ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΑΣΤΗΜΙΚΗΣ

Προσομοίωση της αλληλεπίδρασης Φορτισμένων Σωματιδίων με Ηλεκτρομαγνητικά Κύματα με χρήση Παράλληλης Επεξεργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασίλειος Χαρίτων, ΑΕΜ: 57222

Επιβλέπων Καθηγητής: Θεόδωρος Σαρρής, Αναπληρωτής Καθηγητής, Η.Μ.Μ.Υ., Δ.Π.Θ.

Ξάνθη, 2023

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΑΣΤΗΜΙΚΗΣ

Προσομοίωση της αλληλεπίδρασης Φορτισμένων Σωματιδίων με Ηλεκτρομαγνητικά Κύματα με χρήση Παράλληλης Επεξεργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασίλειος Χαρίτων, ΑΕΜ: 57222

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Επιβλέπων καθηγητής: Θεόδωρος Σαρρής, Αναπληρωτής Καθηγητής Η.Μ.Μ.Υ. Δ.Π.Θ.

2ο Μέλος: Δημήτριος Σαραφόπουλος, Αναπληρωτής Καθηγητής Η.Μ.Μ.Υ. Δ.Π.Θ.

3ο Μέλος: Αυγερινός Αραμπατζής, Αναπληρωτής Καθηγητής Η.Μ.Μ.Υ. Δ.Π.Θ.

Ξάνθη, 2023

DEMOCRITUS UNIVERSITY OF THRACE
SHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING
SECTOR OF TELECOMUNICATIONS AND SPACE SCIENCE

Simulation of Wave-Particle interaction with the use of Parallel Processing

DIPLOMA THESIS

Vasileios Chariton, Registration Number: 5722

COMMITTEE OF EXAMINERS

Supervisor: Sarris Theodoros, Associate Professor D.U.T.H.

Member 2: Sarafopoulos Dimitrios, Associate Professor D.U.T.H.

Member 3: Arampatzis Avgerinos, Associate Professor D.U.T.H.

Xanthi, 2023

Περίληψη

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη ενός αλγορίθμου, για την προσομοίωση της αλληλεπίδρασης φορτισμένων σωματιδίων με κύματα, στις ραδιενεργές ζώνες της Γης. Η φύση του προβλήματος, απαιτεί από τον αλγόριθμο να είναι αποδοτικός και η χρήση τεχνικών παράλληλης επεξεργασίας είναι απαραίτητη. Ο αλγόριθμος δέχεται στην είσοδο του, μια κατανομή σωματιδίων και ένα ηλεκτρομαγνητικό κύμα, ενώ δίνει στην έξοδο τις καταστάσεις των σωματιδίων μετά την αλληλεπίδραση που είχαν με το κύμα. Μελετώντας τις καταστάσεις αυτές είναι δυνατό να ληφθούν συμπεράσματα, σχετικά με την επίδραση που έχουν ηλεκτρομαγνητικά κύματα διαφορετικών χαρακτηριστικών, πάνω σε ποικίλες κατανομές φορτισμένων σωματιδίων.

Τα σωματίδια τα οποία μελετώνται είναι παγιδευμένα στην Μαγνητόσφαιρα της Γης και εκτελούν τρεις γνωστές περιοδικές κινήσεις. Για τη μελέτη αυτών των σωματιδίων, αναπτύσσεται αρχικά μια μέθοδος για τη δημιουργία ποικίλων κατανομών τους, οι οποίες στη συνέχεια ενσωματώνονται στην προσομοίωση. Καθώς εξελίσσεται η προσομοίωση, τυχόν αλληλεπίδραση του εισαγόμενου κύματος με τα σωματίδια, δύναται να τα επιταχύνει και ορισμένα ίσως καταφέρουν να δραπετεύσουν. Κατά την αλληλεπίδραση αυτή, ο συντονισμός των συχνοτήτων μπορεί να επιφέρει μεγάλες αλλαγές σε μία συγκεκριμένη γωνία που εκφράζει το κάθε σωματίδιο. Η γωνία αυτή έχει την ονομασία pitch angle και είναι μία από τις κύριες παραμέτρους μελέτης των πληθυσμών φορτισμένων σωματιδίων της παρούσας διπλωματικής εργασίας. Για την λήψη συμπερασμάτων ακολούθησε εκτενής δημιουργία γραφημάτων και αρχείων ταινίας πάνω στις καταστάσεις των σωματιδίων σε κρίσιμες χρονικές στιγμές. Μεταξύ άλλων, το πρόγραμμα που αναπτύχθηκε λειτουργεί με διεπαφή γραμμής εντολών, ενώ έχει σχεδιαστεί και γραφικό περιβάλλον για την φιλική εκτέλεση των παραπάνω λειτουργιών της προσομοίωσης. Όσον αφορά τον παράλληλο αλγόριθμο που αναπτύχθηκε, υλοποιήθηκαν δύο διαφορετικά πρότυπα και προσεγγίσεις παράλληλου προγραμματισμού, συγκεκριμένα το *OpenMP* και το *MPI*.

Η διπλωματική εργασία πραγματοποιήθηκε στα πλαίσια έρευνας που γίνεται για την προτεινόμενη αποστολή *Λεύκιππος* του Εργαστηρίου Ηλεκτρομαγνητικής Θεωρίας και Διαστημικής Η.Μ.Μ.Υ. Δ.Π.Θ. Η αποστολή Λεύκιππος έχει ως στόχο την αποστολή ενός ζεύγους νάνο-δορυφόρων σε κατάλληλες ζώνες, για την παρατήρηση του παραπάνω φαινομένου. Ένας νάνο-δορυφόρος θα εκπέμπει ένα κύμα πολύ χαμηλής συχνότητας, ενώ το ζεύγος του θα ανιχνεύει την επίδραση που θα έχει αυτό το κύμα, πάνω στα ενεργητικά σωματίδια της περιοχής. Στην διπλωματική αυτή θα εξεταστεί πως ένα παραγόμενο τέτοιο κύμα μπορεί να επηρεάσει διαφορετικούς πληθυσμούς ενεργητικών σωματιδίων.

Ο κώδικας που αναλύεται σε αυτή την διπλωματική έχει αναρτηθεί στο GitHub και δύναται να μορφοποιηθεί στο μέλλον για την βελτίωση του. Η τρέχουσα έκδοση του κώδικα περιγράφεται στο κεφάλαιο 4.

Λέξεις Κλειδιά

Λεύκιππος, Νάνο-δορυφόροι, Προσομοίωση Ηλεκτρικά Φορτισμένων Σωματιδίων, Μαγνητικό Πεδίο Γης, Κίνηση Ηλεκτρικά Φορτισμένων Σωματιδίων, Ηλεκτρομαγνητικά κύματα, Αλληλεπίδραση Κύματος με Σωματίδια, Κατανομές Σωματιδίων, Pitch Angle, Παράλληλος προγραμματισμός

Abstract

This diploma thesis focuses on simulating the resonant interactions between electromagnetic waves and energetic particles within Earth's magnetosphere. Given the complexity of this process, the design of an efficient algorithm is necessary, and the use of parallel processing techniques becomes imperative. The algorithm reads a particle distribution and an electromagnetic wave and returns the resulting particle states after their interaction with the wave. By analyzing these resultant particle states, the effects of various electromagnetic waves on distinct distributions of charged particles can be deduced.

The energetic particles being examined are trapped within Earth's magnetosphere and exhibit three distinct periodic motions. To study these particles, a method for generating diverse particle distributions is initially developed, which are subsequently incorporated into the simulation. During the simulation, the incoming wave may interact and accelerate these particles, with some potentially gaining enough energy to break free from their trapped paths. The resonance achieved between the oscillation frequencies during the wave-particle interaction can significantly alter a particle's trajectory angle. This angle, referred to as the 'pitch angle,' is a central aspect of this study. To draw conclusions, an extensive array of graphs and movie files were generated to illustrate the particle states at key moments. These visual representations are easily reproducible, as the program supports both a command-line and a graphical user interface. The developed parallel algorithm was implemented and compared using distributed and shared memory computing, utilizing the *MPI* and *OpenMP* APIs, respectively.

This thesis was conducted within the framework of research for the proposed *Leucippus* mission at the Laboratory of Electromagnetic Theory and Space within the Department of Electrical and Computer Engineering at D.U.T.H. The mission's objective is to deploy a pair of nanosatellites to specific zones to study the described phenomenon. One nanosatellite will emit a wave of a designated frequency, with its counterpart measuring the wave's impact on the energetic particles in that region. This thesis delves into the potential effects of such a generated wave on various energetic particle populations.

The code analyzed in this thesis has been uploaded to GitHub and may undergo future modifications for improvement. The current version of the code is described in Chapter 4.

Key Words

Leycippus, Nanosatellites, Energetic Particle Simulations, Trapped Particle Motion, Electromagnetic Waves, Wave-Particle Interaction, WPI, Particle Distributions, Pitch Angle, Parallel Programming

Acknowledgments

First and foremost, I want to thank my thesis supervisor, Associate Prof. Theodoros Sarris, for providing me with the valuable opportunity to explore in depth the fascinating subject of energetic particle simulations. I am also thankful for the numerous meetings he organized, during which he guided me and imparted his knowledge on electromagnetic theory, satellite missions, and related simulations. The entire process of preparing the thesis was a rewarding experience, serving as the ideal conclusion for my studies in the Electrical and Computer Engineering Department of D.U.T.H.

I also want to extend my gratitude to the Ph.D. candidates Stylianos Tourgaidis and Konstantinos Papadakis, whose direct guidance on physics and scientific computing was crucial in completing my thesis.

Last but not least, I would like to thank my parents and sisters for their unwavering support throughout the years, enabling me to pursue my education at the Democritus School of Engineering.

Contents

Chapter 1

Introduction

1.1 The problem

In this thesis, the focus is on simulating the wave-particle interaction that occurs within the Earth's radiation belts, commonly known as the Van Allen belts. This phenomenon, extensively researched by numerous scholars, plays a crucial role in understanding the complexities of space physics and the Earth's magnetosphere. With the advancements in scientific computing, it's now feasible to simulate these interactions, providing deeper insights and visual representations. These insights have implications not only for scientific comprehension but also for practical applications in satellite operations and space exploration.

Scientific simulations often go hand in hand with high-performance computing. This is primarily because such simulations typically tackle issues that require intricate mathematical computations on large datasets, often leveraging techniques from numerical analysis. The initial wave-particle interaction algorithm, developed by Ph.D. candidate Stylianos Tourgaidis in Python [1], was designed to simulate the phenomenon for individual particles and small sets of them. Subsequently, there arose a need to modify this simulation to analyze the effect of an electromagnetic wave on large particle distributions. These distributions had to be examined and generated to further explore the wave-particle interaction of populations of particles that are commonly encountered in the radiation belt regions.

The initial challenge was to recreate the simulation using another object-oriented language capable of supporting parallel computing. C++ was chosen due to its prominence in high-performance computing and its extensive parallel computing capabilities. This exploration encompasses shared memory and distributed-memory parallelism, with the respective solutions compared and analyzed.

Another challenge involves understanding and generating particle distributions that mirror those found in nature, especially within the Earth’s radiation belts. Once the traits of these distributions from both inner and outer radiation belts were grasped, they were programmatically created. While it's possible to craft these distributions individually before running the simulation, a more efficient approach involved developing a method to auto-generate them via a user-friendly interface. This proved invaluable, allowing quick creation of generation distributions with specific characteristics in just seconds.

After conducting the simulation, pinpointing key regions within the simulated data is crucial. These specific regions elucidate the states of particles throughout the simulation and serve as a visual representation of the phenomenon. For this simulation, every particle begins with a pre-defined initial state, which evolves as the simulation progresses. This state is defined by various metrics such as momentum, position, or energy. The values of the most relevant metrics will be stored in memory as the simulation progresses and will be analyzed later. Ultimately, the phenomenon will be visually depicted through a range of plots and movie files.

1.2 Purpose

This thesis focuses on creating a high-performance particle simulator to depict interactions between particle distributions and an electromagnetic wave. The objective is to introduce a novel method that illustrates the wave's impact on a specified particle group. After the simulator is developed, its accuracy and behavior are evaluated using validation methods.

In addition to fulfilling the requirements of a diploma thesis, there's a drive to make a meaningful contribution to the broader field of Electrical and Computer Engineering. This thesis serves as a culmination of the insights and knowledge I've gathered during my academic journey in the Democritus School of Engineering. For this purpose, I tried to incorporate many aspects of the Electrical and Computer Engineering field which I'm passionate about. Specifically, this work shows an effort to encapsulate my experience in the fields of Applied Physics, Mathematics, and Software Engineering.

1.3 Thesis organization

The documentation has been organized in the following 8 Chapters:

Chapter 1 introduces the issue at hand and outlines the challenges addressed in this thesis. It also details the organization and structure of the thesis to enhance comprehension of the concept and workflow.

Chapter 2 briefly examines the theoretical and technical aspects of this thesis, setting aside the mathematical details which will be explored in the subsequent chapter. To grasp the concept of this simulation, it's vital to understand the foundational theory of the environment where these interactions occur and the trapped motion of the particles in study. Although there's an infinite variety of natural particle distributions, this chapter focuses on several common distributions. A section on particle detectors used in satellite missions follows, shedding light on how these instruments might be simulated. It concludes with a brief section dedicated to electromagnetic waves of very low frequency, a crucial element of this simulation.

Chapter 3 focuses on the mathematical aspects of the problem. The widely used Lorentz equation is described, showcasing its significance in simulating particle motion. The Runge-Kutta numerical method, which will be applied to determine particle states at specific moments, is also analyzed. The chapter concludes by presenting the different equations used in the code. Understanding these equations is vital to distinguish the differences in the developed simulations of wave-particle interactions.

Chapter 4 examines the code used for the simulation. It traces the journey from the initial pre-existing code to the development of a new one tailored for the task. This chapter demonstrates how distributions and waves are crafted programmatically. It also provides insights into the data structures used for the simulation. Then, the method for comparing simulation outcomes is outlined, aiding in drawing conclusions and the chapter wraps up with a description of the post simulation processing, a pivotal step in finalizing conclusions.

Chapter 5 delves into the parallel programming dimensions of the problem. The simulation was developed using the parallel programming approaches of distributed-memory programming with the Message Passing Interface (MPI) and shared memory programming using OpenMP. Finally, in this chapter the performance outcomes of these methods are compared.

Chapter 6 presents the outcomes of selected simulations. It begins with results from single particle simulations, detailing individual particle motion, its interaction with the wave, and the potential for resonance between the two—highlighting the profound effects of such resonance. The

chapter then transitions to visual outcomes from particle distribution simulations, offering a comprehensive understanding of the phenomenon.

Chapter 7 offers a technical overview of the setup and the systems that were used to develop and run the simulation. In the subsequent section the dependencies required to run the simulator are detailed.

Chapter 8 outlines how to use the simulator, explaining how to run the simulation on your system either through the command line interface or by using the program's graphical user interface.

Chapter 2

Brief reference to the theory

Before delving into the specifics of the problem, it's essential to provide a brief overview of its theoretical aspects. The occurrence that is the focus of this thesis happens within Earth's Magnetosphere; thus, providing some insights about this region would be appropriate. Next, describing the single particle motion will help understand the wave-particle interaction phenomenon later. Subsequently some common particle distributions along with their characteristics and observation regions are analyzed and a brief overview is given for the particle detectors used in space. This knowledge is essential for understanding and creating the programmatic representation of the particle detector, which will be used to capture the particle distribution change during the simulation.

2.1 Magnetosphere

The term "magnetosphere" refers to regions where the behavior of charged particles is primarily influenced by a planet's magnetic field. In this thesis, the focus is on Earth's magnetosphere, the most extensively studied among all planetary magnetospheres which also happens to be the most potent magnetosphere of a rocky planet in our solar system. While functioning as a protective shield against solar and cosmic radiation, which is harmful for biological organisms, the magnetosphere has been pivotal to our planet's ability to sustain life. This entire system has its roots deep within the Earth. The swirling, electrically charged, molten iron in the Earth's outer core produces a magnetic field extensive enough to reach far into space. In particular, the Earth's magnetosphere is divided into different regions, which can be distinguished depending on the physical phenomena prevailing in every one of them. These divisions are also based on the unique characteristics of energy, density, and the origin of magnetic field lines and plasma. The main zones are the bow shock, the magnetopause, the magnetotail, the plasmasphere, and the radioactive zones, which will be further described.

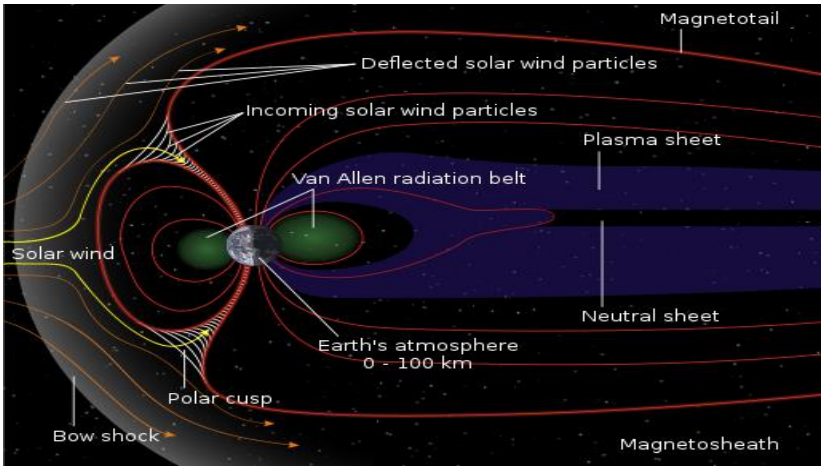


Figure 1: Earth's magnetosphere – Nasa IMAGE Science center.

Bow shock

The bow shock marks the separation between the magnetosphere and the surrounding medium. Bow shocks, named for the crescent-shaped wave created by a ship moving through water, form in various astronomical contexts. In the context of a planetary bow shock, the location and form of a bow shock are shaped by the interaction between the solar wind and the planet's magnetosphere [2]. The bow shock, situated ahead of the magnetopause, forms the distinction between the planet's magnetosphere, where particles are primarily held captive and become part of the planetary environment, and the solar wind, which is filled with plasma originating from the sun. Bow shocks can also appear in comets, because of the interaction between the solar wind and the cometary ionosphere. They are also prevalent in Herbig Haro objects, which are luminous nebulous features linked with the presence of newly formed stars. In this case, huge outflows of gas and dust from the star interact with the interstellar medium, creating bright bow shocks visible at optical wavelengths.

Figure 2: Ship bow wave.

Figure 3: Zeta Ophiuchi, massive star bowshock, Spitzer Space Telescope.

Figure 4: Herbig–Haro object in Orion nebula, Hubble Telescope.

Magnetopause

The magnetopause serves as a thin barrier that differentiates between the shocked solar wind plasma and the magnetosphere's plasma, with the latter's movement being primarily governed by Earth's magnetic field. This region serves as a crucial area for the transition of mass, momentum, and energy from the solar wind into the magnetosphere. The magnetopause is roughly as thick as one thermal ion gyroradius, which is a measure of the characteristic scale of the gyro-motion of a thermal ion in a magnetic field. In terms of meters, it is often estimated to be on the order of about 100 kilometers. However, the size and shape of the magnetopause vary as it responds to changes in pressure exerted by the solar wind [3]. During substantial coronal mass ejections, the magnetopause can be pressed towards Earth, and this can potentially place it within the path of geostationary orbits. When this occurs, certain satellites may lose their protective shield, thereby becoming susceptible to damage from the enhanced flux of high-energy particles.

Magnetotail

This elongated region of a planet's magnetosphere stretches in the direction opposite to the solar wind i.e., away from the Sun and is known as the magnetotail. It is quite extensive, extending over sixty Earth radii (60RE), a distance comparable to the orbit of the Earth's moon.
During periods of geomagnetic disturbance, magnetic field lines in Earth's magnetotail reconnect, leading to the acceleration of electrons into the upper atmosphere, which results in dynamic and sometimes widespread auroral displays [4]. The magnetotail has two lobes, northern with magnetic field lines pointing towards the Earth and southern with magnetic field lines pointing away from the Earth. The lobes contain only a few charged particles opposing the solar wind and are separated by a plasma sheet, an area with weaker magnetic fields and higher density of charged particles.

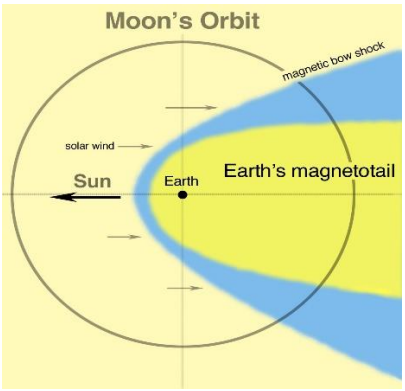


Figure 5: Magnetotail, Tim Stubbs/University of Maryland/GSFC 1Tim Stubbs/University of Maryland/GSFC.

Plasmasphere

This region is characterized by low-energy particles, forming a dense but cold plasma. Above the Earth's atmosphere, these regions are continually exposed to ultraviolet radiation from the Sun, leading to ionization with electrons released from neutral atmospheric particles. Consequently, negative, and positive electrically charged particles are formed, defining this state of matter as plasma, which is essentially ionized gas matter. The plasmasphere allows for easy transmission of magnetic reflection waves, also known as *MR whistlers*, which are very low-frequency waves which are discussed in Section 2.6. These waves, originating from lightning, can be transmitted by reflections to the plasma zone and their interaction with trapped energetic particles has been studied [5].

Radiation Belts – Van Allen Belts

The focus of this thesis is within a region occupied by extremely energetic particles (>100keV), primarily originating from the solar wind. These regions are named after James Van Allen, the scientist who discovered them. Two of these zones encircle the Earth at distances varying from 640

to 58,000 kilometers above its surface, and it's possible that additional such zones may be temporarily formed [6]. Satellites, like the Van Allen Probes [7], that study or traverse these zones must be properly engineered to resist the potential harm caused by these high-energy particles. For manned missions that must traverse these regions to reach outer space, it is essential to pass through them rapidly to limit radiation exposure.

The radiation belts consist of two donut-shaped regions filled with highly energetic charged particles encompassing the Earth's magnetic field. The inner belt, spanning altitudes from 2,000 to 5,000 kilometers above the Earth, mainly comprises high-energy protons and is relatively consistent over time. These protons originate from cosmic ray ions colliding with atoms in the atmosphere. Even though the ions' accumulation in the inner belt is gradual and their overall quantity relatively small, they reach high intensities due to their stable confinement near the Earth, a process that may span several years. The outer belt, a region roughly 6,000 kilometers thick and located at an altitude of 15,000 kilometers, predominantly houses ions and electrons of markedly lower energy. Unlike the inner belt, the outer belt can experience periodic expansions and contractions over time. These are generated by cosmic rays and magnetospheric acceleration processes. The most energetic particles in this region are commonly known as in the outer radiation belt. The space dividing the inner and outer Van Allen belts is often referred to as the "safe zone" or "safe slot", serving as the site for medium Earth orbits. This gap is thought to be created by very low frequency radio waves that scatter particles across different pitch angles, thereby introducing new ions into the atmosphere. Although solar eruptions can temporarily fill the gap with particles, these typically deplete within a few days.

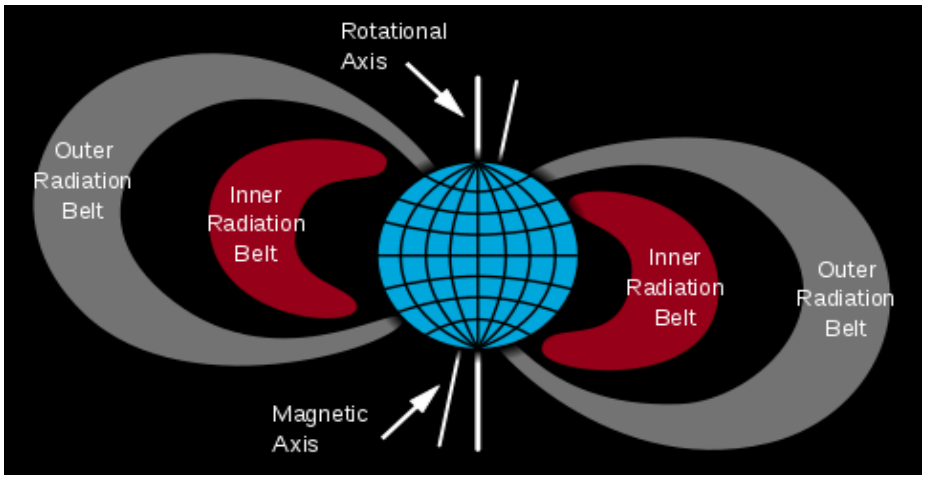


Figure 6: Van Allen radiation belt - Nasa IMAGE Science center.

2.2 Particle motion

The energetic particles, electrons, and protons, that are trapped in the radiation belts, follow three known periodic motions. This expressed particle motion derives after solving the Lorentz equation of energetic particles inside the Earth’s magnetic field which can be modeled as a dipole [8]. Every one of these typical particle motions occurs in a different frequency and is related to an adiabatic invariant. These invariants can be violated and that would mean that these values are no longer constant. An adiabatic invariant is violated, either when the fluctuations over time in the background fields surpass the rate of the corresponding movements, or when the spatial shifts in background fields are less extensive than the radius of the related movements. Adiabatic invariants can be derived by performing a closed line integral of the particle's canonical momentum over one

complete cycle of the related motion. In the following sections, every motion will be described in detail and will be accompanied by its related adiabatic invariant.

Bounce motion

The transit between the Northern and Southern hemispheres is a movement which is characterized as the *bounce motion*. It takes approximately one second to complete the full motion and the mirror points where each particle will be reflected can differ. The bounce of a particle is guided by the speed vector, and specifically by an angle which is referred to as the “pitch angle” and will be elaborated in the subsequent section. Additionally, the frequency of the bounce motion is influenced by the particle's velocity and by the length of the magnetic field line, which the particle is following. As a particle draws closer to the magnetic field, which amplifies in strength, the vertical component of the velocity increases. Simultaneously, the parallel component lessens, eventually reaching zero. This signifies a mirror point, and the particle will bounce back towards the other hemisphere.

The back-and-forth movement along a field line between mirror points is dictated by the particle's kinetic energy. The greater a particle's speed, the shorter its bounce period. For instance, electrons with an energy of 100 KeV at *L-shell* = 9 (planetary magnetic field line set by radius) have a bounce period of roughly 1 second, which is around 43 times less than the bounce period of a proton that possesses the same energy. Only particles with motion nearly parallel to the field line and an almost-zero magnetic moment manage to evade mirroring, but these are rapidly absorbed by the atmosphere and thus, eliminated. This elimination results in a set of directions surrounding the field line that is devoid of particles, creating a so-called “loss cone" which is described in the next section.

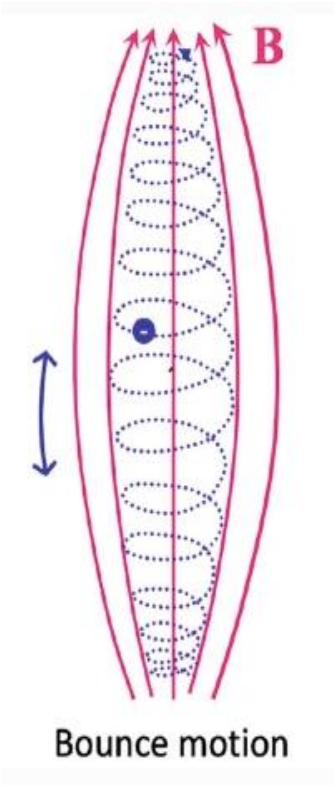


Figure 7: Bounce motion of an energetic particle.

The adiabatic invariant that is associated with this motion is the longitudinal invariant *J* and is preserved when changes to the system happen slowly relative to the particle's bounce frequency. It can be mathematically expressed as an integral of the parallel momentum between the two mirror points:

$$J = \int_a^b p_{\parallel}\, ds$$

The integral is evaluated using an arc-length element *ds* along the path defined by the magnetic field lines. The whole path is initiating at the equator, bounces off both poles and returns to the initial point. Hence, each magnetic field line corresponds to a distinct value of J for the given particle.

Gyro motion

As a particle bounces between the two hemispheres, it also traces a helical path, peaking near the particle's magnetic poles. This revolution around the magnetic field line is the so-called “gyro motion” of the particle. This happens under gyration periods which are contingent on the strength of the magnetic field and the particle's mass. A more potent magnetic field results in a shorter gyro-period, while a heavier particle leads to a longer gyro-period. Given the magnetic field strength of inner magnetosphere, the electron gyro-period is much faster than the bounce-period and usually expressed in milliseconds. The protons which are considered relatively massive can exhibit gyro-periods within the range of 0.1 to 1 second, while a typical electron gyro-period is of the order of 0.1 to 1 millisecond.

This movement is a blend of translational and rotational motions, with the latter centered around the path of the former. The rotational speed of a particle is dictated by the strength of the magnetic field, while the direction of rotation is influenced by the particle's charge.

There's an approximation for particles that preserve the first adiabatic invariant, M. In this scenario, the gyrofrequency of the particle is quicker, and the gyro-radius is notably smaller than the speed at which the background fields alter. This concept is referred to as the *guiding center* approximation, which interprets the particle's motion as circulating around a shifting guiding center.

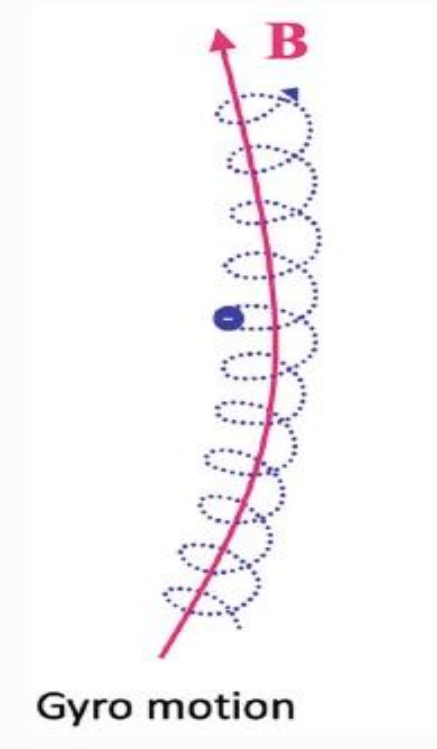


Figure 8: Gyro motion of an energetic particle.

This motion is of utmost importance when the electromagnetic waves are going to be introduced to the simulation. The potential resonance of the gyrofrequency of the particle with the frequency of the wave can produce *gyro-resonance pitch angle scattering* which can cause the particles to precipitate. This phenomenon is further discussed in the following chapter that considers the particle interaction with the electromagnetic waves.

Drift motion

This movement is relatively slow, taking hundreds of seconds, and it orbits the Earth's magnetic axis. The direction of this drift depends on the charge sign of the particles. While protons drift in a westward direction, electrons tend to move eastward. The drift motion is related to the third adiabatic invariant that represents the total magnetic flux. This invariant is denoted as Φ and enclosed by a drift surface. However, since this drift motion tends to be relatively slow, Φ is frequently not preserved in real-world applications. The removal of higher-order terms leads to the formation of the guiding center drift equations. Initially, if a uniform and static magnetic field is present, particle gyration occurs around a guiding center. However, when the electric or magnetic field is not static or homogeneous, drifts occur due to the alternation between larger and smaller curvatures of particle gyration. The standard drift types along with their occurrence conditions:

- **ExB Drift**: When **t**he electric field is static and perpendicular to the magnetic field.
- **Polarization Drift:** When the electric field varies slowly over time.
- **Gradient Drift**: When the magnetic field is not uniform.
- **Curvature Drift**: When the field lines exhibit curvature.

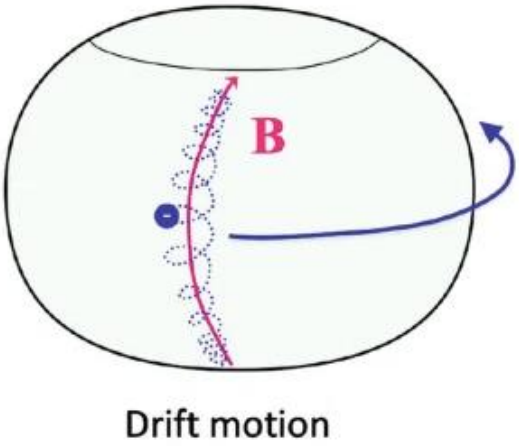


Figure 9: Charged Particles in Near-Earth Space, Hannu E. J. Koskinen & Emilia K. J. Kilpu

In general, the particles exhibit a helical orbit, sliding slowly along the longitude as illustrated in Figure 10.

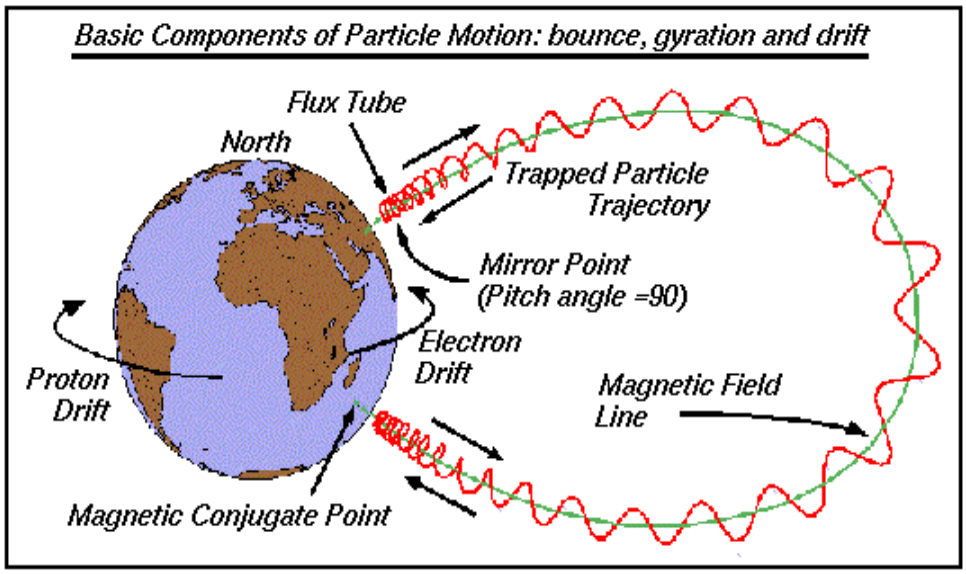


Figure 10: Trapped particle motion - Nasa IMAGE Science center.

