# Effect of the convective electric field on the ion number density around a low activity comet

Simon Alinder
900531-5271
Supervisor: Erik Vigren
Advanced Physics - Project Course, 5.0 c

December 19, 2017

## ABSTRACT

Vigren et al. (2015) presents an integral expression to calculate the ion number density around a low activity comet immersed in the solar wind's convective electric field. A certain parameter of the integral takes values of either 1 or 0 depending on whether a corresponding ion trajectory is feasible or not. The criteria used in the paper has been found not to be strict enough, yielding overestimated ion number densities in the cometary wake. The present project finds two new options for the criteria, one analytical and one numerical. The new numerical condition is tested in the same computations done in the original paper and compares the results of the old and new criteria. The new condition is found to correct the previous error.

# 1 Introduction

The comet 67P/Churyumov–Gerasimenko was visited by the ESA spacecraft *Rosetta* from 2014 to 2016. While at the comet, the spacecraft collected a wealth of data and many papers were, and are still being, published from its findings.

This project is an effort to correct an error in Vigren et. al. (2015). As part of their paper they presented an analytic expression for calculating the ion number density, $n_I$, at position $(x, y, z)$ given a radially expanding neutral atmosphere (expansion velocity $u_{\mathrm{N}}$) subjected to a constant photoionization frequency $f$ and a constant electric field along the $z$ axis forcing ions to be accelerated with acceleration factor $a$ in the negative $z$ direction. Their Eq.(7) reads:

$$n_{\mathrm{I}}(x,\, y,\, z) = \int_{\tau=0}^{\infty} f n_{\mathrm{N}} \left( x, y, z - \frac{a\tau^2}{2} \right) C_\tau d\tau \tag{1}$$

where $n_{\mathrm{N}}(x, y, z - at^2/2)$ is the neutral number density at position $(x, y, z - at^2/2)$ (note that $a$ is negative) and $C_\tau$ takes a value of 0 or 1 according to

$$C_\tau(x, y, z, a, \tau, r_C, u_{\mathrm{N}}) = 1 \text{ if } r' - r_c \geq u_{\mathrm{N}}\tau \,(\text{else } C_\tau = 0). \tag{2}$$

$C_\tau$ is meant to indicate if a particular ion trajectory is allowed or would pass through the comet nucleus, and thus not be possible and should not contribute to the number density of particles. The issue is that the condition on $C_\tau$ placed in Eq.2 is not strict enough. This report investigates several other approaches, their advantages and disadvantages in an attempt to find a better substitute.

The solar wind carries the interplanetary magnetic field. In the reference frame of the comet, newly created cometary ions will thus experience acceleration by the solar wind convective electric field, given by $\vec{E} = -\vec{u}_{SW} \times \vec{B}_{IMF}$ where $\vec{u}_{SW}$ is the velocity of the solar wind (typically $\sim 400\mathrm{kms}^{-1}$) and $\vec{B}_{IMF}$ is the interplanetary magnetic field (typically around 5 nT at 1 AU and decaying with the inverse of the heliocentric distance). For the scales relevant to our problem the electric field is of main relevance. It may be of interest to note that the effect of the electric field can be seen as a deformation of the otherwise straight particle trajectories. This deformation is simple as the acceleration is constant and homogeneous. This means that the distance between a pair of particles (formed at the same time) as a function of time will be the same regardless of whether the (acceleration) deformation is present or not.

The model is only valid for a very low activity comet. If the comet is more active, the use of $\vec{u}_{SW} \times \vec{B}_{IMF}$ is no longer a good approximation for the electric field as the velocity in the equation really should involve the expectation value of the plasma speed computed as $(n_{SW}u_{SW} + n_I u_I)/(n_{SW} + n_I)$ where $n_I$ and $u_I$ are the cometary ion density and drift velocity, respectively, and $n_{SW}$ and $u_{SW}$ are the solar wind density and velocity, respectively (Galand et al., 2016, see their Section 5). Moreover, the approach applied here neglect electric fields of other origin, e.g., the ambipolar electric field, set up by the electron pressure gradient force.

This project was mostly done in MATLAB.

