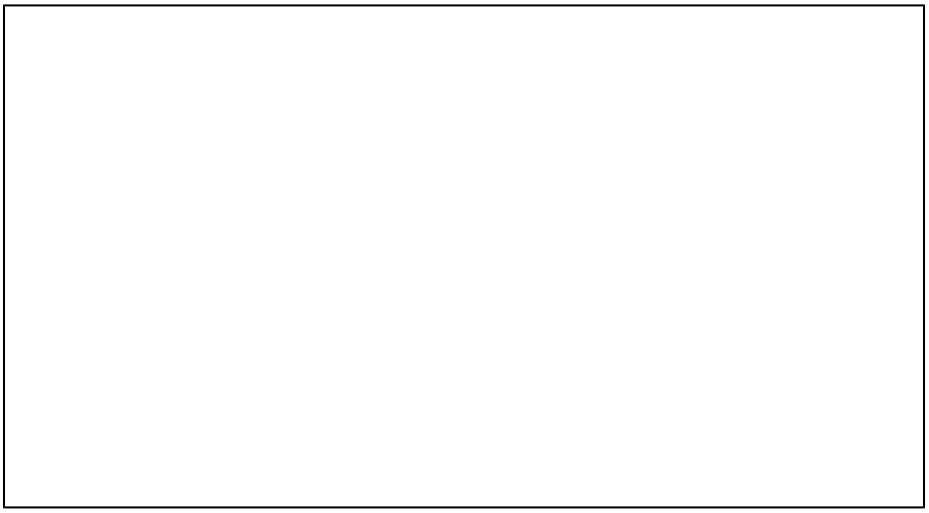
Python Scritpting with AWRDE: 2D and 3D Plotting with Swept Variables

Introduction

This article offers a Python module that can be used to plot 2-dimensional and 3-dimentional graphs in Python. This module can be used as a starting place that can be tailored to your particular needs.

Example Case

The plots shown in this article are based on the following low pass filter circuit. Three of the elements have their values swept using the SWPVAR element.



The AWRDE graphs show S-parameters vs. frequency. The multiple traces correspond to each swept variable iteration.

Graphing Utilities

All the functions required for reading measurent data from the AWRDE graphs and creating the Python graphs are in the file **mwo_2d3d_graphing.py**. The code in its entirety is contained here:

