

TDK dolgozat

Csépány Gergely L.

Kinetic simulation of runaway electrons in tokamaks

Gergely L. Csépany

Physics Msc I.

Supervisors:

Dr. Gergő Pokol, *associate professor*

Gergely Papp, *PhD student*



Budapest University of Technology and Economics
Faculty of Natural Sciences – Institute of Nuclear Techniques
2011. November 2.

Contents

Contents	3
1 Introduction	4
1.1 Runaway electrons	4
1.2 Physical model of ARENA	5
1.3 Aim of the current research	7
2 Code update	8
2.1 Converting from Fortran 77 to 90	8
2.2 Restructuring and speed up	9
2.3 Makefile automatization	9
2.4 Random Number Generators	10
2.5 Documentation	12
2.6 Data structures and database storing	12
3 Standard tests	14
3.1 Maxwell-test	14
3.1.1 Implementation of the Maxwell-test	14
3.1.2 Maxwell-test results	15
3.2 Dreicer-generation test	18
3.2.1 Implementation of the Dreicer-generation test	18
3.2.2 Dreicer-generation test results	19
3.3 Avalanche-test	20
4 ITM	21
4.1 ITM infrastructure	21
4.2 ARENA in ITM	21
5 Conclusions	23
6 Acknowledgements	24
References	26
A Input parameters	27
B Collision operators	29
B.1 Collision operator in the paper	29
B.2 Collision operator in the code	29

1 Introduction

The energy need of our civilization is constantly increasing and, to meet this need, countless power plants are being built. In 2011 the fossil fuel plants still take up the majority of the power plants despite their negative environmental impact [1]. One alternative to the fossil power plants is the nuclear fission energy, but its popular acceptance varies by time. Fusion power might be another solution to the energy problem in the future, which resembles many properties of an ideal energy source [2]. The technically most advanced fusion device is the tokamak, and ITER, the next major tokamak is under construction in Cadarache, France [3]. Tokamaks are a mature technology, but even after decades of research there are still not fully understood effects, such as the runaway electron generation and suppression.

1.1 Runaway electrons

Tokamak plasmas have various instabilities that can degrade the confinement. The most serious outcome of an instability is a disruption, which is a sudden loss of magnetic confinement and a release of stored thermal energy to the surrounding structures [4]. The temperature of the limiting material is raised leading to a release of impurities. The influx of impurities from the edge radiate away the energy because of their high atomic number, and this leads to a quick cooling of the plasma. Due to the cooling, the resistivity (η) of the plasma increases ($\eta \sim T^{-3/2}$) [5], which makes it difficult to drive the current (which is of the order of several megaamperes). As a result, there will be a dramatic rise of the toroidal electric field ($E_{toroidal} \approx R \cdot I$, where R and I are the resistivity and the current of the plasma).

The toroidal electric field accelerates the electrons and if the field is over the critical strength then some electrons accelerate to relativistic velocities because the drag force is not enough to slow them down (a sketch of the friction force as the function of the electron energy is shown in Figure 1). The critical electric field, which is the minimum of the friction force, is

$$E_c = \frac{n_e e^3 \ln \Lambda}{4\pi \epsilon_0^2 m_e c^2}, \quad (1)$$

and the Dreicer field (when the electric field is so strong that all the thermal electrons run away) is

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

where n_e is the electron density, $\ln \Lambda$ is the Coulomb-logarithm (its value is typically between 5 and 20) and T_e is the electron temperature in electronvolts. The

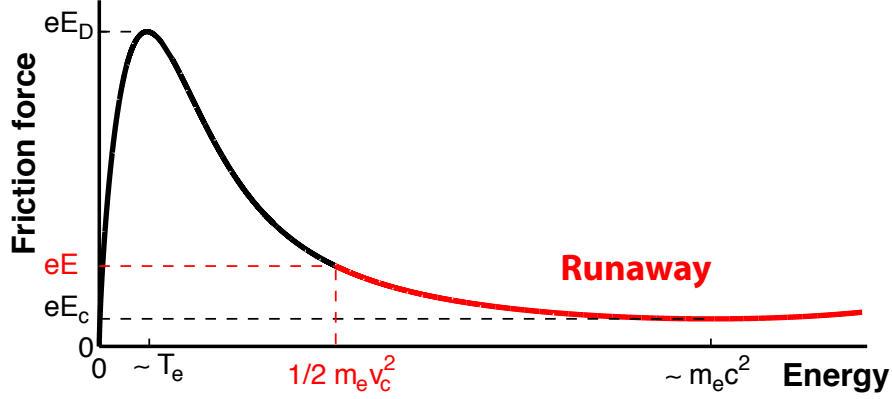


Figure 1: The friction force as a function of electron energy [6]

runaway phenomena can also be described as that the fast electrons have such a long mean free path that they do not collide enough to slow down while the strong toroidal electric field accelerates them. This phenomenon is called the *primary* or *Dreicer* generation and these fast electrons are called as runaway electrons [7]. Runaway electrons are produced in large numbers in disruptions and because of their high velocity they carry a substantial energy (10–100 MeV per runaway electron whereas the thermal electrons have an energy of 5–25 keV). Other primary generation effects are also known, such as hot-tail generation [8], inverse Compton scattering via energetic γ rays, tritium decay [9] and others, but their importance is low in this paper, since the simulation being discussed does not take them into account and they are only significant in special scenarios.

The secondary generation is the *avalanche* effect when a high energy runaway electron hits a thermal electron and passes enough energy to it to run away, while the knocking particle still remains above the runaway threshold. The avalanche generation is orders of magnitude stronger in large tokamaks, the gain is $g \approx \exp(2.5 \cdot I_{pl})$, where I_{pl} is the plasma current in MAs, which gives $g \approx 10^4$ for JET ($I_{pl} = 4$ MA) and $g \approx 10^{16}$ for ITER ($I_{pl} = 15$ MA) [4, 10]. Avalanche generation can only happen if there are runaway electrons, which are produced by the Dreicer or other primary generation mechanism.

1.2 Physical model of ARENA

ARENA was a breakthrough code in combining a significant number of effects in one program and was famous for the self-consistent modeling of the electric field induced by the runaway current [11, 12].

