```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 13:10:28 2017

@author: Armin Galetzka, Denys Bast
Measures and calculates the transfer function of a two port.
Requirements:
    - AWG with two output ports
    - Channel 1 at oscilloscope is output signal of AWG
    - Channnel 3 and 4 are connected to the output of the two port and
      the quantity of interest is calculated by CH3-CH4


Input:  fmax            ------  max frequency of interest
        Vpp             ------  Output peac-peac voltage of AWG
        showPlots       ------  If True plots are shown and saved as .pdf
        createCSV       ------  If True a CSV file for each quantity of interest
                                is created
        formatOutput    ------  0=dB, 1=linear, 2=both
"""

def compute(fmax, Vpp, bits=10, writeAWG=True, showPlots=True, createCSV=True, \
            formatOutput=1):

    import visa
    import MLBS
    import time
    import matplotlib.pyplot as plt
    import numpy as np
    import FFT
    import csv
    import os


    # Create folder for results
    directory = time.strftime("%d.%m.%Y_%H_%M_%S")
    if not os.path.exists(directory):
        os.makedirs(directory)
    if not os.path.exists(directory + "/Plots"):
        os.makedirs(directory + "/Plots")
    if not os.path.exists(directory + "/csv"):
        os.makedirs(directory + "/csv")
    # Parameter
    awg_volt = Vpp
    samplerateAWG = 2.5*fmax
    samplerateOszi = 100*samplerateAWG
    fPlot = fmax
    possibleRecordLength = [500,2500,5000,10e3,25e3,50e3,100e3,250e3,500e3]
    possibleRecordLength = np.array(possibleRecordLength)
    linewidthPlot = 1

    font = {'family' : 'normal',
            'weight' : 'normal',
            'size'   : 12}

    plt.rc('font', **font)

    # Connect to Instruments
    rm = visa.ResourceManager()
    rs = rm.list_resources()
```

```python
awg_id = rs[0]
AWG = rm.open_resource(awg_id)
# am Desktop PC
# dso_ip = rs[1]
# am Gruppenlaptop BTNBG006
dso_ip = 'TCPIP::169.254.225.181::gpib0,1::INSTR'
DSO = visa.ResourceManager().get_instrument(dso_ip)

[signal, seed] = MLBS.get(bits)
Tns = 0.4/fmax
periodTime = signal.size*Tns
horizontalScalePerDiv = 1.5*periodTime/10 #At least one period needs to be
                                          #shown on the DSO

##################################################
################## Write to AWG ##################
##################################################
if writeAWG:
    AWG.write("*RST")
    AWG.write("SOURce1:FUNCtion:ARBitrary:FILTer OFF")
    AWG.write("SOURce2:FUNCtion:ARBitrary:FILTer OFF")
    #time.sleep(5)
    AWG.write("DATA:VOLatile:CLEar")
    #time.sleep(5)
    myrange=max(abs(max(signal)),abs(min(signal)))
     #Data Conversion from V to DAC levels
    data_conv = np.round(signal*32766/myrange);
    data_conv = ",".join(str(e) for e in data_conv)
    AWG.write("SOURce1:DATA:ARBitrary:DAC myarb ," + data_conv)
    AWG.write("SOURce1:FUNCtion:ARBitrary 'myarb'")
    time.sleep(10)
    AWG.write("SOURce1:FUNCtion ARB") #USER
    AWG.write("DISPlay:FOCus CH1")
    AWG.write("DISPlay:UNIT:ARBRate FREQuency")
    AWG.write("SOURce1:FUNCtion:ARBitrary:SRATe " + str(samplerateAWG))
    AWG.write("SOURce2:DATA:ARBitrary:DAC myarb ," + data_conv)
    AWG.write("SOURce2:FUNCtion:ARBitrary 'myarb'")
    time.sleep(10)
    AWG.write("SOURce2:FUNCtion ARB") #USER
    AWG.write("DISPlay:FOCus CH2")
    AWG.write("DISPlay:UNIT:ARBRate FREQuency")
    AWG.write("SOURce2:FUNCtion:ARBitrary:SRATe " + str(samplerateAWG))
    AWG.write("FUNC:ARB:SYNC")
    AWG.write("SOURce1:VOLTage " + str(awg_volt))
    AWG.write("SOURce2:VOLTage " + str(awg_volt))
    time.sleep(5)
    AWG.write("OUTPut1 ON")
    AWG.write("OUTPut2 ON")
    AWG.write("DISPlay:FOCus CH1")

##################################################
################## Write to DSO ##################
##################################################

DSO.write("*RST") #Restores the state of the instrument from a copy of
                  #the settings stored in memory
DSO.write("ACQUIRE:STATE OFF") #This command stops acquisitions
DSO.write("SELECT:CH1 ON") #Turns the channel 1 waveform display on, and
                           #selects channel 1.
