Engineering Electromagnetic Theory

Laboratory 1

Oct 25, 2022

Wang Zhuoyang
12112907

Contents

A Utility Classes

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

A.1 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
```

A.2 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
```

A.3 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
```

A.4 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.

```matlab
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
```

B Case 1: Electric field distribution of two identical point charges

In this case we studied the electric field distribution of two identical point charges.

B.1 Declarations

First we set up the scene by placing the charges with certain positions and charges. In this case we placed two identical point charges of $10^{-9}C$.

Also, we defined the sampling area using the function *meshgrid*.

```
%% Definitions
% Charges
Q_1 = Charge(1e-9, Point([-0.01, 0]));
Q_2 = Charge(1e-9, Point([0.01, 0]));
% Viewport
range_size = 0.1;
range_samples = 80;
x_range = linspace(-range_size / 2, range_size / 2, range_samples);
y_range = linspace(-range_size / 2, range_size / 2, range_samples);
[x_mesh, y_mesh] = meshgrid(x_range, y_range);
```

B.2 Potential Distribution

B.2.1 MATLAB Code

We can use $Charge.EvalPotentialField$ to obtain the potential field V and plot it out.

```
%% Potential Distribution
V = Q_1.EvalPotentialField(x_mesh, y_mesh) + Q_2.EvalPotentialField(x_mesh,
↪ y_mesh);

figure(1);
```

```
mesh(x_mesh, y_mesh, V);
title("Two Identical Point Charges - Potential Distribution (Wang Zhuoyang,
↪  12112907)");
xlabel("x (m)"), ylabel("y (m)"), zlabel("V (V)");
```

B.2.2 Result and Analysis

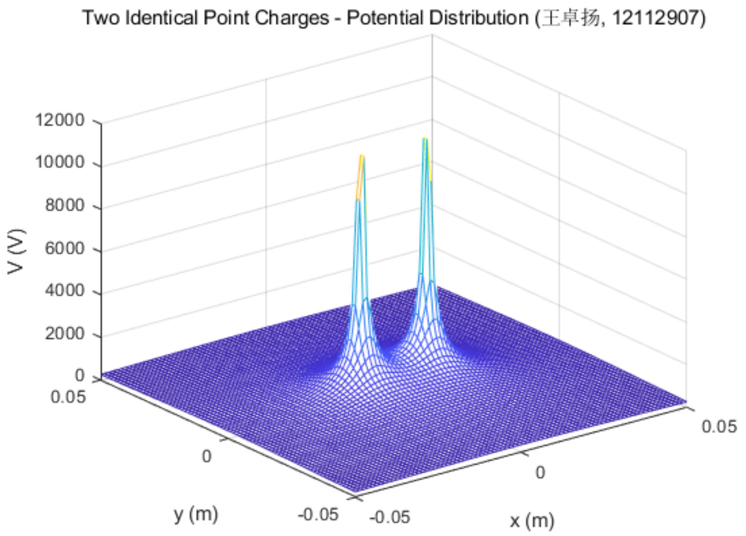

Figure 1: Two Identical Point Charges - Potential Distribution

As shown in Figure 1, there are two infinite peaks in the potential field, which is in line with the expectations.

B.3 Contours Distribution

B.3.1 MATLAB Code

To plot the contours we need to generate an array of potential values using *linspace* (in additional we can make some adjustment). Then we can employ the function *contour* to draw a series of potential contours as below.