This complex motion allows the particles to remain trapped for an indefinite time in the absence of any external influence. Nevertheless, it is known that their concentrations are not constant over time [9]. Hence, some factors can influence this motion, leading to their escape from these zones. It is known that various waves transmitted in the region, with different frequencies, can influence the overall particle concentrations [10]. Before exploring the intricacies of wave-particle interactions, it's essential to introduce a crucial characteristic of the trapped particles: the *pitch angle* (*PA*). The pitch angle represents the angle formed between the velocity vector of a particle and the vector of Earth's magnetic field. The upcoming section will delve into this important aspect.

2.3 Pitch Angle (PA)

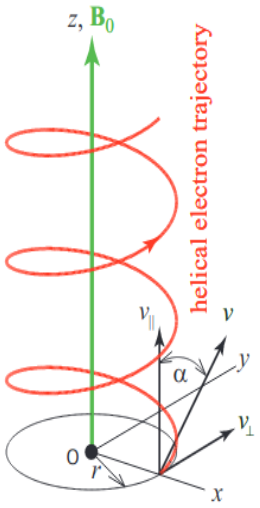


Figure 11: The helical path followed by a particle.

The pitch angle is a fundamental parameter in the study of charged particles' dynamics within magnetic fields, such as Earth's magnetosphere. It is defined as the angle between a particle's velocity vector and the prevailing magnetic field lines. This angle has a potential range from 0 to 180 degrees. A pitch angle of 0 degrees signifies that a particle's trajectory is perfectly aligned with the magnetic field, traveling along its lines of force. In contrast, a pitch angle of 180 degrees indicates movement in the antiparallel direction to the field lines. Particles with pitch angles near these extremes tend to propagate more directly along the magnetic field. Intermediate pitch angles cause particles to adopt helical trajectories around these field lines.

Pitch angle distributions commonly referred to as *PADs* provide insights into the behavior of charged particle populations. In space weather studies, understanding the PADs of charged particles can help scientists predict how particles trapped in Earth's magnetosphere might move, especially during geomagnetic storms or solar wind disturbances. Such distributions can offer valuable information about the source, acceleration mechanisms, and loss processes of these particles. By analyzing PADs, researchers can discern patterns linked to wave-particle interactions, magnetospheric resonances, and boundary crossings. This knowledge is not only crucial for advancing fundamental space science but also has practical implications. Accurate predictions based on PADs can guide satellite operations, safeguarding them from potential radiation hazards and ensuring the safety of astronauts on space missions. Furthermore, understanding these distributions can aid in anticipating and mitigating the effects of space weather events on terrestrial technologies, such as power grids and communication systems. Given their significance in the visualization of this thesis, some of the most prevalent PADs will be analyzed in Section 2.4.

Equatorial pitch angle

The Equatorial Pitch Angle is defined as the pitch angle observed at the Earth's equator, and it plays a crucial role in determining the behavior of charged particles within Earth's magnetosphere. This angle is directly associated with the specific location where a particle reflects (bounces) in its path through the magnetosphere. A larger equatorial pitch angle results in the particle reflecting at a higher altitude and sooner in its trajectory. Conversely, a smaller pitch angle will cause the particle to bounce back at a lower altitude and later in its motion. If the pitch angle is decreased significantly, the particle can descend to very low altitudes, reaching the dense upper layers of the Earth's atmosphere. In that region, it might interact with atmospheric particles, losing its energy and disrupting its previous periodic motion. Such particles, especially those that reach these lower altitudes and consequently lose their energy, fall within what is termed the *loss cone*. Figure 12 illustrates the loss cone concept. When a particle's equatorial pitch angle falls within the cone's boundaries, it will travel to a low enough altitude, resulting in energy depletion and ending its typical bouncing trajectory.

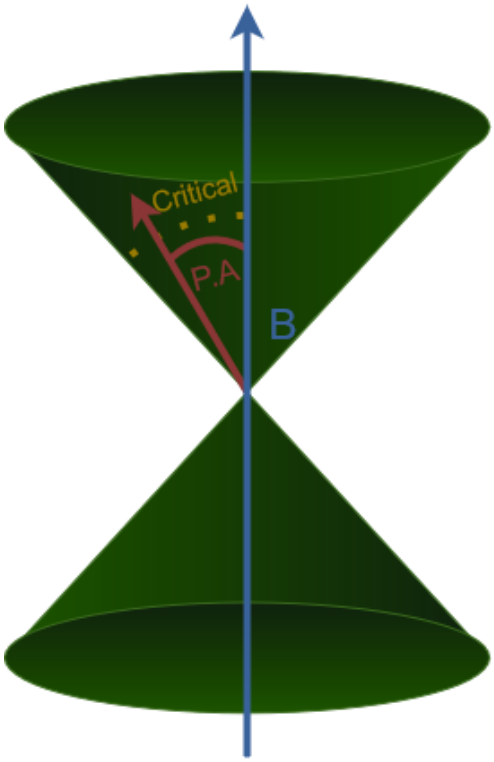


Figure 12: Pitch Angle within the loss cone. The blue arrow represents Earth's magnetic field, while the red arrow represents the velocity vector of a particle within the loss cone. The dashed yellow line illustrates the critical angle that marks the boundaries of the loss cone. A particle with this pitch angle will precipitate into the upper atmosphere.

Of particular interest are the trapped particles with pitch angles nearing the loss cone boundary. A slight reduction in this angle could lead them to the upper atmosphere, where they would release energy to the surrounding ions. One factor that can influence this angle is the interaction with electromagnetic waves, a phenomenon explored in the subsequent chapter. The wave-particle interactions, which are the basis for the simulation of this thesis, play an essential role in a range of astrophysical, space weather, and planetary phenomena. In this case, these interactions can alter particle trajectories and facilitate energy exchanges between the waves and particles, affecting the dynamics of the Earth’s magnetosphere.

2.4 Pitch Angle Distribution (PAD)

The PAD distributions of energetic particles play a significant role in describing the characteristics of Van Allen bands and the processes that take place in different regions. Presented below are some examples of empirical data from Van Allen Probes [7], which illustrate these distributions. Recent advancements in pitch angle-resolved data collection from the Magnetic Electron Ion Spectrometer or else *MagEIS* on the Van Allen Probes allowed researchers to analyse the distribution of energetic electrons in the slot region and inner radiation belt. Among the various distributions observed in these regions, some occur more frequently than others. The most typical PAD distributions are the following:

Normal PAD

Normal distributions are also known as Gaussian distributions and are distributions that exhibit symmetry around their means, have peak values at their respective means, and display a bell-shaped curve. Normal distributions are quite common in nature and are observed in various fields, including natural sciences, social sciences, finance, and more. A normal distribution is a type of PAD of energetic electrons that is commonly found in the outer radiation belt. It is characterized by a peak in electron flux at a pitch angle of ninety degrees, with a smooth decrease in flux at smaller pitch angles. As mentioned earlier, particles with a pitch angle close to the critical angle are more likely to enter the loss cone and be lost. While the exact mechanisms that lead to the formation of normal PADs are not fully understood, it is believed that they result from a combination of loss to the atmosphere and pitch angle diffusion [11]. These distributions are typically found in the outer Van Allen zone and they dominate at specific distances from Earth (L ~ 3.5–4) being more pronounced during the day.

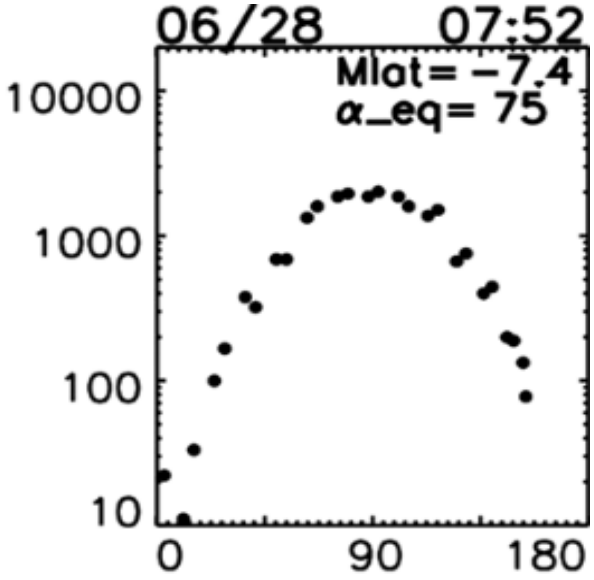


Figure 13: Empirical data for normal PAD - MagEIS [7].

Butterfly PAD

Butterfly PADs of energetic electrons are characterized by two peaks in electron flux, with a minimum around 90 degrees pitch. They are typically observed in the outer radiation belt and are thought to be formed due to the combination of pitch angle scattering and drift shell splitting [12]. Butterfly distributions are also associated with the presence of chorus waves, which can cause rapid energization and transport of electrons in the outer radiation belt.

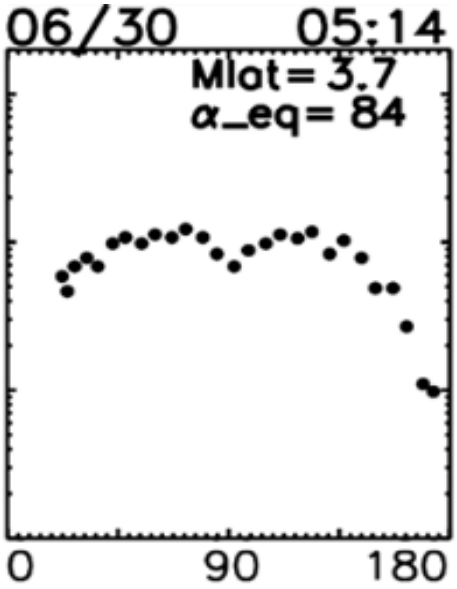


Figure 14: Empirical data for butterfly PAD - MagEIS [7].

Pancake PAD

Pancake PADs are the most common type observed in the outer radiation belt, particularly on the dayside [13]. These equatorial PADs show electron flux peaking at a 90° pitch angle and gradually decreasing toward the field-aligned directions. Pancake PADs form due to PA diffusion caused by electromagnetic waves, resulting in the loss of low PA electrons to the atmosphere. Additionally, inward radial diffusion can also lead to a flux peak around 90° PA, governed by the conservation of the first two adiabatic invariants [14].

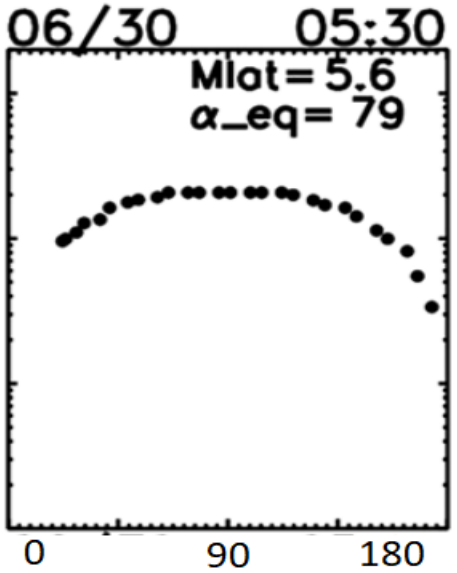


Figure 15: Empirical data for pancake PAD - MagEIS [7].

Flattop PAD:

The flat top distribution is characterized by a flat top wide pitch angle range centered around 90 degrees and a sharp drop-off at larger and smaller pitch angles. It is thought to form because of the balance between the inward radial diffusion or can be a result of strong wave-particle interactions.[15]. Flattop PADs exist in the outer belt at high L region as well as inside of the plasmasphere and can be a transition between the pancake PAD and butterfly PAD.

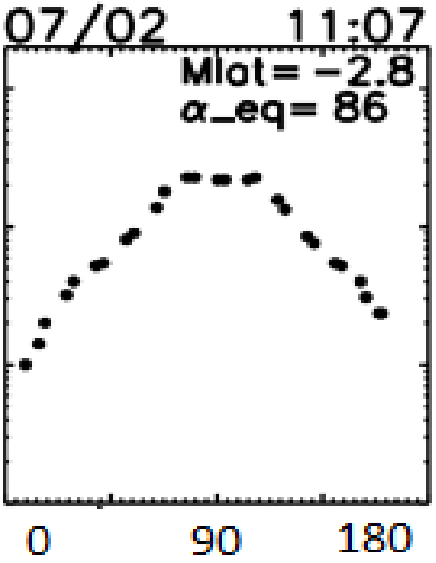


Figure 16: Empirical data for flattop PAD - MagEIS [7].

90° Minimum

The minimum 90 PAD is consistently observed in the inner belt, with a more pronounced minimum at 90° pitch angle for electrons with higher energies. During quiet times, the minima at 90° pitch angle in PADs are not present. However, during moderately disturbed times, the minima at 90° pitch angle reappear, indicating the prevalence of 90°minimum PADs in the slot region. Wave-particle interactions between energetic electrons and fast magnetosonic waves may be a possible mechanism responsible for the formation of this type of PAD [16].

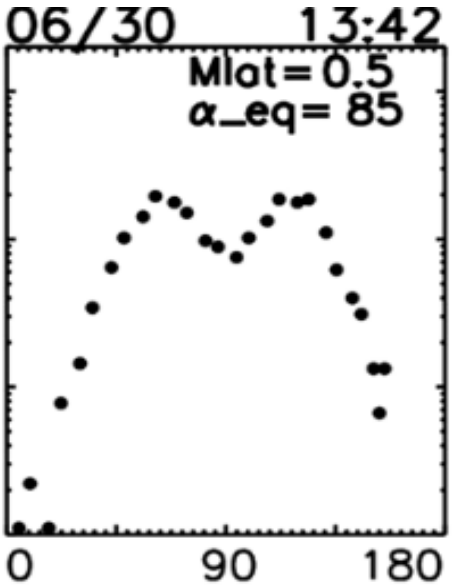


Figure 17: Empirical data for **90°** *Minimum PAD - MagEIS [7].*

2.5 Particle Detectors in space applications

To better understand how the detector was modeled for this simulation, it is helpful to provide a brief description of the characteristics of these instruments. The focus is on detectors used in space applications, specifically those capable of measuring the flux of energetic particles.
In space, the detected particles primarily originate from the Sun, but they can also come from other planets, stars, and even other galaxies. The range of energies these particles possess is vast, spanning from 1eV to over 10^20eV. The particles detected typically include electrons, protons, as well as other charged or neutral molecules and atoms. Furthermore, understanding the plasma environment that will be encountered is vital. Researchers must be prepared for conditions like a cold beam like the solar wind or a hot plasma like Earth's plasma sheet. The presence of intense

radiation belts can cause false counts and decrease the instrument's lifespan. To detect these particles, devices with several components are utilized [17], [18], [19]:

Collimator

This mechanical device restricts the range of angles through which particles can enter the detector. To "collimate" a beam means to get its particles to move in parallel directions. Collimators are used in detectors to enhance their angular resolution and minimize background noise. They achieve this by permitting only particles that come from a specific direction to reach the detector. This selective process significantly enhances the detector's sensitivity towards sources from that direction, which aids scientists in the identification and study of distant astronomical objects and phenomena.

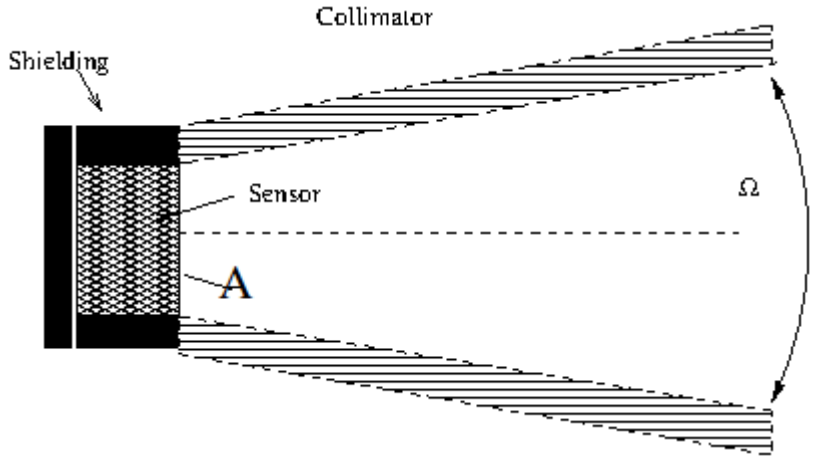


Figure 18: The collimator aperture covers a solid angle Ω*, the detector has an area A. Only particles coming from a direction within the solid angle* Ω *enter the detector [20].*

Their importance is particularly pronounced in space-based detectors. This is due to the space environment being awash with various particles and radiation coming from all possible directions. Without a collimator, it would be challenging to distinguish the signals originating from the astronomical object being studied from the background noise. The design and construction of collimators can differ based on the specific requirements of the detector and the type of particles or waves to be detected. However, they frequently comprise a series of slits or holes which allow the passage of the particles or waves.

Analyzer

It filters specific particles based on predetermined parameters. These selected particles then undergo further analysis. This analysis can range from hardware circuits that process the raw signals from the detectors to software algorithms that analyze the processed data to extract useful information. Analyzers are critical components of space-based detectors because they allow the vast amounts of data collected by these detectors to be transformed into forms that can be understood and studied by human scientists. When designing such analyzers, multiple factors need to be considered, such as the spacecraft's accommodation, available power for the instrument, mass and volume limitations in the payload, and whether the spacecraft is spinning or non-spinning. Maintaining the spacecraft's electrostatic cleanliness and ensuring a sufficient telemetry rate to transmit data back to Earth are also important.

Detector

This is the actual component that measures the quantity of particles, usually based on their energy.

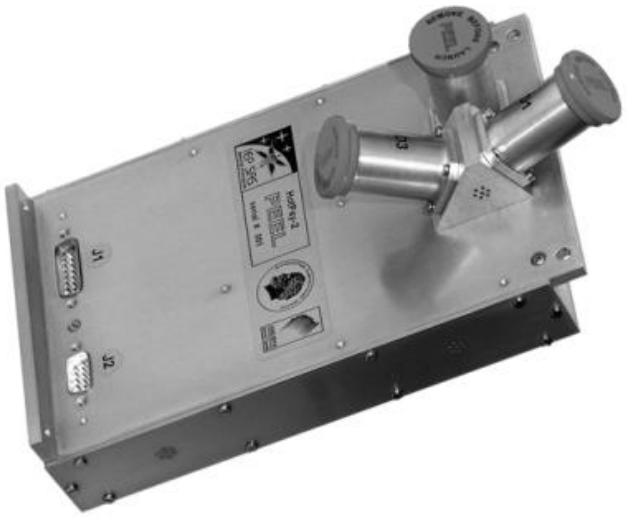


Figure 19: Mechanical design of PEEL, Energetic Electrons Precipitating at High Latitude, HotPay-2 Mission [21].

A few prominent types are:

1. **Silicon semiconductor detectors**: These devices work by using a solid-state silicon material that generates a charge when a particle passes through it. By measuring this charge, the energy of the particle can be determined, and by counting these events, the flux can be quantified. An example of such a detector is the Precipitating Energetic Electrons at high Latitude (*PEEL*) detector [21] which is based on a solid-state surface-barrier silicon detector and shown in Figure 19.
2. **Scintillation Detectors**: These rely on certain materials (scintillators) that emit light when struck by an energetic particle. The resulting light can be measured by a device like a photomultiplier tube or a photodiode, and the energy of the incident particle can be inferred.
3. **Calorimeters**: Used in high-energy physics experiments, these devices measure the total energy deposition of a particle by absorbing it completely. By counting the number of such total absorption events, the flux can be measured. An example is the Fermi Gamma-ray Space Telescope that uses a calorimeter in its Large Area Telescope (*LAT*) to measure the total energy deposited by particle showers generated by high-energy gamma rays [22].
4. **Cherenkov Detectors**: These measure the *Cherenkov* radiation emitted when a charged particle passes through a dielectric medium at a speed greater than the phase velocity of light in that medium. The energy (and hence the flux) of the particles can be determined from the characteristics of the emitted light.
5. **Time-of-Flight Detectors**: These devices measure the time it takes for a particle to traverse a known distance. By knowing the particle's speed and mass (which can often be inferred from the type of particle and the context of the experiment), its energy can be deduced. The Payload for Antimatter Matter Exploration and Light-nuclei Astrophysics (*PAMELA*) mission uses a Time-of-Flight system to measure the velocity, charge, and mass of cosmic rays, contributing to the identification of their nature [23].
6. **Transition Radiation Detectors**: These detectors measure the radiation emitted when a relativistic particle transitions between two media with different refractive indexes. The energy of the particles can be determined from the characteristics of this emitted radiation.

Electronics

This category includes batteries, analog electronics designed to amplify the signal, and convert it for further analysis, among other functions. In many cases, it is essential for these components to be radiation-hardened to withstand the harsh environment. The specific design of the electronics can vary greatly depending on the requirements of the detector and the constraints of the mission, such as the available power, weight, and volume, the expected radiation environment, and the need to withstand the forces experienced during launch.

To achieve scientific objectives, researchers must determine the required resolution in time, angle, energy, and mass per charge. Additionally, defining the necessary energy and density range for measuring encountered plasmas is crucial. These are also some of the requirements for the modelling of the instrument to capture the energetic particles that are being simulated. The detectors can separate the particles that pass through them in terms of time, space, and energy, which is particularly important for the algorithmic part of the simulation, discussed later in Section 4.7. These separations are also called bins. A summary of the separations made by a sensor is described below:

Time

This refers to temporal sampling, which is crucial for particle detectors' ability to accurately determine when particles are detected. Since particle detection events can occur at very short intervals, the detector must distinguish between individual events. One essential consideration is the Nyquist-Shannon sampling theorem, which states that to accurately capture a signal without errors or distortions, the sampling frequency must be at least twice as high as the highest frequency component of the signal. In practical terms, this means that the sampling rate must be sufficiently high to separate individual particle detections while also respecting the Nyquist frequency limit, where $f_N = \frac{f_s}{2}$ [24].

Space

The detector, besides the limitation imposed by the collimator, regarding the total number of particles that can enter, may include other mechanical parts that prevent the entry of larger particles that may not be of particular interest for the task at hand. A particle passing through the sensor has a specific direction defined by its velocity vector, and more specifically, its pitch angle, referenced in Section 2.3. During detection, the detector will gather information not about individual particles but about distributions of particles. The detector may consist of individual detectors of the same or different design, depending on the objectives of each mission, which are suitably positioned and can detect different angular ranges. The separation is made into sectors, each covering a specific range of angles. Detectors that cover all possible angular ranges from which particles can originate are sometimes called *Omni-Directional Particle Detectors* or *Telescopes* (*ODPD/ODPT*). For detectors mounted on rockets, the rotation period must be considered, together with the time sampling offered by the detector, to fully characterize the detection range.

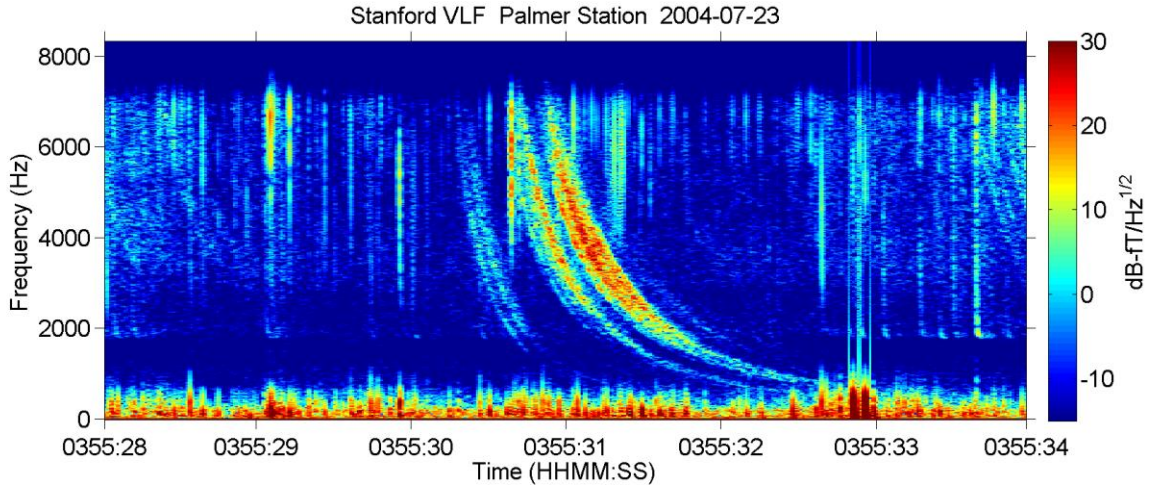
Energy

To determine the energy of the particles, calorimeters are used, which subject the particles to energy absorption effects. This categorizes them in terms of energy. This process requires appropriate electronics to amplify and manage the incoming signal.

2.6 Very Low Frequency (VLF) electromagnetic waves

Very Low Frequency (*VLF*) waves, covering the range of 3 kHz to 30 kHz, occupy a specific position in the electromagnetic spectrum, characterized by wavelengths that extend from 10 to 100 kilometers. VLF waves have been applied in a wide range of fields, including communication with submarines, monitoring space weather, serving as earthquake precursors, and more. Despite their practical significance, VLF waves also evoke scientific curiosity, serving as windows into the ionospheric and magnetospheric behaviors.

Whistler waves are a type of VLF waves typically spanning from a few hundred Hertz (Hz) to a few kilohertz (kHz). These waves originate naturally from lightning discharges, and they possess unique traits that, upon interaction with particles, can lead them to descent towards the ionosphere with quantitative examinations [25]. They usually occur in the Earth's magnetosphere and ionosphere and were named "whistler" waves due to their characteristic audio-like tone resembling the sound of a descending whistle. These waves are produced when lightning discharges generate electromagnetic pulses that propagate along the Earth's magnetic field lines. These waves can engage in intricate interactions with charged particles, thereby altering their trajectories and energies. These wave-particle interactions can adjust the energy and PADs of energetic electrons. Such interactions can scatter these electrons, altering their paths and potentially driving them into the Earth's atmosphere or reshuffling their positions within the radiation belts. This dynamic interplay significantly influences the overall behavior and population distribution of particles within the magnetosphere, with broader implications for satellite operations and space weather forecasting.



Spectrogram of a very low-frequency (VLF) electromagnetic whistler wave captured by the wave receiver at Palmer Station, Antarctica. [Stanford University VLF group's research]

Chapter 3

Mathematical solution

This chapter covers the equations of motion relevant to the adiabatic particle motion and the interaction between particles and waves. The initial section analyzes the simple scenario of particle motion without incorporating any external fields into the equations. Following that, waves are introduced into the equations, illustrating the associated differential equations. Lastly, the description of the numerical approach employed, namely the Runge-Kutta 4 method, is presented. It is important to note that bold characters in the formulas indicate vectors, while regular font signifies scalar values.

3.1 Adiabatic Motion

Before formulating the equations for wave-particle interactions, it's beneficial to examine the motion of a charged particle within Earth's static magnetic field when wave fields are not present. For particles that don't engage with electromagnetic waves, the problem is relatively simple. Their motion can be defined by differential equations. While the Earth's gravitational field's influence on such particles is deemed negligible, the Earth's magnetic field does exert a Lorentz force on them, as captured by the subsequent formula.

$$\boldsymbol{F} = q(\boldsymbol{E} + \boldsymbol{v}x\boldsymbol{B}), \quad (3.1a)$$

The cross-product of velocity and the magnetic field determines the key role of the pitch angle — the angle between these two factors, as elaborated in Section 2.3. A particle's velocity can be split into two parts: one aligned with Earth's magnetic field and the other perpendicular to it. The parallel velocity drives the particle along potential lines, while the perpendicular component causes the particle to circle around these lines. For particles moving at speeds nearing the speed of light, often termed relativistic particles, the formula describing the velocity variation is:

$$\frac{d\boldsymbol{v}}{dt} = \frac{1}{\gamma m}\left(q\boldsymbol{E}(r) + q\boldsymbol{v}x\boldsymbol{B}(r)\right), \quad (3.1b)$$

While the expression for the momentum variation:

$$\frac{d\boldsymbol{p}}{dt} = q\left(\boldsymbol{E}(r) + \boldsymbol{v}x\boldsymbol{B}(r)\right), \quad (3.1c)$$

where $\boldsymbol{v} = \frac{d\boldsymbol{r}}{dt}$ is the particle's velocity vector, $\boldsymbol{E}(r)$ is the vector for the electric field and $\boldsymbol{B}(r)$ is the vector for the magnetic field in the position *r*. Also, $\gamma = \frac{1}{\sqrt{1-v^2/c^2}}$ is the Lorentz factor, $\boldsymbol{p} = \gamma m\boldsymbol{v}$ is the relativistic particle momentum, $\boldsymbol{r}$ is the position of the particle, and *m* is the particle's mass.

From the given equation, the three periodic motions of the particles are derived, as detailed in Section 2.2. Each motion is linked to an adiabatic invariant, a set of quantities that remain constant as trapped particles undergo periodic movements. Especially noteworthy is the first adiabatic constant, since its disruption happens in time frames like the wave-particle interaction durations. These invariants are obtained using the Hamilton-Jacobi theory [26]. The canonical momentum for a particle carrying a charge in a magnetic field, represented by vector potential **A**, is $\gamma m\boldsymbol{v} + q\boldsymbol{A}$. By using the Stokes' Theorem:

$$\oint \boldsymbol{A} dl = \int \nabla x \boldsymbol{A} d\boldsymbol{S}, \quad (3.1d)$$

By integrating the momentum around the particle's trajectory, the following relationship can be obtained:

$$\boldsymbol{J}_n = \oint (m\boldsymbol{v} + q\boldsymbol{A}) d\boldsymbol{l}, \quad (3.1e)$$

where ***dl*** is the elementary distance travelled by the particle, *m* is the particle's mass, *q* is the particle charge, and ***v*** is the particle's velocity. For the helical motion of the particle, this constant takes the form:

$$J_1 = \oint (m\boldsymbol{v} + q\boldsymbol{A}) d\boldsymbol{l} = (\pi\gamma^2 m^2 v_\perp^2)(qB)^{-1} = (\gamma^2 m \boldsymbol{v}_\perp^2)(2B)^{-1}, \quad (3.1f)$$

$$J_1 = \frac{p_\perp^2}{2m_0 B} = M_{adiabatic} \propto \frac{sin^2\alpha}{B}, \quad (3.1g)$$

where m_0 represents the particle's rest mass, $\alpha = tan^{-1}\left(\frac{v_\perp}{v_z}\right)$ is the pitch angle of the particle, $v_\perp$ is the velocity perpendicular to the magnetic field and v_z is the velocity parallel to the magnetic field. The first adiabatic constant, often termed as the *magnetic momentum* is associated with the gyration about magnetic field lines. The focus is on the first adiabatic invariant, particularly its deviation, which takes place within the time scales of the wave-particle interaction. The velocity components relative to the magnetic field are expressed as:

$$v_z = vcos\big(\alpha(\lambda)\big), \quad (3.1h)$$
$$v_\perp = vsin\big(\alpha(\lambda)\big), \quad (3.1i)$$

Where v_z is the motion in the direction of the magnetic field, while $v_\perp$ is the perpendicular motion. The pitch angle and the equatorial pitch angle of the particle are related by:

$$sin^2\big(\alpha(\lambda)\big) = \frac{B(\lambda)}{B_{eq}} sin^2 a_{eq}, \quad (3.1j)$$

From this relationship, it becomes evident that if the pitch angle α is 90 degrees, the parallel velocity component v_z becomes zero. Consequently, the particle bounces back, initiating parallel movement towards the opposite pole. The mirror point of a particle is directly associated with its equatorial pitch angle. As this angle decreases, the reflection point for the particle descends to increasingly lower altitudes. Eventually, there is no reflection; the particle reaches altitudes so low that the denseness of the upper atmosphere prevents its return to its typical periodic motion. This is due to the particle's interactions with the ions in this dense region, causing it to lose energy. A specific altitude serves as a threshold. Beneath it, particles are unable to bounce, becoming absorbed in the dense atmosphere instead. Above this threshold, particles bounce unimpeded.

Particles that get lost correspond to an equatorial pitch angle within the range of the loss cone. Meanwhile, others persist in the Van Allen belts, maintaining their regular motion. The relationship between this minimum height and the loss cone is as follows [27]:

$$sin(\alpha_{loss}) = \sqrt{\frac{{\zeta_m}^3}{\sqrt{1+3(1-\zeta_m)}}}, \quad (3.1k)$$

$$\text{where } \zeta_\mu = \frac{R_E+h_m}{LR_E}$$

The value ζ_μ can be determined using the Earth's radius R_E and a specific L-shell. The minimum height, h_m, is set at 100 meters [28]. Consequently, this angle's values are based on the L-shell, which represents the groups of magnetic lines encircling the Earth.

3.2 Wave-Particle Interaction (WPI)

In the foundational understanding of a particle's motion in a magnetic field, the Lorentz equation serves as the primary descriptor. It precisely captures the force experienced by a charged particle due to both electric and magnetic fields. This description typically presumes an environment with consistent, unvarying fields. However, when the scenario incorporates electromagnetic waves, the narrative becomes more nuanced. Such waves contribute dynamic electric and magnetic field components that oscillate with time and space. In this augmented scenario, the Lorentz force, which dictates the particle's motion, draws influence not only from the Earth's static geomagnetic field but also from the oscillating wave fields. For this reason, and to aid the discussion on wave-particle interactions, the total fields can be distinguished into the wave components $\boldsymbol{E}^w$ and $\boldsymbol{B}^w$ and the static geomagnetic field $\boldsymbol{B}_0(\boldsymbol{r})$to form a new equation:

$$\frac{d\boldsymbol{p}}{dt} = q\left\{\boldsymbol{E}^w + \frac{\boldsymbol{p}}{m_s\gamma} \times [\boldsymbol{B}^w + \boldsymbol{B}_0(\boldsymbol{r})]\right\}, \quad (3.2a)$$

As the gyroradius of resonant energetic electrons tends to be significantly smaller than the spatial scales of B_0 variation at the L-shells under examination, the analysis can be simplified by considering a constant magnetic field. The electric and magnetic fields of an MR-whistler wave can be given by the following formulas [29]:

$$\boldsymbol{B}^w = \hat{\boldsymbol{x}}B_x^w cos\Phi + \hat{\boldsymbol{y}}B_y^w sin\Phi - \hat{\boldsymbol{z}}B_z^w cos\Phi, \quad (3.2b)$$

$$\boldsymbol{E}^w = -\hat{\boldsymbol{x}}E_x^w sin\Phi + \hat{\boldsymbol{y}}E_y^w cos\Phi - \hat{\boldsymbol{z}}E_z^w sin\Phi, \quad (3.2c)$$

with $\Phi(\mathbf{r}) = \int \omega dt - \int \boldsymbol{k} \cdot d\boldsymbol{r}$ being wave phase. For ease of analysis, the wave can be decomposed into two circularly polarized components, each rotating in the opposite direction since the whistler mode wave field exhibits natural elliptical polarization within the plasma environment.

$$\boldsymbol{B}_R^w = \frac{B_x^w + B_y^w}{2}[\hat{\boldsymbol{x}}cos\Phi + \hat{\boldsymbol{y}}sin\Phi], \qquad (3.2d)$$

$$\boldsymbol{B}_L^w = \frac{B_x^w - B_y^w}{2}[\hat{\boldsymbol{x}}cos\Phi - \hat{\boldsymbol{y}}sin\Phi], \qquad (3.2e)$$

Combining B_0 and equations (3.2b) and (3.2c) with the Lorentz force equation (3.2a) will yield a set of equations characterizing the movement of an energetic electron within a whistler-mode wave field that is obliquely propagating. However, this representation would lack practical utility, as it demands integration over time intervals smaller than a gyroperiod. In the context of typical magnetospheric conditions, the dynamics of resonant interactions result in cumulative alterations to both pitch angle and particle energy occurring over time scales significantly exceeding the gyroperiod. Consequently, it proves advantageous to reconfigure the equations of motion in a manner that enables the effective smoothing out of the rapid gyrations exhibited by the particles [29].

