Electric Propulsion Conference* pp IEPC–2019–571

[148] Yang W, Duan H, Li C and Deng W 2014 *Phys. Rev. Lett.* 112(5) 054501 URL https://link.aps.org/doi/10.1103/PhysRevLett.112.054501

[149] Ryan C, Smith K and Stark J 2012 *Journal of Aerosol Science* 51 35–48 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850212000651

[150] Duby M H, Deng W, Kim K, Gomez T and Gomez A 2006 *Journal of Aerosol Science* 37 306–322 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850205001175

[151] Hartman R, Brunner D, Camelot D, Marijnissen J and Scarlett B 2000 *Journal of Aerosol Science* 31 65–95 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850299000348

[152] Dastourani H, Jahannama M R and Eslami-Majd A 2018 *InternationalJournal of Heat and Fluid Flowing* 70 315–335

[153] Gamero-Castaño M and Magnani M 2019 *Journal of Fluid Mechanics* 859 247–267

[154] Huh H and Wirz R E 2019 Numerical simulation of electrospray thruster extraction *The 36th International Electric Propulsion Conference* pp IEPC–2019–565

[155] de la Mora J F and Loscertales I G 1994 *Journal of Fluid Mechanics* 260 155–184

[156] Gañán-Calvo A, Dávila J and Barrero A 1997 *Journal of Aerosol Science* 28 249–275 ISSN 0021-8502 URL https://www.sciencedirect.com/science/article/pii/S0021850296004338

[157] Gañan-Calvo A M 2004 *Journal of Fluid Mechanics* 507 203–212a

[158] Gomez A 1993 The electrospray: Fundamentals and applications *Experimental Heat Transfer, Fluid Mechanics and Thermodynamics 1993* Elsevier Series in Thermal and Fluid Sciences ed Kelleher M, Sreenivasan K, Shah R and Joshi Y (Amsterdam: Elsevier) pp 270–282 ISBN 978-0-444-81619-1 URL https://www.sciencedirect.com/science/article/pii/B9780444816191500289

[159] Abdar M, Pourpanah F, Hussain S, Rezazadegan D, Liu L, Ghavamzadeh M, Fieguth P, Cao X, Khosravi A, Acharya U R, Makarenkov V and Nahavandi S 2021 *Information Fusion* 76 243–297 ISSN 1566-2535 URL https://www.sciencedirect.com/science/article/pii/S1566253521001081

[160] Camporeale E, Chu X, Agapitov O V and Bortnik J 2019 *Space Weather* 17 455–475 (*Preprint* https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2018SW002026) URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018SW002026

[161] Psaros A F, Meng X, Zou Z, Guo L and Karniadakis G E 2023 *Journal of Computational Physics* 477 111902 ISSN 0021-9991 URL https://www.sciencedirect.com/science/article/pii/S0021999122009652

[162] Demmons N, Hruby V, Spence D, Roy T, Ehrbar E, Zwahlen J, Martin R, Ziemer J and Randolph T 2008 St7-drs mission colloid thruster development *44th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit* p 4823

[163] Roudnev V A, Merts S P, Nemnyugin S A and Stepanova M M 2020 *Journal of Physics: Conference Series* 1479 012043 URL https://iopscience.iop.org/article/10.1088/1742-6596/1479/1/012043

[164] Abidi H, Boveia A, Cavaliere V, Furletov D, Gekow A, Kalderon C W and Yoo S 2022 Charged particle tracking with machine learning on fpgas (*Preprint* 2212.02348)

[165] Våge L H 2022 Reinforcement learning for charged-particle tracking *Connecting the Dots Workshop (CTD)* pp PROC–CTD2022–37

[166] Newby J M, Schaefer A M, Lee P T, Forest M G and Lai S K 2018 *Proceedings of the National Academy of Sciences of the United States of America (PNAS)* 115 9026–9031 URL https://www.pnas.org/doi/full/10.1073/pnas.1804420115

[167] Barnes J and Hut P 1986 *Nature* 446–449

[168] Carrier J, Greengard L and Rokhlin V 1988 *SIAM Journal on Scientific and Statistical Computing* 9 669–686

[169] Keller S, Cavelan A, Cabezon R, Mayer L and Ciorba F 2023 Cornerstone: Octree construction algorithms for scalable particle simulations *Proceedings of the Platform for Advanced Scientific Computing Conference* PASC '23 (ACM) URL http://dx.doi.org/10.1145/3592979.3593417

[170] Narayanan R K and Madduri K 2017 Parallel particle-in-cell performance optimization: A case study of electrospray simulation *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* pp 1158–1167

[171] Brieda L 2020 *Plasma Simulations by Example* (CRC Press) chap 4