mwo_2d3d_graphing.py

```
import numpy as np
import matplotlib.pyplot as plt

class GraphMethods():
    def __init__(self, awrde, colors):#-----------------------------------------------------------
        self.project = awrde.Project
        self.awrde = awrde
        self.colors = colors
        print('Project Name: ' + self.project.Name)
        #
    def _SetupMeasurements(self, GraphName, MeasurementName_list):
#-------------------------------------------------------
        """
        Validates that GraphName in in the MWO project
        Converts MeasurementName_list which can be a list of a portion of the measurement name into
        a list of measurements. Each list item is a list of full measurement name and which y-axis

        Parameters
        ----------
        GraphName : Desired graph name (input from user)
        MeasurementName_list : List generated by the user. This is the measurments from the graph
                               to be included. The names can be partial, just enough text to
```

```
                            distiguish one measurement name from another.

        Returns
        -------
        graph : object variable for the AWRDE project graph
        MeasurementName_list : list [Full Measurement Name (string), OnLeftAxis (boolean)]
        """
        #
        #Ensure Graph existis in the project
        try:
            graph = self.project.Graphs(GraphName)
        except:
            raise RuntimeError('Invalid Graph Name: '+GraphName)
        #end try
        #
        YaxisR = False  #Flag. If any of the measurements from the MWO graph use the right y-axis
                            #then this flag gets set True

        #Create list of full measurement name. Used to convert the partial name to full name. Also determine
        #if trace is one the left or right y-axis (rectangular plots only)
        TempMeasName_list = list()
        NumMeasurements = graph.Measurements.Count
        for MeasName in MeasurementName_list:
            for i in range(1,NumMeasurements + 1):
                if MeasName in graph.Measurements(i).Name:
                    TempMeasName_list.append([graph.Measurements(i).Name, graph.Measurements(i).OnLeftAxis])
                    if not graph.Measurements(i).OnLeftAxis:
                        YaxisR = True
                    #end if
                #end if
            #end for
        #end for
        if len(TempMeasName_list) == 0:
            raise RuntimeError('No measurements found')
        #end if
        MeasurementName_list = TempMeasName_list #[Full Measurement Name, OnLeft boolean]
        GraphTitle = graph.Title
        return graph, GraphTitle, MeasurementName_list, YaxisR
        #
    def _GetSweepParameters(self, graph, MeasurementName_list):
#-----------------------------------------------------------
        """
        Retrieves all the SWPVAR variables from all the measurements in the graph.
        One restriction is that all measurement contain the same SWPVAR variables and that the
        SWPVAR variable ranges are the same.

        Parameters
        ----------
        graph : object variable for the AWRDE project graph
        MeasurementName_list : list [Full Measurement Name (string), OnLeftAxis (boolean)]

        Returns
        -------
        SWPVAR_Name_list : list of the SWPVAR variable names (string)
        SWPVAL_Value_list : list of lists. Each list contains the SWPVAR values the correspond to the
                            SWPVAR_Name_list
        Trace_SWPVAR_Value_list : list of lists. Each list contains the SWPVAR values for each
                                  measurement trace
        """
        #
        debug_on = False
        NumMeasurements = len(MeasurementName_list)
        MeasSource_list = list()
        SWPVAR_Name_list = list()
        Trace_SWPVAR_Value_list = list()
        #
        for meas_idx in range(1, NumMeasurements+1):
            meas = graph.Measurements(meas_idx)
            MeasSource_list.append(meas.Source)
            NumSWPVARs = meas.SweepLabels(1).Count
            NumTraces = meas.TraceCount
```

```
            Temp_SWPVAR_Value_list = list()
            for trace_idx in range(1, NumTraces+1):
                TempTemp_SWPVAR_Value_list = list()
                for swpvar_idx in range(1, meas.SweepLabels(1).Count+1):
                    if trace_idx==1 and meas_idx==1:
                        SWPVAR_Name_list.append(meas.SweepLabels(1).Item(swpvar_idx).Name)
                    #end if
                    TempTemp_SWPVAR_Value_list.append(meas.SweepLabels(trace_idx).Item(swpvar_idx).Value)
                #end for
                Temp_SWPVAR_Value_list.append(TempTemp_SWPVAR_Value_list)
            #end for
            Trace_SWPVAR_Value_list.append(Temp_SWPVAR_Value_list)
        #end for
        #
        for meas_idx in range(1, NumMeasurements):
            if MeasSource_list[meas_idx] != MeasSource_list[0]:
                raise RuntimeError('Measurement Source Document must be the same for all measurements')
            #end if
            if Trace_SWPVAR_Value_list[meas_idx] != Trace_SWPVAR_Value_list[0]:
                raise RuntimeError('SWPVAR ranges must be the same for all measurements')
            #end if
        #end for
        Trace_SWPVAR_Value_list = Trace_SWPVAR_Value_list[0]
        #
        SWPVAL_Value_list = list()
        NumSWPVARs = len(SWPVAR_Name_list)
        NumTraces = len(Trace_SWPVAR_Value_list)
        for swpvar_idx in range(NumSWPVARs):
            Temp_val_list = list()
            for trace_idx in range(NumTraces):
                swpvar_val = Trace_SWPVAR_Value_list[trace_idx][swpvar_idx]
                if trace_idx == 0:
                    Temp_val_list.append(swpvar_val)
                else:
                    if swpvar_val not in Temp_val_list:
                        Temp_val_list.append(swpvar_val)
                    #end if
                #end if
            #end for
            SWPVAL_Value_list.append(Temp_val_list)
        #end for
        if debug_on:
            print(SWPVAL_Value_list)
            print(Trace_SWPVAR_Value_list)
            print(SWPVAR_Name_list)
        #end if
        return SWPVAL_Value_list, Trace_SWPVAR_Value_list, SWPVAR_Name_list
        #
    def _GetTraceData(self, graph, MeasurementName_list):#---------------------------------
        """
        Reads the trace data for each measurement

        Parameters
        ----------
        graph : object variable for the AWRDE project graph
        MeasurementName_list : list [Full Measurement Name (string), OnLeftAxis (boolean)]

        Returns
        -------
        TraceData_ay_list : list of arrays. Each array contains x-axis data, y-axis data,
                            y2-axis data
                            y2-axis data only exists if the trace data is complex

        Highlighted_idx : Inititial index that will be used to highligt the plotted data.
                          Onely one trace is highlighted, the rest are drawn faded

        """
        #
        NumMeas = len(MeasurementName_list)
        TraceData_ay_list = list()
        NumTraces = graph.Measurements(1).TraceCount
```

```
        for meas_idx in range(NumMeas):
            meas = graph.Measurements(MeasurementName_list[meas_idx][0])
            TempTraceData_ay_list = list()
            for trace_idx in range(1, NumTraces+1):
                MeasData_Tuple = meas.TraceValues(trace_idx)
                MeasData_ay = np.asarray(MeasData_Tuple)
                TempTraceData_ay_list.append(MeasData_ay)
            #end for
            TraceData_ay_list.append(TempTraceData_ay_list)
        #end for
        #
        if NumTraces==1:
            Highlighted_idx = 0
        else:
            Highlighted_idx = int(np.ceil(NumTraces/2))  #Set to mid point
        #end if
        return TraceData_ay_list, Highlighted_idx
        #
    def _Get_AxisLimits(self, graph, graph_type=None):#-----------------------------------------------------
        """
        Reads the x and y-axis limits from the AWRDE graph. These are initial starting values
        for the Python graph. The user can override these values using the Set_Axis_Scale()
        method

        Parameters
        ----------
        graph : object variable for the AWRDE graph
        graph_type : string value. Either 'Smith Chart' or 'Rectangular'

        Returns
        ------
        PlotScale_list : list of tuples. One tuple for each axis: x, y-left, y-right, z
                         Each tuple contains
                             axis min value (float)
                             axis max value (float)
                             number of axis divisions (float)
                             units -- uses for scaling the data to the min and max axis units (float)
                             axis label (string)
        """
        #
        YaxisR_Scale = ()
        if graph_type == 'Smith Chart':
            xy_Lim = 1.2
            Num_xy_Div = 3
            Xaxis_Scale = (-xy_Lim, xy_Lim, Num_xy_Div, 1, 'rho')
            Yaxis_Scale = (-xy_Lim, xy_Lim, Num_xy_Div, 1, 'rho')
            #
            #For 3D Smith Charts, the AWRDE x-axis is used for the Python graph z-axis
            Zmin = float(graph.Attributes('SweepMinimum').ValueAsString)
            Zmax = float(graph.Attributes('SweepMaximum').ValueAsString)
            Zaxis_Scale = (Zmin, Zmax, 10, 1, 'Frequency [Hz]')
            #
        elif graph_type == 'Rectangular':
            for axis_idx in range(3):
                MinScale = graph.Axes[axis_idx].MinimumScale
                MaxScale = graph.Axes[axis_idx].MaximumScale
                StepSize = graph.Axes[axis_idx].MajorGridlinesStep
