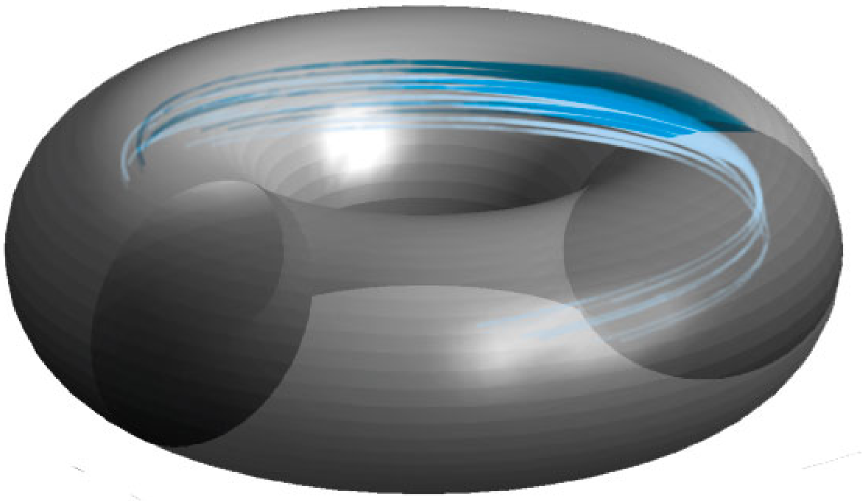
CHALMERS



Monte Carlo Simulation of Runaway Electrons

Thesis for the Degree of Master of Science

JAKOB RYDÉN

Department of Applied Physics
Chalmers University of Technology
Gothenburg, Sweden 2012

Abstract

Runaway electrons appear in tokamak plasmas during thermal quenches - disruptions that change the plasma conductivity. This gives rise to an accelerating electric field, which if, higher than the decelerating Coulomb friction force, can give electrons unlimited acceleration, resulting in relativistic particles, which may damage the first wall of the tokamak.

In this thesis, I am discussing the relevance and application of computer simulation to model runaway electrons. The code that is used is called ARENA (Avalanche of Runaway Electrons Numerical Analysis), which utilises a Monte Carlo approach to solve the three-dimensional bounce averaged Fokker Planck equation. I also compare with the LUKE finite difference solver for primary runaway generation, as well as numerical and theoretical data [1, 2].

The majority of the thesis deals with performance and structural updates to the decade-old ARENA code which has resulted in a new ARENA 90-code which is written in the Fortran 90 language and is under active development by EFDA-ITM (European Fusion Development Agreement - Integrated Tokamak Modelling) task force. I have also made a proof of concept of a parallelised collision operator running on a GPU (Graphics Processing Unit) using the OpenCL API (Application Programming Interface) standard, which demonstrates the flexibility of the new ARENA code.

The thesis is primarily centred around two benchmarks, the preservation of a Maxwellian distribution with no outer electric field, and primary runaway generation under a constant electric field. In addition, there is an in-depth discussion of the simulation parameter space and design solutions of the ARENA code. Secondary generation from a seed of runaway electrons is discussed, but not implemented in the new version of the code.

The final results show a good match of published data for all test cases considered. The speed of simulations is greatly increased compared to the old ARENA code and the portability and usability of the new code should help contribute to future works. Together, this offers valuable insight on the possible applications and limitations of runaway simulation and the ARENA code in particular.

Acknowledgements

I would like to acknowledge Tünde Fülöp and Gergely Papp who supervised me. My collaborator, Gergely Csépány and also my dear friend Emelie Nilsson who got me involved in this project in the first place. They have all been keen to offer support and motivation for me throughout this work and I feel fortunate to have been able to rely on them. Albert with whom I shared an office and Istvan down the hall have always been available to help out with physics and computers at any and all time. A special thanks goes out to Joel Goop, who has spent a considerable amount of work designing the LATEX-template used for the thesis. Finally I wish to send thanks to my family and greater circle of friends who help me to stay focused and happy every day. I am very pleased with the outcome of this work and the road leading up to it.

Jakob Rydén, Göteborg June 2nd 2012.

Contents

1 Introduction **1**
1.1 Fusion on Earth 2
1.2 Ignition and confinement 5
1.3 The tokamak 6
1.4 Disruptions and runaways 8
1.5 ARENA 10
1.6 Roadmap 11

2 The Physics Model **12**
2.1 Coulomb collisions 14
2.2 Runaway electrons 15
2.2.1 Primary generation 16
2.2.2 Secondary generation 18
2.3 Numerical methods 21

3 Numerical Methods **22**
3.1 The Monte Carlo method 22
3.1.1 Metropolis, Markov chains and selection 24
3.2 Monte Carlo in ARENA 25

4 ARENA Code **26**
4.1 ARENA versions 27
4.2 Code overview 27
4.3 Monte Carlo operators 30
4.4 Collision operator 32
4.5 Initial distribution 34
4.6 Benchmarks 35
4.7 Inputs 36
4.7.1 Radial profiles 36
4.7.2 Time step 37

4.7.3 Weights 37
4.8 Outputs 38
4.9 Development and documentation 39
4.10 Parallel version 41

5 Results and Comparison **43**
5.1 Benchmarks 43
5.2 The LUKE code 44
5.3 Primary generation output 48
5.4 The new ARENA 90 50
5.5 Parameter scans 52
5.6 Discussion 53

6 Conclusions **56**

Bibliography **59**

A Definitions and Explanations **60**
A.1 Toroidal coordinates 60

B Physics in Depth **64**
B.1 The Langevin equation 64
B.2 ARENA's Monte Carlo operators 65
B.3 Full range collision operator 66

C Computer Code **70**
C.1 Code and compilers 70
C.2 OpenCL code 71
C.3 More ARENA details 73
C.3.1 Cutoff momentum p 73
C.4 ARENA inputs 74
C.5 Plotting ARENA output 74
C.6 ARENA's evolution 82
C.7 Code example 83

1

Introduction

Power generation for industrial applications and research is something that our modern society have come to rely on for our very basic needs. The discovery of nuclear processes and the invention of nuclear power reactors have made possible a surplus of electricity and heat production without the need to burn fossil fuels. While there is a significant benefit regarding power density, it is also a complex procedure with a finite risk for nuclear contamination.

Power from fossil fuel is a result of *chemical* reactions, whereas nuclear power taps the *strong nuclear force.* This allows the energy released (1 mole $\rightarrow 1.69 \cdot 10^{12}$ Joules) to become on the order of 10^6 times the energy released from propane combustion (1 mole $\rightarrow 2.043 \cdot 10^6$ Joules) [3]. Nuclear energy production comes with its own challenges, however. Today's fission power production facilities produce nuclear waste which must be handled and disposed of safely. In the prospect of replacing very large energy production from fossil fuels, governments must also consider the connection to nuclear weapon manufacturing. Above all fission power needs *active safety* solutions in the event of a critical shut down. An active fission core needs cooling from pumping water, or it will overheat, leading in the worst case scenario to a meltdown, which has been seen throughout history. A fusion reactor allows for nuclear power production with *passive safety.* If the reactor was to break down, the fusion plasma will quickly cool and the nuclear processes would halt within seconds.

This is guaranteed by the low amount of fuel in the reactor at a single point in time. While fuel rods in a fission reactor can contain years worth of fuel, the fusion material is continuously inserted in small quantities. The total amount of fuel in the reactor cavity, which is the size of a large hall, is about the weight of a postal stamp. In the event of an accident, the total energy that can potentially be released is therefore relatively small.

In 2020, if all goes according schedule, a brand new fusion experiment, ITER will be constructed in Cadarache, just north of Marseille on the French Côte d'Azur. It will be twice the linear size, and many times the volume of the current largest fusion reactors and is based on the currently most efficient family of reactor designs, the tokamak. Since the power gain factor

$$Q = \frac{P_{out}}{P_{in}}, \tag{1.1}$$

where P is power, scales with size there is great hope that ITER will produce a net power surplus and a self-sustained *burning* plasma. ITER is not a conceptual power plant, but a necessary research project for establishing solutions to some of the major hindrances that have been discovered throughout fusion research, for example runaway electrons.

1.1 Fusion on Earth

In stars, fusion is the main source of power. The reaction counteracts the gravitational pull of the stars' huge mass and keeps it from collapsing onto itself. The excess energy is radiated out in space, mainly as electromagnetic radiation. It is this energy from our Sun that has enabled life on Earth. In the case of stars, fusion plasma confinement is intrinsic from the gravitational force. For fusion power production on Earth, confining the fusion reaction is a complex challenge that extends from the physical properties of the elements at high temperatures. The most promising idea for a fusion power plant is confinement in a magnetic field. Since the particles that make up the plasma are electrically charged, they will stay bound to magnetic field lines, even at high temperatures due to the Lorentz force.

Physical measurements of the mass of ions show them to be slightly lighter than the individual nuclei that make them up. This is known as the *mass defect* and this fact is used to theoretically explain the prospect of fusion power. The binding energy released when elementary particles fuse to helium is the mass defect times the speed of light squared: $E = \Delta mc^2 = 5.0441 \cdot 10^{-29} \cdot 9 \cdot 10^{16} = 4.54 \cdot 10^{-12}$ J which for a mole is several orders of magnitude higher than for a chemical fossil fuel reaction. In a fission reaction, it works the other way around; excess energy is released when heavy atoms are split by bombarding them with neutrons. In principal, there is a possibility to tap nuclear power by fusing any element lighter than- or splitting any element heavier than iron (Fe), which has a binding energy per nucleon-maximum that can be seen in figure 1.1.

Fission power reactions can be achieved at room temperature. The fact that an incoming (speeding) neutron is not affected by the electrical Coulomb force allows it to penetrate close to the nucleus. Fusion on the other hand, requires two charged particles to get close for long enough time for the reaction to happen which requires breaching the Coulomb barrier. The probability for a fusion reaction to happen between two elements in a certain time is called the reaction cross section.

The difficulty with fusion on Earth, in a lot of ways, stem from the fact that very high thermal energies are needed. The reaction cross section is a function of temperature - a measure of a particle's kinetic energy, which implies momentum. The equivalent cross section for fission is a very large number in comparison. $\sigma_U = 600$ barns for a thermal

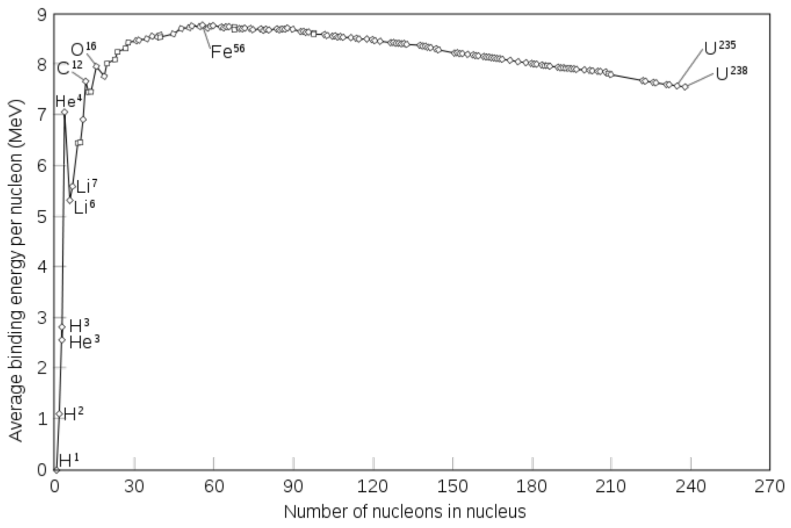


Figure 1.1: The binding energy of some elements in the periodic table. When new elements are formed through fusion (or fission) the difference in binding energy is released as kinetic energy of the particles resulting from the reaction. The large difference in binding energy between ^2H ^3H and ^4He (4 and 6 MeV respectively) makes for a huge potential in excess energy in the fusion reaction. (1.2). Adapted from [4].

neutron $k_n \approx 0.025$ eV colliding with $_{92}U^{235}$ whereas the maximum of the fusion cross section of a Deuterium-Tritium reaction is $\sigma_{DT} = 5$ barns at $k_D \approx 120$ keV. A picture of reaction cross-sections for different fusion reactions can be seen in figure 1.2. Note that the cross section tends to *peak* at some temperature which is the optimum. Deuterium-Tritium reactions is the popular candidate for fusion fuels since the cross section peaks at a high number for relatively low temperature.

In a classical mechanics sense, the requirement for fusion becomes a high enough (thermal) particle velocity of the fuels to overcome the Coulomb force. This corresponds to very high temperatures, around 150 million degrees; much hotter than e.g. in the core of the sun. For fusion researchers, it is usually convenient to express quantities like *temperature* and *electron rest mass* in terms of energy measures in electronvolts (eV). An electronvolt $1\text{eV} = 1.602 \cdot 10^{-19}$ Joules, is the amount of energy gained when a particle of charge e (the elementary charge) moves across the potential of 1 V. For thermal energy it is proportional to Boltzmann's constant $k_B = 1.38 \cdot 10^{-23}$ J/K, so $E_{\texttt{thermal}}[eV] = 3/2 \cdot k_b T \sim T \cdot 10^{-4}$ electronvolts. For $T = 150 \cdot 10^6$K, $E_{\texttt{thermal}} \approx 1.38/1.602 \cdot 150 \cdot 10^{6-23+19} = 12.92$ keV, which is close to the optimum temperature for a D-T plasma. Temperatures in fusion physics are usually expressed in keV units with Boltzmann's constant baked into the temperature $k_B T/(1.602 \cdot 10^{-19}) \to T$ eV.

In the Sun, there is a three-step fusion process taking place where protons form deuterium and ^3He and finally ^4He. The conditions for the reaction cross-section is different than on Earth however. In a tokamak (see below), the pressure is much lower and the reaction frequency cannot be arbitrarily low for a power plant because of

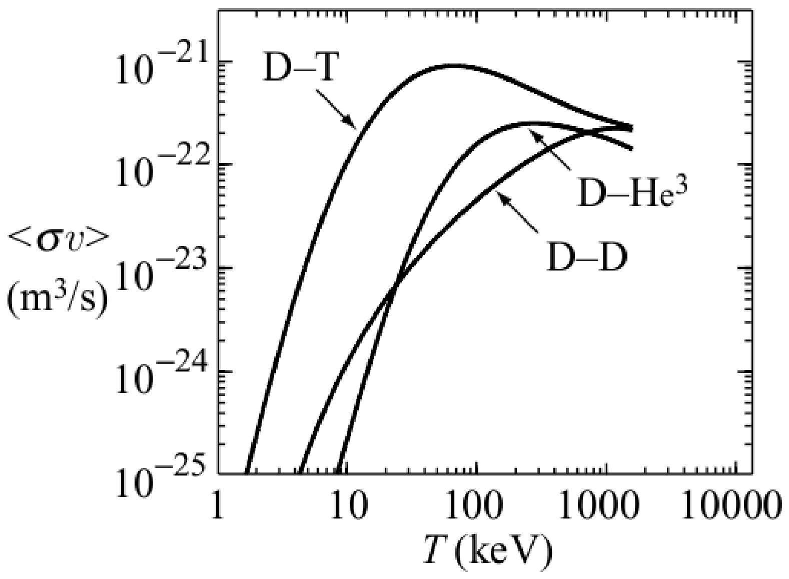


Figure 1.2: The velocity averaged cross section of fusion reactions as a function of temperature. This was obtained numerically from equal temperature Maxwell distributions. The D-T reaction has a maximum reaction cross section at $\sim$ 70 keV (Note the logarithmic scale). Adapted from [3]

effect, the energy output per second, reasons. Currently the main candidate for fuel is deuterium ^2H and tritium ^3H in a so called D-T plasma.

$$^2H + \ ^3H \rightarrow \ ^4He + \mathtt{n}(17.6\text{MeV}). \tag{1.2}$$

These elements resonate in a quantum mechanical sense at temperatures which are attainable in todays tokamaks, which gives a comparably high cross-section.

Deuterium can be distilled from sea water and tritium, a radioactive isotope of hydrogen with a half-life of 12.3 years, can be produced from lithium in a sub-process inside the fusion reactor.

In a reactor, neutrons will be responsible for net heat output, while the other resulting ^4He alpha particles will help sustain the plasma temperature. Since neutrons lack a charge, they will immediately escape the magnetic confinement and hit the reactor wall where their energy must be absorbed. For the D-T reactors, the neutrons will also be used to *breed* radioactive tritium fuel which is unavailable in nature. In the design for ITER, a "blanket" in the outer reactor wall will test an on-site production of tritium which is a practical requirement for the first generation of fusion power plants. The process is not uncomplicated, a single neutron for each fusion reaction will likely not be sufficient so the neutrons must be multiplied in number. The heat load on the wall will be considerable, and one must consider activation of the wall compounds which puts constraints on the materials used.

1.2 Ignition and confinement

The phrase ignition refers to a fusion plasma that is *burning*, self-sustained without the need for outer heating. This is achieved when alpha particles from the fusion reaction give enough energy back to the plasma. Mathematically this puts constraints on the fusion *triple-product* $nT\tau_E$. $n = p/(2T)$ is plasma (number) density, T is temperature and τ_E confinement time.

The plasma density is the number of fuel ions per cubic meter. The energy confinement time is a measure of power loss from the plasma and is defined as: $\tau_E \equiv E/P$ where E is the total plasma energy and P is the rate of energy loss to the environment (the wall). It is analogous to the time constant of e.g. a house cooling down. Since the desire is to keep the plasma hot, τ_E becomes a measure of how good confinement is. Eventually the heat energy from alpha particles should provide the energy required for sustaining the temperature. This happens when the reaction rate is high enough. The phenomena is called *ignition*, likened to how a barbecue can be ignited by an electrical heating element, but will eventually sustain its own burn. The condition for ignition is obtained by setting the alpha particle heating equal to the rate of loss of energy. The energy loss (with Boltzmann's constant baked into the temperature) is:

$$S_L = \frac{3nT}{\tau_E}. \tag{1.3}$$

The energy gained from alpha particles in the fusion reaction is:

$$S_f = S_\alpha = E_\alpha p^2 \langle \sigma v \rangle / (16T^2), \tag{1.4}$$

with the number density $n = p/2T$, and p is the plasma pressure. The radiation loss from Bremsstrahlung, the energy lost in particles reflecting off each other and radiating energy according to energy conservation, is:

$$S_B = \frac{1}{4} C_B Z_{eff} \frac{p^2}{T^{3/2}}, \tag{1.5}$$

where C_B is a radiation constant and Z_{eff} is the effective charge of the plasma. A very pure plasma has $Z_{eff} \simeq 1$ and the balance equation becomes:

$$S_\alpha + S_h = S_L + S_B, \tag{1.6}$$

where S_h is the heating power from ohmic heating and auxiliary heating, from e.g. microwave radiation, together. Ideal ignition assumes no losses and has $S_\alpha = S_B$ while full ignition is $S_\alpha = S_B + S_L$. This leads to an approximate expression for the requirement on the triple-product:

$$\hat{n}\hat{T}\tau_E > 5 \cdot 10^{21} \text{m}^{-3}\text{keVs} \tag{1.7}$$

where hat-variables denotes the peak values of density $[m^{-1}]$, and temperature $[keV]$ respectively. The parabolic relationship between temperature and confinement time makes optimising the product of these the objective rather than increasing either individually. Indeed, the discovery that increased temperature could worsen confinement time was a big blow to fusion research in the 70's.

1.3 The tokamak

The tokamak is a Russian reactor design that showed surprisingly good performance when measures of confinement time and temperature were first announced. The name is an abbreviation of Russian for: toroidal chamber magnetic coil *magitnaya katushka toroidalnaya kamera* $\rightarrow$ `toka·ma·k`. The main objective of plasma confinement is to find a configuration where the plasma is as stable as possible, i.e the confinement time is very large. Many subjects of plasma physics are still being researched, and issues with turbulence driven particle- and energy radial transport have continued to challenge the research community. Although it is difficult to get detailed measurements on the inner workings of a hot plasma, there are a multitude of diagnostics tools that have been developed and experimental physics has continually been making discoveries that puts new boundaries on theory.

A tokamak is rotation symmetric around the toroidal axis. A set of large ring-electromagnets that are thread around the plasma induce the magnetic field $\mathbf{B}$ along the toroidal axis (details in section A on torus geometry). A smaller magnetic field in the *poloidal* direction is induced by a large current flowing through the plasma. This gives a twist to the magnetic confinement field that nullifies radial drift effects that drives the particles out of the plasma, see figure 1.3.

The large current must be produced in the plasma somehow, and the established solution is to induce it with a large transformer. Since the induced current, according to Lenz's law $F_E = -N\Delta\Phi_B/\Delta t$, ($F_E$ is electromagnetic force) is inherently pulsed because of the Δt dependence, a machine like this can only run in short time intervals. This is an obstacle; a power plant would require continuous electricity output and materials tend to wear faster when exposed to constantly changing temperatures. Solutions have been proposed - for instance the bootstrap current, that stems from the fact that there is a radial pressure gradient in the torus geometry, could possibly sustain large parts (70%) of the toroidal current, with the rest being accounted for by neutral beam heating. Pulsed operation could arguably also be manageable if the pulses were long (hours) and restart time relatively short (minutes), but this inherent limitation in tokamak design creates a practical need for research into alternative designs. The tokamak's performance benefits from the large current, which automatically heats the plasma through Ohmic heating - similar to how the filament of a lightbulb is made to glow. The low plasma density puts a limit on the temperature that can be gained from Ohmic heating at a certain temperature however. Other *auxiliary* heating methods like radio frequency heating and neutral beam injection are then applied to increase the temperature further.

The tokamak has been the most successful type of magnetic confinement device to date. The second most established device, the *stellarator* is a *steady state* design with a complex magnetic coil structure that does not require a driving current for the poloidal field component. One major drawback to the stellarator is the complex design of the magnetic confinement coils which complicates engineering as well as analysis. A tokamak can often be regarded as a two-dimensional problems of the cross section, assuming axisymmetry, where stellarators require three-dimensional computer simulations.

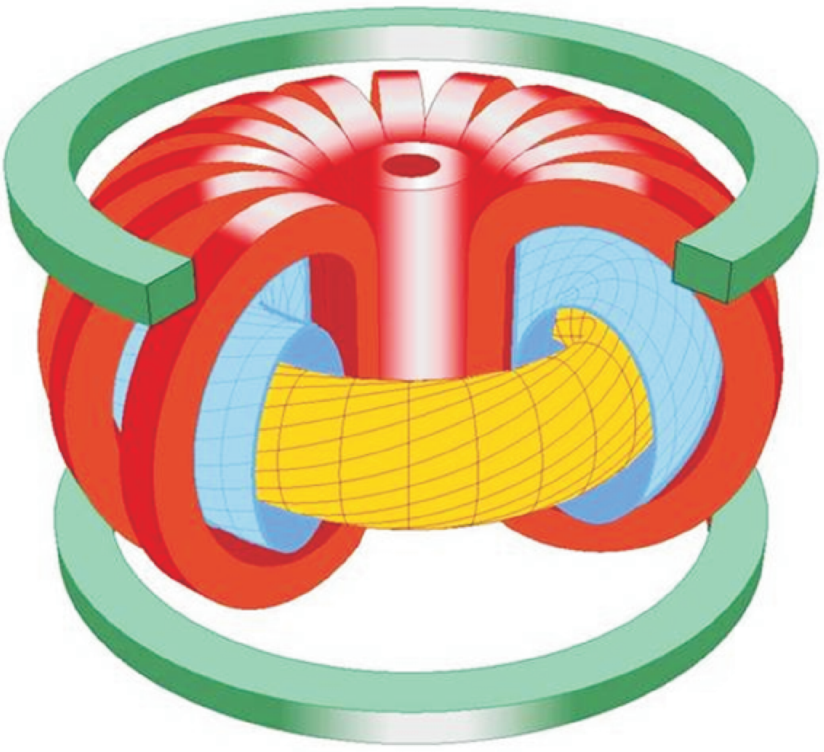


Figure 1.3: A schematic picture of a tokamak. The large D-shaped magnetic coils are responsible for the large toroidal magnetic field $\mathbf{B}_{tor}$, while the induced current (transformer not visible) gives the poloidal *twist.* The large circular coils are schematic representations of vertically stabilising coils. Adapted from [3].

There are many fusion research plants in the world today. Most of the active ones in Asia, primarily Korea and Japan. Currently the world's largest machine is the Joint European Torus (JET), situated in Oxford in England. It was rebuilt with a *divertor* configuration for improved confinement and has been used for some testing in anticipation of ITER.

To reach break-even, the confinement time and temperature have to be just right for the fuels in question. Figure 1.4 shows the break-even region along with the current best shots in JET. Past attempts to increase temperature has been seen to lower confinement time, and it is currently believed that the reactor must be scaled up to reach ignition. A large enough physical size of the reactor is an intrinsic requirement for fusion since the outer area of the plasma is the only way for heat to escape. With a bigger cross-section of the plasma, the temperature can be kept high at the core, while the periphery has a larger area to dissipate heat Another benefit is that the electromagnet dimensions are larger and more easily shielded from the hot plasma. This is important since superconducting magnets is likely a requirement to allow for high energy gain. With present

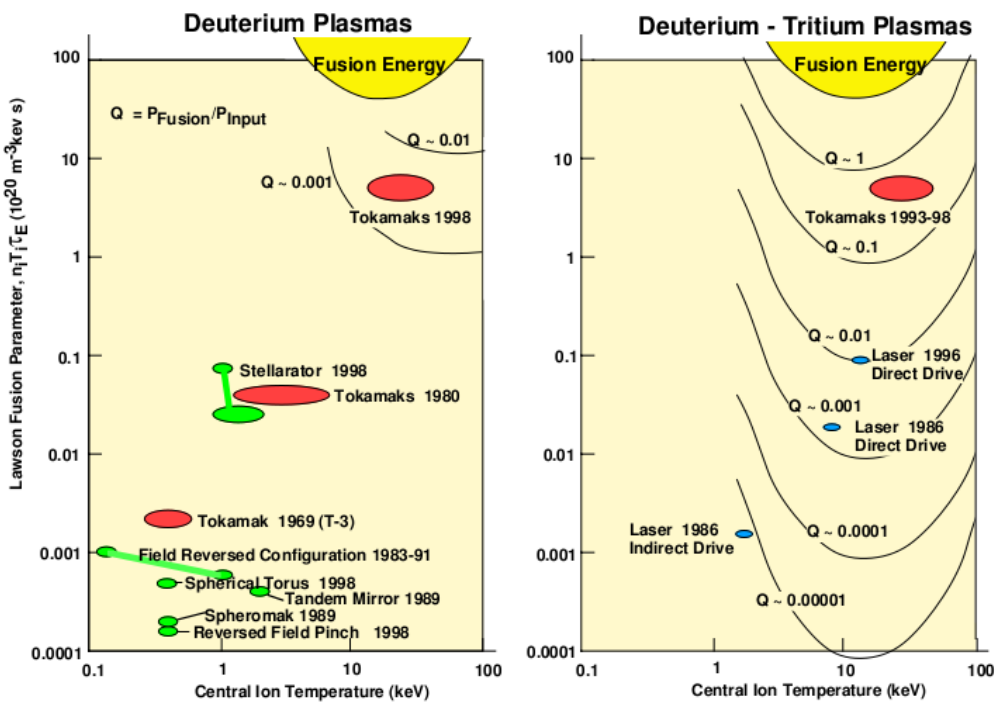


Figure 1.4: Schematic picture of the fusion triple product as a function of plasma temperature on the horizontal axis. An imagined third axis shows the $Q-$factor which needs to be ≥ 1 for ignition. There is also a set of historical graphs showing how the $Q-$factor has been steadily improving with more modern fusion devices.

day technology this implies that they must be kept in a cryostat close to absolute zero temperature. There are also complications with a larger design, however. Besides a higher construction cost, the required neutral beam injector for heating, and other similar equipment, simply has not yet been created. Furthermore, disruptions, which hurt the stability of the plasma, have been seen to scale with reactor size as well. This makes it difficult to build a sturdy enough device to withstand the physical tear from disruptive breakdowns. In an experiment reactor, these critical events are manageable as part of experiments. In a power plant for commercial use these issues must be dealt with.

