Advanced Computation For Energy Assignments

Name: Abisoye Akinloye Samson

Track: Energy Engineering

Academic Year: 2025

Table of Contents

Assignment 1

https://www.wunderground.com/dashboard/pws/IVANDU8/table/2024-08-25/2024-08-25/monthly

Take the csv file, read it with Matlab, extract the columns:

- Average Temperature,
- Average Humidity,
- Average Wind speed

Display the curves over the days of the month.

```
%% Program to read, manipulate, and plot weather data
clc; clear; close all;

%% Load the Data
```

```
data = readtable("weather.csv", "VariableNamingRule","preserve",
"ReadVariableNames",true);
% display 5 rows of the data
data(2:5,:);

% select needed columns from the data (temperature, humidity, speed)
date = data(2:end,"Var1"); % date -> the first column
temp = data(2:end,2:4);    % temperature
humdity = data(2:end,8:10);% humidity
speed = data(2:end,11:13); % speed (mph)

% Date Column
% rename the header variable
date = renamevars(date, "Var1","Date");
% view the first 5 rows of the data
date(1:5,:)
```

ans = 5×1 table

	Date
1	08/07/2024
2	08/08/2024
3	08/09/2024
4	08/10/2024
5	08/11/2024

```
% temperature
temp = renamevars(temp,["Temperature","Dew Point","Humidity"],
["High_Temp","Avg_Temp","Low_Temp"]);
temp(1:5,:)
```

ans = 5×3 table

	High_Temp	Avg_Temp	Low_Temp
1	'81.0 °F'	'73.5 °F'	'63.7 °F'
2	'79.2 °F'	'68.6 °F'	'61.2 °F'
3	'84.4 °F'	'72.3 °F'	'62.6 °F'
4	'85.3 °F'	'73.8 °F'	'63.7 °F'
5	'88.3 °F'	'75.8 °F'	'59.7 °F'

```
% humidity
humdity = renamevars(humdity,["Var8","Var9","Var10"],
["High_Hum","Avg_Hum","Low_Hum"]);
humdity(1:5,:)
```

ans = 5×3 table

	High_Hum	Avg_Hum	Low_Hum
1	'88 %'	'72 %'	'59 %'

	High_Hum	Avg_Hum	Low_Hum
2	'92 %'	'67 %'	'45 %'
3	'80 %'	'64 %'	'45 %'
4	'92 %'	'71 %'	'48 %'
5	'88 %'	'62 %'	'47 %'

```
% speed
% rename speed header variable
speed = renamevars(speed,["Var11","Var12","Var13"],
["High_Speed","Avg_Speed","Low_Speed"]);
speed(1:5,:)
```

```
ans = 5×3 table
```

	High_Speed	Avg_Speed	Low_Speed
1	'11.2 mph'	'2.4 mph'	'0.0 mph'
2	'6.9 mph'	'1.0 mph'	'0.0 mph'
3	'13.7 mph'	'2.8 mph'	'0.0 mph'
4	'7.6 mph'	'1.6 mph'	'0.0 mph'
5	'9.2 mph'	'2.3 mph'	'0.0 mph'

Key Takeaways:

- **Renaming table header variable name. Source:** https://www.mathworks.com/help/matlab/ref/table.renamevars.html
- **Reading table and preserving variable naming. Source:** https://www.academia.edu/96835174/DATA_PRE_PROCESSING_IN_MATLAB

Data Manipulation

- The CSV file downloaded is messy (i.e containing alphanumeric) including charater, degrees, and percentage symbol.
- However, since numeric datapoint (quantitative data) are need for the plotting.
- Characters and symbols should be removed. Hence, Instead of doing it manually using Excel,
- Matlab has some in-built function for data cleaning and manipulation.

```
% Data Manipulation
% remove degree F from the temperature
avg_temp_ = temp(:,2).Variables;
A = string(avg_temp_); % change cell to string array
% data casting string -> number
avg_temp = str2double(strrep(A, ' °F', ''))
```

```
avg_temp = 25×1
   73.5000
```

```
    68.6000
    72.3000
    73.8000
    75.8000
    80.8000
    83.1000
    68.2000
    71.4000
    72.6000
       ⋮
```

```
% remove percentage from humidity
avg_humdity_ =  humdity(:,2).Variables;
B = string(avg_humdity_); % chnage cell to string array
avg_humidity = str2double(strrep(B, '%', ''))
```

```
avg_humidity = 25×1
    72
    67
    64
    71
    62
    62
    59
    90
    73
    62
     ⋮
```

```
% remove mph from speed
avg_speed_ =  speed(:,2).Variables;
C = string(avg_speed_); % chnage cell to string array
avg_speed = str2double(strrep(C, ' mph', ''))
```

```
avg_speed = 25×1
    2.4000
    1.0000
    2.8000
    1.6000
    2.3000
    1.8000
    2.0000
    2.5000
    0.8000
    2.1000
       ⋮
```

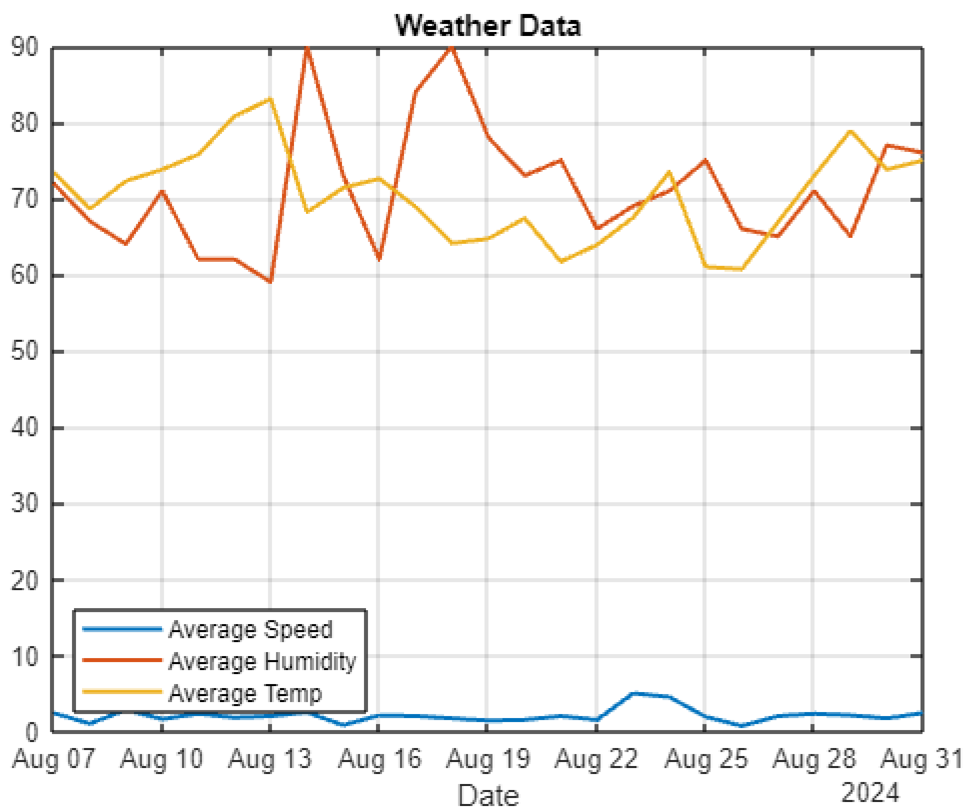
```
% date already in datetime datatype
date = date.Variables;
```

