Theory of Electromagnetic Fields, experiment two

Gan Yuhao, 12211629, SUSTech

***Abstract*—In modern science, it is often necessary to use computer to calculate the potential distribution of charged bodies in the surrounding environment. However, computers can only deal with discrete physical quantities, and how to transform continuous quantities into discrete quantities in the physical world and analyze the differences between them become relatively important problems. In this experiment, the electric field distribution and electric field distribution of linear charged body are studied by using Matlab from the two methods of integral operation and discretization, and the relationship between discretization degree and error is analyzed by drawing pictures**

***Index Terms*—Electric fields, Infinitesimal methods, Numerical analysis, Matlab.**

I. Introduction

THIS paper is to introduce the theoretical principles surrounding matlab simulation, including the calculus algorithm to obtain the real value and the relevant principles of the computer simulation of the element method.

A. The Model of the Continuous Line Charge

The object to study in this paper is the electric field generated by a section of continuous uniform line charge in the 2-D rectangular coordinate (Fig. 1). The line charge starts from $A(-1,0)$ and ends at $B(1,0)$ (The default unit for the coordinate is *meter*), with a line charge density of $\rho = 1 \times 10^{-9} C/m$.

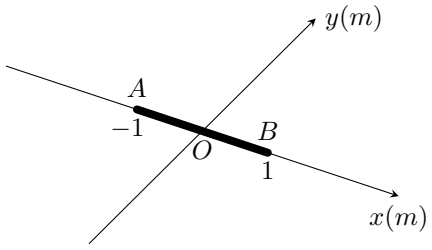

Fig. 1. The model of the line charge. It lies between $A(-1,0)$ and $B(1,0)$ with a line charge density of $\rho = 1 \times 10^{-9} C/m$.

B. Symbols and Notations

For convenience, we list the symbols and notations that may be used as follows (Table. I).

II. Integration Method

To obtain the reference value, we use integration method to calculate the analytical solution for the field and the following values

TABLE I
List of Symbols

Symbol	Quantity	Unit (SI)
k	electrostatic constant	$N \times m^2/C^2$
V	potential	V
Q	electric charge of a point charge	C
$\mathbf{E}$	electric field density	V/m
$\mathbf{R}$	distance from a point charge	m
$\mathbf{a}_R$	unit vector on the direction of $\mathbf{R}$	$-$
Q_i	electric charge of the i^{th} segment	C
R_i	distance from the i^{th} segment	m
L_i	length of the i^{th} segment	m
X_i	x position of the i^{th} segment	m
ρ	line charge density	C/m

A. Theoretical Analysis

To an arbitrary point $P(X_0, Y_0)$ on the platform where lies the line charge, we consider the impact of each $\mathrm{dQ} = \rho \mathrm{dx}$ on the line charge on P. So the potential can be represented as

$$V = k \int_{-1}^{1} \frac{\rho \mathrm{dx}}{R}. \tag{1}$$

And the distance between $P(X_0, Y_0)$ and $(x, 0)$ is

$$R = |\mathbf{R}| = \sqrt{(x - X_0)^2 + Y_0^2}. \tag{2}$$

We will finally have

$$V = k\rho \ln \left(\frac{1 - X_0 + \sqrt{(1 - X_0)^2 + Y_0^2}}{-1 - X_0 + \sqrt{(-1 - X_0)^2 + Y_0^2}} \right). \tag{3}$$

Then according to the relationship between $\mathbf{E}$ and V

$$\mathbf{E} = -\nabla V, \tag{4}$$

we can obtain the electric field density.

B. Numerical Simulation

Through Matlab calculation of the formula derived above, we can get the distribution of the potential(Fig. 2b), he distribution of the equipotential line (Fig. 2c) and the distribution curve of the electric field, (Fig. 2a). which reflects the reference true value.

III. Infinitesimal Method

Then we divide the continuous line charge into N independent point charges which uniformly lies between A and B, and then calculate the distribution.