## 2 Method

In the above equations, $C_\tau$ contains the variable $\tau$ which is the lifetime of the ion. In the below text, $\tau$ is treated as an explicit variable. This way the problem is optimized for numerical solving of the integral (1).

In order to fit the project inside a 5 credit course the problem used several simplifications. The considered problem can be stated as such: Start with a 2D plane and a circle, centered on the origin, representing the comet nucleus. Neutral particles are created at the edge of a circle and start traveling radially outward at constant speed. At some point the particle gets ionized, and is then affected by the electric field which acts with a downward (towards negative z-values) force on it. This ion then gets detected at a specific point. The goal is then to compute if the trajectory (which will always be part of a parabola) of the ion is allowed or not, given the coordinates of the detection, with the time from ionization to detection being a free variable. The trajectory is not allowed if the parabola at any point crosses the circle. This problem is thus simplified to a purely geometric problem of checking if a parabola intersects a circle. The challenge is to do this in an as computationally efficient way as possible. The simplification to 2 dimensions is valid because the problem is cylindrically symmetrical, meaning that values at $(x_1, y_1, z_1)$ and $(x_2, y_2, z_1)$ will be identical if $x_1^2 + y_1^2 = x_2^2 + y_2^2$. To this end the coordinate $\rho$ is introduced as $\rho = \sqrt{x^2 + y^2}$.

Two different methods were developed for solving this problem. The first one was the obvious solution of solving the equation system containing the equation of the parabola and the equation of the circle and checking if there are any crossings. The second method looked at the distance between several points on the parabola and the origin. If the distance of any point was less than the radius of the comet circle, the trajectory is not allowed.

In both methods the starting position for each ion is determined from the detection position and the life-time of the ion. When solving numerically we need to limit ourselves to certain values for the life-time. In the simulation below we consider values between 0.01 and 10 s. For a given $(\rho, z, t)$-tuplet we can identify the position where the particle would be in the absence of acceleration along $z$ as $(\rho', z') = (\rho, z + \frac{at^2}{2})$ (we treat $a$ as positive). As $(\rho', z')$ should be along the same radial as the starting position we can find the initial velocity in the $\rho-$ and $z-$direction directly from $(\rho', z')$ and $u$, the initial radial velocity, i.e. $u_\rho = u\frac{\rho'}{r'}$ and $u_z = \frac{uz'}{r'}$. The starting position of an ion $(\rho_0, z_0)$ is obtained as $\rho_0 = \rho' - u_\rho t$ and $z_0 = z' - u_z t$.

With the starting position known, both methods use similar systems for checking if a trajectory needs to be tested. For example if an ion starts well under the comet, it is not possible for the trajectory to collide with the comet so it is automatically accepted while if the trajectory starts inside the comet it is automatically rejected, no long computation required. Next the coefficients for the parabola is computed. The first method requires the actual equation, the second only needs a fast way of determining the $z$-coordinates of points on the parabola, given the $\rho$-coordinates.

The first method uses the `solve` command in MATLAB to solve the equation system containing the parabola and the circle. If any solutions are found they are checked to see if they are relevant, for example the relevant crossing

have to be in between the starting and final values of the trajectory. If any solution to the equation system fulfill all the criteria to be the point of collision between the ion and comet, the trajectory is not allowed, otherwise it is allowed. The second method is very simple. It selects a number of points between the starting and final $\rho$-coordinates, at least 100 but more if the distance the ion travels is large. It then computes the $z$-coordinates of these points by solving the equation of the parabola with the selected $\rho$-values. Then the distance of every point to the origin is found using the Pythagorean theorem. If any distance is equal to or smaller than the radius of the comet, the trajectory is not allowed, otherwise it is.

Both methods have advantages as well as disadvantages. The advantages of the first method is that is always going to give a correct answer and it is easy for a human to do by hand. The disadvantage of this method is that it is comparatively very slow. In MATLAB it uses the symbolic toolbox and the `solve` command, which is a computationally expensive method. The advantage of the second method is that it is very fast and very simple, requiring only a few lines of code and very little time. The main disadvantage of it is that there is no mathematical guarantee that it will give the correct result, however by using adaptive discretization this risk can be made arbitrarily small.

