Name: Kai Chuen Tan
Course: CSE 276C.
HW#: Homework 3
Date: 10/23/2021
Professor: Dr. Christensen

Problem 1

$V = 10^3 \sin\sqrt{\pi t}$; $R_1 = 1\,k\Omega$; $C_1 = 2\,mF$

$$\frac{dq}{dt} = \frac{V}{R_1} - \frac{q}{R_1 C_1}$$

$$\frac{dq}{dt} = \frac{10^3 \sin\sqrt{\pi t}}{R_1} - \frac{q}{R_1 C_1}$$

$$\frac{dq}{dt} = \frac{10^3 C_1 \sin\sqrt{\pi t} - q}{R_1 C_1} \quad ; \; q(0) = 4C \; ; \; h = 0.1$$

Determine $q(t)$ at $t = 0.1$

a.) Euler's Method

$t_0 = 0 \qquad t_1 = t_0 + h$

$q_0 = 4C \qquad t_1 = 0 + 0.1$

$t_1 = 0.1$

$$q_1 = q_0 + \frac{dq}{dt} h \qquad ; \; \left.\frac{dq}{dt}\right|_{t=t_i} = f(t_i, q_i) \; ; \; \forall\, i \in \{0, 1, 2, \dots, n\}$$

$$= q_0 + f(t_0, q_0) h$$

$$= 4 + \left[\frac{10^3 C_1 \sin\sqrt{\pi t_0} - q_0}{R_1 C_1}\right] h$$

$$= 4 + \left[\frac{10^3 C_1 \sin\sqrt{\pi (0)} - 4}{R_1 C_1}\right] 0.1$$

$$= 4 + \left[\frac{0 - 4}{R_1 C_1}\right] 0.1$$

$$= 4 - \frac{2}{5 R_1 C_1}$$

$$\boxed{q_1 = 4 - \frac{2}{5(1000)(0.002)} = 3.8\,C \quad ; \text{ hence, using Euler's Method, } q(t) \text{ at } t = 0.1s \text{ is } 3.8\,C.}$$

Name: Kai Chuen Tan

Course: CSE 276C

HW#: Homework 3

Date: 10/23/2021

Professor: Dr. Christensen

Problem 1 (Part 2)

b.) 4^{th} Order Runge-Kutta's Method

$$\left.\frac{dq(t)}{dt}\right|_{t=t_i} = f(t_i, q_i) \quad ; \forall i \in \{0, 1, 2, \dots, n\} \quad ; t_0 = 0\,s \quad ; h = 0.1\,s$$

$$t_1 = t_0 + h = 0 + 0.1$$

$$t_1 = 0.1\,s$$

$$K_1 = f(t_0, q_0)$$

$$= \frac{10^3 \sin\sqrt{\pi t_0}}{R_1} - \frac{q_0}{R_1 C_1}$$

$$= \frac{10^3 \sin\sqrt{\pi (0)}}{(1000)} - \frac{4}{(1000)(0.002)}$$

$$K_1 = -2\ A$$

$$K_2 = f\left(t_0 + \tfrac{1}{2}h,\ q_0 + \tfrac{1}{2}K_1 h\right) ; \quad t_0 + \tfrac{1}{2}h = 0 + \tfrac{1}{2}(0.1) \quad ; \quad q_0 + \tfrac{1}{2}K_1 h = 4 + \tfrac{1}{2}(-2)(0.1)$$

$$= 0.05\,s \qquad\qquad = 3.9\ C$$

$$= \frac{10^3 \sin\sqrt{\pi (0.05)}}{1000} - \frac{3.9}{1000(0.002)}$$

$$K_2 = -1.563962\ A$$

$$K_3 = f\left(t_0 + \tfrac{1}{2}h,\ q_0 + \tfrac{1}{2}K_2 h\right) ; \quad t_0 + \tfrac{1}{2}h = 0.05\,s \quad ; \quad q_0 + \tfrac{1}{2}K_2 h = 4 + \tfrac{1}{2}(-1.663962)(0.1)$$

$$K_3 = \frac{10^3 \sin\sqrt{\pi (0.05)}}{1000} - \frac{3.921802}{1000(0.002)} = -1.574863\ A \qquad = 3.921802\ C$$

$$K_4 = f(t_0 + h,\ q_0 + K_3 h) ; \quad t_0 + h = 0.1\,s \quad ; \quad q_0 + K_3 h = 4 + (-1.574863)\,0.1$$

$$= 3.842514\ C$$

$$K_4 = \frac{10^3 \sin\sqrt{\pi (0.1)}}{1000} - \frac{3.842514}{1000(0.002)}$$

$$K_4 = -1.389648\ A$$

Name : Kai Chuen Tan

Course : CSE 276 C

HW # : Homework 3

Date : 10/23/2021

Professor : Dr. Christensen

Problem 1 (Part 3)

$$q_1 = q_0 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)h$$

$$= 4 + \frac{1}{6}\left[-2 + 2(-1.563962) + 2(-1.574863) + (-1.389648)\right]0.1$$

$\boxed{q_1 = 3.838878 \text{ C} \quad ; \text{ hence, using } 4^{th} \text{ Order Runge-Kutta's Method, } q(t) \text{ at } t = 0.1s \text{ is } 3.838878 \text{ C.}}$

Contents


```
% Name          : Kai Chuen Tan
% Title         : Homework 3
% Course        : CSE 276C: Mathematics for Robotics
% Professor     : Dr. Henrik I. Christensen
% Date          : 25 th October 2021

clear all;
clc;

fprintf('Name          : Kai Chuen Tan\n')
fprintf('Title         : Homework 3\n')
fprintf('Course        : CSE 276C: Mathematics for Robotics\n')
fprintf('Professor     : Dr. Henrik I. Christensen\n')
fprintf('Date          : 25 th October 2021\n\n')
fprintf('--------------------------------------------------------\n\n')
```

```
Name          : Kai Chuen Tan
Title         : Homework 3
Course        : CSE 276C: Mathematics for Robotics
Professor     : Dr. Henrik I. Christensen
Date          : 25 th October 2021

--------------------------------------------------------
```

Problem 2 - Solving Expected Value of a PDF with Numerical Integration

```
fprintf('Problem 2 - Solving Expected Value of a PDF with Numerical Integration\n')
fprintf('--------------------------------------------------------------------\n\n')

% Given a probability density function (PDF), f(x)
% f_x = @(x) 1 / exp(1) * exp(x) .* (x + 1);

% x * f(x) function
xf_x = @(x) x / exp(1) .* exp(x) .* (x + 1);

% Given the range from a to b
a = 0; b = 1;

% Given the size of the interval
h = 0.1;

% Calculate the number of intervals, n
% h = (b - a) / n
n = (b - a) / h;

fprintf('Problem 2a - Rectangular Method\n')
fprintf('--------------------------------\n\n')
```

```
EX_rectangular = Rectangular_Method(xf_x, a, b, n, h);

fprintf("The expected value, E(X) using Rectangular Method is E(X) = %.6f.\n\n", EX_rectangular)

fprintf('\nProblem 2b - Midpoint Method\n')
fprintf('-------------------------------\n\n')

EX_midpoint = Midpoint_Method(xf_x, a, b, n, h);

fprintf("The expected value, E(X) using Midpoint Method is E(X) = %.6f.\n\n", EX_midpoint)

fprintf('\nProblem 2c - Trapezoidal Method\n')
fprintf('----------------------------------\n\n')

EX_trapezoidal = trapezoidal_method(xf_x, a, b, n, h);

fprintf("The expected value, E(X) using Trapezoidal Method is E(X) = %.6f.\n\n", EX_trapezoidal)
```

```
Problem 2 - Solving Expected Value of a PDF with Numerical Integration
--------------------------------------------------------------------------

Problem 2a - Rectangular Method
--------------------------------

The expected value, E(X) using Rectangular Method is E(X) = 0.535979.


Problem 2b - Midpoint Method
-------------------------------

The expected value, E(X) using Midpoint Method is E(X) = 0.630192.


Problem 2c - Trapezoidal Method
----------------------------------

The expected value, E(X) using Trapezoidal Method is E(X) = 0.635979.
```