The physics model behind the code is derived from the six-dimensional kinetic equation simplified to a three-dimensional Fokker-Planck by bounce-averaging,

which is solved by a Monte Carlo method. The bounce time is the time which it takes for a particle to complete an orbit in the poloidal plane. In the case of a disruption this treatment is sufficient to simulate the electron dynamics on time scales significantly longer than the bounce time (for the deduction of bounce averaging see e.g. Ref. [13]). The Monte Carlo method was chosen because the relativistic energies of the electrons would have required a very complicated grid setup for finite element or finite difference solutions to solve the Fokker-Planck equation. The self-consistent modeling of the electric field is calculated by a finite difference solver and is coupled to the Monte Carlo core to give a full simulation of runaway electron dynamics.

The ARENA calculates the distribution function of electrons as a function of the radial coordinate of a circular cross section toroidal geometry. The geometry consists of concentric circular flux surfaces and the particles are characterized by their phase space variables

$$\vec{I} = (p, \lambda, r),$$

where r is the minor radius of the flux surface, p is the normalized relativistic momentum

$$p = \frac{mv}{m_0c} = \frac{\gamma m_0 v}{m_0 c} = \frac{\gamma v}{c}$$

and λ is the magnetic momentum normalized in a way to indicate that if a particle is trapped in the magnetic well (on the 'banana orbit', $\lambda > 1$) or passing ($\lambda < 1$)

$$\lambda = \frac{p_{\perp}^2}{p^2 b(\theta)}$$

in which $p_{\perp}$ is the normalized momentum perpendicular to the magnetic field, and $b(\theta)$ is the normalized magnetic field: $b(\theta) = B(\theta)/B_{max}$, where θ is the poloidal angle.

The bounce-averaged Fokker-Planck equation is

$$\frac{\partial f}{\partial t} = \langle L_E(f) \rangle + \langle C(f) \rangle + \langle L_{synch}(f) \rangle + \langle L_{spat.trans.}(f) \rangle + \langle S \rangle + \langle l_B \rangle \quad (2)$$

which includes operators for the all the effects ARENA simulates and $\langle \dots \rangle$ stands for the bounce-averaging.

The operators in Eq. (2) include the effects of the electric field (L_E), the collision of the electrons (C), the synchrotron radiation (L_{synch}), the spatial transport ($L_{spat.trans.}$) and the source of the high energy electrons (S and l_B). For brevity I do not include the detailed description of the operators and their Monte Carlo counterparts, which can be found in Ref. [11].

The cooling of the plasma in ARENA is modeled by a simple exponential decay of the temperature which is enough for a simple disruption simulation, but unsuitable for more complex simulations.

1.3 Aim of the current research

The code of ARENA was used until 2004 but it was not maintained since then. The main goal of this research project is to optimize the ARENA code and improve its physical models. The new physical effects may include the hot-tail generation, the Bremsstrahlung radiation and a new collision operator.

Updating the code includes the following tasks:

- using automated compiler setup realized by makefile,
- writing user and developer documentation,
- creating standard test suites to verify the physical validity as well as optimize the runtime of the code,
- optimizing the code since the Monte Carlo core is called some billion times in each run.

These task are detailed in Section 2. Other important point of the upgrade is to rework the plasma cooling model to be able to simulate cases where the exponential decay would not be an appropriate approximation. Standard tests are needed to validate the physical models of ARENA, which are preferably automated tests as described in Section 3. For the integration in a European-wide simulation workflow the program needs to be restructured to use complex datastructures, which is detailed in Section 4.

2 Code update

ARENA was written in Fortran 77, a programming language invented 33 years ago, lacking a number of programming techniques considered a standard nowadays. Fortran 90 became a standard in 1992 and included important upgrades over its predecessor making a viable target for conversion [14]. Fortran is a science-oriented programming language with support for complex numbers, arbitrary array indexing, array operations and other scientifically often used features therefore I chose to go on with Fortran but use the modern version.

2.1 Converting from Fortran 77 to 90

The main reasons to convert the code from Fortran 77 to Fortran 90 were:

- Availability of modules to store data and functions
- Compatibility with the ITM data structures (see more in Section 4)
- Possibility of parallelization
- More legible and maintainable, clean code [15]

The storage of particle variables in Fortran 77 was carried out by using `COMMON` blocks which can be described as a very basic implementation of global variables. The `COMMON` blocks have an unfriendly behavior that only the order of the variables define the passed values between subroutines, defeating the use of variable names. Fortran 90 introduced the usage of modules, which store data by variable names making it more suitable for shared data storage and also makes it possible to encapsulate parts of the program in single modules (eg. separating the Monte Carlo or the input reading routines). The Fortran 90 also introduced the use of operator overloading, interfaces, long variable names (31 characters instead of 6) and dynamic memory allocation, which greatly help development and maintaining the modularity of the program.

I performed the conversion mainly by hand with some help from `convert.f90` [16]. This part of work included the rewrite of `DO`, `GOTO`, `CONTINUE` statements into the more modern `DO`, `END DO` structure, changing `COMMON` blocks into modules and introducing longer than 6 character variable names. The latter is important for the user and developer usability since it is hard to remember cryptic variable names (eg. `zdlmx` stands for `delta_lambda_max`). Another important change was to make the code fully explicit, ie. disabling the default implicit typing of Fortran. The implicit typing enables the usage of variables without proper definition which can lead to problems with different compilers, because their behavior with an undefined variable might differ, such one might assign a zero value to an undefined variable while another might handle it as undefined or NaN.

2.2 Restructuring and speed up

I started the restructuring of code by removing or changing so called archaic functions to their modern counterparts (eg. DEXP to EXP) and by removing functions copied from external libraries like BLAS or LAPACK. The code relies on these external libraries so instead of copying the source code of these functions, I use GotoBLAS [17] or the Intel Math Kernel Library (MKL) [18] where applicable. These libraries also have the benefit of being optimized for specific hardware meaning that a substantial speed up (a factor of 2-8, see e.g. Ref. [19]) can be achieved by using them.

I continued the restructuring by changing important variable names to more legible ones, for example I now use `E_applied_norm2cr` instead of `EECRAT`. I also collected the physical and mathematical constants into a module, like the definition of π , the speed of light or the electron mass to which I can now refer to in any other module.

Making the code run faster necessities not to (re)calculate things that are not needed – for example the square root of π –, and in general using as few operations as possible. This is part of the so called single processor optimization which means to rework the code to run on one CPU as fast as it can without losing precision and any change in the results. Since ARENA is a Monte Carlo code, its memory requirements are low while its computational requirements are high. ARENA does not use much memory, but since the management of the cache is important for optimization, it will be only done after the code is validated. Therefore I can save time by assigning a variable to every piece of data that will be used more than once, for example x^2 or $\exp(-x^2)$ (where x is position data) or the plasma temperature in keV. The single CPU optimization is important to be carried out before parallelization, since if the code wastes one processor's time then the parallelized code will only waste the time of hundreds of processors which is not what parallelization is aimed at. Parallelization of ARENA in principle is feasible because Monte Carlo codes are usually easy to run on multiple processors, but this may only take place when the code does its best on a single CPU.