                NumDiv = int(np.ceil((MaxScale - MinScale)/StepSize))
                if axis_idx == 0:
                    if graph.Attributes('UseDefaultXLabel').ValueAsString == '-1':
                        Label = graph.Attributes('DefaultXLabel').ValueAsString
                        if 'Frequency' in Label:
                            if '(GHz)' in Label:
                                UnitVal = 1e9
                            elif '(MHz)' in Label:
                                UnitVal = 1e6
                            elif '(kHz)' in Label:
                                UnitVal = 1e3
                            else:
                                UnitVl = 1
                            #end if
```

```
                    #end if
                else:
                    Label = graph.Axes[axis_idx].LabelText
                    UnitVal = 1
                #end if
                Xaxis_Scale = (MinScale, MaxScale, NumDiv, UnitVal, Label)
                #
            elif axis_idx == 1: #Left y-axis
                Label = graph.Axes[axis_idx].LabelText
                Yaxis_Scale = (MinScale, MaxScale, NumDiv, UnitVal, Label)
                #
            elif axis_idx == 2: #Right y-axis
                Label = graph.Axes[axis_idx].LabelText
                YaxisR_Scale = (MinScale, MaxScale, NumDiv, UnitVal, Label)
                #
            #end if
        #end for
        #
        meas = graph.Measurements[0]
        NumTraces = meas.TraceCount
        Zaxis_Scale = (0, NumTraces, 10, 1, '')
    #end if
    PlotScale_list = [Xaxis_Scale, Yaxis_Scale, YaxisR_Scale, Zaxis_Scale]
    return PlotScale_list
    #
def _SetupPlot(self, PlotScale_list, GraphTitle, PlotType='Rectangular', Projection='2D'):#------
    """
    Initializes the Python graph

    Parameters
    ----------
     PlotScale_list : list of tuples. One tuple for each axis: x, y-left, y-right, z
                    Each tuple contains
                        axis min value (float)
                        axis max value (float)
                        number of axis divisions (float)
                        units -- uses for scaling the data to the min and max axis units (float)
                        axis label (string)
    GraphTitle : Python Graph Title (string)
    PlotType : Rectangular | Smith Chart (string)
    Projection : 2D | 3D (string)

    Returns
    -------
    ax : object variable for the Python axis object used for ploting left y-axis data
    ax2 : object variable for the Python axis object used for ploting right y-axis data
    fig : object variable for Python grpah figure object

    """
    #
    if PlotType == 'Rectangular':
        FigSize = (12,8)
    elif PlotType == 'Smith Chart':
        FigSize = (9,9)
    #end if
    #
    fig = plt.figure(1, facecolor=self.colors.light_grey, edgecolor=self.colors.black)
    fig.set_size_inches(FigSize, forward = True)
    if Projection == '2D':
        ax = fig.add_axes([0.1,0.1,0.8,0.8], frameon=True, facecolor=self.colors.white)
    elif Projection == '3D':
        ax = fig.add_subplot(projection='3d')
        ax.set_facecolor(self.colors.LightBlue_pale)
    #end if
    fig.canvas.mpl_connect('key_press_event', self._KBD_Press)
    if self.YaxisR and self.Projection=='2D':
        ax2 = ax.twinx()
    else:
        ax2 = None
    #end if
    self._CreateAxes(PlotScale_list, GraphTitle, ax, ax2, PlotType, Projection)
```

```
        return ax, ax2, fig
        #
    def Set_Axis_Scale(self, Xaxis_Scale=None, Yaxis_Scale=None, YaxisR_Scale=None, Zaxis_Scale=None):
#------------------
        """
        Sets the graph axis paramters. Called by the user. Overrides the initial axis scale settings
        that are read from the AWRDE graph

        Parameters
        ----------
           Xaxis_Scale : Applies only to rectangular graphs
           Yaxis_Scale : Applies only to rectangular graphs. Left y-axis
           YaxisR_Scale : Applies only to rectangular graphs. Right y-axis
           Zaxis_Scale: Applies only to 3D graphs

           All parameters are defined in a Tuple: (Min, Max, Num Divisions, Units, Title)
              Min and Max use base units
              Units parameter is the base units divider
              Title : string value
        """
        if Xaxis_Scale != None:
            Xaxis_Scale = Xaxis_Scale
            self._PlotScale_list[0] = Xaxis_Scale
        #end if
        if Yaxis_Scale != None:
            Yaxis_Scale = Yaxis_Scale
            self._PlotScale_list[1] = Yaxis_Scale
        #end if
        if YaxisR_Scale != None:
            YaxisR_Scale = YaxisR_Scale
            self._PlotScale_list[2] = YaxisR_Scale
        #end if
        if Zaxis_Scale != None:
            Zaxis_Scale = Zaxis_Scale
            self._PlotScale_list[3] = Zaxis_Scale
        #end if
        #
        #Clear grid, axes from graph
        self.ax.cla()
        if self.ax2 != None:
            self.ax2.cla()
        #end if
        #
        #Draw new axes on graph
        self._CreateAxes(self._PlotScale_list, self.GraphTitle, self.ax, self.ax2, self.PlotType, self.
Projection)
        #
    def PlotTraces(self):#-------------------------------------------------------------
        """
        Draws the traces on the graph
        """
        #
        NumMeas = len(self.TraceData_ay_list)
        NumTraces = len(self.TraceData_ay_list[0])
        #
        Xaxis_Scale = self._PlotScale_list[0]
        Yaxis_Scale = self._PlotScale_list[1]
        YaxisR_Scale = self._PlotScale_list[2]
        #
        Xmin = Xaxis_Scale[0]
        Xmax = Xaxis_Scale[1]
        Xunits = Xaxis_Scale[3]
        #
        Ymin = Yaxis_Scale[0]
        Ymax = Yaxis_Scale[1]
        Yunits = Yaxis_Scale[3]
        #
        if self.YaxisR:
            YminR = YaxisR_Scale[0]
            YmaxR = YaxisR_Scale[1]
            YunitsR = YaxisR_Scale[3]
```

```
        #end if
        #
        if self.PlotType == 'Smith Chart' and self.Projection=='3D':
            Zaxis_Scale = self._PlotScale_list[3]
            Zmin = Zaxis_Scale[0]
            Zmax = Zaxis_Scale[1]
            Zunits = Zaxis_Scale[3]
        #end if
        bbox = ([Xmin,Xmax],[Ymin,Ymax])
        Legend_list = list()
        #
        for meas_idx in range(NumMeas):
            LineColor = self.colors.Colors_list[meas_idx]
            for trace_idx in range(NumTraces):

                if trace_idx == self.Highlighted_idx:
                    AlphaVal = 1
                else:
                    AlphaVal = 0.1
                #end if
                if self.PlotType == 'Rectangular':
                    Xdata = self.TraceData_ay_list[meas_idx][trace_idx][:,0]/Xunits
                    Ydata = self.TraceData_ay_list[meas_idx][trace_idx][:,1]/Yunits
                    #
                    if self.Projection == '2D':
                        if self.MeasurementName_list[meas_idx][1]:
                            Ydata = np.array(list(map(lambda x: max(x, Ymin), Ydata)))
                            Ydata = np.array(list(map(lambda x: min(x, Ymax),
Ydata)))
                            L, = self.ax.plot(Xdata, Ydata, color=LineColor, alpha=AlphaVal)
                        else:
                            Ydata = np.array(list(map(lambda x: max(x, YminR), Ydata)))
                            Ydata = np.array(list(map(lambda x: min(x, YmaxR),
Ydata)))
                            L, = self.ax2.plot(Xdata, Ydata, color=LineColor, alpha=AlphaVal)
                        #end if
                        if trace_idx == self.Highlighted_idx:
                            Legend_list.append([L, self.MeasurementName_list[meas_idx][0]])
                        #end if
                        #
                    elif self.Projection == '3D':
                        z_val = trace_idx
                        Ydata = np.array(list(map(lambda x: max(x, Ymin), Ydata)))
                        Ydata = np.array(list(map(lambda x: min(x, Ymax),
Ydata)))
                        self.ax.plot(Xdata, Ydata, zs=z_val, zdir='z', color=LineColor,\
                                     alpha=AlphaVal, clip_on=True, clip_box=bbox)
                    #end if
                elif self.PlotType == 'Smith Chart':
                    if self.Projection == '2D':
                        Xdata = self.TraceData_ay_list[meas_idx][trace_idx][:,1]
                        Ydata = self.TraceData_ay_list[meas_idx][trace_idx][:,2]
                        L, = self.ax.plot(Xdata, Ydata, color=LineColor, alpha=AlphaVal)
                        if trace_idx == self.Highlighted_idx:
                            Legend_list.append([L, self.MeasurementName_list[meas_idx][0]])
                        #end if
                    elif self.Projection == '3D':
                        Xdata = self.TraceData_ay_list[meas_idx][trace_idx][:,1]
                        Ydata = self.TraceData_ay_list[meas_idx][trace_idx][:,2]
                        Zdata = self.TraceData_ay_list[meas_idx][trace_idx][:,0]/Zunits
                        L, = self.ax.plot(Xdata, Ydata, Zdata, color=LineColor, alpha=AlphaVal)
                        if trace_idx == self.Highlighted_idx:
                            Legend_list.append([L, self.MeasurementName_list[meas_idx][0]])
                        #end if
                    #end if
                #end if
            #end for
        #end for
        #
        Line_list = list()
        Label_list = list()
```

```
        for i in range(len(Legend_list)):
            Line_list.append(Legend_list[i][0])
            Label_list.append(Legend_list[i][1])
        #end for
        self.ax.legend(Line_list, Label_list, loc='lower right')
        #
    def _KBD_Press(self, event):#-------------------------------------------------------------
