Czech Technical University
Faculty of Nuclear Sciences and Physical Engineering
Department of Physics

Monte Carlo Simulation of the Two Stream Instability

Research project

Author: **Bc. Libor Novák**
Advisor: **Prof. RNDr. Petr Kulhánek, CSc**
School year: **2008/2009**

Název práce: **Monte Carlo simulace dvousvazkové nestability**

Autor: Libor Novák

Obor: Fyzikální inženýrství

Druh práce: Výzkumný úkol

Vedoucí práce: Prof. RNDr. Petr Kulhánek, CSc

Abstrakt: Metodou Monte Carlo byla simulována dvousvazková nestabilita způsobená coulombickými interakcemi mezi jádry deuteria. Jako programovací jazyk pro 3D simulaci byl zvolen Compaq Visual Fortran Professional Edition 6.6.C. Svazek 10^4 částic byl vstřelen do 10^5 částic terče rychlostí 10^3 kms^{-1}. Jak svazek tak terč měly maxwellovské rozdělení rychlostí charakterizované teplotou 1 keV. Kinetická energie svazku se přeměnila na tepelnou energii částic. Srážka měla za následek změnu v rychlostních rozděleních ve shodě s teorií.

Klíčová slova: metoda Monte Carlo, model svazek - terč, počítačová simulace, dvousvazková nestabilita.

Title: **Monte Carlo Simulation of the Two Stream Instability**

Author: Libor Novák

Abstract: Two stream instability caused by Coulomb interactions among deuterium nuclei was simulated by the Monte Carlo method. For the 3D simulation was chosen Compaq Visual Fortran Professional Edition 6.6.C as a programming language. A beam of 10^4 particles were injected into target containing 10^5 particles at velocity 10^3 kms^{-1}. Both beam and target had Maxwellian velocities distribution characterized by the temperature 1 keV. Kinetic energy of the beam transformed to the thermal energy of the particles. The collision created a change in velocities distributions predicted by the theory.

Key words: Monte Carlo method, computer simulation, beam - target model, two stream instability.

Contents

Chapter 1

Computer Simulation

1.1 Building a Computer Model

To get solution of a physical problem using computer simulation requires performing this process [2]:

- **Problem formulation**
- **Building a model**
- **Solving the model**
- **Comparison of the model and experimental or theoretical results**

The second point is probably the most difficult part of the process. Model is ever only a reality approach. The surveyed phenomena are simplified because of there are not known some properties of it, or we have to make several hardware restrictions. Then only the comparison with experimental data can show, how reliable the model results are. This issue is old as the computer science and is mentioned as GIGO (Garbage In, Garbage Out).

1.2 Computer Simulation of Plasmas

This branch of study is based on 2 different approaches and their combination [1]:

- **Fluid simulation** - numerical solving of the magnetohydrodynamic equations
- **Kinetic simulation** - numerical solving of the plasma kinetic equations (Vlasov, Fokker - Planck, ...) or particle simulation
- **Hybrid simulation** - combination of both approaches

Particle simulations can be roughly further divided into 2 areas and their combination [3]:

- **Molecular dynamics method** - numerical solving of charged particles motions, interacting with each other and with externally fields
- **Monte Carlo method** - probability description by a random quantity
- **Hybrid simulation** - combination of both methods

Chapter 2

Monte Carlo Method

2.1 Description of the Method

Monte Carlo method is an often used computational method, which is based on random numbers use. Although it was developed by physicist in 1940s today appears in many branches of science and technology including economics or biology. It is very useful when exploitation of another tool is impossible. Random numbers carry inaccuracy in calculation natural, but sometimes would exact calculation take too long time (improved by the hardware progress).

Solving a problem by Monte Carlo method consists from these steps [2]:

- **Problem analysis and model creation**
- **Random quantity generation**
- **Random quantity transformation**
- **Previous 2 steps repeating and statistical evaluation of the results**

2.2 Random Numbers Generators

Random numbers can be generated through several ways. In Monte Carlo methods were or are used [2]:

- **Physical generators**
- **Random numbers tables**
- **Calculated random numbers**

The first way represents using a quantum physics phenomena, which produces random results. These can be recorded into tables and used later. However the speed of hardware improvement required some faster way of random numbers generation.

For this reason mathematical generators were developed. However calculated random numbers are not really random, therefore they are called as pseudo-random. At period expiration they will start to repeat. From viewpoint of computer experiment repetition, using pseudo-random numbers can be an advantage. There is no problem to get the same sequence of numbers and so do the same experiment for many times.

2.3 Pseudo-random Numbers Generators

Generally, pseudo-random numbers generators can be expressed in the form [4]:

$$n_i = f(n_{i-1}, n_{i-2}, ..., n_{i-j}). \tag{2.1}$$

It means that a new number is calculated from previous j numbers. Hence we need j initial numbers called seeds to start the evaluation.