The gyro-averaged equations, for both first order and other cyclotron harmonic resonances in the presence of oblique whistler-mode waves have been initially established by 1984 [29]. More recently, a relativistic version of these equations was provided by Bortnik in his Ph.D. thesis [25]. By averaging over a single gyration period, the relativistic gyro-averaged equations of motion pertaining to a broad harmonic resonance indexed by *'m'* are derived.

Variation of the parallel and perpendicular momentum:

$$\frac{d_{p_z}}{dt} = {\omega_{\tau m}}^2 m_e {k_z}^{-1} sin\eta - \frac{1}{m_e\gamma}\frac{{p\perp}^2}{2\omega_H}\frac{\partial\omega_H}{\partial z}, \quad (3.2f)$$

$$\frac{d_{p\perp}}{dt} = -\left[\omega_1\left(\frac{p_z}{\gamma} - m_eR_1\right)J_{m-1}(\beta) - \omega_2\left(\frac{p_z}{\gamma} - m_eR_2\right)J_{m+1}(\beta)\right]sin\eta + \frac{1}{m_e\gamma}\frac{p_zp_\perp}{2\omega_H}\frac{\partial\omega_H}{\partial z}, \quad (3.2g)$$

Variation of the angle η which is the angle between the right circularly polarized component of the wave's magnetic field B_R^w and the perpendicular velocity $v_\perp$:

$$\frac{d\eta}{dt} = \frac{m\omega_H}{\gamma} - \omega - k_z\frac{p_z}{m_e\gamma}, \quad (3.2h)$$

Variation of the pitch angle:

$$\frac{d\alpha}{dt} = \frac{-m_e{\omega_{\tau m}}^2}{k_zp_\perp}\left(1 + \frac{cos^2\alpha}{mY-1}\right)sin\eta + \frac{1}{m_e\gamma}\frac{p_\perp}{2\omega_H}\frac{\partial\omega_H}{\partial z}, \quad (3.2i)$$

where:

$\omega_H = \frac{eB}{m_e}$ is the electron gyrofrequency, $\beta = \frac{k_xp\perp}{m_e\gamma\omega_H}$, $k_z = kcos\theta = \frac{\omega\mu}{c}cos\theta$, $k_x = ksin\theta$,

m_e is the electron rest mass and μ is the refractive index,

$${\omega_{\tau m}}^2 = (-1)^{m-1}{\omega_{\tau 0}}^2[J_{m-1}(\beta) - \alpha_1J_{m+1}(\beta) + \gamma\alpha_2J_m(\beta)],$$

J_i is the Bessel function of the first kind, order I and

$${\omega_{\tau 0}}^2 = \frac{\omega_1k_zp\perp}{\gamma m_e}, \alpha_1 = \frac{\omega_2}{\omega_1}, \alpha_2 = \frac{eE_z^w}{\omega_1p\perp}, R_1 = \frac{E_x^w+E_y^w}{B_x^w+B_y^w}, R_2 = \frac{E_x^w-E_y^w}{B_x^w-B_y^w}$$

$$Y = \frac{\omega_H}{\omega}$$

Following that, a discrepancy emerged in the formulas for perpendicular motion at the lth-order resonance. For this reason, a more comprehensive derivation for the rates was established for particles performing gyro-averaged motion. For resonance (cyclotron resonance) of order l with the wave [30]:

$$\frac{d_{p_\parallel}}{dt} = F_\parallel^w sin\eta - \frac{1}{2B}\frac{{p_\perp}^2}{\gamma m}\frac{\partial B}{\partial z}, \quad (3.2j)$$

$$\frac{d_{p\perp}}{dt} = F_\perp^w sin\eta + \frac{1}{2B}\frac{p_\parallel p_\perp}{\gamma m}\frac{\partial B}{\partial z}, \quad (3.2k)$$

$$\frac{d\eta}{dt} = \frac{lF_\theta^w}{p_\perp} cos\eta + \omega - \frac{k_\parallel p_\parallel}{\gamma m} - \frac{l\omega_{ce}}{\gamma}, \quad (3.2l)$$

$$\frac{d\alpha}{dt} = -\frac{F_\parallel^w}{p_\perp}\left(1 + \frac{cos^2\alpha}{lY-1}\right) sin\eta + \frac{p_\perp}{2\gamma mB}\frac{\partial B}{\partial z}, \quad (3.2m)$$

where the factors are:

$$F_\parallel^w = -e\left[E_z^w J_l + \frac{p_\perp}{\gamma m} B_R J_{l-1} - \frac{p_\perp}{\gamma m} B_L J_{l+1}\right], \quad (3.2n)$$

$$F_\perp^w = -e\left[E_R\, J_{l-1} + E_L\, J_{l+1} - \frac{p_\parallel}{\gamma m} B_R J_{l-1} + \frac{p_\parallel}{\gamma m} B_L J_{l+1}\right], \quad (3.2o)$$

$$F_\theta^w = -e\left[E_R\, J_{l-1} - E_L\, J_{l+1} - \frac{p_\parallel}{\gamma m} B_R J_{l-1} - \frac{p_\parallel}{\gamma m} B_L J_{l+1} + \frac{p_\perp}{\gamma m} B_z^w J_l\right], \quad (3.2p)$$

with:

$Y = \frac{\omega_{ce}}{\gamma\omega}$, $\omega_{ce} = \frac{eB}{m}$ the non-relativistic electron gyrofrequency and $\beta = \frac{k_\perp p_\perp}{eB}$

When these factors are set to zero, the particle undergoes adiabatic motion. Within the simulation, two sets of equations, specifically the ones labeled as (3.2f), (3.2g), (3.2h), (3.2i) and (3.2j), (3.2k), (3.2l), (3.2m), are utilized in individual routines. The user has the option to select which set of equations should be employed. It is also important to note that when a wave is in resonance with an energetic particle [31]:

$$\frac{d\eta}{dt} = m\omega_H - \omega - k_z u_z \simeq 0$$

When the wave's frequency shifts due to the Doppler effect and becomes equal to or a multiple of the particle's gyrofrequency (an integer multiple), the particle temporarily perceives the electro-

magnetic fields of the wave as static. In other words, when these frequencies match, it leads to resonance. Particles that interact with such a wave and enter this resonance can undergo acceleration significant enough to break free from their periodic motion. This can result in them impacting the upper atmosphere, where they interact with other particles and dissipate their energy [32].

3.3 Numerical Integration of Differential Equations

In the realm of differential equations, analytical solutions often remain out of reach, particularly for complex or non-linear systems. In such cases, numerical methods are employed to provide approximations of these solutions. At the foundation of numerical methods for differential equations lies *Euler's method*. It offers a direct approach to approximating solutions by utilizing the slope of the solution curve at the beginning of an interval to predict its value at the end. For a scalar or vector *y* determined by a differential equation, while the exact equation *y* might be unknown, the time rate of *y* is known. Given the initial condition at time t_0 as y_0 and the equation that captures the evolution of *y* over time, the challenge is to predict the subsequent values of *y* without detailed knowledge of its governing equation. Euler's method presents a straightforward approach to this challenge. For differential problems defined as:

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0$$

The formula for Euler's method is:

$$y_{n+1} = y_i + hf(t_n, y_n)$$

Where:

- h is the chosen step size.
- t_n is the current time value.
- y_n is the current function value.

While Euler's method is conceptually simple and easy to implement, its accuracy is often insufficient for many practical problems, especially if the function is rapidly changing or the system is stiff. It has a local truncation error proportional to the square of the step size and a global error proportional to the step size. Therefore, finer steps are required for better accuracy, leading to a higher computational cost.

Building on the foundational ideas set by Euler's method, the *Runge-Kutta* family introduces more nuanced techniques. By considering values at multiple stages within an interval, these methods aim to provide a richer approximation of the solution. Among these methods, the fourth order Runge-Kutta, often abbreviated as *RK4* for simplicity, stands out. It evaluates the function at four distinct stages within each interval, as detailed previously. This multi-stage evaluation makes RK4 especially adept at handling a wide range of problems. Its increased accuracy, due to a global error of O(h^4), makes it preferable for many applications despite the higher computational cost compared to simpler methods like Euler's. Given its enhanced accuracy, RK4 is extensively used in particle tracing, especially where trajectories can be influenced by intricate forces. Its ability to effectively handle rapidly changing conditions makes it suitable for simulating scenarios with complex interactions, such as the wave-particle interaction.

Mathematically, starting with the same hypothesis again:

$$\frac{dy}{dt} = f(t, y) \text{ και } y(t_0) = y_0$$

For step sizes $h > 0$ the formula for the RK4 method, the value of y for the subsequent step will be:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h$$

This formulation suggests that each succeeding value, y_{n+1}, results from adding the previous step's value, y_n, to a certain weighted average. This weighted average is derived from four distinct slopes pertinent to the equation:

- $k_1 = f(t_n, y_n)$, the slope at the step's outset, analogous to the Euler method.
- $k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right)$, the slope at the midpoint of the step, influenced by slope k_1.
- $k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right)$, the slope at the midpoint of the step, but it's based on k_2.
- $k_4 = f(t_n + h, y_n + hk_3)$, slope towards the step's conclusion, derived from k_3

In the RK4 method, when computing the weighted average, the slopes calculated at the midpoint of the method receive greater emphasis. Following this, a graphical representation of the problem in Figure 20 can provide enhanced clarity and understanding:

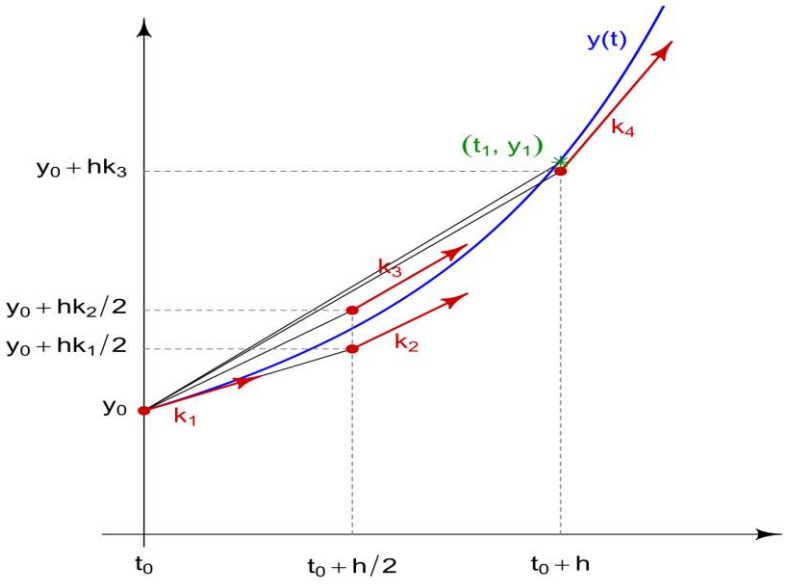


Figure 20: Graphical representation of the method RK4.

These methods have found widespread application in various scientific domains. While Euler's method is often introduced as a pedagogical tool due to its simplicity, its use in critical scientific

problems is limited because of its lower accuracy. In contrast, RK4, owing to its balance between accuracy and computational effort, is frequently chosen for more serious scientific computations.

In the domain of particle tracing, accurate and efficient methods are imperative to simulate and predict the trajectories of particles. The RK4 method stands as a particularly favored technique unlike simpler methods that might miss finer details of particle motion, especially in regions with complex force fields or non-linear interactions. By employing intermediate steps and weighted averages, the RK4 method captures the dynamics with enhanced accuracy. As particles traverse their paths influenced by various forces or fields, the intermediate calculations of RK4 ensure that the traced trajectories are both smooth and closely aligned with the actual physics of the system. For these reasons, the RK4 method is employed in this particle simulation, and its programming details are elaborated in the following section.

Chapter 4

Algorithm

The code under analysis in this thesis pertains to the initial commit within the "*Thesis*" branch of the Git repository and may undergo future modifications for its improvement.

4.1 Original code

The original code is written in *Python* by the PhD candidate Tourgaidis Stylianos [1]. Python is a programming language that is very popular and useful due to its rich libraries and user-friendly nature. However, in problems that involve heavy computation and handling large amounts of data, which may require multiple runs, it is more appropriate to use another solution. In scientific computing the most common choices when it comes to programming languages are *FORTRAN, C, or C++* in combination with *APIs* (Application Programming Interfaces) that enable parallel execution.

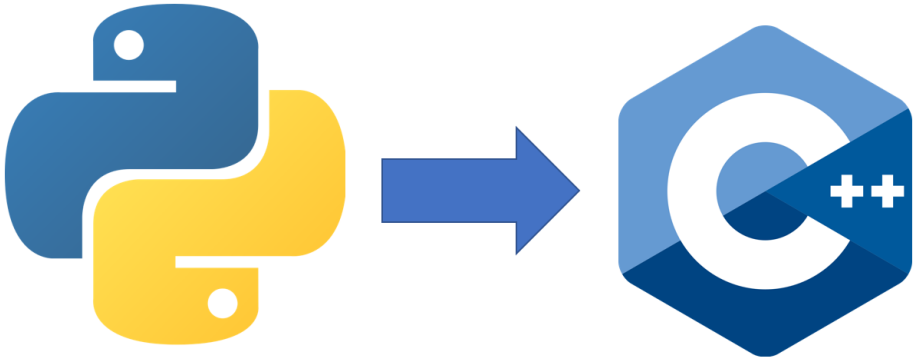


Figure 21: In many cases, a well-designed C++ algorithm can significantly enhance performance compared to its Python counterpart.

Before starting with the program analysis, it's important to introduce all the constants and program parameters. For a better organization during the program's execution, these values have been conveniently placed in the "*constants.h*" header file within their respective namespaces. This ensures that when these values are referenced, it's evident that they remain constant throughout the program. The universal constants required for program execution are listed within an appropriate namespace:

```
//--Universal parameters--//
namespace Universal
{
    const real Re = 6378137; // Earth mean radius in m
    const real c = 2.997956376932163e8; // Speed of light in m/s
    const real m_e = 9.10938291e-31; // Electron mass in kg
    const real q_e = 1.602176565e-19; // Electron charge in C
    const real q_i = 1.602176565e-19; // Electron charge in C
    const real m_O = 2.67616E-026; // Oxygen mass in kg
    const real m_H = 1.6726E-027; // Hydrogen mass in kg
    const real m_He = 6.6904E-027; // Helium mass in kg
    const real e_el = 5.105396765648739e5;
    const real eps0 = 8.854187817e-12;
    const real mu_0 = M_PI*4*pow(10,-7);
    const real D2R = M_PI/180;
    const real R2D = 1/D2R;
    const real B0 = 3.12*pow(10,-5); // Mean value of the field on the equator
    const real ne_0 = 3*pow(10,6); // 10/cm-3 => 10*10^6m/m-3
}
```

Next, the program's parameters are organized within their respective namespaces, including *Wave*, *Distribution*, *Aeq_dstr*, *Lat_dstr*, *Eta_dstr*, *Ekin_dstr*, *Simulation*, and *Satellite* namespaces. To configure the simulation, it is mandatory to modify the parameters before compiling the code. First, the electromagnetic wave parameters:

```
//--Wave parameters--//
namespace Wave
{
    const real f_wave = 2000; // Wave frequency in Hz
    const real w_wave = 2*M_PI*f_wave; // Wave angular frequency
    const real m_res = 1; // WPI resonance number
    //Only ray tracing
    const real pwr = pow(10,0); // Poynting flux [W/m 2]
    const real pulse_duration = 0.1; // Wave pulse duration in seconds
    //Only for omnipresent wave
    const real theta0_deg=0.001; // Initial wave norml angle
    const real theta0 = theta0_deg*Universal::D2R;
    const real By_wave = 1*pow(10,-9);
}
```

Subsequently, the parameters for the various distributions, as well as their associated standard deviations and means:

```
//--Distribution parameters--//
namespace Distribution
{
    const real L_shell = 2; // L_shell of particle. constant for now
```

```
    const int64_t population = 100000;
}

// P.A dstr
namespace Aeq_dstr
{
    const real start_deg = 0;
    const real end_deg = 180;
    const real mean = 90; // Mean of the normal dstr
    const real stdev = 20; // Standard deviation of the normal dstr
    const real steps = 10; // To distribute evenly
    const real value_deg = 37.6708693574307; // If single values are used
    const real value = value_deg * Universal::D2R;
}

// Latitude dstr
namespace Lat_dstr
{
    const real start_deg = -90;
    const real end_deg = 90;
    const real mean = 0; // Mean of the normal dstr
    const real stdev = 20;
    const real domain_step = 0.001; // Latitude domain range precision, degrees
    const real steps = 10; // To distribute evenly
    const real value_deg = 16.2782913046611; // If single values are used
    const real value = value_deg * Universal::D2R;
}

// Eta dstr
namespace Eta_dstr
{
    const real start_deg = 0;
    const real end_deg = 360; // Issue if eta=0 or 360?
    const real mean = 180; // Mean of the normal dstr
    const real stdev = 20; // Standard deviation of the normal dstr
    const real steps = 10; // To distribute evenly
    const real value_deg = 5.01547617274519; // If single values are used
    const real value = value_deg * Universal::D2R;
}

// Ekin dstr
namespace Ekin_dstr
{
    const real start = 100; // keV
    const real end = 1350; // keV
    const real mean = 600; // Mean of the normal dstr
    const real stdev = 20; // Standard deviation of the normal dstr
    const real steps = 10; // To distribute evenly
    const real value = 850.508389025729; // If single values are used
}
```

Finally, the parameters essential for the simulation process including details such as time settings and satellite position:

```
//--Simulation parameters--//
namespace Simulation
{
    const real hm = 100*pow(10,3); // Minimum allowable mirroring altitude in m
    const real zm = (Universal::Re + hm)/(Distribution::L_shell*Universal::Re);
    const real alpha_lc = asin(sqrt(pow(zm,3)/sqrt(1+3*(1-zm)))); // Loss cone an-
gle in radians
    const real h = 0.00001; // Runge kutta stepsize
    const real puls_dur = int(Wave::pulse_duration/h); // Pulse duration (stepsize)
    const int num_threads_wpi_bell = 16;
    const int num_threads_wpi_li = 16;
    const int num_threads_nowpi = 8;
}

//--Satellite parameters--//
namespace Satellite
{
    const real latitude_deg = 0; // Latitude of the satellite
    const real L_shell = Distribution::L_shell; // L_shell of the satellite
}
```

This header, containing these crucial parameters, will be accessible to all parts of the simulation code. In the following sections, it is explained how these parameters are employed in the simulation process.

4.2 Generate Particle Distributions

A separate algorithm in C++ was developed to create the distributions that can generate particle populations distributed in terms of:

- Equatorial PA (*aeq*)
- Latitude (*latitude*)
- Eta parameter i.e., angle between vectors B_R^w and $v_\perp$ as elaborated in Section 3.2 (*eta*)
- Energy (*Ekin*)

Of particular interest is the PAD as elaborated in Section 2.4, which is commonly used to characterize a population of particles, as its variation can indicate a probable wave-particle interaction or other influences stemming from natural phenomena. The four variables mentioned above can be distributed uniquely in four different ways. Specifically, they can be distributed in:

- Linear spacing
- Uniform distribution
- Normal distribution
- Fixed values

Each distribution mode is thoroughly analyzed, along with the specific cases in which it proves to be useful:

Linear spacing

The particles will take on values between the specified initial and final values, which can be set in the simulation parameters. This distribution adheres to the formula *(end - start) / (population - 1)*, resulting in a symmetric particle distribution. Specifically, when placing each particle at a position, its symmetric counterpart over the equator will also be placed. This distribution helps in symmetrically employing whole ranges of parameter values. Figure 22 illustrates how latitude linear spacing works.

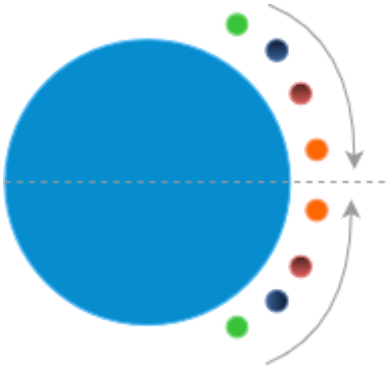


Figure 22: Particles are positioned symmetrically, with particles of the same color sharing identical latitudes and velocities, but with opposite signs.

Uniform distribution

Uniform distributions, also known as rectangular distributions, are a type of probability distribution where all outcomes within a given range are equally likely to occur. In a uniform distribution, each value has the same probability of being selected, and there are no preferred or dominant values. To achieve a uniform distribution, the particles must be randomized, with each one initialized within the specified boundaries in a consistent manner. These boundaries are defined in the code as variables. Within this range, every value has an equal chance of being selected and assigned to a particle. Mathematically, a continuous uniform distribution is defined over a specific interval *[a, b]*, and the probability density function or else *PDF* is given by:

$$f(x) = \frac{1}{b-a}, \text{for } a \leq x \leq b$$

$$f(x) = 0, \text{otherwise}$$

where:

- *f(x)* is the PDF at point *x*.
- *a* is the lower bound of the distribution.
- *b* is the upper bound of the distribution.

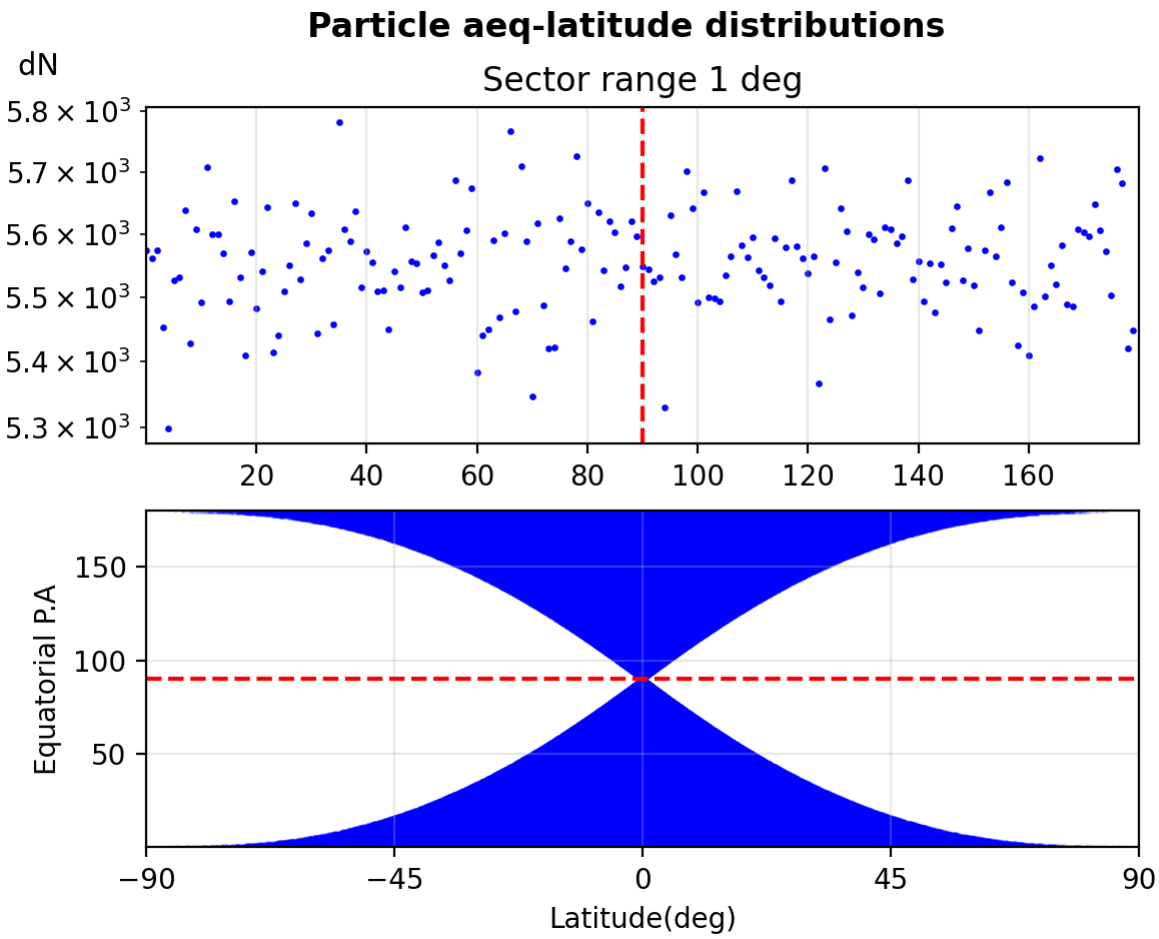


Figure 23: 10^6 *particles uniformly distributed in both latitude and equatorial PA (programmatically in "distribution.cc"). The top diagram presents particle counts within 1-degree PA intervals, yielding consistent particle counts across all PA. The bottom diagram illustrates the correlation between equatorial PA and latitude in degrees. All latitude and PA values have an equal probability of occurrence.*

Normal distribution

This distribution is commonly observed in nature and is particularly relevant in the case of PADs when realistic distributions are required. In a normal particle distribution, the particles' values tend to cluster around the mean or average value, with fewer particles at the extremes. The distribution is symmetric, meaning that the data is evenly distributed around the mean, and it follows a specific probability density function known as the *Gaussian* function.

The Gaussian function is mathematically defined as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$ where:

- *f(x)* is the PDF of the distribution at point *x*.
- μ is the mean or average value of the distribution.

- σ is the standard deviation, which measures the spread or dispersion of the data around the mean.

In the code the distribution parameters are the standard deviation and mean values, allowing for flexibility in shaping the distribution.

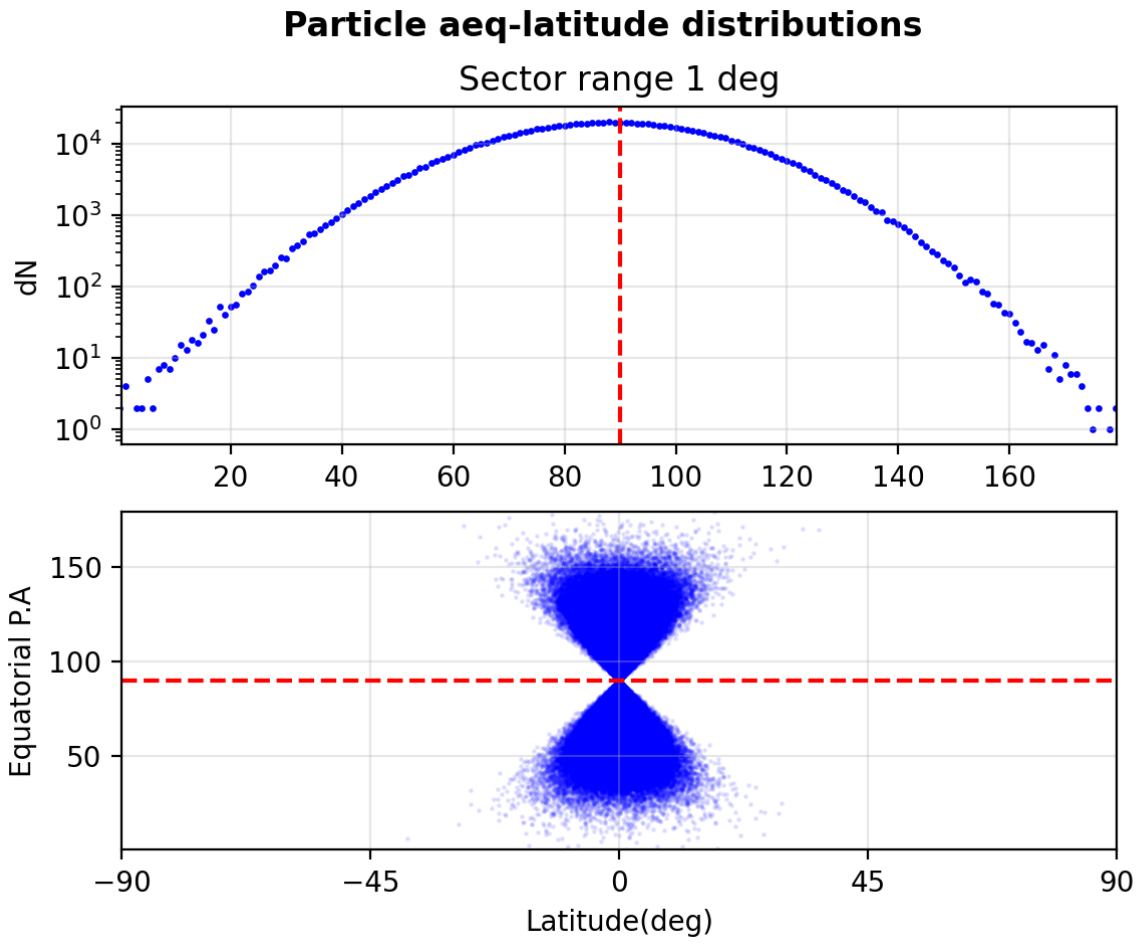


Figure 24: 10^6 *particles normally distributed in both latitude an equatorial PA (programmatically in "distribution.cc"). The top diagram presents particle counts within 1-degree PA intervals, yielding a normal PAD with a standard deviation of 20, centered around mean values of 90 for PA and 0 for latitude. The bottom diagram illustrates the correlation between the equatorial PA and the latitude of the particle distribution, both measured in degrees. It's evident that while moving towards higher latitudes and away from the 90-degree equatorial PA, the particle count smoothly decreases.*

Fixed value distribution

Simple value assignment, useful for testing the behavior of specific particles and for debugging.

Determine the valid domain for the particle

To determine the valid range of values for the pitch angle and latitude of each particle, it is mandatory to establish constraints, as not all values are acceptable. Initially, the pitch angle of a particle can vary from 0 to 180 degrees, while its latitude can range from -90 to 90 degrees. However, these two variables are also related through the following formula:

$$sin^2 a(\lambda) = \frac{B(\lambda)}{B_{eq} sin^2 a_{eq}},$$

where λ is the particle's latitude, α is the PA and a_{eq} the equatorial PA.

To determine the range of valid values programmatically, one can utilize a *brute force* method, which simply involves testing various values within a range to identify those that "break" the function. The primary objective is to pinpoint the boundaries where errors occur. The equation can be transformed into the one shown within the plot of Figure 25, and the conditions that lead to errors are the following:

1. sin a >= 1.
2. sin a <= -1.
3. sin a = 0, indicating that the pitch angle itself is 0, resulting in a vertical momentum of 0. This condition leads to errors in the program, as certain parameters, such as the parameter *a2* described in Section 3.2, are forced to be expressed as divisions by zero.

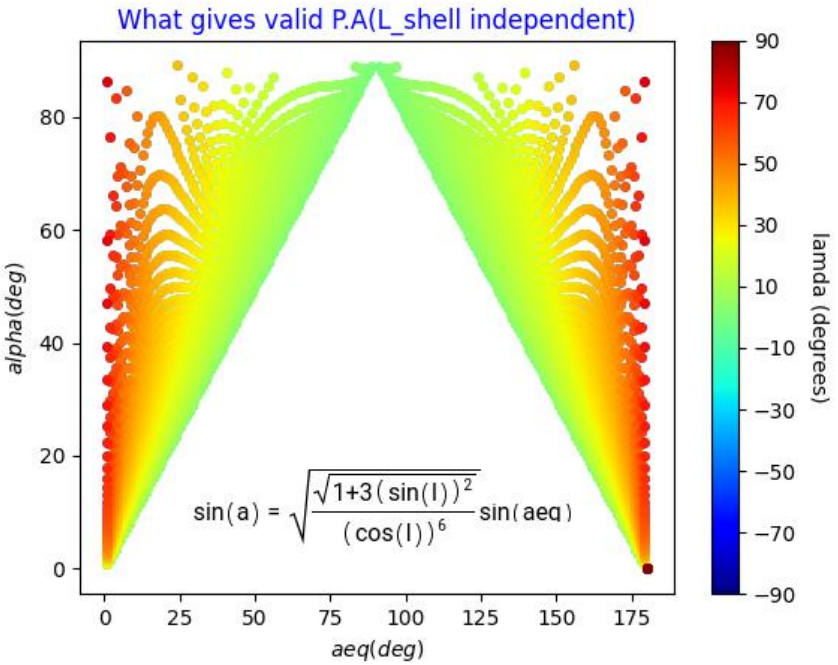


Figure 25: Scatter plot representing the domain of validity for latitude and PA values for a particle in the Earth's Magnetic Field.