        """
        Keyboard press event handler. Only supports the up and down arrow keyboard presses. These
        are used to change the highlighted trace.

        Highlighted_idx variable is incremented for each up/down arrow keyboard press. The graph is cleared
        and new graph data is generated with the new highlighted trace.

        Also prints the SWPVAR variable names and values for the highlighted trace
        """
        #
        if event.key == 'up':
            self.Highlighted_idx += 1
        elif event.key == 'down':
            self.Highlighted_idx -= 1
        #end if
        self.Highlighted_idx = int(max(0, self.Highlighted_idx))
        self.Highlighted_idx = int(min(len(self.TraceData_ay_list[0])-1, self.Highlighted_idx))
        self.ax.cla()
        if self.ax2 != None:
            self.ax2.cla()
        #end if
        self._CreateAxes(self._PlotScale_list, self.GraphTitle, self.ax, self.ax2, self.PlotType, self.
Projection)
        self.PlotTraces()
        self.fig.canvas.draw()
        Print_str = ''
        for i in range(len(self.SWPVAR_Name_list)):
            VarName = self.SWPVAR_Name_list[i]
            VarVal = self.Trace_SWPVAR_Value_list[self.Highlighted_idx][i]
            Print_str += VarName + ': '+str(VarVal) + '  '
        #end for
        print(Print_str)
        #
    def ShowPlot(self):#-----------------------------------------------------
        plt.show()
        #
    def _CreateAxes(self, PlotScale_list, GraphTitle, ax, ax2=None, PlotType='Rectangular', Projection='2D'):
#-------------------------------------------------------
        """
        Sets the limits for the graph axes.
        Sets the tick marks (number of divisions)
        Adds the axis labels

        Parameters
        ----------
        PlotScale_list : list of tuples. One tuple for each axis: x, y-left, y-right, z
                         Each tuple contains
                             axis min value (float)
                             axis max value (float)
                             number of axis divisions (float)
                             units -- uses for scaling the data to the min and max axis units (float)
                             axis label (string)
        GraphTitle : Python Graph Title (string)
        ax : object variable for the Python axis object used for ploting left y-axis data
        ax2 : object variable for the Python axis object used for ploting right y-axis data
        PlotType : Rectangular | Smith Chart (string)
        Projection : 2D | 3D (string)
        """
        #
        Xaxis_Scale = PlotScale_list[0]
        Yaxis_Scale = PlotScale_list[1]
        YaxisR_Scale = PlotScale_list[2]
        Zaxis_Scale = PlotScale_list[3]
        #
```

```
        ax.set_title(GraphTitle, fontsize=22, fontweight='bold')
        if PlotType == 'Rectangular':
            Xmin = Xaxis_Scale[0]
            Xmax = Xaxis_Scale[1]
            Xdiv = Xaxis_Scale[2]
            Xtic = np.linspace(Xmin, Xmax, Xdiv+1)
            Xlabel = Xaxis_Scale[4]
            #
            Ymin = Yaxis_Scale[0]
            Ymax = Yaxis_Scale[1]
            Ydiv = Yaxis_Scale[2]
            Ytic = np.linspace(Ymin, Ymax, Ydiv+1)
            Ylabel = Yaxis_Scale[4]
            #
            ax.set_xlim(Xmin, Xmax)
            ax.set_xlabel(Xlabel, fontsize=16)
            ax.set_xticks(Xtic)
            ax.set_ylim(Ymin, Ymax)
            ax.set_ylabel(Ylabel, fontsize=16)
            ax.set_yticks(Ytic)
            #
            if self.YaxisR and Projection=='2D':
                YminR = YaxisR_Scale[0]
                YmaxR = YaxisR_Scale[1]
                YdivR = YaxisR_Scale[2]
                YticR = np.linspace(YminR, YmaxR, YdivR+1)
                YlabelR = YaxisR_Scale[4]
                #
                ax2.set_ylim(YminR, YmaxR)
                ax2.set_ylabel(YlabelR, fontsize=16)
                ax2.set_yticks(YticR)
            #end if
            #
            if Projection=='2D':
                ax.grid(linestyle=':')
                #
            elif Projection=='3D':
                Zmin = Zaxis_Scale[0]
                Zmax = Zaxis_Scale[1]
                Zdiv = Zaxis_Scale[2]
                Ztic = np.linspace(Zmin, Zmax, Zdiv+1)
                Zlabel = Zaxis_Scale[4]
                #
                ax.set_zlim(Zmin, Zmax)
                ax.set_zlabel(Zlabel, fontsize=16)
                ax.set_zticks(Ztic)
                #
                self._Draw_Rect3D_Grid(ax, PlotScale_list, z_plane='highlighted')
            #end if
            #
        elif PlotType == 'Smith Chart':
            ax.set_xlim(Xaxis_Scale[0], Xaxis_Scale[1])
            ax.set_xlabel(Xaxis_Scale[4], fontsize=16)
            #
            ax.set_ylim(Yaxis_Scale[0], Yaxis_Scale[1])
            ax.set_ylabel(Yaxis_Scale[4], fontsize=16)
            #
            if Projection=='3D':
                Zmin = Zaxis_Scale[0]
                Zmax = Zaxis_Scale[1]
                Num_Zdiv = Zaxis_Scale[2]
                Zticks = np.linspace(Zmin, Zmax, Num_Zdiv+1)
                ax.set_zlim(Zmin, Zmax)
                ax.set_zticks(Zticks)
                ax.set_zlabel(Zaxis_Scale[4], fontsize=16)
            #end if
            self._CreateSmithChartGrid(ax, PlotScale_list, Projection)
        #end if
        #
    def _Draw_Rect3D_Grid(self, ax, PlotScale_list, z_plane='highlighted'):
#-------------------------------------------
```

```
    """
    Draws rectangular grid for rectangular 3D graphs

    Parameters
    ----------
    ax : object variable for the Python axis object used for ploting left y-axis data
    PlotScale_list : list of tuples. One tuple for each axis: x, y-left, y-right, z
                    Each tuple contains
                        axis min value (float)
                        axis max value (float)
                        number of axis divisions (float)
                        units -- uses for scaling the data to the min and max axis units (float)
                        axis label (string)
    z_plane : 'highlighted' | 'all'   (string)
              all: draws a xy grid on each z-plane defined by zticks
              highlighted: draws xy grid only on the highlighted z-plane
    """
    #
    Xaxis_Scale = PlotScale_list[0]
    Yaxis_Scale = PlotScale_list[1]
    YaxisR_Scale = PlotScale_list[2]
    Zaxis_Scale = PlotScale_list[3]
    #
    Xmin = Xaxis_Scale[0]
    Xmax = Xaxis_Scale[1]
    Xdiv = Xaxis_Scale[2] + 1
    Ymin = Yaxis_Scale[0]
    Ymax = Yaxis_Scale[1]
    Ydiv = Yaxis_Scale[2] + 1
    Zmin = Zaxis_Scale[0]
    Zmax = Zaxis_Scale[1]
    #
    Xtic = np.linspace(Xmin, Xmax, Xdiv)
    Ytic = np.linspace(Ymin, Ymax, Ydiv)
    if z_plane == 'all':
        Num_Z_Planes = self.Zaxis_Scale[2] + 1
        Ztic = np.linspace(Zmin, Zmax, Num_Z_Planes)
        alpha_val = 0.1
    elif z_plane == 'highlighted':
        Num_Z_Planes = 1
        Ztic = [self.Highlighted_idx]
        alpha_val = 0.1
    #end if
    #
    for z_idx in range(Num_Z_Planes):
        z_val = Ztic[z_idx]
        for x_idx in range(Xdiv):
            Xdata = [Xmin, Xmax]
            for y_idx in range(Ydiv):
                Ydata = [Ytic[y_idx], Ytic[y_idx]]
                ax.plot(Xdata, Ydata, zs=z_val, zdir='z', color=self.colors.grey, alpha=alpha_val)
            #end for
        #end for
        for y_idx in range(Ydiv):
            Ydata = [Ymin, Ymax]
            for x_idx in range(Xdiv):
                Xdata = [Xtic[x_idx], Xtic[x_idx]]
                ax.plot(Xdata, Ydata, zs=z_val, zdir='z', color=self.colors.grey, alpha=alpha_val)
            #end for
        #end for
    #end for
    #
def _GenerateSmithChartGridData(self):#-------------------------------------------------------
    """
    Generates the x,y data (termed u,v) for the Smitch Chart grid circles for both the r-circles
    and the x-circles

    Returns
    -------
    OuterCircle_list : Smith Chart outer circle (rho = 1). Xdata array and Ydata array
    R_Circle_list : X and Y data for the Smith Chart R circles
```

```
        X_Circle_list : X and Y data for the Smith Chart X circles
        r_circle_radius_list : list of Smith Chart r-circle radii
        x_circle_radius_list : list of Smith Chart x-circle radii
        """
        #
        r_circle_radius_list = [0.2, 0.5, 1, 1.7, 3, 5, 10]
        x_circle_radius_list = [0.2, 0.5, 1, 1.5, 3]
        deg_step_list = [0.5,1,1,2,2]
        #
        #Outer Circle
        Center = [0,0]
        Radius = 1
        [Udata, Vdata] = self._Create_uv_arrays(Center, Radius, 5)
        OuterCircle_list = [Udata, Vdata]
        #
        #r Circles
        Udata_ay_list = list()
        Vdata_ay_list = list()
        Num_r_pts = len(r_circle_radius_list)
        for i in range(Num_r_pts):
            r = r_circle_radius_list[i]
            Center = [r/(r+1),0]
            Radius = 1/(r+1)
            [Udata, Vdata] = self._Create_uv_arrays(Center, Radius, 5)
            Udata_ay_list.append(Udata)
            Vdata_ay_list.append(Vdata)
        #end for
        R_Circle_list = [Udata_ay_list, Vdata_ay_list]
        #
        #x Circles
        Udata_ay_list = list()
        Vdata_ay_list = list()
        Num_x_pts = len(x_circle_radius_list)
        if len(deg_step_list) != Num_x_pts:
            raise RuntimeError("Num deg_step_list must equal Num_x_pts")
        #end if
        for i in range(Num_x_pts):
            x = x_circle_radius_list[i]
            for j in range(2):
                if j == 1:
                    x *= (-1)
                #end if
                Center = [1, 1/x]
                Radius = abs(1/x)
                [Udata, Vdata] = self._Create_uv_arrays(Center, Radius, deg_step_list[i])
                Udata_ay_list.append(Udata)
