CSE 276C Homework 3

Adrian L. Pavlak
U09830551

15. November 2023

1 Question 1

Consider the following differential equation over the interval (0, 1]:

$$\frac{dy}{dx} = \frac{1}{x^2(1-y)}$$

with y(1) = -1. Often we use this function form to describe forces when navigating through a field of obstacles. In its general form, this is called potential field navigation and borrows numerous results from physics.

(a) Obtain an exact analytical solution to the equation. Solve for y(0) (even though in theory the equation is not defined for x=0).

There are a few different ways to solve differential equations if they are in a specific form. Our equation is in the form of a separable differential equation $\frac{dy}{dx} = f(x)g(y)$ and we can solve it like this:

$$\frac{dy}{dx} = f(x)g(y)$$
$$\frac{dy}{g(y)} = f(x)dx$$
$$\int \frac{1}{g(y)}dy = \int f(x)dx$$

For our equation with the initial value condition, we get:

$$\frac{dy}{dx} = \frac{1}{x^2(1-y)}$$
$$(1-y)dy = 1/(x^2)dx$$
$$\int (1-y)dy = \int 1/(x^2)dx$$

We integrate on both sides and get:

$$y - \frac{y^2}{2} + C_1 = -\frac{1}{x} + C_2$$
$$y - \frac{y^2}{2} = -\frac{1}{x} + C$$
$$y^2 - 2y - \frac{2}{x} + 2C = 0$$

We have a quadratic equation for y, so we can use the quadratic formula:

$$y = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
$$y = \frac{2 \pm \sqrt{4 - 4(-\frac{2}{x} + 2C)}}{2}$$
$$y = 1 \pm \sqrt{1 + \frac{2}{x} - 2C}$$

Now we put in the initial condition y(1) = - 1

$$-1 = 1 \pm \sqrt{1 + \frac{2}{1} - 2C}$$
$$-2 = \pm\sqrt{3 - 2C}$$
$$4 = 3 - 2C$$
$$C = -\frac{1}{2}$$

If we now put in C into the equation for y:

$$y = 1 \pm \sqrt{1 + \frac{2}{x} - 2C}$$
$$y = 1 \pm \sqrt{1 + \frac{2}{x} + 1}$$
$$y = 1 \pm \sqrt{2 + \frac{2}{x}}$$

If we put in y(1) = -1

$$-1 \neq 1 + \sqrt{2 + \frac{2}{1}} = 3 \qquad\qquad -1 = 1 - \sqrt{2 + \frac{2}{1}} = -1$$

We keep the one that is right in terms of the initial condition. Note that the expression $y(x) = 1 - \sqrt{2 + \frac{2}{x}}$ is not defined for x = 0, so the equation is not valid for x = 0. But if we take the limit as x approaches 0 from the positive side.

$$\lim_{x\to 0^+} y(x) = \lim_{x\to 0^+} 1 - \sqrt{2 + \frac{2}{x}} = -\infty$$

So we get the analytical expression for y and y(0):

$$y = 1 - \sqrt{2 + \frac{2}{x}} \qquad\qquad \lim_{x\to 0^+} y(x) = -\infty$$

(b) Implement and use Euler's method to solve the differential equation numerically. Use a step size of 0.05. How accurate is your numerical solution?

The forward Eulers method to solve a differential equation numerically uses the formula

$$y_{n+1} = y_n + hf(x_n, y_n)$$

$$h = 0.05(stepsize) \qquad\qquad f(x_n, y_n) = \frac{1}{x_n^2(1 - y_n)}$$

Since I need to find the solution from (0,1] and I have my initial condition at y(1) = -1 I use the Euler method with negative step size. When I integrated from [0, 1], I got the plot:

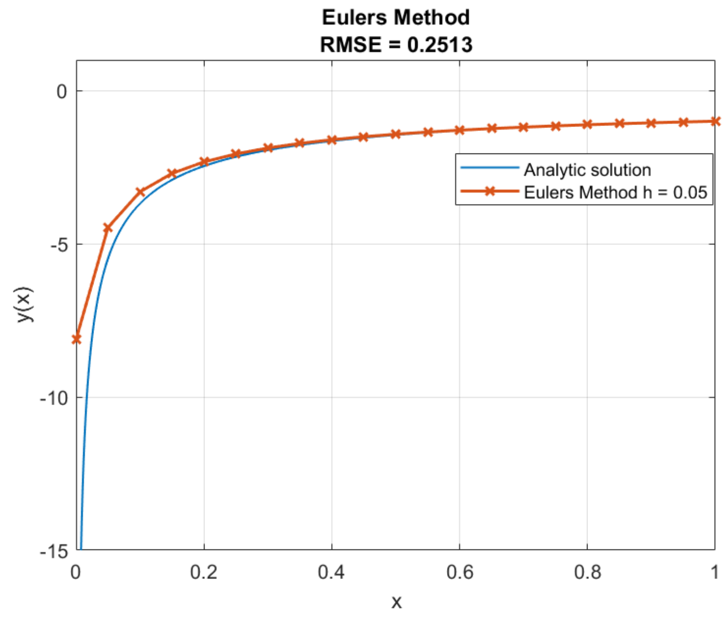


Figure 1: Euler integrate from [0, 1]

As seen in the figure the Euler integration does well from [0.2,1] and the numerical solution is close to the analytic. When x goes to 0 the analytic and numerical solution differs and this is because the derivative starts to get really negative and the Euler integration fails to keep track of the analytic solution. If we had a smaller step size the Euler would to better. The Root Mean Square error(RMSE) is 0.251. When computing the RMSE I have excluded the x=0 since the analytic solution goes down to $-\infty$.

(c) Implement and use a fourth-order Runge-Kutta method to solve the differential equation numerically. Again, use a step size of 0.05. Again, how accurate is your numerical solution?