1.4 Disruptions and runaways

A *disruption* is a disturbance of plasma stability and confinement due to perturbations. Disruptions appear because of the chaotic nature of ionised gas at very high temperatures. Modern fusion experiments have computer controlled automatic control systems that try to stabilise the plasma, using e.g. radio frequency heating to *even out* temperature- and pressure profiles, in an attempt to nullify disruptions. But since the complete physics of the plasma is still not known it is impossible to eliminate them all.

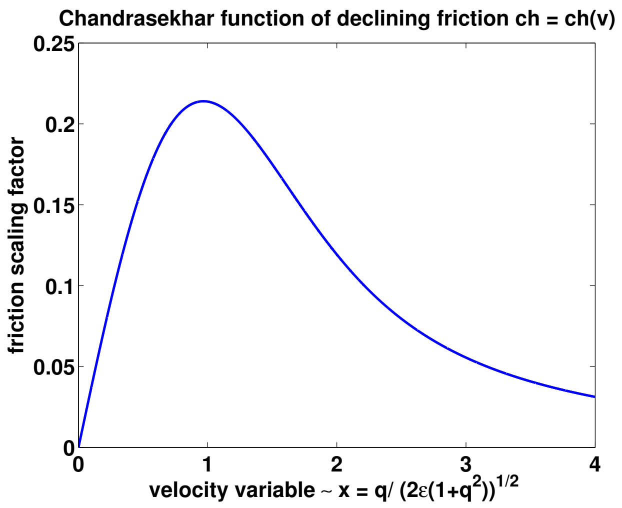


Figure 1.5: The Chandrasekhar function showing the decrease of the friction force in a plasma after the friction peak at the thermal energy $E_{th} = mv_{th}^2 = k_B \cdot T$. Here the input is $x = v/v_{th}$ so the peak is at $x = 1$. Note that the actual friction force in a fusion plasma will not tend to 0 as $x \to \infty$ like here because of relativistic effects such as synchrotron radiation.

This creates a need to build a mechanically robust reactor device which can withstand disruptions and continue sustained operation.

Runaway electrons appear in tokamaks during a so called *thermal quench*, when the plasma is rapidly cooled locally due to a heat flow to the first wall. Because of the large toroidal current, if the plasma is cooled rapidly, the electrical conductivity $\rho \sim 1/T^{3/2}$ changes locally, and the electrons cannot immediately equalise the current. Instead they experience an *electric field*, E because of the difference in electric potential. This electric field acts as a force on electrons (and also ions, but they require a much higher momentum to accelerate). Since electrons experience "friction" only from *Coulomb collisions* they can be accelerated indefinitely. This is very similar to how stars experience friction in galaxies, and the same non-monotonic *Chandrasekhar* function;

$$G(x) \equiv \frac{\phi(x) - x\phi'(x)}{2x^2} \to \begin{cases} \frac{2x}{3\sqrt{\pi}}, x \to 0 \\ \frac{1}{2x^2}, x \to \infty \end{cases} \tag{1.8}$$

can be used to describe the relationship (see figure 1.5). $\phi(x)$ is the error function

$$\phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} dy. \tag{1.9}$$

If the accelerating Lorentz force, $\mathbf{F}_a = e\mathbf{E}$ is larger than the friction force for long enough, electrons will accelerate indefinitely, building up momentum $p \to \infty$. While there is a speed limit for electrons at the speed of light $c = 3 \cdot 10^8$ m/s, momentum $p = mv_e$ is unlimited, since the relativistic mass increases as γm_0, where m_0 is the electron's rest mass and $\gamma = 1/\sqrt{1 - v^2/c^2}$ the relativistic scaling. $\gamma \to \infty$ as $v \to c$). At this point, electrons will lose energy to synchrotron radiation, and also actual *close* collisions with the bulk (non-relativistic) plasma, effectively limiting their energy.

Runaway electrons carry an extremely high energy and build up a hazardous runaway current which usually shoots into the first wall, damaging the reactor. This is known as a *major disruption* which must be avoided. Keeping the plasma stable to avoid the creation of the **E**-field is desirable, but still today a few percent of all shots in every operating tokamak today ends in a major disruption, and problems are believed to increase with larger machines like ITER. There are some ideas of how to mitigate the issue. Sending in a *killer pellet*, a frozen piece of deuterium which is ejected from a powerful centrifuge and vaporises inside the plasma, can create a target for the runaways to hit. Another idea is to puff in gas in attempt to create a local (radially) higher density that could possibly hinder the path of the runaways. Regardless of the method chosen, there is large incentives to investigate the process further.

1.5 ARENA

ARENA is the name of the Monte Carlo-code that is used to solve the distribution function time evolution differential equation (see section 2.2). It is an implementation of the Monte Carlo operator formulation by Eriksson and Helander [2], [5] where the full Fokker-Planck equation is reduced to three dimensions in phase space by averaging over the toroidal symmetry. The code was originally written by Lars-Göran Eriksson in 1998 and then revised throughout 1999-2002 while he was in Cadarache in France.

Development of ARENA has then been mostly dormant until 2010 when the code base was ported from Fortran 77 to Fortran 90 by Gergely Csépány and incorporated as a part of the *Integrated Tokamak Modelling* (ITM) project at *European Fusion Development Agreement* EFDA. EFDA is the umbrella organisation of all fusion research laboratories in Europe [6], and the ITM task force is a subgroup that strives to create a full simulation environment for ITER, though in principal it could be used to model any fusion device [7]. The idea of ITM is that the full suite of software should be independent of a programming language, but rely on a set of common *rules* for how to format the data exchanged between the individual models, so called CPO:s (also see section 4.9).

Within ITM, ARENA is overseen by Gergely Pokol and labeled *in development*. Some thorough changes were made to improve performance, including linking in pre-compiled linear algebra routines as well as adding support for swapping random number generators.

To get good reference data with similar input parameters as the ones I have used for ARENA, I have visited Cadarache in France to make comparison runs with the finite difference code LUKE. Using the output of LUKE has been important in deciding on a good baseline for tests which are described in section 5.2. LUKE is written in `MATLAB`, using `.mex` modules for some time critical calculations. It is a mature code which can model electrons as well as fuel- and impurity ions by setting the charge and weight parameters up accordingly. The code is maintained and developed by Joan Decker and Yves Peysson. It has a sustained number of current users and supporting staff, and has been used in publications [8].

1.6 Roadmap

This was an introduction on fusion, plasmas and simulation in general. In section 2, I will describe the physics of runaway electrons and briefly talk about the various ways to model electrons in a fusion plasma. Section 3 introduces Monte Carlo techniques which are used in ARENA. In section 4 I try to bridge the gap between the model and the actual computer program. There is also some practical information on the software and a description of the structure and features of the software package. In section 5 I go through and comment on the program output and some of the discoveries that were made during the course of this project. I also give examples on experiments that can be of interest with a code like ARENA.

Finally, I put my thoughts and conclusions in section 6, which is followed by appended code and some additional explanatory sections. Appendix A contains common expressions and definitions for reference. Appendix B contains some derivation of e.g. Langevin equations and C includes comments on and examples of `ARENA` and the companion `MATLAB` code.

2

The Physics Model

There are some standard approaches to model a fusion plasma that have been researched over the years. They differ a bit in character and have advantages and disadvantages when it comes to describing the actual physical reality which is diagnosed in fusion experiments. The naive approach, which is not possible in practice, or usually not even beneficial to model is the full plasma model, where each particle, ions and electrons are modelled with position and velocity. Even without modelling the slow ions, there are a lot of electrons in a fusion plasma. With number density $2 \cdot 10^{19}$ m^{-3} and plasma volume 100 m^3, relevant parameters for JET, the Joint European Torus, $2 \cdot 10^{21}$ particles would have to be modelled with 6 variables, three spatial and three velocity measures - a staggering number. Good prediction of the dynamics can still be achieved with a simplified model and stochastic updating which is in principle the Monte Carlo approach used in ARENA.

In general, there are two main types of approaches to plasma modelling: fluid models and kinetic models.

The *fluid model* likens the plasma to a fluid using Navier-Stokes equations with an addition of Maxwell's equations to account for the electromagnetic properties of ionised particles. The simplest of these is called *magnetohydrodynamic* model, where the plasma is treated as a single fluid. It can also be useful to treat the electron- and the ion-population separately, making for a slightly more complex model. The fluid model is usually accurate when there is enough collisions to keep the plasma velocity distribution close to a Maxwell-Boltzmann distribution.

The other approach is a *kinetic model* that centres around a *distribution function* $f = f(\mathbf{x},\mathbf{v},t)$, which counts the number of particles with a certain speed (v_x,v_y,v_z) and position (x,y,z) per unit volume in phase space[1] at a time t.

An equation which takes into account long range Coulomb interactions (see section

[1]when velocities are treated alongside spatial variables one speaks of *phase space* rather then regular three-dimensional space, the wording can also imply time

2.1) can be found by solving the Vlasov equation

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f + \frac{e}{m_e}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f}{\partial \mathbf{v}} = 0, \tag{2.1}$$

which is a modification of the Boltzmann equation with the addition of Coulomb interactions [5]. Because of the low density in a fusion plasma, Coulomb collisions is the main way that the particles sense each others' presence. A *Fokker-Planck* equation describes the time evolution of a stochastic variable by its probability density function. It can be applied to many stochastic differential equations and is used in financial models [9] and for many other applications. By treating the high frequency fluctuations of the electric and magnetic fields as perturbations, the **E** and **B** fields of the Vlasov equation become large scale fields which are macroscopic plasma parameters. The effects of small scale fluctuations are accounted for by the so called *collision operator*, which is a *Fokker-Planck operator*. So for the purpose of plasma modelling, the Vlasov equation becomes a Fokker-Planck equation with the effects of small scale fluctuations gathered in a collision term on the right hand side. Variations of this equation are wide spread and well established for many applications [9]:

$$\frac{df}{dt} = C(f) = \sum_a C_{ab}(f_a, f_b). \tag{2.2}$$

The collision operator $C(f)$ is a sum of collisions between particle species a and b, including $b = a$. In ARENA, the Fokker-Planck equation is simplified to one spatial dimension before being implemented into the model by integrating along the toroidal shape of the tokamak and using symmetry, so called *bounce-averaging* [10].

The time evolution of the Fokker-Planck equation can be found in some different ways, including analytically solving the steady state equation with no time dependence (see section 2.2), or with finite element numerical methods. The fact that the runaway electrons continuously accelerate throughout the simulation makes determining a finite-element grid size and resolution difficult, however. The mean (kinetic) energy of runaway electrons when secondary generation has been sustained has been shown to be around 10-20 MeV [11], but there is no guarantee that this will be sufficient for a full physical simulation. Furthermore, a strong radial variation of the characteristic radial energy of the runaways complicate the calculation, the solution to which would require unphysical modifications of the problem, i.e. artificially increasing the braking effect from Bremsstrahlung [2].

These problems are not present with a Monte Carlo-approach, in which the Fokker-Planck equation 2.17 is replaced by a discretised Langevin equation that can be formulated as a number of Monte Carlo operators which are evaluated in each time iteration. Each test particle represents a number of electrons from the continuous distribution and is updated independently. The distribution function, f can then be reconstructed at any time from the particle ensemble.

The main disadvantage of this technique is its slow convergence ($\sim \sqrt{N}$), with the number of test particles, N. This is handled in ARENA to some extent through a

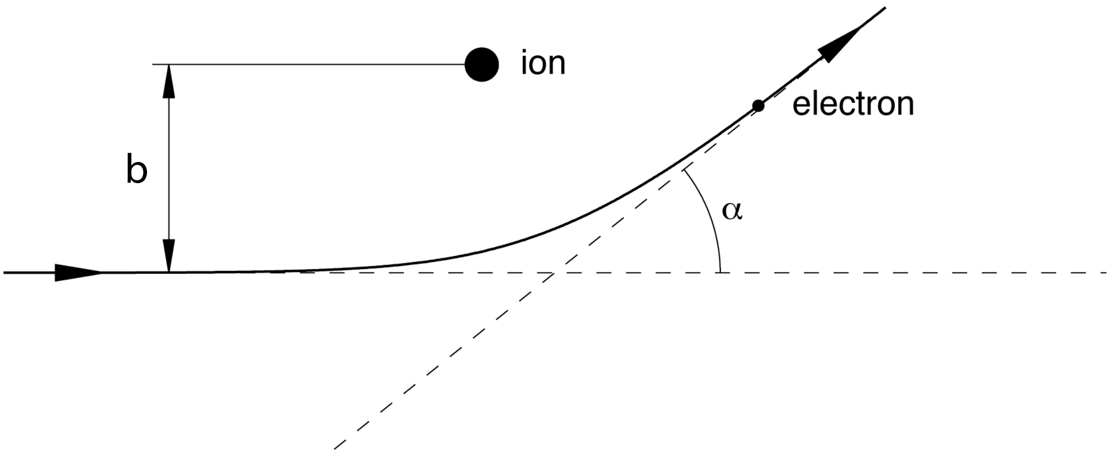


Figure 2.1: An electron scattering off a much heavier ion. The deflection angle α depends on the impact parameter b

weighting scheme giving higher statistical weight to high energy particles, while maintaining physical conservative quantities. The other, perhaps more modern, approach to improving simulation time is to write code that runs calculations as parallel processes, updating the test particles of the population simultaneously. This can be done on e.g. a cluster of CPUs or a GPU, *Graphics Processing Unit.* The structure of a Monte Carlo program, since every test particle is independent, actually lends itself well to this.

2.1 Coulomb collisions

Electrons in a plasma are not significantly deflected under the presence of an ion, or another electron, but extremely rarely. The probability for a head on collision is very small. Instead, the cumulative effect of several small angle deflections give rise to a geometrically large effect in the time spans of interest. Because of this, one speaks of a Coulomb *collision* meaning a change in particle velocity (the direction) after a certain amount of time τ during which the particle has experienced the cumulative effect of many Coulomb interactions with other particles.

Let us suppose that an electron with charge $-e$ and mass m_e passes a stationary ion at distance b with charge $+Ze$ and substantially larger mass. The perpendicular *Coulomb* force on electrically charged particles is given by:

$$F_{C\perp} = \frac{1}{4\pi\epsilon_0}\frac{Ze^2}{b^2}. \tag{2.3}$$

The duration that the force acts on the electron is b/v and so the change in perpendicular velocity is approximately given by:

$$\Delta m_e v_\perp \approx \frac{Ze^2}{4\pi\epsilon_0}\frac{1}{vb}. \tag{2.4}$$

The deflection angle is $\alpha \sim 1/v^2$ (figure 2.1). For a *true collision* event α should be close to 90° [3]. Then the collision cross section σ can be estimated as:

$$\sigma \approx \pi b^2 = \frac{\pi e^4}{(4\pi\epsilon_0 m_e v^2)^2}, \tag{2.5}$$

with e.g. $Z = 1$, the particle is a proton. The collision frequency, for an ion-electron interaction, becomes:

$$\nu_{ei} = n\sigma v \approx \frac{n\pi e^4}{(4\pi\epsilon_0 m_e)^2 v^3} \tag{2.6}$$

The effect from multiple ion-interactions on an electron can be seen as a diffusion of the perpendicular momentum. The diffusion constant is found by integrating the individual changes in momentum squared, multiplied by the rate of collisions in an interval, $I = [b,b+db]$ which is $nv \cdot (2\pi b db)$:

$$D_{v\perp} = \int \left(\frac{Ze^2}{4\pi\epsilon_0}\right)^2 \frac{1}{v^2b^2} nv(2\pi b db) = \left(\frac{Ze^2}{4\pi\epsilon_0}\right)^2 \frac{2\pi n}{v} \int \frac{db}{b} \tag{2.7}$$

Now there is just an integral over $1/b$ which is multiplied by a factor that is constant in b. However, the integral diverges at both plus- and minus infinity, which does not reflect physical reality, so there must be a lower- and upper bound. A reasonable upper bound is chosen to be the initial momentum of the incoming particle. This is reasonable since the total momentum should not change. The lower bound should be comparable to the Debye length, $\lambda_D = \sqrt{\epsilon_0 T_e/(e^2 n_0)}$ (n_0 is the plasma density), outside the reach of which, no interaction is considered. Setting $\Delta m_e v_\perp = mv$ makes the lower bound:

$$b_{\texttt{lower}} = \frac{Ze^2}{4\pi\epsilon_0}\frac{1}{m_e v^2} \tag{2.8}$$

The primitive function of $1/b$ is a logarithm function which by convention its written as $\ln\Lambda$, and evaluated at the lower and upper bounds. The value depends on the kinetic energy of an incoming particle, v and the ion charge Z, which correspond to plasma temperature and effective plasma charge for the fusion plasma case. A common value for a tokamak is $\ln\Lambda \in [10,20]$ [12, pg. 4]. In ARENA, this is an input parameter: `CNLAM`.

2.2 Runaway electrons

If an electric field that is strong enough is applied to a plasma, some electrons will experience unlimited acceleration in the toroidal direction and *run away.* This is due to the fact that the breaking friction force, unlike in a normal fluid or air, does not increase monotonically with the velocity of the plasma particles. Rather there is a global maximum value of $F(v) = mv\nu_{ee}(v)$ [5] at the thermal velocity $v = v_{th}$.

The accelerating electric field is usually created due to a disruption. The origin of the disruption is usually that turbulence forces part of the plasma outside of the magnetic confinement region to the edge of the containment vessel where it essentially touches the

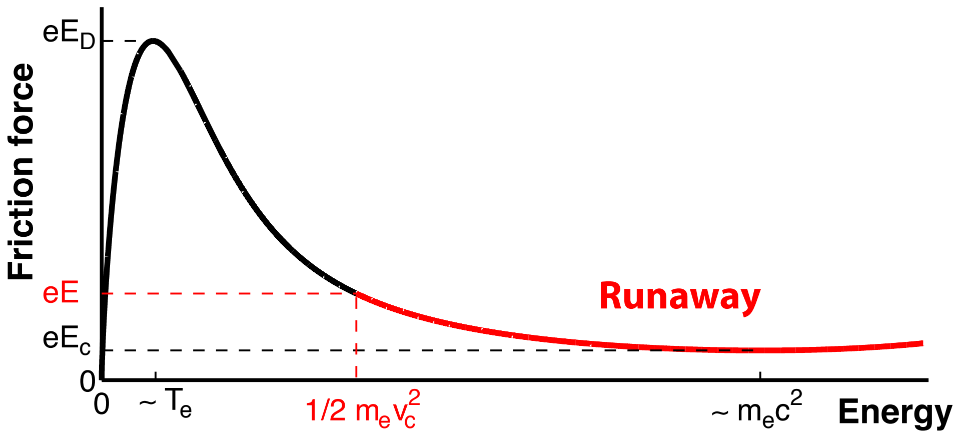


Figure 2.2: The friction force on individual electrons as a function of their kinetic energy. A faster moving electron will experience less friction, and if the acceleration due to an added electric field is high enough, part of the population will become runaway electrons. E_c is the critical field, below which, no runaways are created. E_D is the Dreicer field for which all electrons will run away. The maximum of the curve is at $v_{th} = \sqrt{2T_e/m_e}$. Image taken from [13].

inner wall and picks up impurities. This could be carbon, or something else that the wall is made of, which when it enters the plasma is excited and radiates heat when the excited electrons fall back through the electron shells. This changes the plasma resistivity, ρ, locally and an electric field, $\mathbf{E}$ is induced according to Ohms law;

$$\mathbf{E} = \rho \mathbf{j}. \tag{2.9}$$

This field accelerates the electrons. Ions are also accelerated but experience a higher friction force which scales with the radius of the elementary particle squared.

The threshold for when electrons experience unbounded acceleration can be derived analytically and is called the critical electric field limit, E_c. When the induced electric field from a temperature change is lower than E_c, electrons will experience enough friction to keep their thermal velocity.

At the Dreicer field limit $E_D (> E_c)$, all electrons will run away [5].

2.2.1 Primary generation

The friction force $F(v_e)$ depends linearly on the Coulomb collision frequency $\nu_{ee}(v_e)$. The velocity and the forces are vectors in space, but the magnetic cage keeps electrons well confined radially along the **B**-field. In the toroidal direction, however, friction is the only retarding force, so we will consider scalar equations, $v_{electrons} = v_{||}$ velocity parallel to the magnetic field.

The braking ”friction” force is given by a complex equation that goes approximately like the Chandrasekhar function (equation 1.8) with an added local minimum at the *critical momentum* p_c due to relativistic effects such as synchrotron radiation. Synchrotron radiation is broad band electromagnetic waves radiated by particles moving close to the speed of light in a magnetic field to preserve momentum. A qualitative picture is plotted in figure 2.2. For electrons moving faster than the friction curve maximum at $v = v_{th}$, the friction force $F_f(v_e) = m_e v_e \nu_e(v_e) \sim \nu_e(v)$ the collision frequency, decreases like [5]:

$$F_f(v) = m_e v \nu_{ee} \approx m_e v \frac{n_e e^4 \ln \Lambda}{4\pi\epsilon_0^2 m_e^2 v^3} \sim \frac{1}{v^2} \tag{2.10}$$

For the thermal velocity, baking in Boltzmann's constant in the temperature T and neglecting constants which change the equation by less than an order of magnitude, $m_e v_{th}^2 = T$, and so the Dreicer field under which all electrons in the population run away can be written:

$$E_D = \frac{n_e e^3 \ln \Lambda}{4\pi\epsilon_0^2 T_e} \tag{2.11}$$

This is usually called the Dreicer field, crediting Dreicer for the discovery of this primary generation mechanism. Here, n_e is the number density, the number of electrons per unit volume, e the elementary charge, $\ln \Lambda$ the Coulomb Logarithm, T_e the bulk electron temperature and ϵ_0 the electric permittivity of free space. This formulation also allows for a possible approximation of the critical field, by replacing $T_e \to m_e c^2$ the electron rest mass:

$$E_c = \frac{n_e e^3 \ln \Lambda}{4\pi\epsilon_0^2 m_e c^2} = \frac{T_e E_D}{m_e c^2}, \tag{2.12}$$

with E_D the Dreicer field, m_e the electron mass and c the speed of light. When an electron is accelerated by an outer electric field E, the equation of motion for that one electron becomes:

$$m_e \frac{dv_e}{dt} = eE - m_e \nu_e(v_e) v_e, \tag{2.13}$$

which can be rewritten using 2.10 to get an expression for the *critical velocity*:

$$m_e \frac{dv_e}{dt} = eE \left(1 - \frac{v_e n_e e^4 \ln \Lambda}{4\pi\epsilon_0^2 m_e (eE)} \frac{1}{v_e^3}\right) = eE \left(1 - \frac{v_c^2}{v_e}\right) \tag{2.14}$$

$$v_c^2 \equiv \frac{n_e e^3 \ln \Lambda}{4\pi\epsilon_0^2 m_e E} \tag{2.15}$$

a condition for when the net force will be strictly positive. In the limit where $v_c \to v_{th}$, mean speed of the bulk of the electrons, we have the Dreicer field.

At a lower field strength, only some of the electrons will meet the runaway condition and if the field is so small that that $eE < m_e v_c \nu_e(v_c)$, there will be no runaway generation at all.

In a normal tokamak plasma, the electric field is much lower than the Dreicer field, but during a disruption, it can become many times the critical field, which will slowly allow some electrons to become runaways. The *primary runaway generation rate* γ can be derived by making a quasi-steady state assumption, and deriving an analytical solution valid in some parameter regions. Or alternatively, by making a simulation, assuming that the bulk of the electrons are Maxwell-distributed with a small number of electrons energetic enough to create a runaway *tail* that extends further from the Maxwell equilibrium with time.

The mathematical problem becomes to solve the steady-state kinetic equation:

$$-\frac{e\mathbf{E}}{m_e}\frac{\partial f}{\partial \mathbf{v}} = C(f), \tag{2.16}$$

which is the Vlasov equation 2.1 with zero velocity and no time dependence. $C(f)$ is the Fokker-Planck collision operator for fast electrons colliding with a Maxwellian distribution of ions and electrons.

The Fokker-Planck equation becomes:

$$-\frac{eE_{\|}}{m_e}\left(\frac{\partial f}{\partial v} + \frac{1-\xi^2}{v}\frac{\partial f}{\partial \xi}\right) = \nu_{ee}v_{T_e}^3\left[\frac{1+Z}{2v^3}\frac{\partial}{\partial \xi}(1-\xi^2)\frac{\partial f}{\partial \xi} + \frac{1}{v^2}\frac{\partial}{\partial v}\left(f + \frac{T_e}{m_e v}\frac{\partial f}{\partial v}\right)\right] \tag{2.17}$$

where $\xi = v_{||}/v = \cos\theta$ is the cosine of the pitch angle and Z the effective ion charge[2].

Analytically calculating the flux of particles to infinity in velocity space is a non-trivial calculation that has been carried out with some trouble in a classic series of papers, arriving at the following growth rate, called the *Kruskal-Bernstein* rate [5]:

$$\gamma_D = \frac{dn_r}{dt} = kn_e\nu_{ee}\left(\frac{E}{E_D}\right)^{-3(1+Z_{eff})/16}\exp\left(-\frac{E_D}{4E} - \sqrt{\frac{E_D(1+Z_{eff})}{E}}\right) \tag{2.18}$$

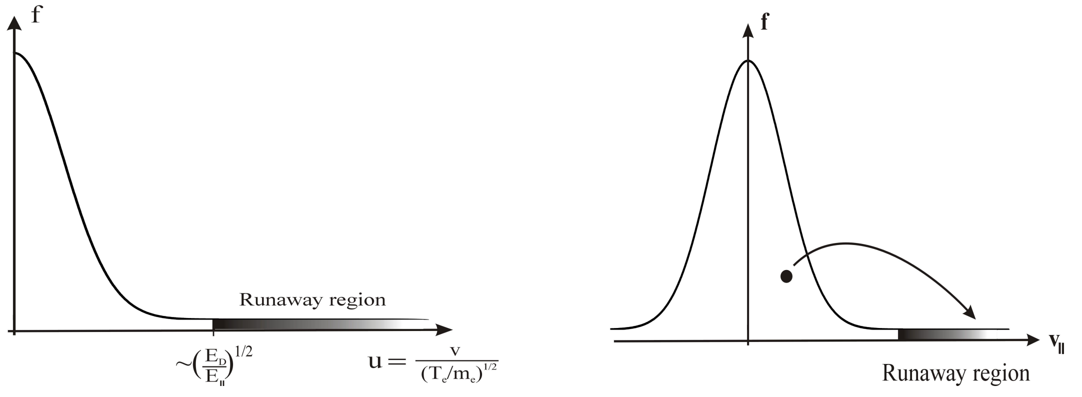
where k is a scaling factor of order unity and E is expressed as a fraction of the Dreicer field, $E_D = E_D(T)$, Z_{eff} is the effective charge of the plasma, a measure of the amount of impurities present. The Kruskal-Bernstein runaway rate has been confirmed numerically by Kulsrud [1] and in the rest of this thesis I will refer to this as the Kruskal-Bernstein- or Kulsrud rate interchangeably.