                Vdata_ay_list.append(Vdata)
            #end for
        #end for
        X_Circle_list = [Udata_ay_list, Vdata_ay_list]
        #
        return OuterCircle_list, R_Circle_list, X_Circle_list, r_circle_radius_list, x_circle_radius_list
        #
    def _CreateSmithChartGrid(self, ax, PlotScale_list, Projection='2D'):
#-----------------------------------------
        """
        Draws the r and x cilcles on the Smith Chart

        Parameters
        ----------

        ax : object variable for the Python axis object used for ploting left y-axis data
        PlotScale_list : list of tuples. One tuple for each axis: x, y-left, y-right, z
                         Each tuple contains
                             axis min value (float)
                             axis max value (float)
                             number of axis divisions (float)
                             units -- uses for scaling the data to the min and max axis units (float)
                             axis label (string)
        Projection : '2D' | '3D' (string)
        """
```

```
        if Projection == '2D':
            Num_Zplanes = 1
            ZplaneVal_ay = [1]
            #
        elif Projection == '3D':
            Zaxis_Scale = PlotScale_list[3]
            Zmin = Zaxis_Scale[0]
            Zmax = Zaxis_Scale[1]
            Num_Zplanes = 3
            ZplaneVal_ay = np.linspace(Zmin, Zmax, Num_Zplanes)
        #end if
        #
        [OuterCircle_list, R_Circle_list, X_Circle_list, r_circle_radius_list, x_circle_radius_list] = \
            self._GenerateSmithChartGridData()
        #
        for z_idx in range(Num_Zplanes):
            z_val = ZplaneVal_ay[z_idx]
            if (z_idx==0) or (z_idx==Num_Zplanes-1):
                alpha_val = 1
            else:
                alpha_val = 0.2
            #end if
            #
            Udata = OuterCircle_list[0]
            Vdata = OuterCircle_list[1]
            if Projection == '2D':
                ax.plot(Udata, Vdata, color=self.colors.black, linewidth=1, alpha=alpha_val)
            elif Projection == '3D':
                ax.plot(Udata, Vdata, zs=z_val, zdir='z', color=self.colors.black, linewidth=1, alpha=alpha_val)
            #end if
            #
            Udata_ay_list = R_Circle_list[0]
            Vdata_ay_list = R_Circle_list[1]
            Num_R_Circles = len(Udata_ay_list)
            for r_idx in range(Num_R_Circles):
                Udata = Udata_ay_list[r_idx]
                Vdata = Vdata_ay_list[r_idx]
                if Projection == '2D':
                    ax.plot(Udata, Vdata, color=self.colors.grey, linewidth=1, alpha=alpha_val)
                elif Projection == '3D':
                    ax.plot(Udata, Vdata, zs=z_val, zdir='z', color=self.colors.grey, linewidth=1,
alpha=alpha_val)
                #end if
            #end for
            #
            Udata_ay_list = X_Circle_list[0]
            Vdata_ay_list = X_Circle_list[1]
            Num_X_Circles = len(Udata_ay_list)
            for x_idx in range(Num_X_Circles):
                Udata = Udata_ay_list[x_idx]
                Vdata = Vdata_ay_list[x_idx]
                if Projection == '2D':
                    ax.plot(Udata, Vdata, color=self.colors.grey, linewidth=1, alpha=alpha_val)
                elif Projection == '3D':
                    ax.plot(Udata, Vdata, zs=z_val, zdir='z', color=self.colors.grey, linewidth=1,
alpha=alpha_val)
                #end if
            #end for
            if Projection == '2D':
                ax.plot([-1,1], [0,0], color=self.colors.grey, linewidth=1, alpha=alpha_val)
                self._Annotate_SmithChart(ax, r_circle_radius_list, x_circle_radius_list, z_val='none')
            elif Projection == '3D':
                ax.plot([-1,1], [0,0], zs=z_val, zdir='z', color=self.colors.grey, linewidth=1, alpha=alpha_val)
                self._Annotate_SmithChart(ax, r_circle_radius_list, x_circle_radius_list, z_val=z_val)
            #end if
            #
        #end for
        #
        if Projection=='3D':
            NumVertLines = 12
            DegStep = 2*np.pi/NumVertLines
```

```
            Deg = 0
            for v_idx in range(NumVertLines):
                x = np.cos(Deg)
                y = np.sin(Deg)
                Xdata = [x,x]
                Ydata = [y,y]
                Zdata = [Zmin, Zmax]
                ax.plot(Xdata, Ydata, Zdata, color=self.colors.grey, alpha=0.4)
                Deg += DegStep
            #end for
        #end if
        #
    def _Annotate_SmithChart(self, ax,  r_circle_radius_list, x_circle_radius_list, z_val='none'):
#-------------------------------------------------
        """
        Add r and x circle numeric labels to the Smith Chart grid

        Parameters
        ----------
        ax : object variable for the Python axis object used for ploting left y-axis data
        r_circle_radius_list : list of Smith Chart r-circle radii
        x_circle_radius_list : list of Smith Chart x-circle radii
        z_val : 'none' | z-axis value
                'none' : used for 2D Smith Chart
                value  : z-axis value for adding annotation text
        """
        alpha_val = 0.9
        #
        #R circles
        Circle_list = [0] + r_circle_radius_list
        NumCircles = len(Circle_list)
        for c_idx in range(NumCircles):
            r = Circle_list[c_idx]
            Center = [r/(r+1),0]
            Radius = 1/(r+1)
            x = Center[0] - Radius - 0.03
            y = Center[1] + 0.01
            if abs(x) > 0.98:
                if x < -0.98:
                    x += 0.05
                else:
                    x -= 0.05
                #end if
            #end if
            if z_val == 'none':
                ax.annotate(str(r), xy=(x,y), rotation=90, color=self.colors.grey, fontsize=8, alpha=alpha_val)
            else:
                ax.text(x, y, z_val, str(r), 'x', color=self.colors.grey, fontsize=8, alpha=alpha_val)
            #end if
        #end for
        #
        #X circles
        Circle_list = x_circle_radius_list
        NumCircles = len(Circle_list)
        for c_idx in range(NumCircles):
            x_cir = Circle_list[c_idx]
            for j in range(2):
                if j == 0:
                    Theda_ay = np.arange(270, 0 , -1)
                elif j == 1:
                    x_cir = (-1)*x_cir
                    Theda_ay = np.arange(95, 360 , 1)
                #end if
                Center = [1, 1/x_cir]
                Radius = abs(1/x_cir)
                Theda_rad_ay = np.deg2rad(Theda_ay)
                for theda in Theda_rad_ay:
                    x = Center[0] + Radius*np.cos(theda)
                    y = Center[1] + Radius*np.sin(theda)
                    if np.sqrt(x**2 + y**2) > 1:
                        break
```

```
                #end if
            #end for
            Radius_sf = 0.95
            rho_mag = np.sqrt(x**2 + y**2)
            rho_ang = np.arctan2(y,x)
            if x_cir>-0.7 and x_cir<0:
                rho_ang = np.deg2rad(np.rad2deg(rho_ang)+3)
            #end if
            x = Radius_sf*rho_mag*np.cos(rho_ang)
            y = Radius_sf*rho_mag*np.sin(rho_ang)
            anno_theda = np.rad2deg(theda)
            if z_val == 'none':
                ax.annotate(str(x_cir), xy=(x,y), rotation=anno_theda, color=self.colors.grey, fontsize=8)
            else:
                ax.text(x, y, z_val, str(x_cir), 'x', color=self.colors.grey, fontsize=8, alpha=alpha_val)
            #end if
        #end for
    #end for
    #
def _Create_uv_arrays(self, Center, Radius, DegStep):#-------------------------
    """
    Creates x and y data for a circle

    Parameters
    ----------
    Center : list of x,y values for the center of the circle
    Radius : float value for the circle's radius
    DegStep : int of number of segments in the circle

    Returns
    -------
    Udata : array of x values for the circle
    Vdata : array of y values for the circle

    """
    #
    Theda_rad = np.deg2rad(np.arange(0,361,DegStep))
    Udata = Radius*np.cos(Theda_rad) + Center[0]
    Vdata = Radius*np.sin(Theda_rad) + Center[1]
    #
    uv_mag = np.sqrt(Udata**2 + Vdata**2)
    for i in range(len(Udata)):
        if uv_mag[i] > 1:
            Udata[i] = float('nan')
            Vdata[i] = float('nan')
        #end if
    #end for
    return Udata, Vdata
    #
def _Set_PlotScale_list(self, PlotScale_list):#-------------------------------------------------
    self._PlotScale_list = PlotScale_list
    #
def _Get_PlotScale_list(self):#-----------------------------------------------------
    return self._PlotScale_list
    #
def _Set_YaxisR(self, YaxisR):#-----------------------------------------------------
    self.YaxisR = YaxisR
    #
def _Set_SWPVAR_Name_list(self, SWPVAR_Name_list):#---------------------------------------
    self.SWPVAR_Name_list = SWPVAR_Name_list
    #
def _Set_Trace_SWPVAR_Value_list(self, Trace_SWPVAR_Value_list):#-------------------------------
    self.Trace_SWPVAR_Value_list = Trace_SWPVAR_Value_list
    #
def _Set_Highlighted_idx(self, Highlighted_idx):
    self.Highlighted_idx = Highlighted_idx
    #
def _Set_GraphTitle(self, GraphTitle):#---------------------------------------------------
    self.GraphTitle = GraphTitle
    #
def _Set_ax(self, ax, fig, ax2=None):#----------------------------------------------------------
```

```
        self.ax = ax
        self.ax2 = ax2
        self.fig = fig
        #
    def _Set_TraceData_ay_list(self, TraceData_ay_list):#-------------------------------------
        self.TraceData_ay_list = TraceData_ay_list
        #
    def _Set_MeasurementName_list(self, MeasurementName_list):
        self.MeasurementName_list = MeasurementName_list