Published with MATLAB® R2021a

```
function [EX] = Rectangular_Method(f_x, a, b, n, h)
% Rectangular Method function that calculates the expected value of a
% probability density function (PDF)
% f_x - x * PDF function
% a   - initial x
% b   - final x
% h   - step size
% n   - number of intervals

% Store all the x values in a vector form.
x = a:h:b;

% Initialize Expected Value.
EX = 0;

% Calculate the Expected Value with rectangular method.
for iter = 1:n

    % Accumulate the EX value
    EX = EX + f_x(x(iter));

end

% Calculate the final EX value
EX = h * EX;

end
```

```
function [EX] = Midpoint_Method(f_x, a, b, n, h)
% Midpoint Method function that calculates the expected value of a
% probability density function (PDF)
% f_x - x * PDF function
% a   - initial x
% b   - final x
% h   - step size
% n   - number of intervals

% Store all the x values in a vector form.
x = a:h:b;

% Initialize Expected Value.
EX = 0;

% Calculate the Expected Value with midpoint method.
for iter = 1:n

    % Calculate the value needed to pass into the function
    c = (x(iter) + x(iter + 1)) / 2;

    % Accumulate the EX value
    EX = EX + f_x(c);

end

% Calculate the final EX value
EX = h * EX;

end
```

```
function [EX] = trapezoidal_method(f_x, a, b, n, h)
% Trapezoidal Method function that calculates the expected value of a
% probability density function (PDF)
% f_x - x * PDF function
% a   - initial x
% b   - final x
% h   - step size
% n   - number of intervals

% Store all the x values in a vector form.
x = a:h:b;

% PDF Function
F_x = f_x(x);

% Calculate the Expected Value with trapezoidal method.
EX = 1 / 2 * h * (F_x(1) + F_x(n + 1)) + h * sum(F_x(2:n));

end
```

Name : Kai Chuen Tan

Course: CSE 276C

HW #: Homework 3

Date : 10/25/2021

Professor: Dr. Christensen

Problem 3

$$\frac{dy}{dx} = \frac{1}{x^2(1-y)} \quad ; \text{ interval } [0, 1] \; ; \; y(1) = -1$$

Step 1:

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

$\cancel{y' = \frac{1}{x^2(1-y)}}$

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

$\cancel{y - \frac{y^2}{2}}$

$$y - \frac{y^2}{2} = -\frac{1}{x} + C \quad ; y = -1 \; ; x = 1$$

$$(-1) - \frac{(-1)^2}{2} = -\frac{1}{1} + C$$

$$C = -\frac{1}{2}$$

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

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

$\cancel{x' = x^2(1-y)}$

$\cancel{x' = x^2 - x^2 y}$

$\cancel{x' = x^2 - x^2 y}$

$\cancel{x^2 y = -x' + x^2}$

$\cancel{y = -\frac{x'}{x^2} + 1}$

$\cancel{y = 1 - \frac{x'}{x^2}}$

Step 2:

$$y - \frac{y^2}{2} = -\frac{1}{x} - \frac{1}{2}$$

$$2y - y^2 = -\frac{2}{x} - 1$$

$$y^2 - 2y = 1 + \frac{2}{x}$$

$$y(y-2) = 1 + \frac{2}{x} \qquad \cancel{x = 0}$$

$\cancel{y(y-2) = \infty}$

$\cancel{y^2 - 2y - \infty = 0}$

$\cancel{y = \frac{-(-2) \pm \sqrt{(-2)^2 - 4(1)(-\infty)}}{2(1)}}$

$\cancel{y = \frac{2 \pm \sqrt{\infty}}{2}}$

$\cancel{\boxed{y = \infty \; , \; y = -\infty}}$

Step 3:

$$y(y-2) = 1 + \frac{2}{x}$$

$$y^2 - 2y - \left(1 + \frac{2}{x}\right) = 0$$

$$y = \frac{-(-2) \pm \sqrt{(-2)^2 - 4(1)\left(-1 - \frac{2}{x}\right)}}{2}$$

$$y = \frac{2 \pm \sqrt{4 + 4\left(1 + \frac{2}{x}\right)}}{2}$$

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

$$y = 1 \pm \sqrt{2\left(1 + \frac{1}{x}\right)}$$

Step 4:

$$\boxed{\text{if } x = 0, \; y = \infty \; , \; y = -\infty}$$

Contents


```
% Name          : Kai Chuen Tan
% Title         : Homework 3
% Course        : CSE 276C: Mathematics for Robotics
% Professor     : Dr. Henrik I. Christensen
% Date          : 25 th October 2021

clear all;
clc;

fprintf('Name          : Kai Chuen Tan\n')
fprintf('Title         : Homework 3\n')
fprintf('Course        : CSE 276C: Mathematics for Robotics\n')
fprintf('Professor     : Dr. Henrik I. Christensen\n')
fprintf('Date          : 25 th October 2021\n\n')
fprintf('--------------------------------------------------------\n\n')
```

```
Name          : Kai Chuen Tan
Title         : Homework 3
Course        : CSE 276C: Mathematics for Robotics
Professor     : Dr. Henrik I. Christensen
Date          : 25 th October 2021

--------------------------------------------------------
```