To address this concern, a function has been developed, which takes the equatorial pitch angle of a particle as input and estimates the valid range for the latitude of the particle. By gradually increasing the latitude, the function identifies the point at which any of the aforementioned errors occur. This boundary indicates that the latitude for a particle with that pitch angle is not defined beyond that value. The step size employed during each iteration is a critical factor, striking a balance be-

tween accurate calculation of the latitude and computational efficiency. A reasonable step size is 0.001 degrees but can be changed as it is a parameter of the program. This function is executed for each particle just before assigning the latitude value, ensuring that only particles with valid latitude ranges are included in the simulation. This practice is important to prevent unnecessary usage of computational resources, as it guarantees that invalid particles, resulting in *NAN* values (Not-A-Number), are not included in the population

The program for the creation of a particle distribution is written in C++ under the name "*distribution.cc"* and utilizes command line interface to generate the desired distribution. For each distribution, whether it is for pitch angle, latitude, energy, or the eta parameter, the program follows a standardized procedure. The C++ library *random* is included in the program and provides convenient routines for uniform and normal distributions. To streamline the program execution, constraints are established for the command line variables. The program analysis follows:

```
int main(int argc, char **argv)
{

    std::string string_evenly   = "evenly";
    std::string string_normal   = "normal";
    std::string string_uniform  = "uniform";
    std::string string_constant = "constant";

    // Validate program arguments
    if(argc!=5) throw std::invalid_argument("\nSet command line arguments from the
list:(evenly, uniform, normal, test) to distribute the particles.\nargv[1] is for
P.A, argv[2] is for latitude, arg[3] is for eta, argv[4] is for Ekin");
    for(int arg=1;arg<=4;arg++)
    {
        if( (argv[arg]!=string_evenly) && (argv[arg]!=string_uniform) &&
(argv[arg]!=string_normal) && (argv[arg]!=string_constant) )
            throw std::invalid_argument("\nArgument variables don't match any of
the program's possible implementations.\nSet 4 command line arguments from the
list:(evenly, uniform, normal, test) to distribute the particles.\nargv[1] for P.A,
argv[2] for latitude, arg[3] for eta, argv[4] for Ekin");
    }
    std::cout<<"\n\nParticle population: " << Distribution::population;
    std::cout<<"\nDistributed as: |From    To|"<<"    (evenly, uniform, normal,
constant) \n";
    std::cout<<"\nEquatorial P.A: |"<<Aeq_dstr::start_deg<<"
"<<Aeq_dstr::end_deg<<"|      "<< argv[1];
    std::cout<<"\nLatitude:       |"<<Lat_dstr::start_deg<<"
"<<Lat_dstr::end_deg<<"|     "<< argv[2];
    std::cout<<"\nEta:            |"<<Eta_dstr::start_deg<<"
"<<Eta_dstr::end_deg<<"|      "<< argv[3];
    std::cout<<"\nEkin:           |"<<Ekin_dstr::start<<" "<<Ekin_dstr::end<<"|
 "<< argv[4]<< std::endl;
```

Subsequently, the *Particles* structure is employed to create a vector that stores the initial state of the particles. A detailed description of the particle structure can be found in Section 4.4. At this point, it

is enough to mention that the particle states are initialized using the distributions generated by the method explained in this chapter.

```
Particles single; // Single particle struct.
// Vector of structs for particle distribution.
std::vector<Particles> dstr(Distribution::population, single);
real latitude0,aeq0,eta0,Ekin0;
int id0;
latitude0=aeq0=eta0=Ekin0=0;
std::random_device seed; // Random seed.
std::mt19937 generator(seed());  // PRNG initialized with seed.
int p=0;
int aeq_count = 0 ;
int latitude_count = 0 ;
int eta_count = 0;
int Ekin_count = 0;
//------------------- DISTRIBUTE PA -----------------//
for(int particle_count=0; particle_count<Distribution::population; parti-
cle_count++)
{
  id0 = particle_count;
  // Evenly with goal: symmetry
  if(argv[1]==string_evenly) {
    real aeq_step_d=(Aeq_dstr::end_deg -Aeq_dstr::start_deg)/(Aeq_dstr::steps-1);
    aeq0 = (Aeq_dstr::start_deg + aeq_count*aeq_step_d) * Universal::D2R;
    aeq_count++;
  }
  // Uniformly with goal: randomness
  else if (argv[1]==string_uniform) {
   std::uniform_real_distribution <real>
   uniform_aeq(Aeq_dstr::start_deg, Aeq_dstr::end_deg);
   aeq0  = uniform_aeq(generator) * Universal::D2R;
  }
  // Normally with goal: reach normal dstr in logarithm scale
  else if (argv[1]==string_normal) {
    real number;
    do {
       std::normal_distribution <real> normal_aeq(Aeq_dstr::mean, Aeq_dstr::stdev);
       number = normal_aeq(generator);
       }while(number<=0 || number>=180); //Until valid aeq=(0,180).
    aeq0 = number * Universal::D2R;
  }
  // Constant with goal: testing
  else if (argv[1]==string_constant) {
    aeq0 = Aeq_dstr::value;
}
```

The subsequent objective is to establish the valid range for the latitude corresponding to a specific equatorial pitch angle. Below is the function responsible for this task, along with the function call.

```
// Member function to find valid latitude range
void latitude_domain(real aeq0, real &latitude_start_d, real &latitude_end_d)
{
    const real Beq0 = Bmag_dipole(0);
    latitude_end_d     = 0;
    latitude_start_d   = 0;
    real salpha0;
    do
    {
        latitude_end_d += Lat_dstr::domain_step;
        // Gradually increase and check if salpha0 is valid.
        // ("Brute Force domain validation")
        real Blam0   = Bmag_dipole(latitude_end_d*Universal::D2R);
        salpha0      = sin(aeq0)*sqrt(Blam0/Beq0);

    }
    while( (salpha0<=1) && (salpha0>=-1) && (salpha0!=0) ) ;
    latitude_end_d   -= Lat_dstr::domain_step; // Last (positive) valid value.
    latitude_start_d  = -latitude_end_d; // First (negative) valid value.
}
```

The next function is called to estimate the domain of validity for the particle's latitude as described in the previous pages:

```
//-------------Find Latitude domain-------------//
real latitude_end_d, latitude_start_d;
latitude_domain(aeq0, latitude_start_d, latitude_end_d);
```

Next, the same procedure is repeated to assign values for *latitude*, *eta*, and *Ekin*. Depending on the command line parameters, particles are distributed using normal, uniform, fixed-step, or fixed-value distributions for each specific variable. Following that, initial values are also assigned to the rest of the particle variables to finalize the overall initial state of the particle. These are deducted using the variables that were mentioned above and through the member function *initialize():*

```
void initialize(int id0, real eta0, real aeq0, real latitude0, real Ekin0, real
zeta0, real time0);
```

This function receives the recently set values as input and proceeds to assign them to the structure of each particle. A more comprehensive description of this process is provided in Section 4.4 where the particle structure is analyzed. Essentially, when all initial input parameters (aeq, latitude, eta, Ekin) are defined for the particle, this initialization function is called:

```
dstr[p].initialize(id0, eta0,aeq0,latitude0,Ekin0,0,0);
```

This procedure is being repeated for all particles in order to initialize their state. Once all the particles are defined, they are saved in *.h5* format for future use in various simulations.

4.3 Generate Electromagnetic Wave

The characteristics of the wave at a specific moment are extracted from a *.csv* file that was generated earlier using a ray tracing algorithm, which is not discussed in this document. This program, which is under the name "*interpolate.cc"*, enables the calculation of the wave's trajectory during its propagation. After reading the file, the wave's values are subjected to linear interpolation at each multiple of the step size *h*. This step size, which is a simulation parameter, can be adjusted as needed. Essentially, it determines the temporal resolution for the iterative Runge-Kutta process that was described in Section 3.3. As a result, the wave's values are obtained at the specific temporal intervals dictated by this numerical method's evolution.

The interpolation function takes the following inputs:

- A vector containing the times at which the Runge-Kutta method is being executed.
- A vector containing the actual times of the traced ray, used for linear interpolation.
- A vector with the corresponding wave values at those times, denoted as y(t).

The following function is defined in the source code with the name "*interpolation.cc*".

```
std::vector <real> interpolate(std::vector <real> time_new, std::vector <real>
timef, std::vector <real> vector_csv)
{
    std::vector <real> vector_int;
    int size = time_new.size();
    vector_int.reserve(size);
    int x=1;
    real yp;
    for(std::size_t p=1; p<time_new.size(); p++)
    {
        if(timef[x-1]<=time_new[p] && time_new[p]<=timef[x]) {
            //Independent variable: time_new -> x
            //x1<=xp<=x2
            //x1,y1 coordinates Below the known xp value
            //x2,y2 coordinates Above the known xp value
            yp = vector_csv[x-1] + ((time_new[p] - timef[x-1])*(vector_csv[x] -
            vector_csv[x-1])/(timef[x]-timef[x-1]));
            vector_int.push_back(yp);
        }
        else
        {
            x++;
            p--;
        }
    }
```

```
    vector_int.insert(vector_int.begin(),vector_csv[0]);
    return vector_int;
}
```

After performing the interpolation, various wave fields and other calculated values are obtained for convenience, outside the simulation. Eventually, the wave is stored in *.h5* format, ready to be read and interact with the particles.

4.4 Structures

For better organization in the code, two structures have been created:

1. **Particle Structure**: This structure is intended for storing both the initial and potentially subsequent states of the particles.
2. **Satellite-Detector Structure**: This structure simulates a detector that can be positioned at any desired latitude and can be used to capture the particles of the simulation. It functions by detecting particles when they pass through a specified location.

Particles Structure

Considering that particles are characterized by multiple attributes, it is appropriate to employ data structures to accommodate them. When managing groups of particles, a vector of such structures is required. The structure includes an initialization function which uses some input arguments to define the overall state of the particle. These arguments are sourced from the generated distribution file with a procedure discussed previously in Section 4.2. In simpler terms, these values represent the equatorial PA, the particle's latitude, its kinetic energy, and finally, the angle between the vectors B_R^W and $v_\perp$ as it was described in Section 3.2. Using these values, it is possible to compute various particle properties, such as the PA (not equatorial), the velocity, the momentum, the adiabatic constant, and other vital particle parameters that define its overall state. Essentially, this initialization function sets the initial variable values (y0) for the starting time (t_0), in the context of the RK4 method detailed in Section 3.3. This happens within the file "*struct_Particles.cc*", as detailed below:

```
void Particles::initialize(int id0, real eta0, real aeq0, real latitude0, real
Ekin0, real zeta0, real time0)
{
    int k;
    const real Beq0 = Bmag_dipole(0);
    real Blam0      = Bmag_dipole(latitude0);
    real salpha0    = sin(aeq0)*sqrt(Blam0/Beq0);
    if(aeq0*Universal::R2D>90)   k=1;
    else                         k=0;
    alpha0 = pow(-1,k)*asin(salpha0)+k*M_PI
    // Find momentum from energy.
    real Ejoule0=1.602176487E-16*Ekin0; // Kev to Joule
    real gama0=(Ejoule0/(Universal::m_e*pow(Universal::c,2))) + 1;
    real speed0=sqrt( 1 - (1/pow(gama0,2)) ) * Universal::c;
    // Assign initial state.
```

```
    this->id0 = id0;
    this->latitude0 = latitude0;
    this->aeq0   = aeq0;
    this->eta0   = eta0;
    this->alpha0 = alpha0;
    this->upar0  = speed0*cos(alpha0) ;
    this->uper0  = speed0*sin(alpha0) ;
    this->pper0  = gama0*Universal::m_e*uper0 ;
    this->ppar0  = gama0*Universal::m_e*upar0 ;
    this->zeta0  = zeta0 ;
    this->M_adiabatic0 = (pper0*pper0) / (2*Universal::m_e*Blam0) ;
    this->Ekin0  = ((gama0-1)*Universal::m_e*Universal::c*Universal::c)*6.2415e15;
    this->time0  = time0 ;
    // Particle is trapped, and won't escape until conditions are met.
    this->trapped  = true;
    this->escaped  = false;
    this->negative = false; // May it develop negative P.A
    this->nan      = false; // May it develop NaN state
    this->high     = false; // May it develop NaN state
}
```

A member function can also be defined which will store the state of the particles when they escape:

```
void Particles::escaping_state(int id, real new_latitude, real new_aeq, real
new_alpha, real new_time)
{
    id_lost = id;
    latitude_lost = new_latitude;
    aeq_lost = new_aeq  ;
    alpha_lost = new_alpha;
    time_lost = new_time;
}
```

There's also a function designed to save values when necessary. It's mainly used for debugging but also helps in tracking specific particles of interest. For instance, it records the largest shifts in energy and pitch angle, along with the corresponding times they occur.

```
void Particles::save_state(int p, real max_dEkin, real maxEkin_time, real  max_dPA,
real maxdPA_time, real min_detadt, real min_detadt_time)
{
    saved_id = p;
    saved_max_dEkin = max_dEkin;
    saved_maxEkin_time = maxEkin_time;
    saved_max_dPA = max_dPA;
    saved_maxdPA_time = maxdPA_time;
    saved_min_detadt = min_detadt;
    saved_mindetadt_time = min_detadt_time;
}
```

For the particles, two Boolean values (true or false) were designated:

- **Trapped Status**: Initially, every particle entering the simulation is considered trapped between the magnetic poles, thus assigned a value of true.
- **Escape Status**: This represents particles that eventually manage to break free from the trapped motion.

Though these two Boolean values might appear to be inverse counterparts, they serve distinct roles within the algorithm. A trapped particle might, over time, meet a certain condition known as the loss cone condition. However, meeting this condition doesn't imply the particle will instantly escape. Instead, it indicates a potential for future escape. The actual escape occurs when the particle reaches the previously described mirror point. Essentially, if the numerical calculations of the RK4 predict that the particle will next gain momentum or velocity in the opposite direction, it can then be deemed appropriate to exclude it from the population. So, while a particle might cease to be trapped, it doesn't automatically escape. The transition occurs as the particle approaches the point of bouncing.

Once this is observed, its simulated movement is promptly halted, and its identification number *id* is recorded. In this step it is important to respect the relationship between the minimum mirroring altitude and the loss cone angle which is calculated and conditioned in the code. The specific condition is outlined below:

```
// Check Trapping:
// True if P.A is less than the loss cone angle(for southward particles too)
// While aeq<loss cone P.A for this L_shell the particle continues oscillating
if( (0<new_aeq && new_aeq<Simulation::alpha_lc) || (new_aeq>M_PI-
Simulation::alpha_lc && new_aeq<M_PI) ) {
   particle.trapped = false;
    // If no longer trapped, don't break immediately
    // The particle will follow its trajectory
    // until reaching the moment just before changing its direction of motion, at
which point it will precipitate.
}

// Check Precipitation:
// If no longer trapped, and is about to change direction
if(!particle.trapped && (ppar*new_ppar<0) ) {
   particle.escaping_state(p, latitude, aeq, alpha, time);
   particle.escaped = true;
   std::cout<<"\n\nParticle(E) "<<p<<" escaped with aeq " <<aeq*Universal::R2D<< "
   at time " << time ;
break;
}
```

Additionally, three binary values are included to ensure the simulation runs correctly. The pitch angles of particles should fall within the permissible range of 0 to 180 degrees.

Detector Structure

The condition that signifies a particle's passage through the detector is based on *latitude* and *L-shell* constraints. Specifically, if the product of the change in latitude between the particle's position

and the satellite's position from one state to the next is negative, it suggests the particle has moved through the satellite. For instance, if a particle was previously beneath the satellite and then moves above it in the subsequent state, this change results in a negative value. It's important to note that in the current version of the code, the L-shell of a tracked particle is assumed to be constant. In these simulations, the detector recognizes particles from all potential pitch angles, covering a range of 180 degrees. Therefore, the device is referred to as ODPT, signifying everything described in Section 2.5. The detector structure is defined in the file *"struct_Telescope.cc"* and is the following:

```
// Constructor - Initialize position of satellite
Telescope::Telescope(real lat, real L) : latitude_deg(lat), L_shell(L) {}

void Telescope::resize(size_t size){
    // Resize the vectors to the initial size
    latitude.resize(size);
    alpha.resize(size);
    aeq.resize(size);
    eta.resize(size);
    time.resize(size);
    id.resize(size);
}
// Returns true if particle crossed satellite
bool Telescope::crossing(real p1_latitude, real p2_latitude, real p_L_shell){
    bool crossed = false;

    if(p_L_shell == L_shell) {
        if ((p1_latitude - latitude_deg) * (p2_latitude - latitude_deg) < 0) {
        // if particle was below and then above the sat, product would be negative
            crossed = true;
        }
    }
    return crossed;
}
// Store detected particles in satellite's memory
void Telescope::store(int id, real latitude, real aeq, real alpha, real time){
    this->id.push_back(id);
    this->latitude.push_back(latitude);
    this->alpha.push_back(alpha);
    this->aeq.push_back(aeq);
    this->time.push_back(time);
}
```

When the Boolean value *crossed* is set to true, it indicates that a particle has passed through the detector and its state could be saved for further analysis. The particle's state encompasses attributes like its velocity, pitch angle, energy, and so on when it moves through the satellite detector. However, saving these states consumes memory during the execution and in the disk later. For this reason, it is recommended to save only necessary data for each simulation and, importantly, to avoid saving data that could be derived later during the post processing procedure.

4.5 Simulation

The algorithm takes two inputs as they were explained earlier: a particle distribution and an electromagnetic wave. To briefly summarize, the procedure unfolds as follows:

1. Generate a particle distribution as described in 4.2.
2. Generate Electromagnetic Wave as described in 4.3.
3. Input both the particle distribution and the wave data into the code to simulate wave-particle interaction and:

    - Detect and eliminate particles lost during the interaction process.
    - Store the raw data.

4. Repeat step 3 using the same particle distribution, but this time, modify or even eliminate the wave component. For a detailed explanation, refer to Section 4.6.
5. Carry out post-processing on the resulting data and create visual representations, such as diagrams and movie files, to compare these two simulations of step 3 and 4. An in-depth description of this process follows in Section 4.7.

To execute the simulations of the steps 3 and 4, two separate implementations have been developed, each living in its dedicated branch within the Git repository. One utilizes distributed memory computing and can be found in the *MPI* branch, while the other leverages shared memory computing and is located in the *OpenMP* branch. This section focuses into the simpler of the two solutions, the one that employs OpenMP. It offers a detailed explanation of the primary source code, "main.cc."

The initial step involves referencing the libraries along with the developed structure and function headers that are utilized in the program.

```
#include <cmath>
#include <iostream>
#include <cinttypes>
#include <vector>
#include <algorithm>
#include <array>
#include <random>
#include <iomanip>
#include <omp.h>
#include <chrono>
#include <cstdlib>
#include <ctype.h>
#include <filesystem>
#include <stdexcept>

#include "no_wpi.h"
#include "bell_wpi.h"
#include "li_wpi.h"
#include "common.h"
#include "struct_Particles.h"
```

```
#include "struct_Ray.h"
#include "struct_Telescope.h"
#include "parameters.h"
#include "constants.h"

//HDF5 I/O
#include <highfive/H5File.hpp>
#include <highfive/H5DataSet.hpp>
#include <highfive/H5DataSpace.hpp>
```

Next, the definitions of the structures and the selection process for the command line arguments:

```
int main(int argc, char **argv)
{

    Particles particle; // Single particle struct
    // Vector of structs for the Particle Distribution
    std::vector<Particles> dstr(Distribution::population, particle);
    // Setup arguments
    SetupArgs setupArgs;
    // Detector
    Telescope ODPT(Satellite::latitude_deg, Distribution::L_shell);
    // Ray
    Ray ray;

    //----------- MANAGE COMMAND LINE  ARGUMENTS --------------//
    InputArguments(argc, argv, setupArgs); // If invalid input, terminates program
    //---------------- READ FILES FROM USER  ------------------//
    std::vector<std::string> dstrFiles, rayFiles;
    std::string selectedFilenameDstr, selectedFilenameRay, rayFilepathStr;
    std::filesystem::path dstrFilepath, rayFilepath;
    std::filesystem::path directory = std::filesystem::path("output") / "files";
    const std::string extension = ".h5";
    const std::string dstrPrefix = "dstr";
    const std::string rayPrefix = "interpolated_ray";
    // Read Particle Distribution
    std::cout << "\nPick a Particle Distribution file:\n";
    dstrFilepath = SelectFile(directory, extension, dstrPrefix);
    readDstr(dstrFilepath, dstr);
    if (dstr.size()!=Distribution::population) {
        throw std::runtime_error("The parameter for the particle population
        differs from the population that is defined in the selected hdf5 file! ");
        }
    else {
        std::cout<<"\nParticle population: "
        << Distribution::population <<std::endl;
        }

        // If WPI -> Select Ray
        if(setupArgs.wpi) {
```

```
        std::cout << "\nPick a Ray file:\n";
        rayFilepath = SelectFile(directory, extension, rayPrefix);
        rayFilepathStr = rayFilepath.string();
        ray.readRay(rayFilepathStr);
    }
```

Essentially, beyond the program's own name, which is stored in `argv[0]`, the second parameter `argv[1]` signifies the simulation time without the presence of a wave, while the third parameter `argv[2]` signifies the simulation time with the wave included. Additionally, there is an optional parameter `argv[3]` that allows a selection between using the first set of equations as (3.2f), (3.2g), (3.2h), (3.2i) as described in Chapter 3 in conjunction with a wave that can interact continuously with the particles. The wave can be modified by adjusting the parameters within the "Wave" namespace of the "constants.h" file. If this argument is not provided, which signifies the default operation, then the revised set of equations (3.2j), (3.2k), (3.2l), (3.2m) is utilized. In this case the program will also require the interpolated ray file as discussed in Section 4.3. During the default operation, the program awaits user input in order to choose the file containing the initial states of the particles and the electromagnetic wave. This allows the user to view the available particle distribution and wave files and to select the appropriate input for the simulation. The available files should have been created in advance by interpolating the wave and creating the particle distribution and will then be displayed in the console after executing the main function. Failure to perform these procedures in advance will result in the program being unable to locate the necessary input files, leading to its termination. To better understand the whole procedure of the simulation, an example of running the program can be found in Chapter 8.

The fundamental simulation functions are now described i.e., the functions which are executed once for every particle and incorporate its whole simulation. Further commentary on parallelism is provided in Chapter 5. These functions are designed to address specific simulation scenarios:

- `no_wpi()`: Simulates adiabatic particle motion.
- `li_wpi()`: Simulates particle interaction with a wave derived from ray tracing using equations (3.2j), (3.2k), (3.2l), (3.2m) and using ray data that are derived after ray tracing and interpolation. This function performs the default simulation of the wave-particle interaction.
- `bell_wpi()`: Simulates particle interaction with a wave that exists universally (in all space and at all times), employing equations (3.2f), (3.2g), (3.2h), (3.2i). This function is employed if the appropriate command line argument is set.

Within these functions, the RK4 mathematical method is employed to determine the states of the particles. RK4 calculates equations governing particle motion in Earth's magnetic field and their potential interaction with electromagnetic waves, as discussed in Sections 3.2 and 3.3. For compound simulations, where particles interact with waves after a predefined period, their states are stored and retrieved before running the second simulation. Additionally, it's necessary to condition escaped particles to prevent their re-entry into the second simulation. First the block of code that performs the adiabatic particle simulation (NOWPI):

```
//-------------- SIMULATION -----------------//
    std::cout << "\nSimulation starts...\n";
    int realthreads;
    real wtime1 = omp_get_wtime();
    //--- NOWPI ---//
    if(setupArgs.nowpi) {
```

```
    omp_set_num_threads(Simulation::num_threads_nowpi);
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        if(id==0) { realthreads = omp_get_num_threads();}
        #pragma omp for schedule(dynamic)
        for (auto &particle : dstr) {
            int id = std::distance(dstr.begin(),
            std::find(dstr.begin(), dstr.end(), particle));
            no_wpi(setupArgs.Nsteps_nowpi, id, particle, ODPT);
        }
    }
    real time1 = omp_get_wtime()-wtime1;
    std::cout<<"\nExecution time using "<<
     realthreads<<" thread(s), is: "<<time1<<std::endl;
}
```

Following this, a WPI simulation is conducted for a duration determined by the third command line parameter *argv[2]*. it is apparent that command-line parameters also determine the specific code chosen to execute the WPI simulation, as previously explained. At this point, all oscillating particles have maintained their last state of the previous adiabatic motion simulation. This state will now serve as the initial state for the WPI simulation.

For simulation using the equations (3.2j), (3.2k), (3.2l), (3.2m):

```
//---WPI---//
real wtime2 = omp_get_wtime();
if(setupArgs.wpi) {
    //--- (3.2j), (3.2k), (3.2l), (3.2m) ---//
    if(setupArgs.use_li_equations) {
        omp_set_num_threads(Simulation::num_threads_wpi_li);
        #pragma omp parallel
        {
            int id = omp_get_thread_num();
            if(id==0) { realthreads = omp_get_num_threads();}
            #pragma omp for schedule(dynamic)
            for (auto &particle : dstr) {
                if(particle.escaped || particle.negative || particle.nan)
                continue;
                int id = std::distance(dstr.begin(),
                std::find(dstr.begin(), dstr.end(), particle));
                li_wpi(setupArgs.Nsteps_wpi, id, particle, ODPT, ray);
            }
        }
        real time2 = omp_get_wtime()-wtime2 ;
        std::cout<<"\nExecution time using "<<realthreads
        <<" thread(s), is: "<<time2<<std::endl;
 }
```

For simulation using the equations (3.2f), (3.2g), (3.2h), (3.2i):

```
//---(3.2f), (3.2g), (3.2h), (3.2i)---//
else if(setupArgs.use_bell_equations) {

    omp_set_num_threads(Simulation::num_threads_wpi_bell);
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        if(id==0) { realthreads = omp_get_num_threads;}
        #pragma omp for schedule(dynamic)
        for (auto &particle : dstr) {
            if(particle.escaped || particle.negative || particle.nan) continue;
            int id = std::distance(dstr.begin(),
             std::find(dstr.begin(), dstr.end(), particle));
            bell_wpi(setupArgs.Nsteps_wpi, id, particle, ODPT);
        }
    }
    real time2 = omp_get_wtime()-wtime2 ;
    std::cout<<"\nExecution time using "<<realthreads
    <<" thread(s), is: "<<time2<<std::endl;
}
```

4.6 Simulation comparison

In order to investigate the impact of an electromagnetic wave on energetic particle distributions, two separate simulations are executed, and their results are compared:

1. **Adiabatic Motion Simulation:** This simulation focuses on the natural motion of trapped particles, free from electromagnetic wave interference. It runs for a duration of *t* seconds.
2. **Wave Interaction Simulation:** This simulation employs identical parameters and particle distributions as the first one. However, it introduces an electromagnetic wave after a specified time frame. It runs for a duration of *t' = t_nowpi + t_wpi* seconds.

In order to compare the two simulations, they both need to be executed for the same durations i.e., *t = t'*. Introducing the wave later in the simulation, rather than at its onset, serves as a randomization for the cumulative state of particle population. This ensures that as particles initiate their oscillations in the simulation, they do so without the wave interacting with them. As a result, the particle flow at any given latitude maintains a relatively consistent value.

After both simulations are completed, their outcomes can be compared to derive meaningful conclusions. Figure 26 illustrates the comparative process, aiding in a better grasp of the concept.

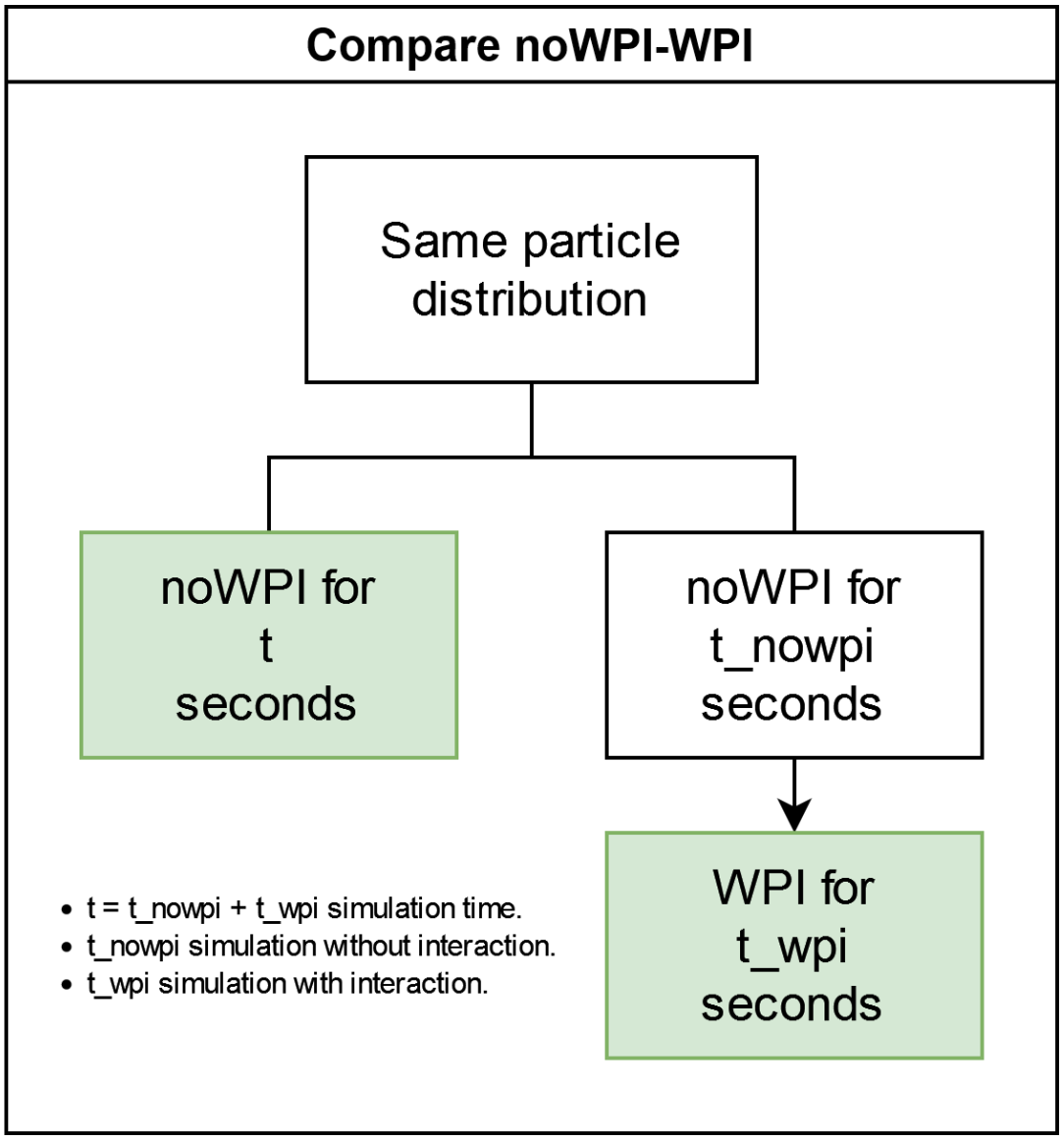


Figure 26: Both simulations are executed using the same particle distribution, with one of the simulations introducing a wave after a certain period of time.

Once the code has been methodically organized into distinct routines and structures and after implementing parallel design, the flowchart of Figure 27 was created. This flowchart outlines the procedure for comparing two simulations and extracting results. For detailed insight into how the code operates, an enriched version of this diagram is uploaded in drawio which includes hyperlinks that can facilitate source code navigating.

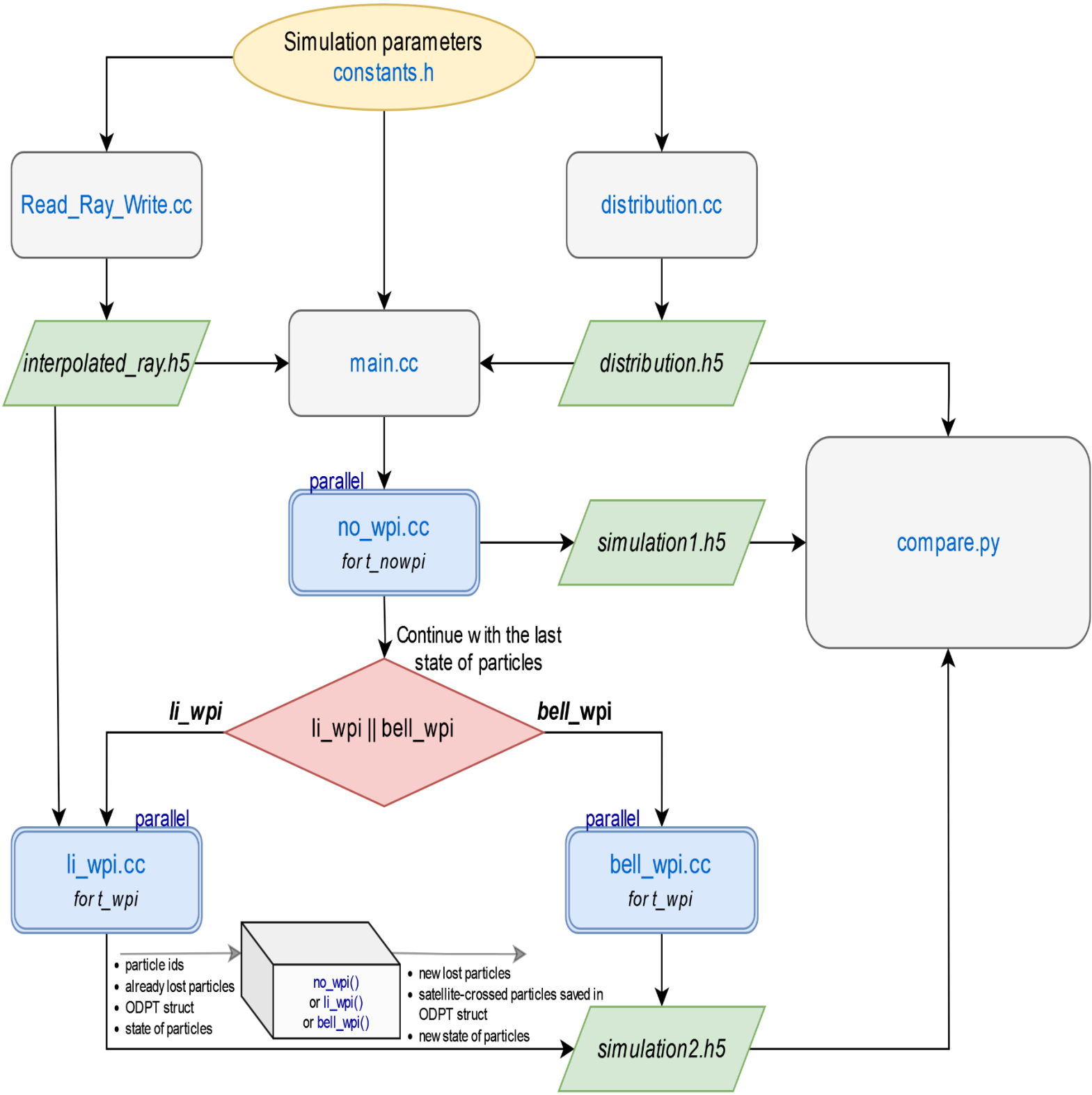


Figure 27: Flowchart illustrating the complete algorithmic process, encompassing particle distribution and electromagnetic wave generation, simulation of wave-particle interaction, and finally visualization of the result with 'compare.py' script.

