State University of New York at New Paltz
Division of Engineering Programs
Department of Electrical and Computer Engineering

EGE331 Computer Simulation

Final Project: Technical User's Manual of GUI

Using ODE Simulation of Dynamic Systems to Model and Demonstrate the Dynamic Behavior of Renewable Energy Systems

Yiwen Jia, Hanne Nicolaisen

May 2023

OVERVIEW

In this project, a GUI is designed to display the behavior of two dynamic systems using ODE simulations. The settings of the two systems are separated on Panel 1 and 2. Graphs of the behavior of the systems are plotted on the right panel. The first system simulates a multiple distributed energy resource islanded microgrid with AC bus. The system of ODEs used describe the dynamics of the microgrid as a linearized state-space model consisting of a wind turbine generator (WTG), photovoltaic generator (PV), fuel cell (FC), microturbine (MT), battery energy storage (BES), and loads. The energy sources that comprise the microgrid will be simulated. The user can manipulate the power change of wind or power change of solar radiation with no limit on the value they enter and can then analyze the response through eight graphs, which display the power change or total energy of the sources. The second system simulates the behavior of a circuit powered by fuel cells with the output voltage magnified by a two phase interleaved boost dc-dc converter. The ODEs of this system are developed based on Kirchhoff's current and voltage laws. The graphs of voltage magnification and the ripple currents are visualized on the GUI. Other behaviors of the system are plotted, too. And the users can view the results calculated in different methods by toggling switches next to the graphs.

Simulation 1: Islanded Multi-DER Microgrid with AC Bus

INTRODUCTION

A micro-grid is a distribution network that consists of a group of distributed generators accompanied by energy storage systems and loads. Thus, a single controllable system is formed. This system can function in both grid-connected and islanded modes [4]. By integrating cost-effective and environmentally sustainable renewable energy sources along with battery banks, microgrids offer a practical solution in providing a reliable and sustainable power supply [5]. This could in turn provide a solution to problems such as pollution, high reliability, efficiency, and other energy and electric world scenario problems [1]. How reliable is this power supply? A simplified ODE model will simulate the behavior of a multiple distributed energy resource (DERs) islanded microgrid (MG) with an AC Bus in order to consider the power of wind and solar radiation as system disturbance inputs [6].

To fully understand microgrids in depth, a few basics need to be established. First, distributed generators are a subset of distributed energy resources, which consist of renewable energy sources, such as wind turbines, solar photovoltaics, batteries, etc. It is important to note that in a micro-grid, the distributed generators possess enough power to supply most, or all of the energy needed by the connected to the micro-grid [4]. A wind turbine generator (WTG), photovoltaic generator (PV), fuel cell (FC), microturbine (MT), battery energy storage (BES), and loads are the energy sources that comprise the microgrid that will be simulated. [6].

A networked microgrid is a group of microgrids that are closely connected in terms of their electrical or spatial proximity and work together with coordinated energy management and interactive supports to one another [2] whereas an islanded microgrid operates independently from a larger power grid. An islanded microgrid is self-sufficient and able to maintain power even when disconnected from a main power grid [3]. In other words, an islanded microgrid fully functions on its own. However, with a fully islanded microgrid comes drawbacks, such as intermittence, nonlinearity, stochasticity, uncertainties and uncontrollability. A slight power imbalance can cause fluctuations in the MG's bus frequency and voltage, which could potentially lead to the collapse of the entire system. To address this power deviation

problem, energy storage (ES) systems, such as battery storage devices (BES) and flywheel energy storage devices (FES) can be utilized. Their purpose is to absorb the differences between power generation and consumption. The issue with this is that energy storage can be expensive and excessive power input/output can damage the ES [6]. This can be seen with the simulation. When the power change is of too high of a value, the total energy in the battery becomes negative. What this means is that the batteries maximum capacity has been reached and energy has been wasted. When the power change of either wind or solar radiation is of an agreeable value (generally between 0 and 1, or smaller values, under 5), the total energy in the battery will be positive, meaning that it has stored excess energy without reaching the cap. Thus, no energy is wasted.

As mentioned previously, an islanded AC microgrid composed of WTG, PV, MT, FC, BES and load only will be considered and simulated. The system will be redefined into a control system that is formed by a group of ordinary differential equations (ODEs). The purpose of the simulation is to understand how different system disturbance inputs, in this case, power change of wind and power change of solar radiation, affect the microgrid. More specifically, their effect on the battery energy storage and the AC bus frequency deviation. [6]. The simplified configuration of the islanded MG that will be simulated can be seen in Figure a below,

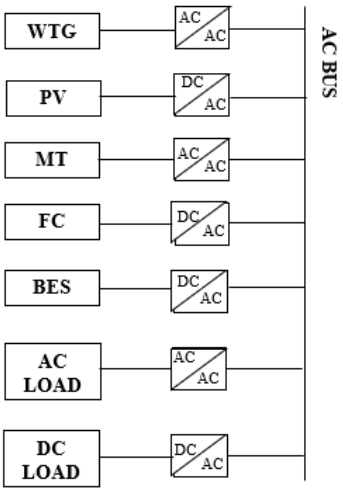


Figure a. - System configuration [6]

For simplification, only low order dynamical models will be considered. To describe the dynamics of the microgrid, the linearized state-space model of ODEs is as follows:

1. $\frac{d\,\Delta P_{WTG}(t)}{dt} = -\frac{1}{T_{WTG}}\Delta P_{WTG}(t) + \frac{1}{T_{WTG}}\Delta P_{W}$
2. $\frac{d\,\Delta P_{PV}(t)}{dt} = -\frac{1}{T_{PV}}\Delta P_{PV}(t) + \frac{1}{T_{PV}}\Delta P_{\varphi}$
3. $\frac{d\,\Delta P_{MT}(t)}{dt} = -\frac{1}{T_{MT}}\Delta P_{MT}(t) + \frac{1}{T_{MT}}u$
4. $\frac{d\,\Delta P_{FC}(t)}{dt} = -\frac{1}{T_{FC}}\Delta P_{FC}(t) + \frac{1}{T_{FC}}u$
5. $\frac{d\,\Delta P_{BES}(t)}{dt} = -\frac{1}{T_{BES}}\Delta P_{BES}(t) + \frac{1}{T_{BES}}\Delta f$
6. $\frac{d\,\Delta f}{dt} = -\frac{2D}{M}\Delta f + \frac{2}{M}\Delta P_{L}$

Where,

1. – 4. Represents the rate of change of the active power output deviation of the distributed generators (P_{WTG}(t) , ΔP_{PV}(t), ΔP_{MT} (t) ,ΔP_{FC} (t)) with respect to time.

5. Represents the rate of change of the active power output deviation of an energy storage device (ΔP_{BES} (t)) with respect to time.

6. Represents the rate of change of the frequency deviation (Δf) of the MG's bus frequency.

Here, ΔP_{W} , ΔP_{φ}represent the power change of wind and solar radiation. u represents the control input and typically, 0 < u < 1. In this case, for simplicity, u is set to 0.008. And the scalars ΔP_{WTG}, ΔP_{PV}, ΔP_{MT}, ΔP_{FC}, ΔP_{BES}, Δf represent the corresponding power output change of WTG, PV, FC, MT, BES and load. The MG power balance relationship is expressed by [6]:

$$P_{WTG} + P_{PV} + P_{FC} + P_{MT} \pm P_{BES} = P_{L}$$

Based on real-world data, the parameters of the system have been provided by [6] and have been used to simulate the MG. These values can be referenced below:

T_{WTG} = 1.5, time constant associated with the rate of change of active power output of WTG

T_{PV} = 1.8, time constant associated with the rate of change of active power output of PV

T_{MT} = 2, time constant associated with the rate of change of active power output of MT

T_{FC} = 1.5, time constant associated with the rate of change of active power output of FC

T_{BES} =0.1, time constant associated with the rate of change of active power output of BES

M = 0.2, interia constant

D = 0.012, damping coefficient associated with the rate of change of frequency deviation

The following concept will be explored: How different system disturbance inputs affect the MG system. The system disturbance inputs will be the power change of wind, and the power change of solar radiation. This will be implemented as the user will have the control to change these values and then analyze how it affects the MG system through the use of graphs. The user can freely use any range of values to analyze the effects of values . The graphs display the change in power generation of each source and the change in AC Bus frequency deviation through the use of MATLAB to solve the system of ODEs using Euler's method. Additionally, a bar graph will display the total energy transferred or used to/from each source. These values were also solved using MATLAB and a numerical method pertaining to the trapezoidal rule to find the integrals since the integration of total power gives total energy.

METHOD

To implement and simulate the response of the MG system, several steps were considered and then executed. The main numerical values that resulted from this simulation is the solved system of ODEs described above, and the integration of those values. Thus, the total power change of the difference sources and the total energy transferred/used to/by the different sources are determined. Please refer to the flow chart, Figure b., below for a more visual and condense representation of the entire implementation for the simulation of the islanded multi-DER microgrid with AC Bus:

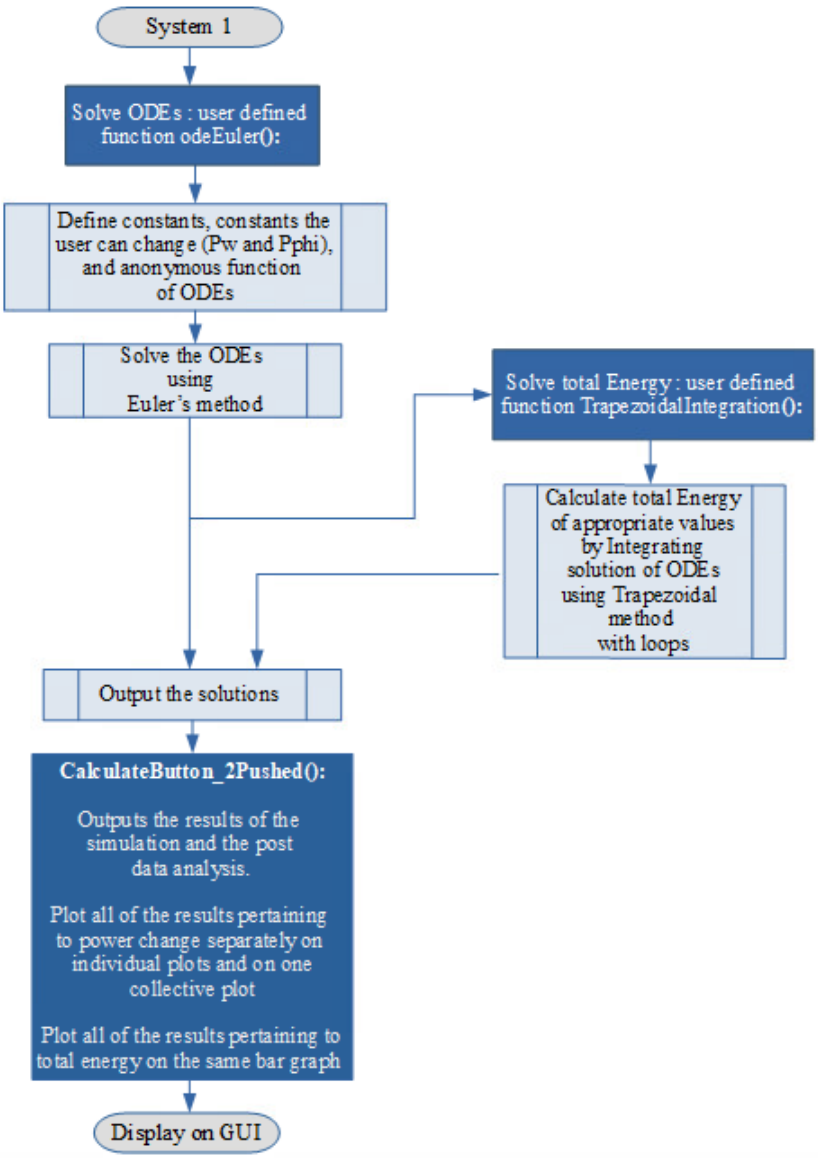


Figure b. - flowchart for implementation of MG

As Figure b. displays: first, the constants and system parameters are defined. As explained in the Introductory section, the simulation parameters were given by [6]. Please refer back to the introductory section for their definitions.

```
% Simulation Parameters
T_wtg = 1.5;
T_pv = 1.8;
T_mt = 2;
T_fc = 1.5;
T_bes = 0.1;
D = 0.012;
M = 0.2;
% Parameters unused but provided by [6]
% a = 1;
% b = 0.8;
% gamma1 = 0.02;
```

Figure c. - MATLAB Simulation Parameters (Part 2)

Next, the control input, the time span and the values that will be manipulated by the user must be defined and declared. Tspan, Initial values (P0) and *u* were selected based on trial and error to best simulate and represent the system and the changes of the system depending on the system disturbance inputs (Pw and Pphi) in a user friendly way. The chosen value for u is 0.008 and the time span, tspan, is between 0 and 10 seconds. This allows for user-friendly results with changing system disturbance values, the power change of wind and power change of solar radiation.

```
u = .008; % control input
tspan = [0, 10]; % time interval

Pw = app.PCWind.Value; % User determines power change of wind
Pphi = app.PCSolar.Value; % User determines power change of solar radiation
```

Figure d. - MATLAB Simulation Parameters (Part 1)

The initial values chosen are P0 = [0.5, 0.8, 0.2, 0.4, 4,1]', which represent the initial values for $\Delta P_{WTG}(t)$, $\Delta P_{PV}(t)$, $\Delta P_{MT}(t)$, $\Delta P_{FC}(t)$, $\Delta P_{BES}(t)$, $and\ \Delta f$, respectively. To solve the system of ODEs, the approach that was taken was to first define the system of ODEs as an anonymous function. The solved system will output as a vector of values. The anonymous function that represents the system of ODEs described in the introduction that represents the dynamics of the MG is below:

```
% System of ODEs
dpdt = @(t,P) [((-1/T_wtg)*P(1) + (1/T_wtg)*Pw);
                ((-1/T_pv)*P(2) + (1/T_pv)*Pphi);
                ((-1/T_mt)*P(3) + (1/T_mt)*u);
                ((-1/T_fc)*P(4) + (1/T_fc)*u);
                ((-1/T_bes)*P(5) + ((1/T_bes)*P(6)));
                ((-2*D/M)*P(6) -(2/M)*(P(1)+P(2)+P(3)+P(4)+P(5)+P(6)))];
```

Figure e. - MATLAB Anonymous Function for System of ODEs

odeEuler( ):

In order to solve the system of ODEs, a user-defined function called odeEuler is implemented. The main steps of odeEuler is:

1. Initialize the output solution arrays
2. Compute 'x' and 'y' using Euler's method. In other words, iterate through each 'x' value except the last one. For each iteration, find the derivative 'dydx' by calling the function with the current 'x' and 'y' values. Then, update the next column of 'y' by adding the product of 'h' and 'dydx' to the previous column of 'y'.
3. Transpose 'x' and 'y' for output

The in-code description of odeEuler and the code for odeEuler is as follows:

```
% User defined function odeEuler implements Euler's method to estimate the solution of a vector-valued ODE.
% Inputs:
%   f - a handle to a MATLAB function of the form dydx = f(x,y) that defines the ODE system, where x is a scalar and
%       both y and dydx are column vectors.
%   range - the range of independent variables over which the simulation will be performed [initial-value, final-value]
%   h - step size
%   y0 - column vector representing the initial-value(s)
% Outputs:
%   x - row vector of x-points (independent variable) in the approximate solution.
%   y - solution array. Each row in y corresponds to the solution(s) at the x-value in the corresponding element of x.
%       y(i,:) corresponds to the solution(s) at x(i)
function [x,y] = odeEuler(f,range,h,y0)

% Initialize the output solution arrays.
x = range(1):h:range(2);
y = zeros(length(y0),length(x));
y(:,1) = y0;

% Compute x and y using Euler's method.
for i = 1:length(x) - 1
    y(:, i+1) = y(:,i) + h*f(x(i),y(:,i));
end

x = x';
y = y';     % output y as a series of column vectors.

end
```

Figure f . - MATLAB User-defined Function odeEuler

In short, odeEuler takes an ODE system, a range of independent values, a step size and initial values and then applies Euler's method to estimate the solution of the ODE system and returns 'x' and 'y', which represents the approximate solution at different points.