2.2.2 Secondary generation

Experiments by Sokolov in 1979 and later experiments and measurements in the 80's and 90's suggested that Dreicer generation cannot alone be responsible for the measured high runaway generation rate at small electric field strengths.

In conventional primary generation, electrons diffuse into the high energy-runaway tail, $v/v_{T_e} \gtrsim E$ at the rate $\gamma_D = n_r'(t)$. This is exponentially small in E, so primary generation is negligible unless $E \gtrsim 0.03$ [5]. The fundamental reason that particles have a *diffusive* motion is that collisions resulting in small changes to the particles velocities play a dominant role in the Fokker-Planck collision operator. In reality *close* collisions do occur but at very small rates and they are nearly always unimportant for plasmas

[2]also see appendix A for an explanation of the normalised parameters


f
Runaway region
$\sim\left(\frac{E_D}{E_\parallel}\right)^{1/2}$
$u = \frac{v}{(T_e/m_e)^{1/2}}$
f
$v_\parallel$
Runaway region

(a) Primary generation (b) Avalanche ← close collisions

Figure 2.3: A schematic of the two types of runaway electron generation. To the left, primary generation through a net accelerating electromagnetic force $e\mathbf{E}$ is show. To the right is secondary generation by close collision. A fast (runaway) electron *kicks* a thermal electron into the high energy spectrum.

with $\ln\Lambda \gg 1$. This changes when a large enough part of the population is made up of runaway electrons. The runaway electrons can kick a thermal electron above the runaway velocity threshold while still retaining enough energy themselves to not slow down, figure 2.3. This process leads to an exponential growth of the runaway population, provided that there is a *seed* population of fast runaways. The necessity to have a seed population leads to this mechanism being known as *secondary* generation and the rapid growth of the runaway population has made secondary generation known as a runaway *avalanche*.

Starting with the orbit-averaged relativistic version of the Fokker-Planck equation:

$$-\frac{eE_{\|}\xi}{m_e c}\left(\frac{\partial f}{\partial p} - \frac{2\lambda}{p}\frac{\partial f}{\partial \lambda}\right) = C(f) + S, \tag{2.19}$$

with $p = \gamma v/c = v/\sqrt{c^2 - v^2}$: the normalised relativistic momentum and $\lambda = (1-\xi)/B$, $B = |\mathbf{B}|$. The relativistic collision operator becomes:

$$C(f) = \frac{1}{\tau_c p^2}\left[\frac{\partial}{\partial p}(1+p^2)f + \frac{1+Z}{2}\sqrt{1+p^{-2}}\frac{\partial}{\partial \xi}(1-\xi^2)\frac{\partial f}{\partial \xi}\right], \tag{2.20}$$

where $\tau_c = (c/v_{T_e})^3 \cdot \tau_{th}$ is the collision time of the relativistic electrons. $\nu_{ee} = 1/\tau_{th}$ is the thermal collision *frequency* which scales with the respective velocities. S a source of secondary generation runaways, which is proportional to the density of existing runaways, the seed, n_r and the frequency of close collisions $\nu_{ee} \cdot \ln\Lambda$). is given explicitly in Ref. [5].

A mathematical treatment was made by Rosenbluth and Putvinski [11]. They start with the bounce-averaged Fokker Planck equation and find analytical solutions in different simplifying limits and then interpolate the solutions together to form a full expression, with some conditions ($E = |E_{\|}|/E_c \gg 1$, $Z_{eff} = 1$, $r/R \to 0$). The full analytical

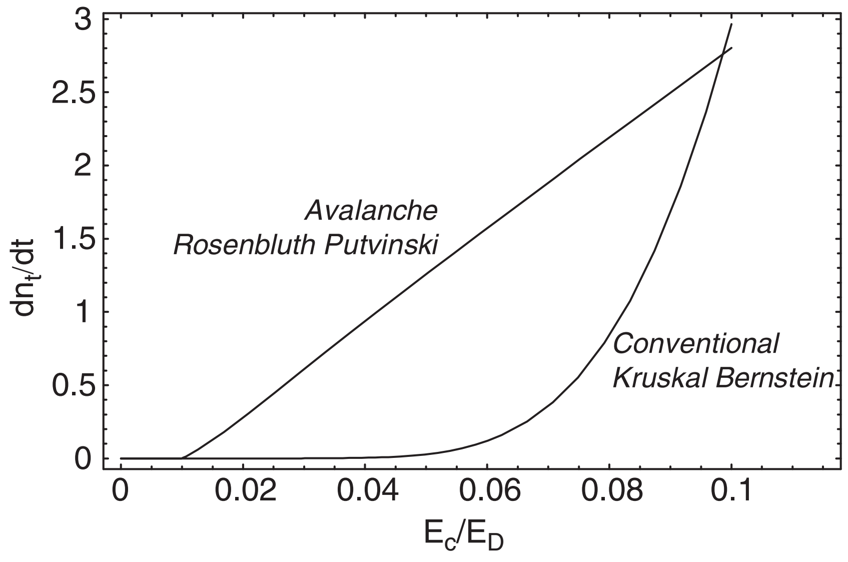


Figure 2.4: Primary runaway generation compared to avalanching. The vertical axis is generation rate $n_r'(t)$ and the horizontal axis is normalised electric field strength $E_c/E_D = T_e/m_ec^2$. Image from [5].

expression of secondary generation:

$$\gamma = \frac{dn_r}{dt} = \frac{(E-1)n_r}{\tau \ln \Lambda}\sqrt{\frac{\pi\phi}{3(Z_{eff}+5)}}\left(1 - \frac{1}{E} + \frac{4\pi(Z_{eff}+1)^2}{3\phi(Z_{eff}+5)(E^2+4/\phi^2-1)}\right)^{(-1/2)}, \tag{2.21}$$

$$\phi(\epsilon) = \frac{3}{4}\int \frac{2\pi \cdot \lambda d\lambda}{\oint \sqrt{1-\lambda b(\theta)}d\theta} \approx (1 + 1.46\sqrt{\epsilon} + 1.72\epsilon)^{-1}, \epsilon \gg 1, \tag{2.22}$$

$\epsilon = r/R$ and $E = |E_{||}|/E_c$ is the electric field normalised to the critical electric field. In the limit where $E \gg 1$, $Z_{eff} = 1$ and when $\epsilon \to 0$ a further simplification can be obtained:

$$\frac{dn_r}{dr} \simeq \sqrt{\frac{\pi}{2}}\frac{(E-1)n_r}{3\tau \ln \Lambda}. \tag{2.23}$$

This was shown to be within 20% of the value from simulation done in the same paper. Which of the primary and secondary generation is the most significant varies during a disruption, but it is clear that secondary generation becomes dominant once there is a large enough seed of runaway electrons. Avalanching tends to dominate for quite small fields, since primary generation is exponentially small in E, while secondary generation goes like $E-1$, see figure 2.4.

The fusion community has gradually realised that this secondary type of generation probably occurs in a large range of tokamaks. Indeed, it seems to be the dominant source of runaways once the seed has been created. It is also a motivation for ARENA which, being a Monte Carlo code, has a good outlook to make simulations with high energy runaways.

2.3 Numerical methods

To express the full dynamics of runaway electrons in a plasma, the time evolution of the Fokker Planck equation must be modelled numerically. This will more accurately simulate the runaway process outside of the limits used in analytical derivations, which is important to be able to make predictions. A numerical model also allows for simulating mitigation techniques such as killer pellet injection. The next chapter will go into explaining Monte Carlo techniques, which the numerics in ARENA is based on, the benefits when solving problems with large dimensionality and the relevance for ARENA.

3

Numerical Methods

ARENA IS BASED ON A MONTE CARLO TECHNIQUE to numerically iterate the solution of the Fokker-Planck equation. The term *Monte Carlo* is a very general description for any calculation which is not completely deterministic. The idea is that the problem at hand approximately follows some simple function, perhaps of a lower dimensionality than the starting problem, with a deviation that is, in most cases, assumed to be from a Gaussian probability distribution. By iterating through the (simple) model function with some stochastic deviating component, the solution to a complex problem, like the electron distribution in a plasma during a disruption, can be modelled in a reasonable time on a computer.

Below is a simple example of the idea behind the Monte Carlo method being used for numerical integration, and also a description of the *Metropolis algorithm*, which is a well established way of making numerical calculations on problems with high dimensionality. ARENA is based on an algorithm of this kind.

3.1 The Monte Carlo method

A Monte Carlo method is a powerful way to do computationally intensive calculations with the help of statistical averaging. In this descriptive example, imagine that you would like to numerically calculate an integral which for simplicity is one-dimensional (1D). As opposed to a numerical method like *trapezoid integration*, the Monte Carlo approach is completely insensitive to the dimensionality of the problem. That is, the error does not scale with the problem's dimension. So if the integral was n-dimensional, the error would decrease in the same way with the number of iterations for any dimensionality $n > 1$. If the integral of some function $f(x)$ is to be evaluated on the interval [0,1], the integral is taken as the *mean value* of some *randomly chosen* points x_i on the interval [14]

$$I_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{3.1}$$

With $f(x_i) = f_i$, the error can be estimated, assuming the mean, $\mu \equiv \mathbf{E}[f_i]$ and variance $\sigma^2 \equiv Var[f_i] = \mathbf{E}[f_i^2] - \mu^2$ exist. Then the sum I_N is also a stochastic variable with mean and variance:

$$\mathbf{E}[I_N] = \mathbf{E}[\frac{1}{N}\sum_{i=1}^{N} f_i] = \frac{1}{N}\sum_{i=1}^{N}\mathbf{E}[f_i] = \mu, \tag{3.2}$$

$$Var[I_N] = Var[\frac{1}{N}\sum_{i=1}^{N} f_i] = \frac{1}{N^2}\sum_{i=1}^{N} Var[f_i] = \frac{\sigma^2}{N}. \tag{3.3}$$

Now, the central limit theorem states that the probability distribution for the stochastic variable I_N will tend to a normal- distribution as $N \to \infty$, i.e. $I_N \sim Norm(\mu, \sigma^2/N)$ for large N. So the Monte Carlo integral takes on the form:

$$I_N = \langle f\rangle \pm \frac{1}{\sqrt{N}}\sqrt{\langle f^2\rangle - \langle f\rangle^2}, \tag{3.4}$$

with plus or minus the error that is 1 standard deviation, which is the square root of the variance.

There are two ways to lower the error: choosing a higher number of random points, that is, increasing N, or by lowering the variance. This can be done by making a weighted selection of points. Instead of choosing random points uniformly, they are chosen according to some probability distribution that is similar to the function that is being integrated. For example, consider integrating the 1D quadratic function

$$I = \int_0^1 x(1-x)dx. \tag{3.5}$$

Then the weight function

$$P(x) = \frac{\pi}{2}\sin x \cdot \pi \tag{3.6}$$

can be used to sample the parameter space, here just the line, and one then solves for:

$$I_N = \int_0^1 \frac{x(1-x)}{P(x)}P(x)dx. \tag{3.7}$$

The weight function should have the same properties as the function of interest. Here, for instance, $P(x)$ has a similar curvature and the same maximum value as $f(x)$.

An example in two dimensions is a circle, where the area can be integrated from random points chosen on the surface $\mathbf{r} \in [-r,r] \times [-r,r]$, where r is the circle radius. simple example is to calculate the area of a circle by integration. The circle area is then

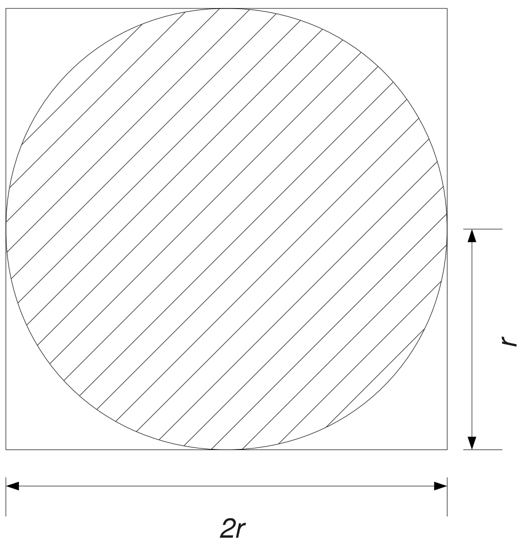


Figure 3.1: The ratio between the surface of the circle and the surface of the square is $\pi/4$.

the fraction of random scattered points (test particles) that happen to appear inside the circle times the square area: $4r^2$, see figure 3.1).

$$A_c = \frac{p_{\texttt{inside}}}{p_{\texttt{tot}}} \cdot (2r)^2 \tag{3.8}$$

One of the main disadvantages with a Monte Carlo method in the context of ARENA is its slow convergence ($\sim \sqrt{N}$) compared to finite difference/finite element methods.

3.1.1 Metropolis, Markov chains and selection

In the Metropolis algorithm, the test points are selected in sequence where each new point depends only on the previous one in a so called *Markov chain.* The next point in the sequence is accepted depending on a fitting function with some probability. The fact that the actual fitness function value (or energy state of some particle configuration) can be hard to calculate is worked around in some sense by this algorithm, since only the fraction of two subsequent states is considered. This cancels out constant terms in the energy expression.

The Metropolis algorithm and its variations are examples of classical optimisation techniques which are applicable for non-smooth problems and high dimensionality. It can be distinguished from some similar approaches to a general optimisation problem, like evolutionary algorithms or neural networks. If a test-function can be easily obtained, and the fitness can be duly calculated, this is a good approach. It does however, have limitations when it comes to the speed of convergence. Already in ARENA this is present, where results tend to improve drastically with a higher number of test particles.

As a simple example, consider the same integration problem as the one in equation 3.5, and using the weight function $P(x)$ as a fitness function, the trial for each new test point is whether

$$T(x_{k+1}) = \begin{cases} \frac{P(x_k+1)}{P(x_k)} > 1, \text{Always keep } x_{k+1} \\ \frac{P(x_k+1)}{P(x_k)} < 1, \text{Keep } x_{k+1} \text{ with some probability} \end{cases} \tag{3.9}$$

Now, choose an arbitrary starting point, in principal this could be anywhere on the line, and run the algorithm until a set of points x_i are obtained. This is the Markov chain, and the mean of the set of values $f_i = f(x_i)$ will be the approximation for the integral according to equation 3.4.

The test points do not have to be points in space, which are averaged over to calculate an integral. One example is for describing an energy state of a helium atom, where the Hamilton equation needs to be solved for some wave equation in space and time. Another is variations of the Ising model for modelling a lattice of dipoles, where there is no simple analytic solution (see e.g. [15]).

3.2 Monte Carlo in ARENA

In ARENA, particles count as runaways once they pass a specified threshold in the momentum variable p_c. When that happens, particles are kept at constant count by using a stochastic normalisation technique. The particle parameters also have a random component where the time updates of the model parameters (i.e. momentum $\dot{p}$) are governed by *Monte Carlo operators*. These are explained in depth in the next section, see specifically section 4.3. The fact that particle count is preserved, but the energies of individual particles are uncorrelated, enables the Monte Carlo model to handle a large span of energies, keeping the momentum variables p that can grow to infinity, unbounded. A finite element method has troubles with a large parameter space, since either the resolution becomes very low with large bins for the energy levels, or the number of bins becomes very large which impacts performance. For the application of runaway electrons, this is most important for secondary generation which directly depends on the distribution of the runaway tail which can only be modelled to a maximum value p_{max} in a finite element code. The finite element model can have a very much faster convergence time $t_{conv} \gg \sqrt{N}$. So, to model the distribution function for low energies, i.e. for primary generation is really fast. This type of solution is employed in LUKE [16].

In ARENA, the electrons in the fusion plasma are described by a smaller number of modelled Monte Carlo particles (scaled up with the number density n_e), and the Fokker-Planck differential equation is solved by iterating the corresponding Monte Carlo operators in time. The rigorous formulation is described in the next chapter.

4

ARENA Code

ARENA is written in the Fortran programming language. It employs a series of program blocks and an input-output system of text files. Since the code has been in development throughout my thesis work, I will try to refer to code outputs with a revision number to clarify what code has been used. The prospect of making a parallel calculation code on a computer cluster or a graphics chip has also been discussed and is included in section 4.10. I have been working closely with Gergely Csépány and Gergely Papp with this, and they will be mentioned in the section below.

To simulate the time evolution of the runaway generation from a disruption is in principal a question of solving the Fokker-Planck equation of motion for the particles of interest; the electrons. There are various approaches to this, some of which were described in the section of general modelling (chapter 2), with advantages and disadvantages to each approach.

The Monte Carlo model describes the continuous distribution function evolution with a set of *test particles* which represent the population of electrons in the ensemble. They can be categorised as either *thermal-* or *runaway-*electrons and have three phase space variables (p,λ,r), describing their properties in phase space. There is also an additional parameter $\sigma \in [-1,1]$ which indicated the toroidal direction of the test particle. Recall that the random velocity of a particle in the *toroidal* direction will determine its travel direction, so ions and electrons will travel around the torus both ways. (if a particle is trapped in a banana orbit $\sigma = 0$ since the particle adds to the current in both directions).

In each iteration the test particles are updated with a set of *Monte Carlo operators* giving the phase space variables a random kick with an appropriate magnitude $\Delta\mathbf{I}, \mathbf{I} = (p,\epsilon,\lambda)$. The distribution function can then be recreated from the set of test particles.

Monte Carlo techniques have proven to be powerful when approaching multidimensional diffusion problems. The benefits are e.g. that the main part of the computer program is not too complex, and there are no particular problems with boundary conditions.

The electric field is calculated self-consistently, which is important, since the runaway current will affect the field strength and profile during the time of the disruption. The loss of energy due to synchrotron radiation will enter into the calculation once the runaways are starting to reach the speed of light.

4.1 ARENA versions

In 2011, ARENA was updated from Fortran 77 to the newer Fortran 90 programming language. This has several benefits; The old code was slow - making it very clear that convergence time scales with $\sqrt{n}$. In the new code, precompiled libraries and smarter reuse of code in combination with modern compilers and benchmarking tools has created a completely different situation from a software development point of view. The two code versions are referred to throughout this paper as ARENA 90 and the *old* ARENA.

The update was made in part by automatic conversion of the code, and in part by manually going through the functions. The differences in the two versions have been documented in several unpublished reports during the course of this spring (2012) which are available from the SVN repository at ITM for reference. There has been a number of additions to the original Fortran 77 code, such as an added quick testing framework to be used in development as well as bug fixes and optimisations. The code is currently checked into version control in ITMs Gforge, available to project collaborators.

It is important to understand the state that ARENA was in when we started working with it. Lacking heavily in documentation and full of logical abbreviations, as well as inconsistent algorithms compared to the published results [2], a significant portion of the work on this code has been reverse-engineering and documentation work. I will spend some time in this section describing the design choices of some calculations, as well as some discoveries of code function that was made during the work. More detail on ARENA and the iterative work on the code can be found in section C, documentation in Gforge and relevant reports.

4.2 Code overview

The core program of ARENA is simple in principal, the input values are chosen appropriately and the program is then iterated for the desired amount of time, $dt \cdot n_{timestep}$ with a set time step length and the desired number of test particles. A schematic picture of the program structure can be seen in figure 4.1. The modelled variables that make up phase space can be chosen somewhat arbitrarily. According to the general theory [17, p. 8], any invariants of motion are possible choices. The choice made in ARENA is *inverse aspect ratio*, ϵ which indicates the radial position, *normalised momentum* $p = m\gamma v/(mc)$, the particle's energy and *perpendicular momentum* λ (see appendix A). The parameter λ is defined to indicate whether a particles is running freely around the torus (passing: $\lambda < 1$) or *trapped* ($\lambda > 1$) in a banana orbit. Finally, an electron's direction is indicated by $\sigma = \pm 1$ (or $\sigma = 0$ for trapped particles) this is assigned randomly and does not change over the course of the simulation. σ enters into the time derivative update of the E-field's influence on the electrons and is also important for recreating the distribution function. The main program loop is in a file called `arena.f90`. The main program loop calls the following subroutines in sequence:

```
    call read_input
    call initial_output
    call initek
    call init
    ! outputs
    call print_global_parameters
    call initial_output
    call momenta_output
    if do quicktest
        open raw_output.txt
    end do
    ! main loop
    call write(p, eps, lam)
    call momenta_output
    control ! <— set sub—timestep and set flag
    do 1, nparticles
        call mccalc ! <— update MC values
        call source ! <— create secondary particles
        call diagn ! <— updates the runaway current, create output vars
        ! call efind ! <— calculate parallel E—field for next time step
        ! call updtpar ! <— update background parameters
        ! call adjvel ! <— adjust velocity distribution to fit new params
    end do
    call momenta_output
    call final_output
print_output
calculate runtime
```

The first routines read in input parameters and calculate static values that are used throughout the simulation, this is `read_input`, `initial_output`, `initek` and `init` with subroutines. `momenta_output` writes every test particle parameter value to a text file for analysis outside of ARENA. This is always done at the end of the simulation, but output can be created at time intervals by setting the `plot_interval` property in `arena_input.xml`. In `control`, the program orders the test particles and assigns calculation flags so that the appropriate update action is taken for runaway- and thermal electrons respectively. `mccalc` applies the collision operator, the energy gain from the **E**-field and the energy loss to synchrotron radiation. The `source`-subroutine has to do with adding new runaway electrons from close collisions and `sortmc` does a sorting and re-weighting of test particles.

Note that the `efind`, `updpar()` and `adjvel()` subroutines for running the self-consistent electric field calculation were commented out with the Fortran `!` comment symbol. These routines serve the secondary generation which is still disabled in ARENA 90. `final_output` outputs the final results to text. The `mcop` subroutine is the most

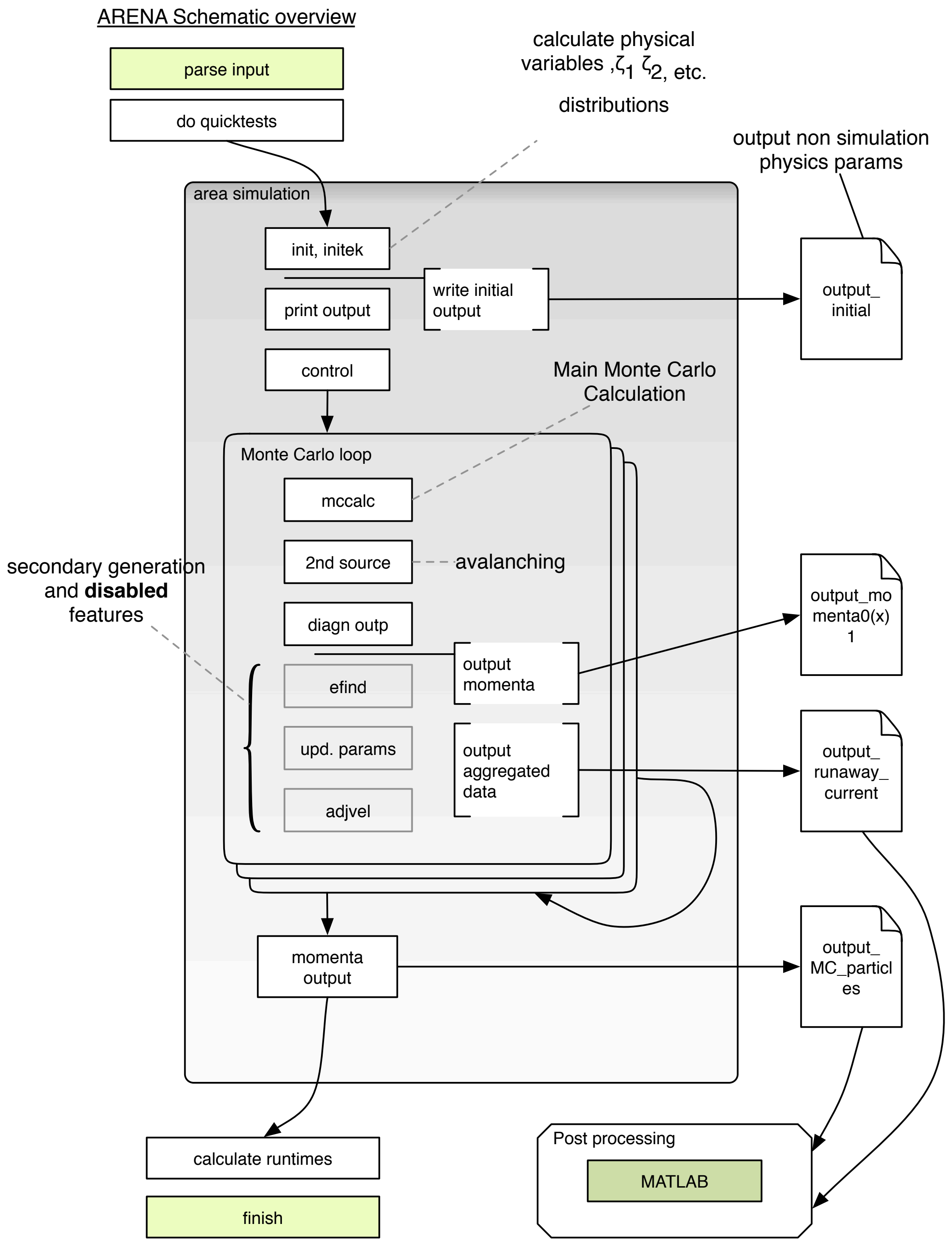


Figure 4.1: An overview of the execution order of ARENA and the output files.

involved subroutine where the particle parameters are updated. A shortened version of the execution order can be found in the code snippet below.

```
subroutine mccalc
    call mcop ! get p_dot and A_p lam_dot, A_lam with collision op.
    call mcspnb ! update eps (NOT IN FINAL CODE: deps = 0)
    ! apply corrective drift terms at boundaries
    lmax = 1.0 / (1.0-2.0*eps)
    if (abs(lam-lmax) .lt. 1.0e-3) zalam = 0.0
    call find_dtmin() ! find small enough timestep

    ! update phase space variables (p, lam, eps)
    dp = p_dot * dt + A_p * r_p * sqrt(dt)
    p = zp + zdp
    deps = eps_dot * dt + r_e * A_eps * sqrt(dt)
    eps = eps + deps
    dlam = lam_dot * dt + r_l * A_lam * sqrt(dt)
    lam = lam + dlam

    call currch ! update the runaway current
    time = time + dt * tconv
end subroutine
```

4.3 Monte Carlo operators

The Monte Carlo operators are used in ARENA to update each term in the Fokker-Planck equation iteratively [2]. The Fokker-Planck equation can be written down schematically with the different terms affecting the time derivative of the test particle distribution in the right hand side:

$$\frac{\partial f}{\partial t} = \langle L_E(f)\rangle + \langle C(f)\rangle + \langle L_{synch}\rangle + \langle L_{s.t.}(f)\rangle + \langle S\rangle + \langle l_B\rangle. \tag{4.1}$$

The terms in angled brackets are orbit-averages - they all share the common form:

$$\langle\Theta\rangle = \frac{\oint(\Theta)\frac{d\theta}{2\pi\sqrt{1-\lambda b(\theta)}}}{\oint\frac{d\theta}{2\pi\sqrt{1-\lambda b(\theta)}}} \tag{4.2}$$

These terms describe the modelled plasma properties: L_E is a term to account for the accelerating **E**-field, C is the collision operator that accounts for Coulomb collisions, L_{synch} The loss of energy to synchrotron radiation, $L_{s.t.}$ is radial transport of particles due to diffusion. S is a source of runaway electrons from secondary generation and l_B a loss term that balances the source term in order to keep the particle count constant.