Some parts of this old code had to be reverse-engineered to understand its logic, and this process revealed that the versioning of the code ARENA garbled through the years. This manifested in a modified collision operator and weighting scheme, which were originally part of the development version of ARENA.

2.3 Makefile automatization

ARENA had a basic makefile to help compilation which was updated to accommodate various systems and to help the debugging process.

The makefile updates can be summarized as:

- Creating three versions of the binary file:
 - optimized, without debugging (-O2 optimization flag at least), this should be used for serious computations
 - non-optimized, for debugging (-g), this can be used for debugging in the development phase
 - non-optimized, for profiling (-pg), this can create gprof compatible profiling data [20]
- Detection of compiling environment: ARENA is used already on a number of systems (server of the department, ITM gateway, developers' laptops, etc.), which have the necessary libraries at different paths. The makefile looks for these libraries and only compiles the binaries with all requirements fulfilled
- Default real is double: for better precision, all real (floating point) numbers are defined as double.
- Separate objects (intermediate binary files in the process of the compilation) for all flavors of builds: the makefile works with Gfortran + GotoBLAS or Intel Fortran Compiler + MKL, and creates all three types of binaries for both combinations. The use of objects reduces the recompilation time when only a small fraction of the code is modified.

2.4 Random Number Generators

The original Random Number Generator (RNG) in Arena was the UNI function. The authors included this function instead of the Fortran built-in because they had problems with the built-in RNG at the time of the development.

In Arena we have easy access to three RNGs:

- the Fortran built-in (since Fortran 90 the `random_number`),
- the UNI function,
- Intel's RNG from the MKL library.

To test the speed and goodness of each of them a simple program was written to generate a large number of random numbers and to calculate their moments to see if they pass the necessary random number tests. (For more through tests of RNGs see e.g. Ref. [21].)

The flow of the test program is as follows:

1. initialize the RNG by using a predefined seed
2. generate n random numbers and measure the time of execution
3. reset the seed to the original value
4. regenerate the same numbers and calculate their moments

5. repeat step 1-4 for all RNG functions

n was set to a big number to have the loops running for a couple of seconds to get reasonable statistics. For the results discussed further $n = 10^{10}$ was used. There is one remarkable difference between MKL and the other two RNGs: the MKL always generate random numbers in a vector, while the others only generate random numbers one-by-one. The MKL is only available as a licensed product of Intel, which is free for non-commercial use but not available on every system, thus pre-processing directives were included in the code to use Intel-specific code only, if the code is compiled by an Intel Compiler (assuming that if the Intel Compiler is available, then MKL is also available). This makes compilation easy using the same makefile with gfortran and ifort. Since these two compilers are the easiest to access (and the Portland Compiler would be usable on the ITM Gateway, but the other two compilers on the Gateway are too old and would require to rewrite much of this test code), two sets of results were created based on these compilers. The run times of generating random numbers is shown in Table 1. An interesting result is that the gfortran performs better with random_number while ifort is better with UNI. Using ifort and MKL is superior above all, with a speed-up factor of 3 and 5 respectively, but one must notice that MKL only generates vectors, not individual numbers. This vector-generation of random numbers might be helpful during optimizations because Fortran is capable of operating on arrays and rewriting the code to do the same operation for all particles at once can result in significant save of time. I ran the same tests for all RNGs to calculate their moments and I measured these calculation times as well, the data are in Table 2. Here the same difference can be found between random_number and UNI, but the MKL slowed down which means that the generation of random numbers is really quick, but accessing them afterwards in calculations is more expensive.

Run times (s)			
	UNI	random_number	MKL
gfortran	128.064236	98.20817	-
ifort	101.780884	161.53134	33.89932

Table 1: Run times of generating 10^{10} random numbers with different compilers and methods

Results show that there is no one superior RNG for easy use in ARENA, but it depends on the compiler and the use of random numbers which one is faster. This is a case where automated standard tests can be used: to see how does the

Run times (s)			
	UNI	random_number	MKL
gfortran	171.70748	156.254188	-
ifort	116.74113	190.464967	137.329998

Table 2: Run times of generating 10^{10} random numbers and calculating their moments with different compilers and methods

execution time changes if I replace the RNG and nothing else or change the way the random numbers are accessed in the code.

2.5 Documentation

ARENA had basically no documentation. A short description of the input and output data were available and the subroutines had some descriptions in comments. This lack of documentation made it hard to find out how it works and what happens in the code and it was decided in the beginning that a new documentation is required.

The new documentation is based on Doxygen [22] which makes it possible to store the documentation in the source code and can automatically create call- and caller-graphs of the subroutines in a user-friendly linked manner. Doxygen also supports the use of $\text{\LaTeX}$ in the text so inclusion of equations and formulas is possible, making it even more readable for users to see what the subroutines and functions do.

2.6 Data structures and database storing

Data in ARENA was stored in COMMON blocks with very little structuring. Using better structured data in a scientific code improves the communication ability with other codes (either directly including other codes or indirectly through input-output files) and the maintainability of the program. The data which should be structured are the particle properties, the geometry of the tokamak, the physical environment, the Monte Carlo properties, etc. These all have different levels of structure and different purposes therefore one structure would not fit all. This problem also rose in the ITM Task Force and they created the Consistent Physical Objects to resolve it [23]. These CPOs have some resemblance with the object-oriented programming, and they represent the best approach of using object-like programming blocks in Fortran 90 (which itself does not have objects).

ARENA is aimed to work both on the ITM Gateway and as a standalone program on custom servers, so the direct inclusion of CPOs would restrict it to run only in the ITM infrastructure. The solution is to create an input-output wrapper layer which translates between the CPOs and the internal structure of ARENA (which is simpler than the CPOs) or, if run standalone, simply reads and writes ARENA's datastructure to files. In addition database storage of the data is possible and it has numerous benefits. If data is stored in a database it can be filtered by many parameters, thus finding specific data or runs is easy. Moreover, every user of the program can upload runs to the same database, and they might access other users' data as well, making run comparisons easy. Another feature of the database approach is that the users do not need to store and categorize input parameters and results on their systems, because it can be done centrally on a server which might be even accessed through a web-service for user convenience.