```
%% Contours Distribution
V_min = 320;
V_max = 3000;
V_samples = 20;
V_eq = linspace(V_min, V_max, V_samples);

figure(2);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
title("Two Identical Point Charges - Contours Distribution (Wang Zhuoyang, 12112907)");
xlabel("x (m)"), ylabel("y (m)");
```

B.3.2 Result and Analysis

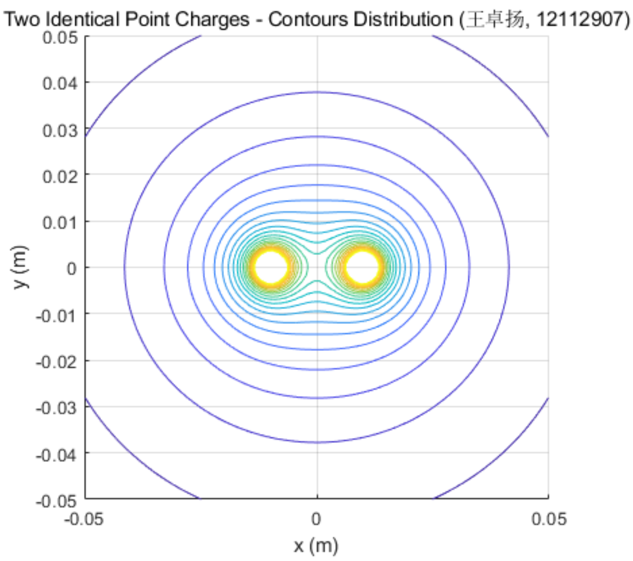

Figure 2: Two Identical Point Charges - Contours Distribution

As shown in Figure 2, the equipotential lines are distributed around the charges and conform to the potential distribution diagram above.

B.4 Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)

B.4.1 MATLAB Code

streamline can plot lines along the vector field from a specified start point. We use *gradient* to figure out the gradient of $-V$, which is the electric field intensity $\mathbf{E}$. Then we take the points surrounding the charges as the start points to plot the streamlines.

```
%% Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)
[E_x, E_y] = gradient(-V);

angle_samples = 4 * (2) + 4 + 1;
angle_dist = 0.0001;
angle = linspace(0, 2 * pi, angle_samples);
angle_x_s = angle_dist * cos(angle);
angle_y_s = angle_dist * sin(angle);

figure(3);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
streamline(x_mesh, y_mesh, E_x, E_y, angle_x_s + Q_1.p.x, angle_y_s + Q_1.p.y);
streamline(x_mesh, y_mesh, E_x, E_y, angle_x_s + Q_2.p.x, angle_y_s + Q_2.p.y);
title(["Two Identical Point Charges - Equipotential Lines and Streamlines
↪  Distributions", "(Smooth Continuous Curves) (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
```

B.4.2 Result and Analysis

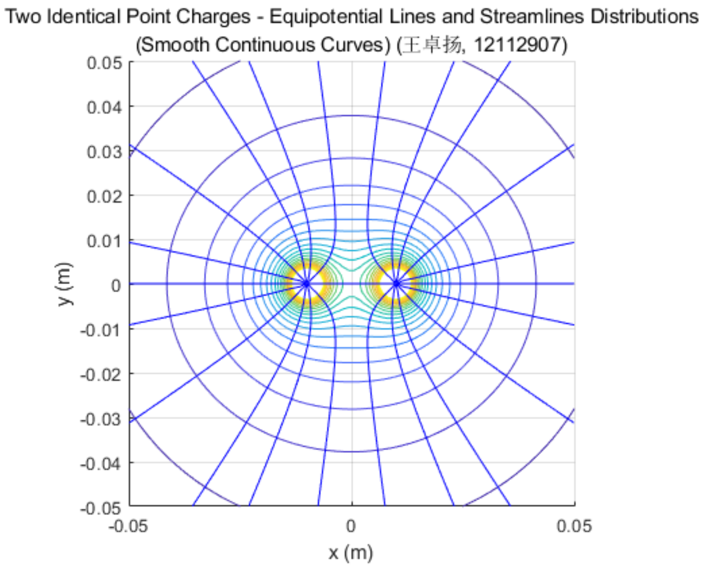

Figure 3: Two Identical Point Charges - Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)

As is shown in Figure 3, the streamlines are perpendicular to the equipotential lines, which conforms to the theoretical derivation.

B.5 Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)

B.5.1 MATLAB Code

Finally we use another method to draw streamlines distributions. We just simply draw the arrows to indicate the electric intensity field **E** as below.