The most involved term is the collision operator. It is described in the next section. The others describe change in p and λ from radial transport through bremsstrahlung, *synchrotron radiation* and the impact of the outer *electric field* respectively. The full update of the momentum p looks like:

$$p(k, t_i + 1) = p(k, t_i) + dt \cdot \dot{p}(k, t_i) + r \cdot A_p \sqrt{dt} \tag{4.3}$$

where dt is a user input which is sometimes recalculated from the sub-time step dt_s (see section 4.7.2, and r is a random uniform number with unit variance and mean zero. The $\dot{p}$ term is made up of three sub-terms:

$$\begin{cases} \dot{p} = \dot{p}_{coll} + \dot{p}_E + \dot{p}_{synch} \\ \dot{\lambda} = \dot{\lambda}_{coll} + \dot{\lambda}_E + \dot{\lambda}_{synch} \end{cases} \tag{4.4}$$

Here,

$$\begin{cases} \dot{p}_E = \dfrac{\sigma}{\zeta_1} \cdot \dfrac{E}{E_c} \\ \dot{\lambda}_E = -\dfrac{2\lambda\sigma}{p} \cdot \dfrac{E}{E_c} \end{cases} \tag{4.5}$$

with ζ_1 the elliptic integral, σ the travel direction of the particle, and E/E_c the input momentum normalised to the critical momentum.

$$\begin{cases} \dot{p}_{synch} = -p^2\sqrt{1+1/p^2}\left[\lambda + \left(\dfrac{m_e c}{eRB}\right)^2 p^2 (1-\lambda)^4\right] \cdot 3.3 \cdot 10^{-4} \dfrac{B^2}{E_c} \\ \dot{\lambda}_{synch} = -2\dfrac{\zeta_2}{\zeta_1}\dfrac{\dot{p}_{synch}}{p(1+p^2)} \end{cases} \tag{4.6}$$

with B, the magnetic field strength, m_e electron mass, e elementary charge, R the major radius, c the speed of light and p the normalised momentum.

These equations are executed in the main loop (see figure 4.1).

The Monte Carlo operators are updated in the `mccalc()`-function which contains some routines for setting a small enough time step as well as corrective drift-terms, as well as the `mcop()` function which updates the program parameters (p,λ,ϵ). The code has been edited for brevity, but the full formulation is available in the code at ITM.

```fortran
subroutine mcop (peps, pp, plam, ...)
    zltest = 1.0 + 0.2 * peps / (1.0-2.0*peps)

    if (plam .gt. 0.98 .and. plam .lt. zltest) then
      zlam = 0.98
      call ifunc (peps, zlam, zi1, zi2) ! creates the elliptic integrals
      call mccoll (peps, pp, zlam, zi1, zi2, zpdc1, zapc1, zldc1, ...
          zalc1) ! gets

             !... reiterations for numerical purposes

      call ifunc (peps, plam, zi1, zi2) ! recalculate elliptic ...
          integrals with new point values.
```

```
        call mcelec (peps, pp, plam, ksig, zi1, zpdel, zldel, zpdeln, ...
            zldeln)
        call mcspat (peps, pp, plam, ksig, zi1, zi2, zepsd, zaeps)
        call mcrad (peps, pp, plam, ksig, zi1, zi2, zpdrad, zldrad)
      else
        call ifunc (peps, plam, zi1, zi2)
              call mccoll (peps, pp, plam, zi1, zi2, zpdc1, zapc1, ...
                  zldc1, zalc1)

        call mcelec (peps, pp, plam, ksig, zi1, zpdel, zldel, zpdeln, ...
            zldeln)
        call mcspat (peps, pp, plam, ksig, zi1, zi2, zepsd, zaeps)
        call mcrad (peps, pp, plam, ksig, zi1, zi2, zpdrad, zldrad)
      end if

      ppdot  = zpdc1 + zpdel + zpdrad ! this outputs p_dot
      pldot  = zldc1 + zldel + zldrad ! this outputs lambda_dot
      pap    = zapc1
      palam  = zalc1
      pedot  = zepsd
      pae    = zaeps
      ppdote = zpdel
      pldote = zldel

      return
  end subroutine mcop
```

4.4 Collision operator

The collision operator which describes collisions between particles updates mainly the particle momentum $p \to p + \dot{p}dt$, but also the pitch angle variable λ. The collision operator in ARENA is a two-part expression. One part is valid for high energy particles and one for low-energy ones. They are then splined together using a logistic function which is plotted in the appendix (C). The low energy part of the collision operator depends on the Chandrasekhar function, which the high energy expression does not. The high energy expression is:

$$\dot{p}_{HE} = -p\nu_p(p) + \beta\left[\frac{2\nu_p(p)}{p} + \nu_p'(p)\right]\sqrt{1+p^2} + \frac{p\nu_p(p)}{\sqrt{1+p^2}} + \frac{\nu_p(p)\sqrt{1+p^2}}{(p+1)\ln\Lambda}, \tag{4.7}$$

$$A_{coll}^{pp} = 2\nu_p(p)\beta\sqrt{1+p^2}, \tag{4.8}$$

with

$$\begin{aligned} \nu_D(p) &= 2\frac{\sqrt{1+p^2}}{p^3}\left[1 + Z_{eff} - \beta\frac{1+2p^2}{p^2(1+p^2)}\right]\left[\frac{1+\ln p+1}{\ln\Lambda}\right], \\ \nu_p(p) &= \frac{1+p^2}{\tau p^3}\left[\frac{1+\ln p+1}{\ln\Lambda}\right]\frac{p^4}{p^4+(0.4p_{th})^4}, \end{aligned} \tag{4.9}$$

where $\beta = T_e/(m_0c^2)$. ζ_1 and ζ_2 are *full elliptic* integrals over the poloidal angle θ. The expressions used in the code are a generalised version of the expressions for the collision Monte Carlo operators in [2]. It can be shown that this agrees with the published theory in the relevant limit $\beta \ll 1$. The low energy expression is

$$\dot{p}_{LE} = p_{th}\nu_p(p)\left[\frac{1.1248 \cdot e^{-x^2}}{x} - G(x)(2+1/x)\right], \tag{4.10}$$

$$A^{pp}_{coll} = 2\nu_p(p)p^2_{th}(1+p^2), \tag{4.11}$$

with $x = v/v_{th}$, v_{th} the thermal velocity (corresponding to p_{th}) and $G(x)$ the *Chandrasekhar* function. The updates for the λ parameter is the same across all energy levels.

$$\dot{\lambda}_{coll} = 2\nu_D(p)\left(\frac{\zeta_2}{\zeta_1} - \frac{\lambda}{2}\right), \tag{4.12}$$

$$A^{\lambda\lambda}_{coll} = 2\nu_D(p)\lambda\frac{\zeta_2}{\zeta_1}. \tag{4.13}$$

The elliptic integrals take on the approximate expressions in the limit $\epsilon = r/R \ll 1$:

$$\zeta_1 = \oint \frac{d\theta}{2\pi\sqrt{1-\lambda b(\theta)}} \approx \begin{cases} \dfrac{2K(1/\kappa)}{\pi\sqrt{1-\lambda+2\epsilon\lambda}}, \text{ trapped particles} \\ \dfrac{2K(\kappa)}{\pi\sqrt{2\epsilon\lambda}}, \text{ passing particles} \end{cases} \tag{4.14}$$

$$\zeta_2 = \oint \frac{\sqrt{1-\lambda b(\theta)}d\theta}{2\pi} \approx \begin{cases} \dfrac{2\sqrt{1-\lambda+2\epsilon\lambda}}{\pi}E(1/\kappa), \text{ trapped particles} \\ \dfrac{2\sqrt{2\epsilon\lambda}}{\pi}(E(\kappa)-(1-\kappa)K(\kappa)), \text{ passing particles} \end{cases} \tag{4.15}$$

Whether the particle is passing or trapped in a banana orbit is determined by the trapping parameter κ which is related to $\lambda = p^2_\perp/p^2 b(\theta)$ as:

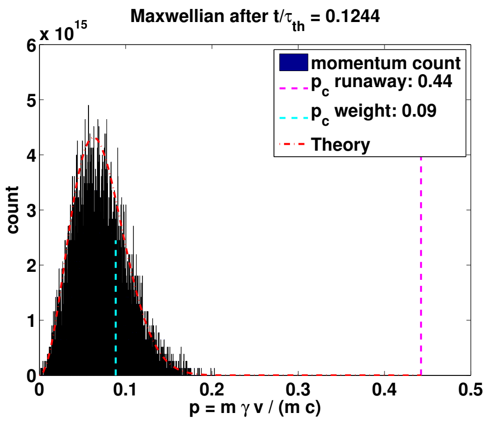
$$\kappa = \frac{1-\lambda+2\epsilon\lambda}{2\epsilon\lambda}. \tag{4.16}$$

Here, $b_{norm}(\theta)$ is the normalised B-field on the poloidal angle θ.

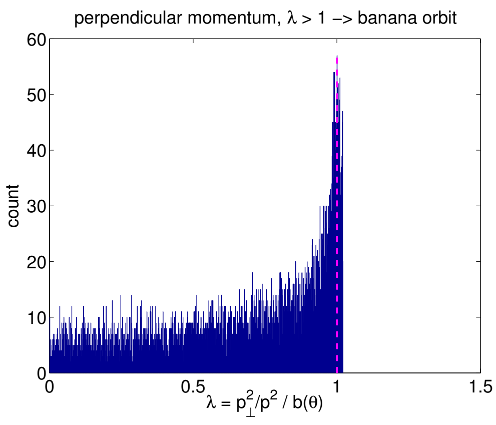
In ARENA, the elliptic integrals are reused depending on the particles radial position $\epsilon = r/R$, however the values depend only on geometry, so the calculation can be made outside of the loop over particles to save time. The logistic function in the collision operator returns a real valued number $\in (0,1)$ and is used to spline together two components p_{HE} for high energies and p_{LE} for low energies, of the collision operator

$$\dot{p}_{coll} = logis(p) \cdot \dot{p}_{LE}(p) + (1 - logis(p)) \cdot \dot{p}_{HE}(p) \tag{4.17}$$

In the original ARENA code, the output value was cast to a logical which is either exactly 0 or 1. The ARENA90 version has been patched up to perform a correct splining of the two operators.



(a) p distribution



(b) λ distribution

Figure 4.2: The initial distribution of momenta for the test particles ARENA 90. In the right hand side figure is the distribution of pitch angle variables $\lambda = p_\perp^2/(p^2 b(\theta)), b(\theta)$ normalised magnetic field strength and θ the poloidal angle. The vertical line shows which particles ($\lambda > 1$) are trapped in banana orbits.

4.5 Initial distribution

ARENA evolves the kinetic equation in time starting with an initial phase space distribution $f_{t=0}(\epsilon,p,\lambda)$. The collision operator is discussed in detail in section 4.4, while the other calculations of interest are described here. The initial distribution for ARENA is hard coded to be a Maxwellian population of thermal electrons. This is done with some stochasticity via a series of loops which are documented in the manual checked into the code repository.

A typical initial distribution from ARENA can be seen in figure 4.2. To the left is the distribution of momenta p and to the right, the distribution of the pitch angle variable λ. Not plotted is the inverse aspect ratio distribution $\epsilon = r/R$, which is simply a uniform distribution of particles along the simulation radius $r \in [r_{min},r_{max}]$. The starting distribution is generated in quite a complicated manner and should probably be replaced by a more flexible input handler that could use different starting distributions. The initial distribution is Maxwell distributed around the input temperature, here $T =$ 1.0 keV. It can be written as

$$f_{Maxw} = C \cdot e^{-\frac{\sqrt{1+p^2}-1}{p_{th}^2}\cdot p^2}. \tag{4.18}$$

Fitting the parameters C and p_{th}^2, one obtains: $C = 11000, p_{th}^2 = 0.0015$ for the initial distribution in figure 4.2. This distribution is the local energy minimum and should be maintained when there are no runaway electrons or an electric field, which is the *Maxwell test*.

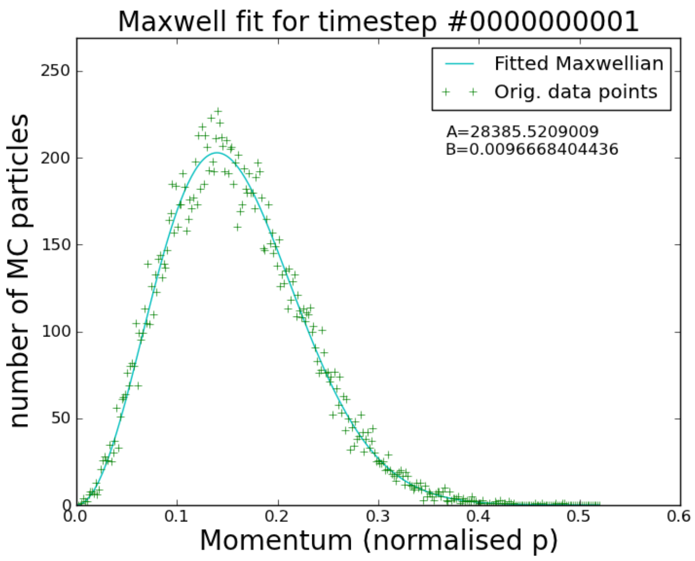


Figure 4.3: This is an example of the Maxwell test output. This image was generated with the Python supporting software which does curve fitting and outputs the graphics.

4.6 Benchmarks

There are three basic tests that we want to be able to run to confirm the validity of the code. These were run by Eriksson and Helander [2] and are basic sanity tests that verify the physical relevance of the computer program.

The *Maxwell test* validates whether the particle velocity distribution will tend towards a Maxwellian (Gaussian) distribution when there is no outer E-field, that is when the outer electric field (`E_applied_norm2cr`) $E/E_c = 0$ which would be the case during normal operation with no thermal quench.

An example of the histogram output of particle momenta can be seen in figure 4.3. The Maxwell distribution function can be written according to equation 4.18, and fitting the bar-chart to the Maxwellian will yield the fitting parameters C and p_{th}^2.

The *Dreicer test* verifies whether the growth rate of runaway electrons increase with the outer electric field $E = E_{||}/E_D$ according to theory. The theoretical expression used for reference is the Kruskal-Bernstein equation (2.18), with the parameter $k = 0.35 \sim 1$, which was fitted from Kulsrud's numerical values [1]. The effective plasma charge $Z_{eff} = 1$ indicates no or low impurities, and the derivative is calculated from $d(n_r/n_{tot})/d(t/\tau_{th})$, with $\tau_{th} = \nu_{ee}^{-1}$, the electron-electron thermal collision frequency. formulations, normalised to a function of the thermal velocity: $k \sim (m_e c^2/2T)^{(3/2)}$. strength and $E_{||}$ is the electric field component in the toroidal direction. The reference curve plotted in figure 4.3 is:

$$\gamma_D = \frac{dn_r}{dt_\tau} = 0.35(E/E_D)^{-3/8} \cdot e^{-\sqrt{2E_D/E} + E_D/4E} \tag{4.19}$$

with the normalised fraction of runaways and time.

4.7 Inputs

ARENA is designed with an input XML-interface for making parameterised simulations. There is a large number of possible inputs to vary and I have tried to list and comment on the physically relevant ones here. The inputs appear in the file `arena_input.xml` sectioned into part that describe their approximate impact on the ARENA simulation. A full list can be found in section C.

The important inputs are the program parameters that decide the update frequency and calculation parameters for ARENA and the physical parameters, which are set appropriately for what simulation is to be run. To set up any simulation, one needs to decide on a number of test particles n_p, time step size dt and number of iterations n. Typically $n_p = n \sim 10000, dt \sim \tau_{th} \sim 10^{-6}$. Since the time step is rescaled what mainly affects simulation time is n and n_p.

The applied electric field E, which is input as fraction of critical electric field E_c, under which no runaway generation occurs. Thus for $E/E_c < 1$, no runaway generation should occur. Next, the plasma number density n_e and core temperature T_e which affect the bulk of momenta for the test particles. The temperature is important in particular since e.g. the Dreicer field is a function of temperature $E_D = E_D(T)$:

$$\frac{E}{E_D} = \frac{E}{E_c \cdot c^2/v_{th}^2} = \left\{ v_{th}^2 = \frac{T}{m_e} \right\} = \frac{E}{E_c} \cdot \frac{T}{m_e c^2}. \tag{4.20}$$

Note that the Dreicer field $|E_{||}|/E_D \in [0,1]$, since it is the limit for when all electrons run away.

4.7.1 Radial profiles

The *temperature profile* and *density profile* as functions of radial position ($\epsilon = r/R$) are used to model the change in plasma temperature and density as one moves along the radial coordinate. They are modelled as exponentially decreasing functions starting from the magnetic axis:

$$T(\epsilon) = T_c \left(1 - T_b \cdot \left(\frac{\epsilon}{\epsilon_{max}} \right)^2 \right)^{\gamma_T}, \tag{4.21}$$

where T_c is the `T_central_electron` input parameter, T_b is `bulk_tem_prof` and γ_T is `bulk_tem_prof_exp`, the exponent. Similarly the density profile is preprogrammed as an exponential decreasing function, the equation reads:

$$\rho(\epsilon) = n_c \left(1 - n_b \cdot \left(\frac{\epsilon}{\epsilon_{max}} \right)^2 \right)^{\gamma_n}, \tag{4.22}$$

with n_c `n_bulk_elec_central`, n_b `bulk_dens_prof` and γ_n `bulk_dens_prof_exp`. For the cases tested here, the range of ϵ is small, $\epsilon \in [0.01,0.011]$ and it is assumed that these functions are not sensitive to such a small change in ϵ, therefore, the temperature can be approximately set equal to the central electron temperature: T_c.

4.7.2 Time step

To make each time step update consistent with the physics involved, recall that a Coulomb collision is measured whenever an electron changes its velocity vector *significantly* due to Coulomb interaction with other particles. It is from this that one defines the collision time which depends on the *velocity* of each particle. The thermal collision time τ_{th} therefore is defined from the bulk expectation particle velocity $v_{th} = \sqrt{T/m_e}$. It is necessary to update the particles every thermal collision time on average, lest the updates cannot keep up with the change in velocity (dp/dt). ARENA changes the sub-time step is dynamically over the course of the simulation, to account for this. Although the user chooses an *output* time step for when to save the output data (see below). If the output time step is chosen larger than a thermal collision time, the time step subdivided into an integer number of steps that are smaller than a thermal collision time:

$$dt_{sub} = \frac{dt}{\texttt{ceil}(dt/\tau_{th})}, \tag{4.23}$$

where `ceil()` denotes the ceiling function that adds up to the nearest integer. Next, the time step is checked for rapid changes in the time derivative of the modelled parameters. For p normalised momentum, the timestep is set so $\Delta t \leq \Delta p(t)$:

$$\frac{1}{100} \cdot dt_{sub} \leq dt_{final} \leq \left| \frac{1}{10} \frac{p}{\dot{p}} \right|. \tag{4.24}$$

Note, that the minimum allowed time step is hard coded to 1/100th of the sub-time step dt_{sub}, this is apparently an arbitrary limit and could have all sorts of implications on the time steps allowed to the simulation time required. strange behaviour if certain time steps are chosen. It would probably be a good idea to change this behaviour to set the dynamic time step to a fraction of the thermal collision time. In fact there is little reason that the user should not get the maximum possible resolution of the output, so the time step input parameter could be replaced with a programmatically set resolution being perhaps equal to the thermal collision time. If this time changes, the output frequency could stay the original thermal collision time τ_{th} while the sub-time step keeps on adjusting as described here.

4.7.3 Weights

The test particles in the old ARENA are weighted to focus on runaway electrons in order to not spend valuable computer cycles calculating collisions between thermal electrons, which are known to be Maxwell distributed. Part of the momentum distribution is assigned a higher weight, while the rest of the distribution is weighted down to keep the physical number of particles constant. The border is denoted by $p_{w-cut} = \texttt{FWB}p_{th}$ where `FWB` is an input parameter. The limit can be seen in figure 4.2. The weights are initialised at the beginning of the simulation to be equal, they then change throughout the simulation according to a re-weighting algorithm.

The ratio of the weights is usually in the order 1, however an issue with the code tended to produce erroneous results with a factor 2 higher growth rate of runaways than expected. Finally the weighting was disabled in ARENA 90. The impact on simulation speed is less than what had been gained through other optimisation measures, and in fact in some times the speed is better, i.e. when updating many electrons, since the resorting and re-weighting itself takes up some computer cycles. The final output matches the benchmark perfectly.

The weight output `weight_fast` and `weight_bulk` scale with the particle density n_e, so it is suitable to consider the weight ratio $w_r = w_f/w_b$, normally $w_r > 1$, with a higher weight for high energy electrons. It is probable that the time saving effect or weighting is more important for secondary generation, which we have not yet implemented in ARENA 90.

4.8 Outputs

To process outputs from the Fortran output to text files, two code suites have been developed: one was written by me using `MATLAB` and one was written by Gergely Csépány using `matplotlib` and the Python programming language. Both plotting tool sets are available in Gforge.

The main outputs that are considered are p, λ and ϵ for the particles. These cannot be output for every iteration, but is output at the end of the simulation and at some interval, given by input parameter `plot_interval`. Secondly, a time step output which is an aggregate of distribution parameters, mainly the number of runaway electrons n_r. The time step output is needed to create the graph for the Dreicer primary generation rate as a function of outer electric field, and the particle property output is required to draw the p distribution graph. outer electric field, To plot a Dreicer curve similar to the one in [2], I need to calculate the saturated growth-rate of runaways during the run. For low E-fields, there is a reasonably long section of the curve which is flattened out, indicating a sustained growth rate (see figure 5.7. For high **E**-fields, $\sim E/E_D = 0.1$ and higher, the sustained rate is a limit rather than a maximum. I find the growth rate to plot in figure 5.1, $\gamma = dn_r/dt$, by choosing the maximum of the smoothened differentiation of the fraction of runaway electrons from my output. It was considered plotting the fraction of thermal electrons decreasing as more runaways are created, which is not directly output from ARENA. But this is the exact same thing as the bulk population minus the runaways, $n_{thermal} = n_{tot} - n_{rw}$, and so:

$$\frac{n_{thermal}}{n_{tot}} = \frac{n_{tot} - n_{rw}}{n_{tot}} = 1 - \frac{n_{rw}}{n_{tot}} \tag{4.25}$$

A positive slope to plot for this value would imply that one plots $-dn_{thermal}/dt = -d/dt(1 - n_{rw}/n_{tot})$ but this is of course just the same as $d/dt(n_{rw}/n_{tot})$ since, the time derivative of the constant is 0.

It is also possible to calculate e.g. the runaway current, which is a constant multiple of the fraction of runaways, however for quantitative comparisons with e.g. the LUKE code, the fraction of runaways is the main output considered.

ARENA 90's time step output is written to a file `output_runaway_current`, which in fact outputs the number of runaway electrons and not the *current*, after scaling up test particles to real electrons. The particle distribution is written to `output_MC_particls` which can be used to recreate a distribution function of momenta, and is used in this thesis for the Maxwell test. To get the actual number of particles, this output must be multiplied with the rescaling weight `weight_fast`, which is equal to `weight_bulk` in the final ARENA program. In the old ARENA code, the respective files are `pl1.res` for the time step output and `pl4.res` for the particle output.

To verify the validity of the output, plotting the number of runaway electrons as a fraction of the total number of electrons is a good idea. With a constant E-field, the fraction should tend towards 1 after a certain simulation time, which is the case for the simulations here.

To plot the output a series of MATLAB scripts are used. They read in the output files along with input parameters from `arena_input.xml` and `input.data` (old ARENA) respectively. Furthermore, some parameters which are used in analysing the output, like τ_{th}, the thermal collision time are written to an output file called `output_initial_parameters` (or `arena.output`). These files are also parsed with MATLAB.

4.9 Development and documentation

While the old ARENA was used in publications, the code that I started working with together with Gergely Csépány and Gergely Papp could not reliably reproduce the benchmarks seen. It was unclear exactly how to interpret the inputs and outputs. The units were not always stated and many physical quantities were not explicitly calculated, but rather written down as a number of constants clumped together. Furthermore the use of input parameters was not consequential, meaning that some input parameters, like the Coulomb logarithm which is an input parameter *Coulomb_logarithm* were hard-coded into some equations to a default number. When we were certain of the interpretation of the code, the primary generation tests would not pass but for an added multiple factor of 2, and the start of the runaway generation process would not correspond in a timely manner to the reference data.

Part of the issues were solved by removing ARENA's weighting, letting all particles have the same weight. This change made the primary generation test and Maxwell test for low temperatures match perfectly and surprisingly also improved performance. Probably the reason for this is that many of the thermal particles are still updated during early phases of primary runaway generation and the added computation in assigning weights added computer cycles, while none were really saved.

The main benefit of ARENA 90 over old ARENA is that it is a completely modular code with support for a modern program language (Fortran 90) and modern compilers. It has been given a suite of additional software tools to process the output data and create final figures based on the written output. Since Gergely Csépány and I have worked separately on this, there is a redundancy in the tools available, which works to confirm the validity of the results. To measure the speed upgrade of the code, I did a suite of tests with the base test case for primary generation with different numbers of particles and time step sizes - the results are in section 5.4.

ITM, the Integrated Tokamak Modelling task force, is an initiative to create a full Tokamak simulation program that connects various more spearheaded projects, like ARENA, through a set of data objects, so called CPO:s (Consistent Physical Objects). This will require more work to be done with input/outputs, which is already initiated to some degree. The current build of ARENA 90 will determine at run time whether the current environment supports IPO:s (basically, looking to see if it is running on the ITM *Gateway* computer cluster), and use them for input/output if that is the case.

Although EFDA integration requires a special design of input/output parameters to conform to the CPO:s, routines for running the tools *ad-hoc* with a previously tested set of parameters are still important to verify the code during development and to be able to run standalone simulations.

Documentation is paramount for projects like this which span multiple generations of students and researchers, but is easily moved last in the list of priorities. There are some requirements, but also structures for keeping good documentation in the EFDA ITM project [6]. ARENA is checked in under `gforge.efda-itm.eu/svn/arena/trunk` and is documented using *doxydocs* which is freely available online. Furthermore the documentation can be generated from comments in the Fortran code, and output into various formats like TeX or HTML, which minimises maintenance. There is also a light weight user's manual in the repository under `trunk/documentation/user-manual` to get you going in the form of a pdf and corresponding TeX file.