This storing data in a database approach and the internal restructuring of ARENA is still work in progress, mainly because it should only be done when the code is verified to produce the expected results.

3 Standard tests

Proper testing of ARENA is essential before the major upgrade of the code. This code calculates the distribution function of the electrons therefore the tests aim to verify that the description and calculation of the electrons and, especially the runaway electrons, is in agreement with the theory of the Dreicer- and avalanche-generation. The three main test are:

1. Maxwell-test: The distribution of the electrons should stay Maxwellian in the lack of external forces, see Section 3.1.
2. Dreicer-test: The Dreicer-generation has a theoretical rate calculated by Kulsrud [24] to which ARENA's result is compared, detailed in Section 3.2.
3. Avalanche-test: The growth rate from the secondary generation in ARENA can be also compared to theoretical calculations described in Section 3.3.

3.1 Maxwell-test

The aim of doing this test (which, for simplicity, is called the Maxwell-test) is to show that the Monte Carlo core of the code can keep the Maxwell distribution if there is no external electric field nor self-consistent electric field, the synchrotron radiation is neglected and the secondary generation of runaways is turned off. The electrons in a plasma like this only collide with themselves therefore the electrons will have a Maxwellian-distribution. In this setup the distribution of the electrons should keep their Maxwellian.

3.1.1 Implementation of the Maxwell-test

The Maxwell-tests require to set the external electric field to zero, the temperature to 100-1000 eV (this is low enough to avoid relativistic effects) and turn off the evolution of the electric field (which is set to zero anyway). I modified the code to be able to run automated Maxwell tests and I also needed the output of particle momenta in the initial and the last time steps for recreating the distribution functions, so I wrote a subroutine to automatically save this data.

The momenta dumps contain only the particle momenta normalized to the thermal momentum, which is processed by a Python [25] script. The script reads the momenta files, determines the scaling of the x- and y-axes, creates the histogram data, and finishes by plotting the histograms and fitting a Maxwellian-distribution to the data.

I separated runs for different time step lengths to see if they modify the result. The too long time step length would mean that the approximations used to derive the equations used in the code become invalid. However, the Monte Carlo core

takes care of splitting the long time steps into smaller time steps so that even a long time step length would produce sensible results. I, however, did not count on this feature and defined the time step lengths short enough. The chosen time step lengths were 10^{-5} , 10^{-6} , 10^{-7} and 10^{-8} , in the unit of the runaway collision time.

Since in this test the number of runaway electrons is very small, the thermal population will have most of the collision between the particles. The importance of the small time steps is fortified if the fraction of the thermal and the runaway collision times is calculated. The factor of the difference between the runaway and thermal collision times is c^3/v^3 where v is the velocity of the thermal population, defined as

$$v = \sqrt{\frac{10^3 e}{m_e}} \cdot \sqrt{T[\text{keV}]} \approx 1.32599 \cdot 10^7 \sqrt{T[\text{keV}]} \quad [\text{m/s}].$$

The electron temperature in the Maxwell tests was set to 1 keV, which is low enough to avoid relativistic effects. Using this plasma temperature the multiplication factor between the runaway and thermal collision times is

$$\frac{c^3}{v^3} = \frac{c^3}{\left(\sqrt{\frac{10^3 e}{m_e}} \cdot \sqrt{T[\text{keV}]}\right)^3} \approx \frac{(3 \cdot 10^8)^3}{\left(1.32689 \cdot 10^7 \cdot \sqrt{1}\right)^3} \approx 11500,$$

which means that the thermal time steps are 10^4 times shorter than the runaway time steps.

3.1.2 Maxwell-test results

I ran several simulations with different parameters to see the effect of changing the time step length. Four runs had 20 runaway collision times and three had 200 runaway collision times, all with different time steps. Table 3 lists the input parameters for the these runs.

The plots for the runs are shown in figures 2 for the 20 runaway collision times and figures 3 for the 200 runaway collision times.

A Maxwellian-distribution was fitted on the histograms using the function

$$f(x) = A \cdot \exp \left\{ -\frac{\sqrt{p^2 + 1} - 1}{B} \right\} \cdot p^2,$$

where the parameters are A and B . I used these fitting parameters to calculate the statistical change of the distribution in each run.

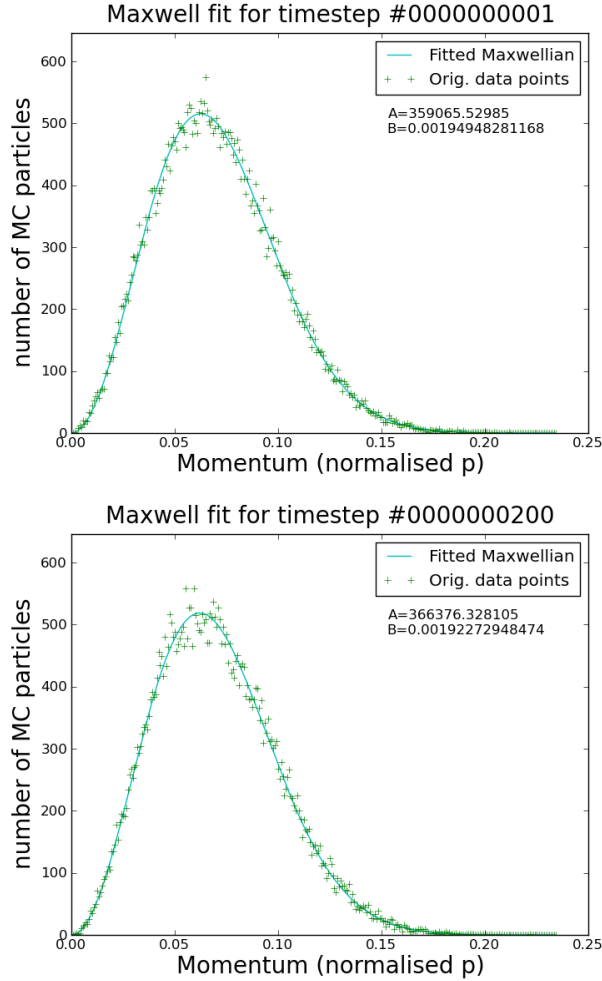


Figure 2: Initial and final momenta distribution of particles in a run of 20 runaway collision times and $1\text{E-}5$ time step length

The results of runs with 20 runaway collision times and time step lengths of 10^{-5} and 10^{-6} showed that the deviation of the fitted parameters did not exceed 5% in any cases, and this amount of deviation can be reasonably caused by the statistical behavior and the relatively low number of Monte Carlo particles ($N=10^4$). The run with 10^{-7} time step length showed worse results, the maximum deviation from the initial distribution's fitted parameters was 7.1 % for C and 5.87 % for t . Based on the previous two datasets, this might be over the statistical scattering. Combining this observation with the last dataset of 10^{-8} time step length show that something is severely went wrong on smaller time step lengths, the deviations reached 46.6 % and 29.55 %, respectively.