Key Takeaways:

- **How to change string datatype to double. (Source:** https://www.mathworks.com/help/matlab/ref/str2double.html**)**

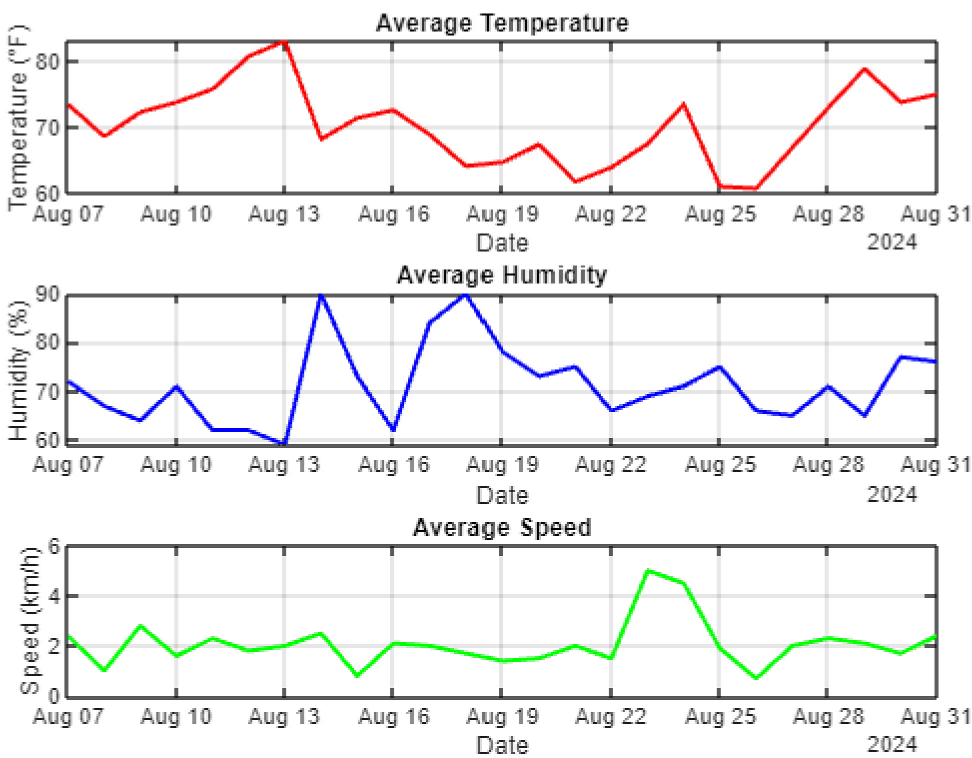
- **How to replace occurrence of a string to another string. (Source:** https://www.mathworks.com/help/matlab/ref/strrep.html**)**

```
% plot the graph
figure(1)
plot(date,avg_speed,date,avg_humidity,date,avg_temp,'LineWidth', 1.5)
xlabel("Date")
legend(["Average Speed","Average Humidity", "Average Temp"],"Location","southwest")
title("Weather Data")
grid on
```




```
%% Using Subplot
figure(2)
% Subplot 1: Average Temperature
subplot(3,1,1);       % 3 rows, 1 column, 1st subplot
plot(date, avg_temp, '-r', 'LineWidth', 1.5);
title('Average Temperature');
xlabel('Date');
ylabel('Temperature (°F)');
grid on;

% Subplot 2: Average Humidity
subplot(3,1,2);       % 3 rows, 1 column, 2nd subplot
plot(date, avg_humidity, '-b', 'LineWidth', 1.5);
```

```
title('Average Humidity');
xlabel('Date');
ylabel('Humidity (%)');
grid on;

% Subplot 3: Average Speed
subplot(3,1,3);       % 3 rows, 1 column, 3rd subplot
plot(date, avg_speed, '-g', 'LineWidth', 1.5);
title('Average Speed');
xlabel('Date');
ylabel('Speed (km/h)');
grid on;
```



M Script Code

```
%% Program to read, manipulate, and plot weather data
clc; clear; close all
%% Load the Data
data = readtable("weather.csv", "VariableNamingRule","preserve", "ReadVariableNames",true);
data

% select needed columns data (temperature, humidity, speed)
date = data(2:end,"Var1"); % date -> the first column
temp = data(2:end,2:4);    % temperature
humdity = data(2:end,8:10);% humidity
speed = data(2:end,11:13); % speed (mph)

%% data manipulation
% remove degree F from the temperature
```

```
avg_temp_ = temp(:,2).Variables;
A = string(avg_temp_); % chnage cell to string array
% data casting string -> number
avg_temp = str2double(strrep(A, ' °F', ''));

% remove percentage from humidity
avg_humdity_ =  humdity(:,2).Variables;
B = string(avg_humdity_); % chnage cell to string array
avg_humidity = str2double(strrep(B, '%', ''));

% remove mph from speed
avg_speed_ =  speed(:,2).Variables;
C = string(avg_speed_); % chnage cell to string array
avg_speed = str2double(strrep(C, ' mph', ''));

% date already in datetime datatype
date = date.Variables;

%% plot the graph
figure(1)
plot(date,avg_speed,date,avg_humidity,date,avg_temp,'LineWidth', 1.5)
xlabel("Date")
legend(["Average Speed","Average Humidity", "Average Temp"])
title("Weather Data")
grid on

%% Using Subplot
figure(2)
% Subplot 1: Average Temperature
subplot(3,1,1);       % 3 rows, 1 column, 1st subplot
plot(date, avg_temp, '-r', 'LineWidth', 1.5);
title('Average Temperature');
xlabel('Date');
ylabel('Temperature (°F)');
grid on;

% Subplot 2: Average Humidity
subplot(3,1,2);       % 3 rows, 1 column, 2nd subplot
plot(date, avg_humidity, '-b', 'LineWidth', 1.5);
title('Average Humidity');
xlabel('Date');
ylabel('Humidity (%)');
grid on;

% Subplot 3: Average Speed
subplot(3,1,3);       % 3 rows, 1 column, 3rd subplot
plot(date, avg_speed, '-g', 'LineWidth', 1.5);
title('Average Speed');
xlabel('Date');
ylabel('Speed (km/h)');
grid on;
```

Assignment 2

Implement the algorithms of Fixed-point theorem and Newton-Raphson to solve the same equation

$f(x) = x - \cos(x)$

Record the values x_k and $f(x_k)$ for both algorithm in arrays as explained in previous Unit Plot, on the same figure.