# 3 Results and Discussion

The relative speed of computation of the two methods are shown in the table (1) below.

| | Method 1 | Method 2 |
|---|---|---|
| 8020000 trajectories | 2318 | 45 |
| 8200 trajectories | 515 | 0.06 |

Table 1: Time in seconds

The required time does not grow linearly with number of points tested. Both methods have checks to determine if a full test is needed or if the trajectory can simply be approved or discarded. This means that adding a lot of obvious trajectories will not increase the computational time by much.

From table (1) it is obvious that method 2 is preferable for computation as it is much faster. There is however a very small risk that method 2 will give a false positive, if the only part of the trajectory that intersects the circle is small enough that no tested point is placed on it. This can only happen when the trajectory only barely hits the comet. In all tests the two methods agreed on all trajectories. Method 1 should always give the correct answer as there is no discretization or guessing used.

Here follows figures illustrating the effect of implementing the updated criteria for $C_\tau$ in Eq.1. For these simulations we have utilized the same parameters as in Vigren et al. (2015), i.e., $u = 650\ m/s$, $a = 2.6647 \times 10^3\ m/s^2$ (in terms of magnitude), $Q = 10^{27}\ s^{-1}$ (the molecular outgassing rate), $f = 4.68 \times 10^{-7}\ s^{-1}$ (the ionization frequency), and $r_c = 2$ km (the radius of the nucleus). Only method 2 was used for generating the figures. This is because method 2 is much faster and more adapted for numerical computation. The figures show

the number density of ions in a segment of the plane with the comet nucleus in the origin. Fig.1 show the results if Eq.1 is computed with the criteria for $C_\tau$ being the second method described in this article. Fig.2 shows the results if Eq.1 is computed with the criteria for $C_\tau$ being as described in Eq.2. Fig.2 has had the densities in the area corresponding with the nucleus set to 0 after the computation is complete, as this was done in the original paper. Finally, Fig.3 shows the ratio of the ion densities (data in Fig.1 divided by data in Fig.2).

The most notable change from the old method to the new is the "shadow" under the nucleus, seen clearly in Fig.3. This is because the old method allows trajectories that start above the nucleus and end below it, going through it, the new method does not allow this. The number density in this area is reduced by about 30%. Also the density inside of the nucleus is automatically set to 0 by the new method.

# 4 Summary

A problem associated with nucleus blocking when estimating the number density around a low activity comet embedded in the solar wind's convective electric field was reduced to a geometry problem of finding if a parabola intersects a circle. Two methods for this problem was developed and tested. One is slow and intuitive, the other is fast but has a small risk of false positives. The updated criterion for trajectory removal affected the calculated ion number densities in the wake of the comet, reducing the densities by ∼30% compared to the original used in Vigren et al. (2015).

# 5 References


E. Vigran et. al., "On the electron-to-neutral number density ratio in the coma of comet 67p/churyumov–gerasimenko: guiding expression and sources for deviations". The Astrophysical Journal, 812:54 (9pp), 2015 October 10 doi:10.1088/0004-637X/812/1/54

M. Galand et. al., "Ionospheric plasma of comet 67P probed by *Rosetta* at 3 au from the Sun", MNRAS 462, S331–S351 (2016) doi:10.1093/mnras/stw2891