Good generator should have these features:

- Long period
- Uniform distribution of the numbers
- Numbers are nor correlated
- High speed of evaluation

For example Linear Congruential Generators (LCG), which are described through this equation:

$$n_i = (an_{i-1} + b)\mathrm{mod}(m), \tag{2.2}$$

where a, b and m are natural constants, are not suitable for Monte Carlo methods. Opposite to LCG, some of Lagged Fibonacci Generators (LFG) can be used there. LFG combine more then 1 from previous generated numbers to gain the next one:

$$n_i = (an_{i-1} + bn_{i-2} + ...)\mathrm{mod}(m). \tag{2.3}$$

Chapter 3

Monte Carlo Coulomb Interaction

3.1 Starting the Binary Collision

Let a particle of species α with mass m_α and charge e_α has a velocity v_α^t at the time t, and the another of species β with mass m_β and charge e_β has a velocity v_β^t start the Coulomb interaction [5].

The relative velocity in the laboratory frame is defined as:

$$\boldsymbol{u}^t = \boldsymbol{v}_\alpha^t - \boldsymbol{v}_\beta^t. \tag{3.1}$$

Let $u = \sqrt{u_x^2 + u_y^2 + u_z^2}$ denotes the magnitude of the relative velocity and $u_{xy} = \sqrt{u_x^2 + u_y^2}$ the magnitude of its projection to the xy - plane. φ is the angle between x - axis and u_{xy}, and θ between z - axis and u. After the rotation $\boldsymbol{u}$ on the z - axis by the angle φ and then on the y - axis by θ will have the vector $\boldsymbol{u}$ the same direction as the z - axis.

This operation can be expressed as matrix multiplication:

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}^t = \begin{pmatrix} 0 \\ 0 \\ u \end{pmatrix}^t. \tag{3.2}$$

Product of 2 matrixes on the left side of the equation is:

$$\begin{pmatrix} \cos\theta\cos\varphi & \cos\theta\sin\varphi & -\sin\theta \\ -\sin\varphi & \cos\varphi & 0 \\ \sin\theta\cos\varphi & \sin\theta\sin\varphi & \cos\theta \end{pmatrix}. \tag{3.3}$$

Now can be expressed the inverse matrix:

$$\begin{pmatrix} \cos\theta\cos\varphi & -\sin\varphi & \sin\theta\cos\varphi \\ \cos\theta\sin\varphi & \cos\varphi & \sin\theta\sin\varphi \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}. \tag{3.4}$$

3.2 Random Angles Generation

The collision does not change the magnitude of the relative velocity. Only the direction is modified. So the vector $(0, 0, u)^t$ is rotated by random angles Φ and Θ. Φ is a random angle from interval $\langle 0, 2\pi)$. To gain the Θ angle, we can use equations:

$$\sin\Theta = \frac{2\delta}{1+\delta^2}, \tag{3.5}$$

$$1-\cos\Theta = \frac{2\delta^2}{1+\delta^2}. \tag{3.6}$$

The variable $\delta = \tan\Theta/2$ is chosen randomly with the Gaussian distribution. The average value is zero and standard deviation σ can be calculated as:

$$\sigma^2 = \frac{e_\alpha^2 e_\beta^2 n_L \lambda}{8\pi\epsilon_0^2 m_{\alpha\beta}^2 u^3}\Delta t, \tag{3.7}$$

where n_L is the lower density between n_α and n_β, ϵ_0 the permitivity of vacuum, $m_{\alpha\beta}$ the reduced mass, λ the Coulomb logarithm, and Δt the time step.

The reduced mass is defined as:

$$m_{\alpha\beta} = \frac{m_\alpha m_\beta}{m_\alpha + m_\beta}. \tag{3.8}$$

The Coulomb logarithm for mixed ion-ion collisions can be calculated according to this relation [8]:

$$\lambda_{1,2} = 23 - \ln\left[\frac{Z_1 Z_2(\mu_1+\mu_2)}{\mu_1 T_2 + \mu_2 T_1}\left(\frac{n_1 Z_1^2}{T_1} + \frac{n_2 Z_2^2}{T_2}\right)^{1/2}\right], \tag{3.9}$$

where Z is the ion charge state, T the temperature in eV, n the density in cm^{-3}, and mass μ is in the units of the proton mass.

3.3 Results of the Binary Collision

After the collision we obtain this relations for the new relative velocity:

$$\begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}^{t+\Delta t} = \begin{pmatrix} \cos\theta\cos\varphi & -\sin\varphi & \sin\theta\cos\varphi \\ \cos\theta\sin\varphi & \cos\varphi & \sin\theta\sin\varphi \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} u\sin\Theta\cos\Phi \\ u\sin\Theta\sin\Phi \\ u\cos\Theta \end{pmatrix}, \tag{3.10}$$

$$\begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}^{t+\Delta t} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}^{t} + \begin{pmatrix} \Delta u_x \\ \Delta u_y \\ \Delta u_z \end{pmatrix}. \tag{3.11}$$