The runs with 200 runaway collision times show the same pattern as the ones

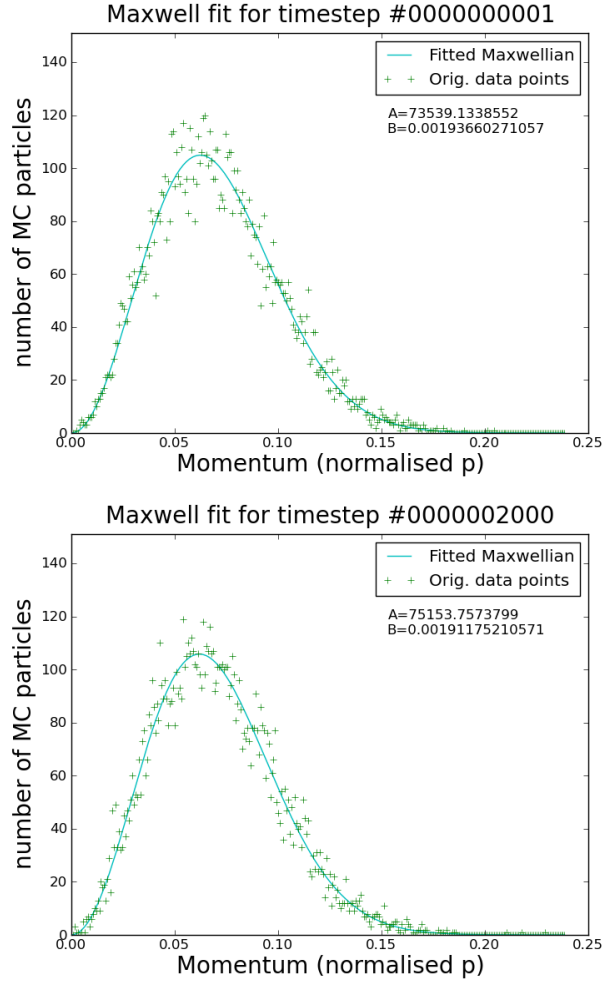


Figure 3: Initial and final momenta distribution of particles in a run of 200 runaway collision times and $1\text{E-}5$ time step length

with 20 runaway collision times, therefore it can be stated that by lengthening the runs by a factor of 10 did not change the value of deviations from the initial distribution's fitted parameters.

The candidate for this behavior is that the momenta might need to be multiplied by particle weights, but the use of weights in ARENA is only partially known as there are only two weights defined: one for thermal, one for fast particles. The reverse-engineering of the weighting functions is still ongoing.

The Maxwell-test had a positive outcome, using a meaningful time step length the Maxwellian-distribution was kept as long as 200 runaway collision times which is ≈ 18.5 ms.

	Run #1	Run #2	Run #3	Run #4	Run #5	Run #6	Run #7
simulation length (in runaway collision time)	20	20	20	20	200	200	200
number of time steps	200	2,000	20,000	200,000	2,000	20,000	200,000
time_step_len	1E-5	1E-6	1E-7	1E-8	1E-5	1E-6	1E-7
plot_interval	10	100	1,000	10,000	20	200	2,000

Table 3: Parameters of the Maxwell tests

3.2 Dreicer-generation test

In the Dreicer-generation test I calculate the growth rate (γ) of the runaway electrons as a function of time and compare it to the theoretical approximation of Kulsrud [24]. The growth rate is defined as the fraction of new runaway electrons to the total number of runaway electrons at a given time. Formally

$$\gamma = \frac{-d(\ln n_{RE}(t))}{dt}, \quad (3)$$

where n_{RE} is the number of runaway electrons.

The Dreicer-generation test set up requires a strong electric field, at least over E_C (see Eq. (1)) to have any primary generated runaway electrons, and a lot of Monte Carlo particles to have statistically meaningful results, especially with electric fields close to E_C .

3.2.1 Implementation of the Dreicer-generation test

Kulsrud used the following formula from as the theoretical approximation of the growth rate:

$$\gamma_K = 0.35E^{-\frac{3}{8}}e^{-\left[\sqrt{2/E}+1/4E\right]}, \quad (4)$$

where E is in the units of $2E_D$ and γ_K gives the fraction of electrons which run away in a thermal collision time τ_{th} .

In ARENA the growth rate is calculated from the number of runaway electrons from Eq. (3) by applying numerical differentiation:

$$\gamma = \frac{n_{RE,+1} - n_{RE,-1}}{(t_{+1} - t_{-1}) \cdot n_{RE,0}} \cdot \tau_{ra} \cdot \tau_{th}, \quad (5)$$

where n_{RE} is the number of runaway electron, t is the time in runaway collision time units. τ_{RE} , τ_{th} are the collision times of the runaway- and the thermal electrons, respectively. The $-1, +1, 0$ indexes stand for the central numerical differentiation. In Eq. (5) the collision times are required to rescale the numerical

result from the runaway collision time to the thermal collision time because the theoretical approximation in Eq. (4) is only for thermal collision times.