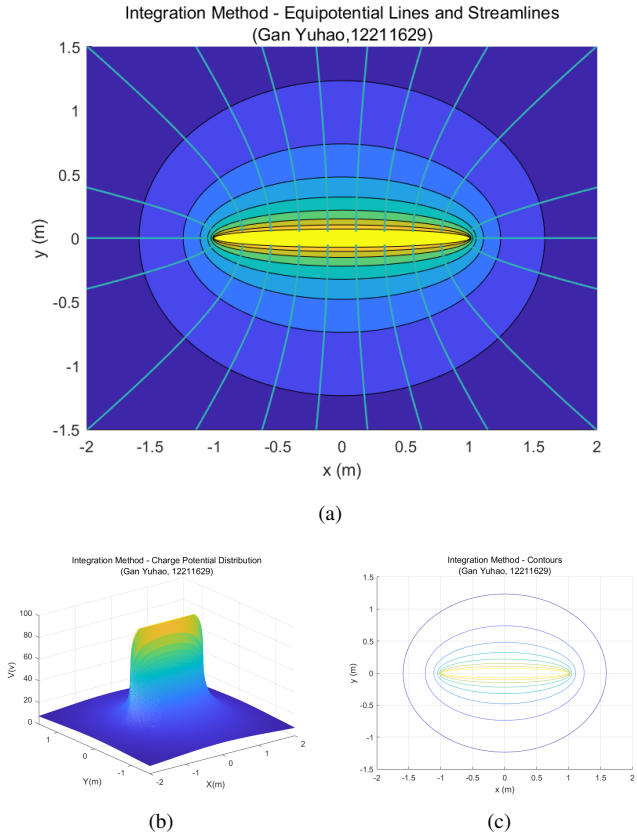

Fig. 2. From the resulting image, we can see that the potential distribution calculated by calculus is smooth and uniform, similar to the stretched point charge.

A. Theoretical Analysis

We can figure out the position and the amount of charge of each segment (i is from 0 to $N-1$):

$$Q_i = \rho \times L_i = \rho \frac{|\vec{AB}|}{N}, \tag{5}$$

$$X_i = (i + \frac{1}{2})L_i + X_A = (i + \frac{1}{2})\frac{|\vec{AB}|}{N} + X_A. \tag{6}$$

So the distance between i_{th} segment to a given point (X_0, Y_0) is

$$R_i = \sqrt{(X_i - X_0)^2 + (Y_0)^2}. \tag{7}$$

Treat the segments as separate point charges and calculate their potential distribution according to the *Superposition Principle*:

$$V = \sum_{i=0}^{N-1} k\frac{Q_i}{R_i}. \tag{8}$$

B. Finite Simulation

When $N = 5, 20, 50, 100$ by divided the whole line into $separations$, and also plot the potential distribution (Fig. 3), equipotential lines distribution (Fig. 4) and distrbution of electirc field lines (Fig. 5) for the four cases.

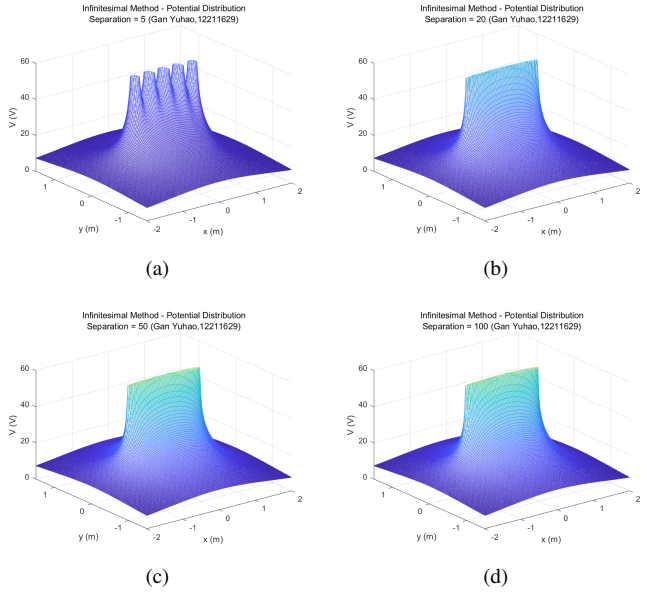

Fig. 3. Potential distributions calculated by infinitesimal method when dividing the line charge into 5, 20, 50 and 100 separations.