Fixed Point Method

```
% Numerical Integration Using Fixed-Point Iteration and Newton-Raphson Method
% f(x) = x - cos(x)

clear; clc; close all;

%% Variables
iter_fp = 0; iter_nr = 0;             % iteration counter for fixed-point and Newton-Raphson
iterlim_fp = 100; iterlim_nr = 100;   % maximum number of iterations for fixed-point and Newton-Raphson
elim_fp = 1e-6; elim_nr = 1e-6;       % error tolerance for fixed-point and Newton-Raphson
x0_fp = 0; x0_nr = 0;                 % initial guess for fixed-point and Newton-Raphson
x_fp = x0_fp; x_nr = x0_nr;           % current approximation
e_fp = 1e20; e_nr = 1e20;             % initialize error to a large number

%% Fixed-point iteration loop
fprintf('Resolution by Fixed-point theorem\n');
```

```
Resolution by Fixed-point theorem
```

```
pt_xt = zeros(1, iter_fp);            % x initialized as empty array
pt_fxt = zeros(1, iter_fp);           % f(x) initialized as empty array

while (e_fp > elim_fp) && (iter_fp < iterlim_fp)
    % fixed point method: x = g(x) = cos(x)
    xnew = cos(x_fp);                 % g_function(x) = cos(x)
    e_fp = abs(xnew - x_fp);             % compute absolute error
    x_fp = xnew;                      % update the approximation
    iter_fp = iter_fp + 1;            % increment the iteration counter

    pt_xt(iter_fp)= x_fp;             % store the x value
    pt_fxt(iter_fp) = x_fp - cos(x_fp);

    fprintf('iter %d: x = %.8f, e = %.8e\n', iter_fp, x_fp, e_fp);
end
```

```
iter 1: x = 1.00000000, e = 1.00000000e+00
iter 2: x = 0.54030231, e = 4.59697694e-01
iter 3: x = 0.85755322, e = 3.17250910e-01
iter 4: x = 0.65428979, e = 2.03263425e-01
iter 5: x = 0.79348036, e = 1.39190568e-01
iter 6: x = 0.70136877, e = 9.21115851e-02
```

```
iter 7: x = 0.76395968, e = 6.25909093e-02
iter 8: x = 0.72210243, e = 4.18572579e-02
iter 9: x = 0.75041776, e = 2.83153367e-02
iter 10: x = 0.73140404, e = 1.90137193e-02
iter 11: x = 0.74423735, e = 1.28333125e-02
iter 12: x = 0.73560474, e = 8.63261446e-03
iter 13: x = 0.74142509, e = 5.82034617e-03
iter 14: x = 0.73750689, e = 3.91819610e-03
iter 15: x = 0.74014734, e = 2.64044505e-03
iter 16: x = 0.73836920, e = 1.77813145e-03
iter 17: x = 0.73956720, e = 1.19799809e-03
iter 18: x = 0.73876032, e = 8.06882338e-04
iter 19: x = 0.73930389, e = 5.43572523e-04
iter 20: x = 0.73893776, e = 3.66135682e-04
iter 21: x = 0.73918440, e = 2.46643056e-04
iter 22: x = 0.73901826, e = 1.66137344e-04
iter 23: x = 0.73913018, e = 1.11914102e-04
iter 24: x = 0.73905479, e = 7.53857828e-05
iter 25: x = 0.73910557, e = 5.07811796e-05
iter 26: x = 0.73907137, e = 3.42066276e-05
iter 27: x = 0.73909441, e = 2.30420801e-05
iter 28: x = 0.73907889, e = 1.55213841e-05
iter 29: x = 0.73908934, e = 1.04554084e-05
iter 30: x = 0.73908230, e = 7.04288099e-06
iter 31: x = 0.73908704, e = 4.74417293e-06
iter 32: x = 0.73908385, e = 3.19573033e-06
iter 33: x = 0.73908600, e = 2.15268313e-06
iter 34: x = 0.73908455, e = 1.45007292e-06
iter 35: x = 0.73908553, e = 9.76786712e-07
```

```
fprintf('The fixed point is approximately x = %.8f\n', x_fp);
```

```
The fixed point is approximately x = 0.73908553
```

Newton Raphson Method

```
%% Newton Raphson iteration loop

fprintf('\nResolution by Netwon Raphson Method\n');
```

```
Resolution by Netwon Raphson Method
```

```
nr_xt = zeros(1, iter_nr);  nr_fxt = zeros(1, iter_nr);    % x and f(x) initialized
as empty array

while ((e_nr > elim_nr) && (iter_nr < iterlim_nr))
    f = x_nr - cos(x_nr);             % f(x) = x - cos(x)
    df = 1 + sin(x_nr);               % f'(x) = 1 + sin(x) differentiating f(x)
    if df ~= 0
        xnew = x_nr - (f/df);         % Newton-Raphson formula
    else
        fprintf('df = 0, end of iterations\n');
        break
    end

    e_nr = abs(xnew - x_nr);
    x_nr = xnew;
```

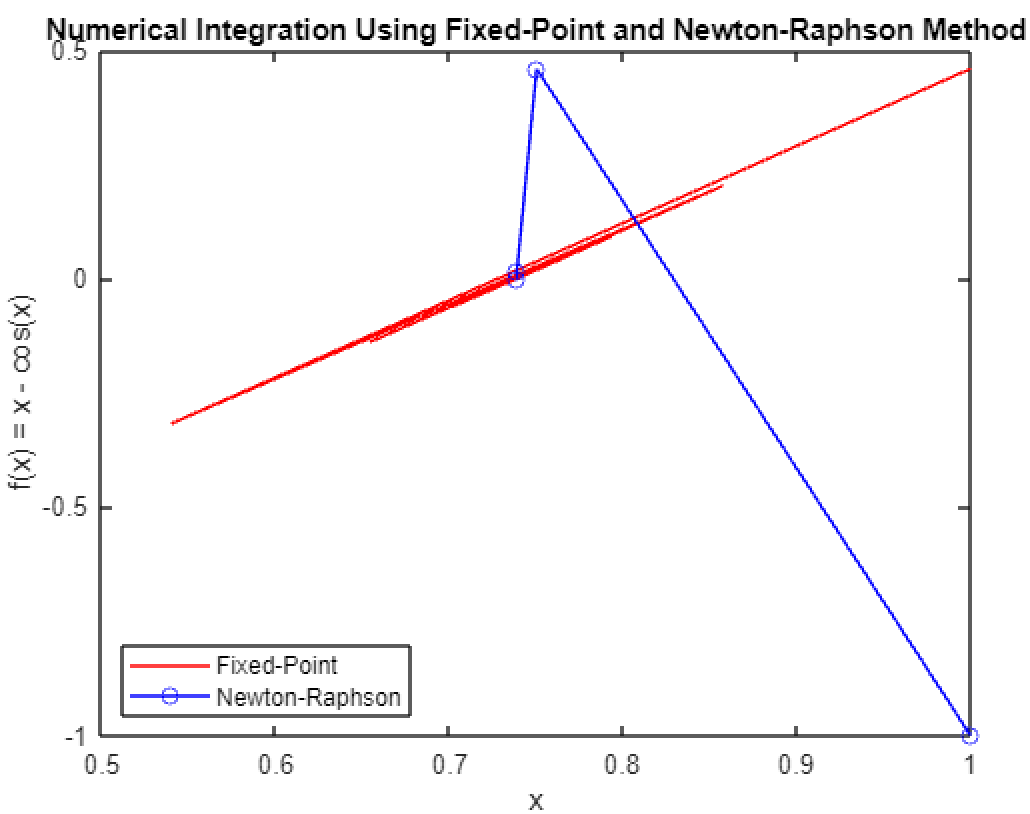
```
    iter_nr = iter_nr + 1;

    nr_xt(iter_nr) = x_nr;
    nr_fxt(iter_nr) = f;

    fprintf('iter %d: x = %.6f, e = %.5e\n', iter_nr, x_nr, e_nr);
end
```

```
iter 1: x = 1.000000, e = 1.00000e+00
iter 2: x = 0.750364, e = 2.49636e-01
iter 3: x = 0.739113, e = 1.12510e-02
iter 4: x = 0.739085, e = 2.77575e-05
iter 5: x = 0.739085, e = 1.70123e-10
```

```

%% Plot the graphs
plot(pt_xt, pt_fxt, '-r')
hold;
```

```
Current plot held
```

```
plot(nr_xt, nr_fxt, 'o-b')
ylabel('f(x) = x - cos(x)')
xlabel('x')
legend('Fixed-Point', 'Newton-Raphson','Location','southwest')
title('Numerical Integration Using Fixed-Point and Newton-Raphson Method')
```