--

Problem 3 - Solving First Order Odinary Differential Equation

```
fprintf('Problem 3 - Solving First Order Odinary Differential Equation\n')
fprintf('--------------------------------------------------------------\n\n')

% First Order Differential Equation
dydx = @(x, y) 1 / (x^2 .* (1 - y));

% Positive Analytical Solution, y_positive = 1 + sqrt(2 * (1 + 1/x))
y_positive =@(x) 1 + sqrt(2 * (1 + 1/x));

% Positive Analytical Solution, y_positive = 1 - sqrt(2 * (1 + 1/x))
y_negative =@(x) 1 - sqrt(2 * (1 + 1/x));

% Interval [a, b]
a = 1;
b = 0;

% Step Size, h
h = 0.05;

% Number of intervals, n
n = abs(b - a)/h;

% Initial of of y.
y_ini = -1;

fprintf('Problem 3a - Analytical Method\n')
fprintf('--------------------------------\n\n')

y_exact = y_negative(0);
```

```
fprintf("Analytical Method's Exact Solution for Problem 3b,c, and d to compare, y(0) = %.6f.\n\n", y_exact)

fprintf('Problem 3b - Euler''s ODE Method\n')
fprintf('--------------------------------\n\n')

[x_Euler, y_Euler] = ODE_Euler(dydx, a, b, n, h, y_ini);

if y_Euler < 0

    %y_exact = y_negative(x_Euler(end));
    y_exact = y_negative(0);

elseif y_Euler > 0

    %y_exact = y_positive(x_Euler(end));
    y_exact = y_positive(0);

end

% Calculate the percentage error and accuracy
error_Euler = abs((y_exact - y_Euler(end))/y_exact) * 100;
%accuracy_Euler = abs(100 - error_Euler);

fprintf("Using ODE Euler's Method, y(0) = %.6f.\n\n", y_Euler(end))
fprintf("Percentage error of the ODE Euler's Method Solution =  %.6f %%\n\n", error_Euler)
%fprintf("Accuracy of the ODE Euler's Method Solution =  %.6f %%\n\n", accuracy_Euler)
fprintf("Since the exact analytical solution is negative infinity and the relative error is infinity\n")
fprintf("divided by infinity, which is an intermediate form,\n")
fprintf("the accuracy of the ODE Euler's Method solution y_Euler is undefined at x = 0.\n\n\n")

fprintf('Problem 3c - Runge-Kutta 4th Order''s ODE Method\n')
fprintf('-------------------------------------------------\n\n')

[x_RK4, y_RK4] = ODE_Runge_Kutta_4(dydx, a, b, n, h, y_ini);

if y_RK4 < 0

    %y_exact = y_negative(x_RK4(end));
    y_exact = y_negative(0);

elseif y_RK4 > 0

    %y_exact = y_positive(x_RK4(end));
    y_exact = y_positive(0);

end

% Calculate the percentage error and accuracy.
error_RK4 = abs((y_exact - y_RK4(end))/y_exact) * 100;
%accuracy_RK4 = abs(100 - error_RK4);

fprintf("Using ODE Runge-Kutta 4th Order's Method, y(0) = %.2f.\n\n", y_RK4(end))
fprintf("Percentage error of the ODE Runge-Kutta 4th Order's Method Solution =  %.6f %%.\n\n", error_RK4)
%fprintf("Accuracy of the ODE Runge-Kutta 4th Order's Method Solution =  %.6f %%.\n\n", accuracy_RK4)
fprintf("Since the exact analytical solution is negative infinity and the relative error is infinity\n")
fprintf("divided by infinity, which is an intermediate form,\n")
fprintf("the accuracy of the ODE Runge-Kutta 4th Order's Method solution y_RK4 is undefined at x = 0.\n\n\n")

fprintf('Problem 3d - Richardson Extrapolation''s ODE Method\n')
fprintf('-------------------------------------------------\n\n')

[x_Richard, y_Richard] = Richardson_Extrpolation_Method(dydx, a, b, n, h, y_ini);

if y_Richard < 0

    %y_exact = y_negative(x_Richard(end));
    y_exact = y_negative(0);
```

```
elseif y_RK4 > 0

    %y_exact = y_positive(x_Richard(end));
    y_exact = y_positive(0);

end

% Calculate the percentage error and accuracy.
error_Richard = abs((y_exact - y_Richard(end))/y_exact) * 100;
%accuracy_Richard = abs(100 - error_Richard);

fprintf("Using ODE Richardson Extrapolation Method, y(0) = %.2f.\n\n", y_Richard(end))
fprintf("Percentage error of the ODE Richardson Extrapolation Method Solution =  %.6f %%.\n\n", error_Richard)
%fprintf("Accuracy of the ODE Richardson Extrapolation Method Solution =  %.6f. %%\n\n", accuracy_Richard)
fprintf("Since the exact analytical solution is negative infinity and the relative error is infinity\n")
fprintf("divided by infinity, which is an intermediate form,\n")
fprintf("the accuracy of the ODE Richardson Extrapolation Method solution y_Richard is undefined at x = 0.\n\n\n")
```

```
Problem 3 - Solving First Order Odinary Differential Equation
--------------------------------------------------------------

Problem 3a - Analytical Method
--------------------------------

Analytical Method's Exact Solution for Problem 3b,c, and d to compare, y(0) = -Inf.

Problem 3b - Euler's ODE Method
--------------------------------

Using ODE Euler's Method, y(0) = -8.124934.

Percentage error of the ODE Euler's Method Solution =  NaN %

Since the exact analytical solution is negative infinity and the relative error is infinity
divided by infinity, which is an intermediate form,
the accuracy of the ODE Euler's Method solution y_Euler is undefined at x = 0.


Problem 3c - Runge-Kutta 4th Order's ODE Method
-------------------------------------------------

Using ODE Runge-Kutta 4th Order's Method, y(0) = -607173044131906172468854784O.00.

Percentage error of the ODE Runge-Kutta 4th Order's Method Solution =  NaN %.

Since the exact analytical solution is negative infinity and the relative error is infinity
divided by infinity, which is an intermediate form,
the accuracy of the ODE Runge-Kutta 4th Order's Method solution y_RK4 is undefined at x = 0.


Problem 3d - Richardson Extrapolation's ODE Method
-------------------------------------------------

Using ODE Richardson Extrapolation Method, y(0) = -21730894784696403949413990 4.00.

Percentage error of the ODE Richardson Extrapolation Method Solution =  NaN %.

Since the exact analytical solution is negative infinity and the relative error is infinity
divided by infinity, which is an intermediate form,
the accuracy of the ODE Richardson Extrapolation Method solution y_Richard is undefined at x = 0.
```

```
function [x, y_Euler] = ODE_Euler(dydx, a, b, n, h, y_ini)
% ODE_Euler solves 1st order initial value ODE with Euler's Method
% dydx - First Order Differential Equation
% a    - starting point of a range
% b    - ending point of a range
% h    - step size
% n    - number of intervals

% Initialize y vectors;
y_Euler = zeros(n, 1);

% Store all the x values in a vector form.
if a > b
    h = -h;
end
%x = a : h : b;

% Initial value of x
x(1) = a;

% Initial value of y
y_Euler(1) = y_ini;

% Apply Euler's Method
for i = 1 : n

    x(i+1) = x(i) + h;

    y_Euler(i + 1) = y_Euler(i) + dydx(x(i), y_Euler(i)) * h;

end

end
```

```
function [x, y_RK4] = ODE_Runge_Kutta_4(dydx, a, b, n, h, y_ini)
% ODE_Runge_Kutta_4 solves 1st order initial value ODE with Runge-Kutta
% Fourth Order's Method
% dydx - First Order Differential Equation
% a    - starting point of a range
% b    - ending point of a range
% h    - step size
% n    - number of intervals

% Initialize y vectors;
%y_RK4 = zeros(n, 1);

% Store all the x values in a vector form.
if a > b
    h = -h;
end
%x = a : h : b;

% Initial value of x
x(1) = a;

% Initial value of y
y_RK4(1) = y_ini;

% Apply Runge-Kutta 4th Order Method
for i = 1 : n

    x(i+1) = x(i) + h;

    K_1 = dydx(x(i), y_RK4(i));

    new_x = x(i) + h / 2;

    y_K1 = y_RK4(i) + K_1 / 2 * h;

    K_2 = dydx(new_x, y_K1);

    y_K2 = y_RK4(i) + K_2 / 2 * h;

    K_3 = dydx(new_x, y_K2);

    y_K3 = y_RK4(i) + K_3 * h;

    K_4 = dydx(x(i + 1), y_K3);

    y_RK4(i + 1) = y_RK4(i) + h / 6 * (K_1 + 2 * K_2 + 2 * K_3 + K_4);

end

end
```

```
function [x, y_Richard] = Richardson_Extrpolation_Method(dydx, a, b, n, h, y_ini)
% Richardson Extrapolation Method Function solves 1st order initial value ODE
% with Richardson Extrapolation Method's
% dydx - First Order Differential Equation
% a    - starting point of a range
% b    - ending point of a range
% h    - step size
% n    - number of intervals

% Initialize x and y vectors;
y_Richard = zeros(n, 1);

% Initialize z vector;
z = zeros(n, 1);

% Store all the x values in a vector form.
if a > b
    H = -h;
else
    H = h;
end

%x = a : H : b;

% Size of Sub-step
sub_H = H / n;

% Initial value of x
x(1) = a;

% Initial value of y
y_Richard(1) = y_ini;

% Initial value of z_0 and z_1
z(1) = y_ini;
z(2) = z(1) + sub_H * dydx(x(1), z(1));

% Apply Richardson Extrapolation Method
for counter = 1 : n

    x(counter+1) = x(counter) + H;

    for i = 2 : n

        z(i + 1) = z(i - 1) + 2 * sub_H * dydx(x(counter) + (i-1)*sub_H, z(i));

    end

    y_Richard(counter + 1) = 1 / 2 * (z(end) + z(end-1) + sub_H * dydx(x(counter) + H,↙
z(end)));

end

end
```

CSE276C_HW3_Problem_4

November 3, 2021

0.0.1 Problem 4

We have multiple robots that can generate point clouds such as those coming from a RealSense camera. In many cases we want to use the robots to detect objects in its enviroment. We provide three data files:

(a) Empty2.asc which containts a data for an empty table
(b) TableWithObjects2.asc contains point cloud for a cluttered table
(c) hallway1b.asc contains data from a hallway

Each file has the point cloud file in a format with each line contains xi yi zi. You can use np.loadtxt to load a pointcloud into a numpy array.

0.0.2 Problem 4: Part 1

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.

Solution:

The general equation of a plane can be written as follows:

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

$ax + by + cz = ax_0 + by_0 + cz_0; d = ax_0 + by_0 + cz_0$

$ax + by + cz = d$

where,

n is the normal vector of the plane, $< a, b, c >$.

The plane equation can also be re-written in this form:

$ax + by - d = -cz$

$-\frac{a}{c}x - \frac{b}{c}y + \frac{d}{c} = z$

$Ax + By + C = z$

The re-written normal vector of the plane, n became $< A, B, -1 >$.