        #
    def _Set_PlotType(self, PlotType):#-------------------------------------------------------
        self.PlotType = PlotType
        #
    def _Set_Projection(self, Projection):#---------------------------------------------------------
        self.Projection = Projection
        #
#
#*********************************************************************************************
#
class _Colors():
    def __init__(self):#----------------------------------------------------------------
        self.black = (0.0, 0.0, 0.0)
        self.white = (1.0, 1.0, 1.0)
        GreyShade = 100.0
        self.grey = ((GreyShade+1)/255, (GreyShade/255), (GreyShade/255))
        GreyShade = 140.0
        self.mid_grey = ((GreyShade+1)/255, GreyShade/255, GreyShade/255)
        GreyShade = 180.0
        self.light_grey = ((GreyShade+1)/255, GreyShade/255, GreyShade/255)
        GreyShade = 220.0
        self.very_light_grey = ((GreyShade+1)/255, GreyShade/255, GreyShade/255)
        GreyShade = 240.0
        self.super_light_grey = ((GreyShade+1)/255, GreyShade/255, GreyShade/255)
        self.red = (1.0, 0.1, 0.0)
        self.pink = (255./255, 204./255, 204.0/255)
        self.BrickRed = (128.0/255, 0.0, 64.0/255)
        self.orange = (1.0, 128.0/255, 0.0)
        self.blue = (0.0, 0.0, 1.0)
        self.LightBlue = (138.0/255, 200.0/255, 1.0)
        self.LightBlue_pale = (220.0/255, 220.0/255, 1.0)
        self.green = (0.0, 1.0, 0.0)
        self.DarkGreen = (18.0/255, 100.0/255, 21.0/255)
        self.brown = (214.0/255, 176.0/255, 137.0/255)
        self.tan = (202.0/255, 168.0/255, 49.0/255)
        self.purple = (167.0/255, 79.0/255, 1.0)
        self.magenta = (1.0, 0.0, 1.0)
        self.GreenYellow = (173/255,255/255,47/255)
        self.GoldenRod = (218/255,165/255,32/255)
        self.turquoise = (64/255,224/255,208/255)
        self.AquaMarine = (127/255,255/255,212/255)
        self.DeepPink = (255/255,20/255,147/255)
        self.Colors_list = [self.red, self.blue, self.green, self.LightBlue, self.orange,\
                            self.magenta, self.brown, self.BrickRed, self.tan, self.turquoise]
        #
#
#*********************************************************************************************
#
class SmithChart_2D(GraphMethods):#-------------------------------------------------------------
    """
    2D Smith Chart

    Reads data from a MWO Smith Chart graph. If the data is swept parameter (from a SWPVAR element in the
    schematic) then the up/down keyboard keys can be used to highlight a single trace. The SWPVAR values of the
    highlighted trace are displayed in the output window.

    If multiple traces are selected, then the data must come from the same schematic and the
    measurement SWPVAR settings in MWO must be the same.

    Prameters
    ---------
    awrde : object variable    awrde = mwo.CMWOffice()
```

```
    GraphName :  Name of the MWO graph

    MeasurementName_list : list of strings. Each string is the name of the measurement in the graph
                           The measurement strings can be partial -- just enough characters to
                           uniquely identify each measurment in the graph

    """
    def __init__(self, awrde, GraphName, MeasurementName_list):#-----------------------------------
        colors = _Colors()
        GraphMethods.__init__(self, awrde, colors)
        self._Main(GraphName, MeasurementName_list)
        #
    def _Main(self, GraphName, MeasurementName_list):
#-------------------------------------------------------------------------------
        [graph, GraphTitle, MeasurementName_list, YaxisR] = self._SetupMeasurements(GraphName,
MeasurementName_list)
        self._Set_YaxisR(YaxisR)
        self._Set_MeasurementName_list(MeasurementName_list)
        self._Set_GraphTitle(GraphTitle)
        #
        [SWPVAL_Value_list, Trace_SWPVAR_Value_list, SWPVAR_Name_list] = self._GetSweepParameters(graph,
MeasurementName_list)
        [TraceData_ay_list, Highlighted_idx] = self._GetTraceData(graph, MeasurementName_list)
        PlotScale_list = self._Get_AxisLimits(graph, graph_type='Smith Chart')
        #
        self._Set_PlotScale_list(PlotScale_list)
        self._Set_TraceData_ay_list(TraceData_ay_list)
        self._Set_PlotType('Smith Chart')
        self._Set_Projection('2D')
        self._Set_Highlighted_idx(Highlighted_idx)
        self._Set_SWPVAR_Name_list(SWPVAR_Name_list)
        self._Set_Trace_SWPVAR_Value_list(Trace_SWPVAR_Value_list)
        [ax, ax2, fig] = self._SetupPlot(PlotScale_list, GraphTitle, PlotType='Smith Chart', Projection='2D')
        self._Set_ax(ax, fig, ax2)
        #
#
#*******************************************************************************************
#
class SmithChart_3D(GraphMethods):#-------------------------------------------------------------
    """
    3D Smith Chart

    Plots a 3D Smith Chart where the z-axis is the Frequency

    Reads data from a MWO Smith Chart graph. If the data is swept parameter (from a SWPVAR element in the
    schematic) then the up/down keyboard keys can be used to highlight a single trace. The SWPVAR values of the
    highlighted trace are displayed in the output window.

    If multiple traces are selected, then the data must come from the same schematic and the
    measurement SWPVAR settings in MWO must be the same.

    Prameters
    ---------
    awrde : object variable    awrde = mwo.CMWOffice()

    GraphName :  Name of the MWO graph

    MeasurementName_list : list of strings. Each string is the name of the measurement in the graph
                           The measurement strings can be partial -- just enough characters to
                           uniquely identify each measurment in the graph

    """
    def __init__(self, awrde, GraphName, MeasurementName_list):#-----------------------------------
        colors = _Colors()
        GraphMethods.__init__(self, awrde, colors)
        self._Main(GraphName, MeasurementName_list)
        #
    def _Main(self, GraphName, MeasurementName_list):
#-------------------------------------------------------------------------------
        [graph, GraphTitle, MeasurementName_list, YaxisR] = self._SetupMeasurements(GraphName,
```

```
MeasurementName_list)
        self._Set_YaxisR(YaxisR)
        self._Set_MeasurementName_list(MeasurementName_list)
        self._Set_GraphTitle(GraphTitle)
        [SWPVAL_Value_list, Trace_SWPVAR_Value_list, SWPVAR_Name_list] = self._GetSweepParameters(graph,
MeasurementName_list)
        [TraceData_ay_list, Highlighted_idx] = self._GetTraceData(graph, MeasurementName_list)
        PlotScale_list = self._Get_AxisLimits(graph, graph_type='Smith Chart')
        self._Set_PlotScale_list(PlotScale_list)
        self._Set_TraceData_ay_list(TraceData_ay_list)
        self._Set_PlotType('Smith Chart')
        self._Set_Projection('3D')
        self._Set_Highlighted_idx(Highlighted_idx)
        self._Set_SWPVAR_Name_list(SWPVAR_Name_list)
        self._Set_Trace_SWPVAR_Value_list(Trace_SWPVAR_Value_list)
        [ax, ax2, fig] = self._SetupPlot(PlotScale_list, GraphTitle, PlotType='Smith Chart', Projection='3D')
        self._Set_ax(ax, fig, ax2)
        #
#
#*********************************************************************************************
#
class Rectangular_2D(GraphMethods):
    """
    2D Rectangular Graph

    Reads magnitude data from a MWO Rectangular. If the data is swept parameter (from a SWPVAR element in the
    schematic) then the up/down keyboard keys can be used to highlight a single trace. The SWPVAR values of the
    highlighted trace are displayed in the output window.

    If multiple traces are selected, then the data must come from the same schematic and the
    measurement SWPVAR settings in MWO must be the same.

    Prameters
    ---------
    awrde : object variable    awrde = mwo.CMWOffice()

    GraphName :  Name of the MWO graph

    MeasurementName_list : list of strings. Each string is the name of the measurement in the graph
                           The measurement strings can be partial -- just enough characters to
                           uniquely identify each measurment in the graph

    """
    def __init__(self, awrde, GraphName, MeasurementName_list):#-----------------------------------
        colors = _Colors()
        GraphMethods.__init__(self, awrde, colors)
        self._Main(GraphName, MeasurementName_list)
        #
    def _Main(self, GraphName, MeasurementName_list):
#-------------------------------------------------------------------------------
        [graph, GraphTitle, MeasurementName_list, YaxisR] = self._SetupMeasurements(GraphName,
MeasurementName_list)
        self._Set_YaxisR(YaxisR)
        self._Set_MeasurementName_list(MeasurementName_list)
        self._Set_GraphTitle(GraphTitle)
        [SWPVAL_Value_list, Trace_SWPVAR_Value_list, SWPVAR_Name_list] = self._GetSweepParameters(graph,
MeasurementName_list)
        [TraceData_ay_list, Highlighted_idx] = self._GetTraceData(graph, MeasurementName_list)
        PlotScale_list = self._Get_AxisLimits(graph, graph_type='Rectangular')
        self._Set_PlotScale_list(PlotScale_list)
        self._Set_TraceData_ay_list(TraceData_ay_list)
        self._Set_PlotType('Rectangular')
        self._Set_Projection('2D')
        self._Set_Highlighted_idx(Highlighted_idx)
        self._Set_SWPVAR_Name_list(SWPVAR_Name_list)
        self._Set_Trace_SWPVAR_Value_list(Trace_SWPVAR_Value_list)
        [ax, ax2, fig] = self._SetupPlot(PlotScale_list, GraphTitle, PlotType='Rectangular', Projection='2D')
        self._Set_ax(ax, fig, ax2)
        #
#
#*********************************************************************************************
```