Comparing these 2 equations leads to this vector of the relative velocity increase expression:

$$\Delta u_x = \frac{u_x^t u_z^t}{u_{xy}^t}\sin\Theta\cos\Phi - \frac{u_y^t u^t}{u_{xy}^t}\sin\Theta\sin\Phi - u_x^t(1-\cos\Theta), \tag{3.12}$$

$$\Delta u_y = \frac{u_y^t u_z^t}{u_{xy}^t}\sin\Theta\cos\Phi - \frac{u_x^t u^t}{u_{xy}^t}\sin\Theta\sin\Phi - u_y^t(1-\cos\Theta), \tag{3.13}$$

$$\Delta u_z = -u_{xy}^t\sin\Theta\cos\Phi - u_z^t(1-\cos\Theta). \tag{3.14}$$

In the case when $u_{xy} = 0$ we have to use this solution:

$$\Delta u_x = u\sin\Theta\cos\Phi, \tag{3.15}$$

$$\Delta u_y = u\sin\Theta\sin\Phi, \tag{3.16}$$

$$\Delta u_z = -u(1-\cos\Theta). \tag{3.17}$$

Using momentum and total energy conservation laws enable postcollision velocities evaluation:

$$\boldsymbol{v}_\alpha^{t+\Delta t} = \boldsymbol{v}_\alpha^t + \frac{m_{\alpha\beta}}{m_\alpha}\Delta\boldsymbol{u}, \tag{3.18}$$

$$\boldsymbol{v}_\beta^{t+\Delta t} = \boldsymbol{v}_\beta^t - \frac{m_{\alpha\beta}}{m_\beta}\Delta\boldsymbol{u}. \tag{3.19}$$

3.4 Gaussian Distribution

The Gaussian distribution is a continuous probability distribution. The probability density function used to be expressed in the form:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{3.20}$$

where μ is the average value and σ the standard deviation.

If the average value equals zero, two succeeding values of this distribution can be created using a random numbers generator and formulas [6]:

$$x_1 = \sigma\sqrt{-2\ln\gamma_1}\cos(2\pi\gamma_2), \tag{3.21}$$

$$x_2 = \sigma\sqrt{-2\ln\gamma_1}\sin(2\pi\gamma_2), \tag{3.22}$$

where γ_1 and γ_2 are random numbers from the interval $(0, 1\rangle$.

The Maxwellian velocity distribution for each velocity component $v_i, i = 1, 2, 3$ has parameters:

$$\mu = 0 \tag{3.23}$$

$$\sigma = \sqrt{\frac{k_\mathrm{B}T}{m}}, \tag{3.24}$$

where k_B is the Boltzmann constant, T temperature and m mass of the particles.

Temperature of a particles system can be calculated from relation [6]:

$$\langle v^2 \rangle - \langle v \rangle^2 = \frac{k_\mathrm{B}T}{m} \left(3 - \frac{8}{\pi}\right), \tag{3.25}$$

where v is the magnitude of the velocity.

For purpose of computer science, it is suitable to use this recurrent formula for average values computation:

$$\langle a_{n+1} \rangle = \langle a_n \rangle \frac{n}{n+1} + \frac{a_{n+1}}{n+1} \tag{3.26}$$

Chapter 4

Two stream instability

4.1 Theory

In a spatial unlimited plasma compound of several components placed in electrical field can emerge the stream instability described by the dispersion relation [7]:

$$\sum_{\alpha} \frac{\omega_{p\alpha}^2}{(\omega - \boldsymbol{k} \cdot \boldsymbol{u}_{0\alpha})^2} = 1, \tag{4.1}$$

where

$$\omega_{p\alpha}^2 = \frac{n_{0\alpha} Q_{\alpha}^2}{m_{\alpha} \epsilon_0} \tag{4.2}$$

is the plasma frequency of particles of species α, $\boldsymbol{u}_{0\alpha}$ the initial velocity of the straightforward motion, ω the frequency, $\boldsymbol{k}$ the wave vector, $n_{0\alpha}$ the initial density, Q_{α} the charge, $m\alpha$ the mass, and ϵ_0 the permitivity of vacuum.

In the case of two stream instability, when a beam is injected to the target maxwellian plasma which does not move the target particle's velocity distribution function changes as shown in Figure 4.1 [9].

4.2 Simulation Set-up

As the programming language for the 3D simulation was chosen Compaq Visual Fortran Professional Edition 6.6.C which is suitable for numeric computation. The source code is enclosed as Appendix.

The spatial unlimited target plasma was simulated through a cube with periodical boundary conditions. The side of the cube was 10^{-5} m long and contained 10^5 nuclei of deuterium. The beam of 10^4 deuterium particles at initial velocity 10^3 kms^{-1} was aimed to the centre of one cube wall. Both beam and target had Maxwellian velocity distribution with temperature 1 keV which were implemented according to the formula 3.21.