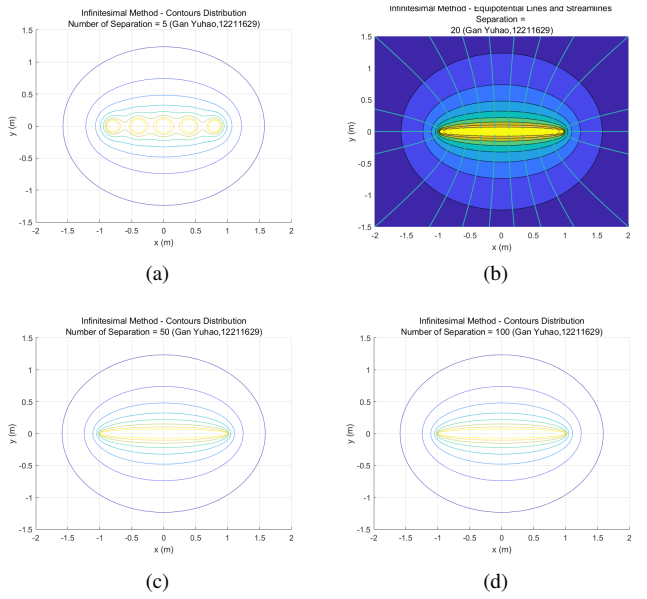

Fig. 4. Contours distributions calculated by infinitesimal method when dividing the line charge into 5, 20, 50 and 100 segments.

C. Error Analysis

It can be found that, especially in Fig. 5, that when the value of N is relatively small, the calculated single point charge can be clearly distinguished in the figure. When N further increases, the potential extreme points formed by the separated dot charges continue to approach and coincide. In Fig. **??**. When $N = 100$, that is, when it is large enough, the extreme values have almost completely coincided, and there is almost no difference between them and the reference value.

In order to obtain the quantitative error description, we

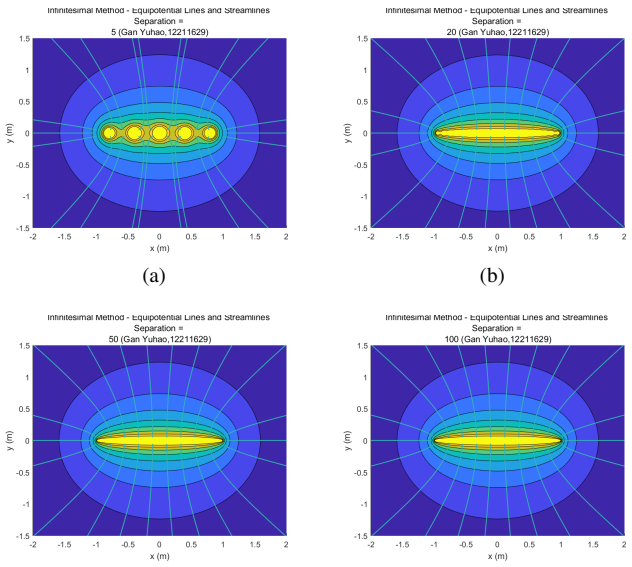

Fig. 5. Streamlines distributions calculated by infinitesimal method when dividing the line charge into 5, 20, 50 and 100 segments.

obtain the root-mean-square error of linear charged bodies divided into different number parts and the reference values calculated by calculus method.(Fig. 6).

At the same time, we can find that the main difference between the two methods lies near the charge distribution when N is small(Fig. **??**).