M Script Code

```matlab
% Numerical Integration Using Fixed-Point Iteration and Newton-Raphson Method
% f(x) = x - cos(x)

clear; clc; close all;

%% Variables
iter_fp = 0; iter_nr = 0;             % iteration counter for fixed-point and Newton-Raphson
iterlim_fp = 100; iterlim_nr = 100;   % maximum number of iterations for fixed-point and Newton-Raphson
elim_fp = 1e-6; elim_nr = 1e-6;       % error tolerance for fixed-point and Newton-Raphson
x0_fp = 0; x0_nr = 0;                 % initial guess for fixed-point and Newton-Raphson
x_fp = x0_fp; x_nr = x0_nr;           % current approximation
e_fp = 1e20; e_nr = 1e20;             % initialize error to a large number

%% Fixed-point iteration loop
fprintf('Resolution by Fixed-point theorem\n');
pt_xt = zeros(1, iter_fp);            % x initialized as empty array
pt_fxt = zeros(1, iter_fp);           % f(x) initialized as empty array

while (e_fp > elim_fp) && (iter_fp < iterlim_fp)
    % fixed point method: x = g(x) = cos(x)
    xnew = cos(x_fp);                 % g_function(x) = cos(x)
    e_fp = abs(xnew - x_fp);             % compute absolute error
    x_fp = xnew;                      % update the approximation
    iter_fp = iter_fp + 1;            % increment the iteration counter

    pt_xt(iter_fp)= x_fp;             % store the x value
    pt_fxt(iter_fp) = x_fp - cos(x_fp);

    fprintf('iter %d: x = %.8f, e = %.8e\n', iter_fp, x_fp, e_fp);
end
fprintf('The fixed point is approximately x = %.8f\n', x_fp);

%% Newton Raphson iteration loop

fprintf('\nResolution by Netwon Raphson Method\n');
nr_xt = zeros(1, iter_nr);  nr_fxt = zeros(1, iter_nr);    % x and f(x) initialized as empty array

while ((e_nr > elim_nr) && (iter_nr < iterlim_nr))
    f = x_nr - cos(x_nr);            % f(x) = x - cos(x)
    df = 1 + sin(x_nr);              % f'(x) = 1 + sin(x) differentiating f(x)
    if df ~= 0
        xnew = x_nr - (f/df);        % Newton-Raphson formula
    else
        fprintf('df = 0, end of iterations\n');
        break
    end

    e_nr = abs(xnew - x_nr);
    x_nr = xnew;
    iter_nr = iter_nr + 1;

    nr_xt(iter_nr) = x_nr;
    nr_fxt(iter_nr) = f;

    fprintf('iter %d: x = %.6f, e = %.5e\n', iter_nr, x_nr, e_nr);
end
```

```
%% Plot the graphs
plot(pt_xt, pt_fxt, '-r')
hold;
plot(nr_xt, nr_fxt, 'o-b')
ylabel('f(x) = x - cos(x)')
xlabel('x')
legend('Fixed-Point', 'Newton-Raphson','Location','southwest')
title('Numerical Integration Using Fixed-Point and Newton-Raphson Method')
```

Assignment 3

- Build a battery emulator with its BMS
- 1s is 10 min (accelerated time)
- The model should include charging, discharging process, State Of Charge (SOC)
- Simulation using accelerated time in a loop

Battery Usage Scenarios

The simulation covers 24 hours with the following usage scenarios of my phone (Samsung Z Fold 5):