```
%% Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)
E = sqrt(E_x .^ 2 + E_y .^ 2);
E_x_normal = E_x ./ E;
E_y_normal = E_y ./ E;

```

```
figure(4);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
quiver(x_mesh, y_mesh, E_x_normal, E_y_normal);
axis([-0.02, 0.02, -0.02, 0.02]);
title(["Two Identical Point Charges - Equipotential Lines and Streamlines
↪  Distributions", "(Normalized Arrowheads) (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
```

B.5.2 Result and Analysis

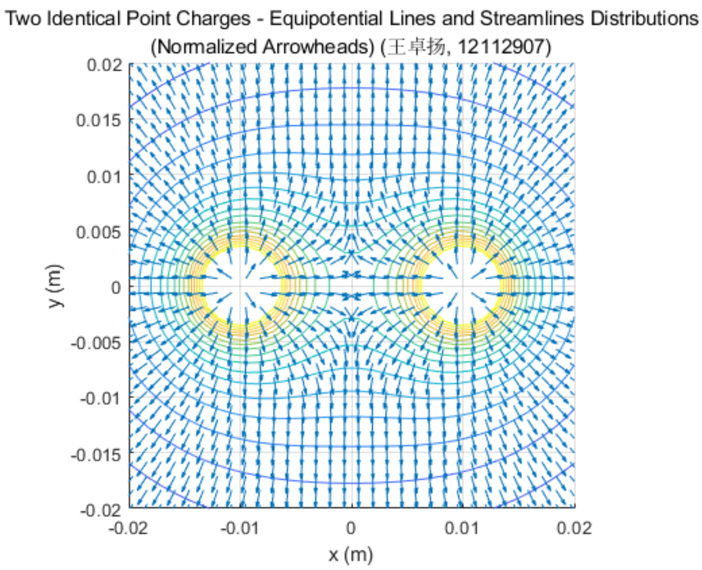

Figure 4: Two Identical Point Charges - Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)

As shown in Figure 4, the arrows are consistent with **E**.

C Case 2: Electric field distribution of two opposite point charges with the same magnitude

In this case we studied the electric field distribution of two opposite point charges with the same magnitude.

C.1 Declarations

First we set up the scene by placing the charges with certain positions and charges. In this case we put two opposite point charges with the same magnitude $5 \times 10^{-9} C$.

Also, we defined the sampling area using the function *meshgrid*.

```
%% Definitions
% Charges
Q_1 = Charge(5e-9, Point([-2, 0]));
Q_2 = Charge(-5e-9, Point([2, 0]));
% Viewport
range_size = 8;
range_samples = 50;
x_range = linspace(-range_size / 2, range_size / 2, range_samples);
y_range = linspace(-range_size / 2, range_size / 2, range_samples);
[x_mesh, y_mesh] = meshgrid(x_range, y_range);
```

C.2 Potential Distribution

C.2.1 MATLAB Code

We can use $Charge.EvalPotentialField$ to obtain the potential field V and plot it out.

```
%% Potential Distribution
V = Q_1.EvalPotentialField(x_mesh, y_mesh) + Q_2.EvalPotentialField(x_mesh,
↪  y_mesh);

```