The function call to call odeEuler with the set of ODEs applicable to the microgrid is: [t,P] = odeEuler(dpdt, tspan, .1, P0). This allows for the ODE to be solved using the user-defined function odeEuler which implements Euler's method to estimate the solution of the set of ODEs. The result is the solved set of ODEs stored in a vector P. The results are then displayed graphically using six individual graphs of each value, and then one collective graph where all of the results can be seen

POST DATA ANALYSIS

TrapezoidalIntegration( ):

In order to establish a numerical value of interest, integration is considered. This is because the integral of power is energy. Thus, the total energy of each source can be determined and evaluated. This is fairly straightforward. The integral is calculated using a user-defined function called TrapezoidalIntegration, which takes an anonymous function, lower and upper limits of integration, and the number of subintervals as inputs. The main steps of TrapezoidalIntegration are as follows:

1. Extract the lower and upper limits of integration
2. Calculate the width of each subinterval ('h') based on the number of subintervals
3. Compute the value of the function at the lower and upper limits and store it in 'Trap'
4. Iterate through each subinterval (compute the value of the function at the intermediate points and add it to the 'TrapSum'
5. Compute the numerical integral using the Trapezoidal rule: NumIntegral = h*(Trap+TrapSum)
6. Return the computed numerical integration

The in-code description of TrapezoidalIntegration and the code for TrapezoidalIntegration is as follows:

```
% User defined function TrapezoidalIntegration computes the trapezoidal rule using an anonymous function
%       Inputs: - ftn: the function handle to the anonymous function
%               - limits: the limits of integration (a vector of two terms)
%               - subIntervals: the number of subintervals to use in the integration
%       Outputs: - NumIntegral: the numerical integratl of the function over the limits provided using the
%                  trapezoidal rule as a scalar value
function [NumIntegral] = TrapezoidalIntegration(ftn,limits,subIntervals)

lower =  limits(:,1)+1; % lower limit
upper = limits(:,2); % upper limit

%fplot(ftn,[lower,upper]); title('plot'); xlabel('x'),ylabel('ftn(x)') % plot of function over range specified by limits
%hold on; plot([lower,upper],[ftn(lower),ftn(upper)]); hold off

h = (upper-lower)/subIntervals; % delta x = h, the width of each subinterval
Trap = (ftn(lower) + ftn(upper))/2; % first and last term

TrapSum = 0; % Initialize the summation
for i = 1:subIntervals-1 % initialize the index variable, increment by 1 until subintervals-1
    TrapSum = TrapSum + ftn(lower+i*h); % take initial value and adding the value of the function xi (applying xi directly into ftn call)
end

NumIntegral = h*(Trap + TrapSum); % summing the two terms together and multiplying by h

end
```

Figure g. - MATLAB User-defined Function TrapezoidalIntegration

The function is then called for each value of interest with the following function call, which is essentially every value aside from the frequency deviation:

- EnergyofWTP = TrapezoidalIntergration(P(:,1), tspan, 0.1);
- EnergyofPV = TrapezoidalIntergration(P(:,2), tspan, 0.1);
- EnergyofMT = TrapezoidalIntergration(P(:,3), tspan, 0.1);
- EnergyofFC = TrapezoidalIntergration(P(:,4), tspan, 0.1);
- EnergyofBES = TrapezoidalIntergration(P(:,5), tspan, 0.1);

Thus, the energy transferred/used by WTP, PV, MT, FC, and BES are found. Last, the values are all plotted together as a bar graph, displaying the different values for total energy of each source. As stated previously, if the result of BES is a positive value, then the amount of energy supplied to the battery is greater than the energy drawn from it and the excess energy is stored in the battery. If the energy for BES is negative, then the battery's maximum capacity has been reached and the excess energy is wasted.

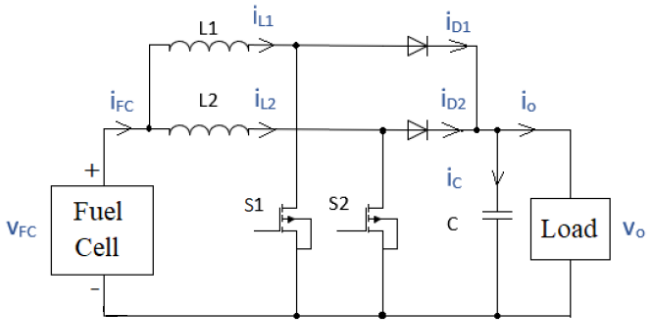
Simulation 2: Two Phase Interleaved Boost DC – DC Converter for Fuel Cell Applications

INTRODUCTION

Polymer electrolyte membrane (PEM) fuel cells are a type of renewable clean energy that generates electricity by catalyzing hydrogen to protons and electrons with PEM at anode, then react with oxygen at cathode. The reaction is expressed as $H_2 + O_2 \rightarrow H_2O$ + Heat + Electrical Energy. [7] The reaction shows that no air pollutants or CO_2 are produced, which is environment friendly and does not exacerbate global warming. Fuel cells are light-weighted, and their operation temperature is low, which reduces wear on system components. [8] They are used in electrical vehicles.

While a stack of fuel cells provide low DC voltages, a boost DC-DC converter is commonly used to raise the fuel cell voltage high enough for the load. In addition, a boost DC-DC converter reduces current ripples which, if high, increase the fuel consumption, shorten the lifespan of the fuel cells and cause overload. [9] In this project, a two phase interleaved boost dc-dc converter is simulated to demonstrate the magnification of voltage from the fuel cells.

The circuit of the system and the ODEs are shown below. The ODEs are formed based on Kirchhoff's current and voltage laws. In the ODEs [10], R's, L's and C's are the resistance and inductance



a. The circuit with a 2-phase interleaved boost dc-dc converter

$$L_1 \frac{d\, i_{L1}(t)}{dt} + R_{L1}\, i_{L1}(t) = v_{FC}(t) - [1 - q_1(t)]\, v_o(t)$$

$$L_2 \frac{d\, i_{L2}(t)}{dt} + R_{L2}\, i_{L2}(t) = v_{FC}(t) - [1 - q_2(t)]\, v_o(t)$$

$$\frac{d\, i_{FC}(t)}{dt} = \frac{d\, i_{L1}(t)}{dt} + \frac{d\, i_{L2}(t)}{dt}$$

$$C \frac{d\, v_o(t)}{dt} = i_{D1}(t) + i_{D2}(t) - i_o(t)$$

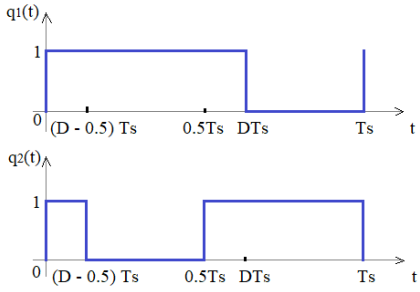
$$= [1 - q_1(t)]\, i_{L1}(t) + [1 - q_2(t)]\, i_{L2}(t) - i_o(t)$$

b. The ODEs of the circuit.

Figure 1

of the inductors and capacitance of the capacitor. The v's and i's are the voltages and currents marked in Figure 1a. The q1(t) and q2(t) are the switching functions of S1 and S2. When S is closed, q(t) is 1 and i_D

is 0. Otherwise, q(t) is 0 and i_D is i_L. The switch frequencies are controlled by a microcontroller. The values of q(t) and i_D at different time intervals in one operation cycle, T_s, are shown in Figure 2. The symbol D in Figure 2 is duty ratio.



a. Switching functions of S1 and S2 within one cycle when D > 0.5Ts

$i_{D1}(t)$	$q_1(t)$	$t \in$	$q_2(t)$	$i_{D2}(t)$
0	1	(0, (D - 0.5)Ts)	1	0
0	1	((D - 0.5)Ts, 0.5Ts)	0	i_{L2}
0	1	(0.5Ts, DTs)	1	0
i_{L1}	0	(DTs, Ts)	1	0

b. The values of iD1, iD2, q1 and q2 in different time intervals within a cycle.

Figure 2

The voltage $V_{FC\_singleCell}$ of each fuel cell can be calculated by $E_{Nernst} - \Delta V_{act} - \Delta V_{Ohm} - \Delta V_{trans}$ where E_{Nernst} is the Nernst voltage that can be approximated as a function of the temperature of the fuel cells and the partial pressure of hydrogen and oxygen. ΔV_{act} is the voltage drop at the electrodes, which is calculated by the Tafel equation. ΔV_{Ohm} is the ohmic loss due to the electrical resistance of the electrode. And ΔV_{trans} is the voltage drop due to mass transportation loss or concentration loss. When the stack voltage is too low for the controller's working range, ΔV_{trans} is removed from the simulation. [10] The formulas to calculate each term is shown below.

$$V_{FC\_singleCell} = E_{Nernst} - \Delta V_{act} - \Delta V_{Ohm}$$

$$E_{Nernst} = 1.482 - 0.00485T - 0.0000431T \ln(P_{H2}P_{O2})$$

$$\Delta V_{act} = A \ln((i_{FC} + i_n) / i_0) \text{ (where A is Tafel slope.)}$$

$$\Delta V_{Ohm} = r_{ohm}\, i_{FC}$$

$$V_{FC} = N_{cell}\, V_{FC\_singleCell}$$

The values of the parameters in all the equations above are shown below. The maximum output power is 1000W. [10] The term $i_o(t)$ is calculated by loadPower/$v_o(t)$ where loadPower represents the output power. This value is entered by users on GUI to mimic different load sizes in the applications of the fuel cells.

N_{cell} = 72, number of fuel cells

T = 338.15K, stack temperature

P_{H2} = 1.494atm, partial pressure of H_2

P_{O2} = 0.21atm, partial pressure of O_2

A = 0.055V, Tafel slope

$i_0 = 1.5x10^{-3}$A, exchange current

i_n = 0.13A, internal current

r_{ohm} = 0.006Ω, ohmic resistance

$L_1 = L_2$ = 0.68mH, inductance

$R_{L1} = R_{L2}$ = 0.1Ω, inductor resistance

C_o = 330μF, output capacitance

$T_s = 4x10^{-5}$s, switching period

D = 0.6, duty ratio

This simulation employs Euler's method and MATLAB built-in function ode45() to solve the ODEs. The simulated data derived by ode45() are used to calculate the maximum current ripple and the energy generated by the fuel cells, using forward and backward difference methods and the MATLAB built-in function trapz() respectively. These two results are displayed on Panel 2 on the GUI. The user inputs are the loadPower, duty cycle and the ending time with respective ranges provided on Panel 2. Details will be introduced in the GUI introduction part.

METHODS

The implementation of this system is shown in Figure 3. There are two main parts: user defined functions (the upper dard blue boxes) and callback functions (the lower dard blue boxes). The two main user defined functions are ODESolver() and maxCurrentRipple(), for the purpose of solving the ODEs and calculating the maximum current ripple respectively. The outputs of the two functions are processed in the callback functions and some of the data are displayed on the GUI.

ODESolver( ):

According to the ODEs, the independent variable is the time t while the dependent variables are i_{L1}, i_{L2} and v_o. All the variables are solved using Euler's method and the MATLAB built-in function ode45(). The procedures are shown in the left subroutines in Figure 3 and the codes are in the function ODESolver() in the Appendix.

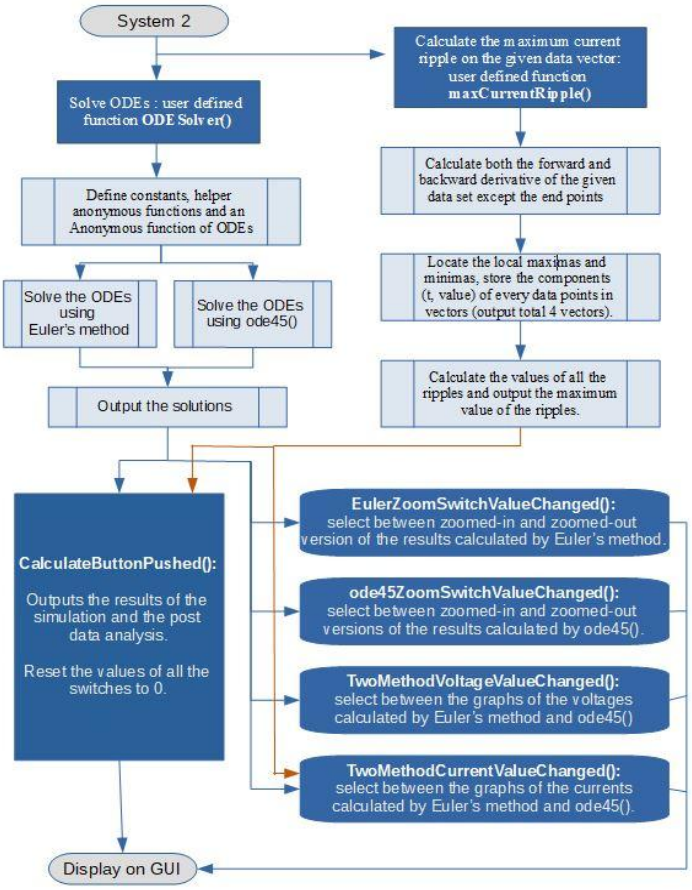


Figure 3

The ODEs are expressed with an anonymous function d_depV = @(t, depV) [expression1; expression2; expression3] where depV = [i_{L1}; i_{L2}; v_o] is a 3-by-n matrix of dependent variables. And n is the number of values of each dependent variable simulated and is determined by the number of time-points the programmer defines. The three expressions correspond to di_{L1}/dt, di_{L2}/dt and dv_o/dt respectively. Each dependent variable is expressed as an indexed element in depV with the form depV(index) in each expression. With n time-points, this anonymous function returns a 3-by-n matrix with each column representing the derivative of the three dependent variables at each time-point.

Due to the complexity of the three expressions, the constants and helper anonymous functions are defined in the beginning, as shown in the code. The help functions are defined as anonymous functions with independent variables being or can be expressed by the independent or dependent variables of the system.

When given a set of ODEs, each ODE shows that the derivative of a dependent variable at every t = t_i can be calculated using the known values of the dependent variables at t = t_i. Therefore, with the initial values of all the dependent variables given and with $\Delta t = h$ being defined by the programmer, the values of all the dependent variables can be approximated numerically one by one using the formula $y_{i+1} = y_i$ + (slope of the tangent line at t = t_i) x h. This process is realized by defining an anonymous function of the ODEs, providing initial values of dependent variables, and calculating every next values of all the dependent variables in the for loop in the code. Due to the way the vectors are defined, each column of depV is calculated in one iteration of the loop. The (i+1)th column of depV is calculated based on the values in the ith column of depV and the values of d_depV that are derived from the ith column of depV. The loop will execute n-1 times so that all the n values of each dependent variable are approximated (recall that n means the number of values of each dependent variable simulated).