To read ARENAs documentation, one can check out the ARENA trunk by doing:

```
svn co http://gforge.efda—itm.eu/svn/arena/trunk arena90
```

in a command window with subversion installed. There should be a compiled doxydocs generated pdf file and a html file under `trunk/documentation/doxydocs/`. Else it can be generated by running

```
doxygen documentation/arena—doxygen.conf
```

from the `trunk` directory with doxydocs installed (do e.g. `apt-get install doxydocs` on Linux to install).

More on the development of the ARENA 90 code can be found in appendix C.

4.10 Parallel version

The notion of running ARENA updates in parallel has been around for some time. Just to be clear on what parallel means in this regard - running ARENA for a range of input parameters, e.g. a range of outer electric fields E/E_D to produce a Dreicer graph (5.1) can be done by running several processes side by side from linear code. Many of the graphs in this thesis have been run on a workstation with multiple cores, which works well since each individual process does not consume a lot of memory.

To get a good resolution of the output however, one might have to use several thousand test particles, and each run will start to take some time. It would be desirable to speed up the actual run-time of a single simulation, and this can be done by updating the tests particles in parallel.

Each particle is updated independently of the others, so it is possible to run these calculations simultaneously, in parallel. If not doing the self-consistent electric field update (which depends on the strength of the runaway current), even updating subsequent time steps could be done in parallel. This is usually not a good idea, except perhaps for code verification purposes, so I will focus on parallelising the Monte Carlo particle updates. Ideally, if the additional time taken to allocate resources on an array of computational units is not too large, having 10 computational units would speed up the program 10 times.

There are some different ways of going about creating a parallel code. Many scientific calculations are done on computer clusters that you log on to remotely over the internet and submit your calculation to. This usually costs money for the university and many researchers sharing a cluster could get in the way of each others' simulations. If the implementation is easy enough, these computers often offer great performance. There are multiple processor cores in most PCs these days, and software is getting more clever at using this power for multitasking all the time. This opens up for the possibility of running a simulation locally, with perfect portability - still speedier than a code that executes in sequence. I could run ARENA on my laptop, and literally have it running without a wire. This requires a different kind of software implementation, usually centred around an API (application programming interface) that offers a way to interact with sub-processing units on a modern CPU without extensive knowledge about the inner workings of it (the runtime drivers).

One such API, or really a framework for an API, is OpenCL, which is becoming increasingly popular. In fact, many graphics chips, primarily responsible for displaying graphics on the computer screen, offer OpenCL implementations letting you harness the power of graphics, usually tens to hundreds of processing cores, to update ARENA test particles.

There is an OpenCL library in the Fortran 90-language available from the Portland Group called *PGI OpenCL*, which is commercial software `www.pgroup.com`. For a freely available alternative, there is `fortrancl`, an open source project at Google Code (`http://code.google.com/p/fortrancl/`) which implements the OpenCL standard, and this is what has been used here. The caveat is that any parallel code must be translated to

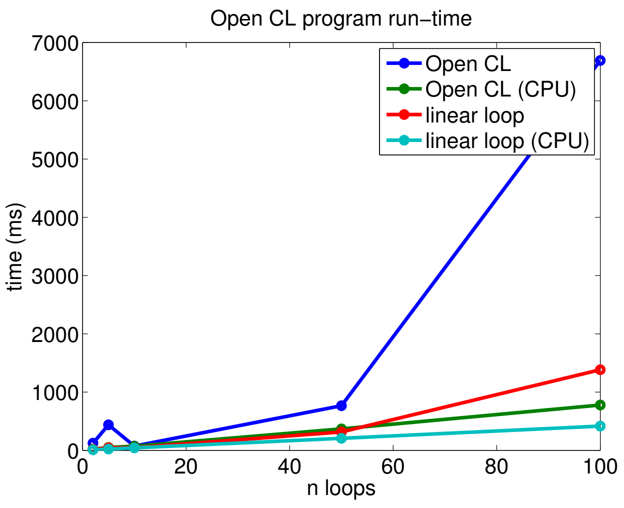


Figure 4.4: A comparison of run times for the collision operator, `mccoll_v1` in an OpenCL framework compared to a linear loop. The initial distributions are loaded from an initial distribution generated by ARENA 90 and calculation of the elliptic integrals ζ_1 and ζ_2 has been done before starting the timer. Unfortunately OpenCL under-performs a linear loop in every number of iterations. There is likely a good reason for this that will have to be researched. Note that CPU time measures the total time that the computational units perform calculations for the program and system procedures combined. A higher CPU time than real time indicates that the process has spawned child processes whose *combined work time* are added together. Thus for this example, real time is the relevant metric.

`C` code. Not a terribly cumbersome task, but difficult to do throughout ARENA while guaranteeing consistent physical results. I have created a test timing suite, translating the *collision operator* to C-code, and timed the calculations against running the original collision operator in Fortran code in sequence. At the end of each run I make sure that the output from the OpenCL program is consistent with what I would get running the old version, thus guaranteeing not breaking the program logic.

The results can be seen in figure 4.4. They indicate for this test, unfortunately that the OpenCL implementation of the collision operator offers poorer performance than the sequential code. This can be due to a number of reasons. To be able to do a calculation in the OpenCL framework, a kernel must be created in software which does necessary allocations of memory and synchronisations on the computation unit (i.e. the graphics chip) that is to perform the calculation. Furthermore, the collision operator is the only piece of code that I have parallelised. It may be that running more of the code in parallel will make a greater improvement in pure calculation time, which would overcome the set-up time in creating the OpenCL kernel.

Regardless the output is consistent with the sequential code and this test has shown that it is quite possible to run ARENA in parallel, on a single computer with maximised portability and standards compliance. Some more details on OpenCL, portability and scalability can be found in Appendix C.

5

Results and Comparison

RESULTS FROM BENCHMARKING AND TESTING OF THE ARENA CODE. This chapter contains the results from a series of runs of the ARENA code along with comparisons with the LUKE code and published numerical and theoretical results. The first section shows the output for the Maxwell test and the Dreicer test explained in section 4.6. Next comes an introduction to the LUKE finite difference code and a series of comparison graphs. Finally there is a comparison of the old ARENA code and the new ARENA 90 and a discussion.

5.1 Benchmarks

The benchmarks published in [2] are plotted first for reference and shown side by side with ARENA output. With the final ARENA 90 code they show good conformance across all of parameter space. The Dreicer curve of the runaway generation rate as a function of changing electric field (normalised to the Dreicer field) E/E_D is plotted in figure 5.1. I have also plotted the Kulsrud tabular values from Ref. [1]. For the lower E/E_D-values in figure 5.1, the conformance is worse due to noisy output data. To verify that theory I tried running ARENA for the lowest input electric field $E/E_D = 0.04$ with $2 \cdot 10000$ particles, and the error indeed goes down. The comparison can be seen in figure 5.2. Plot of the source data for all the data points in the plot are included in section C.

For the Maxwell test, the output shows good performance in maintaining the distribution of particle momenta with no outer electric field for several thousand collision times. It also conforms well with the theoretical parameter $p_{th} = \sqrt{T/m_e c^2} = \{T = 1\text{keV}\} \approx 0.0442$ in equation 4.18. The output is plotted in figure 5.3. The fit does show a slightly less peaked Maxwellian after 10000 iterations at $dt = 10^{-6}$, than after ~ 15 iterations, but there are clearly no runaway electrons with $p > p_c$.

Increasing the temperature used to give issues with an older version of ARENA 90, but are now successful for thousands of collision times up until temperatures of ~ 25keV.

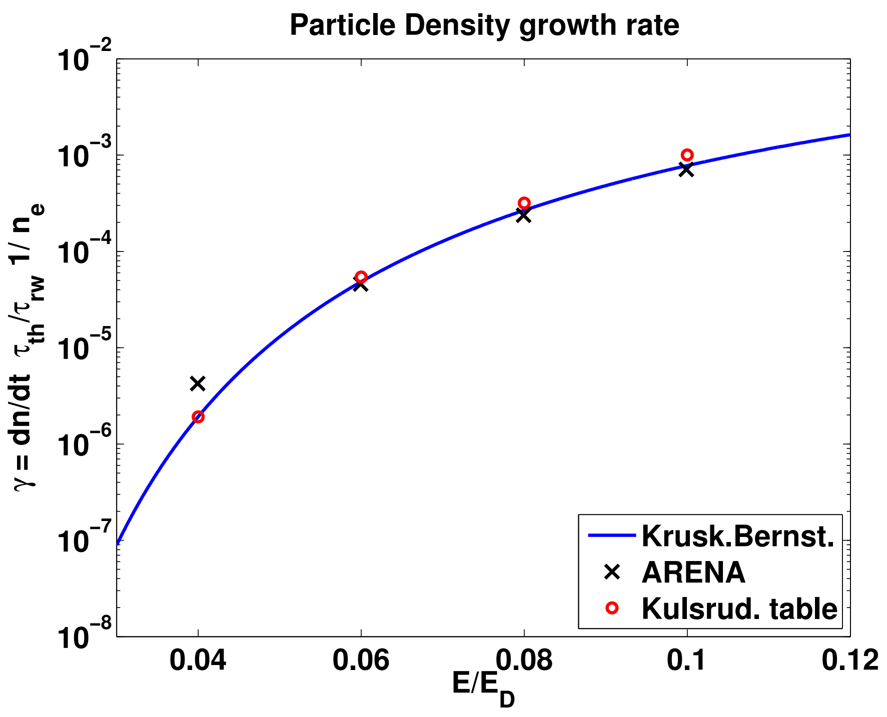


Figure 5.1: The Dreicer generation rate as a function of the applied electric field. The reference is the analytical expression by Kruskal-Bernstein and Kulsrud with parameters like in equation 2.18, and also the table values from numerical simulation in [1]. The spread of the markers in *ARENA output* are from runs with different time steps (see section 4.7.2). The ARENA version used here revision 280 or the `weighting-disabled` branch.

This is plotted in figure 5.4. A comparison of the Maxwellians for different temperatures is in figure 5.5. The issue at high temperatures is plotted in the appendix (figure C.6).

5.2 The LUKE code

For reference I have visited Cadarache in France and decided on a good baseline for tests which is described below. At Cadarache the LUKE finite difference code is used, which is based on a MATLAB container with `.mex` modules doing the time critical computations. It can model electrons as well as fuel- and impurity ions by setting the charge and weight parameters up accordingly. It can also be chained together with ray tracing code C3P0 to create a simulation environment for radio frequency heating experiments. The nature of the code makes it very efficient when it comes to solving the Fokker-Planck equation for primary generation. The drawback is the intrinsic limit on particle energies that can be modelled, which comes from the fact that the distribution is modelled on a grid with finite size. A Monte Carlo code like ARENA has no such limit since each test particle is assigned an individual momentum value p which is a real valued number in the computer.

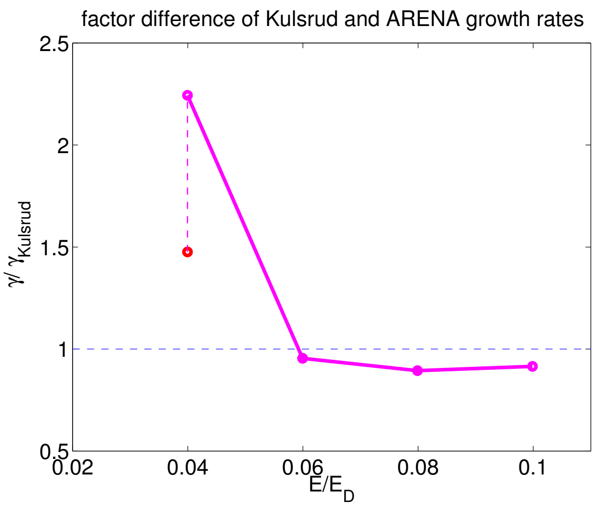
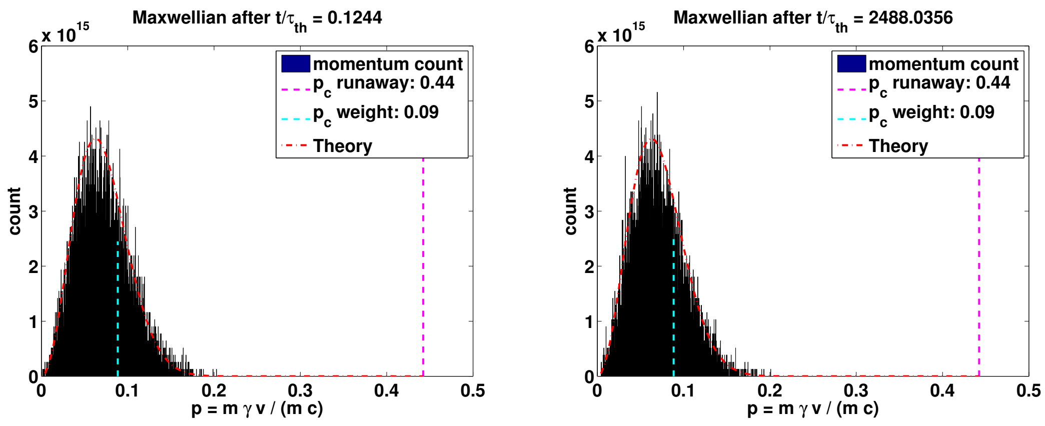


Figure 5.2: The ratio between the Dreicer rate from ARENA and the Kruskal-Bernstein theoretical value (equation 2.18) as a function of outer electric field E/E_D. It is clear that the ratio is close to 1 for all but the first data point. Increasing the number of data points will mitigate this which is illustrated by an additional run with twice the number of test particles, 20000 for $E/E_D = 0.04$.



(a) Initial distribution (b) Distribution at $t_n = 10k$

Figure 5.3: The Maxwellian distribution of particle momenta at simulation start and after 10,000 iterations. Here $p_c = 10.0 \cdot p_{th}$, $T = 1.0\text{keV}, \text{dt} = 10^{-6}, \text{n}_{\text{particles}} = 6000$. The Maxwellian is preserved very well for an extended simulation time, which indicates that the code does not produce runaway electrons with no electric field present, which confirms the physicality of the other operators in the simulation. The *theory* curve is plotted with $p_{th}^2 = 0.0016$ in both curves to give a visual means of comparison.

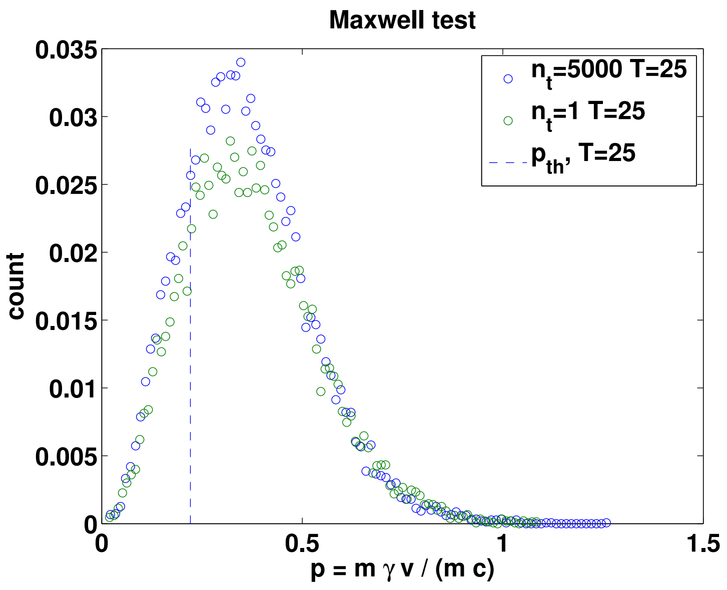


Figure 5.4: The Maxwell test for a higher temperature: 25keV. The test still passes with good conformance. The Maxwellians at 1 iteration ($n_t = 1$) and 25000 iterations ($n_t = 25000$) look all but identical.

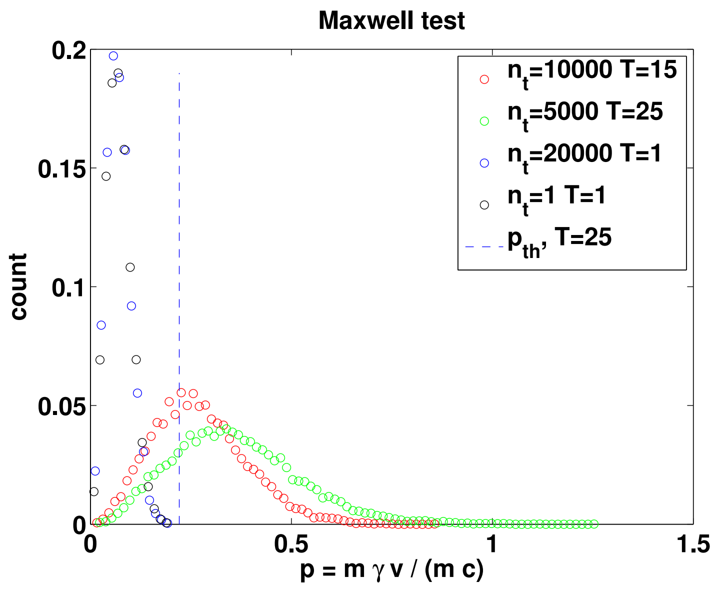


Figure 5.5: A comparison of Maxwell distributions at different temperatures. When the temperature goes up, the distribution is spread out, and the mean (expected) value increases as p_{th} increases.

Thus, ARENA should be much better suited to calculate secondary generation effects. LUKE creates the primary runaway generation outputs by calculating particles which cross a certain limit momentum p_c. ARENA does this too, but when particles increase momentum even further, they are lost from the calculation in LUKE, whereas ARENA can keep them and use them for the secondary generation calculation.

LUKE counts runaway electrons by considering the flux of particles through the limiting surface of the simulation in phase space. Letting the runaway *loss rate* be Γ_R, the flux of particles in the distribution function at the simulation domain limit $p = p_{max}$;

$$\Gamma_R(\phi, \theta) = \int \int_{\Upsilon} \mathbf{S}_p(\psi, p\xi) \cdot d\mathbf{S}, \tag{5.1}$$

$d\mathbf{S}(p) = p^2 d\xi d\phi \hat{p}$, symmetry giving $\int d\phi = 2\pi \Rightarrow$

$$\Gamma_R(\psi, \theta) = 2\pi p_c^2 \int_{-1}^{+1} S_p(\psi, p_c, \xi) d\xi, \tag{5.2}$$

with p_c the critical momentum: the limit where electrons run away. In LUKE, this value is chosen to be sufficiently high that collision drag cannot counteract the accelerating force $e\mathbf{E}$ which is dependent on the collisions model. Once a particle is across the limit p_c, it is counted as a runaway electron. It will then gain energy and eventually leave the simulation parameter space $p \to p_{max}$.

Thus counting the flux through an arbitrary flux surface in momentum-space which is above p_c, will correspond to counting the number of runaways, making p_c a free parameter in the code. A source term of thermal electrons balances the loss of runaways to preserve particle count, similar to what is done in ARENA [16].

The difference in the definition of *when* to count runaways could lead to slight differences in the shape of the *fraction of runaway*-curves (figure 5.6,5.7). Next, the runaway rates are derived from numerical differentiation, which are the values that are then used for the Dreicer test in figure 5.1. With the same physical boundary conditions, although the curves may be different, these rates should align well.

LUKE is a finite element code whereas ARENA is a Monte Carlo code. They have fundamentally different approaches to solving the time evolution of the bounce averaged distribution function. Due to this, they have different benefits and limitations. When solving for primary generation, LUKE greatly outperforms ARENA. A run of 1000 collision times ($t_{tot}/\tau_{th} = 10^3 \Rightarrow t_{tot} = 8\,\mathrm{ms}$ if $\tau_{th} \sim 8 \cdot 10^{-6}$) in LUKE takes only a few seconds, while the old ARENA can take several days. ARENA 90 is significantly faster but a run can still take hours, thus LUKE beats it by several orders of magnitude. However, the finite element approach used by LUKE has an intrinsic limit in the maximum momentum (energy) of the electrons. Picturing a grid that stretches across momentum-space LUKE will slow down if the grid is made too large, and resolution will suffer if it is made too sparse. ARENA does not slow down with the size of the problem at all. Each particle is assigned an arbitrary value for each parameter (p being the one that grows to large values) with no maximum.

The performance and resolution is instead dependent on the number of test particles on which Monte Carlo operators are applied. If too few particles are used, the statistics will not be good enough to get reliable results or a detailed enough smoothed *distribution function*. Since there is nothing that prevents updating all the test particles in parallel, that is a possible solution to speed up ARENA. In fact, there is a loop of calls to each operator in the Fokker-Planck equation:

$$\frac{\partial f}{\partial t} = \langle L_E(f)\rangle + \langle C(f)\rangle + \langle L_{synch}\rangle + \langle L_{s.t.}(f)\rangle + \langle S\rangle + \langle l_B\rangle, \tag{5.3}$$

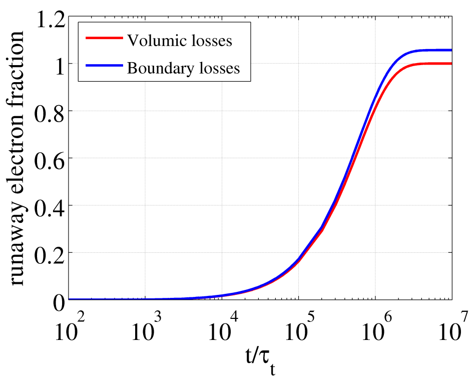
all independent of one another in each time step that could be parallelised which was investigated in section 4.10.

It is fair to ask whether simply increasing the simulation space and resolution of LUKE would work just as well, or even better than optimising ARENA. At this point, it is not clear that this would be enough to simulate the physical reality of avalanching, since the seed of primary runaways reach very high momentum values. The bold solution could be a merged code, where LUKE would handle the main simulation, creating Monte Carlo test particles for when electron values leave LUKE's phase space, only to be reincorporated in the finite element code if and when they loose enough energy. With LUKE being a MATLAB program, this could be realised using the `.mex` module files. The fact that ARENAs collision operator is modular enough to be ported into a parallel code shows that this is very much a possibility, although great care would have to be taken to make sure that the physicality of the model is contained throughout the simulation.

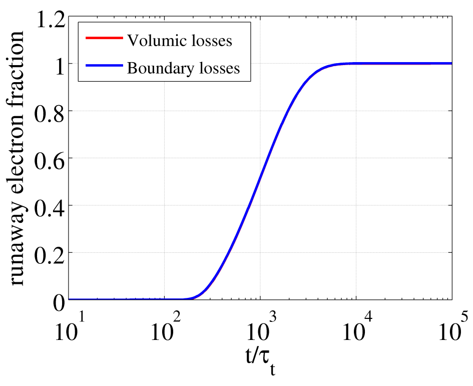
In figure 5.6 you can see the time-evolution of the runaway fraction from LUKE. Figure 5.7 shows the numerical differentiation of the same. Note that the generation rate takes some time to reach a saturated runaway growth rate, which corresponds to the Kulsrud growth rate. It then falls when the bulk of the electrons have already become runaways and there are no more electrons to simulate. The two curves show the *volumic*- and *boundary*-losses, which together show the numerical accuracy of LUKE. Since the fraction of runaways is plotted when applying a constant outer electric field, the fraction of runaways will tend towards unity, but cannot be > 1.

5.3 Primary generation output

It was important to find a good reference test for a detailed benchmark comparison with the LUKE code. The Dreicer- and Maxwell benchmarks (section 5.1) show a good general performance of the code, but it hides information on actual code output. For example, the runaway generation rate γ is calculated from a numerical differentiation of the fraction of runaway electrons, which ARENA writes to file (see section C for details). This output can be directly compared to LUKE output, which is done below. To be able to directly compare to the Kulsrud rates, values from corresponding to the published tabular values $E/E_D = 0.1$ and $E/E_D = 0.04$ were used. To be able to work in a parameter space where ARENA is performing well, the temperature was set to $T = 1$ keV, from which the ARENA input value $E/E_c = E/E_D$ is found.

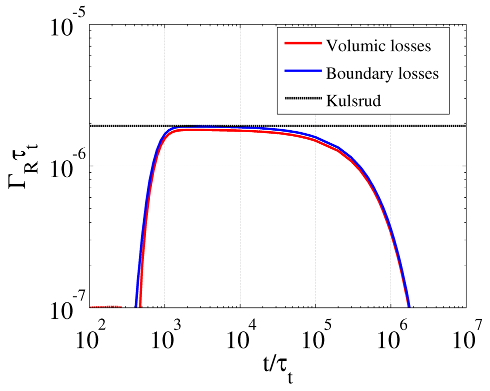


(a) $E/E_D = 0.04$

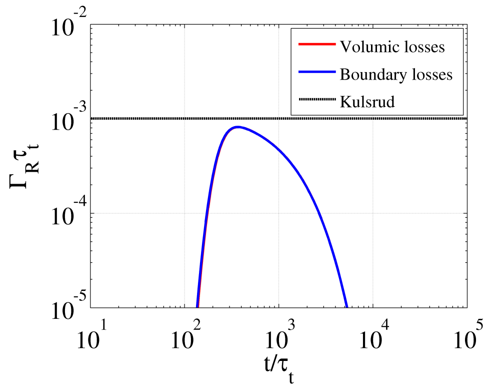


(b) $E/E_D = 0.1$

Figure 5.6: LUKE code output of the fraction of runaway electrons as a function of time normalised to thermal collision times (here $\tau_{th} = 8.03 \cdot 10^{-6}$). The growth is slower at the start before the rate has saturated, it then pans out near $t/\tau_{th} = 2 \cdot 10^6$ when all electrons have become runaways. This happens because the applied electric field is held constant. The *Volumic*- and *Boundary* losses indicate two ways for the code to measure the fraction of runaways and qualitatively shows the accuracy of the LUKE code at the chosen resolution. Apparently, the measurement is more exact for a higher input **E**-field

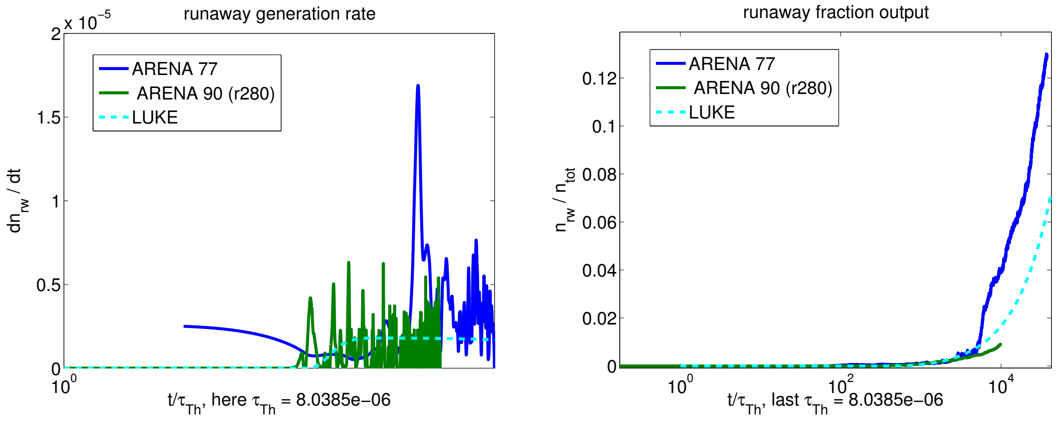


(a) $E/E_D = 0.04$



(b) $E/E_D = 0.1$

Figure 5.7: Rate of change of the fraction of runaways as a function of time from LUKE. Note the initial build-up time before the saturated rate is achieved, and then the fall in the end of the simulation. In the saturated region there is good conformance with the Kulsrud rate. The logarithmic time scale is normalised to thermal collision times τ_{th}.