The fourth-order Runge-Kutta method I used looked like this:

$$\begin{aligned} k_1 &= h \cdot f(x_n, y_n) \\ k_2 &= h \cdot f(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= h \cdot f(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= h \cdot f(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Where h and f are the same as for the Euler method. I integrated the same intervals [0,1] and here are the results:

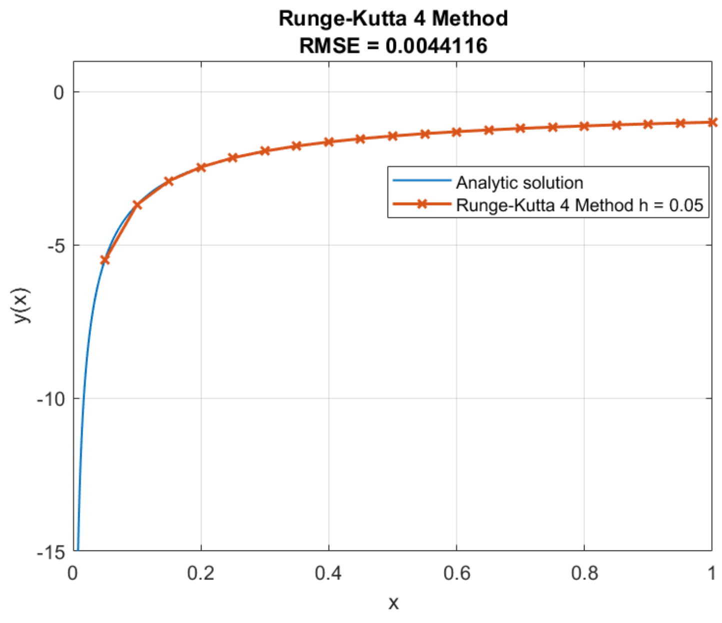


Figure 2: Runge-Kutta 4 integrate from [0, 1]

We see the same pattern in Runge-Kutta 4 as for Euler. Runge-Kutta 4 integration does well from [0.2, 1] with a Root Mean Square error(RMSE) of 0.004, much lower than for Euler, is also seen in figure 3 that the Runge-Kutta 4 follows the analytic solution better then the Euler. The last value of the Runge-Kutta method is actual $-\infty$ compared to Euler value of -8.12. This indicates that the Runge-Kutta method is a better at integrating these functions as we can tell from theory.

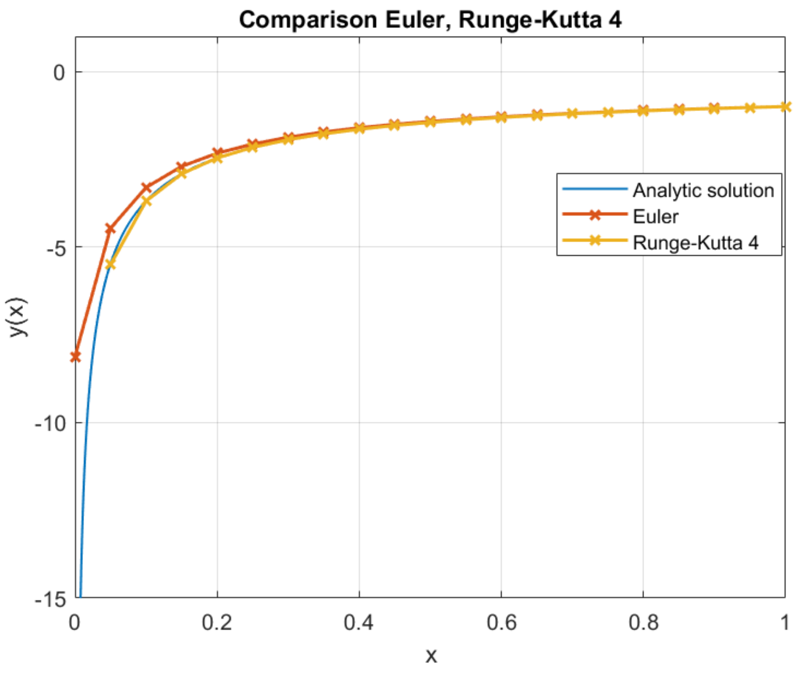


Figure 3: Runge-Kutta 4 and Euler integrate from [0, 1].

(d) Implement and use a Richardson extrapolation to solve the equation, again with a step size of 0.05. How accurate is your solution compared to the analytical solution?

For Richardson extrapolation, we use the formula given on the slide from the lecture 7:

$$\begin{aligned} z_0 &= y(x_n) \\ z_1 &= z_0 + hf(x_n, x_0) \\ z_{m+1} &= z_{m-1} + 2hf(x_n + mh, z_m), m = 1, 2, 3, n-1 \\ y(x_n + h) &= \frac{1}{2}(z_n + z_{n-1} + hf(x + H, z_n) \end{aligned}$$

Where $H = 0.05$ is the stepsize and $n = 2, 3, 6, 8, 10, 12, n_j = 2 * (j + 1)$. The idea is to divide the section from for example $(1, 0.95)$ into n parts and then do a kind of Euler integration to get a better estimate. We do this for every x and as we get further away from x_0 we have a higher to compensate. The results can be seen in here:

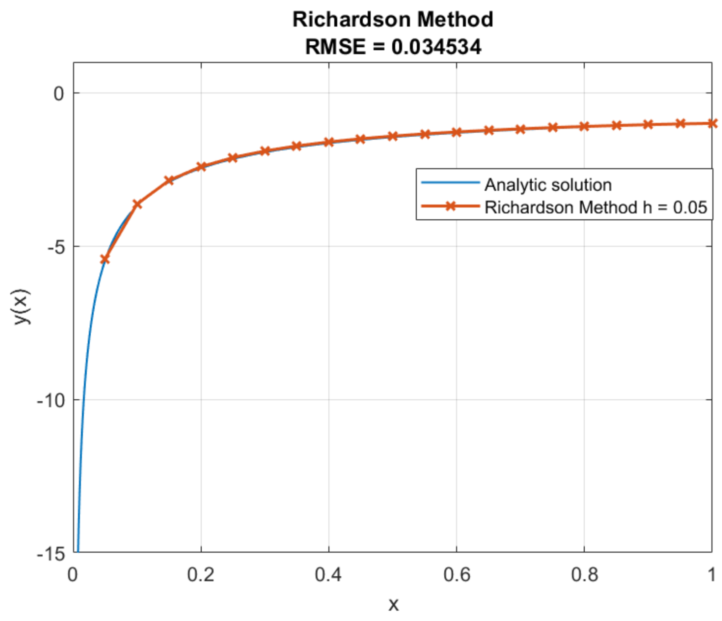


Figure 4: Richardson extrapolation from [0, 1]

The Richardson method follows the analytic solution well and has an RMSE of 0.0345 which is better than the Euler but larger than the Runge-Kutta 4. If we choose a more computationally complex way of doing Richardson, like fixing the n to be 1000 each iteration we would get a better RMSE value.

In 5 you see the results of all the three integration methods plotted against each other. The Euler does worst while Runge-Kutta 4 and Richardson seem to follow the analytic solution better, this is also seen by comparing the RMSE.

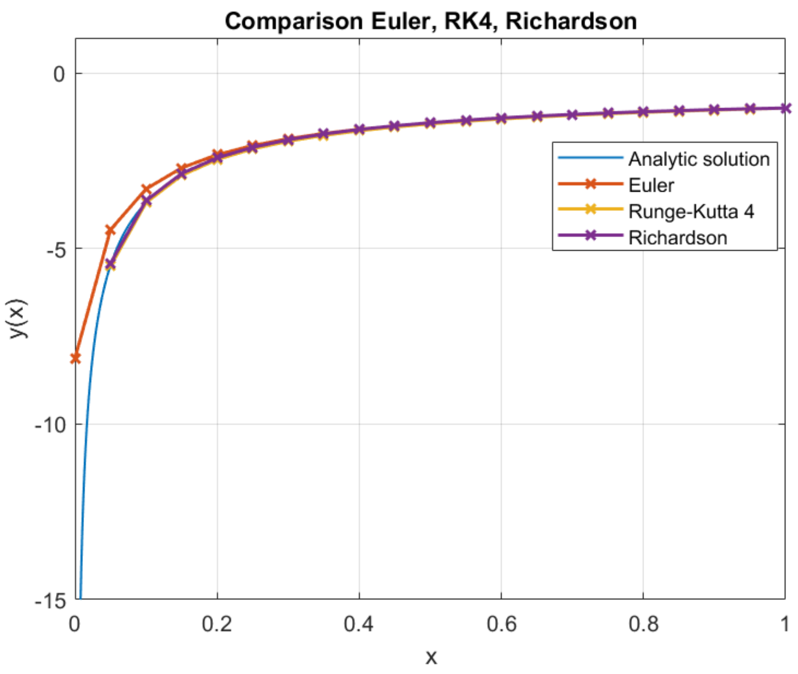


Figure 5: Richardson, Runge-Kutta 4 and Euler integration.

```
%% Problem 1
clear all; clc;

% The analytic solution Parameters
x_analytic = (1:-0.001:0);
y_analytic = analytic(x_analytic);

% Parameters
x0 = 1;
y0 = -1;
x_end = 0;
h = -0.05;

%% 1a) Eulers method
[x_Euler, y_Euler, RMSE_Euler] = Eulers_method(x0,y0,x_end,h);
Plot_solution(x_Euler,y_Euler,RMSE_Euler,"Eulers Method",x_analytic, y_analytic);

%% 1b) Runge-Kutta 4
[x_RK4, y_RK4, RMSE_RK4] = RK4(x0,y0,x_end,h);
Plot_solution(x_RK4,y_RK4,RMSE_RK4,"Runge-Kutta 4 Method",x_analytic, y_analytic);
Plot_comparison_Euler_RK4(x_Euler,y_Euler,y_RK4,x_analytic,y_analytic);

%% 1c) Richardson extrapolation
[x_Rich, y_Rich, RMSE_Rich] = Richardson(x0,y0,x_end,h);
Plot_solution(x_Rich,y_Rich,RMSE_Rich,"Richardson Method",x_analytic, y_analytic);

% Compare
Plot_comparison(x_Euler,y_Euler,y_RK4,y_Rich,x_analytic,y_analytic);

%% Functions
function y = f(x,y)
    y = 1/(x^2*(1-y));
end

function y = analytic(x)
    y = 1 - sqrt(2+2./x);
end

function error = RMS_error(x,y)
    y_correct = analytic(x(1:end-1));
    E     = y_correct-y(1:end-1);
    SQE   = E.^2;
    MSE   = mean(SQE(:));
    error = sqrt(MSE);
end

function [x,y,RMSE] = Eulers_method(x0,y0,x_end,h)
    x = (x0:h:x_end);
    y = zeros(1,length(x));
    y(1)  = y0;

    for i = 1:(length(x)-1)
        y(i+1)  = y(i)  + h*f(x(i),y(i));
    end
    RMSE = RMS_error(x,y);
end

function [x,y,RMSE] = RK4(x0,y0,x_end,h)
    x = (x0:h:x_end);
    y = zeros(1,length(x));
    y(1) = y0;

    for i = 1:(length(x)-1)
        xi = x(i);
        yi = y(i);

        k1 = h * f(xi, yi);
        k2 = h * f(xi + h/2, yi + k1/2);
        k3 = h * f(xi + h/2, yi + k2/2);
        k4 = h * f(xi + h, yi+ k3);

        y(i+1) = yi + (1/6)*(k1+2*k2+2*k3+k4);

    end
    RMSE = RMS_error(x,y);
end

function [x,y,RMSE] = Richardson(x0,y0,x_end,H)
    x = (x0:H:x_end);
    y = zeros(1,length(x));
    y(1) = y0;

    for i = 1:(length(x)-1)
        n = 2*(i+1); % Variable step size as in the slide
        h = H/n;

        z = zeros(n,1);
        z(1) = y(i);
        z(2) = z(1) + h*f(x(i),z(1));
        for m = 2:(n-1)
            z(m+1) = z(m-1) + 2*h*f(x(i)+m*h,z(m));
        end
        y(i+1) = 0.5*(z(n)+z(n-1) + h*f(x(i)+ H, z(n)));
    end
    RMSE = RMS_error(x,y);
end

function Plot_solution(x_sol,y_sol, RMSE, method,x_analytic, y_analytic)
    figure();
    plot(x_analytic, y_analytic,'LineWidth',1);hold on;
    plot(x_sol,y_sol,'-x','LineWidth',1.5);
    title({[method] ["RMSE = " + num2str(RMSE)]});
    ylim([-15,1]);
    xlabel("x");
    ylabel("y(x)");
    legend("Analytic solution", method + " h = 0.05");
    grid();
end
```

```
function Plot_comparison_Euler_RK4(x, y_Euler, y_RK4, x_analytic, y_analytic)
    figure();
    plot(x_analytic, y_analytic,'LineWidth',1);hold on;
    plot(x,y_Euler,'-x','LineWidth',1.5);
    plot(x,y_RK4,'-x','LineWidth',1.5);
    title("Comparison Euler, Runge-Kutta 4");
    ylim([-15,1]);
    xlabel("x");
    ylabel("y(x)");
    legend("Analytic solution", "Euler", "Runge-Kutta 4");
    grid();
end

function Plot_comparison(x, y_Euler, y_RK4, y_Rich,x_analytic, y_analytic)
    figure();
    plot(x_analytic, y_analytic,'LineWidth',1);hold on;
    plot(x,y_Euler,'-x','LineWidth',1.5);
    plot(x,y_RK4,'-x','LineWidth',1.5);
    plot(x,y_Rich,'-x','LineWidth',1.5);
    title("Comparison Euler, RK4, Richardson");
    ylim([-15,1]);
    xlabel("x");
    ylabel("y(x)");
    legend("Analytic solution", "Euler", "Runge-Kutta 4", "Richardson");
    grid();
end
```