One parameter, io, in the ODEs is calculated by loadPower/vo. Since the initial value of vo is 0, io is programmed to be 0 in the first iteration. Otherwise, errors would occur. As vo increases, io is defined as loadPower/vo. This is the conditional part in the for loop in ODESolver(). Due to the same reason, the set of results calculated by Euler's method with the first nonzero vo is used as the initial condition for ode45().

The ODEs are solved by the MATLAB built-in function ode45(), which is based on an Runge-Kutta formula, for reference. The syntax is shown in the code in the appendix. The integrated results calculated by both ways are imaginary numbers. Thus, the results are processed by a helper function imagToReal() to become the magnitude of the imaginary numbers with the signs unchanged.

The function ODESolver() outputs 12 vectors t, iL1, iL2, vo and iFC, vFC (the current and voltage generated from the fuel cells) calculated in both ways. These values are used in the callback functions and for post data analysis.

Callback functions:

The results calculated by the function ODESolver() are used in every callback function. The function CalculateButtonPushed() displays all the outputs on Panel 2 and on the last two tabs on the right

panel, FC - Simulation Result 1 and FC – Simulation Result 2. The rest of the callback functions are coded to graph charts on the two tabs. Each of the functions dictates which chart on the respective location to graph while the switch next to the chart is toggled to different state values. The state values of all the switches are programed to 0 at the end of the function CalculateButtonPushed() because this function plots the images corresponding to the state values of all the switches being 0. Without this step, the chart(s) may not match the switch(es) when a user pressed the Calculate button before toggling the switch(es) back to 0.

POST DATA ANALYSIS

Post data analysis is coded in the callback function CalculateButtonPushed(). Two values, the energy generated by the fuel cells and the maximum current ripple that goes through the fuel cells Δi_{FC}, are calculated from the tabulated data derived from the ode45() in the function ODESolver(). This selection of data is because the results calculated by ode45() are more accurate than the ones calculated by Euler's method.

The energy generated by the fuel cells is calculated by performing element-by-element multiplication on the vectors vFC_45 and iFC_45, and integrating the resulting vector with the MATLAB built-in function trapz() over the entire time interval t_45.

maxCurrentRipple( ):

The maximum current ripple (ΔiFC) is calculated in a user defined function called maxCurrentRipple(), as shown at the top right quarter of the flowchart. It is calculated by finding the maximum value among all the differences between adjacent local maximas and minimas within 1000 data points after t > 0.008s because the system is relatively stable during this time interval. Since the graph of the current is in a zigzag shape, the local maximas and minimas cannot be derived by differentiating the dataset and collecting the zeros of the derivatives. Instead, every maxima and minima are calculated by performing both forward and backward differentiation from the second data through the second last data. Local maximas are the points where their forward derivatives are negative and

backward derivatives are positive. Local minimas are the points where their forward derivatives are positive and backward derivatives are negative. As shown in the code in the Appendix, four vectors are derived to carry the information of the local maximas and minimas of the current iFC. They are called t_max, maxes, t_min, and mins. The current ripples can be calculated by direct-subtracting and shift-subtracting the two vectors of local maximas and minimas. Performing two subtracting methods ensures that the current ripples between every maximas and the minimas on their right and left are both included. The results of direct-subtracting and shift-subtracting are shown in the four scenarios in Figure 4. The magenta circles the local maximas while the green circles the local minimas. The direct-subtracting results in the differences between the points connected by the blue segments. While the shift-subtracting results in the differences between the points connected by the orange segments. They are calculated in different ways as shown in Figure 4. The largest value among all the differences derived by both direct- and shift-subtraction is the maximum current ripple, ΔiFC.

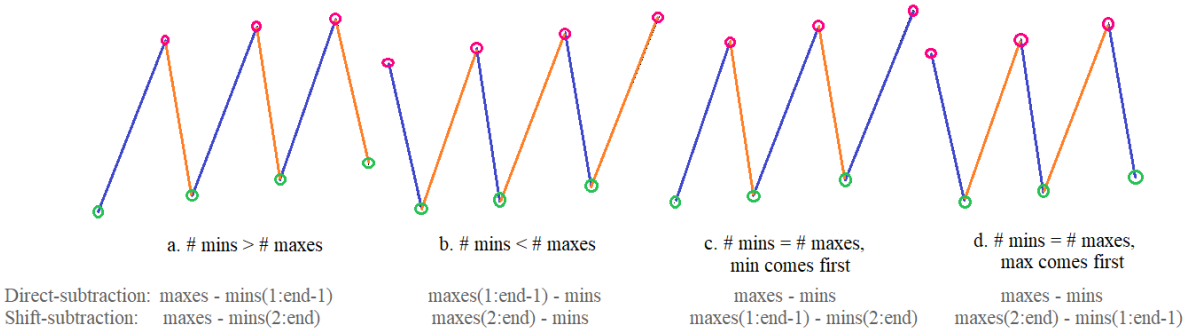


Figure 4. Four scenarios of local maximas and minimas.

GUI INSTRUCTIONS AND OUTPUTS

Simulation 1: Islanded Multi-DER Microgrid with AC Bus

The basic use of the GUI pertaining to Panel 1: Simulation Islanded Multi-DER Microgrid with AC Bus is fairly straightforward. The user can manipulate the system disturbance inputs. In other words, the user can manipulate the power change of wind and the power change of solar radiation (A) and analyze the response of the islanded multi-DER MG system with an AC Bus through the use of graphs. Please refer to the image below to see what values the user can manipulate. The specific values that can be altered are adjacent to a. Wind and b. Solar radiation under Panel 1 (Panel 1 is the first panel, above Panel 2 and to the left of the graphs).. There is no limitation on the range of values the user can use to study and analyze the response of the system.

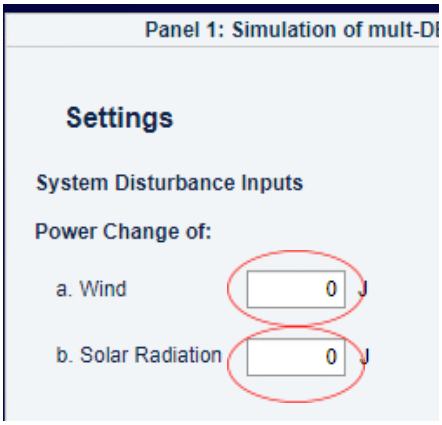


Figure I. - GUI Panel 1 - User Inputs

There are a total of 8 graphs pertaining to the MG simulation. MG-Simulation 1(E) displays a graph of the ODE system solved on the graph “Total Power Generation/Consumption.” This graph displays the total power generation/consumption of each source, the WTG, PV, FC, MT, and BES along with the AC frequency Deviation. Underneath this graph is “Total Energy Transferred/Used”, which displays the total amount of energy transferred or used by each source. In addition to this, on Panel1, there is a dropdown selection if the user is ever confused with the results simulated and displayed by bar graph. To use this, if the case occurs, click on the dropdown and an explanation will display (C):

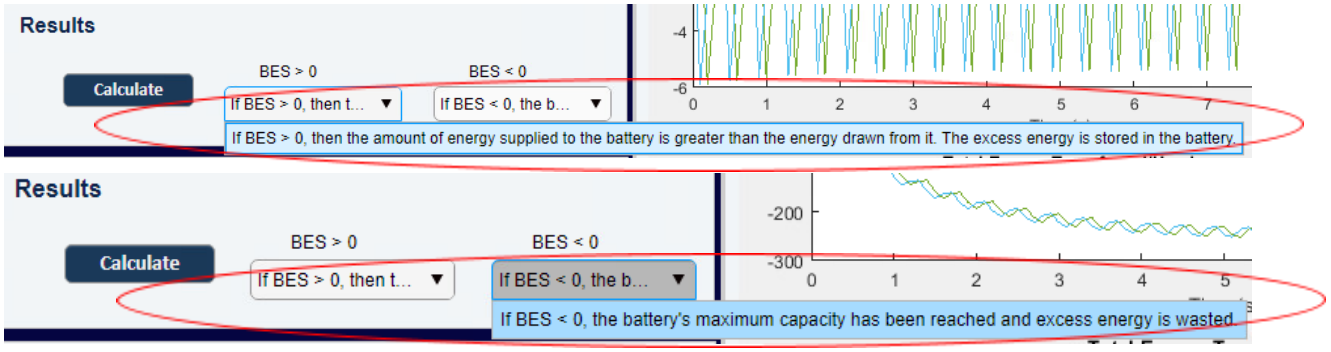


Figure II - GUI Panel 1 - Explanation of Energy Results for BES

There are legends in the upper right hand side of each graph to identify which bar graph or function pertains to which. On MG - Simulation 2 tab (F) , each source and the AC bus frequency deviation as a solution to the solved system of ODEs are displayed on their own individual graph for closer evaluation. If the user is ever unsure of what each abbreviation means, they can refer to the Abbreviations Guide on Panel 1 (B). To run the Simulation, enter the values in (A) and press Calculate (D).

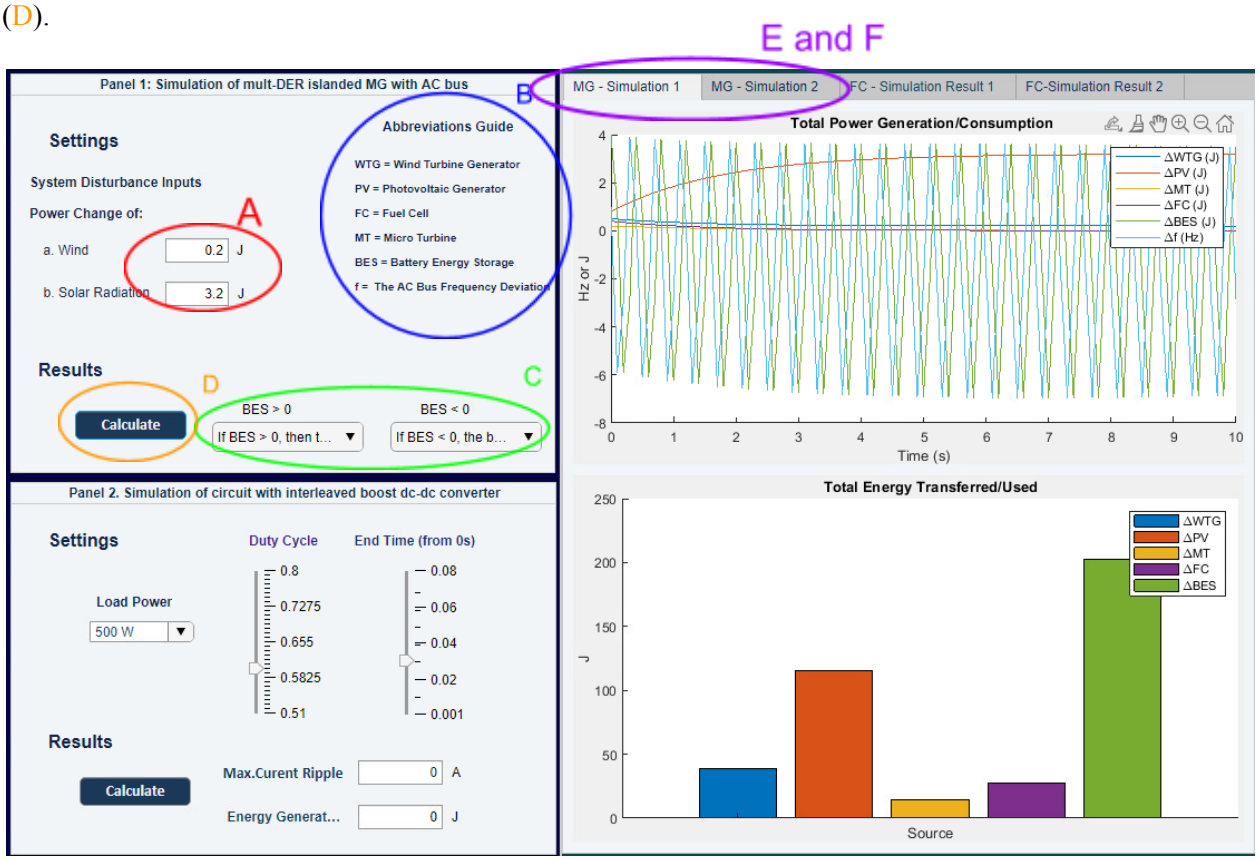
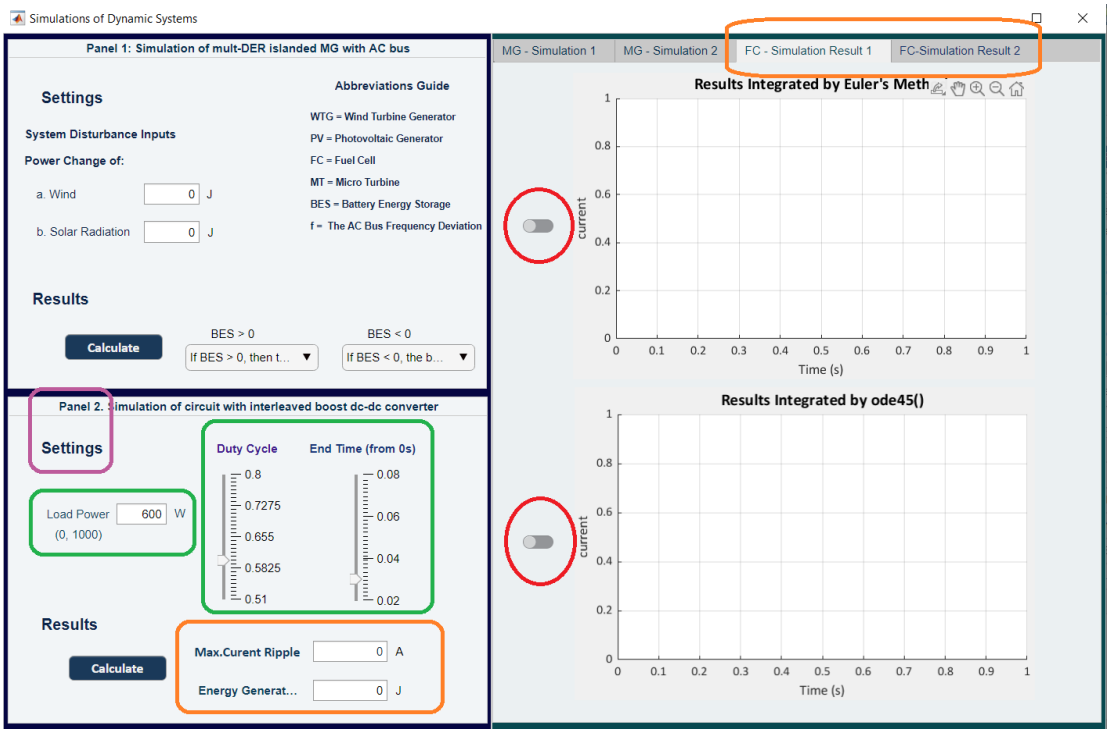