(a) runaway rate: $d(n_r/n_{tot})/dt$ (b) Fraction of runaways

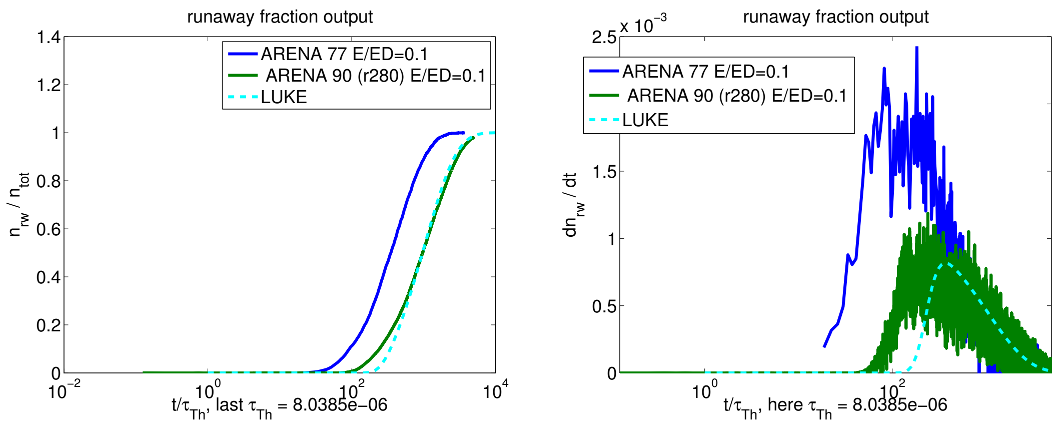
Figure 5.8: Runaway fraction n_r/n_{tot} and the runaway *rate* γ as a function of time normalised to thermal collision time t/τ_{th} of ARENA 90, old ARENA and LUKE at $E/E_D = 0.04$. ARENA shows a similar trend to LUKE, but the output of the rate is quite noisy and it is hard to find a good saturated generation rate. Since ARENA converges slowly, setting the sub-timestep $dt_s \sim \tau_{th}$, it takes a long time to simulate long time series. To get 10^5 collision times would require as many time steps, taking very long in real time. Confirming that increasing the number of particles for this low value of E/E_D (figure 5.2) will suffice to say that the code is valid also for this low **E**-field.

The two main test cases are displayed in figure 5.8 and 5.9. From the output you can see that ARENA is more stable for the higher electric field, and also the simulations are faster since the runaway population will build up quicker. The $E/E_D = 0.04$ case displays a sudden surge in the runaway generation for the old ARENA code when weighting was used. Since it is not present with the new code, it is not likely that this is a difference between the Monte Carlo and finite element models, but rather a code error related to ARENA's bad weighting (section 4.7.3).

Compared to the LUKE output (dashed line), the graphs have similar shapes for the higher **E**-field case, although there is a linear translation in time, which is, however, made smaller for ARENA 90. For the lower E-field case $E/E_D = 0.04$, the curves align well up until 6000 thermal collision times. The likely explanation for the difference in when the generation starts is the different definitions of the cutoff momentum p_c, see section 5.5.

5.4 The new ARENA 90

ARENA was ported from Fortran 77 to Fortran 90 by a joint effort led by Gergely Csépány in 2011. The goal was to create a fully modular, documented and managed code base that could be more easily transitioned between researchers. The new code employs e.g. precompiled libraries for speed improvements and is much easier to work



(a) runaway rate: $d(n_r/n_{tot})/dt$ (b) Fraction of runaways

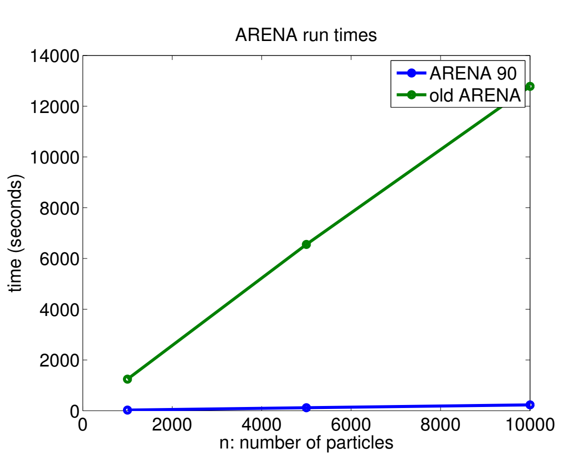
Figure 5.9: This figure shows runaway fraction n_r/n_{tot} and the runaway *rate* γ as a function of time normalised to thermal collision time t/τ_{th} of ARENA 90, ARENA 77 and LUKE at $E/E_D = 0.1$. The saturation of runaways happen faster at the higher electric field which is expected. The output is a lot more stable than for the $E/E_D = 0.04$ curve too, which shows that this is a parameter region where ARENA performs well.

with because of the more modern syntax of Fortran 90.

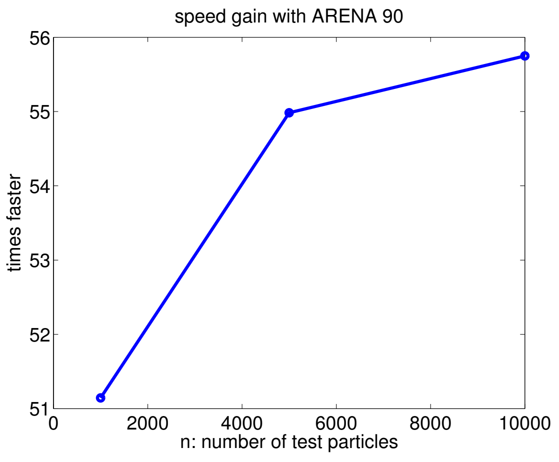
A number of ARENA revisions have been around, producing results of varying accuracy. I have tried to include revision numbers wherever possible, mainly for ARENA 90. The old ARENA, included for reference, is the so called *Merlin*-version. It was handed down from the previous research team, who had used it for publications. It existed in ITM:s SVN under `svn/arena/tags/merljin-version` on March 13 (revision 77). However it is questionable whether the code base that we started out with actually was the original one, given the misalignment LUKE in figure 5.9, which was later fixed.

Most runs with ARENA 90 presented here use a version without weighting from May 23rd (revision 280). Removing the weighting part of the code, finally solved withstanding issues with both the Maxwell test and Dreicer generation test leading to the belief that that part of the code is broken. In actuality, what was done, was to remove the `sortmc()` subroutine, and weighting initiation in the beginning of the program (section 4.2). This fixed long withstanding issues with too early runaway generation regardless of the chosen value of p_c, the cut off-momentum, and the factor 2 higher runaway generation rate than the Kulsrud value. The current best version is the branch `weighting-removed` in `http://gforge.efda-itm.eu/svn/arena/branches/weighting-removed`, revision 280. The LUKE version used is the latest official distribution, version 1.9.1

ARENA 90 has undergone optimisation for a significant speed gain. Running a basic test for $5000 \cdot dt = 5$ milliseconds with varying number of particles, it is clear that the new optimised code gives a speed gain of over 50 times the original ARENA runtime. This is a significant improvement achieved even before getting to parallelising the computation. A graph of the (real) times for the different runs can be seen in figure 5.10 along with



(a) ARENA runtimes



(b) t_{old}/t_{new}

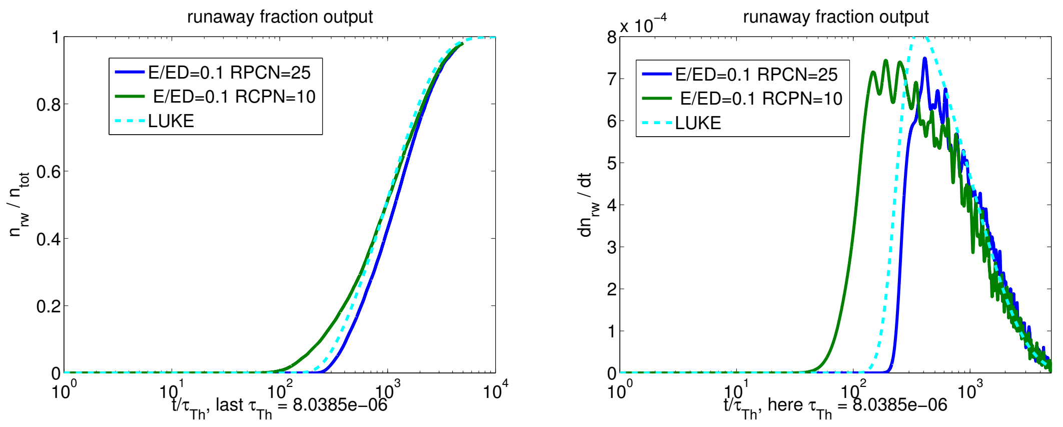
Figure 5.10: The time it takes to run an entire simulation to find the fraction of runaways as a function of the number of test particles with the new ARENA 90 and the old ARENA codes is plotted in the left hand side picture. It is clear that the new code is a lot faster. Here $E/E_D = 0.1956, dt = 1 \cdot 10^{-6}, n_{steps} = 5000$. The fraction t_{old}/t_{new} is plotted in the right hand side picture. The graph shows that the new code is > 50 times faster than the old code for primary generation. A significant improvement with only sequential code improvements.

the speed gain $v_{new}/v_{old} = t_{old}/t_{new}$.

5.5 Parameter scans

This section contains some parameter scans to show the breadth of ARENA's working parameter space. First, does the generation rate depend on the definition of what to count as a runaway electron? I.e. the value $p_c = \texttt{RCPN} \cdot p_{th}$. The higher value for $\texttt{RCPN} = 10 \rightarrow 25$ shows that that is not the case (figure 5.11).

We tried to change the time step taken dt, to be able to speed up the ARENA simulation. With a longer time step, the total simulation time should be able to be extended without increasing the number of iterations. It is known that ARENA modifies the time step to a smaller one if it is deemed to be too large to yield a reasonable continuity between each iteration. This is done in two ways (section 4.7.2) and becomes apparent when timing the simulations. I did not record any quantitative measurement, but increasing the time step by a factor of 10 did almost no difference for the total simulation time, because the sub-time step was set according to equation 4.24. As such, staying with $dt = 1 \cdot 10^{-6}$ seemed an appropriate choice. The time step used is indicated in most figures, although it should not impact the final result because of how the sub-time step is calculated. A test is plotted in figure 5.12, where the time step was changed between 5 different values, keeping all other parameters the same as for the base test



(a) Fraction of runaways (b) dn_r/dt

Figure 5.11: Two outputs with different values of the `RCPN` input parameter, the cutoff momentum $p_c =$ `RCPN`$\cdot p_{th}$. The output shows a slight drift to the right with higher `RCPN`, but no major change in runaway rate, which makes the case that this parameter is not critical for the output of generation rate.

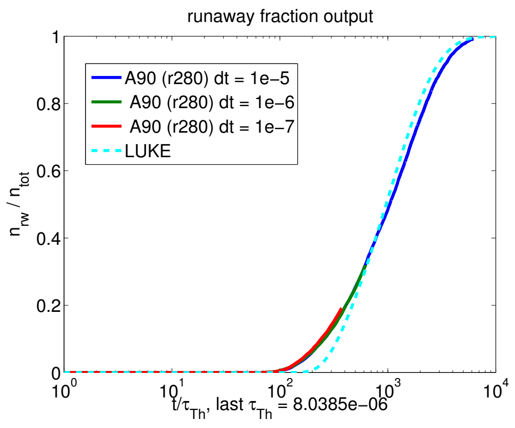
case (figure 5.9).

We also tried changing the number of test particles. Supposedly a large number $n_p \sim$ 10000 particles are needed. Initial test results showed that the overall trend could be qualitatively measured with as little as 1000 particles. However, plotting the generation rates, it becomes cleat that there is a greater dependence on particle resolution. This is also apparent from the Dreicer generation test discrepancy in figure 5.2

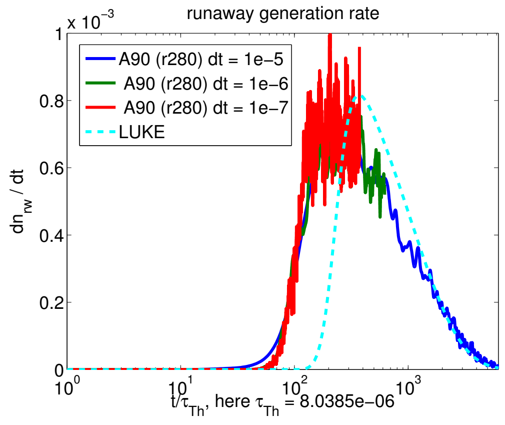
5.6 Discussion

The outputs of the new ARENA confirm well with published results, and the parameter sweeps show that ARENA gives stable output in a range of parameters. The claim that the definition of what to count as a runaway electron, i.e. what to choose for p_c does not matter for the generation rate has been verified.

Whether a large number of test particles are needed has been going back and forth, and apparently it depends on the magnitude of the outer electric field. To reliably establish the the generation rate, which has been deemed the important measure, 10000 particles work well for $E/E_D \geq 0.06$. However for $E/E_D = 0.04$ still 20000 particles, (figure 5.2) gives too noisy output for the difference between ARENA's simulation and the Kruskal-Bernstein curve to be 0. It has been suggested that an automatic test framework for ARENA run sanity tests with only a few hundred particles. To get a good sense of the validity of the generation rate, this is too little to get anything but noise in the calculations. It could possibly be used for sanity testing with a test seed of non-random numbers r in equation 4.3, etc. to make sure there is no numerical deviation - but that has yet to be seen.

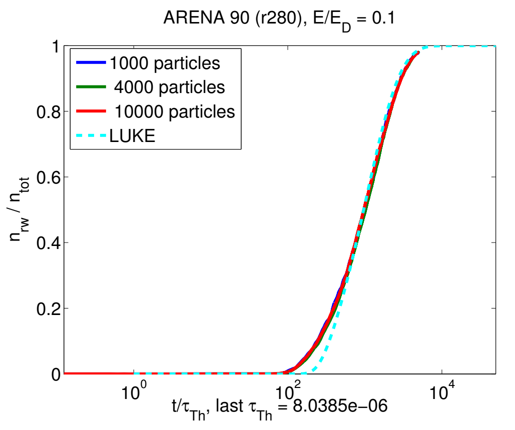


(a) Fraction runaways

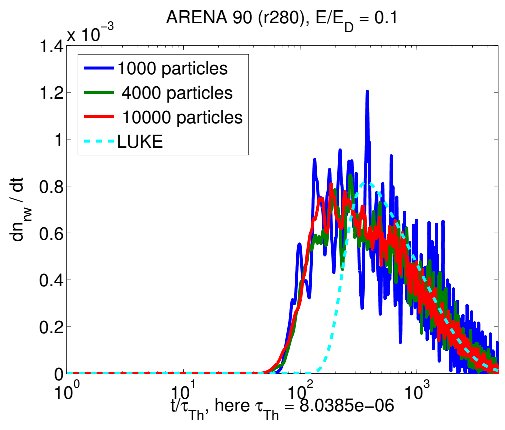


(b) generation rate

Figure 5.12: Plots of ARENA output when varying the time step dt, which is expected not to impact the simulation at all. The change is not visible. Interestingly, the output seems to be noisier with a smaller time step. This could have to do with the fact that ARENA rescales the time step and actually runs *more* iterations with a higher dt, although the output happens only at the input time step intervals. Or it could be related to the time step bug (section 4.7.2). All runs use the same base parameters, $T = 1.0keV, n_e = 2 \cdot 10^19, E/E_D = 0.1$, also note that the shorter time steps run a shorter total simulation time.



(a) Low Particle count compare



(b) num.diff

Figure 5.13: A comparison of ARENA runs with different numbers of particles. It is clear that the number of particles, which greatly impact total simulation time, do not significantly change the overall trend of the curve. It does however, affect the generation rate, which is seen to the right. The dashed line is the LUKE curve.

ARENA with the Maxwell test shows an instability for high plasma temperatures (thermal energies). In theory, with the external electric field set to zero, there should be no change to the Maxwellian distribution. Instead, for very large energies, there is a splitting of the Maxwell curve into two. It is possible that the reason for this is tied in with the two-part collision operator. We were unable to confirm this by using the ANTS operator for reasons described in section 4.4.

The current model is valid for background plasma temperatures below $\sim$ 15 keV. This temperature is high enough for studying the runaway phenomena, but not for all applications involving of actual fusion temperatures in large devices like ITER. Hopefully, this limitation could be lifted if the collision operator from ANTS could be implemented, which is valid for all energies (equation B.24) [18].

6

Conclusions

Looking at the final output, ARENA does show good conformance with published an theoretical results over a large range of parameter values. The benchmarks of a code like this are mainly in place to gain confidence to be able to run other simulated experiments. For such simulations to be meaningful, the layout and structure of the code should be firmly based in theory. The theory surrounding Monte Carlo simulations as such is broad ranging and can be verified in small experiments and simulations like the one in section 4.3. The individual Monte Carlo operators used in updating the model parameters of ARENA are documented in less detail however, which has made this work heavy on understanding and reaffirming calculations and sections of the code. In a sense, ARENA has now gone through some post-finish thorough unit testing.

The stable verification of the two benchmarks across parameter space nevertheless shows that ARENA is able to produce reliable results in the parameter space researched. Extensive reverse engineering and testing done throughout this project shows that these kinds of projects can be narrowed down to bare essentials and bear fruit in useful output, albeit here in a lesser extent than we first envisioned.

One motivation for ARENA compared to other simulation software is the power of Monte Carlo methods for modelling very high momentum particles. This is mainly important for secondary runaway electron generation which we have not been able to implement in ARENA 90 as of yet. Also, redundancy in research is always important, and the fact that ARENA and LUKE show good conformance between each other show that the two methods of iterating the Fokker Planck equation are both viable, producing results that agree with theory.

The test with parallel programming with OpenCL confirms the notion that ARENA can be parallelised without too much effort. By confirming that the output stays constant running ARENAs collision operator with the OpenCL standard in a benchmarking environment, it was shown that it is possible to pick out single modules of the new ARENA code to create separate programs using commonly available API:s and techniques, which is important for any derivative work. The modularity of the new code should be of great help if any attempt was to be made to implement parts of ARENA into LUKE, in order to improve that code to better handle high energy electrons.

Today, ARENA 90 is a much more modern, easily workable, logical and efficient code base which has been successfully made to conform with the old ARENA outputs and published references. Additionally, it has been very beneficial for me to get a good look at the breadth of fusion research and the struggles associated with it. Fusion is a broad field of research that relies on breakthroughs in many subject areas in order to finalise the design for a power plant of the future. To be able to design for the complex magnet coil structure of the Wendelstein 7-X stellarator, numerical optimisation was crucial, and that is just one example. Computer simulation is penetrating society as a tool for anything from visual effects in entertainment to instant messaging, but plays a critical role in research and development.

Bibliography

[1] R. M. Kulsrud, Y.-C. Sun, N. K. Winsor, and H. A. Fallon. Runaway electrons in a plasma. *Physical Review Letters*, 31:690–693, Sep 1973. doi: 10.1103/PhysRevLett.31.690. URL `http://link.aps.org/doi/10.1103/PhysRevLett.31.690`.

[2] L.-G. Eriksson and P. Helander. Simulation of runaway electrons during tokamak disruptions. *Computer Physics Communications*, 154(3):175 – 196, 2003. ISSN 0010-4655. doi: 10.1016/S0010-4655(03)00293-5. URL `http://www.sciencedirect.com/science/article/pii/S0010465503002935`.

[3] T. Fülöp, G. Papp, and I. Pusztai. Fusion energy lecture notes. January 2012.

[4] Wikipedia. Nuclear binding energy, January 2012. URL `http://en.wikipedia.org/wiki/Nuclear_binding_energy`.

[5] P. Helander, L.-G. Eriksson, and F. Andersson. Runaway acceleration during magnetic reconnection in tokamaks. *Plasma Physics and Controlled Fusion*, 44(12B): B247, 2002. URL `http://stacks.iop.org/0741-3335/44/i=12B/a=318`.

[6] EFDA/ITM web site, 2011. URL `http://www.efda.org/`.

[7] January 2012. URL `http://portal2.efda-itm.eu/itm/portal/`.

[8] J. Decker and Y. Peysson. Orbit-averaged guiding-center fokker-planck operator for numerical applications. *Physics of Plasmas*, 17(11):112513–112513–12, 2010. ISSN 10897674. doi: 10.1063/1.3519514.

[9] Wikipedia. Fokker planck equation, January 2012. URL `http://en.wikipedia.org/wiki/Fokker-Planck_equation`.

[10] C. F. F. Karney and N. J. Fisch. Efficiency of current drive by fast waves. *Physics of Fluids*, 28:116–126, January 1985.

[11] M. Rosenbluth and S. Putvinski. Theory for avalanche of runaway electrons in tokamaks. *Nuclear Fusion*, 37(10):1355, 1997. URL `http://stacks.iop.org/0029-5515/37/i=10/a=I03`.

[12] P. Helander and D. J. Sigmar. *Collisional Transport in Magnetized Plasmas.* Cambridge Univ Press, The Edinburth Building, Cambridge, CB2 2RU, UK, 1 edition, 2002.

[13] G. Csépány. Kinetic simulation of runaway electrons in tokamaks. Student Research Paper, November 2011.

[14] G. Wahnström. Monte Carlo methods. Lecture notes for Computational Physics course, 2011.

[15] Wikipedia. Ising model. URL `http://en.wikipedia.org/wiki/Ising_model`.

[16] J. Decker and Y. Peysson. LUKE: a fast numerical solver for the 3-d relativistic bounce-averaged electron drift kinetic equation. Technical documentation, Association EURATOM-CEA sur la Fusion, April 12 2007 2007.

[17] L.-G. Eriksson. Monte Carlo operators for orbit-averaged Fokker-Planck equations. *Physics of Plasmas*, (308), 1994.

[18] G. Papp, M. Drevlak, T. Fülöp, and P. Helander. Runaway electron drift orbits in magnetostatic perturbed fields. *Nuclear Fusion*, 51(4):043004, 2011. URL `http://stacks.iop.org/0029-5515/51/i=4/a=043004`.

[19] Wikipedia. Langevin equation, January 2012. URL `http://en.wikipedia.org/wiki/Langevin_equation`.

A

Definitions and Explanations

Basic terminology and definitions. This appendix contains explanations and definitions that make it easier to follow the main text. Although most relevant parameters should be presented in a straight-forward manner, some basic concepts get an additional mention here.

A.1 Toroidal coordinates

Since the tokamak has a torus-like shape with a circular or close to elliptic cross-section, it is natural to think about the spatial coordinates in a toroidal geometry. The basic coordinates are the radial position r, the toroidal angle ϕ and the poloidal angle θ, see figure A.1

To completely describe a point's coordinates in space, the major radius of the torus R must also be known. The full cartesian coordinate transformation then becomes:

$$\begin{cases} x = R\sin\phi + r\cos\theta \\ y = R\cos\theta \\ z = r\sin\phi, \end{cases} \tag{A.1}$$

or for space-polar coordinates:

$$\begin{cases} R_p = R + r \\ \phi = \phi \\ \theta_p = \arcsin\dfrac{-\sin\theta}{R/r+1}, \end{cases} \tag{A.2}$$

An illustration of the toroidal geometry and the implication of adding a *poloidal* and a *toroidal* field component can be seen in figure A.1

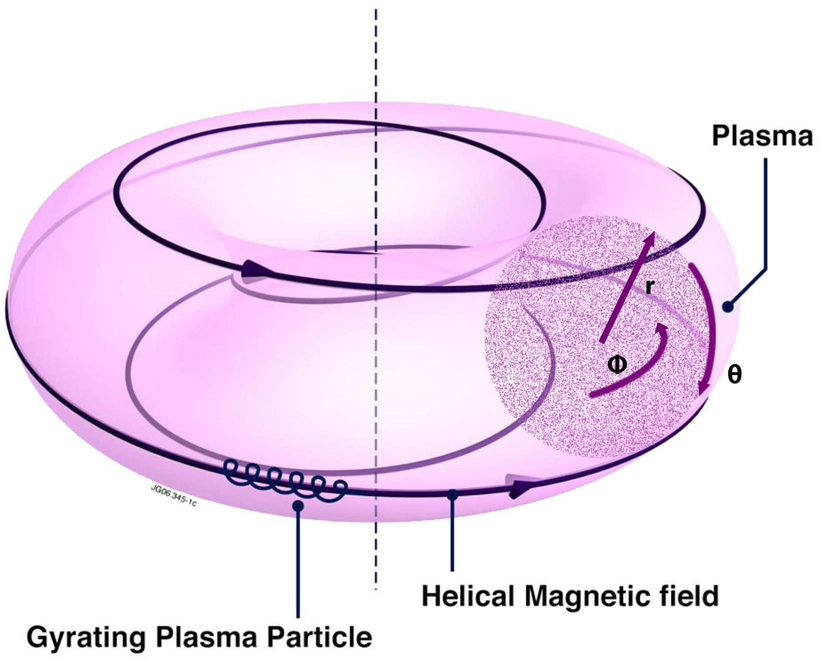


Figure A.1: This is a schematic picture of the spatial coordinates in toroidal geometry. r is the radial variable, ϕ the *toroidal* angle and θ the *poloidal* angle. You can also qualitatively see the twist of magnetic field lines going around the torus shape.

There is a number of ways that the plasma coordinates above are rewritten down in various equations in this paper and it can be a bit difficult to keep track of. In ARENA, calculations are made with normalised variables, since it can simplify the calculations to not keep track of dimension and also because it helps mitigate problems with computer precision. It is fair to say that convention also plays a role in this. The main normalised variables are listed in table: A.1.

The tokamak has a toroidal geometry, and in a simplification ARENA uses a circular torus cross section to model the plasma. Here, the phase space variables are $(r, \theta, \phi, \dot{r}, \dot{\theta}, \dot{\phi})$ and time. The torus volume occurs e.g. when calculating the number of electrons in the plasma from the input *plasma density* variable n_c. Total plasma density $n_e = V_{torus} \cdot n_c = V_{torus} n_c/(n_b(1+\gamma_n)) \cdot [1-(1-n_b)^{\gamma_n}], n_c$: input parameter;

$$V_{torus} = (\pi r^2)2\pi R = 2Rr^2\pi^2 \tag{A.3}$$

basically a cross section area times the circumference of the centre of the big circle, with r the minor radius, R the major radius.