4.7 Post Processing

After completing the simulation, it is essential to arrange the acquired data for subsequent analysis and visualization. During the simulation process, the particles undergo state changes, including alterations in velocity, momentum, and geographic position at each time step, as per the RK4 numerical method. Simultaneously, the satellite records these particle states when they traverse its latitude. Additionally, certain criteria can identify particles as loss cone particles, prompting their data to be captured and stored in memory. This valuable data is organized and stored in an HDF5 file which is a versatile data storage format for handling large, structured datasets in various scientific and computational contexts. This is facilitating easy access for future manipulation and visualization with Python scripts.

Up to this point, the detector has employed a simplified approach by storing all particles in a single list, foregoing the categorization based on their states, such as energy and pitch angle. In practical detector systems, however, particles are captured and meticulously grouped into various categories that encompass temporal, spatial, and energy considerations, as elaborated upon in Section 2.5. This grouping procedure is deferred to post processing, offering greater flexibility, as it allows for the application of different configurations to the same dataset, yielding diverse and valuable insights. Specifically, one of the procedures which must be employed in the post processing is time binning. Time binning is a method that involves partitioning particle data into discrete time intervals or *time bins*. Each time bin corresponds to a specific time range, and particles that traverse the detector during that particular timeframe are grouped together. This approach is particularly valuable for investigating the temporal behavior of particles and analyzing particle fluxes at different time intervals. As emphasized in Section 2.4, the study of PAD holds significant importance when examining particles within radiation zones. By also categorizing particles into *pitch angle bins*, their population can be visualized as a flux number of every look direction of the detector. This visualization aids in identifying trends and patterns in the data, potentially unveiling resonant interactions, estimating particle drift paths, and characterizing different particle populations. Essentially, a pitch angle bin defines an angle range for a particle, categorizing it based on its pitch angle within that range.

Both of these binning procedures necessitate the definition of a sector or bin size, which serves as adjustable parameters within the program for visualizing PADs. These parameters are flexible and can be modified to either zoom in or out on the particle population, enabling the capture of a broader or more focused view of the data. The ability to adjust the bin size facilitates the exploration of various perspectives within the dataset, ultimately contributing to a more comprehensive description of particle dynamics and wave-particle interactions. Furthermore, these adjustments in bin size have practical applications in the design of detectors, allowing for customization to target specific characteristics of the particle distribution. By tailoring the bin size to the detector's objectives, researchers can optimize their instruments to capture the most relevant information and gain deeper insights into the underlying physical processes. Throughout the simulations, an ODPD was employed to capture the complete spectrum of PADs. Figure 28 shows an example of how the particles are categorized for a sector size of 15 degrees. The magnetic field vector of the Earth served as the reference point for determining the degree values of the angles. Suppose a particle has a pitch angle of 52.5 degrees, and the detector covers a full view with pitch angle bins of 15 degrees each. In this case, the particle would fall into the fourth sector (225 to 240 degrees). As a result of this process, lists are generated for each time range and pitch angle range encountered during the simulation.

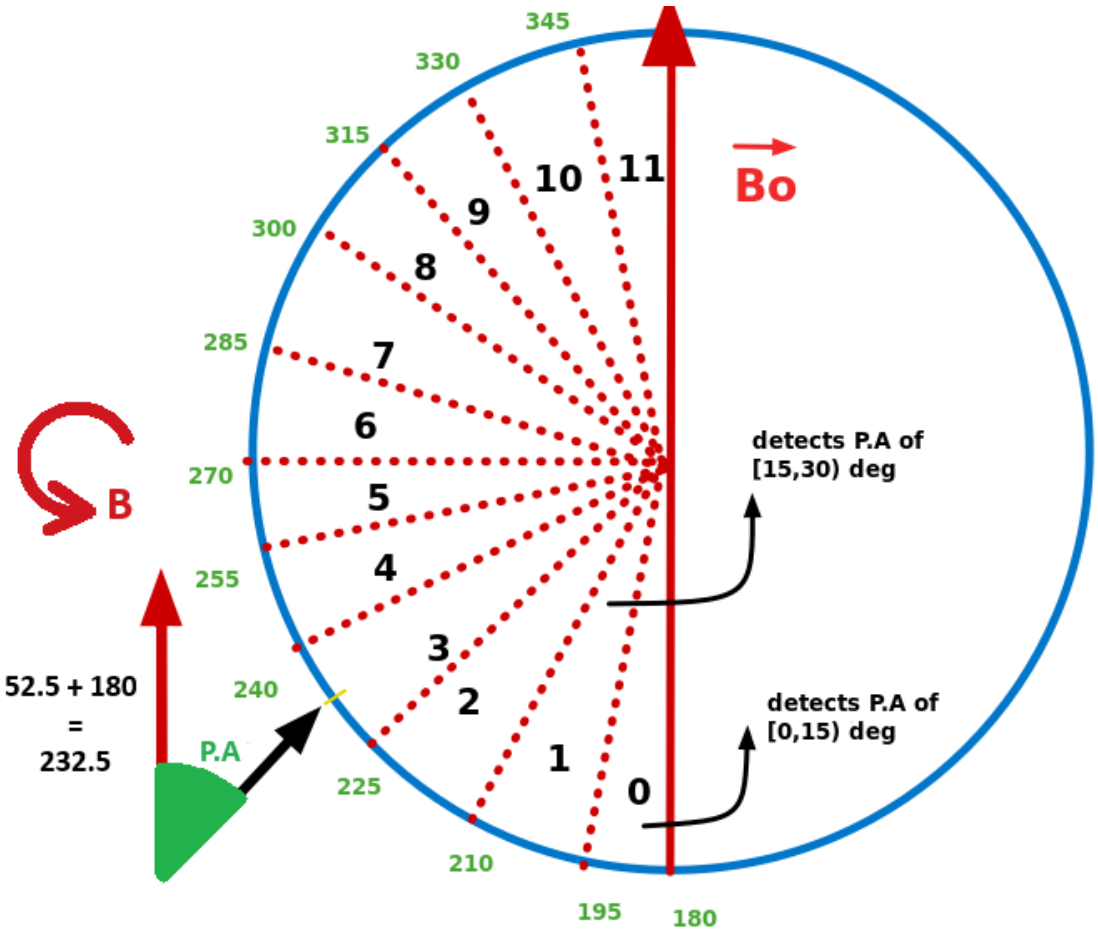


Figure 28: Pitch angle sectors of the detector.

The initial step in managing the binning process involves reading the HDF5 files from one or more simulations. These files contain essential information, including the particle states throughout the simulations, data on lost particles, and more. This binning operation is carried out within a Python program named "*compare.py*" by multiple threads to improve performance. Every thread performs a binning process over a different parameter. Later in this script, as its name suggests, two simulations are compared by analyzing particle fluxes within specific timeframes. This comparison is facilitated by generating plot and movie files, providing a visual representation of the differences between the two simulations. The primary parameters of this program revolve around defining the bin sizes:

```
##TELESCOPE SPECIFICATION -- BINNING PARAMETERS
time_bin = 2
timesteps = math.ceil(t / time_bin)
view = 180
sector_range = 2 #PA bins, look directions
sectors = int(view/sector_range)
```

After categorizing all the particles, the next step is to visualize the data using diagrams. Depending on the type of data, sometimes a single static graph is all that's needed. However, in other instances, it's more effective to use multiple graphs presented in a video format, where each

graph is captured within a single frame. The following discussion will outline the process of generating these diagrams using Python.

The initial output of the *"compare.py"* program is the "*Crossing_particles.png*" graph. An example of this visual output is presented in Figure 29. This plot depicts the presence of all detected particles over time, illustrating how their distribution varies with latitude. This operation is executed once for each simulation. Upon generating these *.png* files, the distinctions in crossing points among particles across various simulations become apparent. Particles moving northward are captured below the satellite, while those traveling southward are captured above the satellite. In simpler terms, particles are recorded just before they cross the satellite's path. The satellite's location is indicated by a dashed blue line, and in this example, it is positioned at the equator, which corresponds to a latitude of zero degrees.

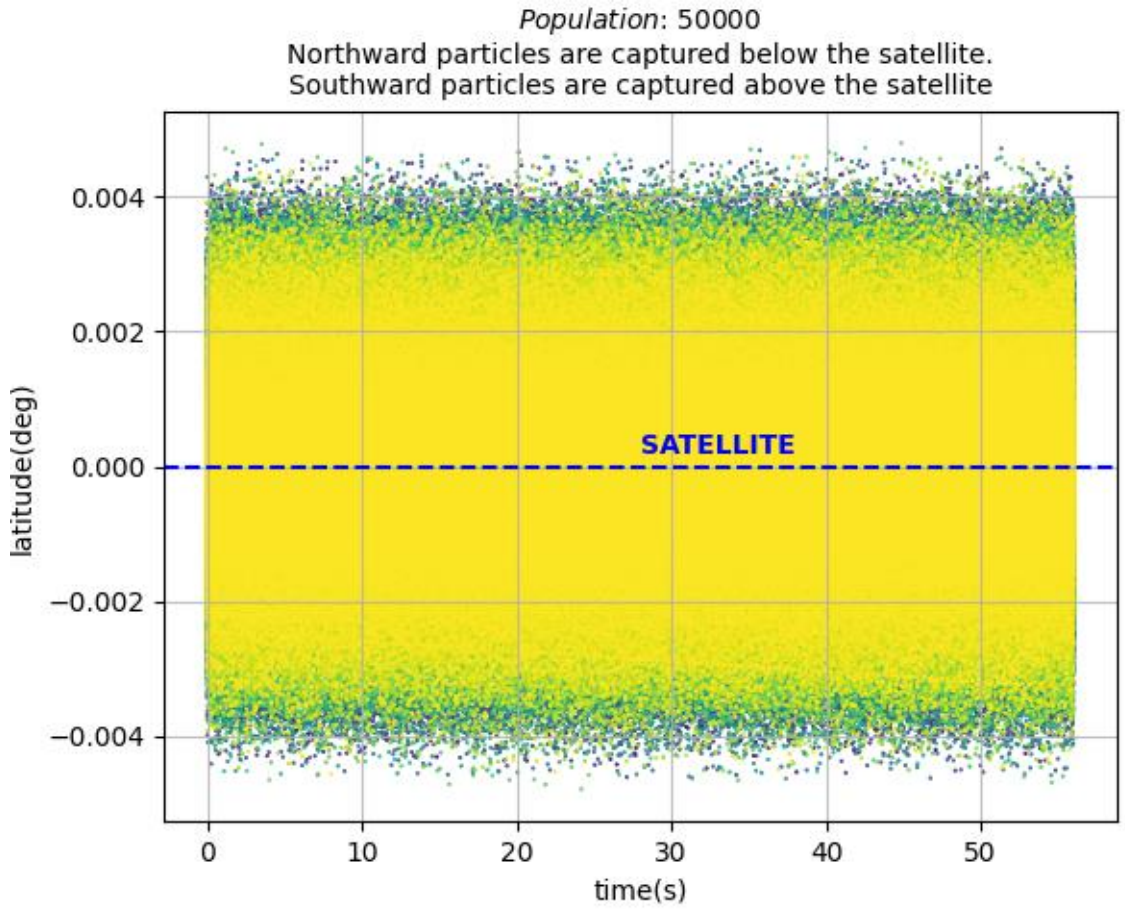


Figure 29: Particles crossing the satellite. The satellite is at the equator.

Additionally, a graph illustrating the total number of particles detected is generated under the name "*Particle_sum.png*." This graph captures the cumulative particle flux in the detector from all directions over time. In this manner, all incoming particles are recorded, allowing for a direct comparison of total flux between two simulations. Specifically, when analyzing the results of two simulations, introducing a wave in one of them at a certain point in time, it is easy to ascertain whether the particle flux is diminished or remains unchanged. This observation enables the detection of particle loss during the process, with the electromagnetic wave introduction being the likely cause. In this particular example, the particle simulation models the adiabatic motion for a duration of 50 seconds before the introduction of the wave which triggers a decrease in the particle count.

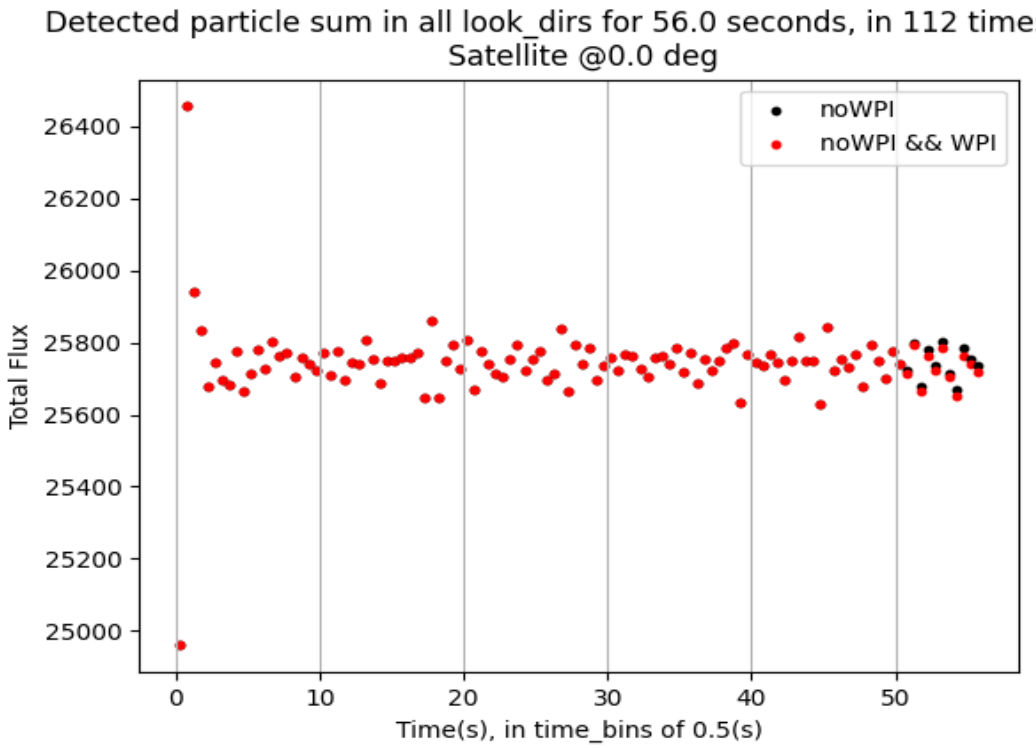


Figure 30: The total flux of particles is decreased when the wave is introduced to the simulation.

An additional visual representation of the phenomenon is created in the form of a video file named *"Bins.mp4."* In this video, each frame captures a snapshot of a time interval determined by the *time_bin* parameter. Within each frame, the detected particles are categorized into bins based on their pitch angle, as defined by the *sector_range* parameter. Within this video file, there are bar charts starting from the bottom of the graph. These charts illustrate the portion of the particle population that either transitioned into different pitch angle bins or precipitated into the upper atmosphere. The vertical axis of these charts represents the particle count, displayed in a logarithmic scale, while on the horizontal axis, the particle bins are categorized into sectors, which is the width of the pitch angle bins. An example frame of a movie follows:

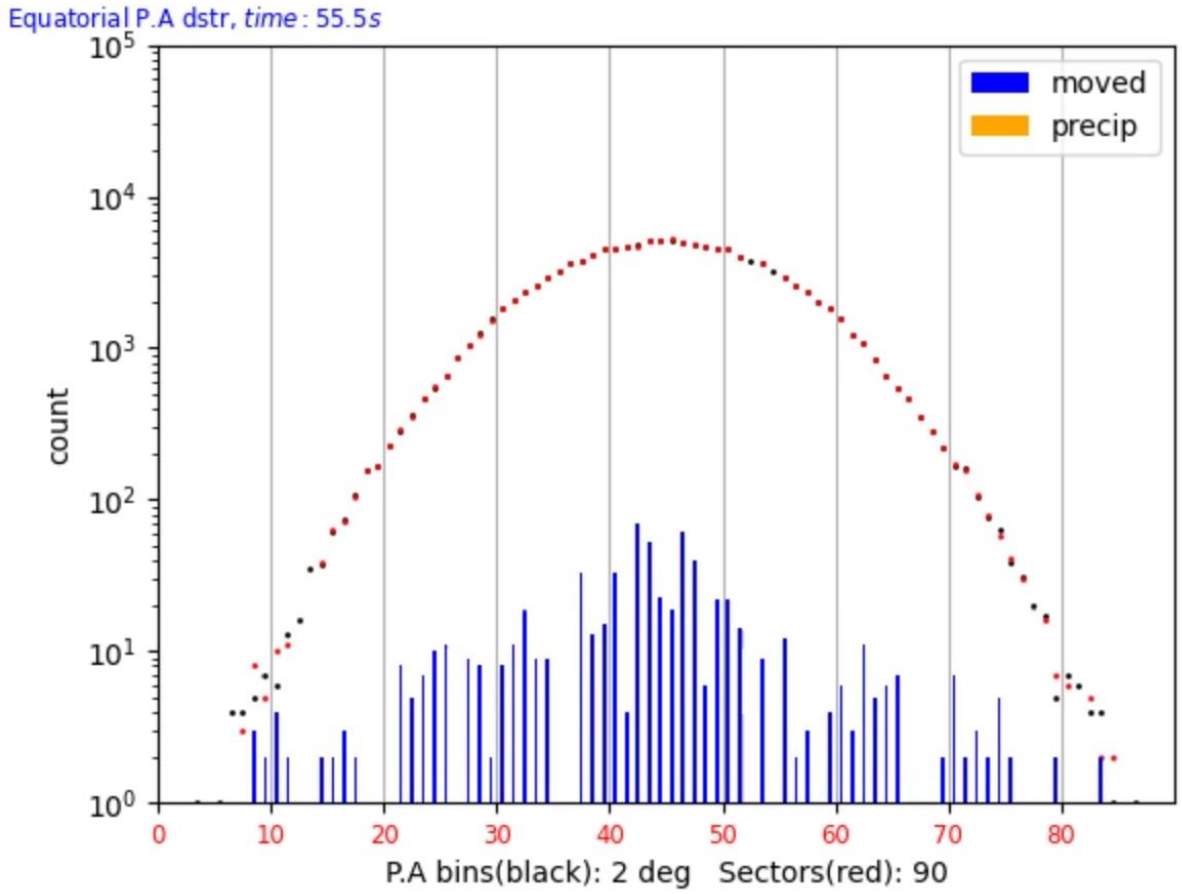


Figure 31: Frame extracted from a video file, illustrating the Pitch Angle Distribution (PAD) between 55.5 and 56.0 seconds into the simulation. The particle count is in logarithmic scale. The bar charts depict particles that have migrated (in blue) from other bins as well as the precipitating particles (in yellow) within this specific time window.

Chapter 5

Parallel algorithm

5.1 Parallel programming

Before exploring the concept of parallel programming, it is valuable to discuss the conventional approach of designing and executing an algorithm, known as serial programming. The steps for designing and executing a serial algorithm are as follows:

1. The problem is broken down into discrete pieces of instructions.
2. These instructions are executed sequentially, one after the other.
3. The process is typically carried out by a single processor.

As each instruction is executed in sequence by a single processor, it becomes evident that only one instruction can be processed at a time.

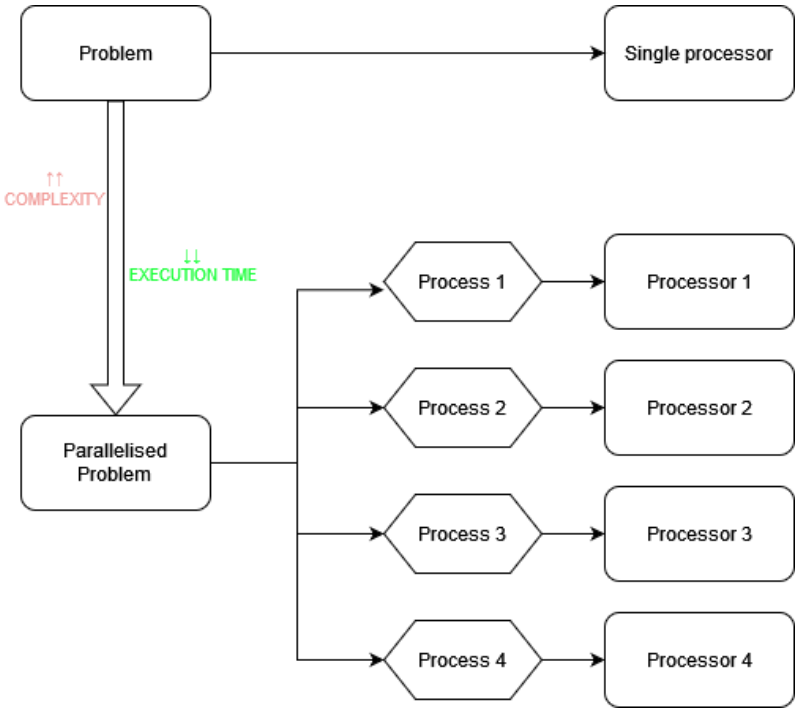


Figure 32: Comparing the execution of a problem on a single processor versus running a parallelized version of the same problem using multiple processors.

The procedure for designing a parallel algorithm follows these steps:

- The problem is divided into discrete chunks of instructions that can be executed simultaneously.
- Each chunk can be further divided into an expression of serial instructions.
- These instructions are executed concurrently or in parallel by different processors.

A crucial aspect of designing a parallel algorithm is the capability to divide it into discrete parts that can function as autonomous code segments. However, it is important to note that not all programs are suitable for this type of design, and in some cases, it may significantly increase the code's complexity. It is important to consider whether the potential speedup can justify the time spent from programmer to design the parallel algorithm and whether it is practical to allocate the computational resources for its execution. If a problem is suitable for parallel processing, the selection of the appropriate method and tools becomes crucial, while it is essential to assess the feasibility of its execution on various computer systems.

In general, various parallel systems are capable of processing parallel algorithms. The most common ones are described below.

Distributed memory

These systems, often referred to as multi-computers, are complete units (computers) with their own local memory. To enable access to another processor's memory, a remote processor communicates through a network of messaging between computers. Ethernet is commonly used for network technology, but other fiber optic networks can also be employed for faster communication. This implementation ensures data security, as modifications to a processor's local memory do not affect other processors, allowing safe modifications to their respective local memory. Additionally, access to local memory is very fast, and these systems exhibit excellent scalability for a large number of computers. However, a significant disadvantage lies in the complexity of data switching and execution synchronization, and communication between computers incurs a cost due to network usage.

Distributed computing is widely used in various scientific disciplines, including physics, to study and simulate complex physical phenomena. It involves breaking down computational tasks into smaller pieces and distributing them across multiple computing nodes or processors, which work in parallel to solve the problem. This approach allows researchers to handle large-scale simulations and data-intensive computations efficiently, enabling them to explore intricate physical phenomena that would be otherwise computationally infeasible using traditional single-machine approaches.

Shared memory

In shared memory parallelism, all participating processing units have access to a universal memory address space. One processor can modify memory regions, affecting other processors. Processors are assigned specific processes but share the same memory space. Sharing the address space can simplify the design, and synchronization between processors is relatively fast if proper techniques are used. However, the scalability of the number of processors is often limited, and the memory bus can become congested when more processors are added. To effectively utilize shared memory parallelism, programmers need to be familiar with specific programming techniques to avoid areas with concentrated high workloads, which could render parallelism ineffective.

GPU Accelerators

In the past, GPUs were primarily designed for computer graphics. However, in modern times, they are being extensively utilized for general-purpose computing, commonly referred to as GPGPU computing. A commonly used API for implementing algorithms on GPU accelerators is CUDA (Compute Unified Device Architecture), developed by Nvidia, a company that produces such processors. CUDA allows the exploitation of GPU resources for general-purpose processing and is designed to work with programming languages like C, C++, and Fortran.

Each parallel processing method has its pros and cons, and the choice depends on the specific application. There is no universally better method; rather, one approach may be more suitable than another depending on the requirements of the task. Hybrid parallel computing, which combines shared and distributed memory, is often used in high-performance computing as it succeeds in meeting the diverse processing needs and desired results. This thesis explores and implements both distributed and shared memory parallelism. Specifically, the MPI and OpenMP APIs are utilized for these two parallel computing approaches.

5.2 Distributed memory (with MPI)

In the case of this simulation, MPI (Message Passing Interface) standard is used for the distributed design and execution of the algorithm. It provides a set of library routines and a communication protocol that allows multiple processes to exchange messages and data with each other in a parallel computing environment. The particle population is divided into chunks, sharing the workload from one processor to all the others to speed up the execution of the simulation. With this type of parallelism, each process has a portion of the data and simulates with limited communication with other nodes. As mentioned earlier, when utilizing MPI, the communication between the nodes incurs substantial network costs. Finally, when all the nodes finish simulating their workload, one processor gathers and stores all the resulting particle data. This one processor that is responsible for managing and coordinating the overall execution of the program is usually referred as the *master*. The master node is in charge of distributing tasks to other worker nodes, collecting their results, and orchestrating the overall workflow of the distributed application. The detailed MPI code is not provided in this document, unlike the OpenMP code, which is thoroughly explained in the following section. However, the MPI implementation’s code can be found in the corresponding Git repository branch, and its performance is later compared to that of the shared memory computing solution in Section 5.4.

5.3 Shared memory (with OpenMP)

OpenMP is an API designed to facilitate parallel programming in multi-threaded environments with shared memory. It simplifies parallelization by allowing programmers to use compiler directives to specify which parts of the code should be parallelized, and it automatically manages the distribution of tasks among the available threads. Each thread handles its own portion of the workload in parallel, while also having access to shared memory that all threads can utilize. However, care

must be taken to avoid race conditions and to properly manage the critical regions i.e., regions where only one processor executes workload, as they can potentially limit the program's speed.

In this simulation, the OpenMP work sharing feature is used to divide and execute the main task of simulating individual particles. Each thread performs the simulation on a dynamically allocated chunk of particles, which ensures a fair and efficient distribution of the workload. As some particles may escape their trapped motion and exit the simulation, the dynamic allocation of workload ensures that when a thread completes the brief simulation of the escaping particles, it can promptly begin simulating another one.

As particles are simultaneously simulated across various threads, they all have access to a shared memory. This shared memory contains the detector structure, responsible for storing particles that pass through the satellite. While shared access is convenient, it necessitates careful management to ensure that only one thread can trigger the saving of a particle that has crossed the satellite, thus avoiding race conditions. Properly defining critical regions is essential to prevent the simultaneous storage of multiple particles that may cross the detector at the same time.

In the simulation, each particle is treated as an independent process, responsible for its own simulation. This parallelism is achieved using *work sharing* or *load sharing* with the *for-loop* directive. Each processor receives one particle at a time and performs a single particle simulation. As mentioned earlier, the work sharing is performed dynamically because it is not possible to predict in advance when a particle will exit the simulation. This uncertainty requires the load to be distributed among processors on the fly, ensuring efficient handling of particles as they progress through the simulation. The code snippet below demonstrates how the load is shared among processors using the OpenMP for directive:

#pragma omp for schedule(dynamic)

This directive specifies how processors will receive new processes, i.e., particles to simulate. Each processor receives a set of particles to work on, and this will be its workload chunk. Once the processor is done with this chunk of particles, it will receive another chunk of particles to process. The simulation of one particle is independent of the simulations of others. However, it is essential for these simulations to share memory accessible by all other processes. In this case, it is crucial to share the structure of the detector, as it is used to store the states of particles when they pass through the satellite. The parallel code block follows:

```
omp_set_num_threads(8); //Performance peaks with 8 threads.
//---PARALLELISM Work sharing---//
#pragma omp parallel
{
int id = omp_get_thread_num();
    if(id==0) { realthreads = omp_get_num_threads(); std::cout<<"\nRunning threads: "<<realthreads<<std::endl; }
     #pragma omp for schedule(dynamic)
    for(int p=0; p<Population; p++) //dynamic  some chunks may have less workload.(particles can precipitate and
break out of loop
    {
    //Void Function for particle's motion. Involves RK4 for Nsteps. Detected particles are saved in ODPT object, which
is passed here by reference.
    no_wpi(Nsteps_nowpi, p, dstr[p], ODPT);
    }
}
std::cout<<"\n\n"<<"Joined"<<std::endl;
real time1 = omp_get_wtime()-wtime;
std::cout<<"\nExecution time using "<<realthreads<<" thread(s), is: "<<time1<<std::endl;
```

The OpenMP API provides a mechanism to define a point in the code that can only be executed by one processor at a time, which is also known as a critical point. This means that when a processor reaches this point, it can execute it only if no other processor is currently executing it. Consequently, this critical point can potentially slow down the execution of the algorithm, acting as a bottleneck. Therefore, using critical points should be approached with caution and care. In this simulation, the critical point is strategically placed only once in the code, specifically within the nested iteration of the RK4 method. The functions executed within RK4 take considerable time compared to the time a processor spends within the critical region. As a result, the probability of congestion at this critical point is low and is directly related to the number of processors executing the program.

```
#pragma omp critical //Only one processor should write at a time. Otherwise there is a chance of 2 processors writing in the same spot.
{ //This slows down the parallel process, introduces bad scalling 8+ cores. Detecting first and storing in the end demands more memory per process.
//Check Crossing:
if( ODPT.crossing(new_latitude*Constants::R2D, latitude*Constants::R2D, Constants::L_shell) )
{
    ODPT.store( p, latitude, aeq, alpha, time); //Store its state(it's before crossing the satellite!).
} }
```

5.4 Benchmarking Analysis

The performance of the OpenMP algorithm is demonstrated with Figure 33 which illustrates the program's performance when executing the algorithm utilizing varying numbers of cores. It is clear that the highest performance is achieved when the workload is distributed across 8 threads. In the context of running two distinct simulations, as depicted in Figure 26, employing 8 threads per simulation was sufficient. This configuration effectively utilized all the threads available in a 16-thread system, as explained later in Section 7.1. The chart also demonstrates that this observation holds true for larger particle sets that undergo longer oscillations. The data was obtained using the *gprof* tool, which allowed to extract normalized execution times for each function. Subsequently, this data was imported into Python and visualized.

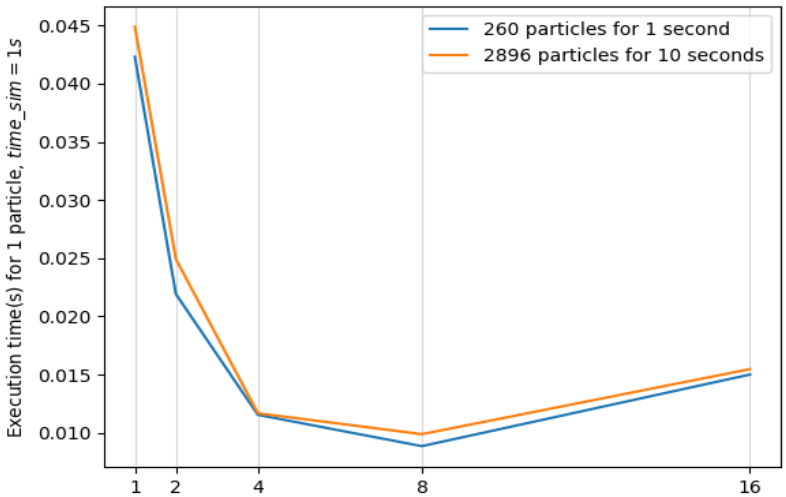


Figure 33: Performance scaling of the OpenMP algorithm is optimized when distributing the workload across 8 threads.

An additional aspect assessed was the simultaneous execution of functions that are executed within the Runge-Kutta 4 method. A function that reads input from another function clearly indicates a dependency. Functions that are not dependent on each other could be executed in parallel. This can be explained better with a flow diagram of the pipeline in Figure 34. After a thorough examination of the problem using profiling, specifically by analyzing the execution times of the mentioned equations with the *gprof* tool, it was concluded that only three processes could be effectively parallelized. However, further investigation revealed that the involved functions of these processes were already sufficiently fast. In other words, the marginal acceleration achieved wouldn't justify the increase in algorithmic complexity resulting from introducing another layer of parallel processing.

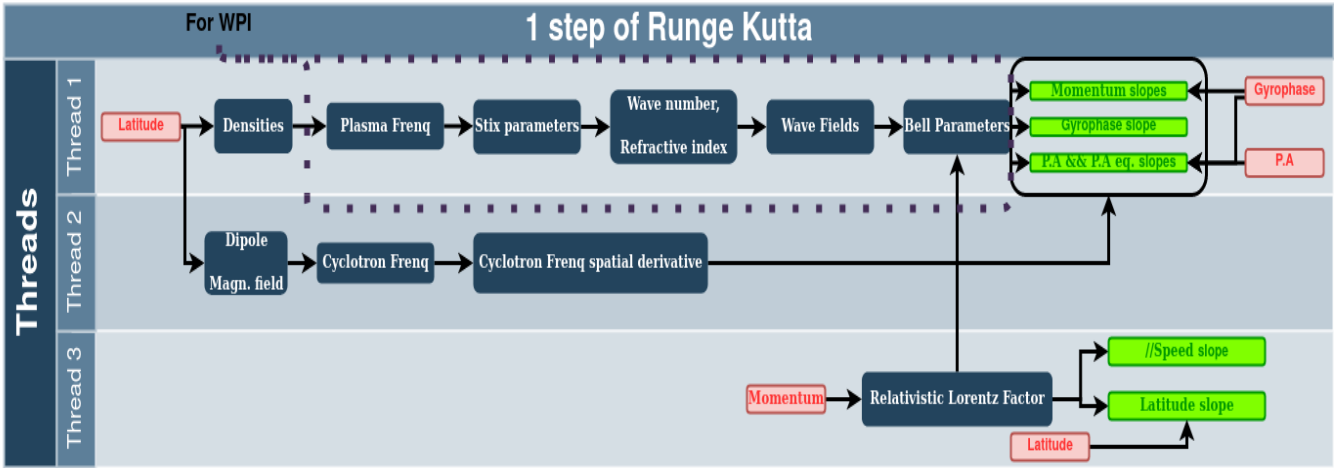


Figure 34: Organizing processes into a pipeline to demonstrate the potential for attainable concurrency.