2 Question 2

We have multiple robots that can generate point clouds such as those coming from a RealSense camera. In many cases we want to use robots to detect objects in its environment. We provide three data files. Each file has the point cloud data in a format where each line contains xi yi zi.

(a) Provide a method to estimate the plane parameter for the table. Test it both with the empty and cluttered table. Describe how you filter out the data from the objects. You have to be able to estimate the table parameters in the presence of clutter.

In this problem, we are going to find the parameters for a plane. The equation for a plane is given by:

$$a(x - x_0) + b(y - y_0) + c(z - z_0) + d = 0$$

Where $\hat{n} = [a, b, c]^T$ is the normal vector of the plane and x_0, y_0, z_0 is a point in the plane. This normal vector can be found if you know three points in the plane for example P1, P2 and P3:

$$\begin{aligned}
P_1 &= (2, 1, 4)\\
P_2 &= (4, -2, 7)\\
P_3 &= (5, 3, -2)\\
\mathbf{v}_1 &= \mathbf{P}_2 - \mathbf{P}_1 = [2, -3, 3]^T\\
\mathbf{v}_2 &= \mathbf{P}_3 - \mathbf{P}_1 = [3, 2, -6]^T\\
\hat{\mathbf{n}} &= \mathbf{v}_1 \times \mathbf{v}_2 = [12, 21, 13] = [a, b, c]\\
a(x - x_0) + b(y - y_0) + c(z - z_0) + d &= 0\\
12(x - 2) + 21(y - 1) + 13(z - 4) &= 0\\
12x + 21y + 13z &= 97
\end{aligned}$$

If I only had three nice points in my dataset, to find the plane I would just follow the procedure above and get my equation for the plane, but since I have a lot of noise and clutter I need to only use points I know is in the plane. For this, I will use the RANSAC algorithm and Kmeans clustering.

The Kmeans clustering clusters the large point clouds into smaller clusters. This filters the data so the point cloud looks more like three large clusters, two chairs, and a table. I tried different values and with 1000 clusters I got a nice clustering of the point cloud. From this clustered pointcloud I use the RANSAC algorithm to find the parameters for a plane like I described above. The RANSAC algorithm goes like this:

1. Sample (randomly) the number of points required to fit the model, in our case, this is going to be three points.

2. Solve for model parameters using samples, in our case, this will be the derivation I showed above.

3. Score by the fraction of inliers within a preset threshold of the model. In our case, three points make a plane and we count the points in a threshold T distance from this plane if we get more inliers than any previous iteration we update the plane parameters.

I repeat this N = 10000 number of times with a threshold of 0.01 until I get the best model for the plane.

Here are the results for the two planes, Table and Empty table:

$$\begin{aligned} Empty: &\quad (0.0021)x + (-0.9124)y + (-0.2094)z + (0.5720)d = 0 \\ Table: &\quad (0.0022)x + (-0.9165)y + (-0.4001)z + (0.5589)d = 0 \end{aligned}$$

These planes are almost the same and when plotted in the point cloud the plane seems to fit the table really well.

Here is a plot of the plane in the clustered and the original pointcloud for the empty table and table.