Figure III. - GUI Panel 1 - Complete User Guide

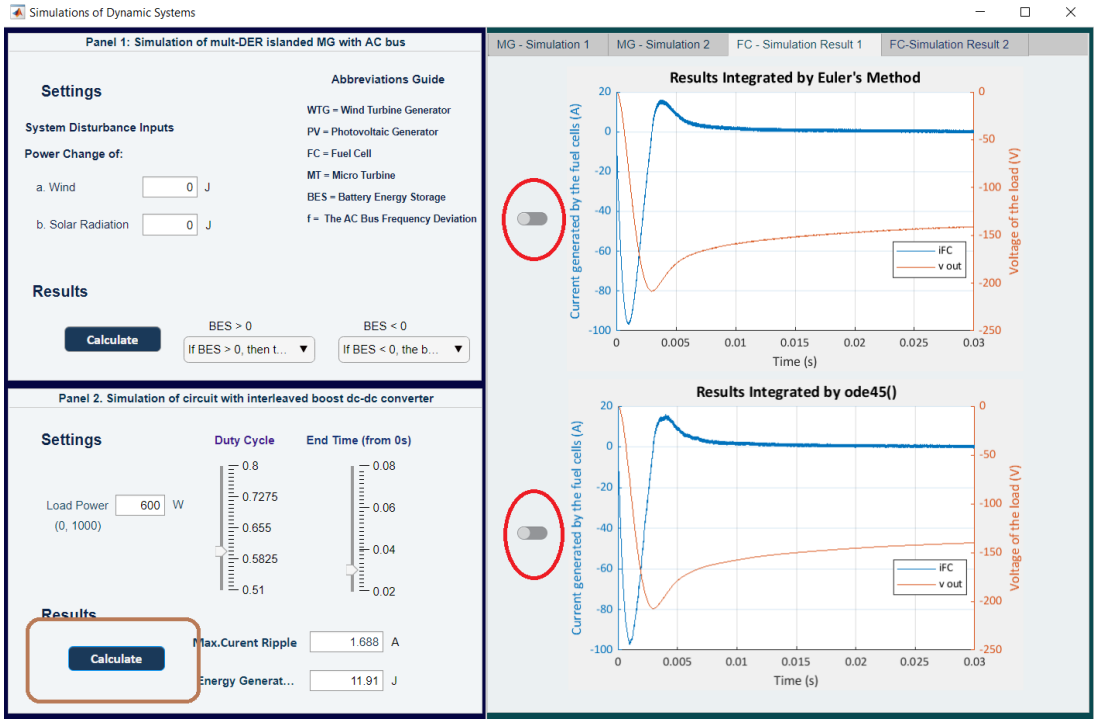
Simulation 2: Two Phase Interleaved Boost DC – DC Converter for Fuel Cell Applications

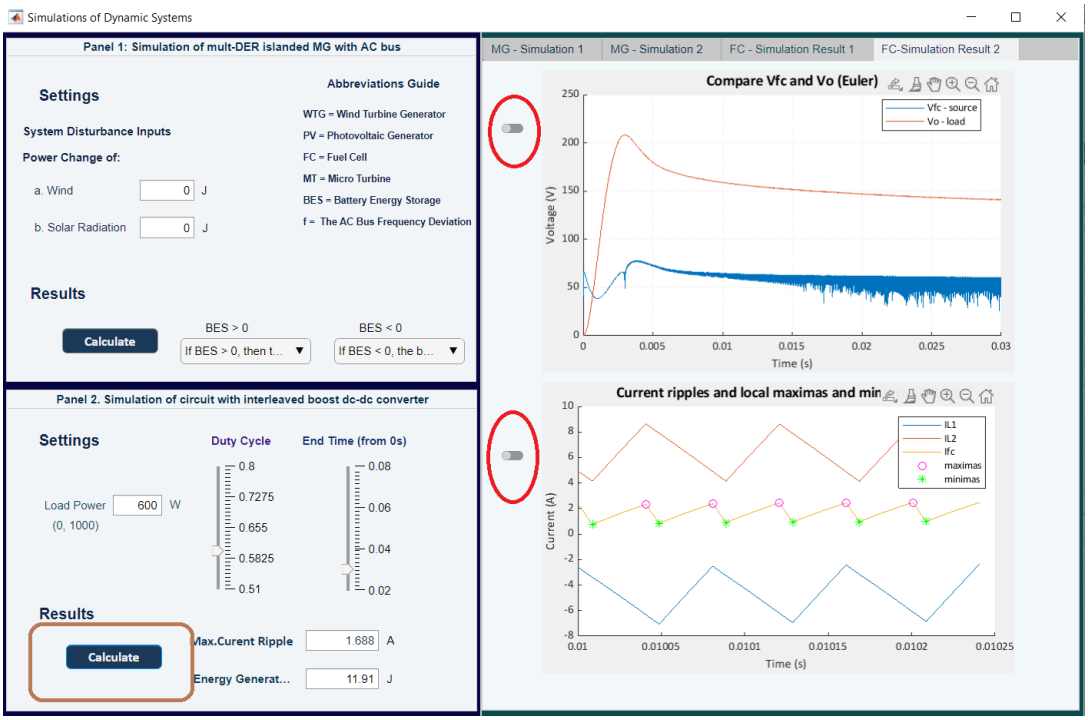
The settings of this simulation is on Panel 2 on the GUI (purple). There are three settings (green): Load Power - user input numeric values between 0 to 1000 Watts; Duty Cycle - a slider between 0.51 and 0.8; And End Time (from 0s) - a slider between 0.02 - 0.08 s. The numeric outputs are displayed at the bottom of Panel 2 (orange), and the graphic outputs are displayed on the tabs FC - Simulation Result 1 and FC - Simulation Result 2 on the right panel (orange). The numerical output Max. Current Ripple is calculated with the 1000 data points after t > 0.008s derived by ode45() due to the inaccuracy of the data calculated by Euler's method. The other numerical output is calculated over the entire time interval determined by the right slider in Panel 2. Pressing the button "Calculate" triggers the function call of CalculateButtonPushed().



The simulation starts after the "Calculate" button is pressed (brown). As shown in the following two pictures, the results of the post data analysis are shown on Panel 2. They are calculated based on settings. On the tab FC - Simulation Result 1, the 1st chart displays the current through the fuel cells and the voltage over the load calculated by Euler's method. The 2nd chart displays the same results calculated

by ode45(). On the tab FC - Simulation Result 2, the 1st chart displays the voltage output from the fuel cells and across the load calculated by Euler's method. The 2nd chart displays 300 data points of currents flowing through the fuel cells (Ifc), and the two inductors (IL1 and IL2) during the stable state. In addition, the local maximas and minimas are marked by magenta circles and green asterisks respectively. These results are also calculated by Euler's method. After the "Calculate" button is pressed, all the switches (red) are forced to toggle to the left (state value = 0).

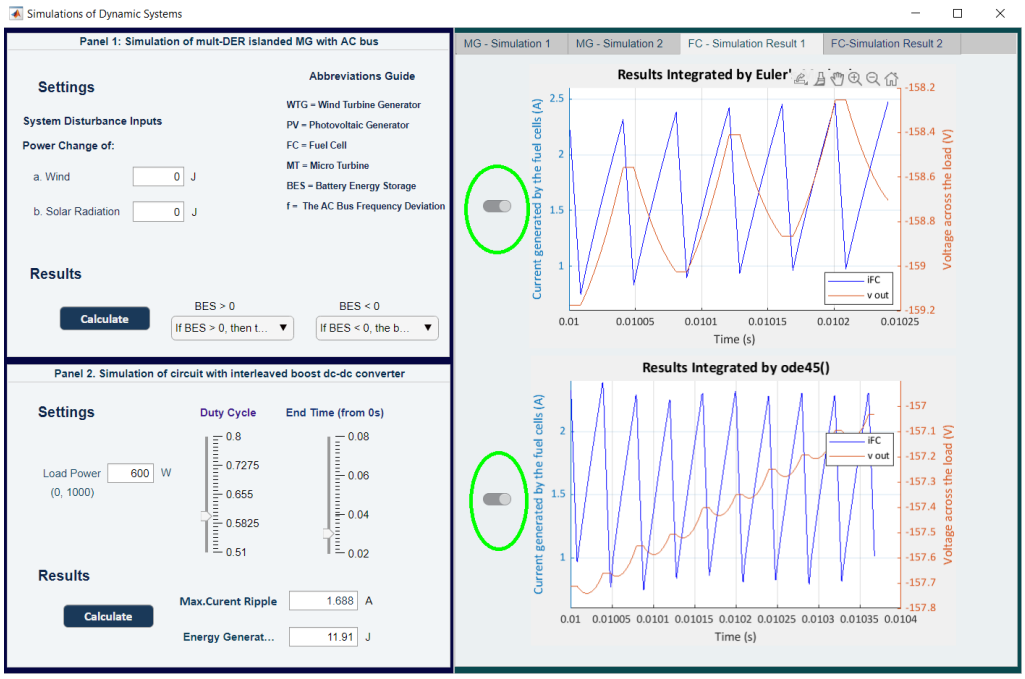


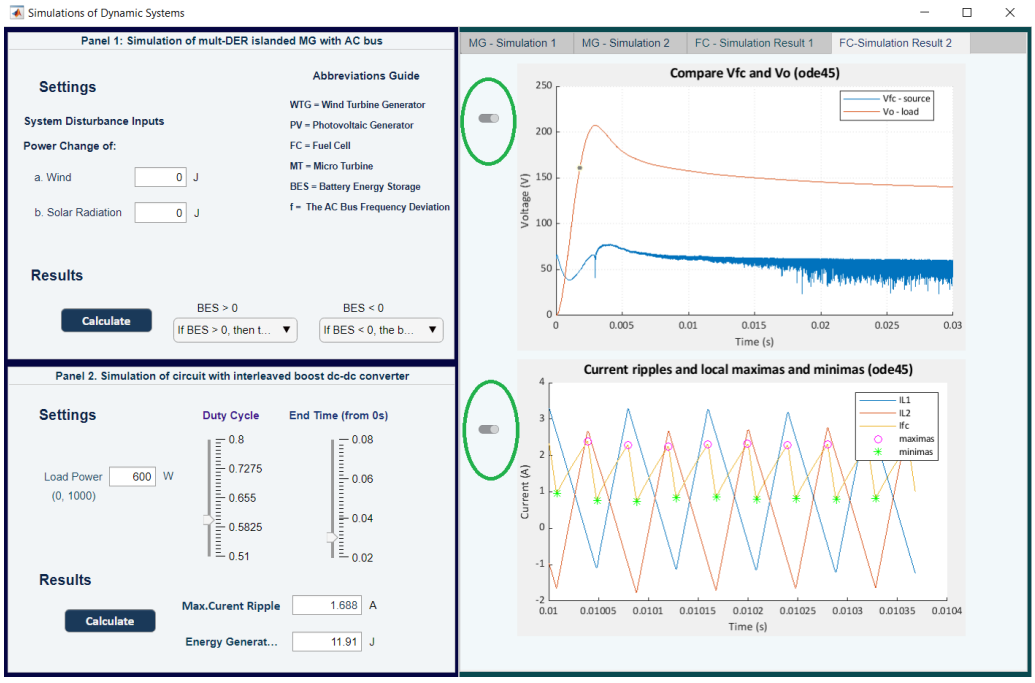


The following two pictures demonstrate the functions of the four switches (green). When the upper switch on the tab FC - Simulation Result 1 is toggled to the right (state value = 1), the chart displays the zoomed-in version of the same chart at that location. If it's toggled to the left again, the zoomed-out chart is displayed (the same chart displayed after the "Calculated" button is pressed). The lower switch functions the same way: when toggled to the right, the chart displays the zoomed-in version of the original chart at that location. When toggled to the left again, the zoomed-out version is shown. The zoomed-in version of the two charts demonstrate the 300 data points after 0.01s calculated in different ways. The upper and lower buttons trigger the callback functions EulerZoomSwitchValueChanged() and ode45ZoomSwitchValueChanged(), respectively.

On the tab FC - Simulation Result 2, the upper/lower switch toggles between the voltages/currents calculated by Euler's method (state value = 0) and by ode45() (state value = 1). They trigger the callback functions TwoMethodVoltageValueChanged() and TwoMethodVoltageValueChanged(), respectively. As mentioned before, if the user presses the "Calculate" button before any of the switches are toggled to the left, all the switches are forced to the left

after the execution of the function CalculateButtonPushed(), as shown in the previous pictures when all the switches are marked red.





EVALUATION OF GUI

The GUI designed in this project is user friendly. The input and output sections are indicated clearly. The purpose of the GUI is to display the behavior of two dynamic systems in response to the user inputs. Specifically, in the first system, the primary goal is to simulate an islanded multi-DER Microgrid with AC Bus and analyze the effects of different system disturbance inputs, power change of wind and power change of solar radiation, on the system. The user can freely manipulate these two values. The simulation generates eight graphs, including six individual graphs: five of the six displays power changes for each source while one displays the system controlled output (AC bus frequency deviation). One graph contains all of these values collectively, displaying the total power changes for all sources and frequency deviation while another graph displays the total energy transferred or used by each source. The GUI provides information on energy storage and limitations based on power changes, helping users determine the need for additional storage sources or identify system peak points.

The simulation utilized numerical techniques such as Euler's method and the Trapezoidal rule for integration. The numerical method techniques developed in class, such as odeEuler and TrapezoidalIntegration, were successfully used and implemented in the MG simulation as user-defined functions. Additionally, the correct use of anonymous functions with initial value ensured that the previous user-defined methods could be successfully implemented. The simulated results are valid. For example, when the power change of either wind or solar radiation is of a large value, then the change in power of the correlating source (WTG for wind and PV for solar radiation), increases and has more total energy, while the energy source, BES, is maxed out and energy is wasted, as seen by the negative total energy for BES.. However, if the total power change for both is under about 30 J, then the energy is successfully stored in BES, as seen by the positive total energy for BES..

While in the second system, the primary goal is to demonstrate the current ripples of the circuit and the magnification of the voltage by the interleaved boost dc-dc converter. The numerical methods adopted in the simulation works well in the post data analysis. However, in solving the ODEs, both Euler's method and ode45() generated imaginary numbers with real initial values and no sqrt() operation

involved. There is a big discrepancy between the currents through the two inductors calculated by two methods. The graph of the currents calculated by ode45() seems more reasonable than the one by Euler's method. This could be caused by the imaginary results. However, although possibly wrong, the rest of the results calculated in both methods are similar, as shown in the graphs after the Calculate button is pressed. What's more, the graphs did show the magnification of the voltage and the ripples of the currents and the voltage. The numerical outputs also show correct changes as the inputs change.

In conclusion, the MG simulation does fulfill its purpose, which is: to help the user analyze the effects of different disturbance inputs on the relevant MG system and to demonstrate the current ripples of a circuit and the magnification of voltage by an interleaved boost dc-dc converter. Additionally, the user interface is clear, labeled, straightforward and includes units. However, there is room for improvement. A time slider for zooming in on specific values could have been incorporated, along with allowing the user to define initial values for the MG system. As for the second system, the generated imaginary values with real initial values could be further explored and potentially fixed or properly explained.

RESOURCES

[1] Bonfiglio, A.; Brignone, M.; Invernizzi, M.; Labella, A.; Mestriner, D.; Procopio, R. A Simplified Microgrid Model for the Validation of Islanded Control Logics. Energies 2017, 10, 1141. https://doi.org/10.3390/en10081141

[2] Narula-Tam, A., Aminifar, F., & Roop, J. (2017). Networked microgrid stability through distributed formal analysis. 2017 IEEE Power & Energy Society General Meeting, 1-5. doi: 10.1109/PESGM.2017.8273926 Available: https://www.eversource.uconn.edu/wp-content/uploads/2019/02/Networked-microgrid-stability-through-distributed-formal-analysis.pdf

[3] M. Jain, S. Gupta, D. Masand, G. Agnihotri and S. Jain, "Real-Time Implementation of Islanded Microgrid for Remote Areas," in Journal of Control Science and Engineering, vol. 2016, Article ID 5710950, pp. 1-9, 2016, doi: 10.1155/2016/5710950. Available: https://www.hindawi.com/journals/jcse/2016/5710950/

[4] A. K. Singh, P. C. Pandey, R. Singh, and A. Singh, "Design and implementation of a microgrid system for rural electrification," Int. J. Innov. Res. Sci. Eng. Technol., vol. 4, no. 7, pp. 5735-5741, Jul. 2015. Available: http://www.ijirset.com/upload/2015/july/30_Design.pdf.