By comparing the two parallel implementations with OpenMP and MPI, the following diagram is generated. This diagram serves as evidence that both techniques yield similar execution times. It presents a bar chart illustrating the execution times in seconds for a 2-second simulation. The first second of the simulation excludes the electromagnetic wave, while in the subsequent second, it is introduced, allowing for interaction with particles. The results are organized into three sets, each with varying particle populations (10, 100, and 1000 particles). In the chart, the pink bars represent the performance of the MPI algorithm, corresponding to the allocated slots, while the purple bars represent the performance of the OpenMP algorithm corresponding to the threads used during execution.

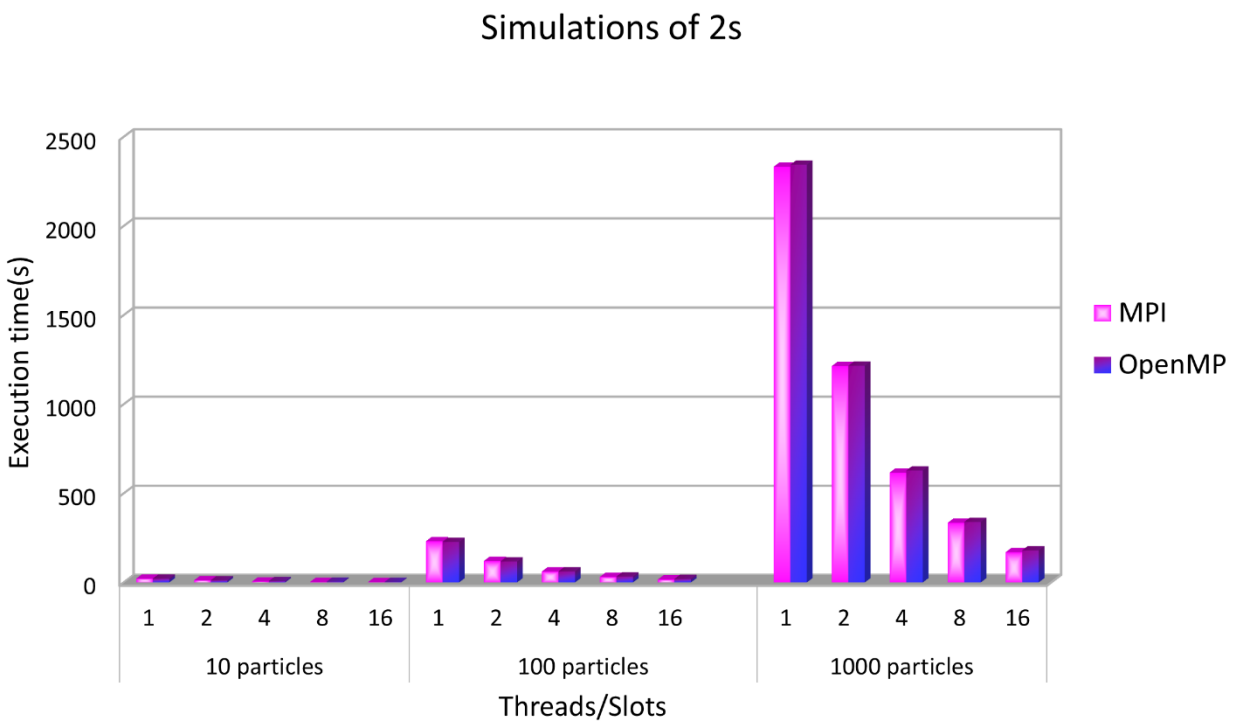


Figure 35: Comparing the performance of both algorithms executed in the same system and using the same resources. A two-second simulation (one second for adiabatic motion followed by one second of wave-particle interaction), it is evident that the results are nearly identical.

One of the key takeaways of Figure 35 is that both techniques deliver comparable performance. However, it's worth noting that the MPI implementation is considerably more complex than the OpenMP counterpart. Developing the OpenMP code was a straightforward process that demanded less focus on workload distribution and memory management. In the OpenMP implementation, there was a single critical region that required attention to prevent race conditions when saving two particles that might simultaneously cross the satellite. Fortunately, this scenario is uncommon and did not significantly slow down the code. Therefore, one could argue that the OpenMP implementation is superior, as it provides similar performance with reduced complexity.

The primary advantage of the MPI implementation lies in its potential for further improvement through hybrid computing. By integrating shared computing techniques on top of the distributed processing algorithm, multiple threads on multiple computing nodes can be leveraged, leading to even greater simulation acceleration. In contrast, the OpenMP solution is constrained by the maximum number of threads a single system can offer. These conclusions are summarized in Figure 36.

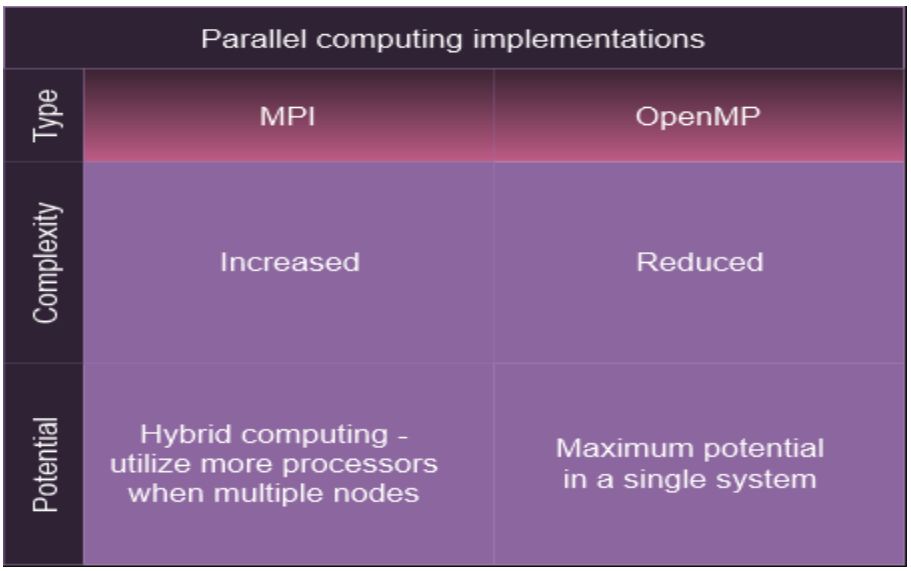


Figure 36: Assessing the complexity and the potential of each of the two parallel programming solutions for the execution of this simulation.

Chapter 6

Simulation results

This chapter provides an overview of the simulations conducted using the algorithm presented in this thesis, along with an analysis of their outcomes. The first section begins with a single particle simulation example, and the second section expands on particle distribution simulations. For visualization, diagrams are used to represent the cumulative flux of particles over the entire simulation duration, while movie files allow the observation of distribution evolution within specified timeframes. Although it's not feasible to directly embed movie files in this document, they are accessible in the GitHub repository, and generating these files is explained in Chapter 8. However, this chapter offers insight into the findings by presenting selected frames from these videos.

6.1 Single particle simulation

For a 1keV particle originating from the equator ($\lambda = 0^{\circ}$) and undergoing 60 seconds of oscillatory motion within Earth's magnetic field, the following diagram can be generated by retaining all the latitudes it traverses at each simulation step. This particle exhibits periodic motion within a magnetic reflector, with reflection points occurring in the North at 25 degrees and in the South at -25 degrees, respectively.

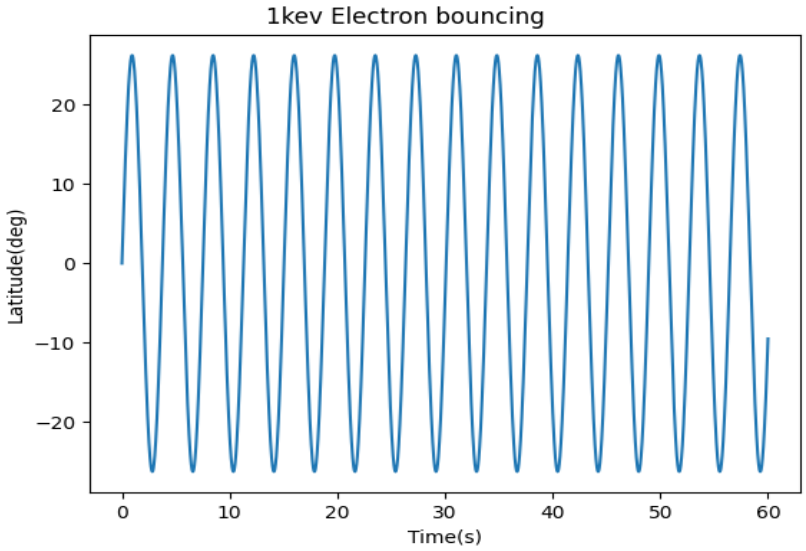


Figure 37: The electron with an energy of 1 keV undergoes oscillations without interacting with a wave for one minute, oscillating between latitudes of -25 and 25 degrees.

To verify this oscillation, the *Fast Fourier Transform (FFT*) can be applied to convert the signal from the time to the frequency domain:

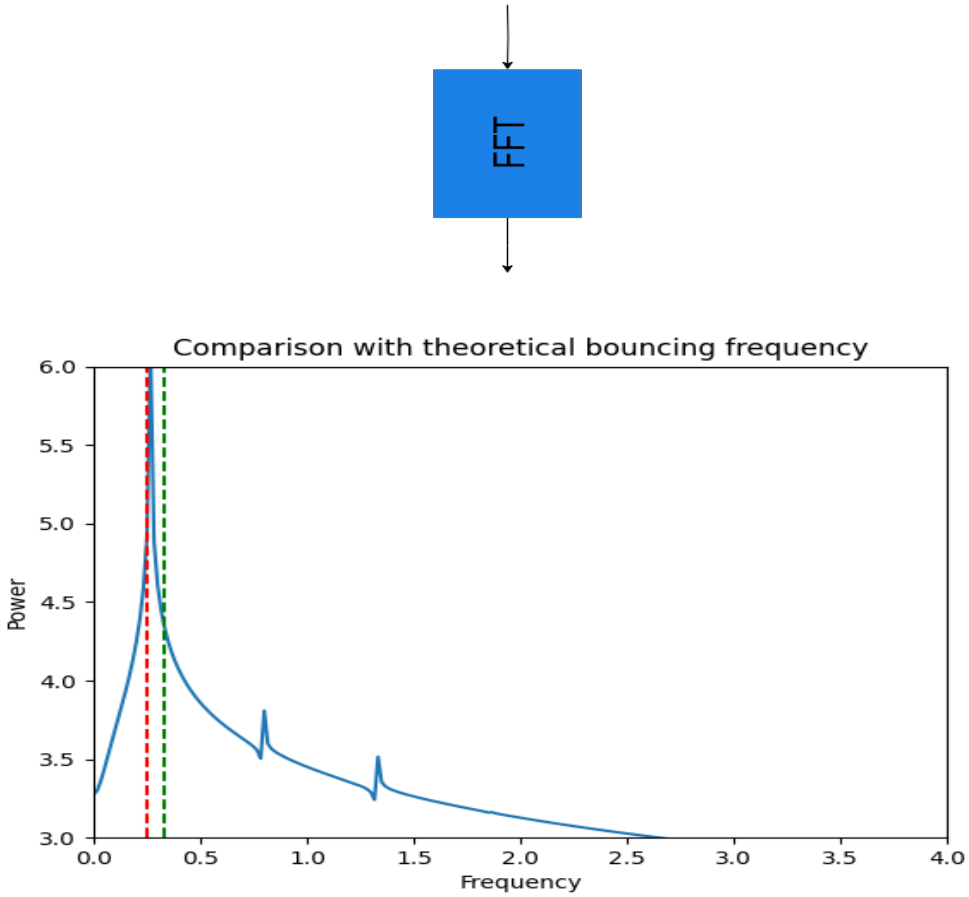


Figure 38: Performing a Fast Fourier Transform to convert data into the frequency domain. The dashed vertical lines represent approximations of the motion frequency within a magnetic reflector.

This allows the comparison of the resulting frequency with the theoretically calculated frequency. The frequency estimate, indicated by the red dotted vertical line, is calculated using the following equation [33]:

$$4L_{shell}\frac{R_E}{u}\left(1.3-0.5sin(aeq)\right) \text{ for particle velocity } u=\sqrt{\left(1-\left(\frac{1}{\gamma}\right)^2\right)}*c \text{ where:}$$

c is the speed of light, γ is the Lorentz factor, R_E is the Earth's radius, aeq is the equatorial pitch angle and L_{shell} is the set of planetary magnetic field lines. The green dotted line represents an estimate obtained from an online calculator [34].

Now starting off with a straightforward scenario: the interaction between a single particle and an electromagnetic wave, as depicted in Figure 39. In this example, a particle starts with an initial equatorial PA (aeq) of 70 degrees and engages with an electromagnetic wave. The points where substantial disruptions occur coincide with moments of resonance between the particle and the wave, a concept elaborated upon in Section 3.2. This resonance phenomenon may occur multiple times, with certain instances leading to more pronounced alterations in the particle's pitch angle. In this example, the particle originates at a latitude of 9-degrees below the equator ($\lambda = -9^{\circ}$) and possesses an energy level of 500 keV. The particle's velocity, perpendicular to Earth's magnetic field, forms a 30-degree angle ($eta = 30^{\circ}$) with the right-cyclically-polarized component of the wave. The wave itself has an intensity of 10 pT and operates at a frequency of 2000 Hz.

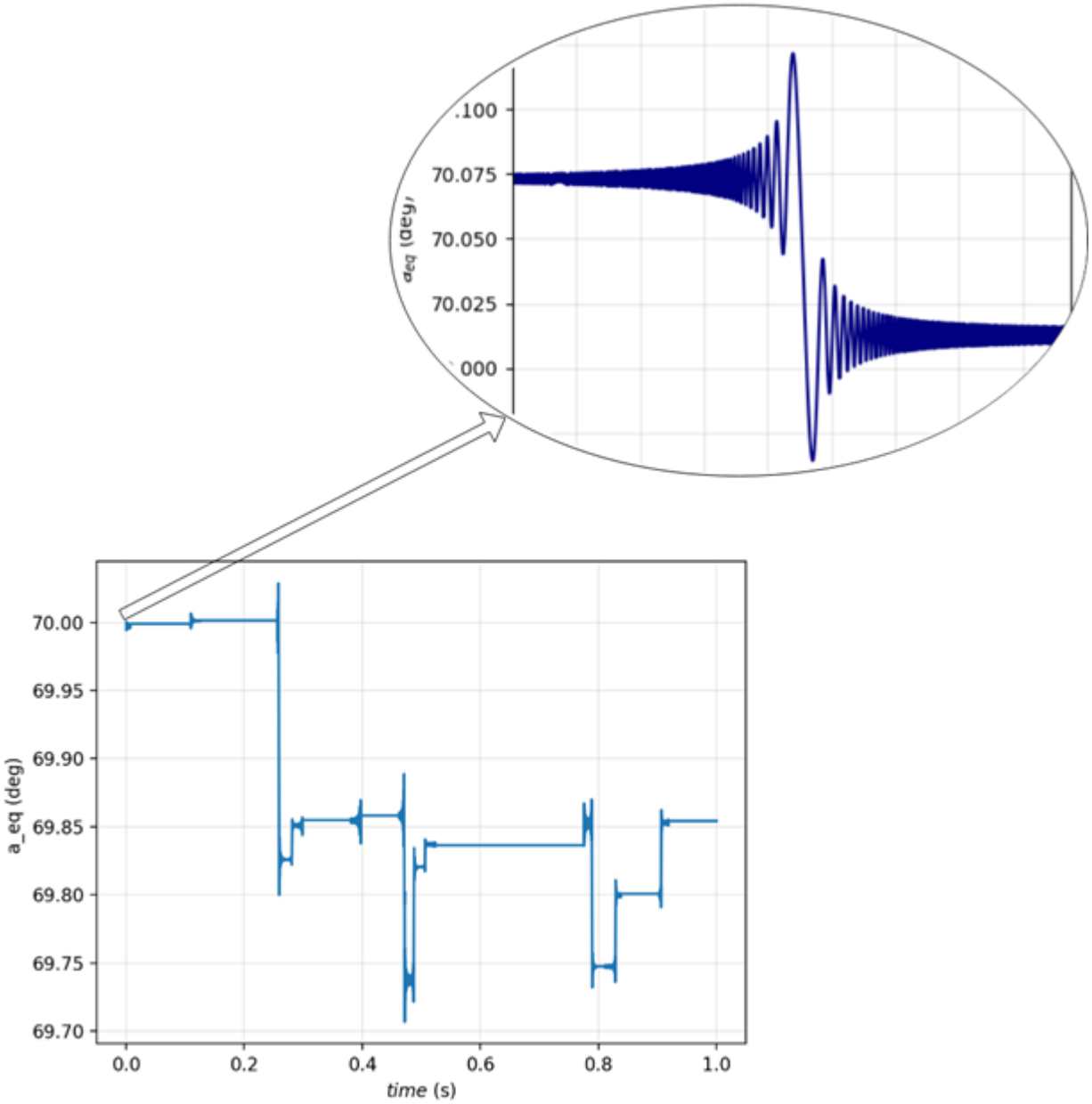


Figure 39: Wave-particle interaction of a 500 keV particle with a 1nT electromagnetic wave. Zooming into the interaction region allows for a clearer observation of the pronounced change in the equatorial PA value.

Now, let's compare two simulations. In the first simulation, a single particle undergoes adiabatic motion, while in the second simulation, it interacts with a wave after a certain period. The focus is on the following particle parameters:

- Equatorial PA (aeq)
- Time variation of eta parameter ($deta_{dt}$)
- Latitude ($lamda$)

The particle's initial state is defined by the following values:$aeq = 64^{\circ}$, $\lambda = 0^{\circ}$, and $eta = 30^{\circ}$. Additionally, the wave used in the simulation has an intensity of 1nT.

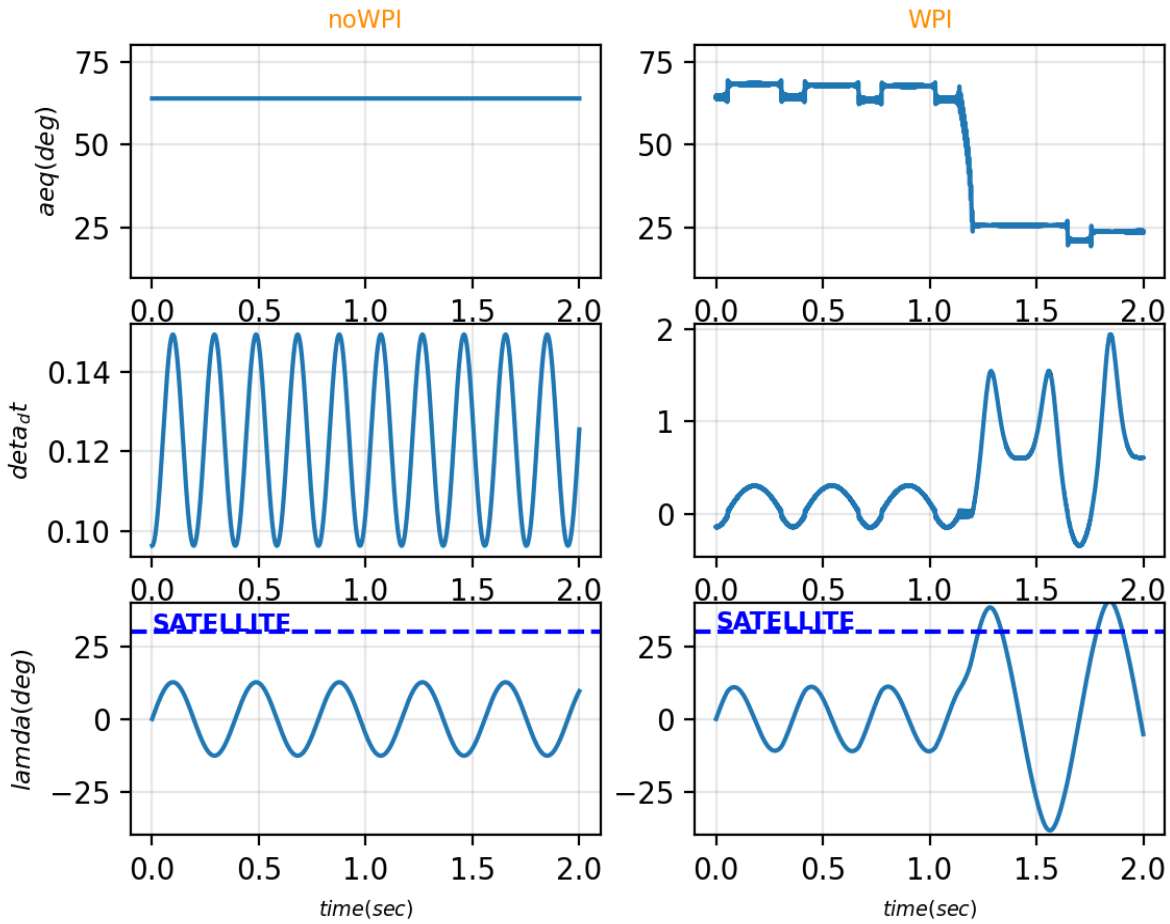


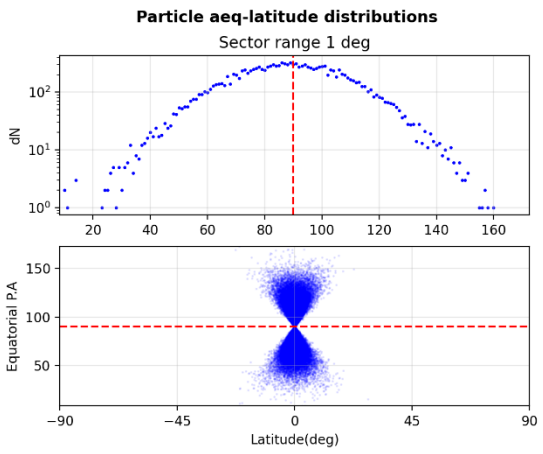
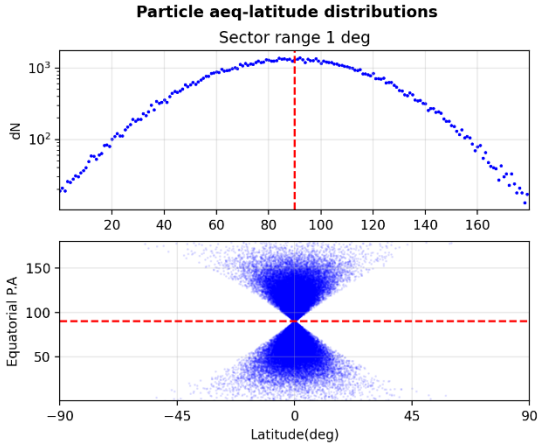
Figure 40: Single particle motion and its interaction with an 1nT electromagnetic wave. Left diagrams illustrate adiabatic motion (noWPI) and right diagrams illustrate wave-particle interaction (WPI) with the wave being introduced at t=1s.
From top to bottom, diagrams feature the respective plots of the evolution over time for the equatorial PA, the latitude, and the eta variation. Significant shifts in equatorial PA and eta variation become evident when the wave is introduced into the simulation. The particle attains higher altitudes and becomes detectable by the satellite after interacting with the wave.

Figure 40 shows that in the first simulation, where the particle undergoes adiabatic motion, it's evident that *aeq* is constant, while both the $deta_{dt}$ and *lamda* exhibit periodic variations. Conversely, in the second simulation involving the particle's interaction with the wave, the particle enters a state of resonance with the wave, resulting in significant fluctuations in the equatorial PA. During this simulation, the satellite is positioned at a latitude of 30 degrees ($lamda = 30^{\circ}$). When the particle is performing adiabatic motion, it oscillates at lower latitudes and is undetectable by the satellite. However, following its interaction with the wave, the particle acquires the necessary acceleration to ascend to higher latitudes (absolute value) above 30 degrees. Consequently, it traverses the satellite's position.

6.2 Simulations of particle distributions

In this section, the focus shifts to simulations involving particle distributions. Emphasizing the objective of simulating particles with distributions akin to those observed in nature, particularly within the Van Allen belts, underscores the importance of crafting these realistic distributions prior to their integration into the simulation. To establish a realistic particle distribution, it is essential to consider the valid domain for particle parameters. By addressing the following considerations, it becomes possible to design more realistic distributions, like the ones detailed in Section 2.4.

In the following table, two examples of normal distributions are generated within a population of 100,000 particles. The values for the mean and standard deviation are also defined. The top plots display the PADs, showcasing particle counts categorized in 1 degree PA bins. The bottom plots demonstrate the relationship between PA and latitude in degrees. The bins are parameters of the Python script “compare.py” that generates these plots and can be adjusted as needed.

Normal PAD - mean: 90 degrees - standard deviation: **20** **Normal latitude distribution** - mean: 0 degrees - standard deviation: **20** P.A bins: 1 degree	**Normal PAD** - mean: 90 degrees - standard deviation: **30** **Normal latitude distribution** - mean: 0 degrees - standard deviation: **30** P.A bins: 1 degree
	

Top diagrams: Particle flux or number of particles per pitch angle degree
Bottom diagrams: Relationship between equatorial pitch angle and latitude

Figure 41 illustrates the data distributions using pie charts.

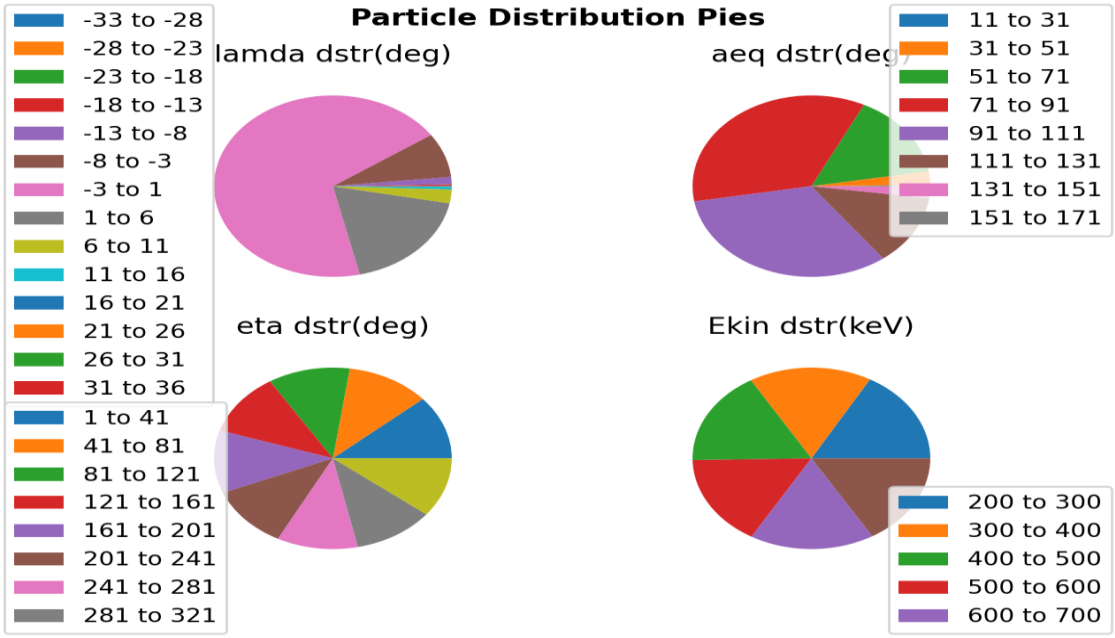


Figure 41: Distribution pies. The colored sectors indicate the ranges to which particles belong. The size of each sector reflects the particle count within that particular range. Notably, latitude (lamda) and equatorial PA (aeq) follow a normal distribution, while eta and kinetic energy (Ekin) adhere to a uniform distribution.

Below is an example of a particle distribution simulated twice in distinct environments. Two separate simulations, each lasting 56 seconds, were executed using the method described in Section 4.6.

1. The first simulation spanned 56 seconds and did not involve an electromagnetic wave. During this period, the particles underwent adiabatic motion.
2. In the second simulation, particles underwent adiabatic motion for 50 seconds. Subsequently, an electromagnetic wave with an intensity ranging from 8 to 12 pT (picotesla) was introduced into the simulation for the remaining 6 seconds.

Other parameters include *L-shell* set to 5; initial wave characteristics include a frequency of 2 kHz, a resonance number of 1 to achieve normal resonance, an initial wave normal angle of 0.001 degrees, wave power of 1, and a wave pulse duration of 0.1 seconds. Additionally, the Runge-Kutta step is set to 0.00001, and the telescope's latitude is 0 degrees, indicating that the satellite is positioned on the equator. All these parameters were defined in the "*constants.h"* header file of the repository. Figure 42 illustrates the particles detected by the satellite as discussed in Section 4.7.

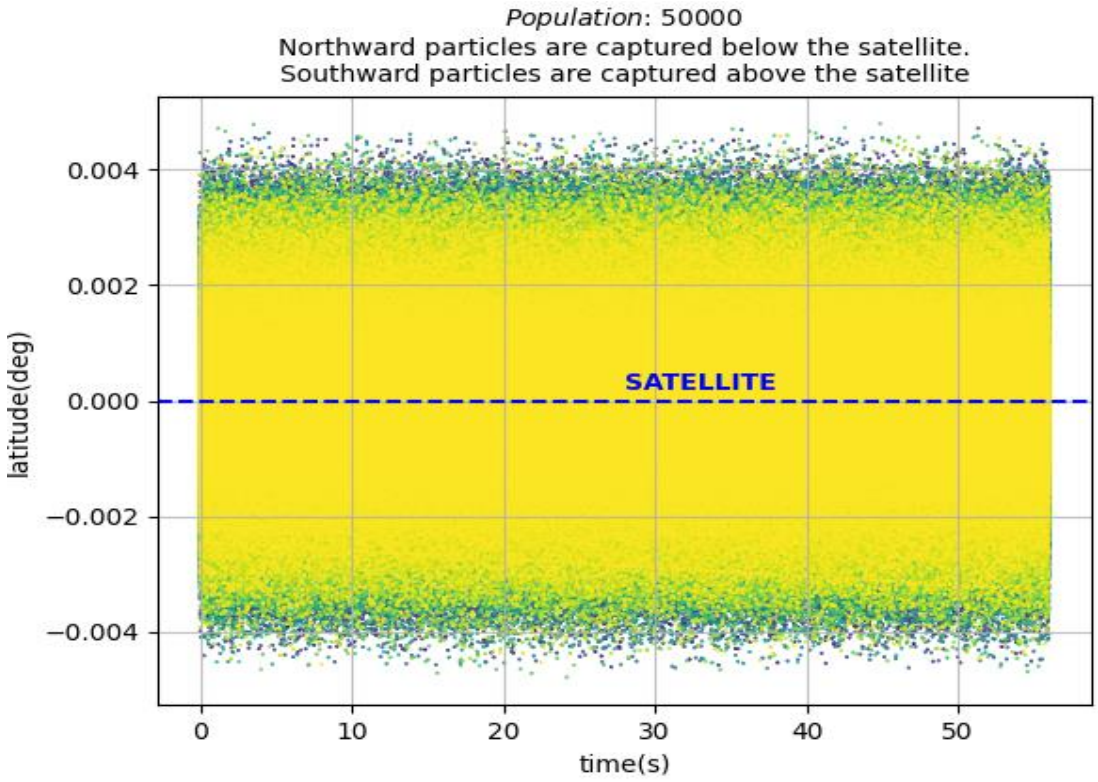


Figure 42: Detected particles crossing the satellite. Particles in adiabatic motion for 56 seconds.

In the second simulation, where a wave is introduced, the recorded particles are depicted below.

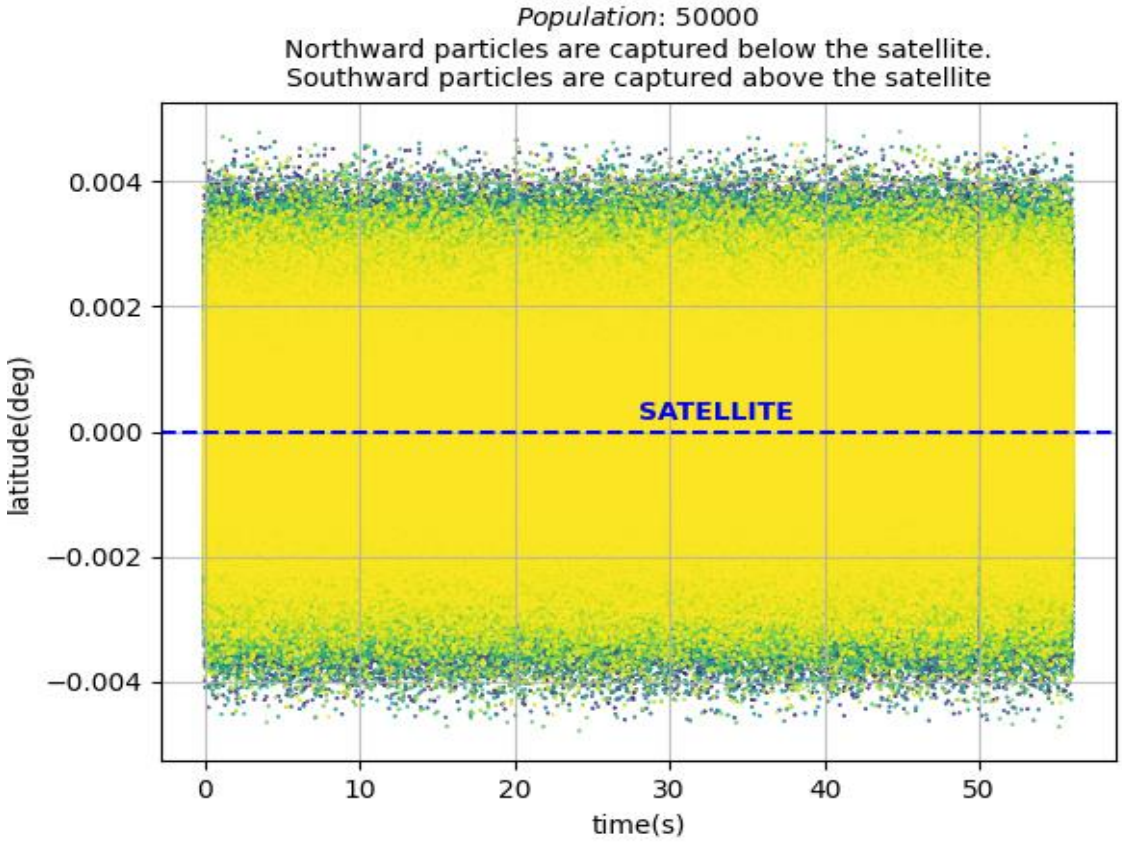


Figure 43: Detected particles crossing the satellite. At 50 seconds a wave enters for the following 6 seconds of the simulation.