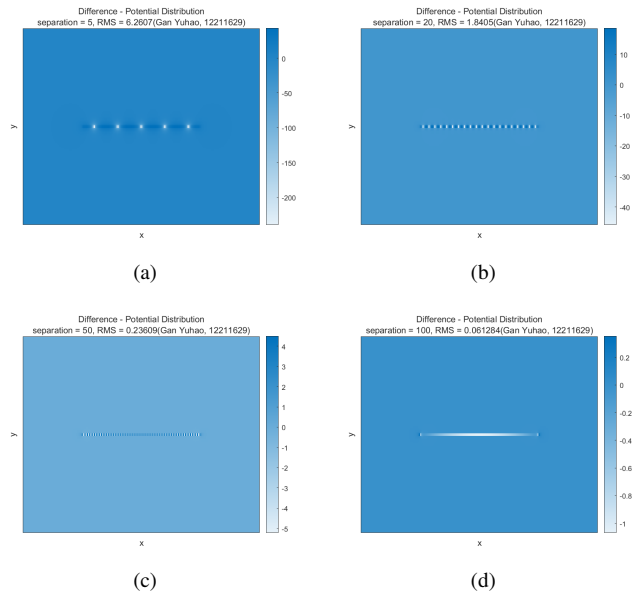

Fig. 6. The differences from true values when dividing the line charge into 5, 20, 50 and 100 segments.

In order to better portray this relationship, we calculate the error RMS when the number of separations are $1, 5, 10, 15, \cdots, 100$, respectively. And we plot the results as an "Error RMS – Number of Separations" Curve (Fig. 7).This

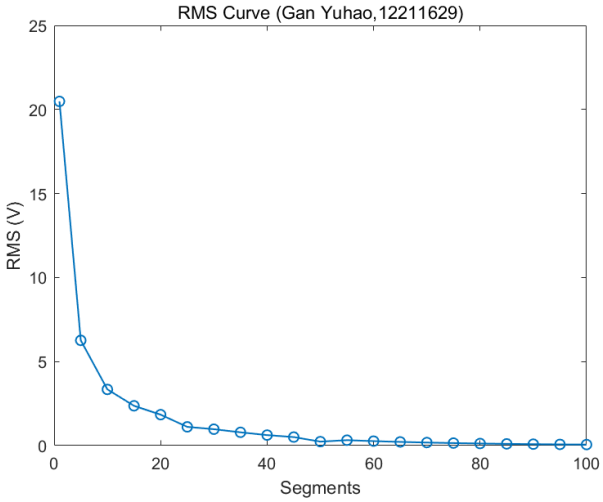

Fig. 7. The "Error RMS – Number of Segments" curve. This is a steep curve with a sharp decline. Near some of occasional points are not strictly declining, mainly caused by the discrete nature of the analysis, and do not affect the overall trend.

curve more obviously shows the changing trend of RMS under different N values.

IV. Conclusion

In this experiment, we use Matlab to simulate the potential distribution, equipotential line distribution and electric field line distribution of linear charged body. In the method, we use calculus and finite element method, and compare the difference between the two methods, calculate the root-mean-square error of the two methods, and get the relationship between the root-mean-square error and the number of partitions. It can be seen from experiments that the finite element method can simulate the real value to a large extent and provide convenience in the case of computer-aided calculation when the number of partitions is sufficient.

Appendix A
Code for Utility Classes

In order to make the structure clearer and reusable, we defined some utility class as below.

A. *Constants.m*

Class $Constants$ is used to store some static parameters, such as k, which is able to be invoked with $Constants.k$ in our main code.

```
classdef Constants
    properties(Constant)
        k = 9e9
    end
end
```

B. Point.m

Class $Point$ is The method is applied to simulate point coordinates of points in space.

```
classdef Point
    properties
        x
        y
        z
    end

    properties(Dependent)
        matr
    end

    methods
        function obj = Point(p)
            %
            obj.x = p(1);
            if(length(p)>=2)
                obj.y = p(2);
            end
            if(length(p)>=3)
                obj.z = p(3);
            end
        end

        function matr = get.matr(obj)
            %
            matr = [obj.x;obj.y;obj.z];
        end
        function y = minus(a,b)
            %minusPointVector
            if(isa(b,"Point"))
                y = Vector(a.matr-b.matr);
            elseif(isa(b,"Vector"))
                y = Point(a.matr-b.matr);
            end
        end
    end
end
```