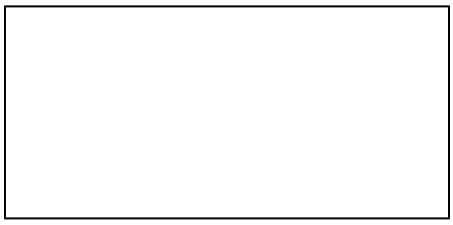
```
#
class Rectangular_3D(GraphMethods):
    """
    3D Rectangular Graph

    Reads magnitude data from a MWO Rectangular. If the data is swept parameter (from a SWPVAR element in the
    schematic) then the up/down keyboard keys can be used to highlight a single trace. The SWPVAR values of the
    highlighted trace are displayed in the output window.

    If multiple traces are selected, then the data must come from the same schematic and the
    measurement SWPVAR settings in MWO must be the same.

    Prameters
    ---------
    awrde : object variable    awrde = mwo.CMWOffice()

    GraphName :  Name of the MWO graph

    MeasurementName_list : list of strings. Each string is the name of the measurement in the graph
                           The measurement strings can be partial -- just enough characters to
                           uniquely identify each measurment in the graph

    """
    def __init__(self, awrde, GraphName, MeasurementName_list):#-----------------------------------
        colors = _Colors()
        GraphMethods.__init__(self, awrde, colors)
        self._Main(GraphName, MeasurementName_list)
        #
    def _Main(self, GraphName, MeasurementName_list):
#-------------------------------------------------------------------------------
        [graph, GraphTitle, MeasurementName_list, YaxisR] = self._SetupMeasurements(GraphName,
MeasurementName_list)
        self._Set_YaxisR(YaxisR)
        self._Set_MeasurementName_list(MeasurementName_list)
        self._Set_GraphTitle(GraphTitle)
        [SWPVAL_Value_list, Trace_SWPVAR_Value_list, SWPVAR_Name_list] = self._GetSweepParameters(graph,
MeasurementName_list)
        [TraceData_ay_list, Highlighted_idx] = self._GetTraceData(graph, MeasurementName_list)
        PlotScale_list = self._Get_AxisLimits(graph, graph_type='Rectangular')
        self._Set_PlotScale_list(PlotScale_list)
        self._Set_TraceData_ay_list(TraceData_ay_list)
        self._Set_PlotType('Rectangular')
        self._Set_Projection('3D')
        self._Set_Highlighted_idx(Highlighted_idx)
        self._Set_SWPVAR_Name_list(SWPVAR_Name_list)
        self._Set_Trace_SWPVAR_Value_list(Trace_SWPVAR_Value_list)
        [ax, ax2, fig] = self._SetupPlot(PlotScale_list, GraphTitle, PlotType='Rectangular', Projection='3D')
        self._Set_ax(ax, fig, ax2)
        #
#
#*******************************************************************************************
#
```