When comparing the two diagrams above, one can observe the differences that arise after the wave is introduced at the 50-second mark. Certain particles exhibit distinct positions between the two simulations after introducing the wave. To facilitate a more comprehensive comparison between the simulations, an additional diagram can be produced that illustrates the *total particle flux* detected. In this diagram:

- Black dots represent the total particle flux recorded during the first simulation (without the wave) at 2-second intervals.
- Red dots represent the total particle flux recorded during the second simulation (with the wave introduced at 50 second mark) at 2-second intervals.

The interval size is subject to variation and is controlled by a variable within the Python script responsible for generating this diagram (“*compare.py”)*. This revision provides a clearer comparison between the simulations.

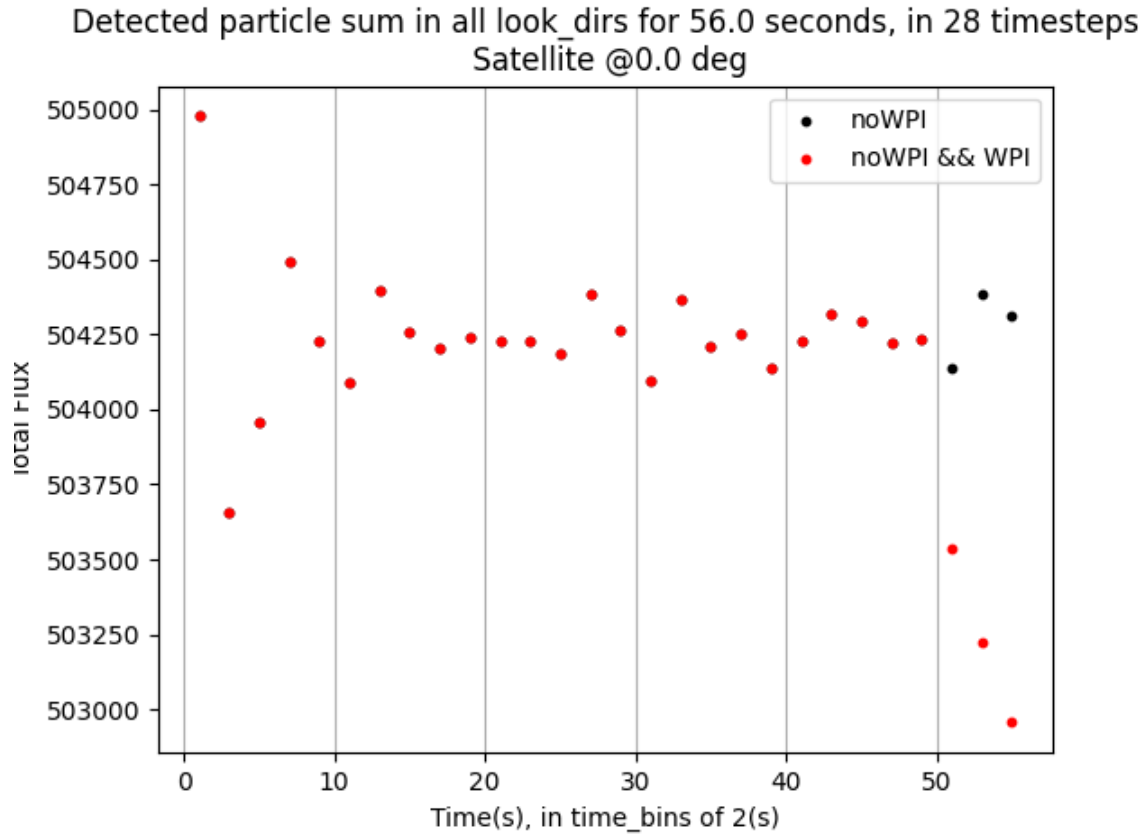


Figure 44: Total particle flux from all look directions in 2 second time bins. For the initial 50 seconds, the red dots overlap with the black ones. However, when the wave is introduced, the total particle flux in the WPI simulation starts to decrease.

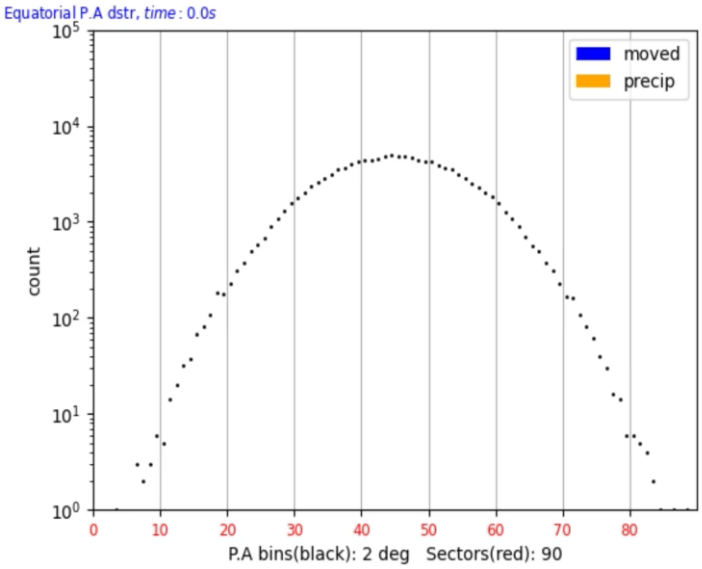
As the simulation of adiabatic particle motion unfolds, a pattern emerges: the total particle flux remains relatively stable within a range of 504,000 to 504,500 particles. However, at the 50-second mark when the electromagnetic wave is introduced and interacts with the particles, a noticeable decline in particle flux becomes apparent. Although some flux variation was evident earlier, the significant drop after 50 seconds strongly indicates that the electromagnetic wave plays a pivotal role in influencing the overall particle flux.

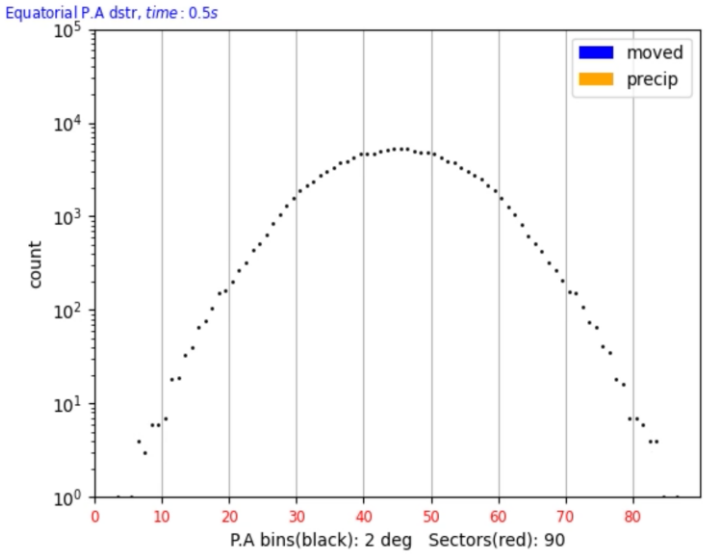
As introduced at the beginning of this chapter, the primary focus lies in examining particle distributions throughout the simulation run. To achieve this, a routine was developed to generate video files in *.mp4* format. As these videos cannot be incorporated in this document, certain frames will be directly extracted. The sector division method is elaborated in Section 4.7. In this example the total incoming particle flux is divided into 90 sectors, each spanning 2 degrees for a total view of 180 degrees. The initial sector captures particles within the 0 to 2-degree range, the second sector within 2 to 4 degrees, and so forth. Every one of the following diagrams corresponds to a frame extracted from the video file. To provide a comprehensive but manageable overview, a total of $4 \times (4 + 1) = 20$ movie-frames are presented in this document. These frames are grouped into four categories *Movie-frames1* to *Movie-frames4* as outlined below:

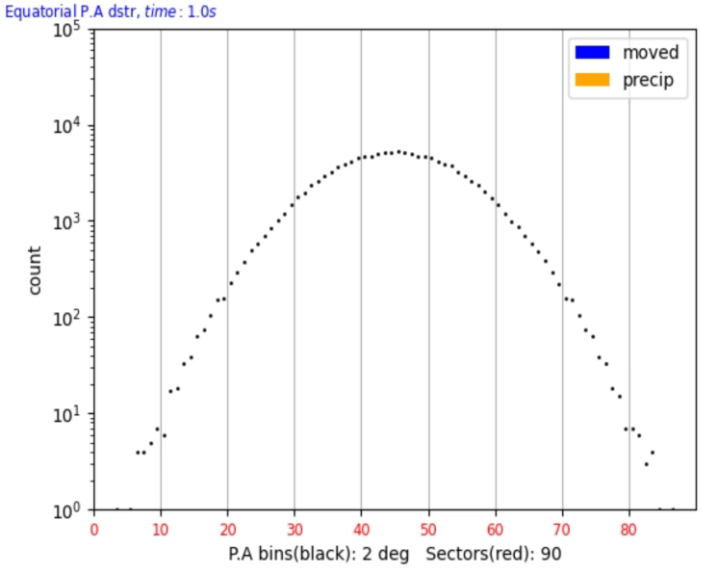
- Movie-frames1 represent the PADs at the beginning of the simulation.
- Movie-frames2 represent the PADs just before the wave is introduced.
- Movie-frames3 represent the PADs immediately after the introduction of the wave.
- Movie-frames4 represent the PADs as the simulation approaches its conclusion.

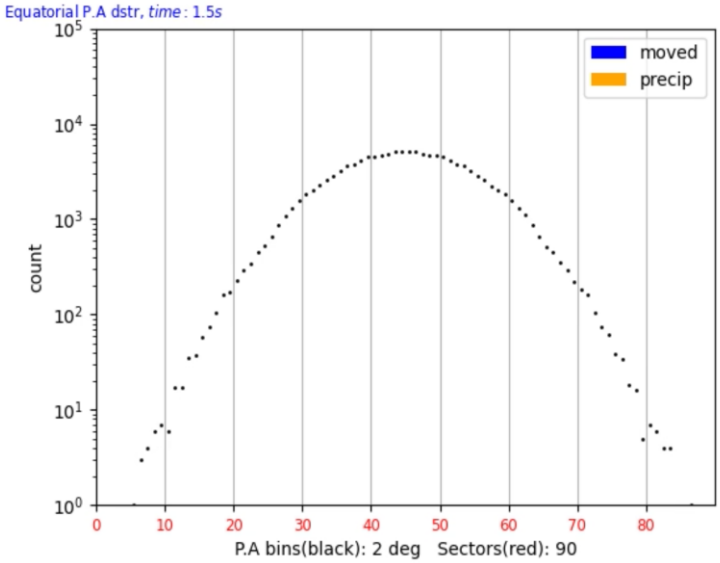
Each movie frame comprises four frames with a time bin of 0.5 seconds and one frame with a time bin of 2 seconds. Essentially, the fifth diagram in each set represents the cumulative flux of that set.

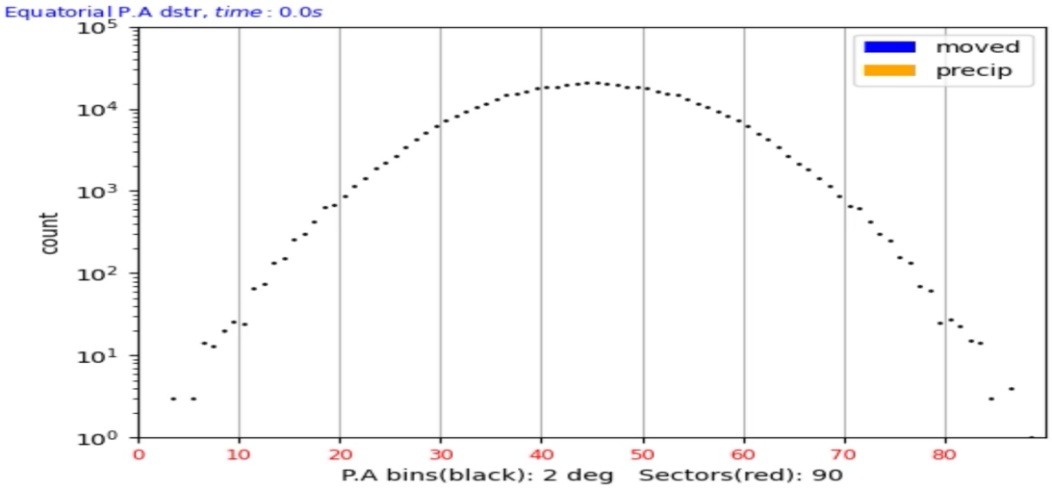
These movie frames illustrate the relative stability of the PAD until the introduction of the wave into the simulation. This observation strongly suggests that the wave has an observable impact on the behavior of the particles. Upon the wave's entry, there is a noticeable decline in flux within each sector, as indicated by the red dots in the diagram that represent the WPI simulation. In contrast, the black dots represent the adiabatic motion simulation, and they overlap with the red ones until the introduction of the wave. To better observe changes in particle flux within each sector, dynamic bar charts are employed and represented in blue. These charts specifically highlight particles that have transitioned between pitch angle bins. Evidently, the majority of particles that transition between different sectors are typically around 90 degrees. Furthermore, the red dots exhibit reduced flux levels at the pitch angle extremes, specifically near 180 and 0 degrees, corresponding to regions where the loss cone is defined. Based on this observation, it suggests that the wave is responsible for accelerating these particles, leading them to escape the simulation. The movie frames follow in the next pages.



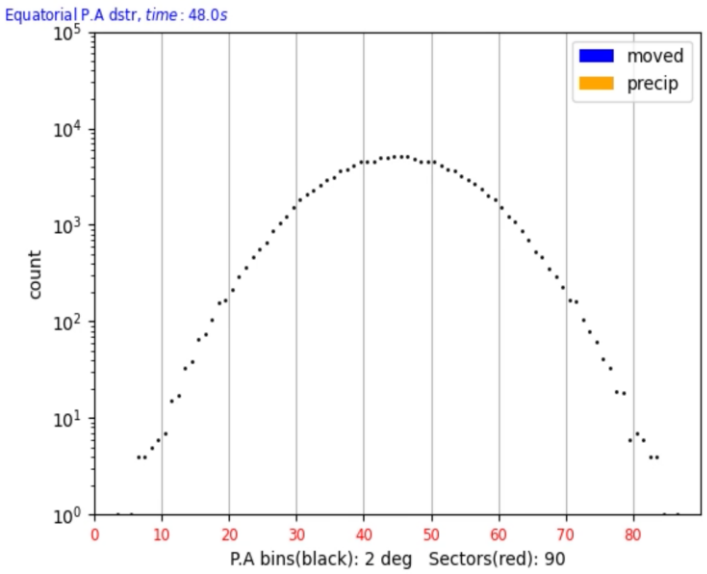


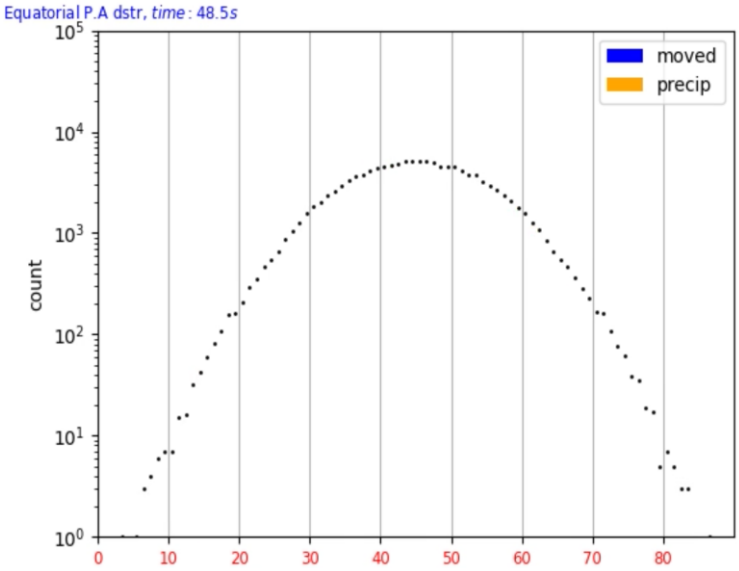


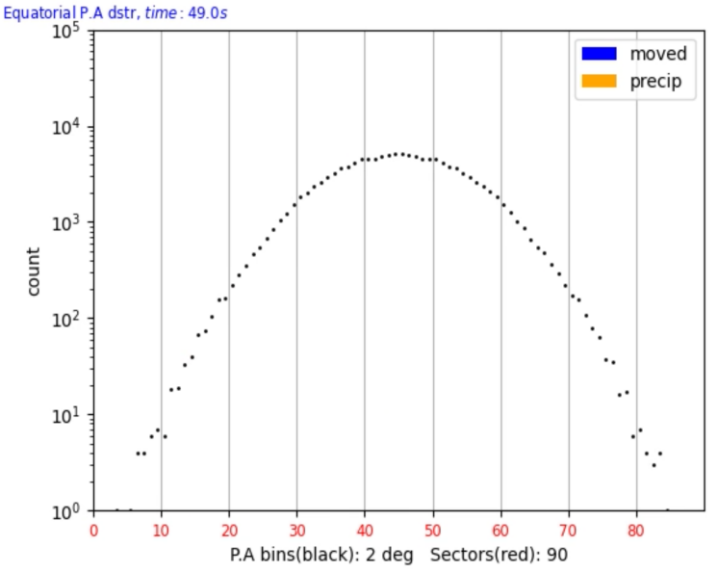


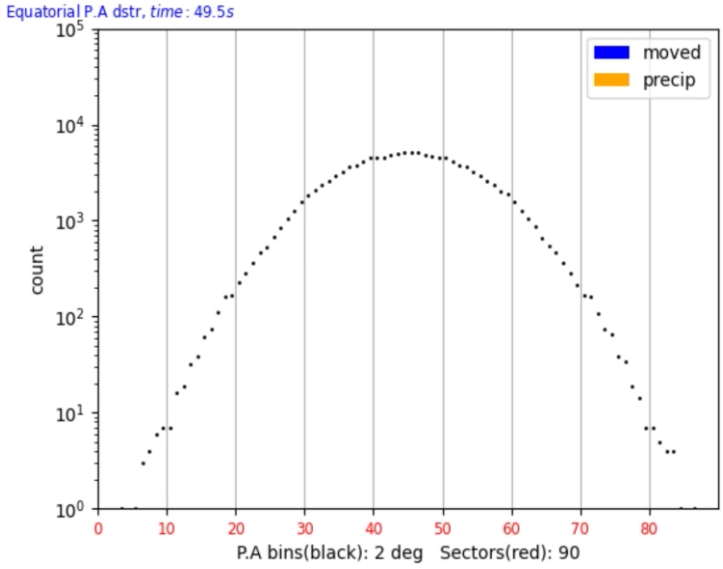


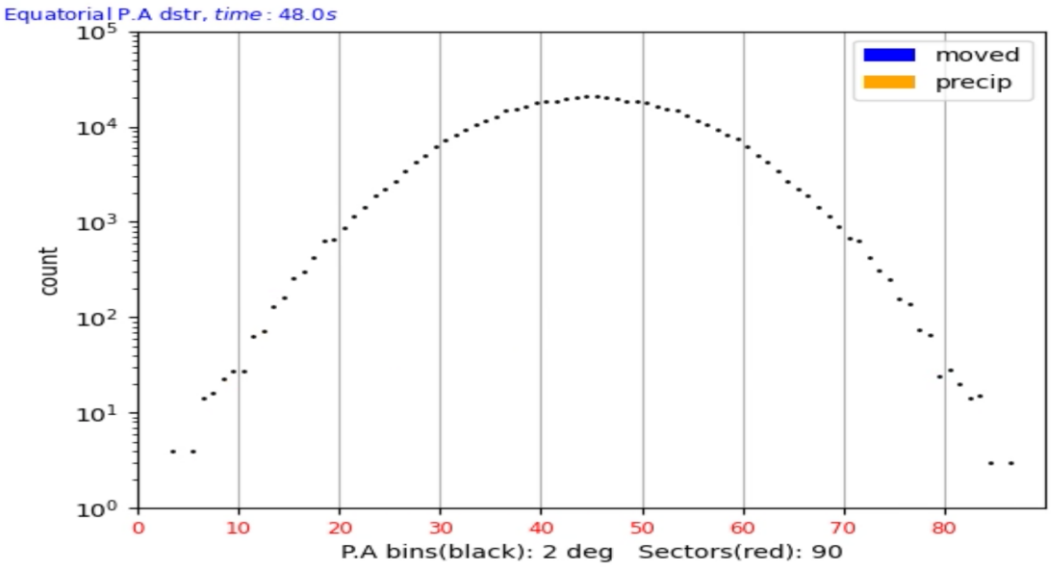
Movie-frames1: Particle flux tracked within 2-degree sectors at intervals of 0.5 seconds during the initial 2 seconds of the simulation. The fifth diagram represents the cumulative particle flux of these frames.



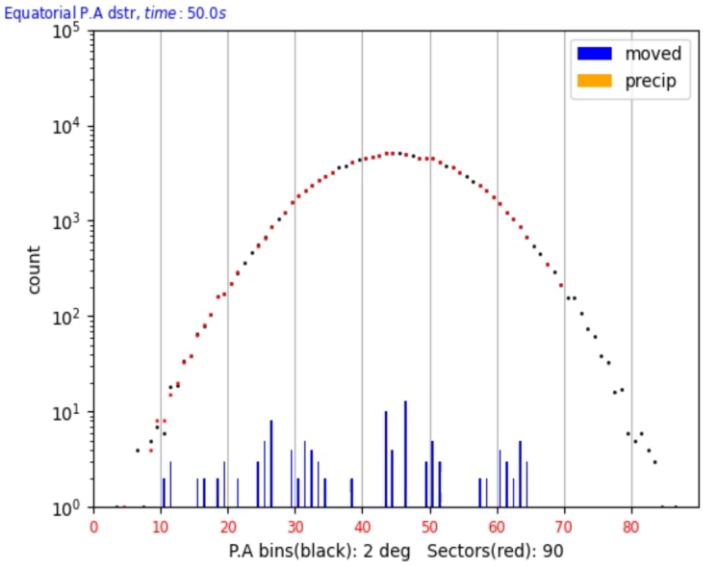


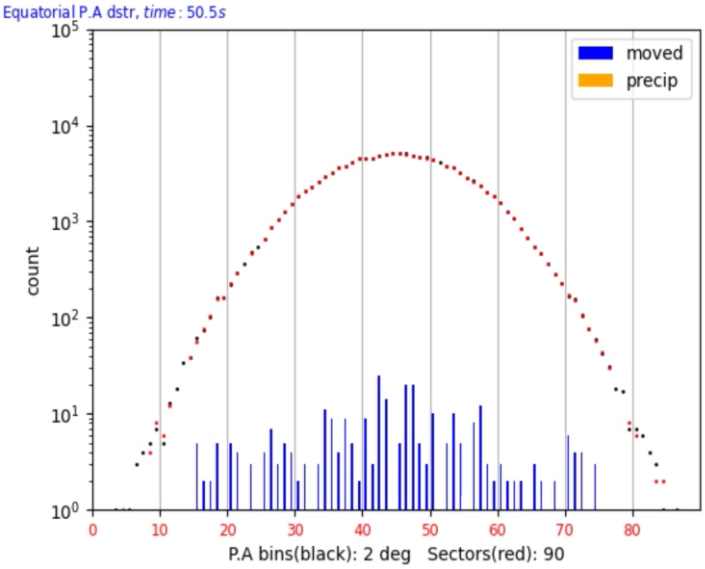


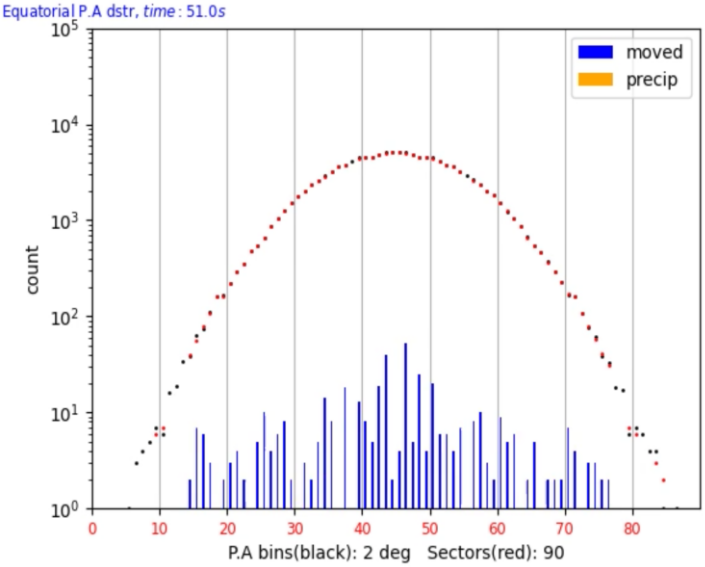


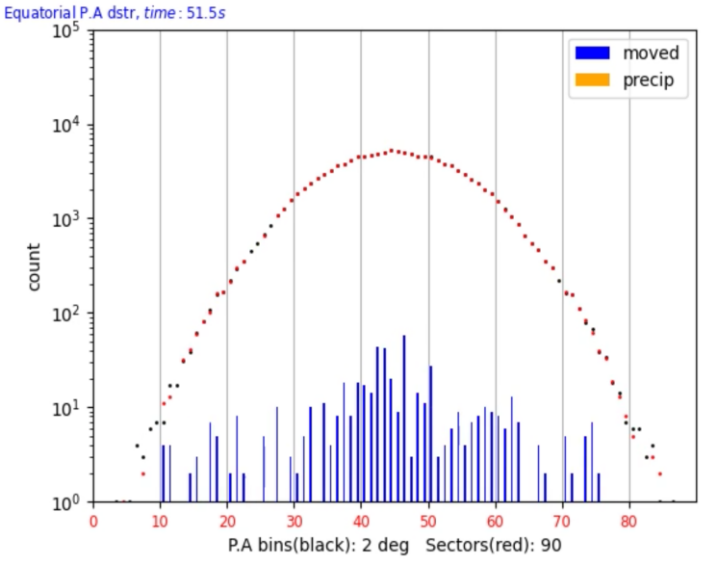


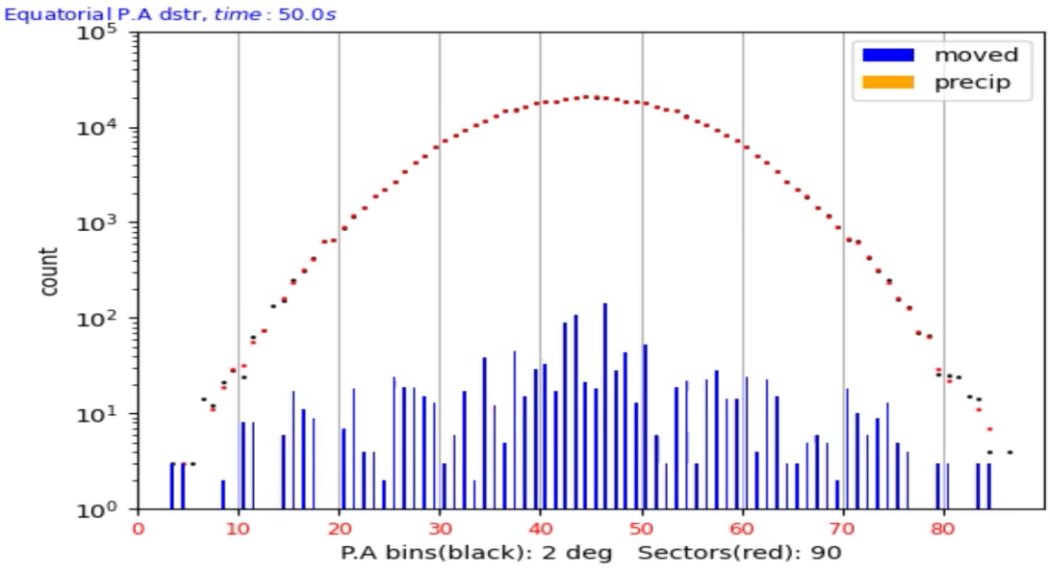
Mavie-frames2: Particle flux tracked within 2-degree sectors at intervals of 0.5 seconds during the last 2 seconds before the wave introduction. The fifth diagram represents the cumulative particle flux of these frames.



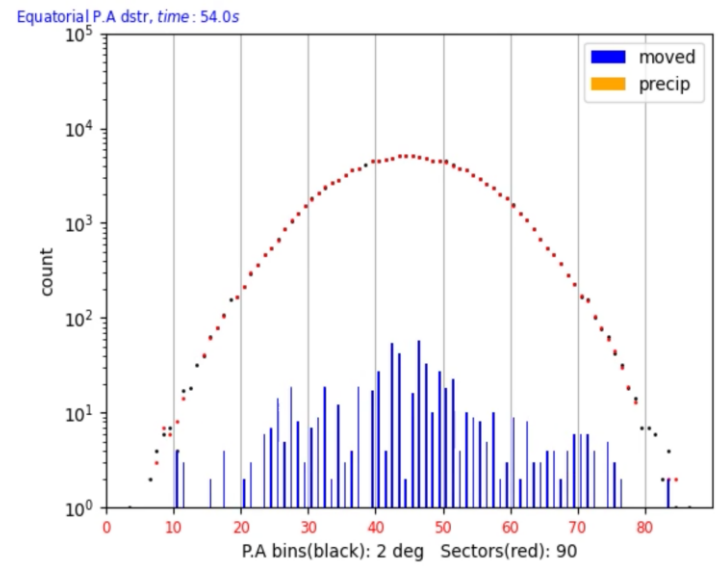


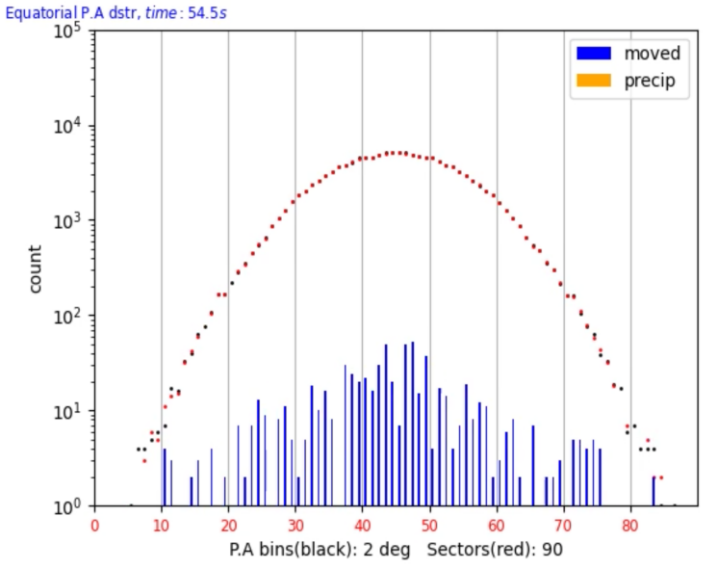


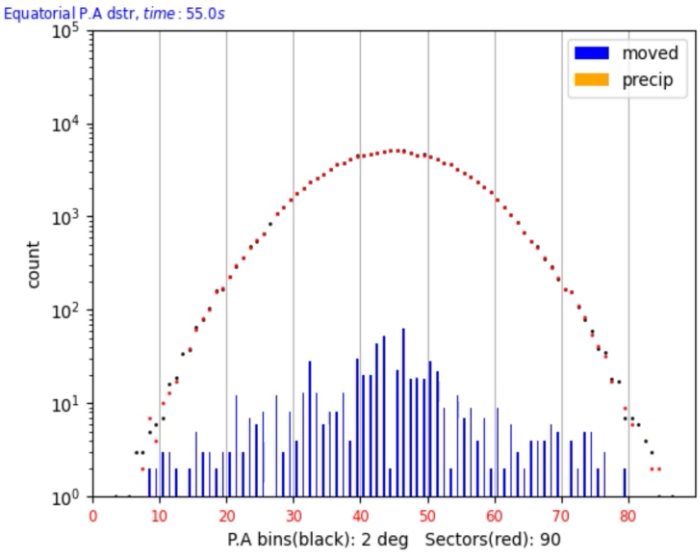


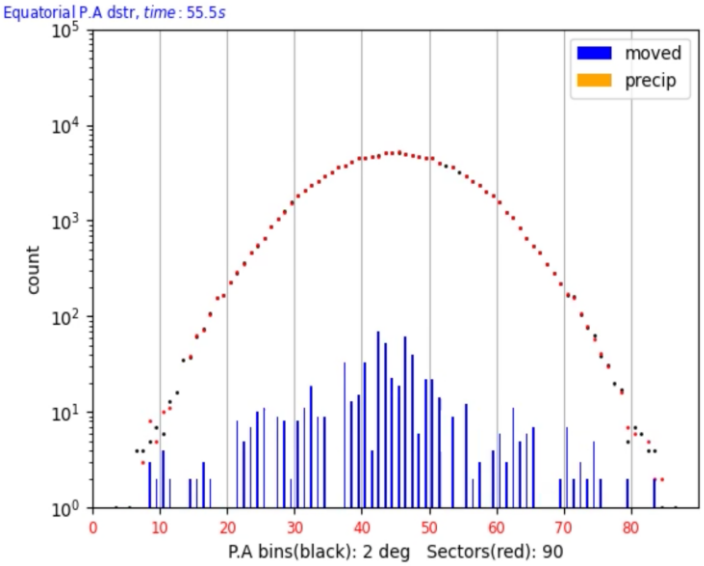


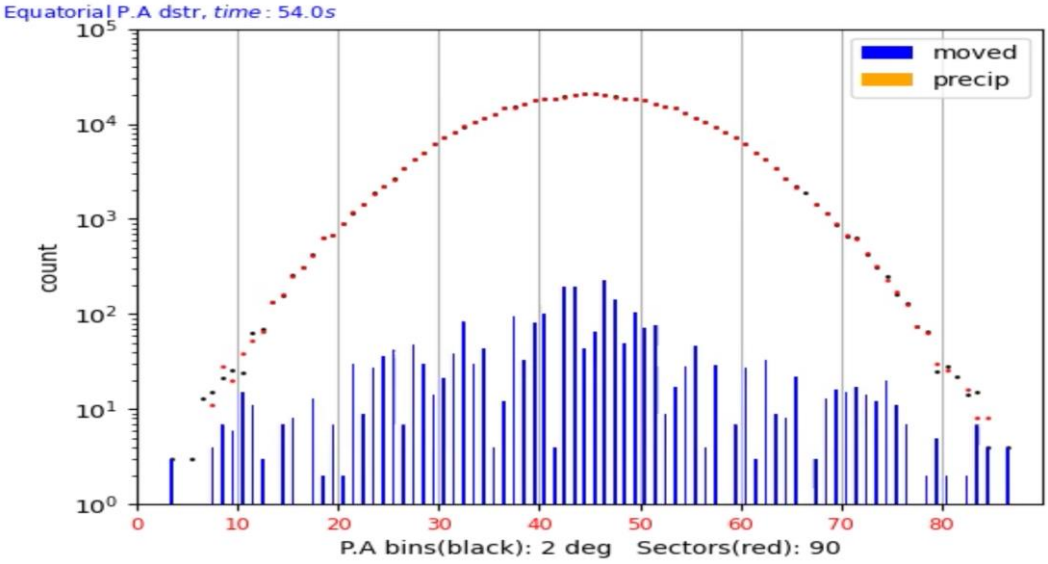
Movie-frames3: Particle flux tracked within 2-degree sectors at intervals of 0.5 seconds during the first 2 seconds after the introduction of the wave to the simulation. The fifth diagram represents the cumulative particle flux of these frames.