```
figure(1);
mesh(x_mesh, y_mesh, V);
zlim([-250, 250]);
title("Two Opposite Point Charges - Potential Distribution (Wang Zhuoyang,
↪  12112907)");
xlabel("x (m)"), ylabel("y (m)"), zlabel("V (V)");
```

C.2.2 Result and Analysis

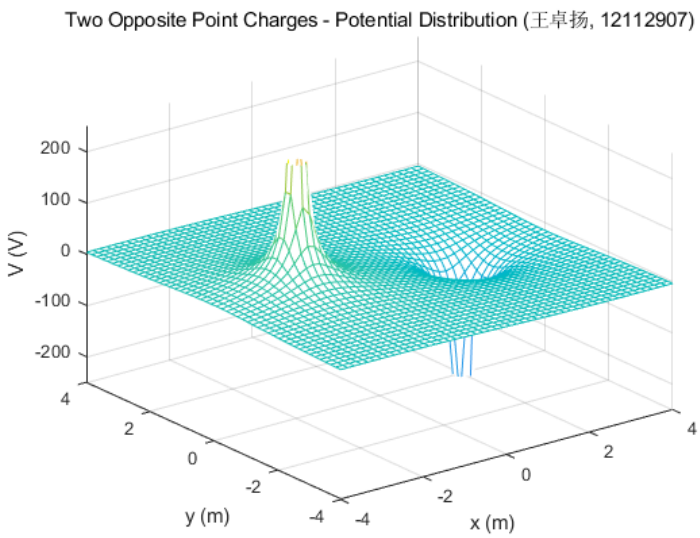

Figure 5: Two Opposite Point Charges - Potential Distribution

As shown in Figure 5, there are two opposite infinite peaks in the potential field, which is in line with the expectations.

C.3 Contours Distribution

C.3.1 MATLAB Code

To plot the contours we need to generate an array of potential values using *linspace*. Then we can employ the function *contour* to draw a series of potential contours as below.

```
%% Contours Distribution
V_min = -4.8;
V_max = 4.8;
V_samples = 21;
V_eq = linspace(V_min, V_max, V_samples) .^ 3;
V_eq(intersect(find(abs(V_eq) > 0), find(abs(V_eq) < 1))) = [];

figure(2);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
title("Two Opposite Point Charges - Contours Distribution (Wang Zhuoyang, 12112907)");
xlabel("x (m)"), ylabel("y (m)");
```

C.3.2 Result and Analysis

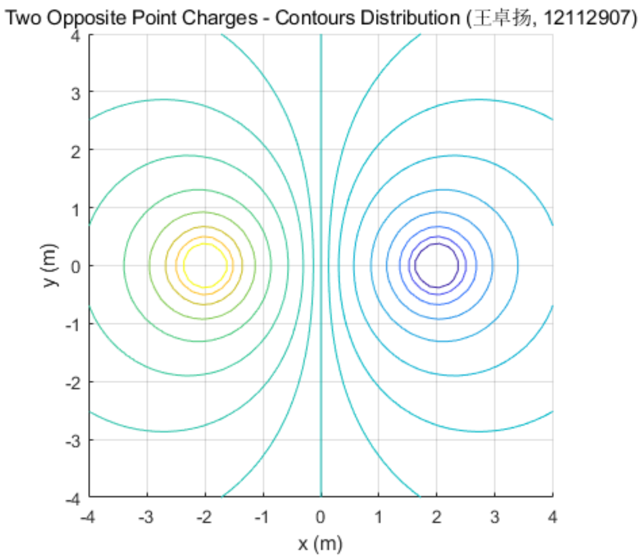

Figure 6: Two Opposite Point Charges - Contours Distribution

As shown in Figure 6, the equipotential lines are distributed around the charges and conform to the potential distribution diagram above.

C.4 Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)

C.4.1 MATLAB Code

streamline can plot lines along the vector field from a specified start point. We use *gradient* to figure out the gradient of $-V$, which is the electric field intensity $\mathbf{E}$. Then we take the points surrounding the charges as the start points to plot the streamlines.

It is worth noticing that *streamline* draw lines according to the direction of the vectors, so when we process the negative charge we need to reverse the sign of $\mathbf{E}$.