DSO.write("MATH3:DEFIne \"CH3-CH4\"") #Defines MATH function
DSO.write("SELECT:MATH3 ON") #Turns MATH3 display on
```

```
DSO.write("MATH1:DEFIne \"CH1\"") #Defines MATH function. CH1 is copied
                                   #to MATH1, because output format of
                                   #MATH1 is easier to handle
DSO.write("SELECT:MATH1 ON") #Turns MATH1 display on
DSO.write("TRIGger:A:EDGE:SOUrce CH1") #This command sets or queries the
                                       #source for the A edge trigger.
DSO.write("TRIGger:A:EDGE:SLOpe FALL") #This command sets or queries the
                                       #slope for the A edge trigger.

DSO.write("HORizontal:MAIn:SCAle " + str(horizontalScalePerDiv)) #Sets the
#time per division for the time base
# Here 1,5 periods are on screen. Necessary since Osci has only discrete
# values for horizontal scale and it needs to be ensured that at least
# one full period is in the screen
horizontalScalePerDiv = DSO.query("HORizontal:MAIn:SCAle?")
horizontalScalePerDiv = [float(s) for s
                         in horizontalScalePerDiv.split(',')]
horizontalScalePerDiv = horizontalScalePerDiv[0]
recordLength = horizontalScalePerDiv*10*samplerateOszi
ind = np.argmin(np.abs(recordLength - possibleRecordLength))
if possibleRecordLength[ind] < recordLength and \
(ind+1)<possibleRecordLength.size:
    recordLength = possibleRecordLength[ind+1]
else:
    recordLength = possibleRecordLength[ind]
DSO.write("HORIZONTAL:RECOrdlength " + str(recordLength)) #1e5
DSO.write("CH1:SCAle " + str(awg_volt/6)) #Sets the vertical scale
DSO.write("MATH1:SCAle " + str(awg_volt/6)) #Sets the vertical scale
DSO.write("CH2:SCAle 20.0E-3") #Sets the vertical scale
DSO.write("CH3:SCAle 50.0E-3") #Sets the vertical scale
DSO.write("CH4:SCAle 50.0E-3") #Sets the vertical scale
DSO.write("MATH3:SCAle 200.0E-3") #Sets the vertical scale
DSO.write("CH1:POSition 0") #Sets the horizontal scale
DSO.write("MATH3:POSition 0") #Sets the horizontal scale
DSO.write("MATH1:POSition 0") #Sets the horizontal scale
DSO.write("CH1:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH2:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH3:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH4:TERmination 1.0E+6") #Sets the termination of the channel
DSO.write("CH1:COUPling DC")  #Sets the coupling of channel 1 to AC
# Coupling to AC since the input signal has no DC component.
# No DC expected at the output. Use AC coupling to reduce influence
# from outside.
DSO.write("DATa:SOUrce MATH1")   #This command sets the location of
                               #waveform data that is transferred from the
                               #instrument by the CURVe? Query
DSO.write("DATa:ENCdg ASCIi") #This command sets the format of outgoing
                              #waveform data to ASCII
DSO.write("ACQUIRE:MODE SAMPLE") #This command sets the acquisition mode
                                 #of the instrument to sample mode
DSO.write("ACQUIRE:STOPAFTER SEQUENCE") #Specifies that the next
                                        #acquisition will be a
                                        #single-sequence acquisition.
DSO.write("HORizontal:MAIn:SAMPLERate " + str(samplerateOszi)) # Sets the
                                        # sample rate of the device.
                                        # Here: 10 times maximum expected
                                        # frequency to reduce aliasing
DSO.write("ACQUIRE:STATE ON") #This command starts acquisitions
DSO.write("DATa:START 1") #This command sets the starting data point
                #for waveform transfer. This command allows for the
                #transfer of partial waveforms to and from the instrument.