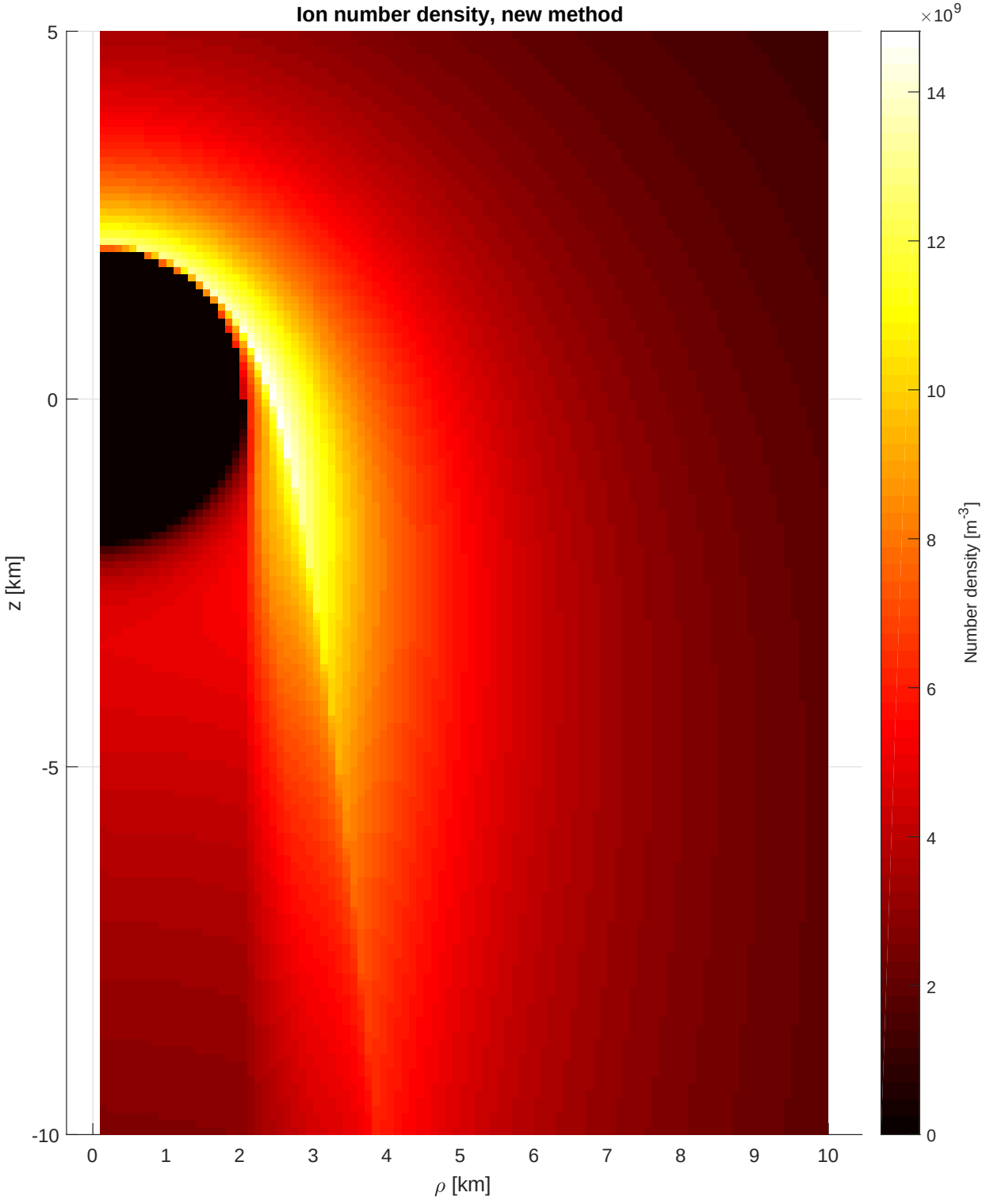




Figure 1: Result of Eq.1 using new cirteria for $C_\tau$ described in the report and input parameters as specified in the final paragraph of section 3.