```matlab
%% Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)
[E_x, E_y] = gradient(-V);

angle_samples = 2 * (2) + 3 + 1;
angle_dist = 0.01;
angle_half = linspace(-pi / 3.5, pi / 3.5, angle_samples + 2);
angle_half = angle_half([1, 3 : end - 2, end]);
angle_half_x_s = angle_dist * cos(angle_half);
angle_half_y_s = angle_dist * sin(angle_half);
angle_half_i = linspace(-pi / 2.1, pi / 2.1, angle_samples);
angle_half_x_s_i = angle_dist * cos(angle_half_i);
angle_half_y_s_i = angle_dist * sin(angle_half_i);

figure(3);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
streamline(x_mesh, y_mesh, E_x, E_y, -angle_half_x_s + Q_1.p.x, -angle_half_y_s + Q_1.p.y);
streamline(x_mesh, y_mesh, E_x, E_y, angle_half_x_s_i + Q_1.p.x, angle_half_y_s_i + Q_1.p.y);
streamline(x_mesh, y_mesh, -E_x, -E_y, angle_half_x_s + Q_2.p.x, angle_half_y_s + Q_2.p.y);
title(["Two Opposite Point Charges - Equipotential Lines and Streamlines Distributions", "(Smooth Continuous Curves) (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
```

C.4.2 Result and Analysis

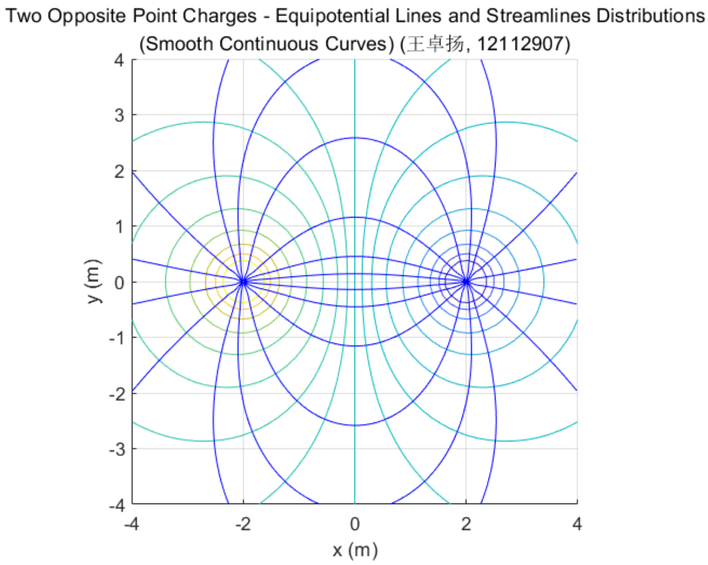

Figure 7: Two Opposite Point Charges - Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)

As is shown in Figure 7, the streamlines are perpendicular to the equipotential lines, which conforms to the theoretical derivation.

C.5 Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)

C.5.1 MATLAB Code

Finally we use another method to draw streamlines distributions. We just simply draw the arrows to indicate the electric intensity field **E** as below.

```
%% Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)
E = sqrt(E_x .^ 2 + E_y .^ 2);
E_x_normal = E_x ./ E;
E_y_normal = E_y ./ E;

```