The distance between a point and the plane, d can be calculated as shown below:

$v = \begin{bmatrix} x_i - \bar{x} & y_i - \bar{y} & z_i - \bar{z} \end{bmatrix}$

$\hat{n} = \frac{1}{||n||} \begin{bmatrix} A \\ B \\ -1 \end{bmatrix}$

$d = |v \cdot \hat{n}|$

where,

$\hat{n}$ is the normalized normal vector of the plane

$\bar{x}$ is the average of the x-coordinate

$\bar{y}$ is the average of the y-coordinate

$\bar{z}$ is the average of the z-coordinate

$||n||$ is the norm of the normalized normal vector of the plane

The least square plane fitting is a common method to estimate plane parameters with a given point cloud, even when the data are a little noisy. However, if the point cloud consists of many outliers, the least square fitting method might not be sufficiently robust to estimate plane parameters, and outliers will lower the quality of the plane fitting process. Hence, to improve the quality and robustness of the plane fitting process, the random sample consensus (RANSAC) plane fitting method is used to estimate the dominant plane parameters of the empty table, and the table with objects even though there are outliers in the point cloud.

Since the RANSAC method is considered as a heuristics algorithm, the RANSAC algorithm will iterate several thousands of times to perform least square plane fitting with a set of randomly selected 3 points from the point cloud data. If the distance between a point and the fitted plane, d, is less than or equal to the acceptable distance between a point and the fitted plane, $d_{threshold}$, then the point is an inliers; otherwise, the point is an outlier. Besides that, the RANSAC algorithm will also count the number inliers; if the current number of inliers is larger than the previous iteration, the algorithm will store the best set of inliers and store the outliers separately. Lastly, the RANSAC algorithm will output the normalized normal vector of the fitted plane, normalized coefficient C, the average of the points, and the distance between a point and the plane for every points, inliers, and outliers. Hence, the RANSAC algorithm successfully filter out the outliers from the point cloud.