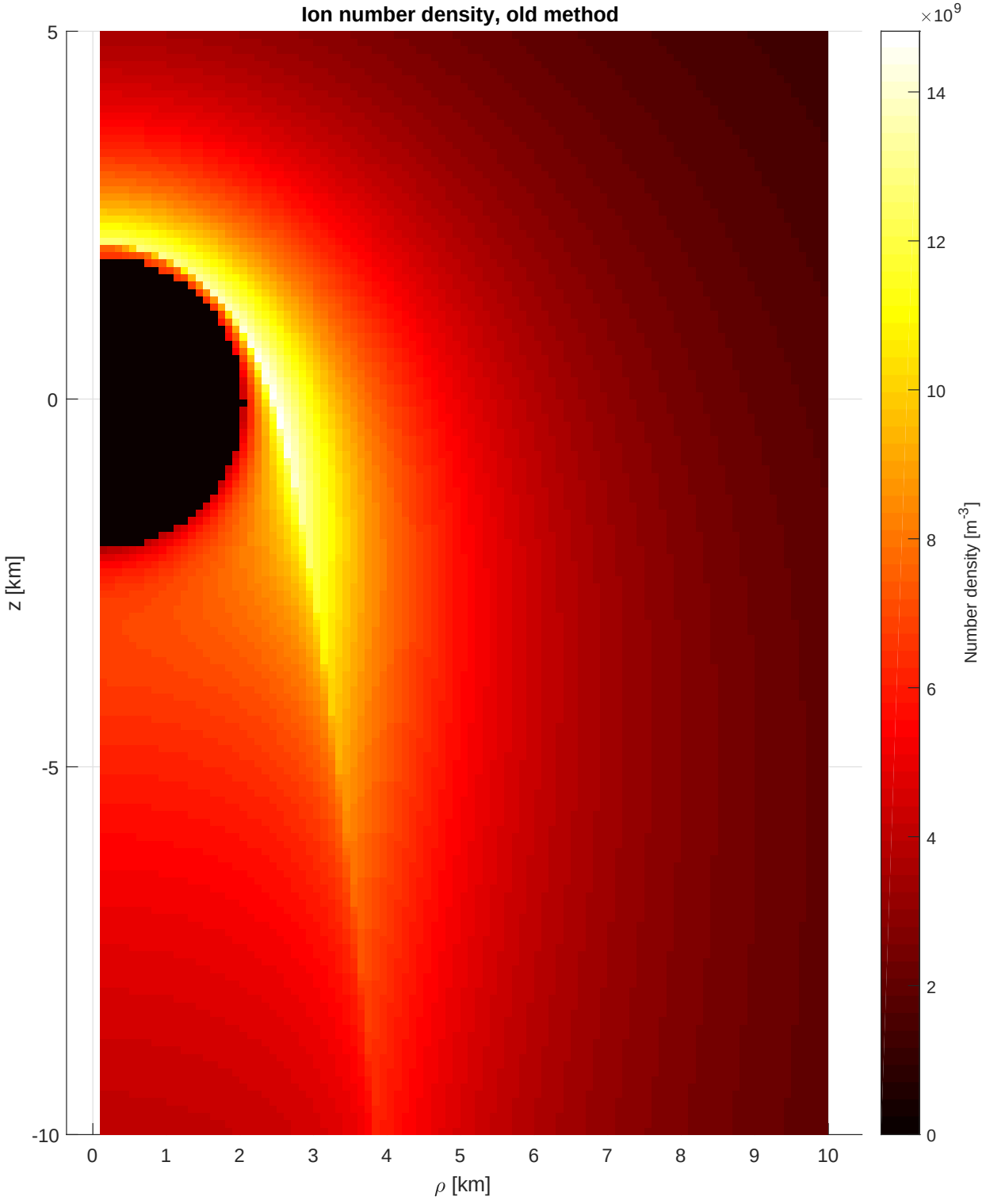




Figure 2: Result of Eq.1 using old cirteria for $C_\tau$ presented in Eq.2 and input parameters as specified in the final paragraph of section 3.