```

```python
DSO.write("DATa:STOP " + DSO.query("HORIZONTAL:RECOrdlength?")) #Sets the
   #last data point that will be transferred when using the CURVe? query
time.sleep(5)
dataUin = DSO.query("CURVe?")
DSO.write("DATa:SOUrce MATH3")   #This command sets the location of
                                #waveform data that is transferred from the
                                #instrument by the CURVe? Query
DSO.write("DATa:ENCdg ASCIi") #This command sets the format of outgoing
                               #waveform data to ASCII
DSO.write("DATa:STARt 1") #This command sets the starting data point
                #for waveform transfer. This command allows for the
                #transfer of partial waveforms to and from the instrument.
DSO.write("DATa:STOP " + DSO.query("HORIZONTAL:RECOrdlength?")) #Sets the
     #last data point that will be transferred when using the CURVe? query
time.sleep(5)
dataUout = DSO.query("CURVe?")

recordLength = DSO.query("HORIZONTAL:RECOrdlength?")
horizontalScalePerDiv = DSO.query("HORizontal:MAIn:SCAle?")
YScalePerDivUin = DSO.query("MATH1:SCAle?")
YScalePerDivUout = DSO.query("MATH3:SCAle?")

##################################################
############## Compute transfer function #########
##################################################

# Change format of data from DSO
dataUin = [float(s) for s in dataUin.split(',')]

dataUout = [float(s) for s in dataUout.split(',')]
dataUin = np.array(dataUin)
dataUout = np.array(dataUout)
recordLength = [float(s) for s in recordLength.split(',')]
recordLength = recordLength[0]
horizontalScalePerDiv = [float(s) for s
                         in horizontalScalePerDiv.split(',')]
horizontalScalePerDiv = horizontalScalePerDiv[0]
YScalePerDivUin = [float(s) for s in YScalePerDivUin.split(',')]
YScalePerDivUin = YScalePerDivUin[0]
YScalePerDivUout = [float(s) for s in YScalePerDivUout.split(',')]
YScalePerDivUout = YScalePerDivUout[0]

# Get time vector
dt = 10*horizontalScalePerDiv/recordLength
time = np.arange(0,10*horizontalScalePerDiv,dt)

# Reduce time vector and signal to one period
tmpTime = periodTime - time[0]
ind = np.argmin(abs(time - tmpTime)) #find next index
time = time[0:ind]
dataUin = dataUin[0:ind]
dataUout = dataUout[0:ind]

# Compute FFT of signals in time domain
[frq, UinAmpl, PhaseUin, Uin] = FFT.get(dataUin, 1/(time[-1]-time[-2]));
[frq, UoutAmpl, PhaseUout, Uout] = FFT.get(dataUout, \
                                            1/(time[-1]-time[-2]));

# Reduce frequency domain signal to maximum frequency fPlot
ind = np.argmin(abs(frq-fPlot))
frq = frq[0:ind]
```

```python
UinAmpl = UinAmpl[0:ind]
UoutAmpl = UoutAmpl[0:ind]
Uin=Uin[0:ind]
Uout=Uout[0:ind]
PhaseUout=PhaseUout[0:ind]
PhaseUin=PhaseUin[0:ind]

# Compute transer function
H = UoutAmpl/UinAmpl
PhaseH=np.angle(Uout/Uin)