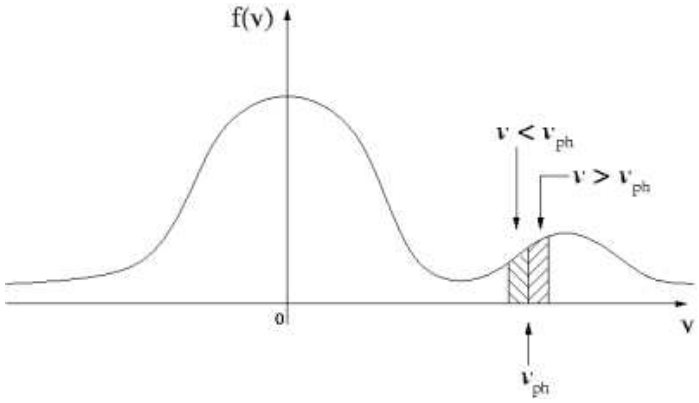

Figure 4.1: Two Stream Instability

As pseudo-random numbers generator was used the generator included in the programming language which is usual for the Monte Carlo simulations. Particles moved with the time step 10^{-12} s. In each time step the Coulomb interactions among beam and target particles described in chapter 3 were proceeded. The Coulomb logarithm was calculated from the formula 3.9 on the start of the simulation only as well as the Debye length. As a condition for the realization of a collision was empirically set the distance of $\lambda_{\mathrm{De}}/100$. The Debye length was calculated according to the formula [7]:

$$\lambda_{\mathrm{De}} = \sqrt{\frac{\epsilon_0 k_{\mathrm{B}} T}{n_0 Q^2}}, \tag{4.3}$$

where ϵ_0 is the permitivity of vacuum, k_{B} the Boltzmann constant, T temperature, Q charge, and n_0 the initial density of the target particles.

Collisions in separated Maxwellians were neglected because of they do not change the shape of the distribution function. In each time step all the particles moved evenly straightforward. At the end of the computation temperatures of the Maxwellians were calculated using the formula 3.25.

4.3 Simulation Results

In figures 4.2 and 4.4 are shown the initial target Maxwellian distributions in the beam move direction. The horizontal axis corresponds to the normalized velocities, the vertical to the rate of particles.

After performance of 100 time steps when the beam gave up to interact with the target the velocities distributions were plotted again. The results are in figures 4.3 and 4.5. It is obvious the agreement with the theory. Kinetic energy of the beam transformed to the thermal energy both beam and target particles. The beam heated up to 3 keV, the target to 2 keV.