[63]:
```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d
from mpl_toolkits.mplot3d import Axes3D
import random
%matplotlib notebook

### Problem 4 Part 1: Use Random Sample Consensus (RANSAC) Algorithm for Plane␣
↪Detection in
###                    the Empty and Clutter Table Point Cloud Data


# Load Empty Table and Table with Objects files
```

```
empty_Table_pts = np.loadtxt("Empty2.asc")
table_with_Obj_pts = np.loadtxt("TableWithObjects2.asc")

# Store Empty Table x, y, z coordinates
#empty_Table_x = empty_Table_pts[:,0]
#empty_Table_y = empty_Table_pts[:,1]
#empty_Table_z = empty_Table_pts[:,2]

# Store Table with Objects x, y, z coordinates
#table_with_Obj_x = table_with_Obj_pts[:,0]
#table_with_Obj_y = table_with_Obj_pts[:,1]
#table_with_Obj_z = table_with_Obj_pts[:,2]


### User-defined Plane Fitting with Least Square Method Function
## Input  : 3D-points (m x 3)
## Outputs: normal_vec_normalized - normalized normal vector of the fitted␣
↪plane (1 x 3)
##         coeff_C_normalized    - Coefficient C from the equation Ax + By + C␣
↪= z
##         mean_point            - Average of the points (1 x 3)
##         dist_pt2pl            - Distance between a point and the plane (m x␣
↪1)
def LSQ_plane_fitting(points_3D):

    """
    Pseudocode:
        1.) Construct Matrix A, [x_i, y_i, 1] (m x 3)
        2.) Construct Z Vector, [z_i] (m x 1)
        3.) Call numpy built-in linear regression to find the coefficients A,␣
↪B, and C
            from the plane equation, Ax + By + C = z
        4.) Nomalize the plane equation
        5.) Determine distance between a point and the fitted plane for every␣
↪points, dist_pt2pl
        6.) Output normalized normal vector, normalized coefficient C, average␣
↪point, dist_pt2pl
    """

    # Determine total number of points
    tot_num_pts = points_3D.shape[0]

    # Determine the number of dimensions
    num_dims = points_3D.shape[1]

    # Ax = b => An = Z
```

```
    # Initialize matrix A, [x_i, y_i, 1]
    A = np.ones((tot_num_pts, num_dims))
    A[:, 0] = points_3D[:, 0] # Pass x-coordinates to matrix A
    A[:, 1] = points_3D[:, 1] # Pass y-coordinates to matrix A

    # Initialize Z column vector [z_i]
    Z = np.zeros((tot_num_pts, 1))
    Z[:, 0] = points_3D[:, 2]

    # Perform Least Square plane fitting
    coeffs, _, _, _ = np.linalg.lstsq(A, Z, rcond = None)

    # Normal Vector of the Plane [A, B, -1]
    normal_vec = (coeffs[0][0], coeffs[1][0], -1)

    # Coefficient C
    coeff_C = coeffs[2][0]

    # Normalized the plane equation
    normal_norm = np.linalg.norm(normal_vec) # Norm of the normal vector
    normal_vec_normalized =  normal_vec / normal_norm

    # Normalized the Coefficient C
    coeff_C_normalized = coeff_C / normal_norm

    # Determine the mean point from the plane
    mean_point = np.mean(points_3D, axis=0)

    # Calculate the distance between a point and the fitted plane for every␣
 ↪points
    dist_pt2pl = abs(np.dot(points_3D-mean_point, normal_vec_normalized))

    return normal_vec_normalized, coeff_C_normalized, mean_point, dist_pt2pl

### LSQ_plane_fitting() function print test
#normal_vec_normalized, coeff_C_normalized, mean_point, dist =␣
 ↪LSQ_plane_fitting(empty_Table_pts)
#print(normal_vec_normalized, coeff_C_normalized, mean_point, dist)


### User-defined Random Sample Consensus (RANSAC) Function
## Input  : 3D-points (m x 3)
##          Maximum Iterations
##          Maximum Threshold for the Distance between a point and the plane
## Outputs: RanSaC_normal_vec_normalized - RanSaC normalized normal vector of␣
 ↪the fitted plane (1 x 3)
```

```
##          RanSaC_coeff_C_normalized    - RanSaC Coefficient C from the␣
↪equation Ax + By + C = z
##          RanSaC_mean_point            - RanSaC Average of the points (1 x 3)
##          RanSaC_dist_pt2pl            - RanSaC Distance between a point and␣
↪the plane (m x 1)
##          inlier_pts                   - Points that are inliers
##          outlier_pts                  - Points that are outliers
def RanSaC_algorithm(points_3D, max_iterations, dist_pt2pl_threshold):

    """
    Pseudocode:
        1.) Initialize the maximum number of inliers and best inliers set
        2.) Start the iteration till it reaches maximum iterations with for loop
            a.) Pick 3 random points
            b.) Perform least square plane fitting with those random n points
            c.) Check the distance threshold for every points
            d.) Determine total number of points that are within the distance␣
↪threshold
            e.) Store the total number inliers and the best set of inliers if␣
↪it is more than the previous one
        3.) Perform another least square plane fitting with best set of inliers
        4.) Outputs the Output normalized normal vector, normalized coefficient␣
↪C, average point, dist_pt2pl from the
            best set of inlier
    """

    # Initialize the maximum number of inliers
    max_num_Inliers = None

    # Initialize the best set of inliers
    best_set_Inliers = None

    # Initialize the set of outlier
    set_Outliers = None

    # Start the iteration to perform RanSaC to discard outliers
    for iteration in range(0, max_iterations):

        # Pick 3 Random Points from the set of the points_3D
        rand_points = points_3D[random.sample(range(0,points_3D.shape[0]), 3),:]

        # Perform least square plane fitting with those randomly selected␣
↪points.
        normal_vec_normalized, coeff_C_normalized, mean_point, _ =␣
↪LSQ_plane_fitting(rand_points)
```

```
        # Determine the distance between a point and the plane.
        dist_pt2pl = abs(np.dot(points_3D - mean_point, normal_vec_normalized))

        # Store the inlier index that are equal or below the distance threshold
        inliers_index = np.where(dist_pt2pl <= dist_pt2pl_threshold)

        # Store the outlier index
        outliers_index = np.where(dist_pt2pl > dist_pt2pl_threshold)

        # Count the Total number of inliers
        num_Inliers = inliers_index[0].shape[0]

        # If the score is better than the previous iteration
        if (max_num_Inliers is None) or (num_Inliers > max_num_Inliers):

            # Store the index of the inliers
            best_set_Inliers = inliers_index

            # Store the index of the outliers
            set_Outliers = outliers_index

            # Store the new maximum number of inliers
            max_num_Inliers = num_Inliers

    # Store outliers
    outlier_pts = points_3D[set_Outliers[0], :]

    # Store only points that are inliers
    inlier_pts = points_3D[best_set_Inliers[0], :]

    # Perform another least square plane fitting with those inlier points.
    RanSaC_normal_vec_normalized, RanSaC_coeff_C_normalized, RanSaC_mean_point,␣
↪RanSaC_dist_pt2pl = LSQ_plane_fitting(inlier_pts)

    return RanSaC_normal_vec_normalized, RanSaC_coeff_C_normalized,␣
↪RanSaC_mean_point, RanSaC_dist_pt2pl, inlier_pts, outlier_pts


### User-defined display of the plane plot function
## Input  : Inliers
##          Outliers
##          3D-points (m x 3)
##          Normal Vector
##          Average point
##          Graph title in string
## Outputs: None
```

```
def display_plane_plot(inlier_pts, outlier_pts, points_3D, normal_vec,␣
↪mean_point, graph_title):

    # Initialize figure object and set figure size
    CSE_276C_fig = plt.figure(figsize=(7,7))
    # Initialize axes object
    CSE_276C_ax1 = Axes3D(CSE_276C_fig)

    # Plot scatter points
    # Inliers are green
    CSE_276C_ax1.scatter(inlier_pts[:,0], inlier_pts[:, 1], inlier_pts[:, 2], c␣
↪= 'green', label = "Inliers")
    # Outliers are red
    CSE_276C_ax1.scatter(outlier_pts[:,0], outlier_pts[:, 1], outlier_pts[:,␣
↪2], c = 'red', label = "Outliers")
    plt.legend(loc="upper right") # Legend location

    # Plot the fitted plane
    X_coord = np.linspace(min(points_3D[:,0]),max(points_3D[:,0]),3) #␣
↪Determine the x-axis limit
    Y_coord = np.linspace(min(points_3D[:,1]),max(points_3D[:,1]),3) #␣
↪Determine the y-axis limit
    x_coord, y_coord = np.meshgrid(X_coord, Y_coord)
    z_coord = -(normal_vec[0] / normal_vec[2]) * x_coord - (normal_vec[1]/
↪normal_vec[2]) * y_coord + (np.dot(normal_vec, mean_point)/normal_vec[2])
    CSE_276C_ax1.plot_wireframe(x_coord,y_coord,z_coord,color='k')

    # Label Axis
    CSE_276C_ax1.set_xlabel("X", fontsize = 15)
    CSE_276C_ax1.set_ylabel("Y", fontsize = 15)
    CSE_276C_ax1.set_zlabel("Z", fontsize = 15)

    # Plot title
    plt.title(graph_title, fontsize = 18, fontweight = "bold")

    # Show Plot
    plt.show()

    # Change View of the Plot
    #CSE_276C_ax1.view_init(0, 0)

    """# Initialize axes object
    CSE_276C_ax2 = CSE_276C_fig.add_subplot(222, projection='3d')

    # Plot scatter points
    # Inliers are green
```

```
    CSE_276C_ax2.scatter(inlier_pts[:,0], inlier_pts[:, 1], inlier_pts[:, 2], c␣
↪= 'green')
    # Outliers are red
    CSE_276C_ax2.scatter(outlier_pts[:,0], outlier_pts[:, 1], outlier_pts[:,␣
↪2], c = 'red')

    # Plot the fitted plane
    x_coord, y_coord = np.meshgrid(X_coord, Y_coord)
    z_coord = -(normal_vec[0] / normal_vec[2]) * x_coord - (normal_vec[1]/
↪normal_vec[2]) * y_coord + (np.dot(normal_vec, mean_point)/normal_vec[2])
    CSE_276C_ax2.plot_wireframe(x_coord,y_coord,z_coord,color='k')

    # Label Axis
    CSE_276C_ax2.set_xlabel("X", fontsize = 8)
    CSE_276C_ax2.set_ylabel("Y", fontsize = 8)
    CSE_276C_ax2.set_zlabel("Z", fontsize = 8)

    # Plot title
    plt.title(graph_title, fontsize = 10, fontweight = "bold")
    CSE_276C_ax2.view_init(0, 0)"""


# Plane Fitting for the Empty Table with RanSaC
RanSaC_normal_vec_normalized, RanSaC_coeff_C_normalized, RanSaC_mean_point,␣
↪RanSaC_dist_pt2pl, inlier_pts, outlier_pts =␣
↪RanSaC_algorithm(empty_Table_pts, 2000, 0.05)

# Plane Fitting for the Table with Objects with RanSaC
RanSaC_normal_vec_normalized2, RanSaC_coeff_C_normalized2, RanSaC_mean_point2,␣
↪RanSaC_dist_pt2pl2, inlier_pts2, outlier_pts2 =␣
↪RanSaC_algorithm(table_with_Obj_pts, 5000, 0.05)


# Display Plane Fitting Plot for the Empty Table
display_plane_plot(inlier_pts, outlier_pts, empty_Table_pts,␣
↪RanSaC_normal_vec_normalized, RanSaC_mean_point, "Empty Table")
# Show the plane equation for the Empty Table
print("The equation of the empty table fitted plane:")
print("%f x + %f y + %f z + %f = 0" % (RanSaC_normal_vec_normalized[0],␣
↪RanSaC_normal_vec_normalized[1], RanSaC_normal_vec_normalized[2],␣
↪RanSaC_coeff_C_normalized))
print("The average distance between an inlier and the fitted plane = %f" % (np.
↪mean(RanSaC_dist_pt2pl)))

# Display Plane Fitting Plot for Table with Objects
```

```
display_plane_plot(inlier_pts2, outlier_pts2, table_with_Obj_pts,␣
↪RanSaC_normal_vec_normalized2, RanSaC_mean_point2, "Table with Objects")
# Show the plane equation for Table with Objects
print("The equation of the table with objects fitted plane:")
print("%f x + %f y + %f z + %f = 0" % (RanSaC_normal_vec_normalized2[0],␣
↪RanSaC_normal_vec_normalized2[1], RanSaC_normal_vec_normalized2[2],␣
↪RanSaC_coeff_C_normalized2))
print("The average distance between an inlier and the fitted plane = %f" % (np.
↪mean(RanSaC_dist_pt2pl2)))


### Plot samples by Professor Christensen and Dr. Wong
#fig = plt.figure()
#ax = fig.add_subplot(111, projection='3d')
#ax.plot_trisurf(empty_Table[:,0], empty_Table[:,1], empty_Table[:,2],␣
↪color='white', edgecolors='grey', alpha=0.5)
#ax.scatter(empty_Table[:,0], empty_Table[:,1], empty_Table[:,2], c='red')
#plt.show()
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


The equation of the empty table fitted plane:
-0.011647 x + -0.906377 y + -0.422310 z + 0.593806 = 0
The average distance between an inlier and the fitted plane = 0.010851

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


The equation of the table with objects fitted plane:
0.027658 x + -0.870000 y + -0.492276 z + 0.670482 = 0
The average distance between an inlier and the fitted plane = 0.018778
```

After the empty table point cloud file, "Empty2.asc" was loaded, the empty table point cloud was input to RANSAC algorithm with a maximum iteration of 2000 and $d_{dest} = 0.05$. The "Empty Table" figure above shows that the equation of the empty table fitted plane is $-0.011647x - 0.906377y - 0.422310z + 0.593806 = 0$. The average distance between an inlier of the empty table and the plane is 0.010851, which is below the set d_{dest}. Then, the table with objects point cloud, "TableWithObjects2.asc" was also loaded, the table with objects point cloud was input to the RANSAC algorithm with a maximum iteration of 5000 and $d_{dest} = 0.05$. The RANSAC algorithm ran 3000 more iteration with table with objects point cloud because there are more outlier to filter out to determine the best fitted plane. The "Table with Objects" figure above shows that the equation of the table with objects fitted plane is $0.027658x - 0.870000y - 0.492276z + 0.670482 = 0$. The average distance between an inlier of the table with objects and the plane is 0.018778, which is below the set d_{dest}.