#PhaseH = PhaseUout-PhaseUin
# No DC component!
H=H[1:]
UinAmpl=UinAmpl[1:]
UoutAmpl=UoutAmpl[1:]
frq=frq[1:]
Uin=Uin[1:]
Uout=Uout[1:]
PhaseH=PhaseH[1:]
##################################################
############## Plots ############################
##################################################

if showPlots:

    f=0
    fig = plt.figure(f+1)
    f+=1
    plt.plot(time*1e6, dataUin, linewidth=linewidthPlot)
    plt.ylabel(r'$U_{\mathrm{in}}(t)$')
    plt.xlabel(r'$t$ in $\mu$s')
    plt.grid(True)
    fig.savefig(directory + "/Plots/Uin_time.pdf", bbox_inches='tight')
    plt.show()

    fig = plt.figure(f+1)
    f+=1
    plt.plot(time*1e6, dataUout, linewidth=linewidthPlot)
    plt.ylabel(r'$U_{\mathrm{out}}(t)$')
    plt.xlabel(r'$t$ in $\mu$s')
    plt.grid(True)
    fig.savefig(directory + "/Plots/Uout_time.pdf", bbox_inches='tight')
    plt.show()

    fig = plt.figure(f+1)
    f+=1
    plt.plot(frq/1e6, PhaseH, linewidth=linewidthPlot)
    plt.ylabel(r'arg($H(\omega)$)')
    plt.xlabel(r'$f$ in MHz')
    plt.grid(True)
    fig.savefig(directory + "/Plots/PhaseH.pdf", bbox_inches='tight')
    plt.show()

    if (formatOutput==0) or (formatOutput==2):
        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, 20*np.log10(UinAmpl), linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{in}}(f)|$ in dB')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UinAmpl_frq_dB.pdf",\
```

```python
                bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, 20*np.log10(UoutAmpl), linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{out}}(f)|$ in dB')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UoutAmpl_frq_dB.pdf",\
                bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, 20*np.log10(H), linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|H(f)|$ in dB')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/H_dB.pdf", bbox_inches='tight')
        plt.show()

    if (formatOutput==1 or formatOutput==2):
        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, UinAmpl, linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{in}}(f)|$')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UinAmpl_frq_linear.pdf",\
                bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, UoutAmpl, linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|U_{\mathrm{out}}(f)|$')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/UoutAmpl_frq_linear.pdf",\
                bbox_inches='tight')
        plt.show()

        fig = plt.figure(f+1)
        f+=1
        plt.plot(frq/1e6, H, linewidth=linewidthPlot)
        plt.grid(True)
        plt.ylabel(r'$|H(f)|$')
        plt.xlabel(r'$f$ in MHz')
        fig.savefig(directory + "/Plots/H_linear.pdf", bbox_inches='tight')
        plt.show()
###################################################
############## Create CSV #########################
###################################################

if createCSV:

    with open(directory + '/csv/UinTime.csv', 'w', newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                    quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,dataUin.size):
```

```
        writer.writerow([str(time[i]), str(dataUin[i])])

with open(directory + '/csv/UoutTime.csv', 'w', newline="") as csvfile:
    writer = csv.writer(csvfile, delimiter=';',
                        quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for i in range(0,dataUout.size):
        writer.writerow([str(time[i]), str(dataUout[i])])

with open(directory + '/csv/PhaseH.csv', 'w', newline="") as csvfile:
    writer = csv.writer(csvfile, delimiter=';',
                        quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for i in range(0,PhaseH.size):
        writer.writerow([str(frq[i]), str(PhaseH[i])])

if (formatOutput==1) or (formatOutput==2):

    with open(directory + '/csv/UinAmplFrq_linear.csv', 'w',\
              newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,UinAmpl.size):
            writer.writerow([str(frq[i]), str(UinAmpl[i])])

    with open(directory + '/csv/UoutAmplFrq_linear.csv', 'w',\
              newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,UoutAmpl.size):
            writer.writerow([str(frq[i]), str(UoutAmpl[i])])

    with open(directory + '/csv/HAmpl_linear.csv', 'w',\
              newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,H.size):
            writer.writerow([str(frq[i]), str(H[i])])

if (formatOutput==0 or formatOutput==2):

    with open(directory + '/csv/UinAmplFrq_dB.csv', 'w',\
              newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,UinAmpl.size):
            writer.writerow([str(frq[i]),
                             str(20*np.log10(UinAmpl[i]))])

    with open(directory + '/csv/UoutAmplFrq_dB.csv', 'w',\
              newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,UoutAmpl.size):
            writer.writerow([str(frq[i]),
                             str(20*np.log10(UoutAmpl[i]))])

    with open(directory + '/csv/HAmpl_dB.csv', 'w',\
              newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=';',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i in range(0,H.size):
            writer.writerow([str(frq[i]), str(20*np.log10(H[i]))])
```

```
with open(directory + '/csv/UinFrq.csv', 'w', newline="") as csvfile:
    writer = csv.writer(csvfile, delimiter=';',
                        quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for i in range(0,Uin.size):
        writer.writerow([str(frq[i]), str(Uin[i])])

with open(directory + '/csv/UoutFrq.csv', 'w', newline="") as csvfile:
    writer = csv.writer(csvfile, delimiter=';',
                        quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for i in range(0,Uout.size):
        writer.writerow([str(frq[i]), str(Uout[i])])
```