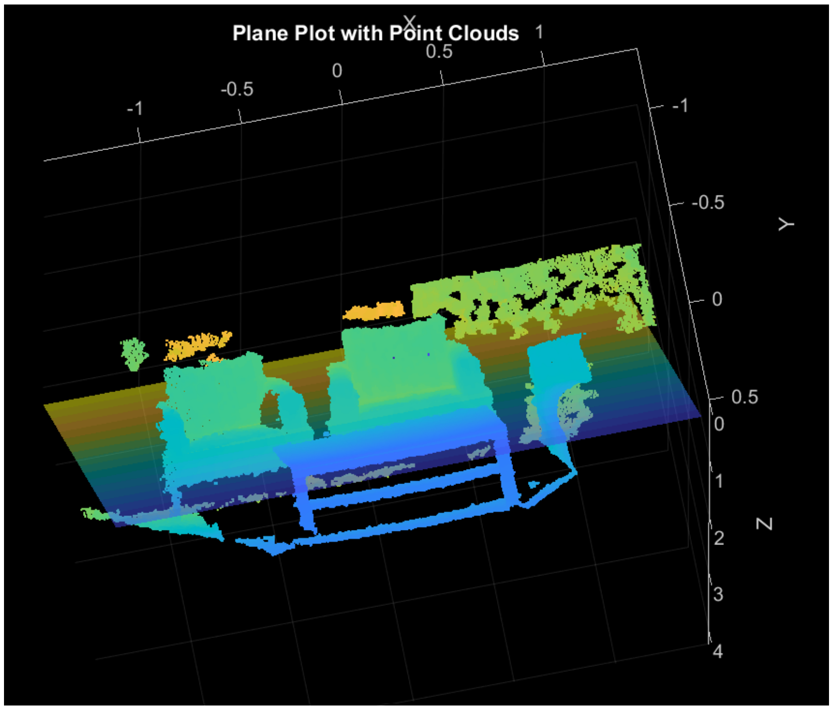


Figure 6: Empty table with the plane in the original point cloud.

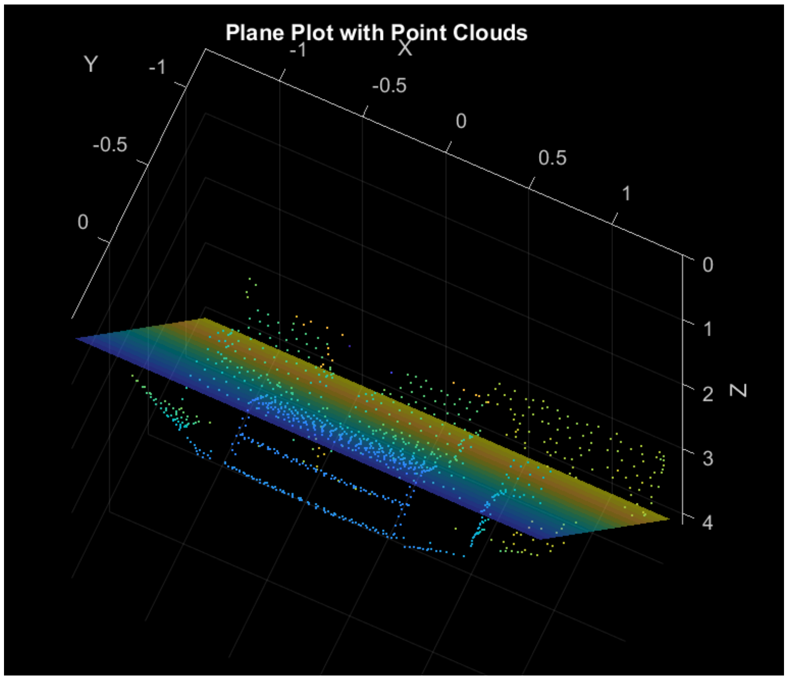


Figure 7: Empty table clustered with the plane in the clustered point cloud.

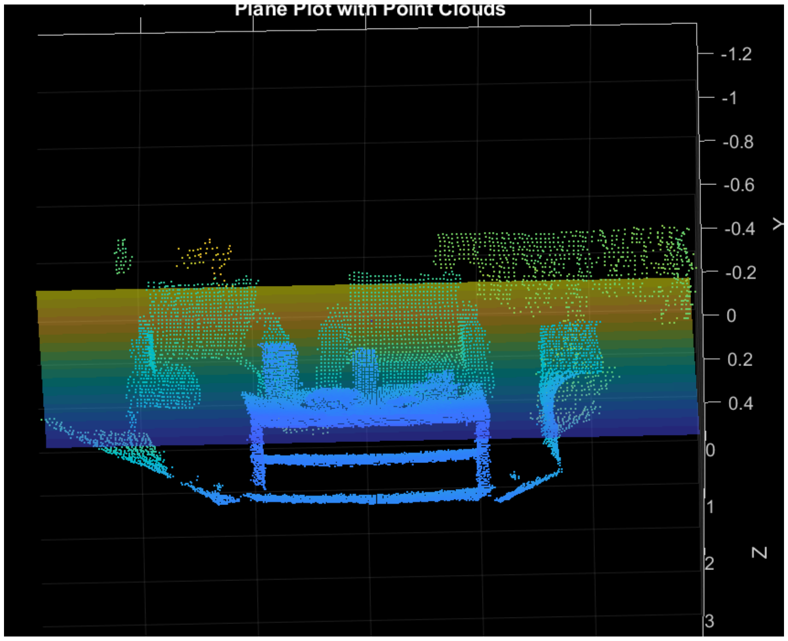


Figure 8: Table with objects and the plane in the original point cloud.