C. Vector.m

Class $Vector$ is the abstraction of N-dimension ($N \leq 3$) vectors in the space. Differs from a $Point$, $Vector$s can be added up and measured length. Also, we defined $length$ as a dependent member to denote its inference.

```
classdef Vector
    properties
        x
        y
        z
    end

    properties(Dependent)
        length;
        matr;
    end

    methods
        function obj = Vector(p)
            obj.x = p(1);
            if (length(p)>=2)
                obj.y = p(2);
            end
            if (length(p)>=3)
                obj.z = p(3);
            end
        end

        function length = get.length(obj)
            length = sqrt(obj.x*obj.x+obj.y*obj.y);
        end

        function matr = get.matr(obj)
            matr = [obj.x;obj.y;obj.z];
        end
    end
end
```

D. Charge.m

Class $Charge$ describes charges in the space. A $Charge$ instance records point coordinates p ($Point$) and charge information q. We can call $Distance$ to obtain its distance relative to a arbitrary point. And we can use $PotentialField$ to calculate the total potential distribution information.

```
classdef Charge
    %Charge

    properties
        q
        p
    end

    methods
        function obj = Charge(q,p)
            obj.p = p;
            obj.q = q;
        end

        function d = Distance(obj,p)
            displacement = obj.p-p;
            d = displacement.length;
        end

        function V = PotentialField(obj,x_mesh,y_mesh)
            [ny,nx] = size(x_mesh);
            V = zeros(ny,nx);
            for x_dx = 1:nx
                for y_dy = 1:ny
                    r = obj.Distance(Point([x_mesh(y_d
                    ↪  y,x_dx),y_mesh(y_dy,x_dx)]));
                    V(y_dy,x_dx) = Constants.k * obj.q/r;
                end
            end
        end
    end
end
```

APPENDIX B
CODE FOR CALCULATION

Here are the main codes.

A. Integration Method

Apply the calculus results and plot the distribution.

```
clear;
clc;

%Parameters
rho = 1e-9;
%ViewLimits
range_size_x = 4;
range_size_y = 3;
range_sampling_rate = 100;
range_vec_x = linspace(-range_size_x / 2, range_size_x
↪  /...
 2, range_size_x * range_sampling_rate);
range_vec_y = linspace(-range_size_y / 2, range_size_y
↪  /...
 2, range_size_y * range_sampling_rate);
[mesh_x, mesh_y] = meshgrid(range_vec_x, range_vec_y);

%Use formula to calculate V
V = Constants.k .* rho .* ...
log((1 - mesh_x + sqrt((1 - mesh_x) .* (1 -mesh_x) +
↪  mesh_y .* mesh_y)) ...
```

```
./ (-1 - mesh_x + sqrt((-1 - mesh_x) .* (-1 -mesh_x) +
↪  mesh_y .* mesh_y)));

%draw the condition of V
figure(1);
grid on, axis equal;
mesh(mesh_x, mesh_y, V);
title(["Integration Method - Charge Potential
↪  Distribution","(Gan Yuhao, 12211629)"]);
axis([-range_size_x / 2, range_size_x / 2, -range_size_y /
↪  2, range_size_y / 2, 0, 100]);
xlabel('X(m)'),ylabel('Y(m)'),zlabel('V(v)');

%draw COntours
V_eq_min = 0;
V_eq_max = 60;
V_eq_sampling_num = 10;
V_eq = linspace(V_eq_min, V_eq_max, V_eq_sampling_num);

figure(2);
hold on,grid on, axis equal;
contour(mesh_x, mesh_y, V, V_eq);
title(["Integration Method - Contours", "(Gan Yuhao,
↪  12211629)"]);
xlabel("x (m)"), ylabel("y (m)");

%draw Streamlines
[E_x, E_y] = gradient(-V);
%Set place where streamlines appear
angles = linspace(-pi / 2, pi / 2, 3);
Ori_Point = 0.05;
appear_x = [-1 - Ori_Point .* cos(angles), linspace(-1, 1,
↪  10), linspace(-1, 1, 10), 1 + Ori_Point .*
↪  cos(angles)];
appear_y = [Ori_Point * sin(angles), Ori_Point * ones(1,
↪  10),-Ori_Point * ones(1, 10), -Ori_Point *
↪  sin(angles)];

figure(3);
hold on, grid on, axis equal;
contourf(mesh_x, mesh_y, V,V_eq);
fig_3 = streamline(mesh_x, mesh_y, E_x, E_y, appear_x,
↪  appear_y);
set(fig_3,"lineWidth", 1.2, "color", [0.2, 0.7, 0.7]);
xlabel("x (m)"), ylabel("y (m)");
title(["Integration Method - Equipotential Lines and
↪  Streamlines", "(Gan Yuhao,12211629)"]);
```