[5] MTU Onsite Energy. (2019). Microgrids for reliable, efficient and sustainable energy supply. MTU Report, 02/2019, 1-8. Available: https://www.mtu-solutions.com/content/dam/mtu/download/technical-articles/21162_Microgrids_TA.pdf

[6] H. Hua, Y. Qin, and J. Cao, "A Class of Optimal and Robust Controller Design for Islanded Microgrid," Research Institute of Information Technology, Tsinghua University, Beijing, P.R. China, 2023. Available: http://web.mit.edu/~caoj/www/pub/doc/jcao_c_mg.pdf

[7] P. Sarakarn, W. Subsingha, “4 Phase Interleaved DC Boost Converter for PEMFC Applications,” Procedia Engineering, Vol. 32, pp. 1127-1134, 2012, doi: 10.1016/j.proeng.2012.02.066.

[8] Hydrogen and Fuel Cell Technologies Office. “Types of Fuel Cells.” ENERGY.gov https://www.energy.gov/eere/fuelcells/types-fuel-cells (Accessed Apr. 8, 2023).

[9] A. Badji, N. Benamrouche, N. Benyahia, H. Denoun, D. Rekioua, T. Rekioua, M. Zaouia “MPPT controller for an interleaved boost dc–dc converter used in fuel cell electric vehicles,” International Journal of Hydrogen Energy, vol. 39, no. 27, pp. 15196-15205, 2014, doi: 10.1016/j.ijhydene.2014.03.185.

[10] C. Sirisamphanwong, S. Somkun, S. Sukchai, “A DSP-based interleaved boost DC-DC converter for fuel cell applications,” International Journal of Hydrogen Energy, vol. 40, no. 19, pp. 6391-6404, 2015, doi: 10.1016/j.ijhydene.2015.03.069.