The *pitch angle* θ_p is the angle between the direction of the plasma momentum $\mathbf{p} = m_a\mathbf{v}$ which is *parallel* to the magnetic field $\mathbf{B}$, and the total momentum: see figure

Table A.1: Parameter definitions for reference.The subscript $_a$ is a species of particle. Since the particle of concern for most calculations in this paper is the electron, I will frequently write for $_a$: $_e$

param	expression	explanation
r	r	radial coordinate in a toroidal geometry (sect. A.1) this can be the *minor radius* of a flux surface that the particle in question sits on
R	R	major radius of the tokamak.
ϵ	$\frac{r}{R}$	inverse aspect radio used for simulating physical space in ARENA
Z	Z_{eff}	effective plasma charge measures the purity of the plasma. $Z_{eff} > 1$ is a plasma with impurities (higher atom number) which change the plasma properties
ρ_L	ρ_L	Larmor radius of a particle's gyration along a magnetic field line
ξ	$v_{\|\|}/v = \cos\theta_p$	the cosine of the pitch angle θ_p, see figure A.1
p	$\gamma v/c = v/\sqrt{c^2 - v^2}$	normalised momentum. The third simulation variable for ARENA
λ	$p_\perp^2/(p_2 b(\theta))$	pitch angle variable, describing the particle toroidal alignment. $\lambda > 1$ indicates a trapped particle
$b(\theta)$	$B(\theta)/B_{max}$	normalised magnetic field along the magnetic field line

A.2. The pitch angle is implicitly referenced through the model parameter λ; the cosine of this angle is the fraction of the parallel momentum $\cos\theta = p_{||}/p$.

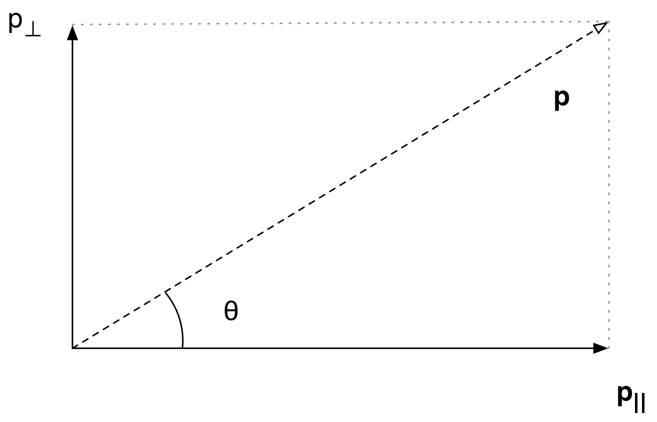


Figure A.2: The angle between the particle velocity vector $\mathbf{v} = \mathbf{p}/m_a$ and the magnetic field is non-zero due to the gyrating motion that a particle makes around a magnetic field line when it enters the field with non-zero speed. The radius is given by balancing the centripetal force and the magnetic force: $mv^2/r_L = qv_\perp B$.

B

Physics in Depth

Derivations of some relevant concepts and equations This section contains definitions and derivations of relevant physics intended to give context to the main text. It expands on appendix A and is more about in depth explanation than understanding. The final appendix C talks about programming suites and numerical solutions.

B.1 The Langevin equation

The Langevin equation is a stochastic differential equation that describes the time evolution of a subset of degrees of freedom which are typically macroscopic variables changing slowly compared to the microscopic variables of the system. They are considered as fluctuations, making the equation stochastic. The Fokker Planck equation (2.2) can be written as an equivalent Langevin equation [2], which is used to derive the form of the Monte Carlo operators used in time stepping.

There is a formal way to derive a generic Langevin equation from classical mechanics. If $A = \{A_i\}$ denotes the slow variables, the generic Langevin equation is [19]:

$$\frac{dA_i}{dt} = k_B T \sum_j [A_i, A_j] \frac{dH}{dA_j} - \sum_j \lambda_{i,j}(A) \frac{dH}{dA_j} + \sum_j \frac{d\lambda_{i,j}(A)}{dA_j} + \eta_i(t), \tag{B.1}$$

where η_i is the fluctuating force with a Gaussian probability distribution and a correlation function:

$$\langle \eta_i(t), \eta_j(t') \rangle = 2\lambda_{i,j}(A)\delta(t - t'). \tag{B.2}$$

A special case is the Langevin description of Brownian motion:

$$m\frac{d^2\mathbf{x}}{dt^2} = -\lambda\frac{d^2\mathbf{x}}{dt^2} + \eta(t), \tag{B.3}$$

with the force $\eta(t)$ with a Gaussian probability distribution having a correlation function:

$$\langle \eta_i(t), \eta_j(t') \rangle = 2\lambda k_B T \delta_{i,j} \delta(t-t'), \tag{B.4}$$

k_B is Boltzmann's constant, T is the temperature and the Dirac $\delta(\Delta t)$ is an approximation of the actual finite correlation time of the random force which depends on the collision time of the molecules.

B.2 ARENA's Monte Carlo operators

For ARENAs solution of equation 2.2 the Langevin operators $\{A_i\}$ are derived from the following (difference equation) expression:

$$\Delta I^i = (\dot{I}^i_E + \dot{I}^i_c + \dot{I}^i_c + \dot{I}^i_c)\Delta t + (A^{i,j}_c + A^{i,j}_{s.t.})\xi^j \sqrt{\Delta t}, \tag{B.5}$$

where the Δ denotes a step in time. The dotted variables are expectation values of the phase-space random variables, $A^{i,j}_\zeta$ is a variance and ^{i} is a particle index. For the momentum, p, this is equation 4.3. The subscripts for the different terms match the ones in equation 4.1. ξ is a random variable, with a normal distribution with unit variance, which introduces stochasticity through the variance of the collision term $\langle C(f) \rangle$ and the radial transport term $\langle L_{s.t.}(f) \rangle$ of equation 4.1.

The derivation of collision operator terms was done in some detail in [17]. The general expression for any term in the right hand side of equation 4.1 is found to be:

$$C(f) = \frac{1}{\sqrt{g}} \frac{\partial}{\partial x^i} (\sqrt{g}\, \Gamma^i), \tag{B.6}$$

where Γ^i are contravariantly transformed components of C^i; $\Gamma^i = C^j \partial x^i / \partial z^j$ and $\sqrt{g}$ is the transformation scaling Jacobi matrix. In the case of runaway electrons, the general form of either right hand side term in equation 4.1, including the bounce-averaged collision operator term ($\langle C(f) \rangle$), becomes:

$$\langle L \rangle = \frac{1}{\sqrt{g}} \frac{\partial}{\partial I^i} \left[\sqrt{g} \left(a^i + b^{ij} \frac{\partial f}{\partial I^j} \right) \right], \tag{B.7}$$

and

$$\sqrt{g} = R_{min} r p^2 \oint \frac{1}{2\pi \sqrt{1 - \lambda b(\theta)}}. \tag{B.8}$$

At this point, the collision operator can be written down in terms of the mean and covariances of each of the Langevin phase space variables $\mathbf{I} = (p,\lambda,r)$ which are given in the following set of equations:

$$\begin{aligned} \dot{I^i} &= \int I^i \langle L(f(t=t_0)) \rangle \sqrt{g} d^3 I = -a^i + \frac{1}{\sqrt{g}} \frac{\partial}{\partial I^j} (\sqrt{g} b^{ij}), \\ \dot{s^{ij}} &= \int (I - \overline{I}^i)(I - \overline{I}^j) \langle L(f(t=t_0)) \rangle \sqrt{g} d^3 I = A^{ik} A^{jk} = 2b^{ij}. \end{aligned} \tag{B.9}$$

Here, the dotted variables are time derivatives and when there are no cross terms in equation B.9, which is the case for the operators in all of equation 4.1, the variances simplify to:

$$A^{kk} = \sqrt{2b^{kk}}. \tag{B.10}$$

For the collision operator, starting with the mathematical expression in equation 4.7, the time derivative update eqeuations for p and λ, $\dot{p} = \dot{p}(p)$ and $\dot{\lambda} = \dot{\lambda}(\lambda)$ become:

$$\dot{p}_{coll} = -\nu_p p + \beta\sqrt{1+p^2}\frac{2p^2-1}{\tau_c p^4}, \tag{B.11}$$

$$\dot{\lambda}_{coll} = 2\nu_D\left(\frac{\zeta_2}{\zeta_1} - \frac{\lambda}{2}\right), \tag{B.12}$$

$$A^{pp}_{coll} = \left(2\beta\nu_p\sqrt{1+p^2}\right)^{\frac{1}{2}}, \tag{B.13}$$

$$A^{\lambda\lambda}_{coll} = \left(2\nu_D\lambda\frac{\zeta_2}{\zeta_1}\right), \tag{B.14}$$

where

$$\begin{aligned} \nu_D(p) &= \frac{\sqrt{1+p^2}}{\tau p^3}\left[1 + Z - \beta\frac{1+2p^2}{p^2(1+p^2)}\right], \\ \nu_p(p) &= \frac{1+p^2}{\tau p^3}, \end{aligned} \tag{B.15}$$

and $\beta = T_e/(m_0c^2)$. ζ_1 and ζ_2 are *full elliptic* integrals over the poloidal angle θ, that enter into the calculation.

B.3 Full range collision operator

The collision operator is a central part of code since it is responsible for the basic physical simulation of the electrons time evolution in the plasma. The old ARENA's original collision operator is based on the analytical solution to the Fokker-Planck equation in certain limits. For too high plasma temperatures, in the order of $\sim$ 15keV, a zero-external electric field will not retain the Maxwell distribution for the Maxwell test. This is an anomaly, since the Maxwell test is supposed to be contained under any circumstances where there is no outer E-field. In the current code, the collision operator ported from `Fortran 77` is a two part expression, each valid for a span of electron energies. The parameters are adjusted so that the operator fits together (splined) in the centre energy region. The high energy part of the collision operator, which is derived from the general expression for any bounce-averaged variable (B.7), reads:

$$C(f) = \frac{1}{p^2}\frac{\partial}{\partial p}p^2\left(A(p)\frac{\partial f}{\partial p} + F(p)f\right) + \frac{2B(p)}{p^2}\mathcal{L}(f) \tag{B.16}$$

The abbreviated functions $A(p)$, $B(p)$, $F(p)$ are:

$$A(p) = \frac{\Gamma v_t^2}{cv^3}, \tag{B.17}$$

$$B(p) = \frac{\Gamma}{2cv}\left(1 + v_t^2 \frac{v^4 - 1}{v^2}\right), \tag{B.18}$$

$$F(p) = \frac{\Gamma v_t^2}{T_e v^2}, \tag{B.19}$$

(note that $p = m_e v$) and

$$\Gamma = \frac{n_e e^4 \ln \Lambda}{4\pi\epsilon_0}, \tag{B.20}$$

is the collision cross section v is normalised so $v \to \gamma v = q, \gamma = 1/\sqrt{1-v^2}$ and $v_t^2 = T_e/mc^2$ is the thermal velocity. This formulation of the collision operator is matched asymptotically matched to a collision operator valid for lower energy but *non-relativistic* electrons.

Here is for reference the derivation for the full range collision operator used e.g. in ANTS by Papp et al. [18]. The goal is to implement this is ARENA, but as is explained below, the operator it is not possible to directly insert the new operator in ARENA's code, also see figure B.1

To begin with, the parameter functions $A(p)$, $B(p)$ and $F(p)$ are rewritten to conform to the relativistic limit. This is done by means of the Chandrasekhar function $G(x)$ which describes the falling friction force from Coulomb collisions as a function of particle velocity. The parameter functions become:

$$A(p) = \frac{\Gamma v_t^2}{cv^3} 2\left(\frac{v}{2c^2 v_t^2}\right)^2 G\left(\frac{v}{vT_e}\right), \tag{B.21}$$

$$B(p) = \frac{\Gamma}{2cv}\left[\phi\left(\frac{v}{2c^2 v_t^2}\right) - G\left(\frac{v}{2c^2 v_t^2}\right) + v_t^2 v^2\right], \tag{B.22}$$

$$F(p) = \frac{\Gamma v_t^2}{T_e v^2} 2\left(\frac{v}{2c^2 v_t^2}\right)^2 G\left(\frac{v}{2c^2 v_t^2}\right) = A(p)\cdot\frac{T_e v}{c}. \tag{B.23}$$

Now, the collision operator 4.7 can be rewritten and for to simplify the expression, the following parameters are introduced: $q = v/\sqrt{1-v^2}$, the normalised momentum and $\tau^{-1} = \Gamma/(m_e^2 c^3)$ is the relativistic collision time and $\epsilon = v_t^2$ is the thermal electron speed squared.

$$C(f) = \frac{\sqrt{1+q^2}}{\tau q^3}\left[Z_{eff} + \Phi\left(\frac{q}{\sqrt{2\epsilon(1+q^2)}}\right) - G\left(\frac{q}{\sqrt{2\epsilon(1+q^2)}}\right) + \epsilon\frac{q}{1+q^2}\right]\mathcal{L}(f) + \frac{1}{\tau q^2}\frac{\partial}{\partial q}\left[\frac{q^2}{2\epsilon(1+q^2)} G\left(\frac{q}{\sqrt{2\epsilon(1+q^2)}}\right)\left[(1+q^2)f + \epsilon\frac{(1+q^2)^{3/2}}{q}\frac{\partial f}{\partial q}\right]\right]. \tag{B.24}$$

Now, to implement the iterative solution by Monte Carlo, operators need to be derived that correspond to the mean and variance required to write down the Langevin equation B.1.

The following expressions are constructed for brevity:

$$\mathcal{J}(q) = \frac{q^2}{\epsilon(1+q^2)} G\left(\frac{q}{\sqrt{2\epsilon(1+q^2)}}\right), \tag{B.25}$$

$$\mathcal{P}(q) = \frac{\epsilon(1+q^2)^{3/2}}{q}, \tag{B.26}$$

$$\mathcal{I}(q) = \frac{\sqrt{1+q}}{\tau q^3}\left[Z_{eff} + \Phi\left(\frac{q}{\sqrt{2\epsilon(1+q^2)}}\right) - G\left(\frac{q}{\sqrt{2\epsilon(1+q^2)}}\right) + \epsilon\frac{q^2}{1+q^2}\right]. \tag{B.27}$$

Now, the moment generating function is

$$\langle q^i \lambda^j \rangle = \frac{2\pi}{n}\int\int q^i \lambda^j f(q,\lambda) q^2 dq d\lambda, \tag{B.28}$$

so

$$\frac{d}{dt}\langle q^i \lambda^j \rangle = \frac{2\pi}{n}\int\int q^i \lambda^j C(f) q^2 dq d\lambda, \tag{B.29}$$

where n is the number of particles and λ is the pitch angle relation $v_{||}/v$. Integration by parts with $f(q,\lambda) = \delta(q-q_0)\delta(\lambda-\lambda_0)$ yields:

$$\frac{d}{dt}\hat{\lambda} = -\mathcal{I}(q)\hat{\lambda}, \tag{B.30}$$

$$\frac{d}{dt}\hat{\sigma_\lambda}^2 = \mathcal{I}(q)(1-\hat{\lambda}^2), \tag{B.31}$$

$$\frac{d}{dt}\hat{q} = \frac{1}{\tau\hat{q}^2}\left[-\mathcal{J}(\hat{q})(q+\hat{q}^2) + \frac{\partial}{\partial\hat{q}}[\mathcal{I}(\hat{q})\mathcal{P}(\hat{q})]\right], \tag{B.32}$$

$$\frac{d}{dt}\hat{\sigma_q}^2 = \frac{2}{\tau\hat{q}^2}\mathcal{J}(\hat{q})\mathcal{P}(\hat{q}). \tag{B.33}$$

A comparison of the three collision operators considered, the ARENA two-part operator, the ANTS operator, and the expression published in [2] is plotted in figure B.1. Note that the ANTS operator and the published expression look similar while the ARENA operator expressions give a smaller change in momentum $\dot{p}$ for lower momenta, close to the thermal peak p_{th}. If the ANTS operator could be rescaled in a way similar to the ARENA operator, it is possible that it could be incorporated into ARENA 90. Tests were made, but the ANTS operator tended to give too large changes in momenta when iterating. The full Fortran 90 code is listed in section C.7

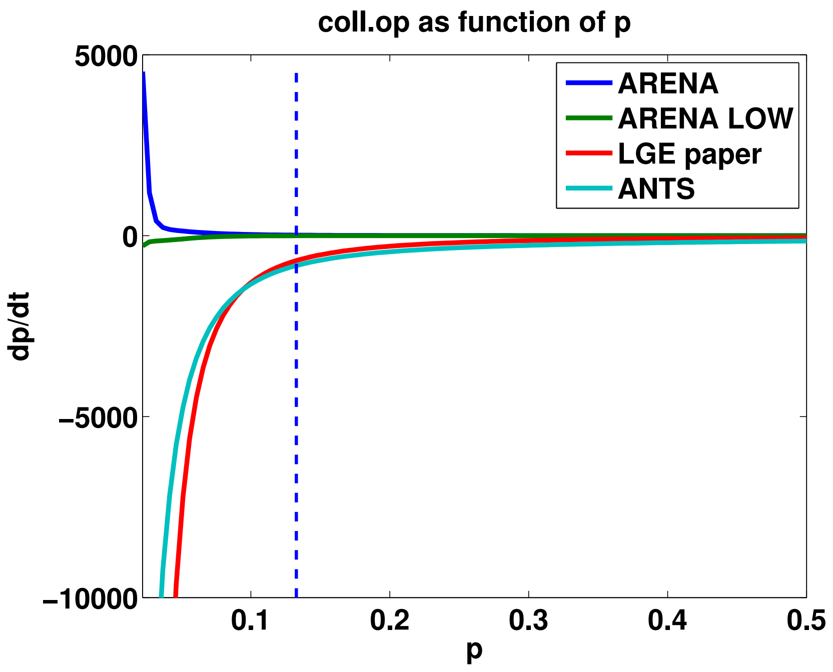


Figure B.1: A comparison of three collision operator expressions: the line where $\dot{p}(p)$ increases for small p is the high energy operator implemented in ARENA. It does not include a dependence on the Chandrasekhar function in its definition. The ARENA operator which has been used for simulations throughout this thesis uses the low-energy operator up to a splining point: $p_{spl} = 3 \cdot p_{th}$, where it is gradually replaced by the high energy expression.

C

Computer Code

Algorithms and code structure are discussed in this appendix. I give some general information on Fortran code and OpenCL, then comes descriptions of some of the subroutines in ARENA with comments. Then there is a section on ARENAs inputs and their physical meaning and how data is processed from ARENAs output. Finally the full code of ARENA's collision operator is included. Many of these parts felt too involved to keep in the main text, but are important for understanding the function of ARENA in detail, and are available here for reference.

C.1 Code and compilers

ARENA is written in Fortran, a programming language. It is an old standard - the name Fortran 77 implies that the standard was set around 1977, almost 30 years ago. Any programming language is interpreted and *compiled* to computer code by a compiler program. The compiler is created to follow a standard, like Fortran 77 or Fortran 90. If the compiler does not support a certain feature of a computer language it is said to not be compatible with that language. Different compilers can also offer varying performance of the output files. So *arena-intel* created by the `ifort` compiler by Intel, might perform differently than `arena-gfortran` created by the open source `gfortran` compiler. There are several compilers that can be used to compile Fortran 77 and Fortran 90 code. The `pgf` compiler by the Portland Group is not freely available, whereas the *ifort* compiler by Intel can be used on Linux-system free of charge if the software is not used for profit.

Portability refers to the possibility of copying code from one computer to another, running the same program. To copy ARENA, since Fortran compiles to *machine code* (Java rather, compiles to *byte code* which can be copied to any computer running a *Java virtual Machine*) it must be recompiled whenever the program is moved to a new PC.

C.2 OpenCL code

OpenCL is a framework used to create parallel programs on computer graphics chips which usually have multiple processor units that work with very simple logic. Very simple logic here means that a lot of work usually needs to be done by program authors to be able to perform any calculations. OpenCL is a new standard for writing non-graphics applications utilising the parallel power inherit in graphics processing chips (GPU:s). A more mature standard developed by NVIDIA is called *CUDA* and does the same thing, but it is tied in with NVIDIA hardware. Although CUDA is free to use for non-profit applications, this limits portability. To actually write a program, an API is needed. There is a core API for calling OpenCL routines available in the C language, and other *wrapper* API:s available.

To create the ARENA collision operator parallel proof of concept, I have used the `fortrancl` API available from Google Code: `http://code.google.com/p/fortrancl/`. The code sets up a series of tunnels to memory on the graphics chip and creates a recipe for performing the calculation. The end results can then be read back into the main program and output, for instance to ARENA.

```
call clGetPlatformIDs(platform, num, ierr)

  ! get the device ID
  call clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, device, num, ierr)

  ! get the device name and print it
  call clGetDeviceInfo(device, CL_DEVICE_NAME, info, ierr)
  print*, "CL device: ", info

  context = clCreateContext(platform, device, ierr)
  command_queue = clCreateCommandQueue(context, device, ...
      CL_QUEUE_PROFILING_ENABLE, ierr)

  ! BUILD THE KERNEL
  ! read program source file
  open(unit = iunit, file = 'coll_op.cl', access='direct', status = ...
      'old', action = 'read', iostat = ierr, recl = 1)

  source = ''
  irec = 1
  do
    read(unit = iunit, rec = irec, iostat = ierr) source(irec:irec)
    if (ierr /= 0) exit
    if(irec == source_length) stop 'Error: CL source file is too big'
    irec = irec + 1
  end do
  close(unit = iunit)

  ! create the program
  prog = clCreateProgramWithSource(context, source, ierr)

```

```
! build
call clBuildProgram(prog, '-cl-mad-enable', ierr)

! finally get the kernel and release the program
kernel = clCreateKernel(prog, 'coll_op', ierr)
call clReleaseProgram(prog, ierr)

! RUN THE KERNEL
! read in the ARENA distribution AND SIZE from file

vec1 = 1.0
vec2 = 2.0

temperature = 1.0
Z_{eff} = 1.0
lnLam = 18

! loop here?
call cpu_time(time0)
call system_clock(tick, clock_rate0, clock_max)
do k = 1,nloop

! allocate device memory
cl_peps = clCreateBuffer(context,CL_MEM_READ_ONLY, size_in_bytes, ierr)
cl_pp = clCreateBuffer(context,CL_MEM_READ_ONLY, size_in_bytes, ierr)
cl_plam = clCreateBuffer(context,CL_MEM_READ_ONLY, size_in_bytes, ierr)
...

cl_ppcdot = clCreateBuffer(context,CL_MEM_READ_WRITE, size_in_bytes, ...
    ierr)
cl_pacp = clCreateBuffer(context,CL_MEM_READ_WRITE, size_in_bytes, ierr)
cl_plcdot = clCreateBuffer(context,CL_MEM_READ_WRITE, size_in_bytes, ...
    ierr)
cl_paclam = clCreateBuffer(context,CL_MEM_READ_WRITE, size_in_bytes, ...
    ierr)


! copy data to device memory
call clEnqueueWriteBuffer(command_queue, cl_vec1, cl_bool(.true.), ...
    0_8, size_in_bytes, vec1(1), ierr)
call clEnqueueWriteBuffer(command_queue, cl_vec2, cl_bool(.true.), ...
    0_8, size_in_bytes, vec2(1), ierr)
...

call clEnqueueWriteBuffer(command_queue, cl_peps, cl_bool(.true.), ...
    0_8, size_in_bytes, peps(1), ierr)
call clEnqueueWriteBuffer(command_queue, cl_pp, cl_bool(.true.), ...
    0_8, size_in_bytes, pp(1), ierr)
call clEnqueueWriteBuffer(command_queue, cl_plam, cl_bool(.true.), ...
    0_8, size_in_bytes, plam(1), ierr)


! set the kernel arguments
```

```
call clSetKernelArg(kernel, 0, size, ierr)
call clSetKernelArg(kernel, 1, cl_peps, ierr)
call clSetKernelArg(kernel, 2, cl_pp, ierr)
call clSetKernelArg(kernel, 3, cl_plam, ierr)
call clSetKernelArg(kernel, 4, cl_I1, ierr)
call clSetKernelArg(kernel, 5, cl_I2, ierr)
call clSetKernelArg(kernel, 6, cl_ppcdot, ierr)
call clSetKernelArg(kernel, 7, cl_pacp, ierr)
call clSetKernelArg(kernel, 8, cl_plcdot, ierr)
call clSetKernelArg(kernel, 9, cl_paclam, ierr)
call clSetKernelArg(kernel, 10, temperature, ierr)
call clSetKernelArg(kernel, 11, Z_{eff}, ierr)
call clSetKernelArg(kernel, 12, lnLam, ierr) ! 13 args total


  ! execute the kernel
  call clEnqueueNDRangeKernel(command_queue, kernel, (/globalsize/), ...
      (/localsize/), ierr)
  call clFinish(command_queue, ierr)

  ! read the resulting vector from device memory
  call clEnqueueReadBuffer(command_queue, cl_ppcdot, ...
      cl_bool(.true.), 0_8, size_in_bytes, ppcdot(1), ierr)
  call clEnqueueReadBuffer(command_queue, cl_pacp, cl_bool(.true.), ...
      0_8, size_in_bytes, pacp(1), ierr)
  call clEnqueueReadBuffer(command_queue, cl_plcdot, ...
      cl_bool(.true.), 0_8, size_in_bytes, plcdot(1), ierr)
  call clEnqueueReadBuffer(command_queue, cl_paclam, ...
      cl_bool(.true.), 0_8, size_in_bytes, paclam(1), ierr)


! RELEASE EVERYTHING
call clReleaseKernel(kernel, ierr)
call clReleaseCommandQueue(command_queue, ierr)
call clReleaseContext(context, ierr)
```

C.3 More ARENA details

This section serves to describe some of the subroutines in the ARENA code, explain what role they play and to give possible explanations to certain effects in the final output. This does not cover all parts of the code, but act to show that we went over many major pieces of the code with great care when developing ARENA 90.

C.3.1 Cutoff momentum p

The momentum crossover value p_c acts as a limit for when to count runaway electrons having been created. It can be defined to some large value momentum p, which should be large enough for the electrons to act like runaways in any meaningful way, i.e. contributing to the runaway current and as a seed for secondary generation runaways.

In both versions of ARENA, there are two definitions of the limit present, and it is unclear how they overlap, i.e. the code might very well be using one limit in one subroutine, and the other limit somewhere else leading to inconsistent outputs and difficult interpretion of the results.

The first definition of the cross-over momentum p_c is a multiple of the thermal momentum

$$p_c = n \cdot p_{th} = n \cdot \sqrt{\frac{T}{m_e c^2}}, \tag{C.1}$$

if the temperature measured in keV:s. This is what was used in the Kulsrud reference paper [1] with $p_c = 10 p_{th}$.

Another definition comes from minimising a certain expression of the braking *friction* force $F_f(v_{norm})$ which is hard-coded into ARENA:

$$F_f(p = v_{norm}/c) = \left(1 + \frac{1}{p^2}\right)\left(1 + \frac{\ln p + p_0}{\ln \Lambda}\right), \tag{C.2}$$

with $p_0 = 0.1$ and $\ln \Lambda$ the Coulomb logarithm. This definition is also limited by an input parameter `p_norm_min` which is set to `p_norm_min`$= 2 \cdot 10^{-4}$ as default. It is likely that this definition is related to the critical energy level, which is the minimum of the friction force (figure 2.2).

In figure C.1 one can see how the first definition sets a higher bar for when a test particle should be restructured into a runaway-test particle. However, if a relevant parameter like the input electric field is changed to be closer to the critical field E_c: $E/Ec = 50 \rightarrow 10$, the opposite is true. As long as the definition is the same, this should not be a problem, still this is a possible room for error that should be considered.