B. *Infinitesimal Method*

Divide the continuous line charge into N (*separations*) independent point charges and then calculate the distribution.

```
clear;
clc;
%The same as Integration Part
rho = 1e-9;
separation = 100;
range_size_x = 4;
range_size_y = 3;
range_sampling_rate = 50;
range_vec_x = linspace(-range_size_x / 2, range_size_x /2,
↪  range_size_x * range_sampling_rate);
range_vec_y = linspace(-range_size_y / 2, range_size_y /
↪  2, range_size_y * range_sampling_rate);
[mesh_x, mesh_y] = meshgrid(range_vec_x, range_vec_y);

%Calculate the Sum Potiential of cell Charges
V = zeros(size(mesh_x));
charges_part = 2 / separation;
charges_q = rho * charges_part;
%Take middle of small charge to represent the whole small
↪  charge
for i = 0 : (separation - 1)
    V = V + Charge(charges_q, Point([-1 + charges_part /
    ↪  2+ charges_part *
    ↪  i,0])).PotentialField(mesh_x,mesh_y);
end

figure(1);
grid on, axis equal;
mesh(mesh_x, mesh_y, V);
axis([-range_size_x / 2, range_size_x / 2, -range_size_y
↪  /2, range_size_y / 2, 0,60]);
xlabel("x (m)"), ylabel("y (m)"), zlabel("V (V)");
title(["Infinitesimal Method - Potential
↪  Distribution","Separation = " + separation + " (Gan
↪  Yuhao,12211629)"]);

%Calculate Contours and draw
V_eq_min = 0;
V_eq_max = 60;
V_eq_sampling_num = 10;
V_eq = linspace(V_eq_min, V_eq_max, V_eq_sampling_num);

figure(2);
hold on, grid on, axis equal;
contour(mesh_x, mesh_y, V, V_eq)
xlabel("x (m)"), ylabel("y (m)");
title(["Infinitesimal Method - Contours
↪  Distribution","Number of Separation = " + separation +
↪  " (Gan Yuhao,12211629)"]);

%Draw Streamlines
[E_x, E_y] = gradient(-V);
angles = linspace(-pi / 2, pi / 2, 3);
Ori_Point = 0.05;
appear_x = [-1 - Ori_Point .* cos(angles), linspace(-1, 1,
↪  10), linspace(-1, 1, 10), 1 + Ori_Point .*
↪  cos(angles)];
appear_y = [Ori_Point * sin(angles), Ori_Point * ones(1,
↪  10),-Ori_Point * ones(1, 10), -Ori_Point *
↪  sin(angles)];

figure(3);
hold on, grid on, axis equal;
contourf(mesh_x, mesh_y, V, V_eq);
fig_3 = streamline(mesh_x, mesh_y, E_x, E_y, appear_x,
↪  appear_y);
set(fig_3, "lineWidth", 1.2, "color", [0.2, 0.7, 0.7]);
title(["Infinitesimal Method - Equipotential Lines and
↪  Streamlines ", "Separation = " +separation + " (Gan
↪  Yuhao,12211629)"]);
xlabel("x (m)"), ylabel("y (m)");
```