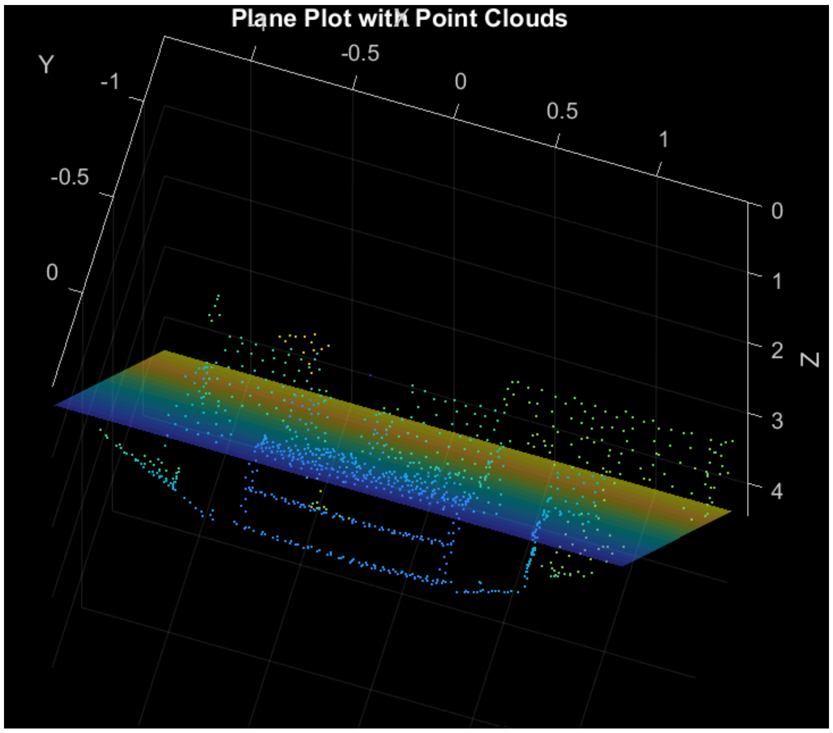


Figure 9: Table with objects and the plane in the clustered point cloud.

(b) Describe and show how the method can be generalized to extract all the dominant planes of the CSE building viewed in the outdoor environment.

There are several ways to generalize the method to extract all the dominant planes of the CSE building. I got inspiration from [1] but made an algorithm by my self.

I take advantage/or I assume that we know we are looking for vertical and horizontal planes. A vertical plane would have a normalized normal vector with a low value for $|c|$, if $\boldsymbol{n} = [a, b, c]$ and a horizontal plan would have a high value for $|c|$. We update the RANSAC algorithm with the criterion to only update the best plane if $|c| < 0.01$ or $|c| > 0.99$ and then follow the steps.

1. Use Kmeans clustering to filter the pointcloud.

2. Use the RANSAC algorithm modified to store the plane found and return the inliers from the plane found.

3. Remove all the points that are inliers*, in this way we remove the plane.

4. Now run the RANSAC algorithm again.

*A note here is I have made a new function "return_inliers". This function takes in that plane from the RANSAC and find the inliers but with a larger threshold. This way I can find the best points to use to get a plane and then remove all the points associated/close to the plane. See this in the code below.

By using this algorithm we extract all the dominant planes from the pointcloud, but there are some problems with this solution.

One is that we do not know how many planes they are. For instance, this can be solved by having a threshold of how many inliers you need to classify it as a plane. In my case I tried with different numbers and ended up at 4.

Another problem is that with my algorithm I only search for vertical and horizontal planes. This is an assumption I made by looking at the raw data but in reality, we can have a plane that has an angle but for this problem, I did not include that.

Below is the plots of the plane. I have plotted the clustered point cloud after removing the previous planes together with the planes and then the plane in the original point cloud.

The most dominant plane **??** is the ground plane which gives meaning since it has points spread at the same z-value throughout the whole point cloud.

The second most dominant plane 12 seems to be a long wall.

If you look at the reduced clustered pointcloud now there seem to be two more planes and that is 12 and 13.

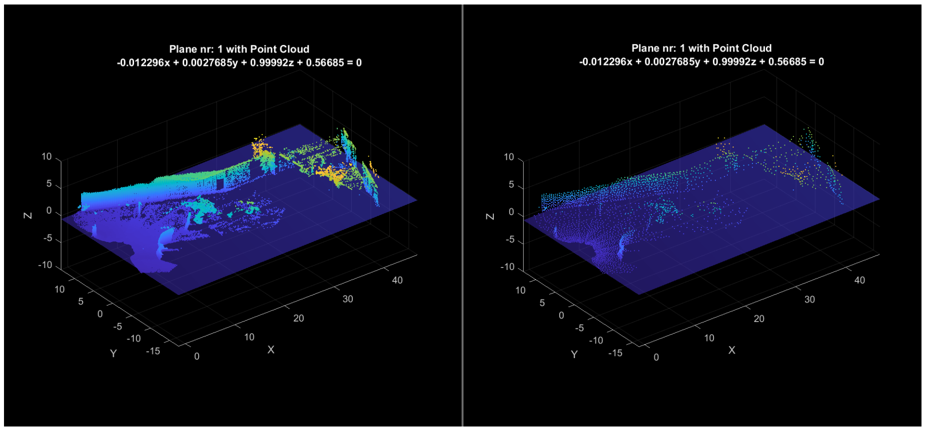

Plane nr: 1 with Point Cloud
-0.012296x + 0.0027685y + 0.99992z + 0.56685 = 0
Plane nr: 1 with Point Cloud
-0.012296x + 0.0027685y + 0.99992z + 0.56685 = 0

Figure 10: Most dominate plane.