- **0 - 9am:** The phone was idle and the discharging rate was very low. **(**$0 \leq \text{time (hrs)} < 9)$
- **9 - 10:30am:** The phone was on standby mode as some background apps were running. $(9 \leq \text{time (hrs)} < 10:30)$
- **10:30 - 10:45am:** Tea break, Login to the social media $(10:30 \leq \text{time (hrs)} < 10:45)$
- **10:45 - 12pm**: The phone was in idle mode as running background was disable to concentrate in class. $(10:45 \leq \text{time (hrs)} < 12)$
- **12 - 1:30pm:** Lunch Break; surfed internet, chatted on social media, and play games. $(12 \leq \text{time (hrs)} < 13:30)$
- **1:30 - 3pm:** The phone was on standby mode $(13:30 \leq \text{time (hrs)} < 15)$
- **3 - 3:15pm:** Short break $(15 \leq \text{time (hrs)} < 15:15)$
- **3:15 - 4pm:** The phone was on standby mode.$(15:15 \leq \text{time (hrs)} < 16)$
- **4 - 5pm:** Bus station; watching YouTube videos to reinforce topics taught in class.$(16 \leq \text{time (hrs)} < 17)$
- **5 - 6:30pm:** Charging Mode when I got to cite. $(17 \leq \text{time (hrs)} < 18:30)$
- **6:30 - 10pm**: Reading and Studying with my mobile phone $(18:30 \leq \text{time (hrs)} < 22)$
- **10 - 11:30pm:** Bed time and charging $(22 \leq \text{time (hrs)} < 23:30)$
- **11:30pm - 12am:** Phone on Idle mode $(23:30 \leq \text{time (hrs)} < 00)$

```
% Battery SOC Estimation: Discharging, Charging, and Idle Scenario
clc; clear; close all;

%% Simulation parameters
tf = 24; % Total simulation time in hours
```

```
dt = 1/10; % Accelerated time step
Ntf = floor(tf/dt); % Number of simulation steps
t = 0; % Initial time in hour
fprintf("Simulation parameter initialized\n")
```

```
Simulation parameter initialized
```

```
%% Battery parameters
Qn = 5; % Battery capacity in Ah
soc = 1; % Initial SoC (100%)
i_dl = 0.015; % Idle current consumption in A
i = 0; % Initial current consumption in A
fprintf("Battery parameter initialized\nBattery Capacity = %d \nInitial State of
Charge = %d%%",Qn,soc*100)
```

```
Battery parameter initialized
Battery Capacity = 5
Initial State of Charge = 100%
```

```
%% Initialization
tt = zeros(1,Ntf); % Time vector
soc_t = zeros(1,Ntf); % SoC vector
i_c = zeros(1,Ntf); % Current charging vector
i_d = zeros(1,Ntf); % Current discharging vector
tk = 1; % Indexing array
fprintf("Initializing array needed for plotting graph")
```

```
Initializing array needed for plotting graph
```

```
%% Numerical integration (Euler) using a while loop
while (t < tf)
    if t <= 9
        i = i_dl; % Idle scenario (0 - 9am)
        i_d(tk) = i;
    elseif t > 9 && t <= 10.5
        % i = (remainingSoc - desiredSoc) * Qn/dt
        % dt -> t2 - t1 (analytically);
        % dt = 0.1 (accelerated time for t iteration)
        i = 0.1; % Standby scenario (9 - 10:30am)
        i_d(tk) = 1;
    elseif t > 10.5 && t <= 10.75
        i = 0.5; % Tea break (10:30 - 10:45am)
        i_d(tk) = i;
    elseif t > 10.75 && t <= 12
        i = i_dl; % Idle scenario
        i_d(tk) = i;
    elseif t > 12 && t <= 13.5
        i = 1.0; % Lunch break (12 - 1:30pm)
        i_d(tk) = i;
    elseif t > 13.5 && t <= 15
        i = 0.1; % Standby mode (1:30 - 3pm)
        i_d(tk) = i;
```

```
    elseif t > 15 && t <= 15.25
        i = 0.55; % Short break (3 - 3:15pm)
        i_d(tk) = i;
    elseif t > 15.25 && t <= 16
        i = 0.1; % Standby mode (3:15 - 4pm)
        i_d(tk) = i;
    elseif t > 16 && t <= 17
        i = 0.3; % Bus station and Surfing Internet (4 - 5:00pm)
        i_d(tk) = i;
    elseif t > 17 && t <= 18.5
        i = -1.6973; % Charging Mode (5 - 6:30pm)
        i_c(tk) = i;
    elseif t > 18.5 && t <= 22
        i = 1.32; % Discharging mode (6:30 - 10pm)
        i_d(tk) = i;
    elseif t > 22 && t <= 23.5
        i = -2.98; % Charging (10 - 11:30pm)
        i_c(tk) = i;
    else
        i = i_dl; % Idle mode (11:30 - 12am)
        i_d(tk) = i;
    end

    % Euler integration
    % deltaQ = i * dt; % Used Capacity
    % soc = soc - deltaQ/Qn; % Discharging scenario
    soc = soc - (i * dt)/Qn; % Discharging scenario

    % Fill the array
    soc_t(tk) = soc;
    tt(tk) = t;

    % Increment
    t = t + dt;
    tk = tk + 1;
end
fprintf("Remaining State of Charge based on the Scenario: %.f%%",soc*100)
```

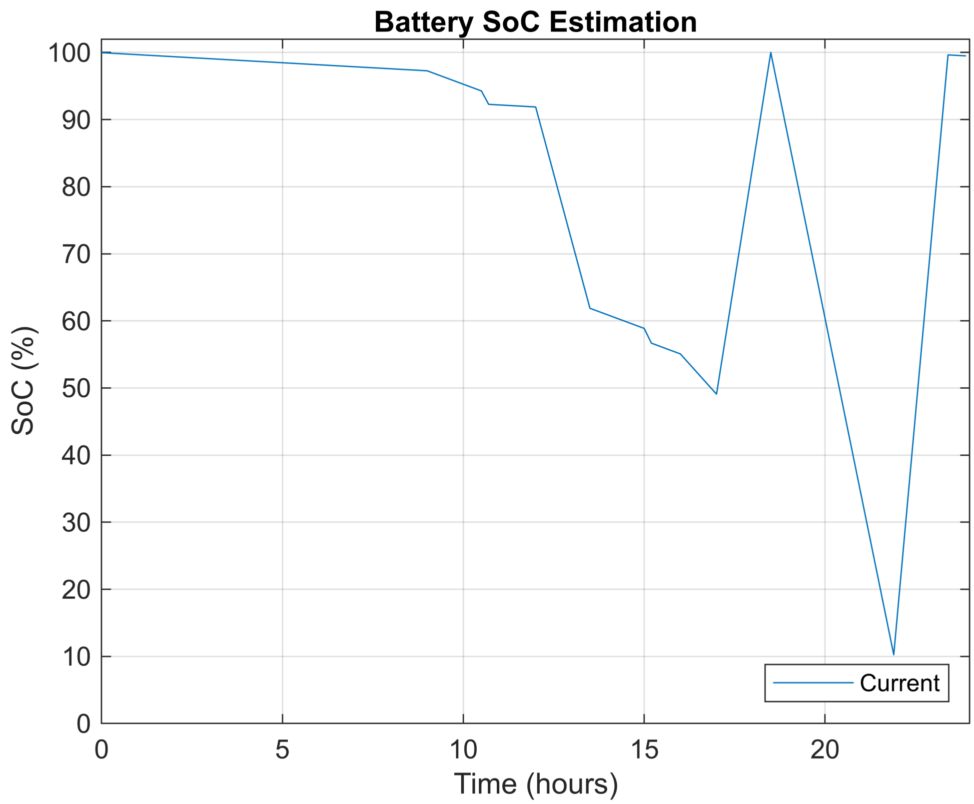
```
Remaining State of Charge based on the Scenario: 99%
```

$$i(t) = (\mathrm{soc}_k - \mathrm{soc}_{k+1}) * \frac{\mathrm{Qn}}{t_2 - t_1}$$

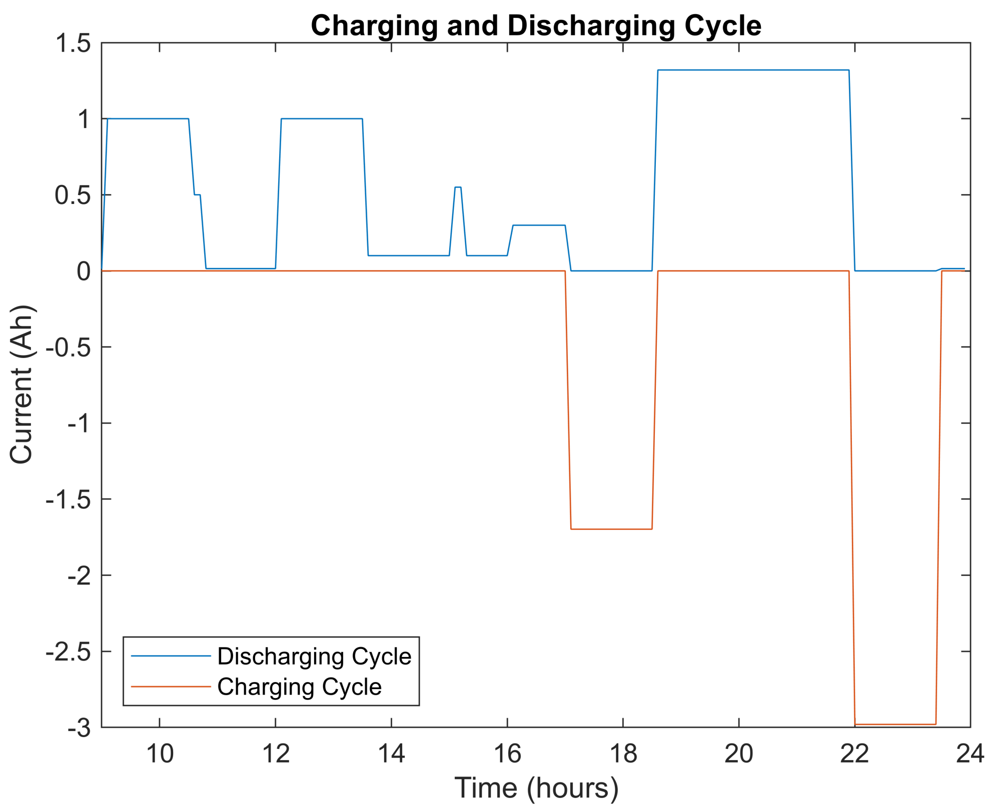
$$\text{Time range : } t_1 < t \le t_2$$

```
%% Plot Figure
figure;
plot(tt,soc_t*100)
xlabel('Time (hours)');
ylabel('SoC (%)');
title('Battery SoC Estimation');
```

```
xlim([0 tf])
ylim([0 102])
legend('Current','Location','southeast');
grid on;
```




```
% Discharging and Charging Cycle
figure;
plot(tt,i_d, tt, i_c);
xlim([9 tf])
xlabel('Time (hours)');
ylabel('Current (Ah)');
title('Charging and Discharging Cycle');
legend('Discharging Cycle','Charging Cycle','Location','southwest');
```