[15]:
```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d
from mpl_toolkits.mplot3d import Axes3D
import random
%matplotlib notebook

# Load Hallway files
hallway = np.loadtxt("hallway1b.asc")

### Points Cloud Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_trisurf(hallway[:,0], hallway[:,1], hallway[:,2], color='white',␣
↪edgecolors='grey', alpha=0.5)
ax.scatter(hallway[:,0], hallway[:,1], hallway[:,2], c='red')
# Label Axis
ax.set_xlabel("X", fontsize = 15)
ax.set_ylabel("Y", fontsize = 15)
ax.set_zlabel("Z", fontsize = 15)

# Plot title
plt.title("Hallway Point Cloud", fontsize = 18, fontweight = "bold")

plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

0.0.3 Problem 4: Part 2

Describe and show how the method can be generalized to extract all the dominant planes in a relatively empty hallway.

Solution:

In order to detect multiple planes in the hallway, a user-defined function, "dominant_planes_Hunter()" was created. The dominant_planes_Hunter() user-defined function will run the RANSAC algorithm with the hallway 3D point cloud and store the first plane parameters including the normal vector, coefficient C, mean point, etc. Then, the outliers will be stored, and re-run the RANSAC algorithm with the outliers to exclude the previous inliers. The process will be repeated another 3 times to find all the remaining dominant plane in the hallway. The RANSAC algorithm maximum iterations was set to be 10000, and the distance threshold, d_{thres}, to be 0.2. The planes in the hallway was plotted as shown in the figure below:

[55]:

```python
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d
from mpl_toolkits.mplot3d import Axes3D
import random
%matplotlib notebook

# Load Hallway files
hallway_pts = np.loadtxt("hallway1b.asc")

### User-defined dominant planes finder and plotter function
## Input  : 3D-points (m x 3)
##          Maximum iterations for each plane (array form)
##          Maximum distance threshold for each plane (array form)
##          Number of Expected Planes
## Outputs: All the normal vectors in a list form
##          All coefficient Cs in a list form
##          All the mean points in a list form
##          All the inlier points in a list form
##          All the outlier points in a list form
def dominant_planes_Hunter(points3D, max_iterations, dist_pt2pl_thres,␣
↪num_dom_pln):

    # Initialize figure object and set figure size
    CSE_276C_fig = plt.figure(figsize=(7,7))

    # Initialize axes object
    CSE_276C_ax = Axes3D(CSE_276C_fig)

    # Plot Point Cloud Hallway
    CSE_276C_ax.scatter(hallway_pts[:,0], hallway_pts[:,1], hallway_pts[:,2],␣
↪c='cyan')

    # Label Axis
    CSE_276C_ax.set_xlabel("X", fontsize = 15)
    CSE_276C_ax.set_ylabel("Y", fontsize = 15)
    CSE_276C_ax.set_zlabel("Z", fontsize = 15)

    # Plot title
    plt.title("Hallway Planes", fontsize = 18, fontweight = "bold")

    # Set plot colors
    colors = ['r', 'g', 'b', 'y', 'm', 'k']

    # Labels
    labels = ["Plane 1", "Plane 2", "Plane 3", "Plane 4", "Plane 5", "Plane 6"]
```

```
    # Create empty lists that store all the normal vectors, coefficent Cs, mean␣
↪points, inliers, and outliers
    # for every dominant plane
    normal_vec_ALL = []
    coeff_C_ALL = []
    mean_point_ALL = []
    inlier_pts_ALL = []
    outlier_pts_ALL = []

    # Initialize New Set of Point
    new_set_pts = hallway_pts

    for plane_ID in range(0, num_dom_pln):

        # Run RanSaC Plane Fitting
        normal_vec, coeff_C, mean_point, _, inlier_pts, outlier_pts =␣
↪RanSaC_algorithm(new_set_pts, max_iterations[plane_ID],␣
↪dist_pt2pl_thres[plane_ID])

        # Plot Plane
        X_coord = np.linspace(min(inlier_pts[:,0]),max(inlier_pts[:,0]),3) #␣
↪Determine the x-axis limit
        Y_coord = np.linspace(min(inlier_pts[:,1]),max(inlier_pts[:,1]),3) #␣
↪Determine the y-axis limit
        x_coord, y_coord = np.meshgrid(X_coord, Y_coord)
        z_coord = -(normal_vec[0] / normal_vec[2]) * x_coord - (normal_vec[1]/
↪normal_vec[2]) * y_coord + (np.dot(normal_vec, mean_point)/normal_vec[2])
        #CSE_276C_ax.plot_wireframe(x_coord,y_coord,z_coord,color='k')
        #CSE_276C_ax.
↪plot_wireframe(x_coord,y_coord,z_coord,color=colors[plane_ID])
        pln = CSE_276C_ax.
↪plot_surface(x_coord,y_coord,z_coord,color=colors[plane_ID], label =␣
↪labels[plane_ID])
        pln._facecolors2d = pln._facecolors3d
        pln._edgecolors2d = pln._edgecolors3d
        CSE_276C_ax.legend()

        # Store Plane Information to the lists
        normal_vec_ALL.append(normal_vec)
        coeff_C_ALL.append(coeff_C)
        mean_point_ALL.append(mean_point)
        inlier_pts_ALL.append(inlier_pts)
        outlier_pts_ALL.append(outlier_pts)

        # Store the remaining points to the new_set_pts
```

```
        new_set_pts = outlier_pts

    # Show plot
    plt.show()

    return normal_vec_ALL, coeff_C_ALL, mean_point_ALL, inlier_pts_ALL,␣
 ↪outlier_pts_ALL

# Set parameters
max_iterations =[10000, 10000, 10000, 10000]
dist_pt2pl_thres =[0.2, 0.2, 0.2, 0.2]
num_dom_pln = 4

# Let's hunt down the planes in the hallway!
normal_vec_ALL, coeff_C_ALL, mean_point_ALL, inlier_pts_ALL, outlier_pts_ALL =␣
 ↪dominant_planes_Hunter(hallway_pts, max_iterations, dist_pt2pl_thres,␣
 ↪num_dom_pln)



### Plot samples by Professor Christensen and Dr. Wong
#fig = plt.figure()
#ax = fig.add_subplot(111, projection='3d')
#ax.plot_trisurf(hallway[:,0], hallway[:,1], hallway[:,2], color='white',␣
 ↪edgecolors='grey', alpha=0.5)
#ax.scatter(hallway[:,0], hallway[:,1], hallway[:,2], c='red')
#plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[58]:

```
# Show the 1st plane equation from the hallway
print("The equation of 1st fitted plane from the hallway:")
print("%f x + %f y + %f z + %f = 0" % (normal_vec_ALL[0][0],␣
 ↪normal_vec_ALL[0][1], normal_vec_ALL[0][2], coeff_C_ALL[0]))

# Show the 2nd plane equation from the hallway
print("\nThe equation of 2nd fitted plane from the hallway:")
print("%f x + %f y + %f z + %f = 0" % (normal_vec_ALL[1][0],␣
 ↪normal_vec_ALL[1][1], normal_vec_ALL[1][2], coeff_C_ALL[1]))

# Show the 3rd plane equation from the hallway
print("\nThe equation of 3rd fitted plane from the hallway:")
print("%f x + %f y + %f z + %f = 0" % (normal_vec_ALL[2][0],␣
 ↪normal_vec_ALL[2][1], normal_vec_ALL[2][2], coeff_C_ALL[2]))
```

```python
# Show the 4th plane equation from the hallway
print("\nThe equation of 4th fitted plane from the hallway:")
print("%f x + %f y + %f z + %f = 0" % (normal_vec_ALL[3][0],␣
↪normal_vec_ALL[3][1], normal_vec_ALL[3][2], coeff_C_ALL[3]))
```

```
The equation of 1st fitted plane from the hallway:
-0.975413 x + -0.145478 y + -0.165546 z + 1.098832 = 0

The equation of 2nd fitted plane from the hallway:
-0.987955 x + -0.009184 y + -0.154469 z + -0.748513 = 0

The equation of 3rd fitted plane from the hallway:
-0.047033 x + -0.985149 y + -0.165134 z + 1.250932 = 0

The equation of 4th fitted plane from the hallway:
0.172249 x + -0.133171 y + -0.976010 z + 6.806987 = 0
```

The equations of each plane in the hallway are presented above. The RANSAC algorithm was run at a higher iterations and d_{dest}, which are 10000 and 0.2 for every plane in the hallway because the point cloud data in the hallway was not that clean and a little scatter. Hence, the implemented "dominant_planes_Hunter()" user-defined function that also uses the RANSAC algorithm succesfully extracted all the dominant plane from the relatively empty hallway.