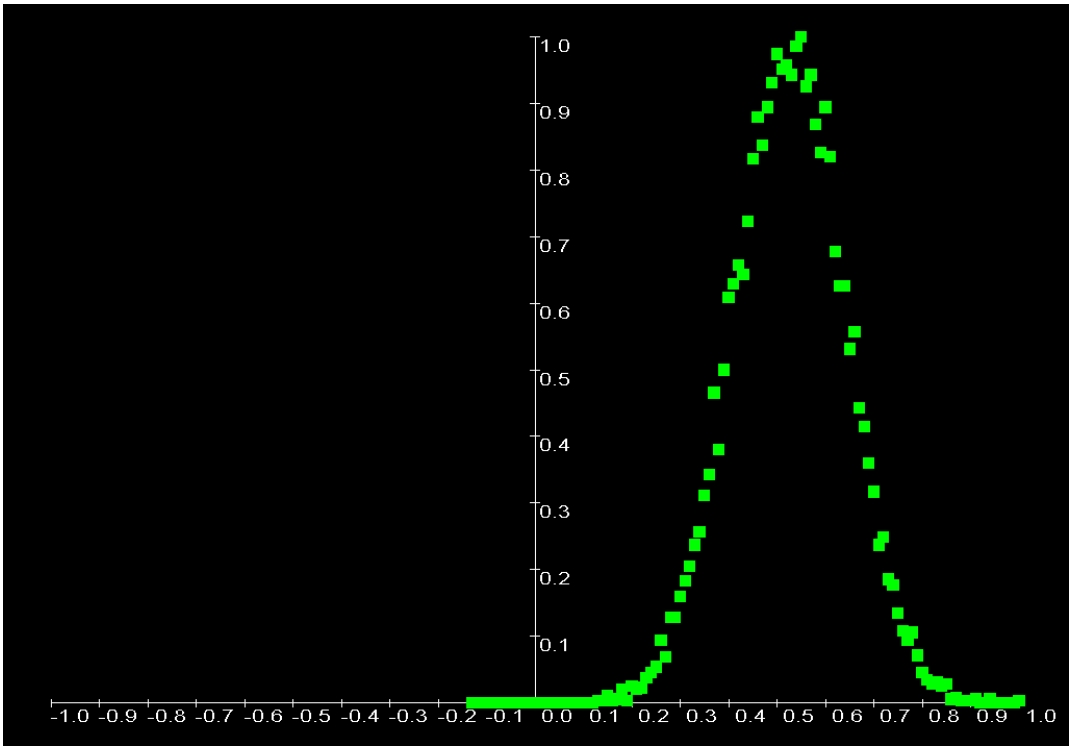

Figure 4.2: Initial beam velocities distribution

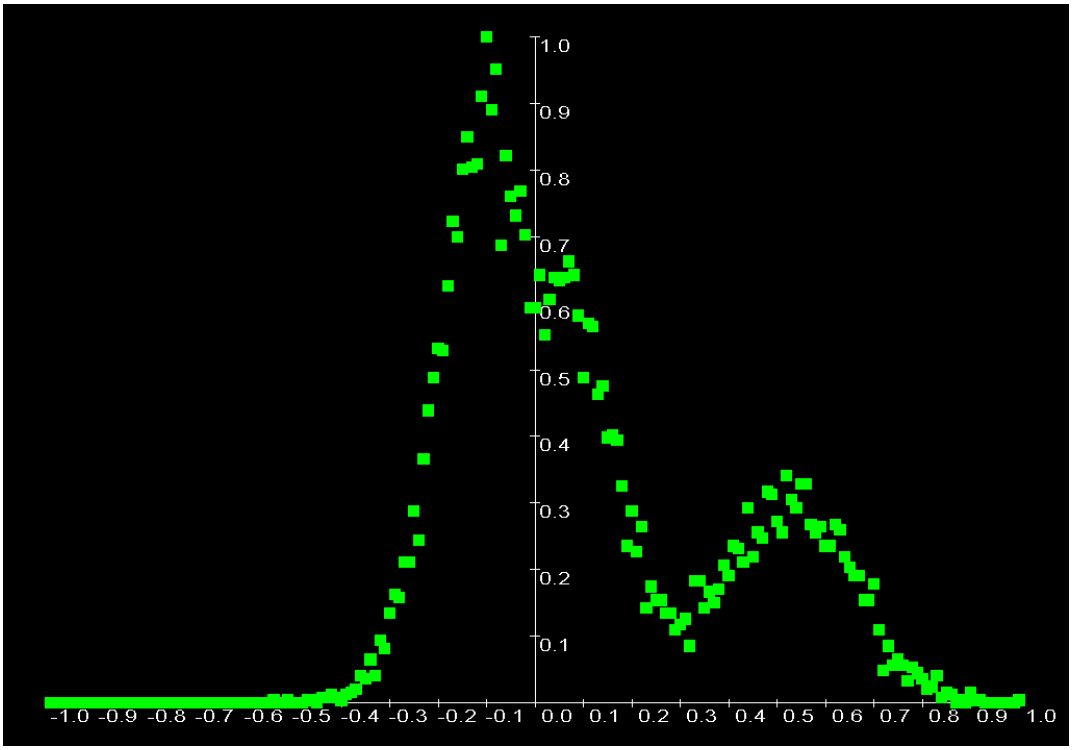

Figure 4.3: Final beam velocities distribution

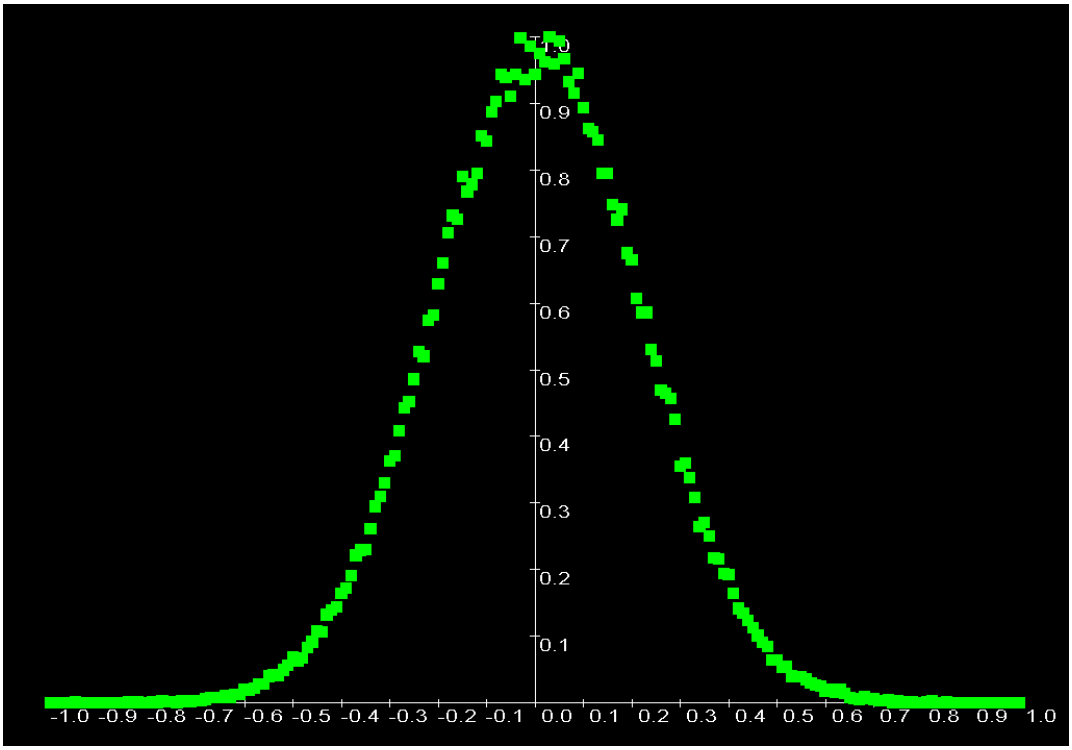

Figure 4.4: Initial target velocities distribution

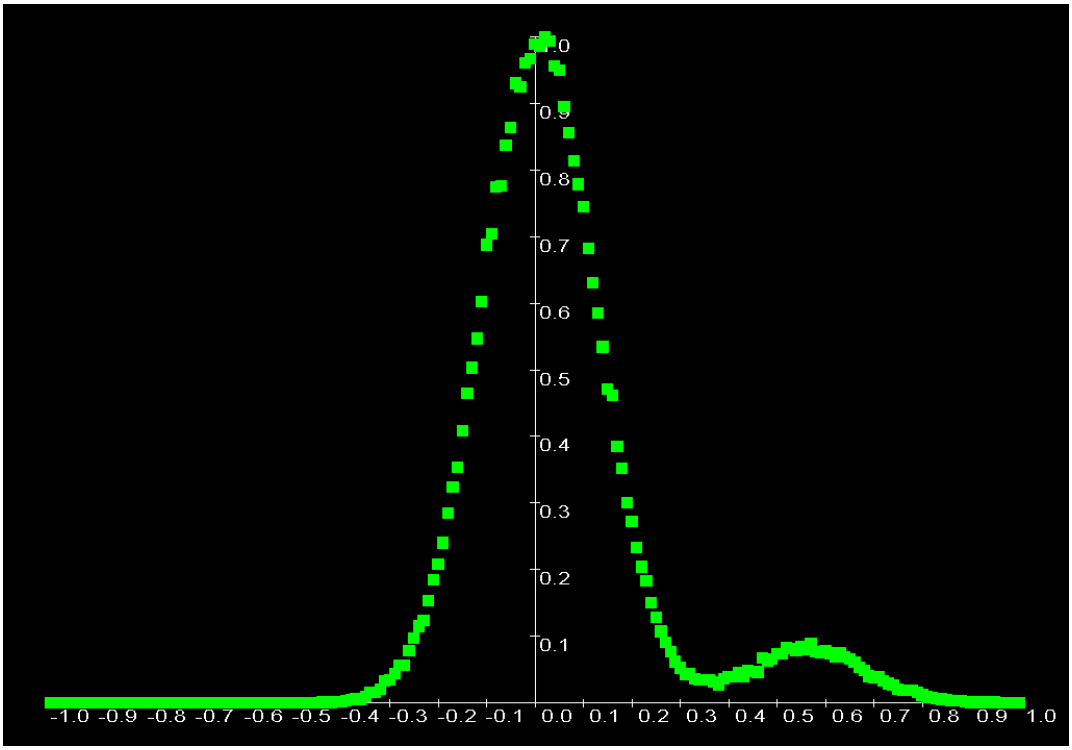

Figure 4.5: Final target velocities distribution

Appendix A

Source Code

```
PROGRAM Two Beams Instability
use Msflib
implicit none

integer N_t, N_b, time, cells
real n_0, v_0, constmaxw_t, constmaxw_b, T_t, T_b, dist, lambda, dt, coulomblog, cube
real, allocatable, dimension(:,:) :: P_t, P_b
real(8) pi, m_D, q_e, permitivity, const_Moca

call Init_const
call Init_target
call Init_beam

do time = 1,100
    write(*,*) time
    call Interaction
    call Move
    call Bound
end do

write(*,*) Temp(P_b,N_b)
write(*,*) Temp(P_t,N_t)

call Distribution_t
call Distribution_b
```

```
CONTAINS

Subroutine Init_const
implicit none

N_t = 1E5 !amount of target particles
N_b = 1E4 !amount of beam particles
v_0 = 1E6 !initial beam velocity in ms-1
T_t = 1E3 !initial target particles temperature in eV
T_b = 1E3 !initial beam particles temperature in eV
cube = 1E-5 !target area proportion in m
n_0 = N_t/cube**3 !target density
dt = 1E-12 !time step in s

Pi = 3.141592653
m_D = (1.660538E-27) * 2
q_e = 1.602189E-19
permitivity = 8.854188E-12

lambda = sqrt((permitivity*T_t)/(n_0*q_e)) !Debye length,T[eV] = T[J]/q_e
constmaxw_t = sqrt(T_t*q_e/m_D)
constmaxw_b = sqrt(T_b*q_e/m_D)
coulomblog = 23 - log(2*(sqrt((1E-6)*n_0*(1/T_t + 1/T_b)))/(T_t + T_b))
const_Moca = sqrt(n_0*dt/(2*PI*(m_D*permitivity)**2))
cells = 100

allocate(P_t(1:6,1:N_t),P_b(1:6,1:N_b)) !arrays for positions and velocities

end subroutine Init_const

!*************************************************************************

Subroutine Init_target
implicit none
integer a
real x,y,z

do a=1,N_t

    call random_number(x)
    call random_number(y)
    call random_number(z)
```

```
    P_t(1,a) = x*cube !target particles positions
    P_t(2,a) = y*cube
    P_t(3,a) = z*cube

    call random_number(x)
    call random_number(y)

    P_t(4,a) = sqrt(-2*log(1-x))*cos(2*PI*y)*constmaxw_t !target particles velocities

    call random_number(x)
    call random_number(y)

    P_t(5,a) = sqrt(-2*log(1-x))*cos(2*PI*y)*constmaxw_t

    call random_number(x)
    call random_number(y)

    P_t(6,a) = sqrt(-2*log(1-x))*cos(2*PI*y)*constmaxw_t

end do

end subroutine Init_target

!*************************************************************************

Subroutine Init_beam
implicit none
integer a
real x,y,z

do a=1,N_b

    call random_number(x)
    call random_number(y)
    call random_number(z)

    P_b(1,a) = x*cube !beam particles positions
    P_b(2,a) = y*cube
    P_b(3,a) = z*cube