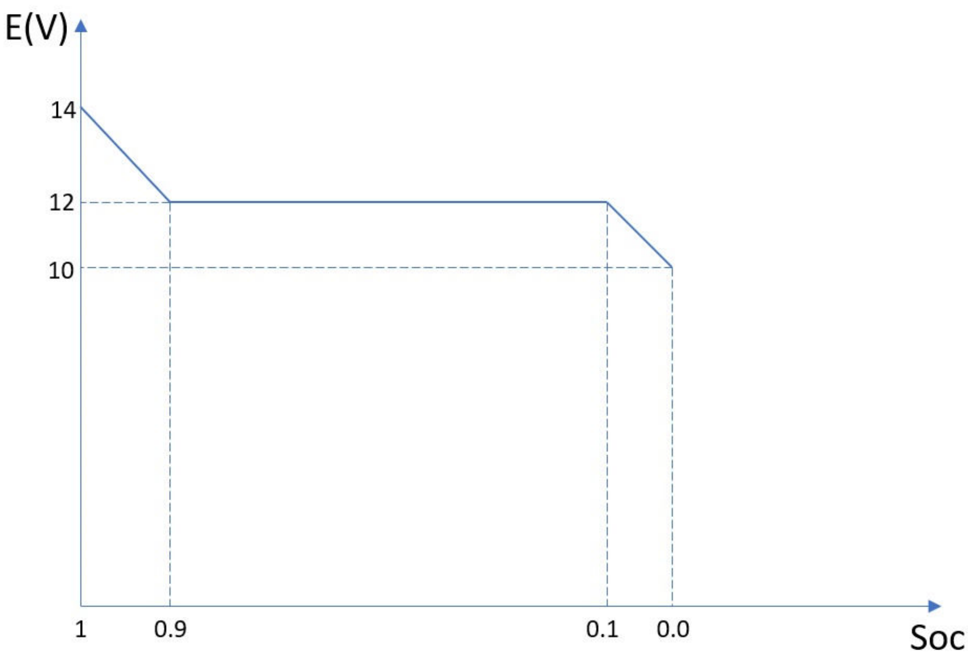
M Script Code

```
% Battery SOC Estimation: Discharging, Charging, and Idle Scenario
clc; clear; close all;

%% Simulation parameters
tf = 24; % Total simulation time in hours
dt = 1/10; % Accelerated time step (0.1)
Ntf = floor(tf/dt); % Number of simulation steps
t = 0; % Initial time in hour

%% Battery parameters
Qn = 5; % Battery capacity in Ah
soc = 1; % Initial SoC (100%)
i_dl = 0.015; % Idle current consumption in A
i = 0; % Initial current consumption in A

%% Initialization
tt = zeros(1,Ntf); % Time vector
soc_t = zeros(1,Ntf); % SoC vector
i_c = zeros(1,Ntf); % Current charging vector
i_d = zeros(1,Ntf); % Current discharging vector
tk = 1; % Indexing array

%% Numerical integration (Euler) using a while loop
while (t < tf)
    if t <= 9
        i = i_dl; % Idle scenario (0 - 9am)
        i_d(tk) = i;
```

```
    elseif t > 9 && t <= 10.5
        i = 0.1; % Standby scenario (9 - 10:30am)
        i_d(tk) = 1;
    elseif t > 10.5 && t <= 10.75
        i = 0.5; % Tea break (10:30 - 10:45am)
        i_d(tk) = i;
    elseif t > 10.75 && t <= 12
        i = i_dl; % Idle scenario
        i_d(tk) = i;
    elseif t > 12 && t <= 13.5
        i = 1.0; % Lunch break (12 - 1:30pm)
        i_d(tk) = i;
    elseif t > 13.5 && t <= 15
        i = 0.1; % Standby mode (1:30 - 3pm)
        i_d(tk) = i;
    elseif t > 15 && t <= 15.25
        i = 0.55; % Short break (3 - 3:15pm)
        i_d(tk) = i;
    elseif t > 15.25 && t <= 16
        i = 0.1; % Standby mode (3:15 - 4pm)
        i_d(tk) = i;
    elseif t > 16 && t <= 17
        i = 0.3; % Bus station and Surfing Internet (4 - 5:00pm)
        i_d(tk) = i;
    elseif t > 17 && t <= 18.5
        i = -1.6973; % Charging Mode (5 - 6:30pm)
        i_c(tk) = i;
    elseif t > 18.5 && t <= 22
        i = 1.32; % Discharging mode (6:30 - 10pm)
        i_d(tk) = i;
    elseif t > 22 && t <= 23.5
        i = -2.98; % Charging (10 - 11:30pm)
        i_c(tk) = i;
    else
        i = i_dl; % Idle mode (11:30 - 12am)
        i_d(tk) = i;
    end


    % Euler integration
    % deltaQ = i * dt; % Current consumption
    % soc = soc - deltaQ/Qn; % Discharging scenario
    soc = soc - (i * dt)/Qn; % Discharging scenario

    % Fill the array
    soc_t(tk) = soc;
    tt(tk) = t;

    % Increment
    t = t + dt;
    tk = tk + 1;
end

%% Plot Figure
figure;
plot(tt,soc_t*100)
xlabel('Time (hours)');
```

```
ylabel('SoC (%)');
title('Battery SoC Estimation');
xlim([0 tf])
ylim([0 100])
legend('Current','Location','southeast');
grid on;

figure;
plot(tt,i_d, tt, i_c);
xlim([9 tf])
xlabel('Time (hours)');
ylabel('Current (Ah)');
title('Charging and Discharging Cycle');
legend('Discharging Cycle','Charging Cycle','Location','southwest');
```

OCV Vs SoC



Questions

1. Determine the linear equations of the characterisitics and their operating range.
2. Write a matlab script, the corresponding function of $E = f(\mathrm{SoC})$
3. The battery has an internal resistance of $r = 0.05\,\Omega$ Write another function $V = f(I, \mathrm{SoC})$ that outputs the terminal voltage $V = E - r\,I$ and uses previous function
4. (Replicate the figure) Plot the graph of E(V) against SoC. **Additional:** Use Sigmoid function to transform the linear to smooth curve.