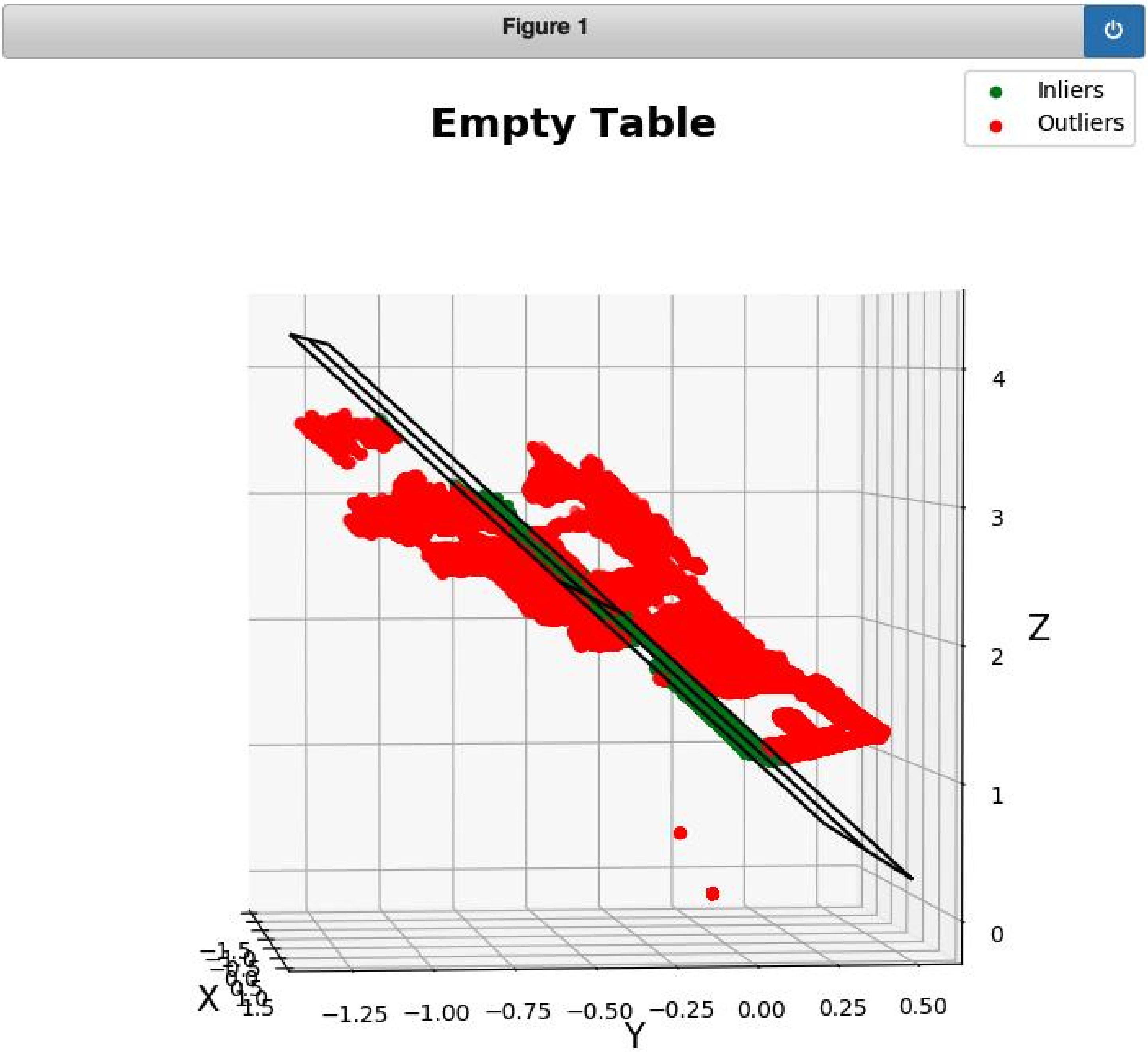


Figure 1
Empty Table
Inliers
Outliers
4
3
2
1
0
Z
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
Y
X