```
figure(4);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
quiver(x_mesh, y_mesh, E_x_normal, E_y_normal);
axis([-3, 3, -3, 3]);
title(["Two Opposite Point Charges - Equipotential Lines and Streamlines
↪  Distributions", "(Normalized Arrowheads) (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
```

C.5.2 Result and Analysis

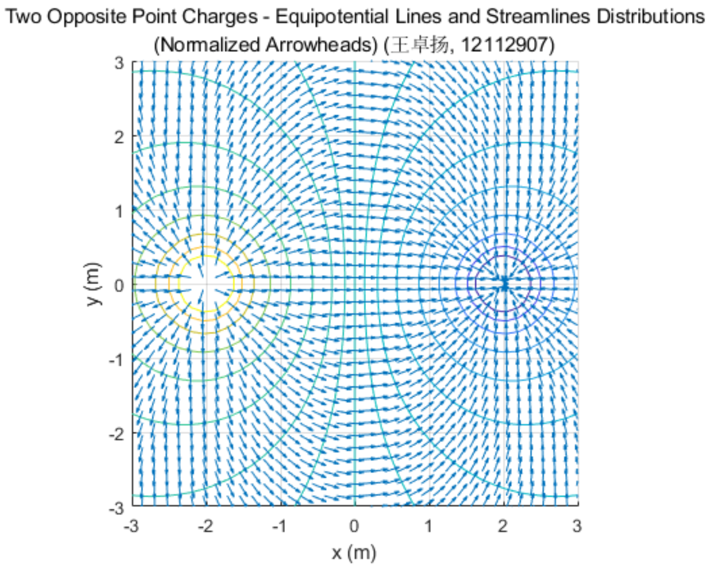

Figure 8: Two Opposite Point Charges - Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)

As shown in Figure 4, the arrows are consistent with **E**.

D Case 3: Electric field distribution of three identical point charges located at the vertices of an equilateral triangle

In this case we studied the electric field distribution of three identical point charges located at the vertices of an equilateral triangle.

D.1 Declarations

First we set up the scene by placing the charges with certain positions and charges.

```
%% Definitions
% Charges
Q_1 = Charge(8e-9, Point([-sqrt(3), -1]));
Q_2 = Charge(8e-9, Point([sqrt(3), -1]));
Q_3 = Charge(8e-9, Point([0, 2]));
% Viewport
range_size = 8;
range_samples = 80;
x_range = linspace(-range_size / 2, range_size / 2, range_samples);
y_range = linspace(-range_size / 2, range_size / 2, range_samples);
[x_mesh, y_mesh] = meshgrid(x_range, y_range);
```

D.2 Potential Distribution

D.2.1 MATLAB Code

We can use $Charge.EvalPotentialField$ to obtain the potential field V and plot it out.

```
%% Potential Distribution
V = Q_1.EvalPotentialField(x_mesh, y_mesh) + Q_2.EvalPotentialField(x_mesh,
↪ y_mesh) + Q_3.EvalPotentialField(x_mesh, y_mesh);

```

```
figure(1);
mesh(x_mesh, y_mesh, V);
zlim([0, 600]);
title("Three Identical Point Charges - Potential Distribution (Wang Zhuoyang,
↪  12112907)");
xlabel("x (m)"), ylabel("y (m)"), zlabel("V (V)");
```

D.2.2 Result and Analysis

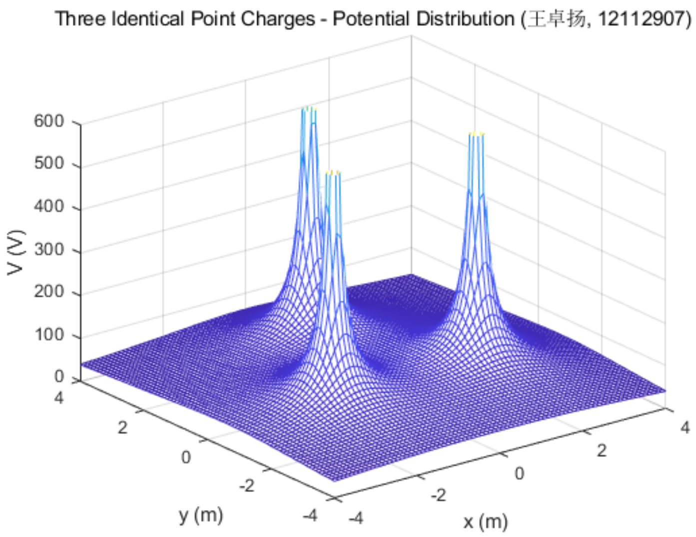

Figure 9: Three Identical Point Charges - Potential Distribution

As shown in Figure 9, there are three infinite peaks in the potential field, which is in line with the expectations.

D.3 Contours Distribution

D.3.1 MATLAB Code

To plot the contours we need to generate an array of potential values using *linspace*. Then we can employ the function *contour* to draw a series of potential contours as below.

