Application of Infinitesimal Method in Numerical Simulation of Continuous Charge Electric Field Distribution

Wang Zhuoyang, Undergraduate, SUSTech

***Abstract*—The calculation of the electric field distribution of complex systems usually requires the support of efficient computer arithmetic. In numerical simulation the number of sampling points is limited, inducing to the need for discretization. This paper employs infinitesimal method to discretize the continuously charged object and calculate the electric distribution. The results figured out by integration method are adopted as the true value and the comparative analysis between the two methods are done. The paper finally drew the conclusion that the error decreases significantly as the number of subdivisions increases.**

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

I. Introduction

THIS paper is a report for the second experiment of the course Engineering Electromagnetic Theory Laboratory. The purpose is to analyze the electric field distribution of charged systems by both theoretical calculations and infinitesimal analysis, and to compare the applicability of both in the numerical analysis of electric fields.

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 $\rho = 1 \times 10^{-9} C/m$.

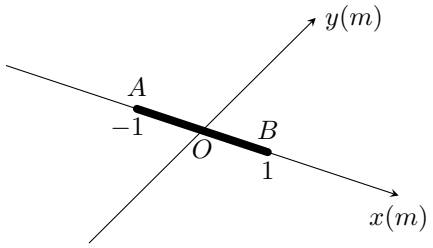


Fig. 1. The model of the line charge. It lies between $A(-1,0)$ and $B(1,0)$ with a line charge density of $\rho = 1 \times 10^{-9} C/m$.

B. Symbols and Notations

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

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

II. Integration Method

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

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

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

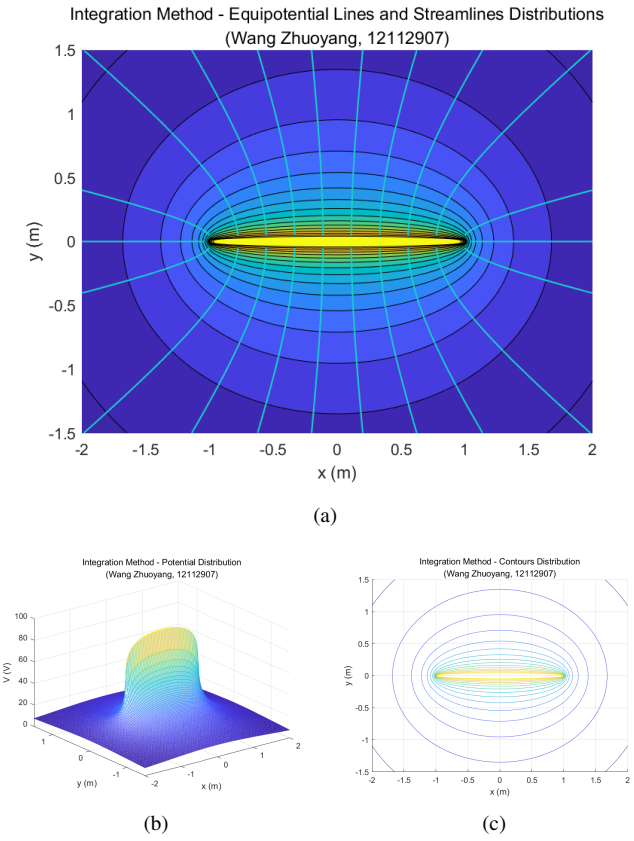
Then according to the relationship between $\mathbf{E}$ and V

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

we can obtain the electric field density.

B. Numerical Simulation

Apply the formulas in MATLAB we can plot out the potential distribution (Fig. 2b), equipotential lines distribution (Fig. 2c) and streamlines distribution (Fig. 2a). We will take these results as the true values in the comparative analysis later.



(a) (b) (c)

Fig. 2. Results of the simulation by integration method. The potential distribution for the line charge looks like the distribution for a "stretched" point charge. It is important that the distribution looks smooth and continuous. Theoretically the potential at the points on the line charge are infinite, and we did not emphasize this when choosing the sampling points.

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.