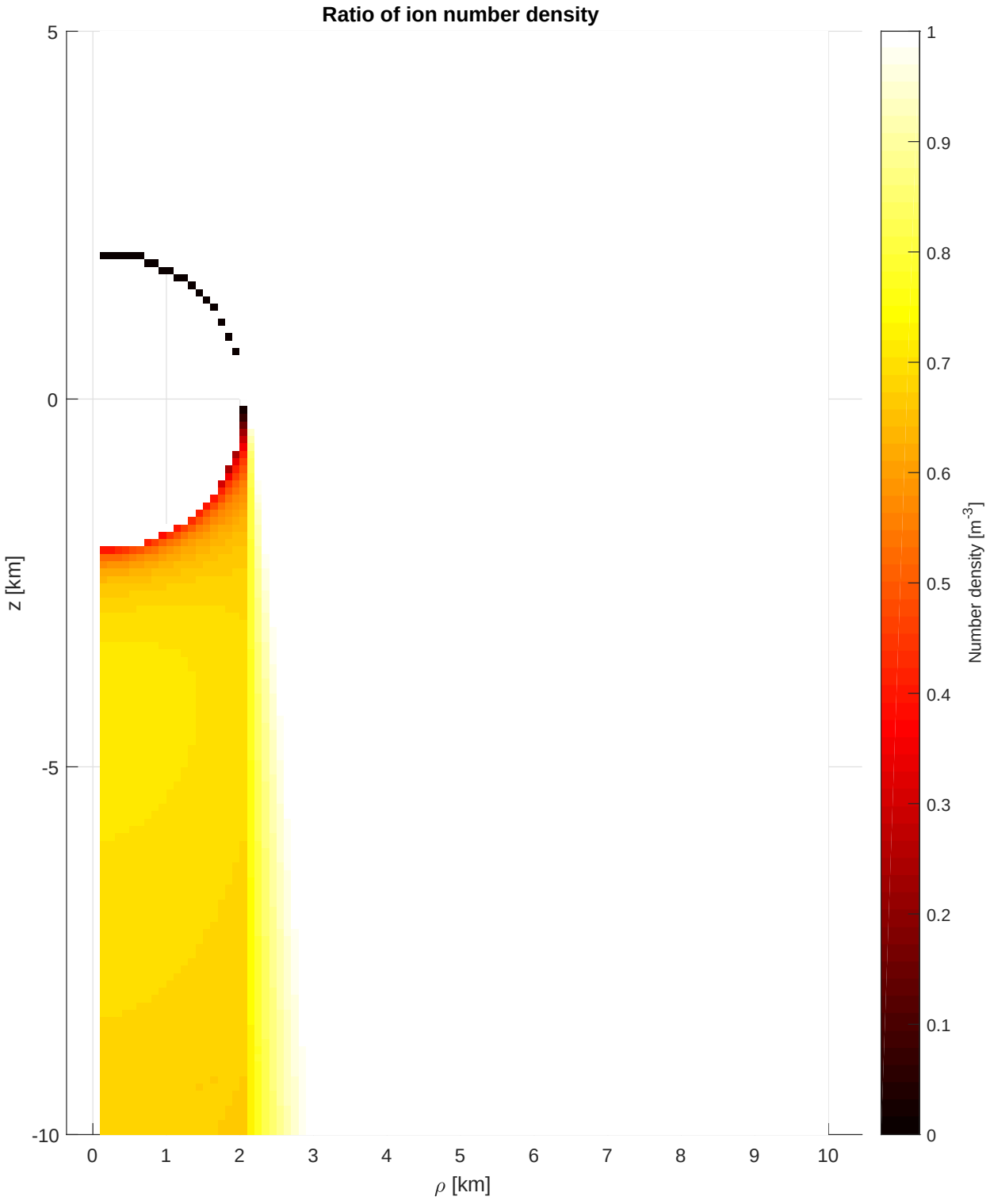




Figure 3: Ratio between results using new and old criteria.

# 6 Appendix

Here follows the MATLAB-code used in this project.