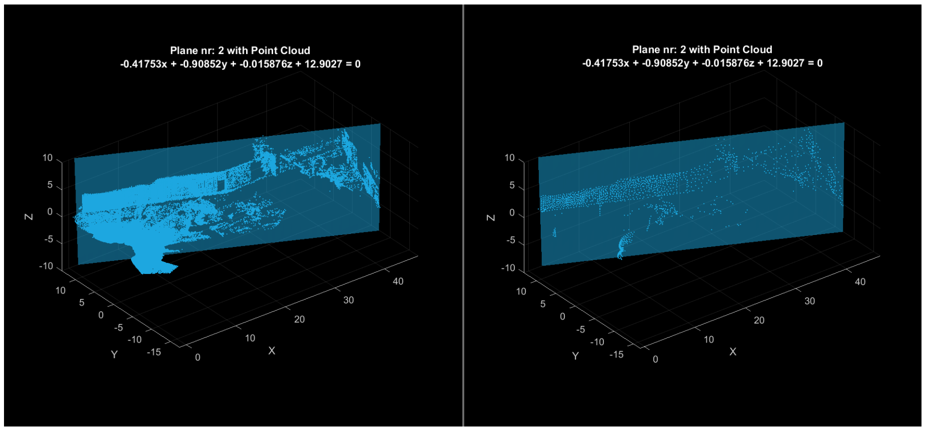
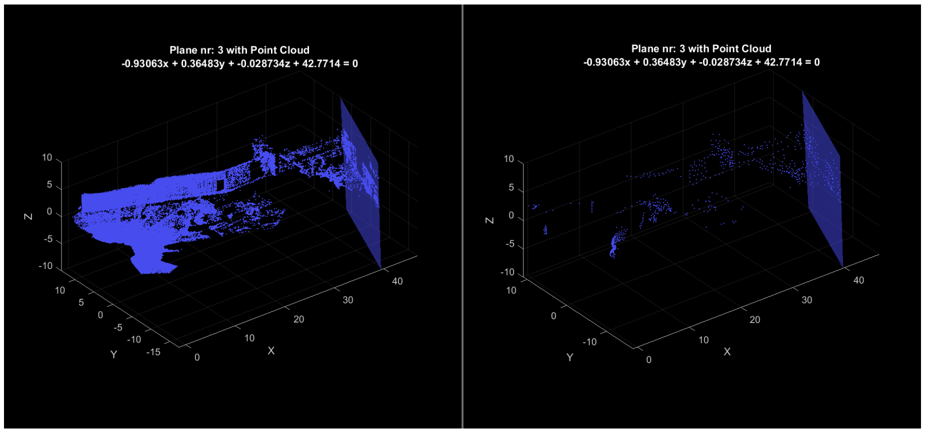
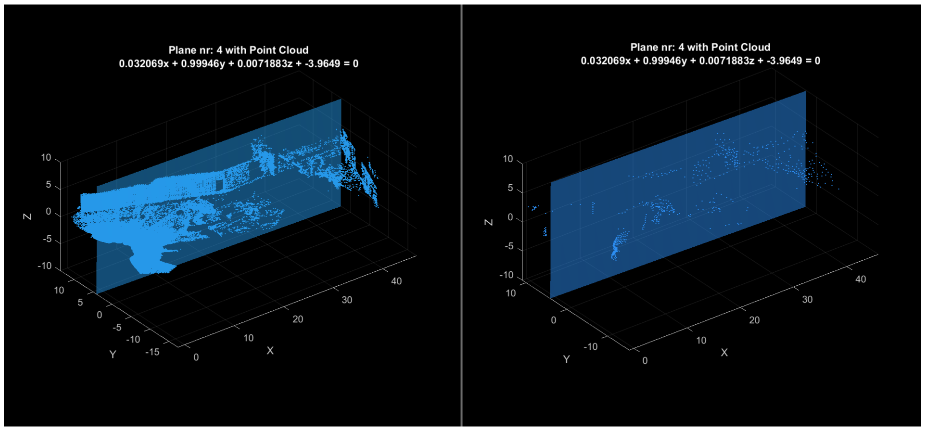

Plane nr: 2 with Point Cloud
-0.41753x + -0.90852y + -0.015876z + 12.9027 = 0
Plane nr: 2 with Point Cloud
-0.41753x + -0.90852y + -0.015876z + 12.9027 = 0

Figure 11: Second most dominate plane.



Plane nr: 3 with Point Cloud
-0.93063x + 0.36483y + -0.028734z + 42.7714 = 0
Plane nr: 3 with Point Cloud
-0.93063x + 0.36483y + -0.028734z + 42.7714 = 0
Z
Y
X

Figure 12: Third most dominate plane.



Plane nr: 4 with Point Cloud
0.032069x + 0.99946y + 0.0071883z + -3.9649 = 0
Plane nr: 4 with Point Cloud
0.032069x + 0.99946y + 0.0071883z + -3.9649 = 0
Z
Y
X