C. *Differences of the Results*

give the difference between the above finite method and the true values, and calculate the RMS of the error at the sampling points.

```
clear;
clc;
rho = 1e-9;
separation = 100;
range_size_x = 4;
range_size_y = 3;
range_sampling_rate = 50;
range_vec_x = linspace(-range_size_x / 2, range_size_x /2,
↪  range_size_x * range_sampling_rate);
range_vec_y = linspace(-range_size_y / 2, range_size_y /
↪  2, range_size_y * range_sampling_rate);
[mesh_x, mesh_y] = meshgrid(range_vec_x, range_vec_y);
%Calculate the true value first
V_diff = Constants.k .* rho .* ...
log((1 - mesh_x + sqrt((1 - mesh_x) .* (1 -mesh_x) +
↪  mesh_y .* mesh_y)) ...
./ (-1 - mesh_x + sqrt((-1 - mesh_x) .* (-1 - mesh_x) +
↪  mesh_y .* mesh_y)));
charges_part = 2 / separation;
V_diff_inf = zeros(size(mesh_x));
charges_q = rho * charges_part;
%minus the infinitesimal ones
for i = 0 : (separation - 1)
    V_diff_inf = V_diff_inf+Charge(charges_q, Point([-1 +
    ↪  charges_part / 2+ charges_part * i,
    ↪  0])).PotentialField(mesh_x,mesh_y);
end
V_diff = V_diff - V_diff_inf;
val_rms = rms(V_diff(:));

%draw the error heatmap
```

```
figure(1);
axis equal;
heatmap(V_diff, "GridVisible", "off");
%grid only
%set labels
ax = gca;
ax.XDisplayLabels = nan(size(ax.XDisplayData));
ax.YDisplayLabels = nan(size(ax.YDisplayData));
title(["Difference - Potential Distribution", "separation
↪  = " + separation + ", RMS = " + val_rms + "(Gan Yuhao,
↪  12211629)"]);
```

D. "Error RMS – Number of separations" Curve

Change the variable *separations* and calculate the RMS of the errors, and then plot the curve.

```
clear;
clc;
% The same as before
rho = 1e-9;
separation = 100;
range_size_x = 4;
range_size_y = 3;
range_sampling_rate = 50;
range_vec_x = linspace(-range_size_x / 2, range_size_x /2,
↪  range_size_x * range_sampling_rate);
range_vec_y = linspace(-range_size_y / 2, range_size_y /2,
↪  range_size_y * range_sampling_rate);
[mesh_x, mesh_y] = meshgrid(range_vec_x, range_vec_y);

%Calculate the Potential Distribution Difference
val_range = [1, 5 : 5 : 100];
% range segment from 5 to 100, separation of 5
val_rms = [];
i = 0;
for separation = val_range
disp(separation);
i = i + 1;
V_diff = Constants.k .* rho .* ...
log((1 - mesh_x + sqrt((1 - mesh_x) .* (1 - mesh_x) +
↪  mesh_y .* mesh_y)) ...
./ (-1 - mesh_x + sqrt((-1 - mesh_x) .* (-1 -mesh_x) +
↪  mesh_y .* mesh_y)));
charges_part = 2 / separation;
charges_q = rho * charges_part;
for i = 0 : (separation - 1)
V_diff = V_diff - Charge(charges_q, Point([-1 +
↪  charges_part / 2+ charges_part *
↪  i,0])).PotentialField(mesh_x,mesh_y);
end
val_rms = [val_rms, rms(V_diff(:))];
end

%plot
figure(1);
grid on, axis equal;
plot(val_range, val_rms, '-o', 'lineWidth', 1.0);
xlim([0, 100]);
title("RMS Curve (Gan Yuhao,12211629)");
xlabel("Segments"), ylabel("RMS (V)")
```

ACKNOWLEDGMENT

Thanks to our course for offering us the template and the theoretical part of the integration method.