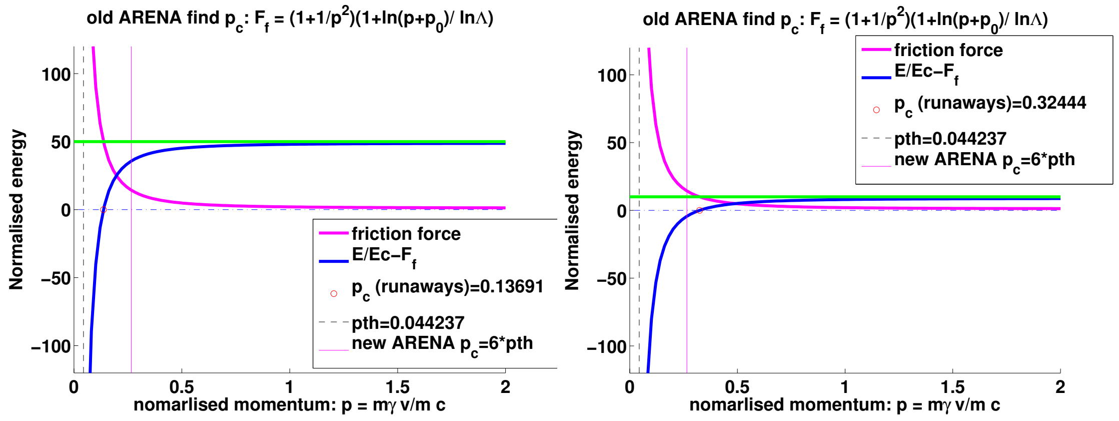
Two other cut-offs are present in ARENA, they are all multiples of the thermal momentum p_{th}. The second is the cut-off for weighting which is controlled from the input parameter `FWB`. The third is the splining of the two collision operators. Splining is made with a logistic function 4.17 centred in $x_0 = 3p_{th}$, an apparently arbitrary limit. This can be seen in figure C.1, and the function value for the high energy ARENA collision operator to the right of this limit should not be considered.

C.4 ARENA inputs

Tables C.1 through C.4 show ARENA input parameters in the XML input file of ARENA 90, a short description of each paramter's role and their equivalences in ARENA 77 input where applicable. The different tables have grouped the inputs by their function in the code, similarly to how the input parameters are groups in the input file.

C.5 Plotting ARENA output

ARENA outputs particle data at the end of each run and aggregated data for each time-step. The particle data can also be output several times during the simulation



(a) $E/E_c = 50$ (b) $E/E_c = 10$

Figure C.1: The two different ways of defining the cut-off for when to count a runaway electron that are present in different versions of ARENAs code. The first way uses a multiple of the thermal momentum p_{th} while the second minimises equation C.2 with a Newton method. A possible issue arises if a low electric field E/E_c is chosen (and then an appropriately high temperature to keep E/E_D constant, because the "new" definition of $p_c = \texttt{RCPN}p_{th}$ gives a too low value for the cut-off compared to the minimisation algorithm gives. A solution to this is to choose a higher value for `RCPN`, but it is important to keep in mind, and possibly these definitions should not be separate.

Table C.1: Monte Carlo main simulation parameters. These parameters control the number of test particles and the time step length.

Parameter name	Description	A77
`number_of_particles`	number of Monte Carlo test particles	NPART
`number_of_timesteps`	number of iterations for the main loop, this is rescaled into a sub-time step dt_s if $dt < \tau_{th}(T)$	NTSTEP
`timestep_length`	dt times the number of time steps gives the simulated time	TIMEST

Table C.2: Parameters describing the physics of the tokamak, such as the reactor size and the **B**-field strength. This section contains the parameters that are mainly of interest when running a simulation, such as the magnitude of the applied E-field as a result of the thermal quench and the plasma bulk temperature T, etc.

Parameter name	Description	A77
`inverse_aspect_ratio_min`	ϵ_{min}, the start of the radial simulation border and is used e.g. when choosing a random position of a new particle	EPSMIN
`inverse_aspect_ratio_max`	ϵ_{max} the maximum ϵ which is the end of the simulation spatial domain	EPSMAX
`inverse_aspect_ratio_iron`	used to find the wall spatial coordinate, used for e.g. calculating the E-field profile	EPSFE
`E_applied_norm2cr`	outer magnetic field resulting from the thermal quench normalised to the critical E-field E/E_c	EECRAT
`Z_eff`	$Z_{eff} = \sum_i n_i Z_i^2 / n_e$ is the effective charge of the plasma. This is a measure of plasma purity where $Z_{eff} = 1$ is a pure plasma. Usually this is the input parameter. It comes in with the calculation of β in the collision operator	ZEFF
`n_bulk_elec_central`	central bulk electron density, this is approximately the density of the plasma in the simulation region. see section 4.7.1 and equation 4.22	CNE0
`bulk_dens_prof`	density profile scaling factor ≈ 1	EQUDEN
`bulk_dens_prof_exp`	density profile exponent $\approx 1 \cdot 10^{-8}$	EXPDEN
`T_e`	plasma electron temperature	TEMPE0
`bulk_tem_prof`	temperature profile scaling factor ≈ 1	EQUTEM
`bulk_tem_prof_exp`	temperature profile exponent $\approx 1 \cdot 10^{-8}$	EXPTEM
`B_0`	magnetic field on the centre axis of the circular plasma	B0
`R_major`	major radius of the tokamak at centre axis	RMAJOR

Table C.3: The physics relevant program parameters such as the number of flux surfaces over which to do calculations like the elliptic integrals for the collision operator update. Also the parameters for changing *cut-off*- and *weight* limits reside here.

Parameter name	Description	A77
`p_norm_max`	a limit for the momentum after which it is no longer interesting to simulate this test particle. This is mainly used for saving computation cycles.	CPMAX
`p_norm_min`	minimum for the test particle momentum. This is used as an additional user input way of determining the minimum crossover momentum p_c	CPZERO
`number_of_flux_surfaces`	is used in various context when calculating profiles in the simulation, e.g. the **E**-field is evaluated as a function of $\epsilon = r/R$ at these many positions. Should be an uneven number!	NSURF
`FWB`	boundary in momentum space on the magnetic axis. Particles with momenta below this boundary have the weight `wghtb` and those with momenta above has the weight `wghtf`, normally `wghtb` $\gg$ `wghtf`	FWB
`RCPN`	scaling parameter to set the cut-off momentum, $p_c = \texttt{RCPN} \cdot p_{th}$	RCPN
`dB_per_B`	$\nabla B/B$, the normalised perturbed magnetic field	DBBRAT
`Coulomb_logarithm`	$\ln \Lambda$ is the Coulomb logarithm (section 2.1)	CLNLAM
`ttedec`	electron temperature characteristic decay time	TTEDEC
`Te_min`	electron temperature lower limit, to keep electron temperature higher than 0 at all times	TEMIN
`T_e_boundary`	electron edge temperature for a linear electron temperature profile	TEMPEB

Table C.4: Program execution parameters and *switches*. These parameters control how the simulation is executed, i.e. what calculations and diagnostics outputs are enabled.

Parameter name	Description	A77
`frac_init_runaways`	initial fraction of runaways, this is not currently enabled in the code and is used for diagnostics purposes. 1 is the default, though the initial fraction of runaways is 0, when running ARENA	FRACIN
`sw_selfcons_electric_field`	flag for enabling self consistent E-field calculation	-
`sw_source_term`	flag for enabling secondary runaway generation through close collisions	-
`sw_sync`	flag for including synchrotron radiation losses in the simulation	-
`nepsd`	number of flux surfaces for some diagnostic output	-
`npd`	number of momenta points for diagnostic output	-
`NTAV`	width of the moving average time window for self-consistent E-field calculation	
`do_maxwellian_test`	overrides the calculation setting for thermal electrons so that collision are updated even for low energy particles. This is used to do the Maxwell test (section 4.6)	-
`plot_interval`	number of iterations after which the particles' momenta are dumped to a file. This allows for outputting data during the course of the simulation	-

Table C.5: The ARENA version refers to the different default file names of each ARENA version. The input file of ARENA 90 is an XML file with separate sections i.e. *physics*, *monte carlo* etc., whereas the ARENA 77 input is read by row. Distribution output is the output at the end of the run, showing individual test-particle data.

	ARENA 77	ARENA 90
input file:	input.data	arena_input.xml
parameter output file:	arena.output	output_initial_parameters
output file:	pl1.res	output_runaway_current
time output:	pl4.res	output_MC_particles

by setting the `plot_interval` option in the input file to a number $\neq 0$ (ARENA 90). The output files and their contents are listed in table C.5. The full particle output from mid-simulation of ARENA 90 are stored in files named `output_momenta.[0]i`, $i = 0,10,20...$ or whatever the plot interval is set to. There are two columns in each file, the first is the normalised momentum value $p = m\gamma v/mc$ for each test particle. The second is the weight associated with each particle w_b and w_{rw} (see section 4.7.3). The weights also scale up the particle count to actual physical electrons, i.e. $w_i = w_i(n_e = n_c \cdot V_{torus}), i = bulk, runaway$. Note that the weighting has been removed in the final draft of the ARENA code.

To plot the output graphs I used a series of MATLAB scripts to get the input-data and output for each time step. The important parameters are the *number of runaway electrons* from ARENA. This will be a fraction of the total number of test particles, so it makes sense to plot the fraction in the output, the total number of electrons to is the value `NETOT` from `arena.output` in the Fortran 77 code and `netot` from `output_initial_parameters` in the ARENA 90. The time has been normalised to thermal collision times, τ_{th} this value is `TAUT` in ARENA 77 and `tau_thermal` in ARENA 90. My MATLAB script parses each of these *initial output* files to a list of key-value pairs.

The Dreicer generation curve as a function of outer electric field E/E_D was created from the raw output data from ARENA. The slope is calculated as the *maximum value* of the *smoothed* differentiation of the *fraction of* runaway electrons;

$$\gamma_D = \texttt{max}(\texttt{smooth}(\frac{dn_r}{dt})), \tag{C.3}$$

where smooth is a smoothing function in MATLAB using an average window of $\sim$ 50...500 The size of the windows depends on how noisy the data was. For example, a higher electric field makes for better parameter space for ARENA, so dn_r/dt when $E/E_D = 0.1$ is a lot less noise than for e.g. $E/E_D = 0.04$.

Examples of the numerical derivatives to find the growth rates can be seen in figures C.2 through C.5.

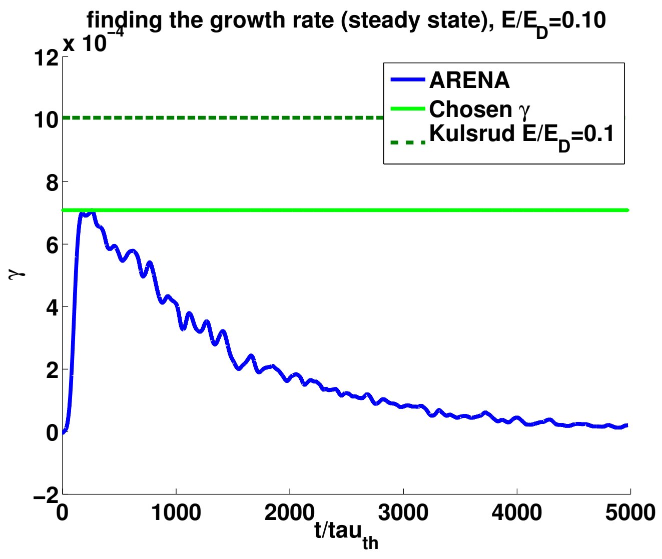


Figure C.2: The numerical differentiation of the runaway fraction of the total number of electrons give the Dreicer generation rate. The fraction ends up at one, where the rate then pans out, since no more runaway electrons can be generated. This is one of the main tests of ARENA working according to expectations. Note that the rate does not quite reach the kulsrud numerical rate in this picture, the reason being that the total number of electrons become runaways too quickly for the rate to saturate. Also see LUKE rates in figure 5.7. Here $E/E_D = 0.10, T_e = 1\text{ keV}, \mathrm{n_e} = 2 \cdot 10^{19}\text{ m}^{-3}$.

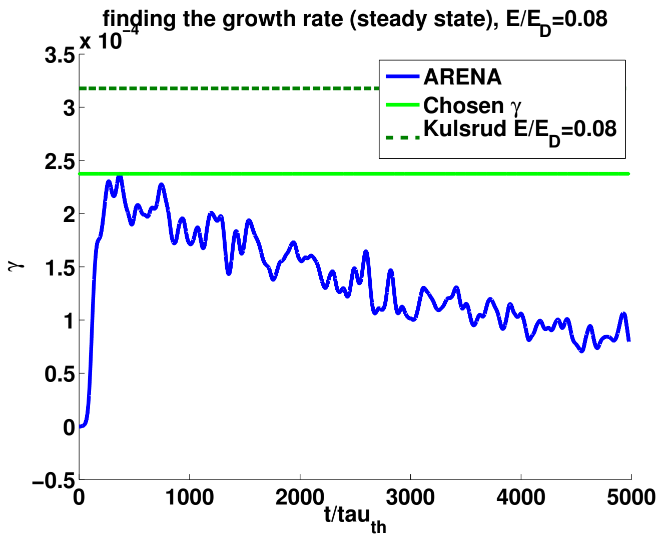


Figure C.3: $E/E_D = 0.08, T_e = 1\text{ keV}, \mathrm{n_e} = 2 \cdot 10^{19}\text{ m}^{-3}$, as the Dreicer rate gets lower, the rate gets a chance to saturate, however at the same time the data output gets noisier and the simulation takes a longer time. Note that the Kulsrud rate here is still *higher* than ARENAs rate.

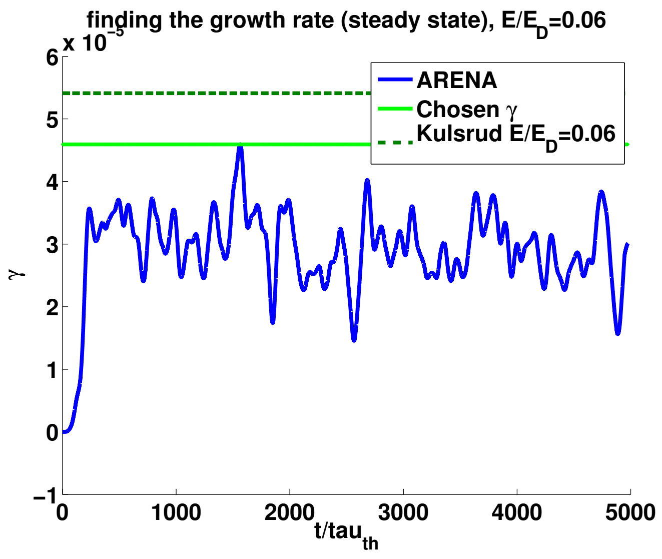


Figure C.4: The Dreicer rate when $E/E_D = 0.06, T_e = 1\ \text{keV}, \text{n}_\text{e} = 2 \cdot 10^{19}\ \text{m}^{-3}$.

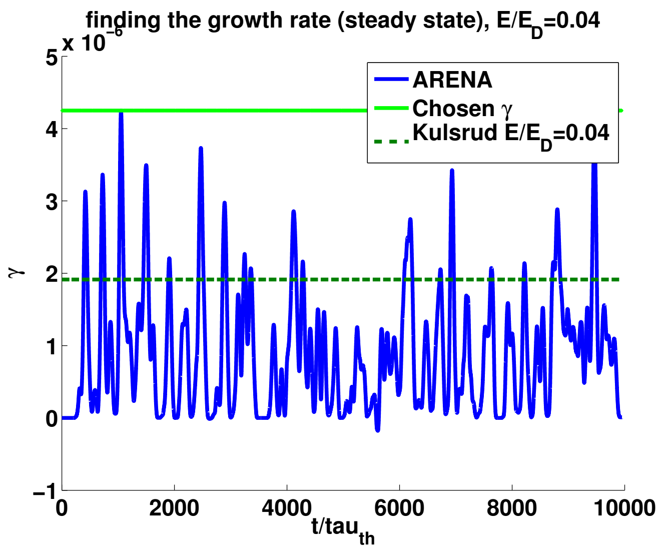
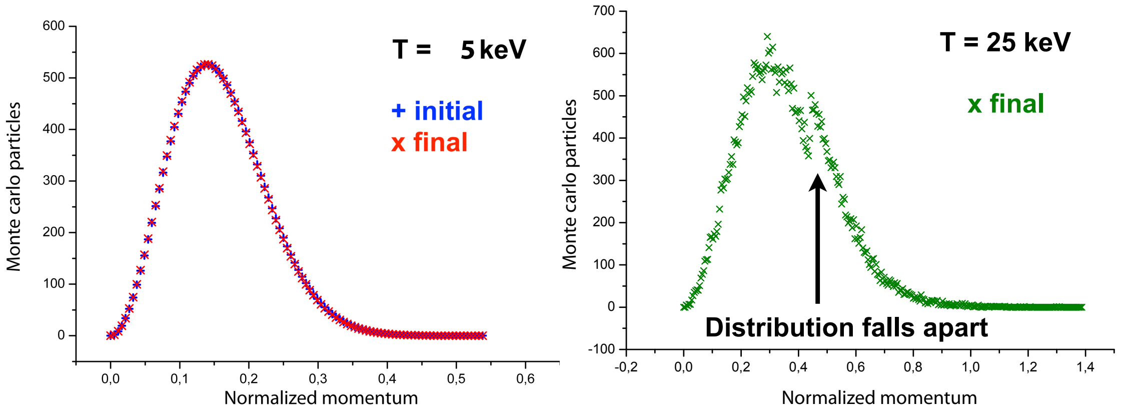


Figure C.5: The Dreicer rate with $E/E_D = 0.04, T_e = 1\ \text{keV}, \text{n}_\text{e} = 2 \cdot 10^{19}\ \text{m}^{-3}$. This data is the worst fit, which is apparent from all the noise. Unfortunately, since this is the lowest outer electric field, it also saturates slowest, so to get a simulation throughout the rate increasing the number of particles cannot be set too high. Hopefully a parallelised code could speed up the process. Also see the difference to the Dreicer rate as a function of electric field strength in figure 5.2.



(a) keeping Maxwellian (b) split Maxwellian

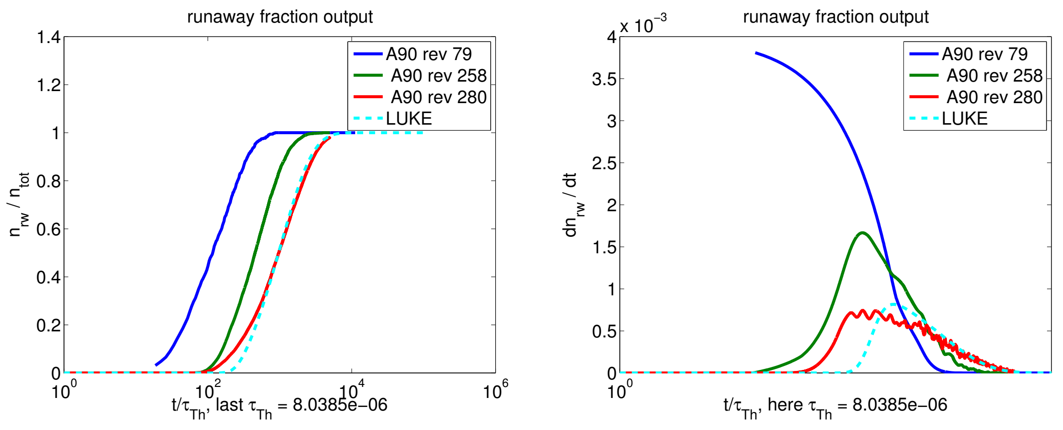
Figure C.6: The Maxwellian test works for a range of temperatures, but not arbitrary ones. If the temperature becomes $T \gtrsim 25$ keV the maxwellian will start to display a cusp in the distribution. This could be related to the weighting scheme or the two-part collision operator.

C.6 ARENA's evolution

ARENA has been going through many iterations. Primarily the new ARENA 90 version which has been under development during most of the time that I have written this thesis. Some of the major changes that have been made involves removing the faulty weighting scheme which created a too high premature runaway generation rate and a cusp in the Maxwell preservation test - see figure C.6. The Maxwell preservation test would not work for arbitrary temperatures. One possibility is that this is related to the way that the collision operator consists of two expressions. From a physics point of view, the only variable effecting the Maxwell distribution is the applied electric field, so if this is null, the Maxwellian should be preserved regardless of other parameter settings, thus if the test does not pass it indicates a limitation in the parameter space which ARENA models. In the final version, the Maxwell test passes for high temperatures up to 25keV, like in picture 5.4.

Furthermore, experiments were made with different collision operators, primarily to try to insert the ANTS collision operator in ARENA. A rescaling issue which can be seen in figure B.1 prevented this from being done without reformulating the ANTS operator to ARENA's units, which in the end, we could not do qualitatively.

Figure C.7 shows the output of ARENA 90 at different stages of the development. The three revisions 79, 258 and 280 that are plotted show qualitatively the difference between the long running bad version of ARENA which had faulty "corrections" to some strange equations form the old ARENA; the version with weighting and the version without weighting. Only the latest version met both the Dreicer benchmark and the Maxwell test with temperatures up to 25keV.



(a) Runaway fraction (b) generation rate

Figure C.7: Plots of ARENA output for different versions. This graph shows how the fraction of runaways has been made to correspond better to the reference LUKE output over time. The final revision also shows much better conformance with the differentiated curve with no factor 2 difference from revision 258.

C.7 Code example

This is a code example of the new ARENA 90 code. The Collision operator that was discussed in section 4.4 and also ported to a C-program for the parallel computing proof of concept is included in full.

```
module collision_operator
    use numeric_constants
    use globals
    use comdis
    implicit none

contains
    subroutine mccoll_v1 (peps, pp, plam, pi1, pi2, ppcdot, pacp, ...
        plcdot, paclam)
      use globals
      use comdis
      implicit none
      real, intent (in) :: peps, pp, plam, pi1, pi2
      real, intent (out) :: ppcdot, pacp, plcdot, paclam
      ! ! define real valued variables
      ! logisv and logder were logicals in the merljin and lge ...
          versions since their
      ! first letter is 'L'. However they are integers here, because ...
          Gfortran
      ! has problems of converting real−to−logical, but can do ...
          real−to−integer conversion.
      ! This is sufficient here since their value can be in the ...
```

```
          interval [0;1],
      ! which upon conversion will be 0 or 1, which are equialent to ...
          logical
      ! .false. or .true., so in this special case using logical or ...
          integer gives
      ! the same results
      ! CHANGE: fixed this error by using reals.
      real :: logisv, logder

      temperature=tempe(peps) ! getting the radially dependent temperature
      !MJ: normalised thermal velocity
      zpth = sqrt (temperature/electron_mass_in_ev)
      zpth2 = temperature / electron_mass_in_ev
      ! Second definition of thermal velocity for the low—energy part ...
          of the coll.op
      zzpth = zpth * sqrt (2.0)
!
      zp = pp + 1.0E—6
      zp2 = zp * zp
      zp3 = zp2 * zp
      zp4 = zp3 * zp
!
      zpc = 0.4 * zpth
      zpc4 = zpc * zpc * zpc * zpc
      zpcpa = 2.0 * zpth
!
      if (zp .lt. zpcpa) then
        zpb = zpcpa
        zpb2 = zpcpa * zpcpa
        zpb3 = zpb2 * zpcpa
        zpb4 = zpb3 * zpcpa
      else
        zpb2 = zp * zp
        zpb3 = zp2 * zp
        zpb4 = zp3 * zp
      end if
!
      ztheta = 1.9563E—3 * temperature
!
      zt1 = (1.0+zp2)
      zt2 = (1.0+log(zp+1.0)/codeparams%coulomb_logarithm)
      zt3 = zpc4 + zp4
      zt4 = sqrt (zt1)
      zz = ztheta * (1.0+2.0*zpb2) / (zpb2*(1.0+zpb2))
      zz = min (zz, 1.0)
!
      zbeta = 2.0 * (1.0+physparams%Z_eff—zz) * sqrt (1.0+zpb2) / zpb3 ...
          * &
        (1.0+log(zp+1.0)/codeparams%coulomb_logarithm)

      znu = zt1 * zt2 / zp3
      znu = znu * zp4 / zt3
      zdnu = (1.0+3.0*zp2—4.0*(1.0+zp2)*zp4/zt3) / zt3
```

```
!
      ppcdot1 = - zp * znu + ztheta * &
     & ...
         (((2.0*znu/zp+zdnu)*zt4+znu*zp/zt4)*zt2+znu*zt4/((zp+1.0)*codeparams%coulomb_logar
!
      pacp1 = sqrt (2.0*znu*ztheta*zt4)
      plcdot = - zbeta * (0.5*plam-pi2/pi1)
      paclam = sqrt (2.0*zbeta*plam*pi2/pi1)
!
!     LOW ENERGY Collision Operator part for p_dot
      zx = zp / zzpth
      zx2 = zx * zx
      zg = gstix (zx)
      zzpth2 = zzpth * zzpth
      zve = speed_of_light * zzpth / sqrt (1.0+zzpth2)
      znuve = speed_of_light * speed_of_light * speed_of_light / ...
          (zve*zve*zve)
      exp_x2 = exp (-zx2)
!
      pacp2 = sqrt (2.0*znuve*zzpth2*zg/zx)
      ppcdot2 = znuve * zzpth * (-zg*(2.0+1.0/zx2)+1.1248*exp_x2/zx)
!
!     Calculating final components of p_dot:
      zalpha = zp / sqrt (2*zpth2*(1+zp2))
      zb1 = zpth2 * zt1 * zt4 / zp3
      zb2 = zpth2 * zt1 * zt4 * gstix (zalpha) / zp
!
      ! logistics function for splining together the two energy region ...
          expressions:
      logisv = logistic (zp, temperature)
      logder = logisv * (1-logisv)
!
      pacp = pacp1 * logisv + pacp2 * (1-logisv)
      ppcdot = ppcdot1 * logisv + ppcdot2 * (1-logisv) + zb1 * logder ...
          - zb2 * logder

      return
    end subroutine mccoll_v1

!  G: The Chandrasekhar function: calculate gstix(x)=(erf(x) - x * ...
   erf'(x) / (2 * x**2)
    real function gstix (x)
      implicit none
      real :: x,x2,exp_x2
      x2=x*x
      if (x .lt. 0.01) then
        gstix = 1.1284 * (0.33333D0*x+0.2D0*x2*x)
      else
        if (x2 .lt. 20.0D0) then
          gstix=psi (x, x2, exp(-x2))
        else
          gstix = 1.0D0 / (2.0D0*x2)
        end if
      end if
      return
    end function gstix

!  Logistic function: CALCULATE logistic(x) = 1/(1 + exp(-(x-x0)/Δx))
!    x0     = centering parameter, to be set by the user
    real function logistic (px, temperature)
      implicit none
      real :: px, px0, pdx, temperature

      !MJ: treshold for transition between low and high energy region
      px0 = 3 * sqrt (temperature/electron_mass_in_ev)
      pdx = 0.1 * px0

      logistic = 1 / (1+exp(-(px-px0)/pdx))
      return
    end function logistic
end module collision_operator
```