Movie-frames4: Particle flux tracked within 2-degree sectors at intervals of 0.5 seconds during the last 2 seconds of the simulation. The fifth diagram represents the cumulative particle flux of these frames.

Chapter 7

Setup

7.1 Simulation development and execution systems

Initially, all tools were installed on a Linux operating system, specifically opting for the Linux Mint distribution because of its simplicity and ease of use. However, any other Linux distribution can be utilized. Simulations for a large population of particles were conducted using the high-end computing resources of the Electromagnetic Theory Laboratory, namely the servers *Odysseus* and *Autolycus*. The specifications of the systems in which the algorithm was designed and underwent testing runs are described in the following table:

System Specifications						
	OS	Chip	Cores/Socket	Threads/Core	Threads	RAM
PC	Linux Mint 20.2	Intel i5-8250 @ 1.6GHz	4	2	8	12GB
Autolycus	Ubuntu 22.04.2 LTS	Intel i9-10850K @ 3.60GHz	10	2	20	126GB
Odysseas	Ubuntu 22.04.2 LTS	Intel i9-9900K @ 3.60GHz	8	2	16	126GB

Figure 45: System specifications in which the algorithm was developed and executed.

7.2 Dependencies

The code was developed and executed using the following tools:

Compiler gcc 7.5.0

While most UNIX distributions typically come with a GNU compiler pre-installed, it's advisable to verify the installed version. The GNU Compiler Collection (*gcc*), among its capabilities, can also compile C++ code.

Debugger gdb 8.1.1 (optional)

The *GNU Project Debugger (gdb)* is also commonly found on UNIX distributions. To assure their installation, use the command `gcc -v` or `g++ -v` respectively in the command line. This will display information about the installed version following the statement "Using built-in specs." If they are not already installed, you can typically use your package manager, such as `apt`, to install them, as demonstrated below:

```
$ sudo apt update
$ sudo apt install build-essential
$ sudo apt install {installing package}
```

OpenMP 4.5:

To verify the *OpenMP* version on *Ubuntu/Debian Linux* for gcc versions 4.2.0 and later, you can use the following command:

```
$ echo | cpp -fopenmp -dM | grep -i open
```

This command will display information in the format: `#define _OPENMPyear/month`, where you can find the supported OpenMP version. Additional information about the OpenMP specifications can be found on the official OpenMP website at https://www.openmp.org/specifications/.

HDF5 1.10.4

HDF5, which stands for *Hierarchical Data Format* version 5, is a data file format and library for managing and storing large and complex datasets. To install HDF5, you typically need to download and install the HDF5 library and, optionally, any programming language bindings you plan to use (e.g., Python, C++, or Java). Below are instructions for installing HDF5 with apt package manager:

```
sudo apt-get update
sudo apt-get install libhdf5-dev
```

Python 3.8.10

Many Linux distributions come pre-installed with a Python interpreter. The following command confirms a successful installation:

```
python –version
```

If Python is not already installed in the system's Linux distribution, it can be easily installed using the package manager specific to the system's distribution. For example, for apt package manager:

```
sudo apt-get update
sudo apt-get install python3
```

Install Python Libraries: numpy, matplotlib, h5py, etc.

For convenience, the project repository includes a "*requirements.txt*" file listing all the necessary packages for installation. The following steps are required to install the python libraries:

1. Create virtualenv:

   ```
   python3 -m venv tracer_venv
   ```

   Name the environment 'tracer_venv' to correctly ignore it when using version control.

2. Then activate the environment:

   ```
   source tracer_venv/bin/activate
   ```

3. Upgrade pip:

   ```
   python3 -m pip install --upgrade pip
   ```

4. Install requirements:

   ```
   python3 -m pip install Particle-Tracing-Earth/requirements.txt
   ```

Chapter 8

Usage

8.1 Command Line Interface (CLI)

The program's flow diagram, located at the end of Chapter 4, serves as a valuable guide for the execution of the tracer. To use the particle simulator, please follow these steps:

i. **Ensure Tool Installation:** Before proceeding, please ensure that the dependencies mentioned in the previous chapter are satisfied.
ii. **Clone the Package:** Clone the package and its necessary submodules from Git using the following command:

```
git clone --recursive https://github.com/Vasichar11/Particle-Tracing-Earth
```

This recursive clone operation will also fetch required submodules for the simulator's functionality, specifically the *HighFive* repository. HighFive offers a contemporary header-only interface, compatible with C++11, for seamless interaction with libhdf5.

Once all dependencies are satisfied and the cloning process is complete, the next steps include configuration, building, execution, and visualization. Here's a breakdown of each step:

1. **Configuration**

   Currently, the simulation can be configured by editing the header file located at this path: "*Particle-Tracing-Earth/Telescope/src/headers/constants.h*." Within this file, the program's parameters are defined. Any parameters modification requires recompiling of the project to successfully apply the changes. In the future, a more user-friendly configuration method is planned. This could involve the introduction of a configuration file, such as an *.ini* or *.xml* file. This approach will eliminate the need for project recompilation after parameter modification.

2. **Building**

   A *makefile* is employed for building the project in order to create executables. It is an utility that enables the automation of the compilation and the linking of the object files by defining a series of tasks to execute. The required commands vary depending on the type of simulation, as illustrated in the table in Figure 46:

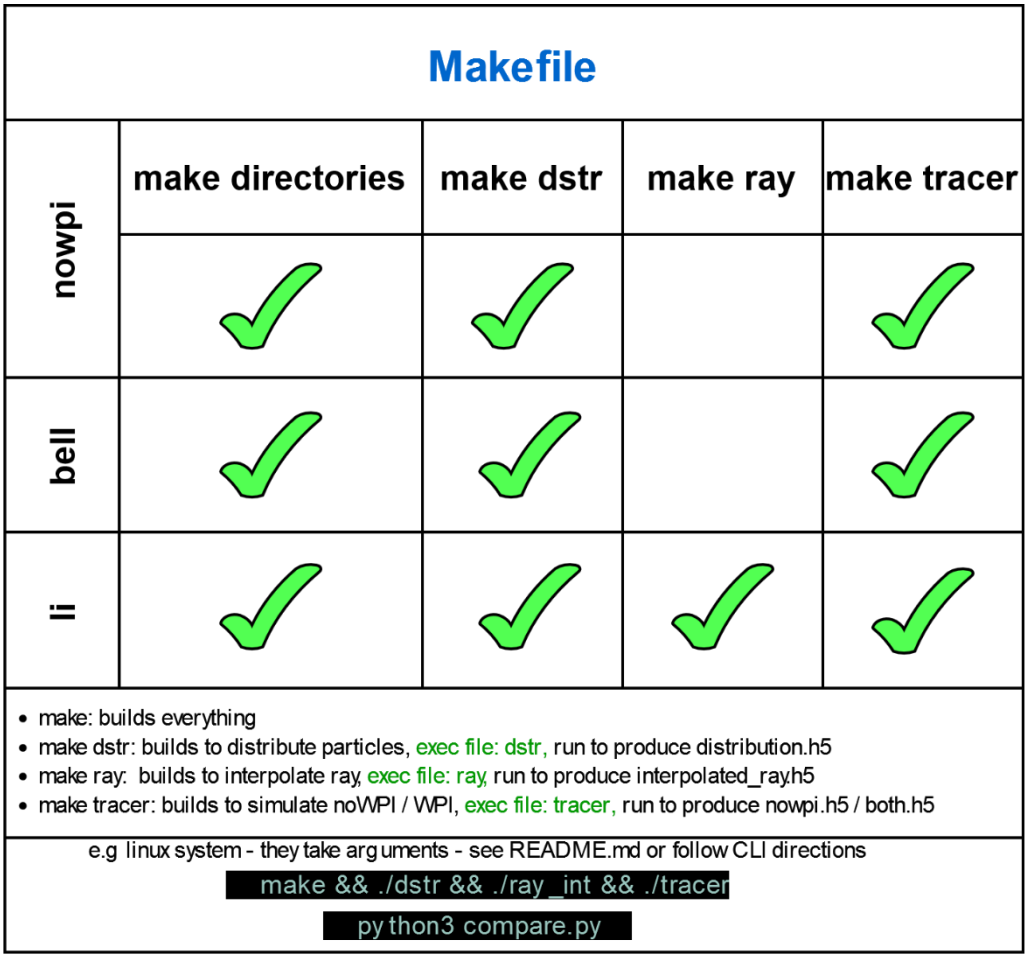


Figure 46: Building process. Distrinct programms require dinstinct builds. Everything can be built with "make all" command.

For instance, simulations that don't require a wave component won't need the "*ray*" executable. Here are the make commands explained:

a) The command "`make directories`" generates the necessary folders for storing the simulation output and media files.
b) The command "`make dstr`" creates an executable file named "*dstr*" for handling particle distributions.
c) Similarly, "`make ray`" generates the "*ray*" executable to interpolate the ray.
d) Lastly, "`make tracer`" generates the "*tracer*" executable for the simulation.

You can execute all these actions (a-d) at once by running "`make`" in the command line.

3. Execution

Once the executable files have been successfully created, you can proceed with their execution by doing the following:

a) To generate a particle distribution, use the command with five command line arguments, which are described in detail in Section 4.2. The command is:

```
./dstr <arg1> <arg2> <arg3> <arg4>
```

For example:

```
./dstr normal normal uniform uniform
```

This will create a population of particles with a normal distribution for pitch angle and latitude, and a uniform distribution for eta and Ekin.

b) To calculate the magnetic fields of the wave at precise time intervals through interpolation:

```
./ray
```

The time intervals match the time step of the Runge Kutta 4 method specified in the "*constants.h*" header, namely parameter *"h"* as described in Section 4.3.

c) Finally the commands for the particle tracer:

```
./tracer <noWPI_time> <WPI_time>

mpirun -np <p> ./tracer <noWPI_time> <WPI_time>
```

The simulation will model particles experiencing adiabatic motion for `<noWPI_time>` seconds. Afterward, it will shift to simulating a wave-particle interaction for `<WPI_time>` seconds.
For example the command:

```
./tracer 50 6
```

It will model particles undergoing adiabatic motion for 50 seconds, and at the 50-second mark, the wave interaction simulation will commence, lasting until the 56-second mark.

4. Visualization

a) To visualize the initial particle distribution that enters the simulation:

```
python3 src/visualization/distribution_plot.py
```

b) To visualize the electromagnetic wave's fields and latitude:

```
python3 src/visualization/Ray_plot.py
```

c) To compare simulations and draw results:

```
python src/visualization/compare.py
```

8.2 Graphical User Interface (GUI)

To facilitate the execution of the simulation, a Graphical User Interface (GUI) is being developed. Please note that this GUI is a work in progress and is expected to undergo further improvements in the future. To ensure you have the latest version, it is recommended to check the most recent commit in the master branch and access the latest application.

As of now, all configuration variables are grouped together within the same tab, categorized by the nature of each parameter, while hovering the cursor over the parameter names will reveal their descriptions. At the bottom of the application, there is a console displaying messages that are valuable for the execution of the simulation. Moreover, the GUI incorporates dropdown menus that offer the convenience of multiplication factors on input values, which can also be defined using scientific notation. Each parameter is associated with a physical unit displayed on the far right. Following this description, screenshots of the application are provided, showcasing the parameters that can be adjusted. Let's start by examining the parameters related to the wave:

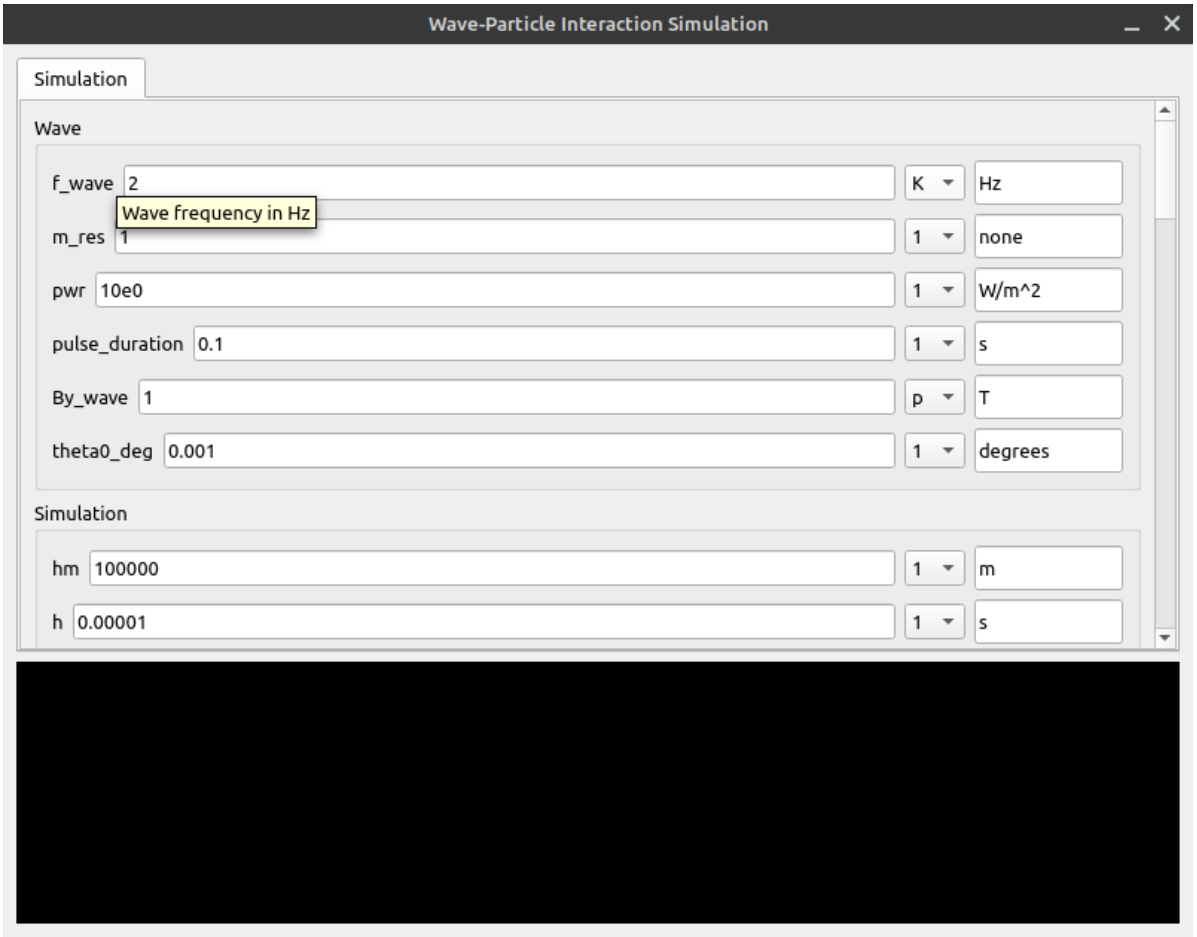


Figure 47: Tracer GUI. The frequency of the wave is one of the parameters and is set to 2KHz.

Moving on, certain general parameters essential for simulating the distribution and determining the satellite's position are listed. Once more, to gain a deeper understanding of each parameter's role, simply hover the cursor over the parameter names, and a detailed description will be provided:

Figure 48: Tracer GUI. General simulation, distribution, and satellite parameters.

Finally, the parameters for the distribution in PA, latitude, eta parameter, and kinetic energy are listed. For example, let's consider the definition of the PAD:

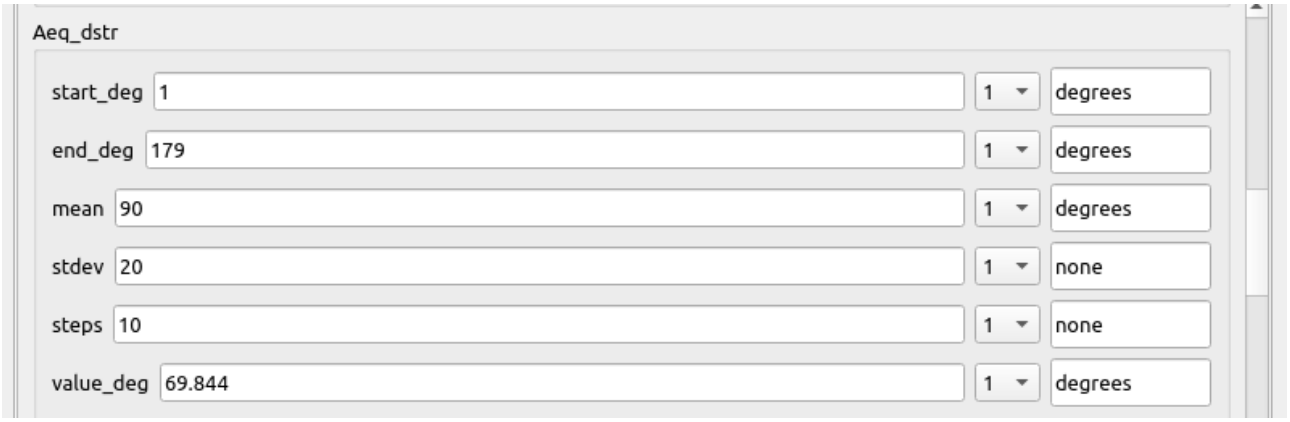


Figure 49: Tracer GUI. PAD parameters.

Once all the parameter values have been defined, you can initiate the building process by clicking the *Build* button:

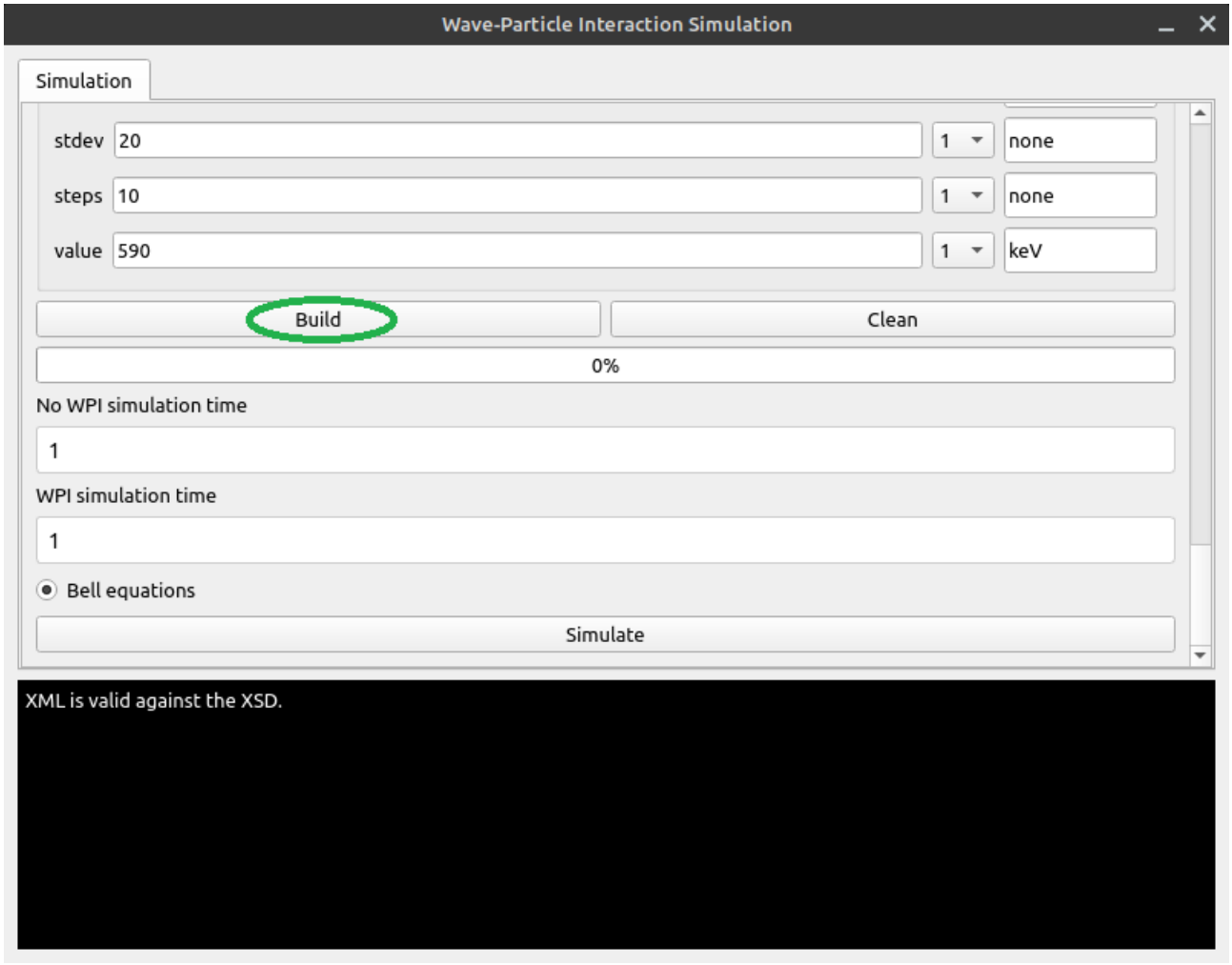


Figure 50: Tracer GUI. Building process.

The parameter values are programmatically defined in an *XML* (Extensible Markup Language) configuration file and when the "*Build*" button is clicked, the parameter values will undergo validation using an *XSD* document. XSD (XML Schema Definition) is a specification used for describing the structure and constraints of the XML configuration file and serves as a blueprint for valid simulation configurations. This XSD document incorporates specific rules and restrictions for the parameters, which are pre-established but can also be modified. If the values conform to the predefined restrictions, a confirmation message will appear in the console at the bottom of the application.

Following the building process, the user can specify the simulation duration of the adiabatic motion and for the wave-particle interaction. Additionally, there is a radio button that enables the execution of the simulation using the "*Bell equations"*, denoted as equations (3.2f), (3.2g), (3.2h), and (3.2i) as discussed in Section 3.2. The default execution mode for the simulation utilizes the revised set of equations, (3.2j), (3.2k), (3.2l), and (3.2m).

To initiate the simulation, adjust the simulation time accordingly and click the *Simulate* button:

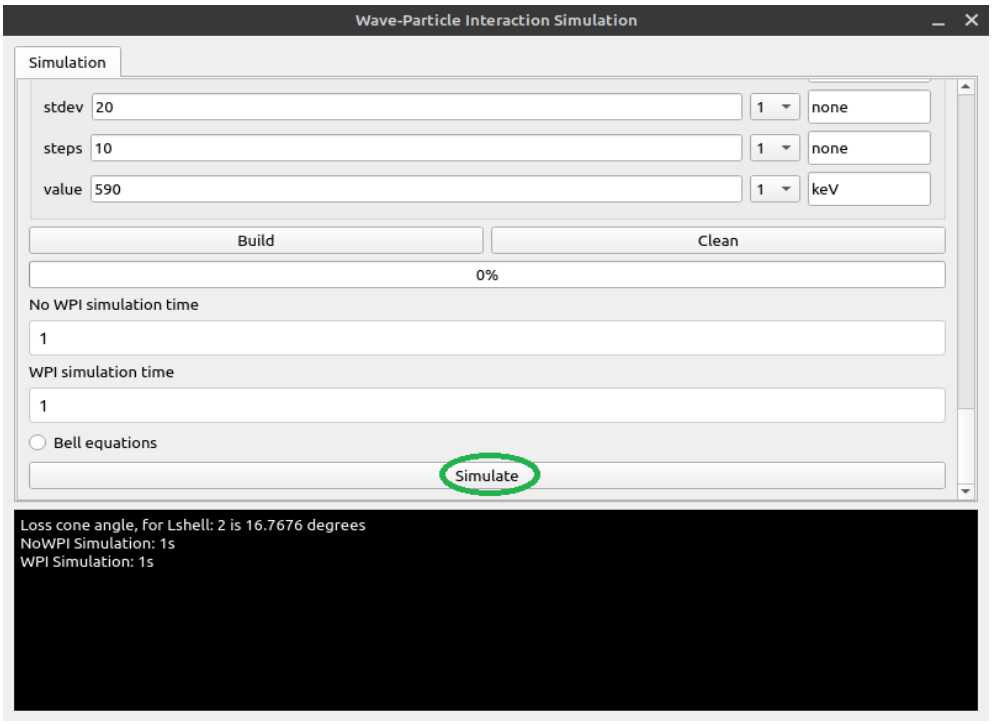


Figure 51: Tracer GUI. Simulation process.

Then the simulation starts, and the resulting data is currently stored in the *Particle-Tracing-Earth/output/files* directory. A future plan is to incorporate the visual results in the application. To begin a new simulation, first clear the build data and executables by clicking the *Clean* button:

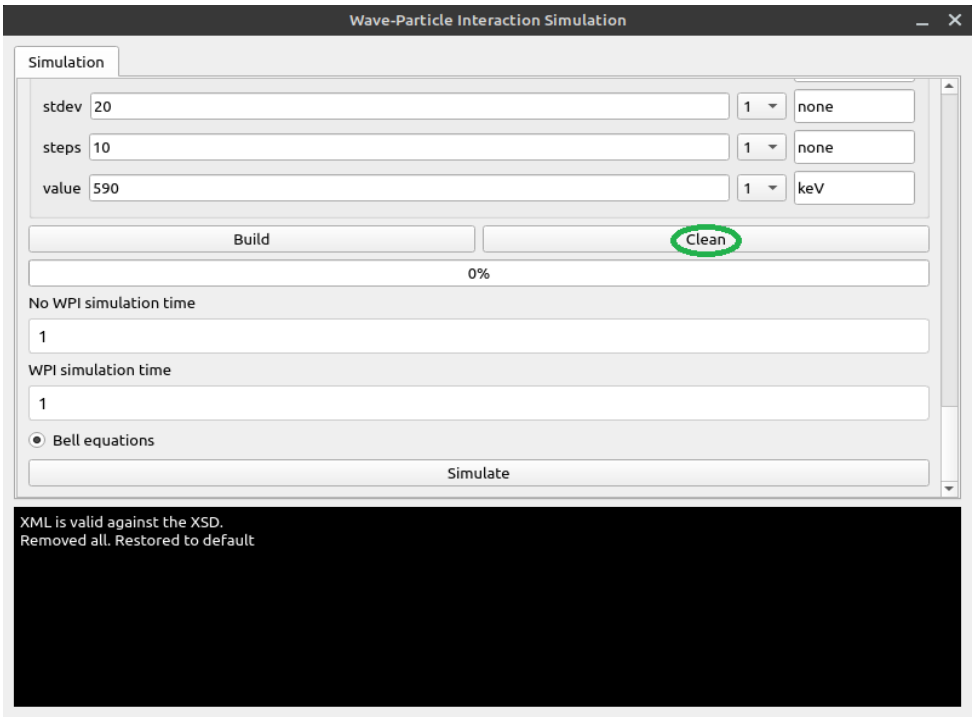


Figure 52: Tracer GUI. Cleaning process.

Considering the numerous possibilities and potential improvements for the application, one of the upcoming plans is to organize the parameter inputs into corresponding tabs and integrate the visualization directly into the application. It is advisable to check the master branch for updates before using the GUI application.

Bibliography

[1] Tourgaidis Stelios, Sarris Theodore, 2022, Wave-particle interactions toolset: A python-based toolset to model wave-particle interactions in the magnetosphere, Frontiers in Astronomy and Space Sciences, https://www.frontiersin.org/articles/10.3389/fspas.2022.1005598, doi:10.3389/fspas.2022.1005598, ISSN:2296-987X
[2] Spreiter et al., 1966, Hydromagnetic flow around the magnetosphere.
[3] Stern, David P.; Peredo, Mauricio (20 November 2003). "The Magnetopause". The Exploration of the Earth's Magnetosphere. NASA. Retrieved 19 August 2019.
[4] Ridpath, I. (2012). A Dictionary of Astronomy.: Oxford University Press. Retrieved 5 Aug. 2023.
[5] Inan, U.S., T.F. Bell, J. Bortnik, and J.M. Albert, Controlled precipitation of radiation belt electrons, Journal of Geophysical Research-Space Physics, 108 (A5), 1186, doi:10.1029/2002JA009580, 2003.
[6] Baker, D. N. et al. Sciencehttp://dx.doi.org/10.1126/science.1233518 (2013)
[7] Van Allen Probes MagEIS, ECT Science Operations and Data Center (http://www.rbsp-ect.lanl.gov).
[8] Rodger, C. & Clilverd, M. & Ulich, Th & Verronen, P. & Turunen, Esa & Thomson, N.. (2006). The atmospheric implications of radiation belt remediation. Annales Geophysicae. 24.10.5194/angeo-24-2025-2006.15.
[9]Lauben et al. 2001 Precipitation of radiation belt electrons induced by obliquely propagating lightning-generated whistlers
[10] Dungey, J.W., Loss of Van Allen Electrons Due to Whistlers, Planetary and Space Science, 11 (6), 591-595, 1963.
[11] Chen et. al. (2014), REPAD: An empirical model of pitch angle distributions for energetic electrons in the Earth’s outer radiation belt, J. Geophys. Res. Space Physics, 119, 1693–1708, doi:10.1002/2013JA019431.
[12] Sibeck, D. G., R. W. McEntire, A. T. Y. Lui, R. E. Lopez, and S. M. Krimigis (1987), Magnetic field drift shell splitting.
[13] Gannon, J. L., Li, X., & Heynderickx, D. (2007). pitch angle distribution analysis of radiation belt electrons based on Combined Release and Radiation Effects Satellite Medium Electrons A data. Journal of Geophysical Research, 112, A05212. https://doi.org/10.1029/2005JA011565 distributions during storm time electron acceleration to megaelectronvolt energies. Journal of Geophysical Research, 108(A1), 1016.
[14] Schulz, M., & Lanzerotti, L. (1974). Particle diffusion in the radiation belts.
[15] Horne, R. B., Meredith, N. P., Thorne, R. M., Heynderickx, D., Iles, R. H. A., & Anderson, R. R. (2003). Evolution of energetic electron pitch angle.
[16] Baker, D. N., S. G. Kanekal, R. B. Horne, N. P. Meredith, and S. A. Glauert (2007), Low-altitude measurements of 2-6 MeV electron trapping lifetimes at 1.5 < L < 2.5, Geophys. Res. Lett., 34, L20110, doi:10.1029/2007GL031007.
[17] Knoll, Glenn F. "Radiation detection and measurement." (2010).
[18] Spacecraft Systems Engineering, 4th Edition Peter Fortescue (Editor), Graham Swinerd (Editor), John Stark (Editor).
[19] The Physics of Particle Detectors (Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology, Series Number 12) 1st Edition by Dan Green.
[20] Picture from The South Atlantic Magnetic Anomaly, Available from: https://www.researchgate.net/figure/Schematic-representation-of-a-particle-detector-The-collimator-aperture-covers-a-solid_fig18_303756778.
[21] PEEL, Energetic Electrons Precipitating at High Latitudes: PEEL Data from HotPay-2 Mission, J. Balá 1 , K. Kudela *,1 , T. Sarris 2 and I. Strhársk 1.

[22] Performances of the GLAST/LAT calorimeter for cosmic-proton background rejection; Study of a Gamma Ray Burst model with leptonic and hadronic components
Sylvain Guiriec 2007.
[23] New Measurement of the Antiproton-to-Proton Flux Ratio up to 100 GeV in the Cosmic Radiation O. Adriani et al. (PAMELA Collaboration).
[24] Shannon, C. E. (1949). Communication in the Presence of Noise. Proceedings of the IRE, 37(1), 10–21. https://doi.org/10.1109/jrproc.1949.232969.
[25] Bortnik, Jacob & Inan, Umran. (2004). Precipitation of radiation belt electrons by lightning-generated magnetospherically reflecting whistler waves.
[26] Goldstein, H., C. Poole, and J. Safko, Classical mechanics - Third Ed., Addison-Wesley Publishing Co., 2002.
[27] Lauben, D.S., U.S. Inan, T.F. Bell, and D.A. Gurnett, Source characteristics of ELF/VLF chorus,Journal of Geophysical Research-Space Physics, 107 (A12), 1429
[28] Salby, 1996 Salby, M.L., Fundamentals of atmospheric physics, Academic Press Inc., 1996.
[29] Bell, T.F., The Nonlinear Gyroresonance Interaction between Energetic Electrons and Coherent VLF Waves Propagating at an Arbitrary Angle with Respect to the Earths Magnetic-Field, Journal of Geophysical Research, 89 (A2), 905-918, 1984.
[30] Li, Jinxing & Bortnik, Jacob & Xie, Lun & Pu, Zuyin & Chen, Lunjin & Ni, Binbin & Tao, Xin & Thorne, Richard & Fu, Suiyan & Yao, Zhonghua & Guo, Ruilong. (2015). Comparison of formulas for resonant interactions between energetic electrons and oblique whistler-mode waves. Physics of Plasmas. 22. 052902. 10.1063/1.4914852.
[31] Lauben et al. 2001 Precipitation of radiation belt electrons induced by obliquely propagating lightning-generated whistlers.
[32] Inan, U.S., T.F. Bell, J. Bortnik, and J.M. Albert, Controlled precipitation of radiation belt electrons, Journal of Geophysical Research-Space Physics, 108 (A5), 1186, doi: 10.1029/2002JA009580, 2003.
[33] Orlova K. G., Y. Y. Shprits, (2011), On the bounce-averaging of scattering rates and the calculation of bounce period, Physics Of Plasmas, 18, 092904, doi:10.1063/1.3638137
[34] Gyration, bounce, and drift periods estimate of a particle trapped in the Earth's magnetic field, Solène Lejosne – Space Science Research, https://solenelejosne.com/bounce/