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

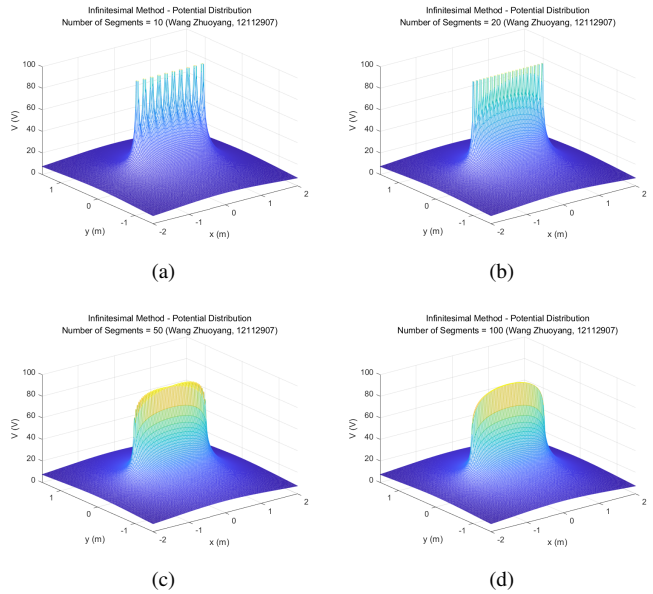
$$R_i = \sqrt{(X_i - X_0)^2 + (Y_0)^2}. \tag{7}$$

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. Numerical Simulation

We study the cases when $N = 10, 20, 50, 100$ by modifying the value of $segments$, and also plot the potential distribution (Fig. 3), equipotential lines distribution (Fig. 4) and streamlines distribution (Fig. 5) for the four cases.



(a) (b) (c) (d)

Fig. 3. Potential distributions calculated by infinitesimal method when dividing the line charge into 10, 20, 50 and 100 segments.

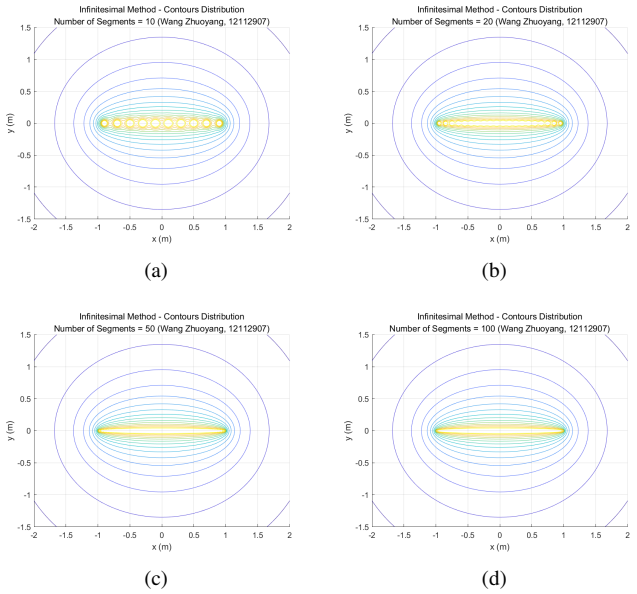
C. Comparative Analysis

It can be found that, especially in Fig. 5, that when N is relatively small, the edges of the discrete point charges in their distribution are very distinct. We can count out the number of segment in Fig. 5a. When N increases, it is more and more harder to identify the distribution of each point charge. When $N = 100$, it just looks the same to the true values in a general scale.

To demonstrate more clearly the differences between the two methods, we differ the results of the potential distribution at different number of segments from the true values in the sampling range, in order to obtain the quantified error (Fig. 6). And we calculate the root mean square (RMS) value for the differences as a measure of the error magnitude.