    call random_number(x)
    call random_number(y)
```

```
    P_b(4,a) = sqrt(-2*log(1-x))*cos(2*PI*y)*constmaxw_b !beam particles velocities

    call random_number(x)
    call random_number(y)

    P_b(5,a) = sqrt(-2*log(1-x))*cos(2*PI*y)*constmaxw_b

    call random_number(x)
    call random_number(y)

    P_b(6,a) = sqrt(-2*log(1-x))*cos(2*PI*y)*constmaxw_b

end do

end subroutine Init_beam

!*************************************************************************

real Function Temp(L,ind) !temperature evaluation of particles array
implicit none
integer i,j,k,ind
real L(1:6,1:ind)
real v(ind),v2(ind)
real Avv,Avv2,Avvx,Avvy,Avvz

    Avvx = L(4,1) !average value of v_x component
do k=1,ind-1
    Avvx = Avvx*k/(k+1) + L(4,k+1)/(k+1)
end do

    Avvy = L(5,1) !average value of v_y component
do k=1,ind-1
    Avvy = Avvy*k/(k+1) + L(5,k+1)/(k+1)
end do

    Avvz = L(6,1) !average value of v_z component
do k=1,ind-1
    Avvz = Avvz*k/(k+1) + L(6,k+1)/(k+1)
end do

do i=1,ind
    v2(i) = (L(4,i)-Avvx)**2 + (L(5,i)-Avvy)**2 + (L(6,i)-Avvz)**2 !squared magnitude
of chaotic velocity
```

```
    v(i) = sqrt(v2(i)) !magnitude of chaotic velocity
end do

    Avv = v(1) !average value of magnitude of chaotic velocity
    Avv2 = v2(1) !average value of squared magnitude of chaotic velocity
do j=1,ind-1
    Avv = Avv*j/(j+1) + v(j+1)/(j+1)
    Avv2 = Avv2*j/(j+1) + v2(j+1)/(j+1)
end do

Temp = (Avv2-Avv**2)*m_D/(3-8/PI)/q_e !Temperature in eV

end Function Temp

!*************************************************************************

Subroutine Move
implicit none
integer a,b

do a=1,N_b
    P_b(1,a) = P_b(1,a) + P_b(4,a)*dt
    P_b(2,a) = P_b(2,a) + P_b(5,a)*dt
    P_b(3,a) = P_b(3,a) + P_b(6,a)*dt
end do

do b=1,N_t
    P_t(1,b) = P_t(1,b) + P_t(4,b)*dt
    P_t(2,b) = P_t(2,b) + P_t(5,b)*dt
    P_t(3,b) = P_t(3,b) + P_t(6,b)*dt
end do

end Subroutine Move

!*************************************************************************

Subroutine Interaction
implicit none
integer a,b,help
real P(1:3)
real D,U,Uxy,sigma,x,y,z,delta,sintheta,costheta,sinfi,cosfi
real deltaux,deltauy,deltauz
```

```
P(1) = 0; P(2) = 0; P(3) = 0; D = 0; U = 0; Uxy = 0; help = 0

do a=1,N_b
do b=1,N_t

    P(1) = P_b(1,a)-P_t(1,b)
    P(2) = P_b(2,a)-P_t(2,b)
    P(3) = P_b(3,a)-P_t(3,b)

if ((abs(P(1)) <= lambda/100) .or. (abs(P(2)) <= lambda/100) .or. (abs(P(3)) <=
lambda/100)) then

    D = sqrt(P(1)**2 + P(2)**2 + P(3)**2)
    P(1) = 0; P(2) = 0; P(3) = 0

if (D <= lambda/100) then

    deltaux = 0; deltauy = 0; deltauz = 0; Uxy = 0
    help = help + 1

    P(1) = P_b(4,a)-P_t(4,b) !Array P used for another purpose...mutual velocity
    P(2) = P_b(5,a)-P_t(5,b)
    P(3) = P_b(6,a)-P_t(6,b)

    U = sqrt(P(1)**2 + P(2)**2 + P(3)**2)