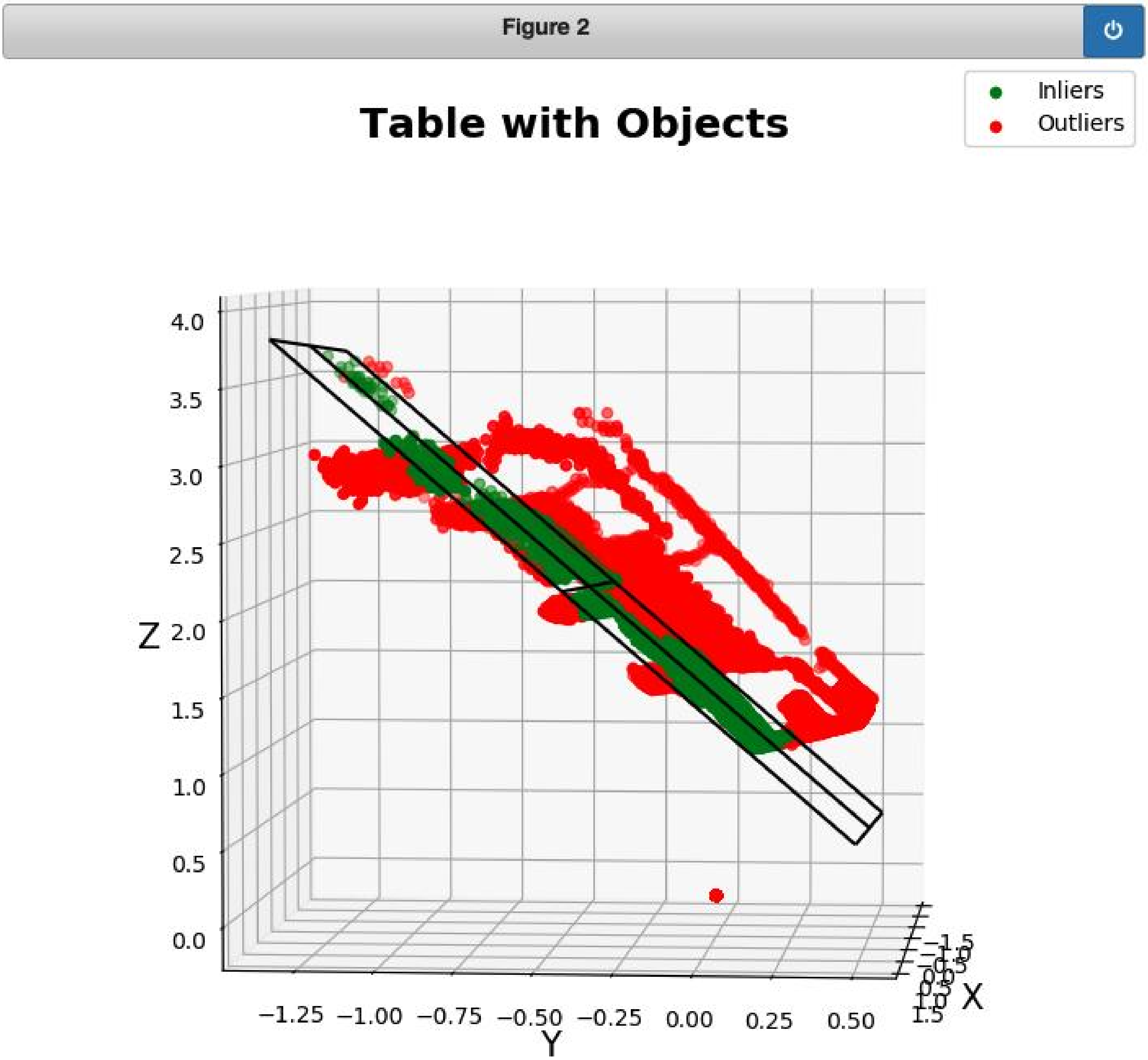
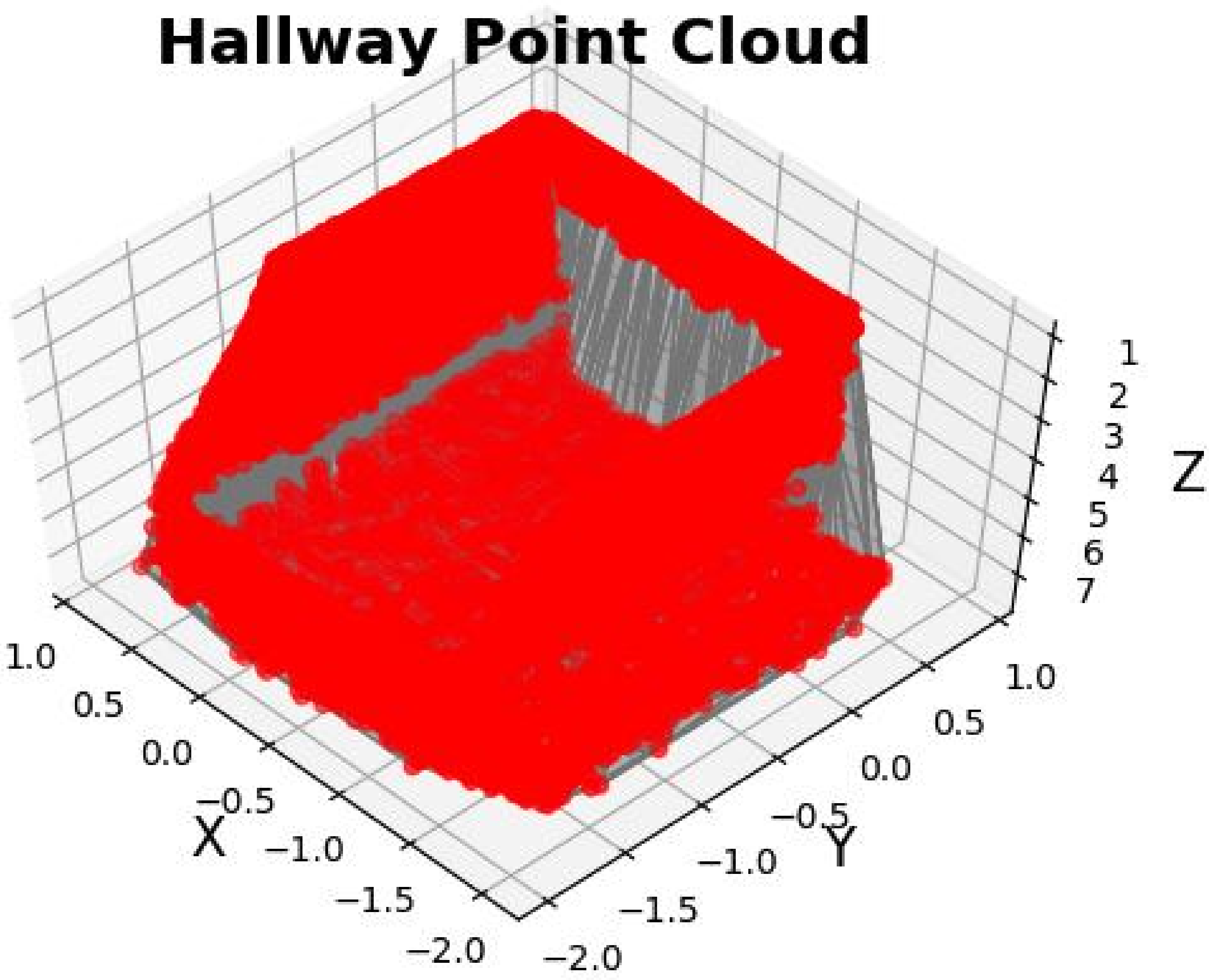
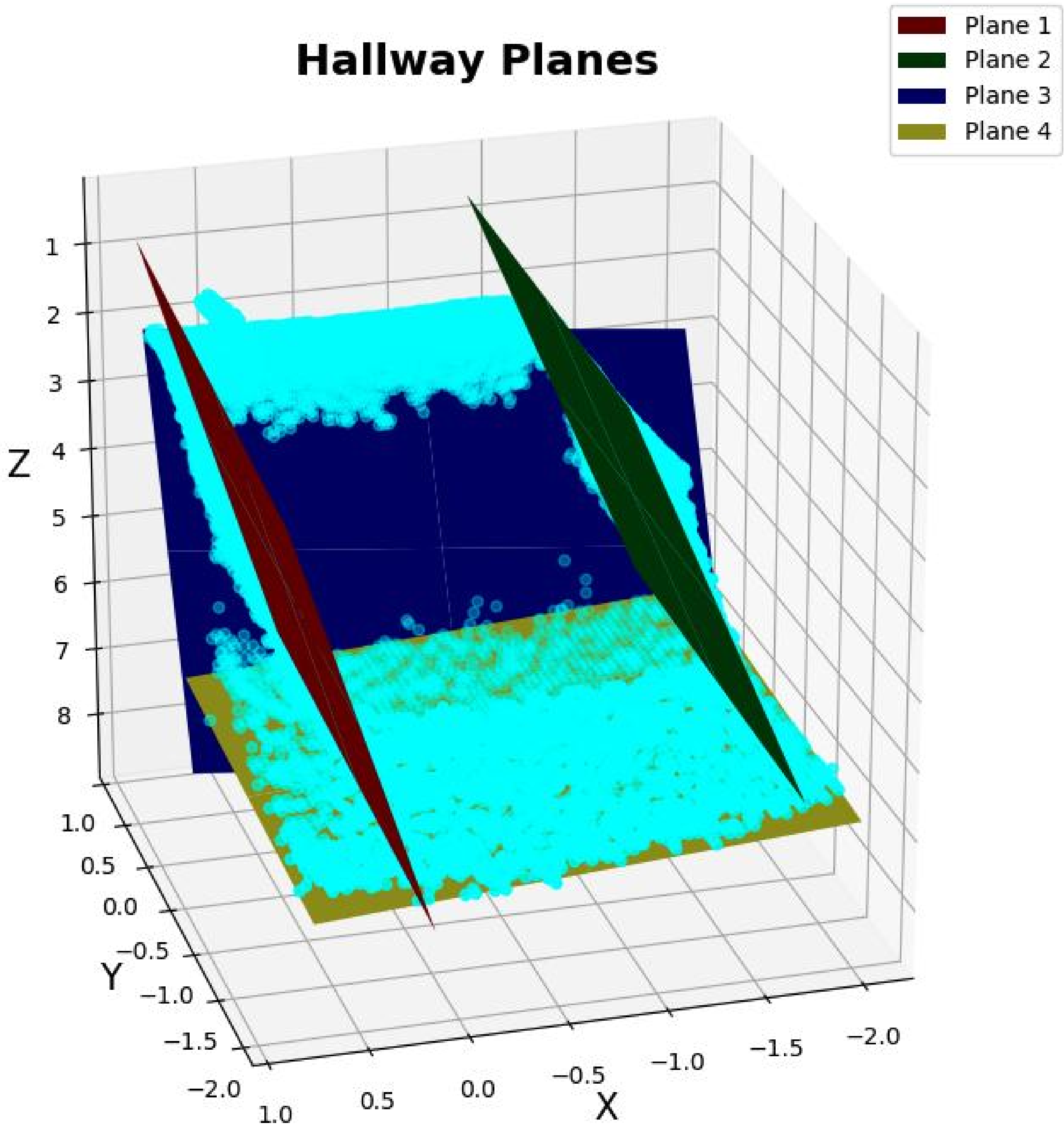


Figure 2
Table with Objects
Inliers
Outliers
Z
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Y
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
X
−1.5
1.5


Hallway Point Cloud
X
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Y
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Z
1
2
3
4
5
6
7


Hallway Planes
Plane 1
Plane 2
Plane 3
Plane 4
Z
1
2
3
4
5
6
7
8
Y
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
X
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0