Segment A

@ soc = 1 to 0.9; ocv drops from 14v to 12v

Equation of line : $y = \mathrm{ax} + b$ $- - - - - - - - - - -(i)$

where $y = E, x = \mathrm{soc}$

slope $(a) = \dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{E_2 - E_1}{\mathrm{soc}_2 - \mathrm{soc}_1} - - - - - - - - -$ (ii)

$a = \dfrac{12 - 14}{0.9 - 1} = \dfrac{-2}{-0.1} = 20$

Hence,

$E = 20 * \mathrm{soc} + b - - - - - - - - - - - - - - - - -$ (iii)

$(\mathrm{soc}, E) = (1, 14)$

Substitute the value in (iii)

$14 = 20 * 1 + b$

$14 - 20 = b$

$b = -6 -$ (iv)

substitute (iv) into (iii)

The equation of line is $E = 20 * \mathrm{soc} - 6$; Operating range : $0.9 < \mathrm{soc} \leq 1$

Segment B

@ soc 0.9 to 0.1, The graph is constant.

E = **12V**

Operating range : $0.1 < \mathrm{soc} \leq 0.9$

Segment C

@ soc = 0.1 to 0, ocv drops from 12v to 10v

$\text{Equation of line}: y = \text{ax} + b \; ------------(i)$

where $y = E, x = \text{soc}$

$\text{slope}\,(a) = \dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{E_2 - E_1}{\text{soc}_2 - \text{soc}_1} --------- (\text{ii})$

$a = \dfrac{10 - 12}{0 - 0.1} = \dfrac{-2}{-0.1} = 20 ----------- \quad (\text{iii})$

Hence,

$E = 20 * \text{soc} + b ---------------- (\text{iv})$

$(\text{soc}, E) = (0.1, 12)$

Substitute the value in (iv)

$12 = 20 * 0.1 + b$

$12 - 2 = b$

$b = 10 \; ---------------------- (v)$

substitute (v) into (iv)

The equation of line is $E = 20 * \text{soc} + 10$; Operating range : $0 \leq \text{soc} \leq 0.1$

$E = f(\text{Soc})$

```
function ocv = ocv(soc)
% E = f(soc)
    ocv = zeros(size(soc));

    for i = 1:length(soc)
        s = soc(i);
        if (s > 1 || s < 0)
            disp('SoC value out of range')
        elseif s > 0.9
            ocv(i) = 20 * s - 6;
        elseif s > 0.1
            ocv(i) = 12; % constant OCV
        else
            ocv(i) = 20 * s + 10;
        end
    end
end
```

$V = f(I, \text{SoC})$

```
function V = terminal_voltage(I, SoC)
    r = 0.05; % Internal resistance in ohms
    V = ocv(SoC) - r * I;
end
```

Sigmoid function

$$\sigma = \frac{1}{1 - e^{-x}}$$

```
function ocv = ocv_sigmoid(soc)
    alpha = 0.01;
    beta = 0.01;

    % anonymous function
    sigmoid = @(x) 1./(1 + exp(-x));

    % ocv and sigmoid (x) relationship
    ocv = 10 + 2 * sigmoid((soc - 0.1) / beta) + 2 * sigmoid((soc - 0.9) / alpha);
end
```

$$\mathrm{ocv} = 10 + 2 * \sigma\left(\frac{\mathrm{soc} - 0.1}{\beta}\right) + 2 * \sigma\left(\frac{\mathrm{soc} - 0.9}{\alpha}\right)$$

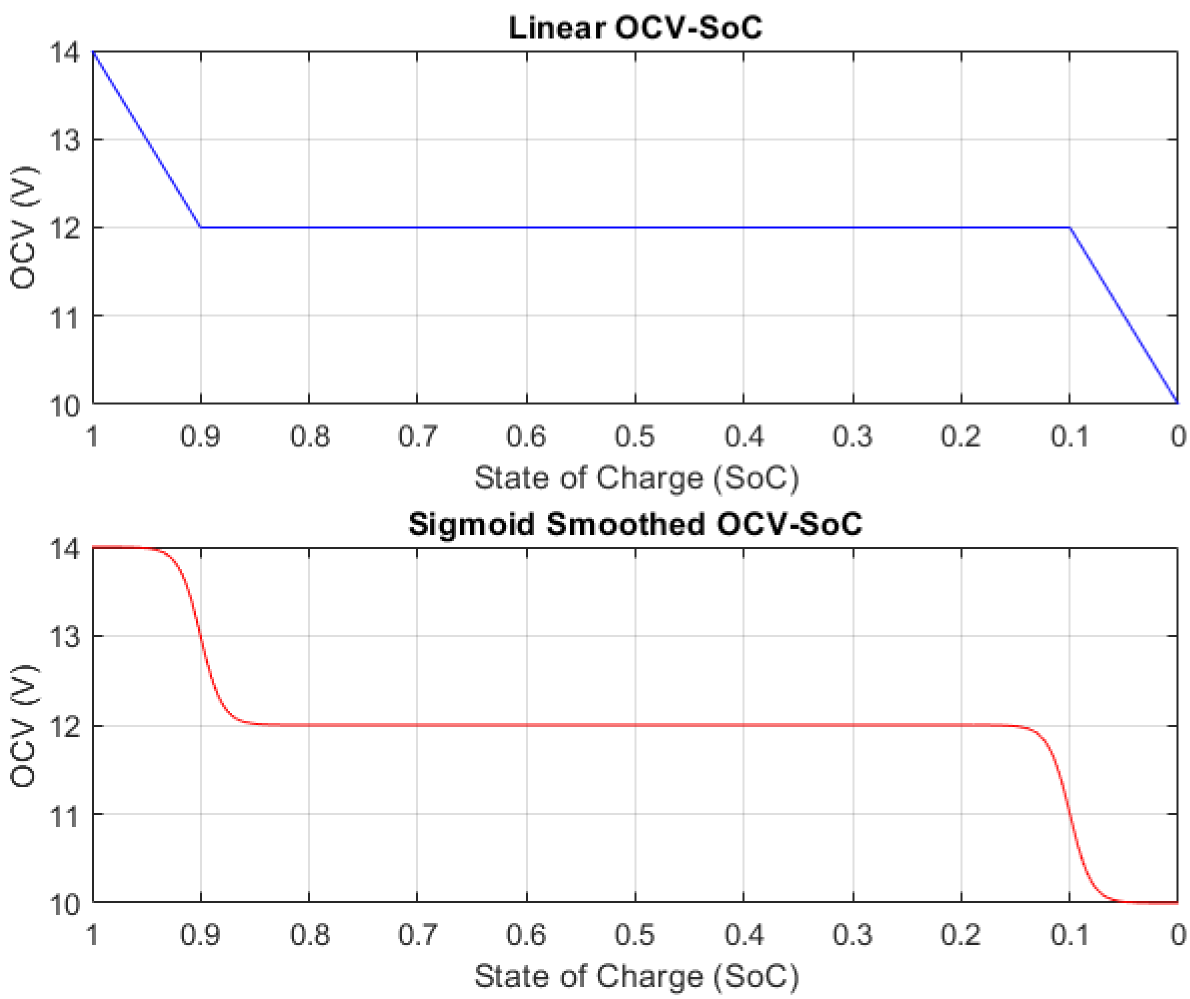
- The equation provides a smooth, continuous transition between these voltage levels without abrupt jumps,
- Which is especially useful in simulations where smooth behavior are desired.
- The base value (10V) sets the minimum voltage. This means that at very low soc values, the OCV will be near 10 V.
- $2 * \sigma\left(\frac{\mathrm{soc} - 0.1}{\beta}\right)$ adds 2 volts as soc increases past 0.1. The parameter β controls how gradual or sharp this transition.
- $2 * \sigma\left(\frac{\mathrm{soc} - 0.9}{\alpha}\right)$ adds another 2 V as soc increases past 0.9. The parameter α\alphaα similarly controls the steepness of this transition.