APPENDIX

```
classdef CompSim_FinalProjectGUI_Combo < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
    FinalProjectUIFigure     matlab.ui.Figure
    GridLayout               matlab.ui.container.GridLayout
    LeftPanel                matlab.ui.container.Panel
    Panel2                   matlab.ui.container.Panel
    ALabel                   matlab.ui.control.Label
    JLabel                   matlab.ui.control.Label
    CalculateButton          matlab.ui.control.Button
    MaxCurentRippleLabel             matlab.ui.control.Label
    MaxCurentRippleEditField  matlab.ui.control.NumericEditField
    EnergyGeneratedbytheFuelCellsEditFieldLabel  matlab.ui.control.Label
    EnergyGeneratedbytheFuelCellsEditField  matlab.ui.control.NumericEditField
    DutyCycleSliderLabel    matlab.ui.control.Label
    DutyCycleSlider          matlab.ui.control.Slider
    SettingsLabel            matlab.ui.control.Label
    EndTimefrom0sLabel    matlab.ui.control.Label
    EndTimefrom0sSlider    matlab.ui.control.Slider
    ResultsLabel             matlab.ui.control.Label
    LoadPowerEditFieldLabel   matlab.ui.control.Label
    LoadPowerEditField      matlab.ui.control.NumericEditField
    WLabel                   matlab.ui.control.Label
    Label_2                  matlab.ui.control.Label
    Panel1SimulationofmultDERislandedMGwithACbus  matlab.ui.container.Panel
    aWindLabel               matlab.ui.control.Label
    PCWind                   matlab.ui.control.NumericEditField
    bSolarRadiationLabel    matlab.ui.control.Label
    PCSolar                  matlab.ui.control.NumericEditField
    SystemDisturbanceInputsPowerChangeofLabel  matlab.ui.control.Label
    SettingsLabel_2          matlab.ui.control.Label
    ResultsLabel_2           matlab.ui.control.Label
    BES0DropDownLabel                matlab.ui.control.Label
    BES0DropDown             matlab.ui.control.DropDown
    BES0DropDown_2Label          matlab.ui.control.Label
    BES0DropDown_2                   matlab.ui.control.DropDown
    AbbreviationsGuideLabel   matlab.ui.control.Label
    Label               matlab.ui.control.Label
    JLabel_2                 matlab.ui.control.Label
    JLabel_3                 matlab.ui.control.Label
    CalculateButton_2        matlab.ui.control.Button
```

```
        RightPanel                  matlab.ui.container.Panel
        TabGroup                    matlab.ui.container.TabGroup
        MGSimulation1Tab            matlab.ui.container.Tab
        GridLayout2                 matlab.ui.container.GridLayout
        UIAxes2AllEnergyChange      matlab.ui.control.UIAxes
        UIAxesAllPowerChange        matlab.ui.control.UIAxes
        MGSimulation2Tab            matlab.ui.container.Tab
        WTG                         matlab.ui.control.UIAxes
        PV                  matlab.ui.control.UIAxes
        MT                  matlab.ui.control.UIAxes
        FC                  matlab.ui.control.UIAxes
        BES                         matlab.ui.control.UIAxes
        f                   matlab.ui.control.UIAxes
        FCSimulationResult1Tab      matlab.ui.container.Tab
        EulersMethodUIAxes          matlab.ui.control.UIAxes
        ode45UIAxes                 matlab.ui.control.UIAxes
        EulerZoomSwitch             matlab.ui.control.Switch
        ode45ZoomSwitch             matlab.ui.control.Switch
        FCSimulationResult2Tab      matlab.ui.container.Tab
        CompareVoltagesUIAxes       matlab.ui.control.UIAxes
        CurrentRipplesUIAxes        matlab.ui.control.UIAxes
        TwoMethodVoltage            matlab.ui.control.Switch
        TwoMethodCurrent            matlab.ui.control.Switch
        end

        % Properties that correspond to apps with auto-reflow
        properties (Access = private)
        onePanelWidth = 576;
        end

        methods (Access = private)
        % Yiwen Jia
        % This section contains some user defined functions to assist the calculations in some of the
callback functions
        function [t_Euler, t_45, iL1_Euler, iL1_45, iL2_Euler, iL2_45, vo_Euler, vo_45, iFC_Euler,
iFC_45, vFC_Euler, vFC_45] = ODESolver(app)
        % This function solves the ODEs of the FC system using both Euler method and ode45()
        % Outputs: t_Euler, t_45 - the independent vectors calculated by the two methods
        %       iL1_Euler, iL1_45 - the current throught the inductor L1 calculated by the two methods
        %       iL2_Euler, iL2_45 - the current throught the inductor L2 calculated by the two methods
        %       vo_Euler, vo_45 - the voltage over the load calcualted by the two methods
        %       iFC_Euler, iFC_45 - the current throught the fuel cells calculated by the two methods
        %       vFC_Euler, vFC_45 - the voltage across the fuel cells calculated by the two methods
```

```
% Define the constants. All the values comes from [1]
Ncell = 72;        % number of fuel cells used to supply energy
T = 338.15;        % temperature inside the fuel cells while they are working, in K
PH2 = 1.494*101325;   % partial pressure of H2, Pa
PO2 = 0.21*101325;      % partial pressure of O2, Pa
A = 0.055;         % Tafel slope, together with the following two constants are used to calculate the deltaV_act, in V
i0 = 1.5e-3;   % exchange current, in A
in = 0.13;         % internal current, in A
Rohm = 0.006;  % resistance of the electrode, in Ohms
L = 0.68e-3;   % inductance of the two inductors, in H
RL = 0.1;          % resistance of inductors, in Ohms
C = 330e-6;        % capacity of the capacitor, in F
Ts = 8e-5;         % operation period, in s
D = app.DutyCycleSlider.Value;              % duty cycle
loadPower = app.LoadPowerEditField.Value; % output power ranging from 0W to 1000W. User input on GUI.

% Define help functions. The piecewise functions discribes the chart in Figure 2. Other help functions are provided in [1]
q1 = @(t) 1*double(t>=0 && t<D*Ts) + 0*double(t>=D*Ts && t<Ts);
q2 = @(t) 1*double((t>=0 && t<(D-0.5)*Ts) || (t>=Ts/2 && t<Ts)) + 0*double(t>=(D-0.5)*Ts && t<Ts/2);
V_Nernst = 1.482-0.00485*T-0.0000431*T*log(PH2*PO2);
del_V_act = @(iL1, iL2) A*log((iL1+iL2+in)/i0);
del_V_Ohm = @(iL1, iL2) Rohm*(iL1+iL2);
V_FC = @(iL1, iL2) Ncell*(V_Nernst-del_V_act(iL1, iL2)-del_V_Ohm(iL1, iL2));
io = 0;
% Define the ODEs. depV = [iL1; iL2; vo], and d_depV = [diL1/dt; diL2/dt; dvo/dt].
d_depV = @(t, depV) [(V_FC(depV(1), depV(2))-RL*depV(1)-(1-q1(mod(t, Ts)))*depV(3))/L;
                     (V_FC(depV(1), depV(2))-RL*depV(2)-(1-q2(mod(t, Ts)))*depV(3))/L;
                     ((1-q1(mod(t, Ts)))*depV(1)+(1-q2(mod(t, Ts)))*depV(2)-io)/C];

%*************** Solve the ODEs using Euler's wmethod ****************************************************
% Initialize variables
h = 8e-7;   % time step size
t_end = app.EndTimefrom0sSlider.Value;  % ending time, user input
t_Euler = 0:h:t_end;   % output t_Euler
depV0 = [0.0; 0.0; 0.0]; % initial conditions: iL1 = 0, iL2 = 0, vo = 0
depV_Euler = zeros(length(depV0), length(t_Euler)); % initialize the solution
depV_Euler(:, 1) = depV0; % set the initial conditions to the solution
for i = 1:length(t_Euler)-1
% set the value of io
```

```
        if(abs(depV_Euler(3,i)) < 1e-10)
                io = 0;
        else
                io = loadPower/depV_Euler(3,i);
        end
        % calculate every column of depV based on the previous column of depV
        depV_Euler(:, i+1) = depV_Euler(:, i) + d_depV(t_Euler(i), depV_Euler(:, i))*h;
        end

        %*************** Solve the ODEs using ode45()
**********************************************************************
        % Locate the first value of vo that is nonzero, then set the values of the dependent variables
        % at that location as the initial conditions, and set the start time at that location too.
        vo_Euler = depV_Euler(3, :);
        index_voNonZero = find(abs(vo_Euler) > 0, 1)
        depV0_45 = depV_Euler(:, index_voNonZero);
        depV0_45 = imagToReal(app, depV0_45);
        startTime_45 = h*(index_voNonZero-1);
        % Solve the ODEs
        [t_45, depV_45] = ode45(d_depV, [startTime_45 t_end], depV0_45);

        depV_Euler = depV_Euler'; % turn results to column vectors

        %*************** Outputs
**************************************************************************************
****
        % Extract outputs from the resulting matrices
        iL1_Euler = depV_Euler(:, 1);
        iL2_Euler = depV_Euler(:, 2);
        vo_Euler = depV_Euler(:, 3);
        iL1_45 = depV_45(:, 1);
        iL2_45 = depV_45(:, 2);
        vo_45 = depV_45(:, 3);
        % Calculate the magnitude of every imaginary element with their signs unchanged.
        iL1_Euler = imagToReal(app, iL1_Euler);
        iL2_Euler = imagToReal(app, iL2_Euler);
        vo_Euler = imagToReal(app, vo_Euler);
        iL1_45 = imagToReal(app, iL1_45);
        iL2_45 = imagToReal(app, iL2_45);
        vo_45 = imagToReal(app, vo_45);
        % Compute the rest of the outputs.
        iFC_Euler = iL1_Euler+iL2_Euler;
        vFC_Euler = V_FC(iL1_Euler, iL2_Euler);
        iFC_45 = iL1_45+iL2_45;
```

```
vFC_45 = V_FC(iL1_45, iL2_45);
end

%********* User defined function: Calculate max. current ripple
**********************************************************
function [maxRipple, t_max, maxes, t_min, mins] = maxCurrentRipple(app, data, t)
% This function calculates the maximum ripple of the given data.
% Inputs:  data  - the corresponding dependent vector with ripple data points
%        t       - given independent vector
%        h       - the step size of t
% Outputs: maxRipple - the maximum ripple of data
%        t_max   - a vector of independent values at local maximas
%        maxes   - a vector of local maximas
%        t_min   - a vector of independent values at local minimas
%        mins    - a vector of local minimas

maxIndex = 0;  % initiate index number of the vector of maximas
minIndex = 0;  % initiate index number of the vector of minimas
% calculate both the forward and backward derivatives on points from the 2nd through the 2nd
last one using vectorization method
forwardDer = (data(3:end)-data(2:end-1))./(t(3:end)-t(2:end-1));
backwardDer = (data(2:end-1)-data(1:end-2))./(t(2:end-1)-t(1:end-2));
for i = 1:length(forwardDer)
% add elements to local maxima vectors
if (forwardDer(i)<0 && backwardDer(i)>0)
        maxIndex = maxIndex + 1;
        maxes(maxIndex) = data(i+1);
        t_max(maxIndex) = t(i+1);
% add elements to local minima vectors
elseif (forwardDer(i)>0 && backwardDer(i)<0)  % Create a vector of minimas
        minIndex = minIndex + 1;
        mins(minIndex) = data(i+1);
        t_min(minIndex) = t(i+1);
end
end
% Calculate all the differences between every adjacent local maximas and minimas, and place
them in one vector.
if (length(maxes) > length(mins))
ripple = [maxes(1:end-1)-mins, maxes(2:end)-mins];
elseif (length(maxes) < length(mins))
ripple = [maxes-mins(1:end-1), maxes-mins(2:end)];
else
if(t_max(1) < t_min(1))
        ripple = [maxes-mins, maxes(2:end)-mins(1:end-1)];
```

```
        elseif (t_max(1) > t_min(1))
                ripple = [maxes-mins, maxes(1:end-1)-mins(2:end)];
        end
        end

        maxRipple = max(ripple);  % Find the maximum current ripple, A. Displayed on GUI
        end

        %********* User defined function: Select data from t > 0.01s
************************************************************
        function [T, IL1, IL2, Ifc, Vo] = selectData(app, t, numOfData, iL1, iL2, vo)
        % This is a helper function, selecting certain amount of data from the given vectors, then storing
them new vectors.
        % Inputs: t, iL1, iL2, vo - vectors where data is selected from
        %       numOfData - number of data selected
        % Outputs: T, IL1, IL2, Vo - new vectors of selected data from the respective given vectors
        %       Ifc - the sum of IL1 and IL2.

        index = find(t>0.01, 1); % find the index of the start of the stable state of the system
        % select numOfData amount of data from each given vectors.
        T = t(index:index+numOfData);
        IL1 = iL1(index:index+numOfData);
        IL2 = iL2(index:index+numOfData);
        Ifc = IL1+IL2;
        Vo = vo(index:index+numOfData);
        end

        %********* User defined function: Calculate the magnitude of the i
************************************************************
        function realVector = imagToReal(app, imag)
        % This function translates a vector of imaginary numbers to the vector of their respective
magnitudes, with signs unchanged
        for i = 1:length(imag)
        if(real(imag(i)) >= 0)
                imag(i) = abs(imag(i));
        else
                imag(i) = -abs(imag(i));
        end
        end
        realVector = imag; % Output the result
        end
        end
```

```
        % Callbacks that handle component events
        methods (Access = private)

        % Changes arrangement of the app based on UIFigure width
        function updateAppLayout(app, event)
        currentFigureWidth = app.FinalProjectUIFigure.Position(3);
        if(currentFigureWidth <= app.onePanelWidth)
        % Change to a 2x1 grid
        app.GridLayout.RowHeight = {715, 715};
        app.GridLayout.ColumnWidth = {'1x'};
        app.RightPanel.Layout.Row = 2;
        app.RightPanel.Layout.Column = 1;
        else
        % Change to a 1x2 grid
        app.GridLayout.RowHeight = {'1x'};
        app.GridLayout.ColumnWidth = {499, '1x'};
        app.RightPanel.Layout.Row = 1;
        app.RightPanel.Layout.Column = 2;
        end
        end

        % Button pushed function: CalculateButton
        function CalculateButtonPushed(app, event)
        % Yiwen Jia
        % This function simulates the system of FC, and outputs the results

    [t_Euler, t_45, iL1_Euler, iL1_45, iL2_Euler, iL2_45, vo_Euler, vo_45, iFC_Euler, iFC_45,
vFC_Euler, vFC_45] = ODESolver(app);

        % Plot the results calculated by Euler's method
        yyaxis(app.EulersMethodUIAxes, 'left')
        plot(app.EulersMethodUIAxes, t_Euler, iFC_Euler); hold on
        title(app.EulersMethodUIAxes, "Results Integrated by Euler's Method");
        xlabel(app.EulersMethodUIAxes, 'Time (s)');
        ylabel(app.EulersMethodUIAxes, 'Current generated by the fuel cells (A)');
        yyaxis(app.EulersMethodUIAxes, 'right')
        plot(app.EulersMethodUIAxes, t_Euler, vo_Euler);
        ylabel(app.EulersMethodUIAxes, 'Voltage of the load (V)');
        legend(app.EulersMethodUIAxes, {'iFC', 'v out'}, 'Location',"best");

        % Plot the results calculated by ode45()
        yyaxis(app.ode45UIAxes, "left")
        plot(app.ode45UIAxes, t_45, iFC_45); hold on
        title(app.ode45UIAxes, 'Results Integrated by ode45()');
```

```
        xlabel(app.ode45UIAxes, 'Time (s)');
        ylabel(app.ode45UIAxes, 'Current generated by the fuel cells (A)');
        yyaxis(app.ode45UIAxes, "right")
        plot(app.ode45UIAxes, t_45, vo_45); hold off
        ylabel(app.ode45UIAxes, 'Voltage of the load (V)');
        legend(app.ode45UIAxes, {'iFC', 'v out'},'Location',"best");

        % Compare the voltage output from the fuel cells (vFC) and the voltage over the load (vo)
        plot(app.CompareVoltagesUIAxes, t_Euler, [abs(vFC_Euler), abs(vo_Euler)]);
        title(app.CompareVoltagesUIAxes, 'Compare Vfc and Vo (Euler)');
        xlabel(app.CompareVoltagesUIAxes, 'Time (s)');
        ylabel(app.CompareVoltagesUIAxes, 'Voltage (V)');
        legend(app.CompareVoltagesUIAxes, {'Vfc - source', 'Vo - load'}, 'Location', 'best');

        % Display the current ripples on the selected data
        % Select 300 tabulated data points from results derived from ode45
        [T, IL1, IL2, Ifc, Vo] = selectData(app, t_Euler, 300, iL1_Euler, iL2_Euler, vo_Euler);
        % Calculate the local maximas and minimas
        [maxRipple, t_localMax, localMaxes, t_localMin, localMins] = maxCurrentRipple(app, Ifc, T);
        % Plot the selected data and the local maximas and minimas
        plot(app.CurrentRipplesUIAxes, T, [IL1, IL2, Ifc], '-', t_localMax, localMaxes, 'mo', t_localMin,
localMins, 'g*');
        title(app.CurrentRipplesUIAxes, 'Current ripples and local maximas and minimas (Euler)');
        xlabel(app.CurrentRipplesUIAxes, 'Time (s)');
        ylabel(app.CurrentRipplesUIAxes, 'Current (A)');
        legend(app.CurrentRipplesUIAxes, 'IL1', 'IL2', 'Ifc', 'maximas', 'minimas');

        % Post Data Process
        % Compute the power generated by the fuel cells, and calculate the energy generated over time T
using trapz()
        power = abs(vFC_45).*abs(iFC_45);  % Watts
        energyInput = trapz(t_45, power); % J. Value displayed on GUI
        app.EnergyGeneratedbytheFuelCellsEditField.Value = energyInput;
        % Calculate the maximum current ripple during the steady state (t > 0.008s, 1000 data points)
        index = find(t_45>0.008, 1);
        t_post = t_45(index:index+1000);
        iFC_post = iFC_45(index:index+1000);
        [max_deltaIfc, t_max, maxes, t_min, mins] = maxCurrentRipple(app, iFC_post, t_post);
        app.MaxCurentRippleEditField.Value = abs(max_deltaIfc);

        % Reset the control switches of the app
        app.EulerZoomSwitch.Value = 0;
        app.ode45ZoomSwitch.Value = 0;