    Uxy = sqrt(P(1)**2 + P(2)**2)

    call random_number(x)
    call random_number(y)
    call random_number(z)

    sigma = const_moca*sqrt(coulomblog/(U**3))
    delta = sqrt(-2*log(1-x))*cos(2*PI*y)*sigma

    sintheta = 2*delta/(1 + delta**2)
    costheta = 1 - 2*delta*delta/(1 + delta**2)
    sinfi = 2*z - 1
    cosfi = cos(asin(sinfi))

if (Uxy == 0) then

    deltaux = U*sintheta*cosfi
    deltauy = U*sintheta*sinfi
```

```
    deltauz = -U*(1 - costheta)
else
    deltaux = P(1)*P(3)*sintheta*cosfi/Uxy - P(2)*U*sintheta*sinfi/Uxy - P(1)*(1-costheta)
    deltauy = P(2)*P(3)*sintheta*cosfi/Uxy + P(1)*U*sintheta*sinfi/Uxy - P(2)*(1-costheta)
    deltauz = -Uxy*sintheta*cosfi - P(3)*(1-costheta)

end if

    P_b(4,a) = P_b(4,a) + deltaux/2
    P_b(5,a) = P_b(5,a) + deltauy/2
    P_b(6,a) = P_b(6,a) + deltauz/2

    P_t(4,b) = P_t(4,b) - deltaux/2
    P_t(5,b) = P_t(5,b) - deltauy/2
    P_t(6,b) = P_t(6,b) - deltauz/2

end if
end if
end do
end do

write(*,*) help

end Subroutine Interaction

!*********************************************************************************

Subroutine Bound !periodical boundary conditions
implicit none
integer a

do a = 1,N_t

if (P_t(1,a) < 0) then
    P_t(1,a) = P_t(1,a) + cube
end if

if (P_t(2,a) < 0) then
    P_t(2,a) = P_t(2,a) + cube
end if

if (P_t(3,a) < 0) then
    P_t(3,a) = P_t(3,a) + cube
```

```
end if

if (P_t(1,a) > cube) then
    P_t(1,a) = P_t(1,a) - cube
end if

if (P_t(2,a) > cube) then
    P_t(2,a) = P_t(2,a) - cube
end if

if (P_t(3,a) > cube) then
    P_t(3,a) = P_t(3,a) - cube
end if

end do

end Subroutine Bound

!************************************************************************

Subroutine Distribution_t
implicit none
integer a
real b,max
real Array(-cells:cells)

do a = -cells,cells
    Array(a) = 0
end do

    max = abs(P_t(4,1))
do a = 2,N_t
    b = abs(P_t(4,a))
if ( b > max) then
    max = b
end if
end do

do a = 1,N_t
    Array(ceiling(cells*P_t(4,a)/max)) = Array(ceiling(cells*P_t(4,a)/max)) + 1
end do

Array = Array / maxval(Array)
```

```
open(1,FILE='Distribution_t.txt')
write(1,100) (a,Array(i),a = -cells, cells)
100 format (I6,I6)

end Subroutine Distribution_t

!*************************************************************************

Subroutine Distribution_b
implicit none
integer a
real b,max
real Array(-cells:cells)

do a = -cells,cells
    Array(a) = 0
end do

    max = abs(P_b(4,1))
do a = 2,N_b
    b = abs(P_b(4,a))
if ( b > max) then
    max = b
end if
end do

do a = 1,N_b
    Array(ceiling(cells*P_b(4,a)/max)) = Array(ceiling(cells*P_b(4,a)/max)) + 1
end do

Array = Array / maxval(Array)

open(1,FILE='Distribution_b.txt')
write(1,100) (a,Array(i),a = -cells, cells)
100 format (I6,I6)

end Subroutine Distribution_b

END PROGRAM Two Beams Instability
```