```
%% Contours Distribution
V_min = 0;
V_max = 6.5;
V_samples = 21;
V_eq = linspace(V_min, V_max, V_samples) .^ 3;

figure(2);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
title("Three Identical Point Charges - Contours Distribution (Wang Zhuoyang,
↪  12112907)");
xlabel("x (m)"), ylabel("y (m)");
```

D.3.2 Result and Analysis

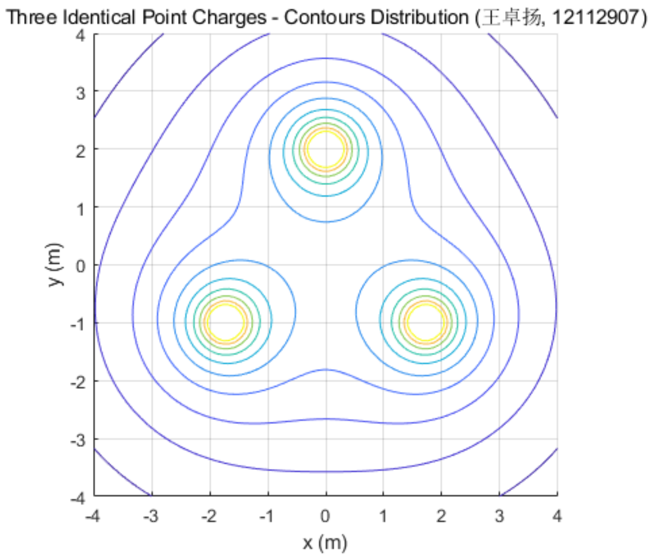

Figure 10: Three Identical Point Charges - Contours Distribution

As shown in Figure 10, the equipotential lines are distributed around the charges and conform to the potential distribution diagram above.

D.4 Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)

D.4.1 MATLAB Code

streamline can plot lines along the vector field from a specified start point. We use *gradient* to figure out the gradient of $-V$, which is the electric field intensity $\mathbf{E}$. Then we take the points surrounding the charges as the start points to plot the streamlines.

```
%% Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)
[E_x, E_y] = gradient(-V);

angle_samples = 4 * (1) + 4 + 1;
angle_dist = 0.1;
```

```
angle = linspace(-pi / 8, pi + pi / 8, angle_samples);
angle_u = [angle, -pi / 3.45, -pi + pi / 3.45];
angle = [angle, -pi / 2.7, -pi + pi / 2.7];

figure(3);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
streamline(x_mesh, y_mesh, E_x, E_y, angle_dist * cos(angle + pi / 3 * 2) + Q_1.p.x, angle_dist * sin(angle + pi / 3 * 2) + Q_1.p.y);
streamline(x_mesh, y_mesh, E_x, E_y, angle_dist * cos(angle - pi / 3 * 2) + Q_2.p.x, angle_dist * sin(angle - pi / 3 * 2) + Q_2.p.y);
streamline(x_mesh, y_mesh, E_x, E_y, angle_dist * cos(angle_u) + Q_3.p.x, angle_dist * sin(angle_u) + Q_3.p.y);
title(["Three Identical Point Charges - Equipotential Lines and Streamlines Distributions", "(Smooth Continuous Curves) (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
```

D.4.2 Result and Analysis

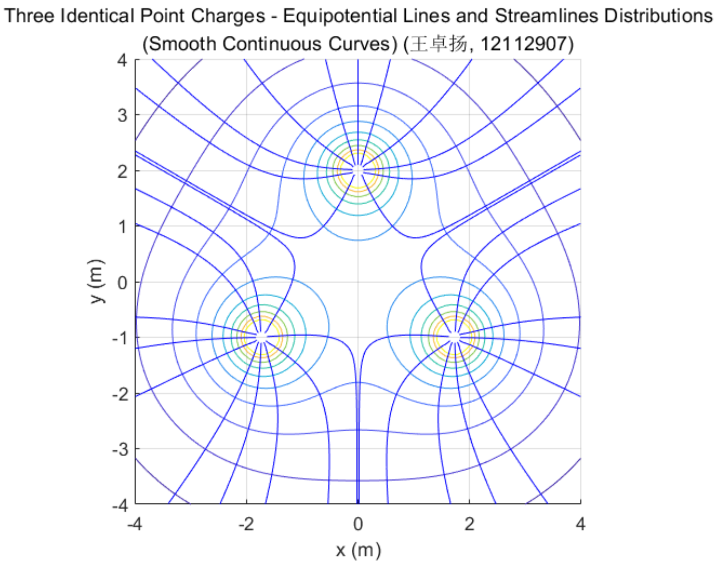

Figure 11: Three Identical Point Charges - Equipotential Lines and Streamlines Distributions (Smooth Continuous Curves)

As is shown in Figure 11, the streamlines are perpendicular to the equipotential lines, which conforms to the theoretical derivation.

D.5 Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)

D.5.1 MATLAB Code

Finally we use another method to draw streamlines distributions. We just simply draw the arrows to indicate the electric intensity field **E** as below.