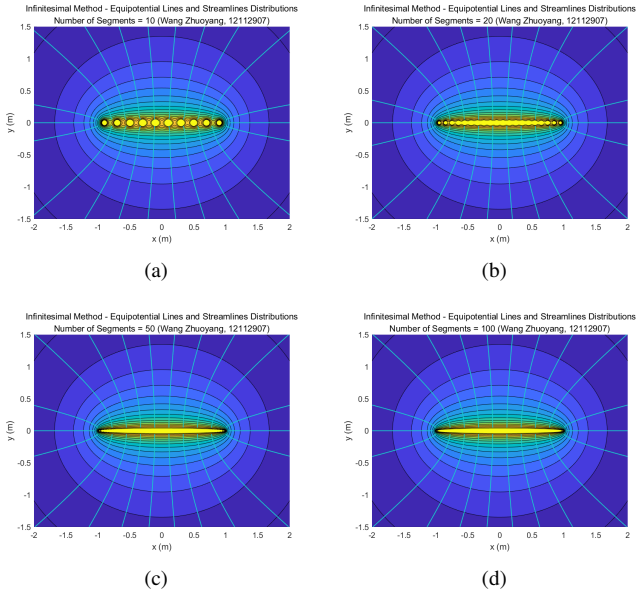
It is obviously that the RMS value of the error decreases significantly as the number of segments increases. This error is minimal when $N = 100$, and is reflected as a very small and smooth error on the heatmap (Fig. 6d).

At the same time, we can find that the difference between the electric field distribution of the two mainly lies in the space



(a) (b) (c) (d)

Fig. 4. Contours distributions calculated by infinitesimal method when dividing the line charge into 10, 20, 50 and 100 segments.

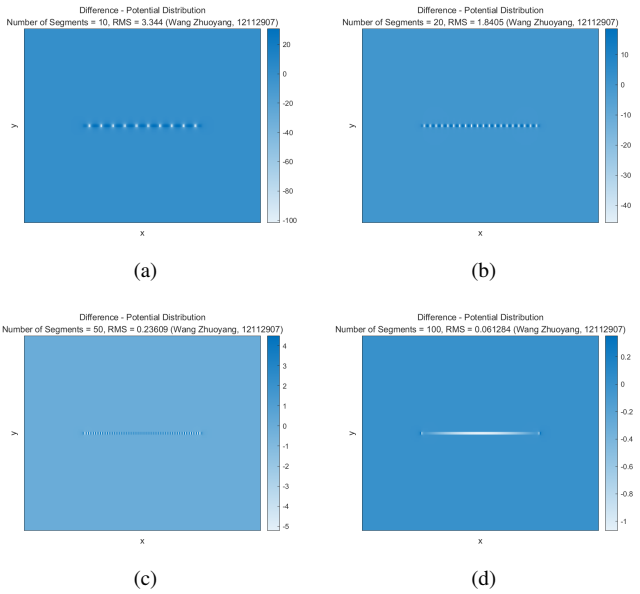


(a) (b) (c) (d)

Fig. 5. Streamlines distributions calculated by infinitesimal method when dividing the line charge into 10, 20, 50 and 100 segments.

near the line charge, and the relatively distant distribution is also relatively similar even in the case of smaller N.

In order to better portray this relationship, we calculate the error RMS when the number of segments are $1, 5, 10, 15, \cdots, 100$, respectively. And we plot the results as an "Error RMS – Number of Segments" Curve (Fig. 7). The sharp downward trend of the curve is a good illustration of the conclusions obtained earlier.



(a) (b) (c) (d)

Fig. 6. The differences from true values when dividing the line charge into 10, 20, 50 and 100 segments. RMS values are noted on the title, we can found which decreasing significantly as N increases.

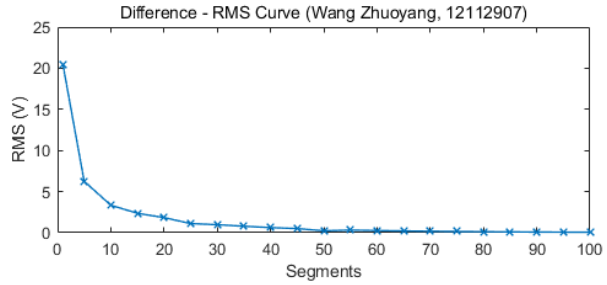


Fig. 7. The "Error RMS – Number of Segments" curve. This is a steep curve with a sharp decline. Near some of occasional points are not strictly declining, mainly caused by the discrete nature of the analysis, and do not affect the overall trend.