Copy and paste the code into a file labelled *mwo_2d3d_graphing.py*. Place this file is a directory that is in the search path for Python. This file will be treated as a module in your top level Python file.

Rectangular 2D Graph

This plots scalar data vs. frequency on 2D graph. In this particular case, magnitude s21 data is shown in red and magnitude s11 data is shown in blue

The highlighted traces can be changed with the up and down keyboard buttons. As the highlighted trace changes, the swept parameter values are printed as shown:



The Python code for the 2D rectangular graph is shown here:

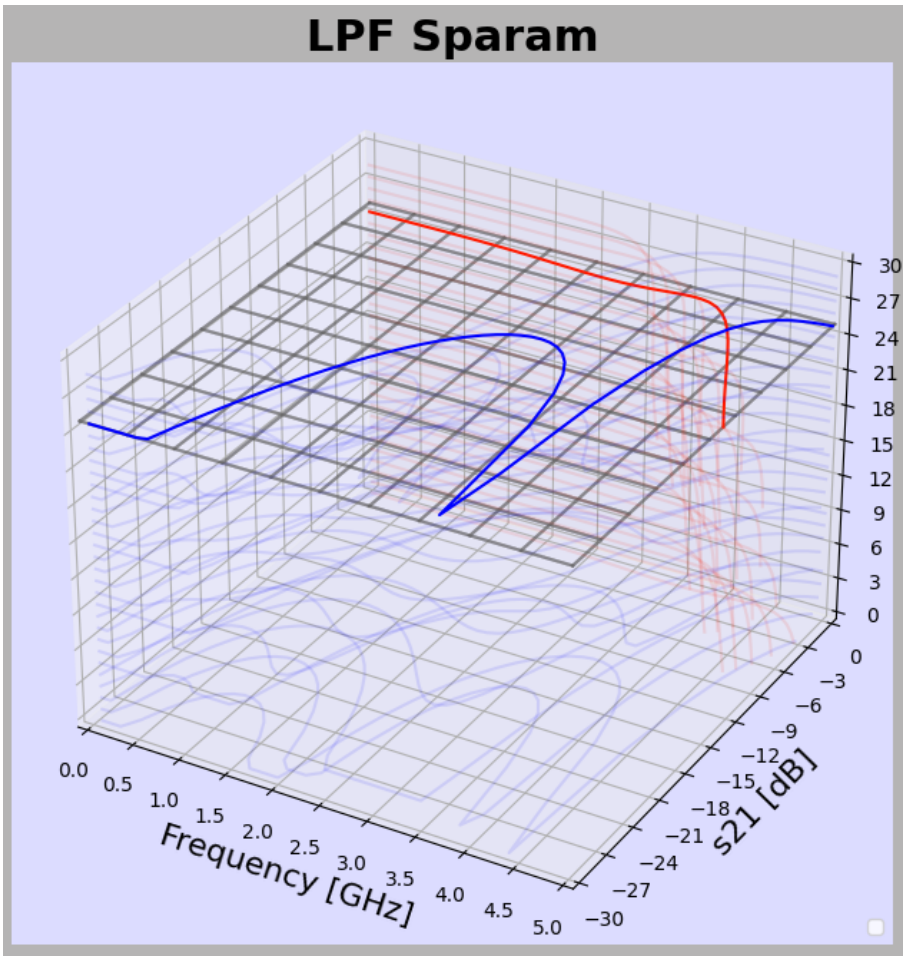
Rectangular 2D Graph Top Level

```
import pyawr.mwoffice as mwo #pyawr interface to AWRDE
import mwo_2d3d_graphing

awrde = mwo.CMWOffice()

#help(mwo_2d3d_graphing.Rectangular_2D)        #un-comment to print help information
GraphName = 'LPF Sparam'                       #Name of the AWRDE rectangular graph
MeasNamePartial_list = ['S(2,1)', 'S(1,1)']    #Measurement names in the AWRDE graph. These can be partial
names.
rect2d = mwo_2d3d_graphing.Rectangular_2D(awrde, GraphName, MeasNamePartial_list)
#
#Axis scaling parameters (Min, Max, Number of divisions, scaling factor, axis label)
rect2d.Set_Axis_Scale(Xaxis_Scale=(0,5,10,1e9,'Frequency [GHz]'),\
                      Yaxis_Scale=(-20,0,10,1,'s21 [dB]'),\
                      YaxisR_Scale=(-50,0,10,1,'s11 [dB]'))
rect2d.PlotTraces()
rect2d.ShowPlot()
```

See the comments in the code for explanation.

Rectangular 3D Graph

Very similar to the Rectangular 2D graph except in this case the trace index along the z-axis in the 3D graph:



Rectangular 3D Graph Top Level

```
import pyawr.mwoffice as mwo #pyawr interface to AWRDE
import mwo_2d3d_graphing

awrde = mwo.CMWOffice()

#help(mwo_2d3d_graphing.Rectangular_3D)                    #un-comment to print help information
GraphName = 'LPF Sparam'                                   #Name of the AWRDE rectangular graph
MeasNamePartial_list = ['S(2,1)', 'S(1,1)']                #Measurement names in the AWRDE graph. These can be
partial names.
#
#Axis scaling parameters (Min, Max, Number of divisions, scaling factor, axis label)
rect3d = mwo_2d3d_graphing.Rectangular_3D(awrde, GraphName, MeasNamePartial_list)
rect3d.Set_Axis_Scale(Xaxis_Scale=(0,5,10,1e9,'Frequency [GHz]'),\
                      Yaxis_Scale=(-30,0,10,1,'s21 [dB]'))
rect3d.PlotTraces()
rect3d.ShowPlot()
```

Like the 2D graph, the up/down keys change the highlighted trace index and prints out the swept parameter values for the hightlighted index.

2D Smith Chart

2D Smith Chart reads complex data from an AWRDE Smith Chart graph. Like the Rectangular graphs, the highlighted trace index can be changed with the keyboard up and down arrow keys.

2D Smith Chart Top Level

```
import pyawr.mwoffice as mwo #pyawr interface to AWRDE
import mwo_2d3d_graphing

awrde = mwo.CMWOffice()

#help(mwo_2d3d_graphing.SmithChart_2D)            #un-comment to print help information
GraphName = 's11'                                 #Name of the AWRDE rectangular graph
MeasNamePartial_list = ['S(1,1)', 'S(2,2)']       #Measurement names in the AWRDE graph. These can be partial
names.
smith2d = mwo_2d3d_graphing.SmithChart_2D(awrde, GraphName, MeasNamePartial_list)
smith2d.PlotTraces()
smith2d.ShowPlot()
```

3D Smith Chart

Similar to the 2D Smith Chart, but with frequency along the z-axis

3D Smitch Chart Top Level

```
import pyawr.mwoffice as mwo #pyawr interface to AWRDE
import mwo_2d3d_graphing

awrde = mwo.CMWOffice()

#help(mwo_2d3d_graphing.SmithChart_3D)                      #un-comment to print help information
GraphName = 's11'                                           #Name of the AWRDE rectangular graph
MeasNamePartial_list = ['S(1,1)', 'S(2,2)']                 #Measurement names in the AWRDE graph. These can
be partial names.
smith3d = mwo_2d3d_graphing.SmithChart_3D(awrde, GraphName, MeasNamePartial_list)
#
#Axis scaling parameters (Min, Max, Number of divisions, scaling factor, axis label)
smith3d.Set_Axis_Scale(Zaxis_Scale=(0,5,10,1e9,'Frequency [GHz]'))
smith3d.PlotTraces()
smith3d.ShowPlot()
```