```
%% Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)
E = sqrt(E_x .^ 2 + E_y .^ 2);
E_x_normal = E_x ./ E;
E_y_normal = E_y ./ E;

```

```
figure(4);
hold on;
grid on;
axis equal;
contour(x_mesh, y_mesh, V, V_eq);
quiver(x_mesh, y_mesh, E_x_normal, E_y_normal);
axis([-2.5, 2.5, -2, 3]);
title(["Three Identical Point Charges - Equipotential Lines and Streamlines
↪  Distributions", "(Normalized Arrowheads) (Wang Zhuoyang, 12112907)"]);
xlabel("x (m)"), ylabel("y (m)");
```

D.5.2 Result and Analysis

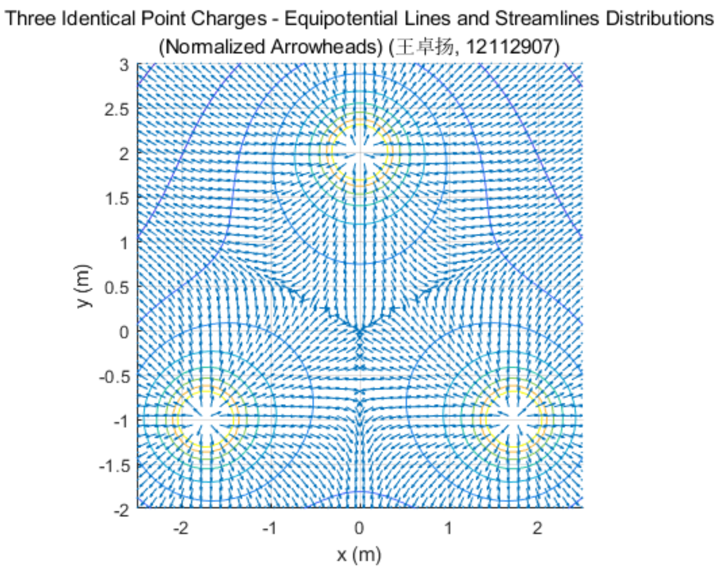

Figure 12: Three Identical Point Charges - Equipotential Lines and Streamlines Distributions (Normalized Arrowheads)

As shown in Figure 4, the arrows are consistent with **E**.

E Conclusion

From this laboratory we have learnt that how to use fundamental functions in MATLAB to plot the vector fields, like *meshgrid*, *contour*, *streamline* and *quiver*.

Also, the clear and intuitive images deepened our understanding of what we have learned in class.

In particular, unlike the theoretical analysis, the results of the numerical analysis are very dependent on the setting of the sampling points. In order to get the appropriate results, we need to design a reasonable sampling density and sampling range.