IV. Conclusion

To be concluded, we use both integration method and infinitesimal method to calculate the electric distribution of a continuous line charge in the 2-D coordinate.

We adjust the number of segments in infinitesimal method analyze the differences between the results, and draw the conclusion that the error decreases rapidly with increasing number of segments, and finally it is negligible.

Therefore, it is effective to use the infinitesimal method in the electric field analysis, where the continuous charge is divided into a sufficient number of small segments and considered as a set of separate point charges to obtain a result very similar to the analytical solution. This sufficient number of segments is not too large and relevant to the specific situation.

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 $Constants.k$.

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

B. Point.m

Class $Point$ is the abstraction of N-dimension ($N \leq 3$) points in the space. Dependent member mat always converts the parameters into matrix form. We overwrote the $minus$ operator to allow operations like $Point - Point$ $(= Vector)$ and $Point - Vector$ $(= Point)$.

```
classdef Point
    properties
        x
        y
        z
    end

    properties(Dependent)
        mat_
    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 mat_ = get.mat_(obj)
            mat_ = [obj.x; obj.y; obj.z];
        end

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

D. Charge.m

Class $Charge$ describes charges in the space. A $Charge$ instance records its position p ($Point$) and amount of charge q. We can call $EvalDistance$ to obtain its distance relative to a arbitrary point. And we can use $EvalPotentialField$ to get the potential in the specified sampling interval, which is described with x_{mesh} and y_{mesh}.

```
classdef Charge
    properties
        q
        p
    end

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

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

        function V = EvalPotentialField(obj, x_mesh,
↪   y_mesh)
            [n_y, n_x] = size(x_mesh);
            V = zeros(n_y, n_x);
            for x_idx = 1 : n_x
                for y_idx = 1 : n_y
                    r_ =
↪   obj.EvalDistance(Point([x_mesh(y_idx, x_idx),
↪   y_mesh(y_idx, x_idx)]));
                    V(y_idx, x_idx) = 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 theoretical results and plot the distribution.

```
clear;
clc;

%% Definitions
% Parameters
rho = 1e-9; % [C/m]
% Viewport
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);

%% Potential Distribution
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)));

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, 100]);
title(["Integration Method - Potential Distribution",
↪  "(Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)"), zlabel("V (V)");
% saveas(1, "../fig/int_1.png");

%% Contours Distribution
V_eq_min = 0;
V_eq_max = 70;
V_eq_sampling_num = 18;
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 Distribution",
↪  "(Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
% saveas(2, "../fig/int_2.png");

%% Equipotential Lines and Streamlines Distributions
[E_x, E_y] = gradient(-V);
angles = linspace(-pi / 2, pi / 2, 3);
sp_dist = 0.05;
sp_x = [-1 - sp_dist .* cos(angles), linspace(-1, 1, 10),
↪  linspace(-1, 1, 10), 1 + sp_dist .* cos(angles)];
sp_y = [sp_dist * sin(angles), sp_dist * ones(1, 10),
↪  -sp_dist * ones(1, 10), -sp_dist * sin(angles)];

figure(3);
hold on, grid on, axis equal;
contourf(mesh_x, mesh_y, V, V_eq);
fig_sl = streamline(mesh_x, mesh_y, E_x, E_y, sp_x, sp_y);
set(fig_sl, "lineWidth", 1.0, "color", [0.1, 0.8, 0.8]);
title(["Integration Method - Equipotential Lines and
↪  Streamlines Distributions", "(Wang Zhuoyang,
↪  12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
% saveas(3, "../fig/int_3.png");
```