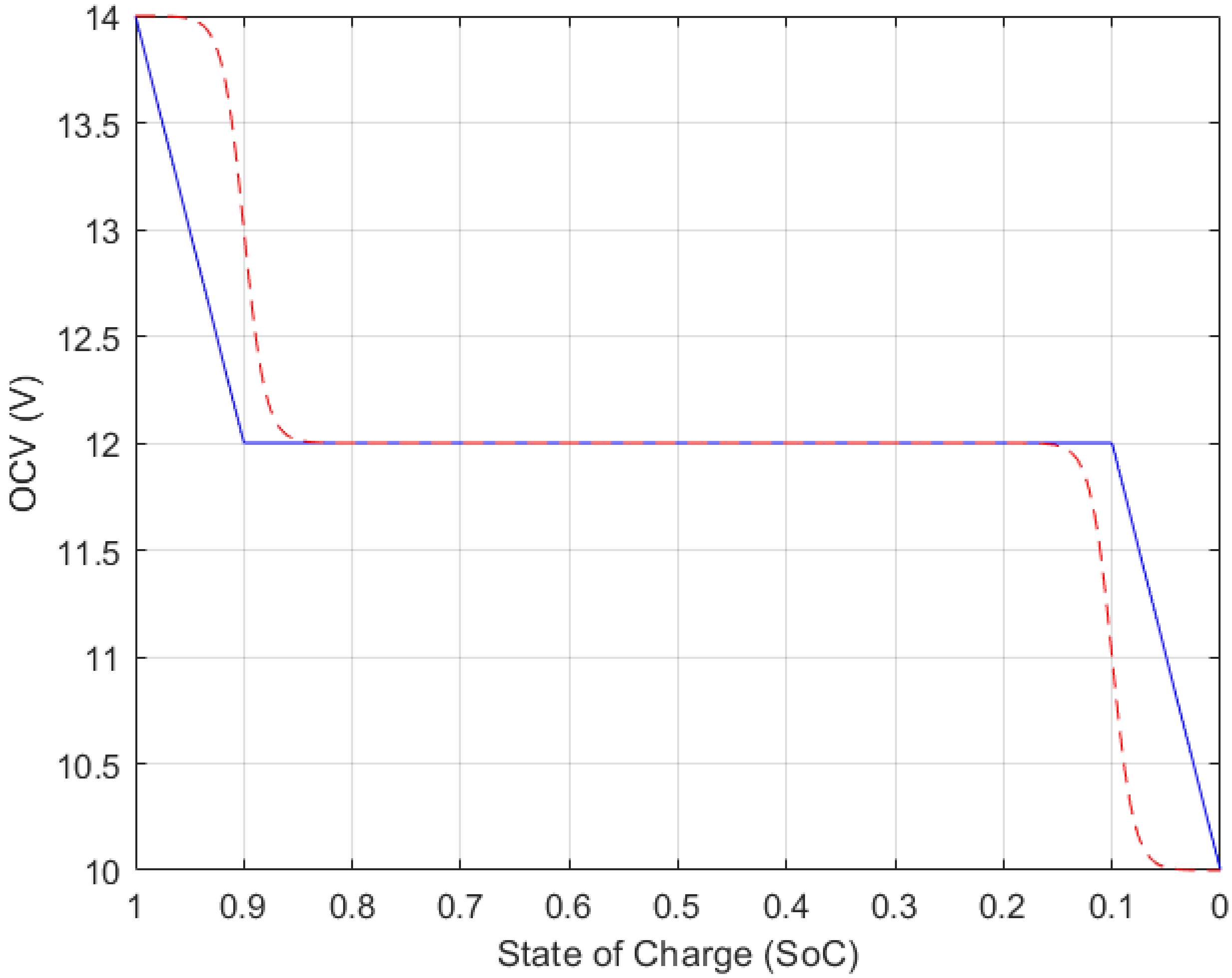
Plotting M Script

```
clc; clear; close all;

% Parameters
SoC_Vec = linspace(1, 0, 1000); % Create a vector of SoC values from 1 (full) down to 0 (empty) with
time step of 0.001
% Qn = 5;         % Nominal capacity in Ah
% SoC = 1.0;       % Initial state-of-charge (1 = fully charged)

% Compute OCV using the linear and sigmoid smoothing
ocv_linear = ocv(SoC_Vec); % Call the ocv function
ocv_sigmoid = ocv_sigmoid(SoC_Vec); % Call the ocv_sigmoid function

subplot(2,1,1);
plot(SoC_Vec, ocv_linear, 'b');
xlabel('State of Charge (SoC)');
ylabel('OCV (V)');
title('Linear OCV-SoC');
grid on;
set(gca, 'XDir', 'reverse');  % Reverse the x-axis so 1 is on the left

subplot(2,1,2);
```

```
plot(SoC_Vec, ocv_sigmoid, 'r');
xlabel('State of Charge (SoC)');
ylabel('OCV (V)');
title('Sigmoid Smoothed OCV-SoC');
grid on;
set(gca, 'XDir', 'reverse');
% gca -> get current axis. By setting the 'XDir' property to 'reverse', the x-axis is flipped so that the values decrease from left to right

figure;
plot(SoC_Vec, ocv_linear, 'b');
hold on;
plot(SoC_Vec, ocv_sigmoid, '--r');
xlabel('State of Charge (SoC)');
ylabel('OCV (V)');
grid on;
set(gca, 'XDir', 'reverse');
% gca -> get current axis. By setting the 'XDir' property to 'reverse', the x-axis is flipped so that the values decrease from left to right
```

Figures





Key Takeways

- How to reverse a graph (source: https://www.mathworks.com/matlabcentral/answers/22321-plotting-and-reversing-axis-direction)
- Sigmoid function (source: https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/)
- Matlab Anonymous function (Source: https://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html)