The main program, used to generate the figures.

```
% This code solves the intrgral in (7) in E. Vigrens paper using ...
    two versions
% of (8), plots both and the ratio
% For project 5 c
clear
close all
%% Integrals
t_min = 0.01; % minimum and maximum values for the life-time
t_max = 10;
rho = [0.1:0.1:10]*1000; % Coordinates
z = [-10:0.1:5]*1000;
Q = 1e27; % s^-1
u = 650; % m/s
a = 2.6647e+03; %m/s^2, all from Eriks code
f = 4.68e-7; % s^-1

% The function to be integrated
% erik = @(t,rho,z) f.*n_N.*C;
% erik = @(t,rho,z) f.*Q./(4.*pi.*u.*r_p).*C;
% erik = @(t,rho,z) f.*Q./(4.*pi.*u.*sqrt(rho.^2+z1.^2).*C;
% erik = @(t,rho,z) ...
    f.*Q./(4.*pi.*u.*sqrt(rho.^2+(z+0.5.*a.*t^2).^2).*C;
erik = @(t,rho,z) ...
    4.68e-7.*1e27./(4.*pi.*650.*sqrt(rho.^2+(z+0.5.*2.6647e+03.*t.^2).^2).^2)...
    .*old_func_num_vec(rho,z,t);

n_I_old = zeros(length(rho),length(z)); % assign coordinates
for i = 1:length(rho)
    for j = 1:length(z)
        erik2 = @(t) erik(t,rho(i),z(j));% define coordinate values ...
            to equation
        n_I_old(i,j) = integral(erik2,t_min,t_max); % do integral
    end
end

% Same thing again, but different function for C_tau
erik = @(t,rho,z) ...
    4.68e-7.*1e27./(4.*pi.*650.*sqrt(rho.^2+(z+0.5.*2.6647e+03.*t.^2).^2).^2)...
    .*new_func_num_vec(rho,z,t);
n_I = zeros(length(rho),length(z));
for i = 1:length(rho)
    for j = 1:length(z)
        erik2 = @(t) erik(t,rho(i),z(j));
        n_I(i,j) = integral(erik2,t_min,t_max);
    end
end


for I = 1:length(rho) % Find points that are inside the nucleus
    for J = 1:length(z)
        if sqrt((J-100)^2+I^2) <= 20
            n_I_old(I,J) = 0; % Mark each value inside the nucleus ...
                as 0
        end
    end
end
```

```
%% Plots
% Print density plots

figure('outerPosition',[0 40 850 1040])
surf(rho./1000,z./1000,zeros(length(z),length(rho)),n_I','EdgeColor','none')
colormap hot
view(0,90)
c = colorbar;
c.Label.String = 'Number density [m^{-3}]';
axis equal
xlabel('\rho [km]')
ylabel('z [km]')
title('Ion number density, new method')

figure('outerPosition',[850 40 850 1040])
surf(rho./1000,z./1000,zeros(length(z),length(rho)),n_I_old','EdgeColor','none')
colormap hot
view(0,90)
c = colorbar;
c.Label.String = 'Number density [m^{-3}]';
axis equal
xlabel('\rho [km]')
ylabel('z [km]')
title('Ion number density, old method')

figure('outerPosition',[0 40 850 1040])
surf(rho./1000,z./1000,zeros(length(z),length(rho)),n_I'./n_I_old','EdgeColor','none')
colormap hot
view(0,90)
c = colorbar;
c.Label.String = 'Number density [m^{-3}]';
axis equal
xlabel('\rho [km]')
ylabel('z [km]')
title('Ratio of ion number density')
```

The old criteria in vector format, used to generate figures.

```
% Function for computing the fate of particles. Simpel version. Can ...
    take t
% as a vector
function[C] = old_func_num_vec(rho,z,t)
r_c = 2000; % comet radius in m
u = 650; % m/s
a = 2.6647e+03; % m/s^2, from Erik's code

z1 = z + 0.5.*a.*t.^2; % position in absence of acceleration
r_p = sqrt(rho.^2+z1.^2); % y = 0
C = zeros(1,length(t));
for i = 1:length(t)
    if r_p(i) - r_c >= u*t(i) % else C = 0;
        C(i) = 1;
    end
end
end
```

The old criteria in simple form.

```
% Function for computing the fate of particles. Simpel version.
function[C] = old_func_num(rho,z,t)
r_c = 2000; % comet radius in m
```

```
u = 650; % m/s
a = 2.6647e+03; % m/s^2, from Erik's code

z1 = z + 0.5*a*t^2; % position in absence of acceleration
r_p = sqrt(rho^2+z1^2); % y = 0

if r_p-r_c >= u*t
    C = 1;
else
    C = 0;
end
end
```