        app.TwoMethodVoltage.Value = 0;
```

```
        app.TwoMethodCurrent.Value = 0;

        end

        % Value changed function: EulerZoomSwitch
        function EulerZoomSwitchValueChanged(app, event)
        % Yiwen Jia
        % This function enables the switch next to the 1st chart on the tab
        % FC-Simulation Result 1 to toggle between the zoomed-in (state
        % value = 1) and zoomed-out (state value = 0) versions of the data
        % calculated by Euler's method

        value = app.EulerZoomSwitch.Value;
        [t_Euler, t_45, iL1_Euler, iL1_45, iL2_Euler, iL2_45, vo_Euler, vo_45, iFC_Euler, iFC_45,
vFC_Euler, vFC_45] = ODESolver(app);
        if (value == 0)
        % Plot the results calculated by Euler's method (zoom out)
        yyaxis(app.EulersMethodUIAxes, 'left')
        plot(app.EulersMethodUIAxes, t_Euler, iFC_Euler); hold on
        ylabel(app.EulersMethodUIAxes, 'Current generated by the fuel cells (A)');
        yyaxis(app.EulersMethodUIAxes, 'right')
        plot(app.EulersMethodUIAxes, t_Euler, vo_Euler); hold off
        ylabel(app.EulersMethodUIAxes, 'Voltage of the load (V)');
        elseif (value == 1)
        % Select zoomed-in tabulated data from Euler's results
        [T, IL1, IL2, Ifc, Vo] = selectData(app, t_Euler, 300, iL1_Euler, iL2_Euler, vo_Euler);
        % plot the zoomed-in data
        yyaxis(app.EulersMethodUIAxes, 'left')
        plot(app.EulersMethodUIAxes, T, Ifc,'b-'); hold on
        ylabel(app.EulersMethodUIAxes, 'Current generated by the fuel cells (A)');
        yyaxis(app.EulersMethodUIAxes, 'right')
        plot(app.EulersMethodUIAxes, T, Vo); hold off
        ylabel(app.EulersMethodUIAxes, 'Voltage across the load (V)');
        end
        title(app.EulersMethodUIAxes, "Results Integrated by Euler's Method");
        xlabel(app.EulersMethodUIAxes, 'Time (s)');
        legend(app.EulersMethodUIAxes, {'iFC', 'v out'}, 'Location',"best");
        end

        % Value changed function: ode45ZoomSwitch
        function ode45ZoomSwitchValueChanged(app, event)
        % Yiwen Jia
        % This function enables the switch next to the 2nd chart on the tab
        % FC-Simulation Result 1 to toggle between the zoomed-in (state
```

```
        % value = 1) and zoomed-out (state value = 0) versions of the data
        % calculated by ode45()

        value = app.ode45ZoomSwitch.Value;
        [t_Euler, t_45, iL1_Euler, iL1_45, iL2_Euler, iL2_45, vo_Euler, vo_45, iFC_Euler, iFC_45,
vFC_Euler, vFC_45] = ODESolver(app);
        if (value == 0)
        % Plot the results calculated by ode45() (zoom out)
        yyaxis(app.ode45UIAxes, "left")
        plot(app.ode45UIAxes, t_45, iFC_45); hold on
        ylabel(app.ode45UIAxes, 'Current generated by the fuel cells (A)');
        yyaxis(app.ode45UIAxes, "right")
        plot(app.ode45UIAxes, t_45, abs(vo_45)); hold off
        ylabel(app.ode45UIAxes, 'Voltage of the load (V)');
        elseif (value == 1)
        % Select tabulated data from ode45's results (zoom in)
        [T, IL1, IL2, Ifc, Vo] = selectData(app, t_45, 300, iL1_45, iL2_45, vo_45);
        % Plot tabulated data from ode45's results (zoom in)
        yyaxis(app.ode45UIAxes, 'left')
        plot(app.ode45UIAxes, T, Ifc,'b-'); hold on
        ylabel(app.ode45UIAxes, 'Current generated by the fuel cells (A)');
        yyaxis(app.ode45UIAxes, 'right')
        plot(app.ode45UIAxes, T, Vo);
        ylabel(app.ode45UIAxes, 'Voltage across the load (V)');
        end
        title(app.ode45UIAxes, 'Results Integrated by ode45()');
        xlabel(app.ode45UIAxes, 'Time (s)');
        legend(app.ode45UIAxes, {'iFC', 'v out'},'Location',"best");
        end

        % Value changed function: TwoMethodVoltage
        function TwoMethodVoltageValueChanged(app, event)
        % Yiwen Jia
        % This function enables the switch next to the 1st chart on the tab
        % FC-Simulation Result 2 to toggle between the voltages calculated
        % by Euler's method (state value = 0) and by ode45() (state value = 1)

        value = app.TwoMethodVoltage.Value;
        [t_Euler, t_45, iL1_Euler, iL1_45, iL2_Euler, iL2_45, vo_Euler, vo_45, iFC_Euler, iFC_45,
vFC_Euler, vFC_45] = ODESolver(app);
        % Define the data under different state values of the switch
        if (value == 1)
        T = t_45;
        vFC = vFC_45;
```

```
        vo = vo_45;
        title(app.CompareVoltagesUIAxes, 'Compare Vfc and Vo (ode45)');
        elseif  (value == 0)
        T = t_Euler;
        vFC = vFC_Euler;
        vo = vo_Euler;
        title(app.CompareVoltagesUIAxes, 'Compare Vfc and Vo (Euler)');
        end
        % Plot the data
        plot(app.CompareVoltagesUIAxes, T, [abs(vFC), abs(vo)]);
        xlabel(app.CompareVoltagesUIAxes, 'Time (s)');
        ylabel(app.CompareVoltagesUIAxes, 'Voltage (V)');
        legend(app.CompareVoltagesUIAxes, {'Vfc - source', 'Vo - load'}, 'Location', 'best');

        end

        % Value changed function: TwoMethodCurrent
        function TwoMethodCurrentValueChanged(app, event)
        % Yiwen Jia
        % This function enables the switch next to the 2nd chart on the tab
        % FC-Simulation Result 2 to toggle between the currents calculated
        % by Euler's method (state value = 0) and by ode45() (state value = 1)

        value = app.TwoMethodCurrent.Value;
        [t_Euler, t_45, iL1_Euler, iL1_45, iL2_Euler, iL2_45, vo_Euler, vo_45, iFC_Euler, iFC_45,
vFC_Euler, vFC_45] = ODESolver(app);
        % Display the current ripples on the selected data
        % Select 300 tabulated data points from results calcualted by both methods
        if (value == 1)
        [T, IL1, IL2, Ifc, Vo] = selectData(app, t_45, 300, iL1_45, iL2_45, vo_45);
        title(app.CurrentRipplesUIAxes, 'Current ripples and local maximas and minimas (ode45)');
        elseif (value == 0)
        [T, IL1, IL2, Ifc, Vo] = selectData(app, t_Euler, 300, iL1_Euler, iL2_Euler, vo_Euler);
        title(app.CurrentRipplesUIAxes, 'Current ripples and local maximas and minimas (Euler)');
        end
        % Calculate the local maximas and minimas
        [maxRipple, t_localMax, localMaxes, t_localMin, localMins] = maxCurrentRipple(app, Ifc, T);
        % Plot the selected data and the local maximas and minimas
        plot(app.CurrentRipplesUIAxes, T, [IL1, IL2, Ifc], '-', t_localMax, localMaxes, 'mo', t_localMin,
localMins, 'g*');
        xlabel(app.CurrentRipplesUIAxes, 'Time (s)');
        ylabel(app.CurrentRipplesUIAxes, 'Current (A)');
        legend(app.CurrentRipplesUIAxes, 'IL1', 'IL2', 'Ifc', 'maximas', 'minimas');
        end
```

```
        % Button pushed function: CalculateButton_2
        function CalculateButton_2Pushed(app, event)
        % Hanne Nicolaisen
        % Microgrid Simulation
        % Final Project

% Simulation Parameters
T_wtg = 1.5;
T_pv = 1.8;
T_mt = 2;
T_fc = 1.5;
T_bes = 0.1;
D = 0.012;
M = 0.2;
% Parameters unused but provided by [6]
% a = 1;
% b = 0.8;
% gamma1 = 0.02;

u = .008; % control input
tspan = [0, 10]; % time interval

Pw = app.PCWind.Value; % User determines power change of wind
Pphi = app.PCSolar.Value; % User determines power change of solar radiation

% System of ODEs
dpdt = @(t,P) [((-1/T_wtg)*P(1) + (1/T_wtg)*Pw);
                ((-1/T_pv)*P(2) + (1/T_pv)*Pphi);
                ((-1/T_mt)*P(3) + (1/T_mt)*u);
                ((-1/T_fc)*P(4) + (1/T_fc)*u);
                ((-1/T_bes)*P(5) + ((1/T_bes)*P(6)));
                ((-2*D/M)*P(6) -(2/M)*(P(1)+P(2)+P(3)+P(4)+P(5)+P(6)))];

% Inital Values
% P0 = [P_wtg, P_pv, P_mt, P_fc, P_bes, f]'; This is key for inputs
P0 = [.5, .8, .2, .4, 4, 1]';


% solve ODE using userdefined function odeEueler
[t,P] = odeEuler(dpdt, tspan, .1, P0);

% plot solution for T_wtg
plot(app.WTG,t,P(:,1))
```

```
% plot solution for T_pv
plot(app.PV,t,P(:,2))

% plot solution for T_mt
plot(app.MT,t,P(:,3))

% plot the solution for T_fc
plot(app.FC,t,P(:,4))

% plot solution for T_bes
plot(app.BES,t,P(:,5))

% plot solution for f
plot(app.f,t,P(:,6))

% plot change in power all together
plot(app.UIAxesAllPowerChange, t, P(:,1));
hold(app.UIAxesAllPowerChange,'on');
plot(app.UIAxesAllPowerChange,t,P(:,2));
hold(app.UIAxesAllPowerChange,'on');
plot(app.UIAxesAllPowerChange,t,P(:,3));
hold(app.UIAxesAllPowerChange,'on');
plot(app.UIAxesAllPowerChange,t,P(:,4));
hold(app.UIAxesAllPowerChange,'on');
plot(app.UIAxesAllPowerChange,t,P(:,5));
hold(app.UIAxesAllPowerChange,'on');
plot(app.UIAxesAllPowerChange,t,P(:,6));
hold(app.UIAxesAllPowerChange,'off');
legend(app.UIAxesAllPowerChange, '\DeltaWTG (J) ','\DeltaPV (J)','\DeltaMT (J)','\DeltaFC
(J)','\DeltaBES (J)','\Deltaf (Hz)','Location','northeast');

% solve for integral, or total energy using trapezoidal rule
EnergyofWTP = TrapezoidalIntegration(P(:,1),tspan,.1);
EnergyofPV = TrapezoidalIntegration(P(:,2),tspan,.1);
EnergyofMT = TrapezoidalIntegration(P(:,3),tspan,.1);
EnergyofFC = TrapezoidalIntegration(P(:,4),tspan,.1);
EnergyofBES = TrapezoidalIntegration(P(:,5),tspan,.1);

% plot the integrals together as bar graphs
bar(app.UIAxes2AllEnergyChange, 1, EnergyofWTP);
hold(app.UIAxes2AllEnergyChange,'on');
bar(app.UIAxes2AllEnergyChange, 2, EnergyofPV);
hold(app.UIAxes2AllEnergyChange,'on');
```

```
bar(app.UIAxes2AllEnergyChange, 3, EnergyofMT);
hold(app.UIAxes2AllEnergyChange,'on');
bar(app.UIAxes2AllEnergyChange, 4, EnergyofFC);
hold(app.UIAxes2AllEnergyChange,'on');
bar(app.UIAxes2AllEnergyChange, 5, EnergyofBES);
hold(app.UIAxes2AllEnergyChange,'off');
set(app.UIAxes2AllEnergyChange,'XTickLabel',[]);
legend(app.UIAxes2AllEnergyChange,'\DeltaWTG','\DeltaPV','\DeltaMT','\DeltaFC','\DeltaBES','Locatio
n','northeast');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% User defined functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% User defined function odeEuler implements Euler's method to estimate the solution of a vector-valued
ODE.
% Inputs:
%   f - a handle to a MATLAB function of the form dydx = f(x,y) that defines the ODE system, where x is
a scalar and
% both y and dydx are column vectors.
%   range - the range of independent variables over which the simulation will be performed [initial-value,
final-value]
%   h - step size
%   y0 - column vector representing the initial-value(s)
% Outputs:
%   x - row vector of x-points (independent variable) in the approximate solution.
%   y - solution array. Each row in y corresponds to the solution(s) at the x-value in the corresponding
element of x.
%       y(i,:) corresponds to the solution(s) at x(i)
function [x,y] = odeEuler(f,range,h,y0)

% Initialize the output solution arrays.
x = range(1):h:range(2);
y = zeros(length(y0),length(x));
y(:,1) = y0;

% Compute x and y using Euler's method.
for i = 1:length(x) - 1
        y(:, i+1) = y(:,i) + h*f(x(i),y(:,i));
end

x = x';
y = y';   % output y as a series of column vectors.
```

```
end

% User defined function TrapezoidalIntegration computes the trapezoidal rule using an anonymous
function
%       Inputs: - ftn: the function handle to the anonymous function
%               - limits: the limits of integration (a vector of two terms)
%               - subIntervals: the number of subintervals to use in the integration
%       Outputs: - NumIntegral: the numerical integratl of the function over the limits provided using the
%               trapezoidal rule as a scalar value
function [NumIntegral] = TrapezoidalIntegration(ftn,limits,subIntervals)

lower =  limits(:,1)+1; % lower limit
upper = limits(:,2); % upper limit

%fplot(ftn,[lower,upper]); title('plot'); xlabel('x'),ylabel('ftn(x)') % plot of function over range specified by
limits
%hold on; plot([lower,upper],[ftn(lower),ftn(upper)]); hold off

h = (upper-lower)/subIntervals; % delta x = h, the width of each subinterval
Trap = (ftn(lower) + ftn(upper))/2; % first and last term

TrapSum = 0; % Initialize the summation
for i = 1:subIntervals-1 % initialize the index variable, increment by 1 until subintervals-1
        TrapSum = TrapSum + ftn(lower+i*h); % take initial value and adding the value of the function
xi (applying xi directly into ftn call)
end

NumIntegral = h*(Trap + TrapSum); % summing the two terms together and multiplying by h

end
        end
        end

        % Component initialization
        methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

        % Create FinalProjectUIFigure and hide until all components are created
        app.FinalProjectUIFigure = uifigure('Visible', 'off');
        app.FinalProjectUIFigure.AutoResizeChildren = 'off';
        app.FinalProjectUIFigure.Color = [0.949 0.9412 0.9412];
```

```
app.FinalProjectUIFigure.Position = [100 100 1127 715];
app.FinalProjectUIFigure.Name = 'Simulations of Dynamic Systems';
app.FinalProjectUIFigure.SizeChangedFcn = createCallbackFcn(app, @updateAppLayout, true);

% Create GridLayout
app.GridLayout = uigridlayout(app.FinalProjectUIFigure);
app.GridLayout.ColumnWidth = {499, '1x'};
app.GridLayout.RowHeight = {'1x'};
app.GridLayout.ColumnSpacing = 0;
app.GridLayout.RowSpacing = 0;
app.GridLayout.Padding = [0 0 0 0];
app.GridLayout.Scrollable = 'on';

% Create LeftPanel
app.LeftPanel = uipanel(app.GridLayout);
app.LeftPanel.ForegroundColor = [0.149 0.149 0.149];
app.LeftPanel.BackgroundColor = [0.0196 0.0196 0.2588];
app.LeftPanel.Layout.Row = 1;
app.LeftPanel.Layout.Column = 1;
app.LeftPanel.Scrollable = 'on';

% Create Panel2
app.Panel2 = uipanel(app.LeftPanel);
app.Panel2.Tooltip = {''};
app.Panel2.ForegroundColor = [0.0196 0.1804 0.3098];
app.Panel2.TitlePosition = 'centertop';
app.Panel2.Title = 'Panel 2. Simulation of circuit with interleaved boost dc-dc converter';
app.Panel2.BackgroundColor = [0.949 0.9608 0.9686];
app.Panel2.FontWeight = 'bold';
app.Panel2.Scrollable = 'on';
app.Panel2.Position = [5 7 490 335];

% Create ALabel
app.ALabel = uilabel(app.Panel2);
app.ALabel.Position = [396 63 25 22];
app.ALabel.Text = 'A';

% Create JLabel
app.JLabel = uilabel(app.Panel2);
app.JLabel.Position = [396 23 25 22];
app.JLabel.Text = 'J';

% Create CalculateButton
app.CalculateButton = uibutton(app.Panel2, 'push');
```

```
        app.CalculateButton.ButtonPushedFcn = createCallbackFcn(app, @CalculateButtonPushed,
true);
        app.CalculateButton.BackgroundColor = [0.102 0.2235 0.349];
        app.CalculateButton.FontName = 'Calibri';
        app.CalculateButton.FontSize = 14;
        app.CalculateButton.FontWeight = 'bold';
        app.CalculateButton.FontColor = [0.9412 0.9412 0.9412];
        app.CalculateButton.Position = [62 44 100 26];
        app.CalculateButton.Text = 'Calculate';

        % Create MaxCurentRippleLabel
        app.MaxCurentRippleLabel = uilabel(app.Panel2);
        app.MaxCurentRippleLabel.HorizontalAlignment = 'center';
        app.MaxCurentRippleLabel.FontName = 'Calibri';
        app.MaxCurentRippleLabel.FontSize = 14;
        app.MaxCurentRippleLabel.FontWeight = 'bold';
        app.MaxCurentRippleLabel.FontColor = [0.0549 0.2235 0.3686];
        app.MaxCurentRippleLabel.Position = [188 63 113 22];
        app.MaxCurentRippleLabel.Text = 'Max.Curent Ripple';

        % Create MaxCurentRippleEditField
        app.MaxCurentRippleEditField = uieditfield(app.Panel2, 'numeric');
        app.MaxCurentRippleEditField.FontColor = [0.0471 0.1647 0.2706];
        app.MaxCurentRippleEditField.Position = [312 63 76 22];

        % Create EnergyGeneratedbytheFuelCellsEditFieldLabel
        app.EnergyGeneratedbytheFuelCellsEditFieldLabel = uilabel(app.Panel2);
        app.EnergyGeneratedbytheFuelCellsEditFieldLabel.HorizontalAlignment = 'center';
        app.EnergyGeneratedbytheFuelCellsEditFieldLabel.FontWeight = 'bold';
        app.EnergyGeneratedbytheFuelCellsEditFieldLabel.FontColor = [0.0588 0.2392 0.349];
        app.EnergyGeneratedbytheFuelCellsEditFieldLabel.Position = [192 14 106 41];
        app.EnergyGeneratedbytheFuelCellsEditFieldLabel.Text = 'Energy Generated by the Fuel Cells';

        % Create EnergyGeneratedbytheFuelCellsEditField
        app.EnergyGeneratedbytheFuelCellsEditField = uieditfield(app.Panel2, 'numeric');
        app.EnergyGeneratedbytheFuelCellsEditField.ValueDisplayFormat = '%6.4g';
        app.EnergyGeneratedbytheFuelCellsEditField.Position = [312 23 76 22];

        % Create DutyCycleSliderLabel
        app.DutyCycleSliderLabel = uilabel(app.Panel2);
        app.DutyCycleSliderLabel.HorizontalAlignment = 'right';
        app.DutyCycleSliderLabel.FontName = 'Eras Light ITC';
        app.DutyCycleSliderLabel.FontWeight = 'bold';
        app.DutyCycleSliderLabel.FontColor = [0.2549 0.1137 0.5608];
```

```
app.DutyCycleSliderLabel.Position = [209 271 67 22];
app.DutyCycleSliderLabel.Text = 'Duty Cycle';

% Create DutyCycleSlider
app.DutyCycleSlider = uislider(app.Panel2);
app.DutyCycleSlider.Limits = [0.51 0.8];
app.DutyCycleSlider.Orientation = 'vertical';
app.DutyCycleSlider.Position = [219 127 3 128];
app.DutyCycleSlider.Value = 0.6;

% Create SettingsLabel
app.SettingsLabel = uilabel(app.Panel2);
app.SettingsLabel.FontName = 'Adobe Ming Std L';
app.SettingsLabel.FontSize = 16;
app.SettingsLabel.FontWeight = 'bold';
app.SettingsLabel.FontColor = [0.0471 0.1412 0.2902];
app.SettingsLabel.Position = [34 271 87 22];
app.SettingsLabel.Text = 'Settings ';

% Create EndTimefrom0sLabel
app.EndTimefrom0sLabel = uilabel(app.Panel2);