Bibliography

[1] Birdsal C.K., Langdon A.B.: *Plasma physics via Computer Simulation*, Institute of Physics Publishing 1991, ISBN 0 7503 0117 1

[2] Hrach R.: *Počítačová fyzika I*, study text for UJEP

[3] Nezbeda I., Kolafa J., Kotrla M.: *Úvod do počítačových simulací*, Karolinum, Praha 1998

[4] Kaňok M.: *Generátory pseudonáhodných čísiel v metódach Monte Carlo*, Československý časopis pro fyziku, 6/2008

[5] Abe H., Takizuka T.: *A binary collisional model for plasma simulation with a particle code*, Journal of computational physics 25, 205-219, 1977

[6] Kulhánek P.: *Statistická fyzika*, PhD study text, FEL ČVUT 2002, http://www.aldebaran.cz/studium/statistika.pdf

[7] Kulhánek P.: *Teorie plazmatu*, study text, FJFI ČVUT 2008, http://www.aldebaran.cz/studium/fpla.pdf

[8] Huba J.D.: *NRL plasma formulary 2007*, Beam Physics Branch, Plasma Physics Division, Naval Research Laboratory, Washington DC 20375

[9] *http://en.wikipedia.org/wiki/File:Bump_on_tail_dist.png*

Acknowledgement:

This research has been supported by the grant IAA101210801: 'Simulations of the DD fusion reactions' of the Grant Agency of the Academy of Sciences of the Czech Republic.