The new criteria in vector format, used to generate figures.

```
% Function for computing C for many particles, uses distances in many
% points. Adapted for vector input
function[C] = new_func_num_vec(rho,z,t)
r_c = 2000; % comet radius in m
[rho0,z0,flag] = find_start_v(rho,z,t); % function for finding starting
% position

C = zeros(1,length(rho0));

r = sqrt(rho^2 + z^2); % distance to final position
r0 = sqrt(rho0.^2 + z0.^2); % distance to first position

if r < r_c
    % Start or end inside the comet
    C(:) = 0;

else
    % to test
    P = find_parab_v(rho,z,t); % function for finding parabola
    for i = 1:length(rho0)
        if r0(i) < r_c || flag(i) == 1
            C(i) = 0; % starting inside
        elseif z >= sqrt(r_c^2-rho^2) && z0(i) >= ...
            sqrt(r_c^2-rho0(i)^2) ||...
                z < -sqrt(r_c^2-rho^2) && z0(i) < ...
                    -sqrt(r_c^2-rho0(i)^2)
            C(i) = 1; % particle is far from nucleus
        elseif r0(i) >= r_c % else C = 0;
            % acutal test
            points = ...
                linspace(rho0(i),rho,max(100,ceil(abs(z-(-z0(i)))/10)));
            % find points on rho-axis
            z = ...
                P(1,i).*(points-rho0(i)).^2+P(2,i).*(points-rho0(i))+z0(i);
            % find their z-values
            D = sqrt((points).^2+(z).^2) - r_c;
            % find the distance between points and nucleus
            if min(D) > 0 % else C = 0;
                C(i) = 1;
            end
        end
    end
end
```

The new criteria in simple form.

```
% Function for computing C for many particles, uses distances in many
% points.
function[C] = new_func_num(rho,z,t)
r_c = 2000; % comet radius in m
u = 650; % m/s, from Erik's code
[rho0,z0,flag] = find_start(rho,z,t); % function for finding starting
% position
r = sqrt(rho^2 + z^2); % distance to final position
r0 = sqrt(rho0^2 + z0^2); % distance to first position

if r0 < r_c || r < r_c || flag == 1
    % Start or end inside the comet
    C = 0;

elseif rho > r_c+u*t || z >= sqrt(r_c^2-rho^2) && z0 >= ...
    sqrt(r_c^2-rho0^2)...
        || z < -sqrt(r_c^2-rho^2) && z0 < -sqrt(r_c^2-rho0^2)
    % Start far from the comet, under the comet, or end above the comet
    C = 1;

else
    % to test
    P = find_parab(rho,z,t); % function for finding parabola
    points = linspace(rho0,rho,max(100,ceil(abs(z-(-z0))/10)));
    % find points on rho-axis
    z = P(1).*(points-rho0).^2+P(2).*(points-rho0)+z0;
    % find their z-values
    D = sqrt((points).^2+(z).^2) - r_c;
    % find the distance between points and nucleus

    if min(D) > 0
        C = 1;
    else
        C = 0;
    end
end
```

Function for finding the coefficients for the parabolic trajectory of the particle.

```
% Function for finding coefficients for the parabola that is the ...
    trajectory
% of the particle, from final position and lifetime
function[P] = find_parab_v(rho2,z2,t)

u = 650; % m/s
a = 2.6647e+03; %m/s^2, from Erik's code

z1 = z2 + 0.5.*a.*t.^2; % position in absence of acceleration
r = sqrt(rho2.^2+z1.^2); % distance to nucleus
P = zeros(3,length(t));
for i = 1:length(t)
    % polynomial in the form y(x) = ax^2+bx+c this vector is [a,b,c].
    % From physics handbook p.156
    P(:,i) = [-a/(2*u^2*(rho2/r(i))^2),z1(i)/rho2,0];
end
end
```

Function for finding the starting position of the particle.

```
% Function for finding the starting position of the particle given its
```

```
% final position and life-time
function[rho0,z0,flag] = find_start_v(rho2,z2,t)
u = 650; % m/s
a = 2.6647e+03; % m/s^2, from Erik's code

% First tranformation
z1 = z2 + 0.5.*a.*t.^2;  % z-position in absence of acceleration
r = sqrt(rho2.^2+z1.^2); % distance to nucleus
u_rho = u.*rho2./r; % speed in rho-direction
u_z = u.*z1./r; % speed in z-direction

% Second transformation
rho0 = rho2 - u_rho.*t; % rho-position at start
z0 = z1 - u_z.*t; % z-position at start
flag = zeros(1,length(t)); % ev. warning
for i = 1:length(t)
    if not(isequal(sign(rho2), sign(rho0(i)))) % if signs are different
        flag(i) = 1;
    end
end
end
```