B. *Infinitesimal Method*

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

```
clear;
clc;

%% Definitions
% Parameters
rho = 1e-9; % [C/m]
segments = 20;
% Viewport
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);

%% Potential Distribution
V = zeros(size(mesh_x));
charges_len = 2 / segments;
charges_q = rho * charges_len;
for i = 0 : (segments - 1)
    V = V + Charge(charges_q, Point([-1 + charges_len / 2
↪  + charges_len * i, 0])).EvalPotentialField(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, 100]);
title(["Infinitesimal Method - Potential Distribution",
↪  "Number of Segments = " + segments + " (Wang Zhuoyang,
↪  12112907)"]);
xlabel("x (m)"), ylabel("y (m)"), zlabel("V (V)");
% saveas(1, "../fig/inf_seg=" + segments + "_1.png");

%% Contours Distribution
V_eq_min = 0;
V_eq_max = 70;
V_eq_sampling_num = 18;
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(["Infinitesimal Method - Contours Distribution",
↪  "Number of Segments = " + segments + " (Wang Zhuoyang,
↪  12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
% saveas(2, "../fig/inf_seg=" + segments + "_2.png");

%% Equipotential Lines and Streamlines Distributions
[E_x, E_y] = gradient(-V);
angles = linspace(-pi / 2, pi / 2, 3);
sp_dist = 0.05;
sp_x = [-1 - sp_dist .* cos(angles), linspace(-1, 1, 10),
↪  linspace(-1, 1, 10), 1 + sp_dist .* cos(angles)];
sp_y = [sp_dist * sin(angles), sp_dist * ones(1, 10),
↪  -sp_dist * ones(1, 10), -sp_dist * sin(angles)];

figure(3);
hold on, grid on, axis equal;
contourf(mesh_x, mesh_y, V, V_eq);
fig_sl = streamline(mesh_x, mesh_y, E_x, E_y, sp_x, sp_y);
set(fig_sl, "lineWidth", 1.0, "color", [0.1, 0.8, 0.8]);
title(["Infinitesimal Method - Equipotential Lines and
↪  Streamlines Distributions", "Number of Segments = " +
↪  segments + " (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
% saveas(3, "../fig/inf_seg=" + segments + "_3.png");
```

C. *Differences of the Results*

Plot the difference between the above results and the true values, and calculate the RMS of the error at the sampling points.

```
clear;
clc;

%% Definitions
% Parameters
rho = 1e-9; % [C/m]
segments = 100;
% Viewport
```

```
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);

%% Potential Distribution
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_len = 2 / segments;
charges_q = rho * charges_len;
for i = 0 : (segments - 1)
    V_diff = V_diff - Charge(charges_q, Point([-1 +
↪  charges_len / 2 + charges_len * i,
↪  0])).EvalPotentialField(mesh_x, mesh_y);
end
val_rms = rms(V_diff(:));

%% Plotting
figure(1);
axis equal;
heatmap(V_diff, "GridVisible", "off");
ax = gca;
ax.XDisplayLabels = nan(size(ax.XDisplayData));
ax.YDisplayLabels = nan(size(ax.YDisplayData));
title(["Difference - Potential Distribution", "Number of
↪  Segments = " + segments + ", RMS = " + val_rms + "
↪  (Wang Zhuoyang, 12112907)"]);
xlabel("x"), ylabel("y");
% saveas(1, "../fig/diff_seg=" + segments + ".png");
```

D. "Error RMS – Number of Segments" Curve

Adjust the variable *segments* and calculate the RMS of the errors, and then plot them out.

```
clear;
clc;

%% Definitions
% Parameters
rho = 1e-9; % [C/m]
segments = 100;
% Viewport
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);

%% Potential Distribution
val_range = [1, 5 : 5 : 100];
val_rms = [];
i = 0;
for segments = val_range
    disp(segments);
    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_len = 2 / segments;
    charges_q = rho * charges_len;
    for i = 0 : (segments - 1)
        V_diff = V_diff - Charge(charges_q, Point([-1 +
↪  charges_len / 2 + charges_len * i,
↪  0])).EvalPotentialField(mesh_x, mesh_y);
    end
    val_rms = [val_rms, rms(V_diff(:))];
end

%% Plotting
figure(1);
grid on, axis equal;
plot(val_range, val_rms, '-x', 'lineWidth', 1.0);
xlim([0, 100]);
title(["Difference - RMS Curve (Wang Zhuoyang,
↪  12112907)"]);
xlabel("Segments"), ylabel("RMS (V)");
% saveas(1, "../fig/rms.png");
```


ACKNOWLEDGMENT

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

Wang Zhuoyang "Meow, meow, meow." – An anonymous cat who passed by and knew nothing about the world said like this.