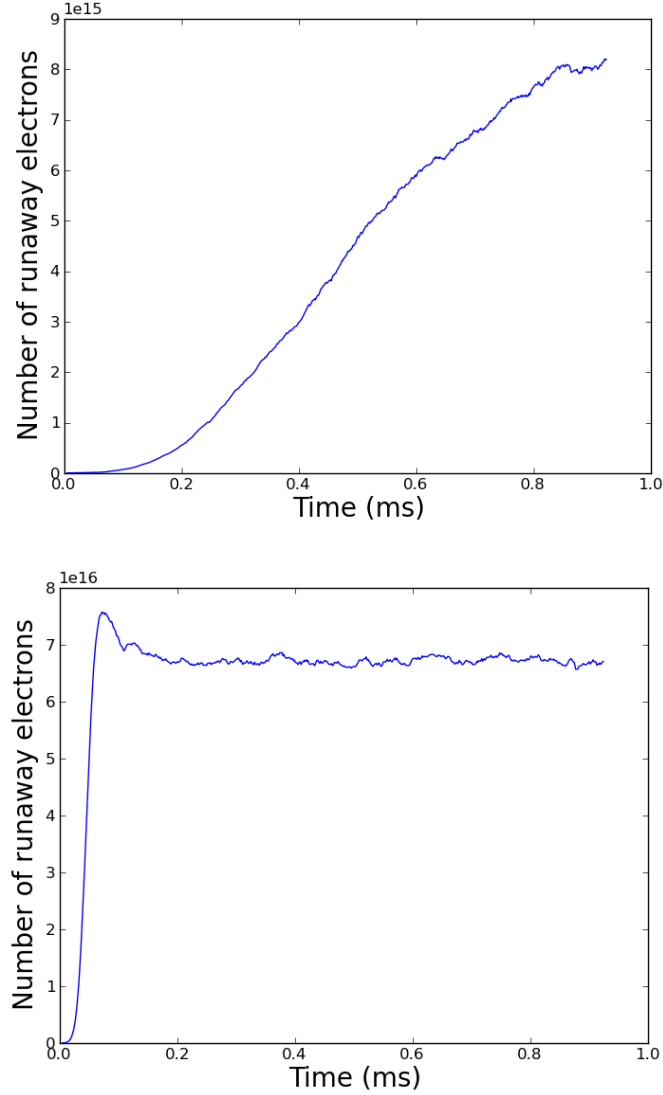


Figure 4: Evolution of the number of runaway electrons at $E_C = 9$ and $E_C = 80$.

3.2.2 Dreicer-generation test results

I ran tests for different electric field strengths from $E_C = 9$ to $E_C = 100$. Two examples of the number of runaway electrons as a function of time are shown in Figure 4.

I computed the growth rates by averaging the number of runaway electrons with a moving average of 5 points and calculating the derivative of the current according to Eq. (5).

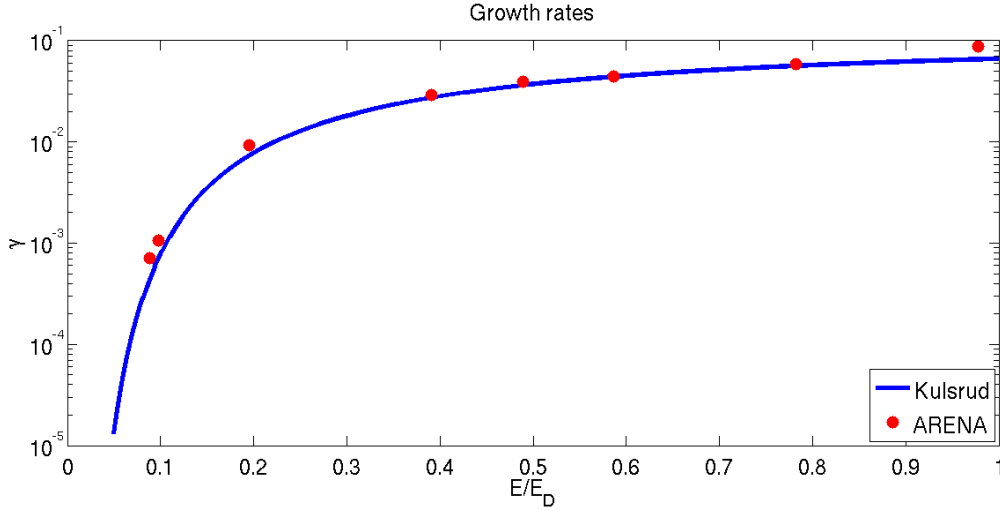


Figure 5: Calculated and theoretical growth rates of the Dreicer-generation in the function of the normalized electric field.

In the beginning of the primary generation there are very few runaway electrons. This results in high statistical scattering therefore I omitted the initial, clearly non-linear interval and I also omitted the saturated region. The cause for this saturation is yet unknown. However the analytical expression (Eq. 4) is in excellent agreement with the growth rate calculated from the linear interval over a wide range of electric field strengths, as shown in Figure 5. This positive result suggests that the Dreicer-generation model is correctly implemented in ARENA. I also carried out calculations for weaker electric fields ($E_C \leq 7$) but the statistical scattering was too high to be meaningful, and it is not trivial to reduce the scattering therefore these results are omitted for the time being.

3.3 Avalanche-test

The avalanche test is meant to show that the secondary generation is correctly implemented in ARENA. This can be done similarly to the Dreicer-test by calculating the expected growth rate and comparing it to the numerical results. This test is still work in progress and will be part of the final test suite of ARENA.

4 ITM

The aim of the Integrated Tokamak Modeling Task Force (ITM-TF) is to build a validated suite of simulation codes for ITER plasmas and to provide a software infrastructure framework for European integrated modeling activities [26] .

4.1 ITM infrastructure

The ITM provides a number of different tools and systems easing the development and integration of simulations. The cornerstones of the ITM infrastructure are:

1. Consistent Physical Objects (CPO) and their access layer (UAL):
The CPOs are physics based structures which describe a part of the properties of the plasma, such as equilibrium or core profile. The CPO is also the basic block of the ITM database in which the run data are written. The UAL is a C and Fortran library that makes it possible to use CPOs in the codes by calling simple subroutines.
2. Gforge task management system and SVN repository for every project:
The task management system helps users by integrating a repository and a bug/feature tracker in one system and by creating a shared workplace for project participants.
3. Kepler workflow editor and manager:
The Kepler workflow manager is a tool which makes it possible to connect various codes on an easy-to-use graphical interface provided the codes have the required wrappers set up. To use a code in Kepler it needs to be converted to an actor which has specially defined inputs and outputs. The ITM-TF has expanded the original Kepler Project [27] and included CPO based actors to ease its use.
4. Doxygen documentation system which is similar to the one mentioned in Section 2.

4.2 ARENA in ITM

Currently the ITM workflow is not ready to take input from ARENA, since they do not have a disruption simulation part yet. However, it is planned to create this module in the near future where ARENA could be the first code by calculating the runaway electrons' distribution function. For this integration ARENA needs to be upgraded in several ways:

- take in 2D geometry input (which also includes real machine descriptions),
- have an interface to input various numerical profiles (temperature, density, etc.),

- optimize and possibly parallelize it in order to make complex workflow calculations feasible.

The code has been already optimized to make the test run finish in reasonable times and it also passed two important tests of its physical model which is a prerequisite of integration.