Figure 13: Fourth most dominate plane.

```
clear all; clc;

%% Import the point clouds
empty = load('Empty2.asc');
table = load('TableWithObjects2.asc');
cse = load('CSE.asc');

pointCloud = empty;

%% Cluster the pointcloud to filter it
numClusters = 1000; % I tuned and 1000 worked well
[idx, pointCloud_clustered] = kmeans(pointCloud, numClusters);

%% Use RANSAC to estimate the plane parameters
[a,b,c,d,inliers] = get_plane_RANSAC(pointCloud_clustered);
plot_plane(a,b,c,d,pointCloud_clustered);
plot_plane(a,b,c,d,pointCloud);

%% Functions
function [a,b,c,d, inliers_selected] = get_plane_RANSAC(point_cloud)
    N = 10000;
    T = 0.01;
    k = 0;
    best_model = [];
    best_num_inliers = 0;
    best_inliers = [];

    while k < N
        % Randomly select 3 points
        indices = randperm(size(point_cloud, 1), 3);
        points = point_cloud(indices, :);

        % Check if the points are collinear
        while det(points) == 0
            indices = randperm(size(point_cloud, 1), 3);
            points = point_cloud(indices, :);
        end

        % Compute the plane parameters from the 3 points
        P1 = points(1, :);
        P2 = points(2, :);
        P3 = points(3, :);

        v1 = P2 - P1;
        v2 = P3 - P1;
        n = cross(v1, v2);
        n = n / norm(n);

        a = n(1);
        b = n(2);
        c = n(3);
        d = - (a * P1(1) + b * P1(2) + c * P1(3));

        % Compute the error
        error = abs(a * point_cloud(:, 1) + b * point_cloud(:, 2) + c * point_cloud↙
(:, 3) + d) / sqrt(a^2+b^2+c^2);

        % Count the inliers
        inliers = error < T;

        % Check if the model is better
        num_inliers = sum(inliers);
        if num_inliers > best_num_inliers
            best_model = [a, b, c, d];
            best_num_inliers = num_inliers;
            best_inliers = inliers;

            disp('The best model is: ');
            disp(best_model);
        end
        k = k + 1;
    end
    a = best_model(1);
    b = best_model(2);
    c = best_model(3);
    d = best_model(4);
    inliers_selected = best_inliers;
end

function plot_plane(a,b,c,d,point_cloud)
    figure;
    pcshow(point_cloud); hold on;

    % Plot the plane
    [x, y] = meshgrid(min(point_cloud(:, 1)):0.1:max(point_cloud(:, 1)), min↙
(point_cloud(:, 2)):0.1:max(point_cloud(:, 2)));
    z = (-a * x - b * y - d) / c;
    surf(x, y, z, 'EdgeColor', 'none', 'FaceAlpha', 0.5);

    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    title('Plane Plot with Point Clouds');
    axis equal;
    grid on;
    view(3);
    hold off;
end
```

```
clear all; clc;

%% Import the point cloud for the CSE building
cse = load('CSE.asc');
pointCloud = cse;

%% Cluster the pointcloud to filter it
numClusters = 3000; % I needed more detail here so 3000 worked well
[idx, pointCloud_clustered] = kmeans(pointCloud, numClusters);

%% Find the best plane, then remove it and do it again
NUM_PLANES = 4; % Just my "guess" of how many planes
plane_parameters = [];
remainingPoints = pointCloud_clustered;

% Find the best plane from modified RANSAC and remove the inliers
for i = 1:NUM_PLANES
    % Get the plane
    [a,b,c,d,inliers] = get_plane_RANSAC(remainingPoints);
    plane_parameters = [plane_parameters; a,b,c,d];

    % Plot the plane in the removed clustered pointcolud and original
    plot_plane(a,b,c,d,remainingPoints, i);
    plot_plane(a,b,c,d,pointCloud, i);

    % Remove the outliers but in the modified function to have a larger
    % threshold
    remainingPoints = remove_inilers(remainingPoints,a,b,c,d);
end

%% Functions
% Modified RANSAC to only store the horizontal and vertical planes
function [a,b,c,d, inliers_selected] = get_plane_RANSAC(point_cloud)
    N = 100000;
    T = 0.01;
    k = 0;

    best_model = [];
    best_num_inliers = 0;
    best_inliers = [];

    while k < N
        % Randomly select 3 points
        indices = randperm(size(point_cloud, 1), 3);
        points = point_cloud(indices, :);

        % Check if the points are collinear
        while det(points) == 0
            indices = randperm(size(point_cloud, 1), 3);
            points = point_cloud(indices, :);
        end

        % Compute the plane parameters from the 3 points
        P1 = points(1, :);
        P2 = points(2, :);
        P3 = points(3, :);

        v1 = P2 - P1;
        v2 = P3 - P1;
        n = cross(v1, v2);
        n = n / norm(n);

        a = n(1);
        b = n(2);
        c = n(3);
        d = - (a * P1(1) + b * P1(2) + c * P1(3));

        % Compute the error
        error = abs(a * point_cloud(:, 1) + b * point_cloud(:, 2) + c * point_cloud↙
(:, 3) + d) / sqrt(a^2+b^2+c^2);

        % Count the inliers
        inliers = error < T;

        % Check for normalvector that corrospond to a vertical plane or
        % horizontal plane
        normal_z = abs(n(3));
        min_z = 0.2;    % I have more slack on the vertical
        max_z = 0.9999; % Horizontal is set to be higher

        % Check if the model is better
        num_inliers = sum(inliers);
        if (num_inliers > best_num_inliers) && (normal_z < min_z || normal_z >↙
max_z)
            best_model = [a, b, c, d];
            best_num_inliers = num_inliers;
            best_inliers = inliers;
        end
        k = k + 1;
    end
    a = best_model(1);
    b = best_model(2);
    c = best_model(3);
    d = best_model(4);
    inliers_selected = best_inliers;
    clc;
    disp('The best model is: ');
    disp(best_model);
end

% Modified remaining points
function remainingPoints = remove_inilers(point_cloud, a,b,c,d)
        % Compute the error
        error = abs(a * point_cloud(:, 1) + b * point_cloud(:, 2) + c * point_cloud↙
(:, 3) + d) / sqrt(a^2+b^2+c^2);

        % Count the inliers with a higher tolerance
        % I do this do its easier to remove the points beloing to the plane
        % and still get a "strong" plane by having a small threshold in
        % the RANSAC function
        T = 0.5;
        inliers = error < T;

        remainingPoints = point_cloud(~inliers, :);
end

% Plot the plane and the cloud
function plot_plane(a,b,c,d,point_cloud, i)
    figure;
    pcshow(point_cloud); hold on;


    [x, y] = meshgrid(min(point_cloud(:, 1)):0.1:max(point_cloud(:, 1)), min↙
(point_cloud(:, 2)):0.1:max(point_cloud(:, 2)));
    z = (-a * x - b * y - d) / c;
    surf(x, y, z, 'EdgeColor', 'none', 'FaceAlpha', 0.5);

    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    eq = [num2str(a) 'x + ' num2str(b) 'y + ' num2str(c) 'z + ' num2str(d) ' = 0'];
    title(['Plane nr: ', num2str(i), ' with Point Cloud', newline, eq]);
    axis equal;
    grid on;
    view(3);
    zlim([-10, 10]);

    hold off;
end
```

References

[1] Michael Ying Yang and Wolfgang Förster. (2010). *Plane Detection in Point Cloud Data*. Technical Report, University of Bonn, Department of Photogrammetry, Institute of Geodesy and Geoinformation. Retrieved from `http://www.ipb.uni-bonn.de/technicalreports/`