app.EndTimefrom0sLabel.HorizontalAlignment = 'right';
app.EndTimefrom0sLabel.FontWeight = 'bold';
app.EndTimefrom0sLabel.FontColor = [0.098 0.2039 0.4706];
app.EndTimefrom0sLabel.Position = [303 271 114 22];
app.EndTimefrom0sLabel.Text = 'End Time (from 0s)';

% Create EndTimefrom0sSlider
app.EndTimefrom0sSlider = uislider(app.Panel2);
app.EndTimefrom0sSlider.Limits = [0.02 0.08];
app.EndTimefrom0sSlider.MajorTicks = [0.001 0.02 0.04 0.06 0.08];
app.EndTimefrom0sSlider.MajorTickLabels = {'0.001', '0.02', '0.04', '0.06', '0.08'};
app.EndTimefrom0sSlider.Orientation = 'vertical';
app.EndTimefrom0sSlider.Position = [354 126 3 129];
app.EndTimefrom0sSlider.Value = 0.03;

% Create ResultsLabel
app.ResultsLabel = uilabel(app.Panel2);
app.ResultsLabel.FontName = 'Adobe Ming Std L';
app.ResultsLabel.FontSize = 16;
app.ResultsLabel.FontWeight = 'bold';
app.ResultsLabel.FontColor = [0.0471 0.1412 0.2902];
app.ResultsLabel.Position = [34 90 87 22];
app.ResultsLabel.Text = 'Results';
```

```
        % Create LoadPowerEditFieldLabel
        app.LoadPowerEditFieldLabel = uilabel(app.Panel2);
        app.LoadPowerEditFieldLabel.HorizontalAlignment = 'right';
        app.LoadPowerEditFieldLabel.FontColor = [0.0706 0.2706 0.4];
        app.LoadPowerEditFieldLabel.Position = [34 204 70 22];
        app.LoadPowerEditFieldLabel.Text = 'Load Power';

        % Create LoadPowerEditField
        app.LoadPowerEditField = uieditfield(app.Panel2, 'numeric');
        app.LoadPowerEditField.Limits = [0 1000];
        app.LoadPowerEditField.Position = [111 204 51 22];
        app.LoadPowerEditField.Value = 600;

        % Create WLabel
        app.WLabel = uilabel(app.Panel2);
        app.WLabel.FontColor = [0.0588 0.2431 0.349];
        app.WLabel.Position = [170 205 25 22];
        app.WLabel.Text = 'W';

        % Create Label_2
        app.Label_2 = uilabel(app.Panel2);
        app.Label_2.FontColor = [0.0471 0.2392 0.349];
        app.Label_2.Position = [48 183 54 22];
        app.Label_2.Text = '(0, 1000)';

        % Create Panel1SimulationofmultDERislandedMGwithACbus
        app.Panel1SimulationofmultDERislandedMGwithACbus = uipanel(app.LeftPanel);
        app.Panel1SimulationofmultDERislandedMGwithACbus.ForegroundColor = [0.0196 0.1804
0.3098];
        app.Panel1SimulationofmultDERislandedMGwithACbus.TitlePosition = 'centertop';
        app.Panel1SimulationofmultDERislandedMGwithACbus.Title = 'Panel 1: Simulation of
mult-DER islanded MG with AC bus';
        app.Panel1SimulationofmultDERislandedMGwithACbus.BackgroundColor = [0.949 0.9608
0.9686];
        app.Panel1SimulationofmultDERislandedMGwithACbus.FontWeight = 'bold';
        app.Panel1SimulationofmultDERislandedMGwithACbus.Position = [5 350 489 359];

        % Create aWindLabel
        app.aWindLabel = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.aWindLabel.HorizontalAlignment = 'right';
        app.aWindLabel.FontColor = [0.051 0.1412 0.2902];
        app.aWindLabel.Position = [24 189 46 22];
        app.aWindLabel.Text = 'a. Wind';
```

```
        % Create PCWind
        app.PCWind = uieditfield(app.Panel1SimulationofmultDERislandedMGwithACbus, 'numeric');
        app.PCWind.Position = [139 189 57 22];

        % Create bSolarRadiationLabel
        app.bSolarRadiationLabel = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.bSolarRadiationLabel.HorizontalAlignment = 'right';
        app.bSolarRadiationLabel.FontColor = [0.051 0.1412 0.2902];
        app.bSolarRadiationLabel.Position = [24 151 101 22];
        app.bSolarRadiationLabel.Text = 'b. Solar Radiation';

        % Create PCSolar
        app.PCSolar = uieditfield(app.Panel1SimulationofmultDERislandedMGwithACbus, 'numeric');
        app.PCSolar.Position = [139 150 57 22];

        % Create SystemDisturbanceInputsPowerChangeofLabel
        app.SystemDisturbanceInputsPowerChangeofLabel =
uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.SystemDisturbanceInputsPowerChangeofLabel.FontWeight = 'bold';
        app.SystemDisturbanceInputsPowerChangeofLabel.FontColor = [0.051 0.1412 0.2902];
        app.SystemDisturbanceInputsPowerChangeofLabel.Position = [17 226 207 42];
        app.SystemDisturbanceInputsPowerChangeofLabel.Text = {'System Disturbance Inputs'; '';
'Power Change of:'};

        % Create SettingsLabel_2
        app.SettingsLabel_2 = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.SettingsLabel_2.FontName = 'Adobe Ming Std L';
        app.SettingsLabel_2.FontSize = 16;
        app.SettingsLabel_2.FontWeight = 'bold';
        app.SettingsLabel_2.FontColor = [0.051 0.1412 0.2902];
        app.SettingsLabel_2.Position = [34 287 87 22];
        app.SettingsLabel_2.Text = 'Settings ';

        % Create ResultsLabel_2
        app.ResultsLabel_2 = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.ResultsLabel_2.FontName = 'Adobe Ming Std L';
        app.ResultsLabel_2.FontSize = 16;
        app.ResultsLabel_2.FontWeight = 'bold';
        app.ResultsLabel_2.FontColor = [0.0471 0.1412 0.2902];
        app.ResultsLabel_2.Position = [25 81 87 22];
        app.ResultsLabel_2.Text = 'Results';

        % Create BES0DropDownLabel
```

```
            app.BES0DropDownLabel = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
            app.BES0DropDownLabel.HorizontalAlignment = 'right';
            app.BES0DropDownLabel.Position = [203 46 49 22];
            app.BES0DropDownLabel.Text = 'BES > 0';

            % Create BES0DropDown
            app.BES0DropDown = uidropdown(app.Panel1SimulationofmultDERislandedMGwithACbus);
            app.BES0DropDown.Items = {'If BES > 0, then the amount of power supplied to the battery is
greater than the power drawn from it. The excess power is stored in the battery.'};
            app.BES0DropDown.Tooltip = {''};
            app.BES0DropDown.Position = [181 19 136 27];
            app.BES0DropDown.Value = 'If BES > 0, then the amount of power supplied to the battery is
greater than the power drawn from it. The excess power is stored in the battery.';

            % Create BES0DropDown_2Label
            app.BES0DropDown_2Label =
uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
            app.BES0DropDown_2Label.HorizontalAlignment = 'right';
            app.BES0DropDown_2Label.Position = [363 46 49 22];
            app.BES0DropDown_2Label.Text = 'BES < 0';

            % Create BES0DropDown_2
            app.BES0DropDown_2 =
uidropdown(app.Panel1SimulationofmultDERislandedMGwithACbus);
            app.BES0DropDown_2.Items = {'If BES < 0, the battery''s maximum capacity has been reached
and excess power is wasted.'};
            app.BES0DropDown_2.Position = [341 19 136 27];
            app.BES0DropDown_2.Value = 'If BES < 0, the battery''s maximum capacity has been reached
and excess power is wasted.';

            % Create AbbreviationsGuideLabel
            app.AbbreviationsGuideLabel =
uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
            app.AbbreviationsGuideLabel.FontWeight = 'bold';
            app.AbbreviationsGuideLabel.FontColor = [0.051 0.1412 0.2902];
            app.AbbreviationsGuideLabel.Position = [334 299 136 25];
            app.AbbreviationsGuideLabel.Text = 'Abbreviations Guide';

            % Create Label
            app.Label = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
            app.Label.FontSize = 10;
            app.Label.FontWeight = 'bold';
            app.Label.FontColor = [0.051 0.1412 0.2902];
            app.Label.Position = [309 159 180 127];
```

```
        app.Label.Text = {'WTG = Wind Turbine Generator'; ''; 'PV = Photovoltaic Generator'; ''; 'FC =
Fuel Cell'; ''; 'MT = Micro Turbine'; ''; 'BES = Battery Energy Storage'; ''; 'f =  The AC Bus Frequency
Deviation'};

        % Create JLabel_2
        app.JLabel_2 = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.JLabel_2.Position = [203 189 25 22];
        app.JLabel_2.Text = 'J';

        % Create JLabel_3
        app.JLabel_3 = uilabel(app.Panel1SimulationofmultDERislandedMGwithACbus);
        app.JLabel_3.Position = [204 150 25 22];
        app.JLabel_3.Text = 'J';

        % Create CalculateButton_2
        app.CalculateButton_2 = uibutton(app.Panel1SimulationofmultDERislandedMGwithACbus,
'push');
        app.CalculateButton_2.ButtonPushedFcn = createCallbackFcn(app, @CalculateButton_2Pushed,
true);
        app.CalculateButton_2.BackgroundColor = [0.102 0.2235 0.349];
        app.CalculateButton_2.FontName = 'Calibri';
        app.CalculateButton_2.FontSize = 14;
        app.CalculateButton_2.FontWeight = 'bold';
        app.CalculateButton_2.FontColor = [0.9412 0.9412 0.9412];
        app.CalculateButton_2.Position = [58 30 100 26];
        app.CalculateButton_2.Text = 'Calculate';

        % Create RightPanel
        app.RightPanel = uipanel(app.GridLayout);
        app.RightPanel.BackgroundColor = [0.0353 0.2863 0.3098];
        app.RightPanel.Layout.Row = 1;
        app.RightPanel.Layout.Column = 2;
        app.RightPanel.Scrollable = 'on';

        % Create TabGroup
        app.TabGroup = uitabgroup(app.RightPanel);
        app.TabGroup.Position = [1 8 622 701];

        % Create MGSimulation1Tab
        app.MGSimulation1Tab = uitab(app.TabGroup);
        app.MGSimulation1Tab.Title = 'MG - Simulation 1';
        app.MGSimulation1Tab.BackgroundColor = [0.949 0.9608 0.9686];
        app.MGSimulation1Tab.ForegroundColor = [0.0196 0.1804 0.3098];
```

```
% Create GridLayout2
app.GridLayout2 = uigridlayout(app.MGSimulation1Tab);
app.GridLayout2.ColumnWidth = {'1x'};
app.GridLayout2.RowHeight = {'1x', '1.04x'};
app.GridLayout2.Padding = [9 10 9 10];

% Create UIAxes2AllEnergyChange
app.UIAxes2AllEnergyChange = uiaxes(app.GridLayout2);
title(app.UIAxes2AllEnergyChange, 'Total Energy Transferred/Used')
xlabel(app.UIAxes2AllEnergyChange, 'Source')
ylabel(app.UIAxes2AllEnergyChange, 'J')
zlabel(app.UIAxes2AllEnergyChange, 'Z')
app.UIAxes2AllEnergyChange.Layout.Row = 2;
app.UIAxes2AllEnergyChange.Layout.Column = 1;

% Create UIAxesAllPowerChange
app.UIAxesAllPowerChange = uiaxes(app.GridLayout2);
title(app.UIAxesAllPowerChange, 'Total Power Generation/Consumption ')
xlabel(app.UIAxesAllPowerChange, 'Time (s)')
ylabel(app.UIAxesAllPowerChange, 'Hz or J')
zlabel(app.UIAxesAllPowerChange, 'Z')
app.UIAxesAllPowerChange.Layout.Row = 1;
app.UIAxesAllPowerChange.Layout.Column = 1;

% Create MGSimulation2Tab
app.MGSimulation2Tab = uitab(app.TabGroup);
app.MGSimulation2Tab.Title = 'MG - Simulation 2';
app.MGSimulation2Tab.BackgroundColor = [0.949 0.9608 0.9686];
app.MGSimulation2Tab.ForegroundColor = [0.0196 0.1804 0.3098];

% Create WTG
app.WTG = uiaxes(app.MGSimulation2Tab);
title(app.WTG, 'WTG')
xlabel(app.WTG, 't(s)')
ylabel(app.WTG, 'Change in Power (J)')
zlabel(app.WTG, 'Z')
app.WTG.Position = [10 450 288 215];

% Create PV
app.PV = uiaxes(app.MGSimulation2Tab);
title(app.PV, 'PV')
xlabel(app.PV, 't(s)')
ylabel(app.PV, 'Change in Power (J)')
zlabel(app.PV, 'Z')
```

```
app.PV.Position = [316 450 288 215];

% Create MT
app.MT = uiaxes(app.MGSimulation2Tab);
title(app.MT, 'MT')
xlabel(app.MT, 't(s)')
ylabel(app.MT, 'Change in Power (J)')
zlabel(app.MT, 'Z')
app.MT.Position = [10 231 288 215];

% Create FC
app.FC = uiaxes(app.MGSimulation2Tab);
title(app.FC, 'FC')
xlabel(app.FC, 't(s)')
ylabel(app.FC, 'Change in Power (J)')
zlabel(app.FC, 'Z')
app.FC.Position = [316 231 288 215];

% Create BES
app.BES = uiaxes(app.MGSimulation2Tab);
title(app.BES, 'BES')
xlabel(app.BES, 't(s)')
ylabel(app.BES, 'Change in Power (J)')
zlabel(app.BES, 'Z')
app.BES.Position = [10 11 288 215];

% Create f
app.f = uiaxes(app.MGSimulation2Tab);
title(app.f, 'f')
xlabel(app.f, 't(s)')
ylabel(app.f, 'System Controlled Output (Hz)')
zlabel(app.f, 'Z')
app.f.Position = [316 11 288 215];

% Create FCSimulationResult1Tab
app.FCSimulationResult1Tab = uitab(app.TabGroup);
app.FCSimulationResult1Tab.Title = 'FC - Simulation Result 1';
app.FCSimulationResult1Tab.BackgroundColor = [0.9216 0.9412 0.949];
app.FCSimulationResult1Tab.ForegroundColor = [0.051 0.2941 0.349];

% Create EulersMethodUIAxes
app.EulersMethodUIAxes = uiaxes(app.FCSimulationResult1Tab);
title(app.EulersMethodUIAxes, 'Results Integrated by Euler''s Method')
xlabel(app.EulersMethodUIAxes, 'Time (s)')
```

```
        ylabel(app.EulersMethodUIAxes, 'current')
        zlabel(app.EulersMethodUIAxes, 'Z')
        app.EulersMethodUIAxes.FontName = 'Calibri';
        app.EulersMethodUIAxes.FontSize = 13;
        app.EulersMethodUIAxes.ColorOrder = [0 0.4471 0.7412;0.851 0.3255 0.098;0.9294 0.6941
0.1255;0.4941 0.1843 0.5569;0.4667 0.6745 0.1882;0.302 0.7451 0.9333;0.6353 0.0784 0.1843];
        app.EulersMethodUIAxes.XGrid = 'on';
        app.EulersMethodUIAxes.YGrid = 'on';
        app.EulersMethodUIAxes.TitleFontSizeMultiplier = 1.3;
        app.EulersMethodUIAxes.Position = [82 351 472 314];

        % Create ode45UIAxes
        app.ode45UIAxes = uiaxes(app.FCSimulationResult1Tab);
        title(app.ode45UIAxes, 'Results Integrated by ode45()')
        xlabel(app.ode45UIAxes, 'Time (s)')
        ylabel(app.ode45UIAxes, 'current')
        zlabel(app.ode45UIAxes, 'Z')
        app.ode45UIAxes.FontName = 'Calibri';
        app.ode45UIAxes.FontSize = 13;
        app.ode45UIAxes.XGrid = 'on';
        app.ode45UIAxes.YGrid = 'on';
        app.ode45UIAxes.TitleFontSizeMultiplier = 1.3;
        app.ode45UIAxes.Position = [84 22 470 321];

        % Create EulerZoomSwitch
        app.EulerZoomSwitch = uiswitch(app.FCSimulationResult1Tab, 'slider');
        app.EulerZoomSwitch.Items = {'', ''};
        app.EulerZoomSwitch.ItemsData = [0 1];
        app.EulerZoomSwitch.ValueChangedFcn = createCallbackFcn(app,
@EulerZoomSwitchValueChanged, true);
        app.EulerZoomSwitch.FontColor = [0.149 0.149 0.149];
        app.EulerZoomSwitch.Position = [31 501 33 14];
        app.EulerZoomSwitch.Value = 0;

        % Create ode45ZoomSwitch
        app.ode45ZoomSwitch = uiswitch(app.FCSimulationResult1Tab, 'slider');
        app.ode45ZoomSwitch.Items = {'', ''};
        app.ode45ZoomSwitch.ItemsData = [0 1];
        app.ode45ZoomSwitch.ValueChangedFcn = createCallbackFcn(app,
@ode45ZoomSwitchValueChanged, true);
        app.ode45ZoomSwitch.Position = [31 177 33 14];
        app.ode45ZoomSwitch.Value = 0;

        % Create FCSimulationResult2Tab
```

```
        app.FCSimulationResult2Tab = uitab(app.TabGroup);
        app.FCSimulationResult2Tab.Title = 'FC-Simulation Result 2';
        app.FCSimulationResult2Tab.BackgroundColor = [0.949 0.9725 0.9804];
        app.FCSimulationResult2Tab.ForegroundColor = [0.0784 0.1882 0.4196];

        % Create CompareVoltagesUIAxes
        app.CompareVoltagesUIAxes = uiaxes(app.FCSimulationResult2Tab);
        title(app.CompareVoltagesUIAxes, 'Compare Vfc and Vo')
        xlabel(app.CompareVoltagesUIAxes, 'Time (s)')
        ylabel(app.CompareVoltagesUIAxes, 'Voltage (V)')
        zlabel(app.CompareVoltagesUIAxes, 'Z')
        app.CompareVoltagesUIAxes.FontName = 'Calibri';
        app.CompareVoltagesUIAxes.GridLineStyle = ':';
        app.CompareVoltagesUIAxes.XGrid = 'on';
        app.CompareVoltagesUIAxes.YGrid = 'on';
        app.CompareVoltagesUIAxes.TitleFontSizeMultiplier = 1.3;
        app.CompareVoltagesUIAxes.Position = [63 351 491 314];

        % Create CurrentRipplesUIAxes
        app.CurrentRipplesUIAxes = uiaxes(app.FCSimulationResult2Tab);
        title(app.CurrentRipplesUIAxes, 'Current ripples and local maximas and minimas')
        xlabel(app.CurrentRipplesUIAxes, 'Time (s)')
        ylabel(app.CurrentRipplesUIAxes, 'current')
        zlabel(app.CurrentRipplesUIAxes, 'Z')
        app.CurrentRipplesUIAxes.FontName = 'Calibri';
        app.CurrentRipplesUIAxes.TitleFontSizeMultiplier = 1.3;
        app.CurrentRipplesUIAxes.Position = [63 38 491 303];

        % Create TwoMethodVoltage
        app.TwoMethodVoltage = uiswitch(app.FCSimulationResult2Tab, 'slider');
        app.TwoMethodVoltage.Items = {'', ''};
        app.TwoMethodVoltage.ItemsData = [0 1];
        app.TwoMethodVoltage.ValueChangedFcn = createCallbackFcn(app,
@TwoMethodVoltageValueChanged, true);
        app.TwoMethodVoltage.Position = [21 600 22 10];
        app.TwoMethodVoltage.Value = 0;

        % Create TwoMethodCurrent
        app.TwoMethodCurrent = uiswitch(app.FCSimulationResult2Tab, 'slider');
        app.TwoMethodCurrent.Items = {'', ''};
        app.TwoMethodCurrent.ItemsData = [0 1];
        app.TwoMethodCurrent.ValueChangedFcn = createCallbackFcn(app,
@TwoMethodCurrentValueChanged, true);
        app.TwoMethodCurrent.Position = [21 259 22 10];
```

```
        app.TwoMethodCurrent.Value = 0;

        % Show the figure after all components are created
        app.FinalProjectUIFigure.Visible = 'on';
        end
        end

        % App creation and deletion
        methods (Access = public)

        % Construct app
        function app = CompSim_FinalProjectGUI_Combo

        % Create UIFigure and components
        createComponents(app)

        % Register the app with App Designer
        registerApp(app, app.FinalProjectUIFigure)

        if nargout == 0
        clear app
        end
        end

        % Code that executes before app deletion
        function delete(app)

        % Delete UIFigure when app is deleted
        delete(app.FinalProjectUIFigure)
        end
        end
end
```