5 Conclusions

The ARENA code calculates the distribution function of electrons including various generation and loss mechanisms. Large tokamaks can have a significant runaway current which, upon a disruption, may damage the vacuum vessel. The simulation of the runaway electrons is important to know and predict the distribution of the runaway electrons in tokamaks in order to aim the numerous runaway mitigation methods. The code includes two main generation methods of the runaway electrons: the primary- or Dreicer-generation and the secondary- or avalanche-generation which are described in Section 1.

The development of the original code [11, 12] finished in 2004. As the first stage of the restarted development I converted the original code from Fortran 77 to Fortran 90, as was described in Section 2. This conversion allows more sophisticated programming structures to support the development. I wrote an automated makefile to further ease the development process on various architectures. The first stage of optimization was carried out in order to conduct verification tests in reasonable time. I started the restructuring of the data storage to accommodate complex physical models and to ease the modularization of the code.

Automated tests are necessary in order to validate the physical models behind ARENA. Out of the three proposed tests two were finished with positive results as detailed in Section 3. First, the distribution function of electrons remained Maxwellian in the lack of external forces that proves that the collision operator implemented in the code is valid. Second, the Dreicer-generation growth rate agrees with the theoretical expectation [24].

The code is part of the Integrated Tokamak Modeling Task Force (ITM-TF) project which aims to build a validated suite of simulation codes for tokamak plasmas and to provide a software infrastructure framework for European integrated modeling activities [26]. The ITM integration of ARENA is important since the new, large tokamaks under construction are expected to have large number of runaway electrons. The general ITM workflows will enable ARENA to calculate runaway electron distributions for various complex setups and machines.

6 Acknowledgements

This work, supported by the European Communities under the contract of association between EURATOM and the Hungarian Academy of Sciences, was carried out within the framework of the European Fusion Development Agreement. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

The author is grateful for the support of the System International Foundation.

The author would like to thank Lars-Göran Eriksson for the help provided to understand the original code of ARENA.

References

- [1] BP. BP Statistical Review Of World Energy June 2011. Technical Report June, BP p.l.c. (2011).
- [2] Chen, F.F. *An Indispensable Truth: How Fusion Power Can Save the Planet*. Springer (2011). ISBN 978-1-4419-7819-6. <http://dx.doi.org/10.1007/978-1-4419-7820-2>.
- [3] The ITER Team. Progress in the ITER Physics Basis. Nuclear Fusion, **47** (2007)(6). <http://iopscience.iop.org/0029-5515/47/6>.
- [4] Hender, T. et al. Progress in the ITER physics basis chapter 3: MHD stability, operational limits and disruptions. Nuclear Fusion, **47** (2007)(6), S128–S202. <http://stacks.iop.org/0029-5515/47/i=6/a=S03>.
- [5] Helander, P. and Sigmar, D.J. *Collisional transport in magnetized plasmas*. Cambridge University Press, Cambridge (2002).
- [6] Papp, G. The role of magnetic perturbations in runaway electron and sawtooth dynamics. Ph.D. thesis, Chalmers Technical University, Göteborg, Sweden (2011).
- [7] Dreicer, H. Electron and ion runaway in a fully ionized gas. I. Physical Review, **115** (1959)(2), 238–249. doi:10.1103/PhysRev.115.238. http://prola.aps.org/abstract/PR/v115/i2/p238_1.
- [8] Smith, H.M. et al. Hot tail runaway electron generation in tokamak disruptions. Physics of Plasmas, **15** (2008)(7), 072502.
- [9] Fehér, T. Simulation of runaway electron generation during plasma shutdown by impurity injection. Ph.D. thesis, Chalmers University of Technology (2011).
- [10] Rosenbluth, M.N. and Putvinski, S.V. Theory for avalanche of runaway electrons in tokamaks. Nuclear Fusion, **37** (1997)(10), 1355. <http://stacks.iop.org/0029-5515/37/i=10/a=I03>.
- [11] Eriksson, L.G. and Helander, P. Simulation of runaway electrons during tokamak disruptions. Computer Physics Communications, **154** (2003)(3), 175–196. ISSN 0010-4655. doi:DOI:10.1016/S0010-4655(03)00293-5. <http://www.sciencedirect.com/science/article/B6TJ5-494HRX8-1/2/cdfd3f57818537f8be913c8d0a9c6f3d>.

- [12] Eriksson, L.G. et al. Current dynamics during disruptions in large tokamaks. *Physical Review Letters*, **92** (2004)(20), 1–4. ISSN 0031-9007. doi: 10.1103/PhysRevLett.92.205004. <http://prl.aps.org/abstract/PRL/v92/i20/e205004>.
- [13] Bernstein, I. and Baxter, D. Relativistic theory of electron cyclotron resonance heating. *Physics of Fluids*, **24** (1981), 108.
- [14] Reid, J. The advantages of Fortran 90. *Computing*, **48** (1992)(3-4), 219–238. ISSN 0010485X. doi:10.1007/BF02238635.
- [15] Martin, R.C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin series. Prentice Hall (2008). ISBN 0132350882.
- [16] Metcalf, M. convert.f90. <ftp://ftp.numerical.rl.ac.uk/pub/MandR/convert.f90> (1991).
- [17] Goto, K. Gotoblas. <http://www.tacc.utexas.edu/tacc-projects/gotoblas2> (2009).
- [18] Intel Corporation. Intel Math Kernel Library. <http://software.intel.com/en-us/intel-mkl/>.
- [19] Maidanov, S. Monte Carlo European Options Pricing Implementation Using Various Industry Library Solutions.
- [20] GNU gprof. <http://sourceware.org/binutils/docs/gprof/>.
- [21] Random Number Generation. <http://csrc.nist.gov/groups/ST/toolkit/rng/index.html>.
- [22] Van Heesch, D. Doxygen Manual. (2011).
- [23] Imbeaux, F. et al. A generic data structure for integrated modelling of tokamak physics and subsystems. *Computer Physics Communications*, **181** (2010)(6), 987–998. ISSN 00104655. doi:10.1016/j.cpc.2010.02.001.
- [24] Kulsrud, R. et al. Runaway electrons in a plasma. *Physical Review Letters*, **31** (1973)(11), 690–693.
- [25] Python Programming Language. <http://www.python.org/>.
- [26] Integrated Tokamak Modeling. <http://www.efda-itm.eu/>.
- [27] The Kepler Project. <https://kepler-project.org/>.

A Input parameters

The following table summarizes the input parameters of ARENA with some description on each one. The values represent a typical Maxwell-tests values and shown here only for convenience.

<i>Name of input parameter</i>	<i>Value</i>	<i>Description</i>
number_of_particles	10000	number of Monte Carlo particles
number_of_time_steps	200	the number of time steps
time_step_len	1e-5	the length of time step normalized to the runaway collision time
inverse_aspect_ratio_min	0.01	minimum inverse aspect ratio
inverse_aspect_ratio_max	0.011	maximum inverse aspect ratio
inverse_aspect_ratio_iron	0.4	inverse aspect ratio of the iron vessel
p_norm_max	3e7	maximum normalized momentum in the calculation (particles above this energy are considered as runaways, but are no longer in the calculation)
p_norm_min	0.02e-2	minimum momentum in the calculation (normally zero or very small)
number_of_flux_surfaces	11	number of flux surface intervals used to reconstruct profiles (density, etc)
E_applied_norm2cr	0.0	the applied electrical field normalized to the critical
Z_eff	1.0	effective charge of the plasma
n_bulk_elec_central	2.0e19	central bulk electron density
bulk_dens_prof	0.9900001	bulk density profile
bulk_dens_prof_exp	1e-7	-
T_central_electron	1.0	central electron temperature (KeV)
bulk_tem_prof	0.900001	bulk temperature profile
bulk_tem_prof_exp	1e-10	-
B0	3.0	magnetic field on the axis (T)
R_major	3.0	major radius at the axis (m)
dbbrat	0.0	normalized perturbed magnetic field
Coulomb_logarithm	18.0	Coulomb logarithm
ttedec	7.900e-3	electron temperature characteristic decay time
Te_min	0.0050	electron temperature lower limit
TEMPEB	2.2	Electron edge temperature for the linear profile
frac_init_runaways	1.0	initial fraction of runaway electrons
fwb	2.0	weighting boundary in momentum space normalized to the thermal momentum on the magnetic axis.
sw_selfcons_electric_field	0	if set to 1, the electric field is calculated self-consistently; if not, only the applied electric field is used

sw_source_term	0	if set to 1, the close collision source term is switched on; if not, no knock on particles are added in the calculation
sw_sync	0	if set to 1, the synchrotron radiation is switched on
nepsd	1	number of flux surfaces for diagnostic output
npd	10	number of momenta points for diagnostic output
NTAV	4	number of time steps, width of the moving average time window for self-consistent E-field calculation
cpmaxd	20.0	maximum normalized momentum for diagnostic output
sw_diag_out	0	switch for diagnostic output. If set to 1 the coordinates for all the Monte Carlo particles at the last time step are written to a file
do_maxwellian_test	1	switch if the calculation should be run for too low electric field
plot_interval	20	number of time steps when the particles' momenta should be written to a file

Table 4: Input parameters of ARENA with typical values for a Maxwell-test

B Collision operators

I unfolded the code of the Monte Carlo collision operator to find out how different was the modified operator compared to the original. The relativistic part of the original operator is defined in Ref. [11].

B.1 Collision operator in the paper

The following terminology is used: when $\beta = T_e/(m_0 c^2) \ll 1$ then two frequencies are defined:

$$v_D(p) = \frac{\sqrt{1+p^2}}{\tau p^3} \left[1 + Z - \beta \frac{1+2p^2}{p^2(1+p^2)} \right]$$

and

$$v_p(p) = \frac{1+p^2}{\tau p^3}$$

where $\tau = \frac{4\pi\epsilon_0^2 m_e^2 c^3}{n_e e^4 \ln \Lambda}$.

With these definitions the relativistic part of Monte Carlo operator follows (although τ_c is not defined in the paper):

$$\dot{\vec{p}}_{coll} = -v_p p + \beta \sqrt{1+p^2} \frac{2p^2-1}{\tau_c p^4} \quad (6)$$

$$A_{coll}^{pp} = \sqrt{2\beta v_p \sqrt{1+p^2}} \quad (7)$$

$$\dot{\vec{\lambda}}_{coll} = 2v_D \left(\frac{\varsigma_1}{\varsigma_2} - \frac{\lambda}{2} \right) \quad (8)$$

$$A_{coll}^{\lambda\lambda} = \sqrt{2v_D \lambda \frac{\varsigma_2}{\varsigma_1}} \quad (9)$$

B.2 Collision operator in the code

In the modified code the collision operator, which presumably was derived to contain the description both of the thermal and runaway electrons, was defined as:

$$\beta = T_e/(m_0 c^2)$$

$$v_D = \frac{\sqrt{1+p^2}}{p^3} \cdot 2 \left(1 + \frac{\ln(p+1)}{\ln \Lambda} \right) \cdot \left(1 + Z - \beta \frac{1+2p^2}{p^2(1+p^2)} \right)$$

τ is redefined as

$$\frac{1}{\tau} = 2 \left(1 + \frac{\ln(p+1)}{\ln \Lambda} \right).$$

Another v_D is defined as:

$$v_{D2} = \frac{1 + 3p^2 - 4(1 + p^2) \cdot p^4}{((0.4p_{th})^4 + p^4)^2}$$

And lastly v_p is

$$v_p = \frac{(1 + p^2) \cdot \left(1 + \frac{\ln(p+1)}{\ln \Lambda} \right) \cdot p}{(0.4p_{th})^4 + p^4}$$

With these three frequencies follows the modified Monte Carlo operator:

$$\begin{aligned} \dot{\vec{p}}_{coll} = & -p \cdot v_p + \beta \left\{ \left[\left(\frac{2v_p}{p} + v_{D2} \right) \sqrt{1 + p^2} + v_p \frac{p}{\sqrt{1 + p^2}} \right] \right. \\ & \cdot \left. \left(1 + \frac{\ln(p+1)}{\ln \Lambda} \right) + v_p \frac{\sqrt{1 + p^2}}{(p+1) \cdot \ln \Lambda} \right\} \end{aligned} \quad (10)$$

($\dot{\vec{p}}_{coll}$ is the only formula which uses v_{D2})

$$A_{coll}^{pp} = \sqrt{2v\beta\sqrt{1+p^2}} \quad (11)$$

$$\dot{\vec{\lambda}}_{coll} = v_D \cdot \left(\frac{\varsigma_2}{\varsigma_1} - \frac{\lambda}{2} \right) \quad (12)$$

$$A_{coll}^{\lambda\lambda} = \sqrt{2v_D\lambda\frac{\varsigma_2}{\varsigma_1}} \quad (13)$$