TestImportHomeMadeFunction

May 15, 2020

[3]:
```
import sys, os
currentpath = os.getcwd()
foldername = currentpath + '\\HomeMadeFunction'
sys.path.append(foldername)
```

[4]:
```
sys.path
```

[4]:
```
['C:\\Users\\Tim\\Jupyter\\CoreProject_EPRSuite',
 'C:\\Python\\Python38\\python.exe',
 'C:\\Users\\Tim\\Jupyter\\CoreProject_EPRSuite\\
C:\\Users\\Tim\\anaconda3\\python.exe',
 'C:\\Users\\Tim\\anaconda3\\python37.zip',
 'C:\\Users\\Tim\\anaconda3\\DLLs',
 'C:\\Users\\Tim\\anaconda3\\lib',
 'C:\\Users\\Tim\\anaconda3',
 '',
 'C:\\Users\\Tim\\anaconda3\\lib\\site-packages',
 'C:\\Users\\Tim\\anaconda3\\lib\\site-packages\\win32',
 'C:\\Users\\Tim\\anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\Users\\Tim\\anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\Users\\Tim\\anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\Tim\\.ipython',
 'C:\\Users\\Tim\\Jupyter\\CoreProject_EPRSuite\\HomeMadeFunction',
 'C:\\Users\\Tim\\Jupyter\\CoreProject_EPRSuite\\HomeMadeFunction']
```

[8]:
```
filename1 ='\\20190930_TC_20190826_NOSLY173F_1e_t=0s_T60K.spc'
```

[6]:
```
import numpy as np
def winepr(filename, dimname=''):
    """For opening WinEPR files.

    Parameters
    ----------
    filename : str
        The filename that ends with either ``.par`` or ``.spc``.
    """
    # {{{ determine the pair of filenames that we need
```

```
    filename = filename[:-4]+filename[-4:].upper()# case insensitive extension
    if filename[-4:] == '.SPC':
        filename_spc,filename_par = filename,filename.replace('.SPC','.PAR')
    elif filename[-4:] == '.PAR':
        filename_spc,filename_par = filename.replace('.PAR','.SPC'),filename
    else:
        raise ValueError(strm("When guessing that the filename is a"
                " WinEPR file, the extension must be either .SPC or"
                " .PAR\n"
                "This one is called",repr(filename)))
    # {{{ check if the extension is upper or lowercase
    if not os.path.exists(filename_spc):
        filename_spc = filename_spc[:-4] + filename_spc[-4:].lower()
        filename_par = filename_par[:-4] + filename_par[-4:].lower()
    # }}}
    # }}}
    # {{{ load the data
    with open(filename_spc,'rb') as fp:
        data = fp.read()
    data = np.frombuffer(data,'<f4')
    # }}}
    # load the parameters
    v = winepr_load_acqu(filename_par)
    # {{{ use the parameters to determine the axes
    return data,v
```

[7]:
```
import re
def winepr_load_acqu(filename):
    "Load the parameters for the winepr filename"
    with open(filename,'r') as fp:# the U automatically converts dos format
        lines = fp.readlines()
    vars = {}
    line_re = re.compile(r'([_A-Za-z0-9]+) +(.*)')
    lines = map(str.rstrip,lines)
    #lines = [j.rstrip('\n') for j in lines] # because it's just \n, even on␣
↪windows
    v = {}
    for line in lines:
        m = line_re.match(line)
        if m is None:
            raise RuntimeError('Warning:',lsafen(repr(line)),'does not appear␣
↪to be a valid WinEPR format line, and I suspect this is a problem with the␣
↪terminators!')
        else:
            name = m.groups()[0]
            value = m.groups()[1]
            try:
```

```
                value = int(value)
            except:
                try:
                    value = double(value)
                except:
                    pass
            v[name]=value
    jss = int(v['JSS'])
    parameters = [ 'DUAL', '2D', 'FT', 'MANO', 'MAN1', 'PROT', 'VEPR', 'POW',␣
↪'ABS', 'FTX', 'FTY', 'POW2', 'ABS2']
    parameters = map((lambda x: 's_'+x),parameters)
    #masks = [0x00000001L,0x00000002L,0x00000004L,0x00000008L,␣
↪0x00000010L,0x00000020L,0x00000040L,0x00000080L,0x00000100L,0x00000200L,0x00000400L,0x00000
    #values = map((lambda x: x&jss),masks)
    #values = map(bool,values)
    #values = map(bool,values)
    #v.update(dict(zip(parameters,values)))
    return v
```

[9]:
```
data, par = winepr(foldername+filename1)
```

[10]:
```
data
```

[10]:
```
array([  4580.756,   5129.756,   6121.756, …, -15136.244, -13634.244,
       -16156.244], dtype=float32)
```

[11]:
```
data.shape
```

[11]:
```
(1024,)
```

[23]:
```
a = {(k1,v1) for (k1,v1) in par.items()}
```

[28]:
```
par['DOS']
```

[28]:
```
'Format'
```

[32]:
```
par
```

[32]:
```
{'DOS': 'Format',
 'ANZ': 1024,
 'MIN': '-51706.242188',
 'MAX': '93789.757813',
 'JSS': 0,
 'GST': '3365.000000',
 'GSI': '160.000000',
 'JUN': 'G',
 'JON': 'Bruker BioSpin GmbH',
```

```
'JCO': '26/08/2019 NosL Y173F 1e- +W  T=60K t=0s 23 dB',
'JDA': '30/Sep/2019',
'JTM': '12:59',
'JRE': 'c:\\program files\\bruker emx\\syscal\\dualmode.cal',
'JEX': 'field-sweep',
'JNS': 200,
'JSD': 107,
'HCF': '3445.000000',
'HSW': '160.000000',
'RCT': '40.960000',
'RTC': '40.960000',
'RRG': '2.000000e+005',
'RMA': '3.500000',
'MF': '9.654603',
'MP': '1.003e+000',
'MPD': '23.0'}
```

[103]:
```python
def xepr(filename, dimname='', verbose=False):
    """For opening Xepr files.

    Parameters
    ----------
    filename : str
        The filename that ends with ``.DSC``, ``.DTA``, or ``.YGF``.
    """
    filename = filename[:-4]+filename[-4:].upper()# case insensitive extension
    if filename[-4:] == '.DTA':
        filename_spc,filename_par = filename,filename.replace('.DTA','.DSC')
    elif filename[-4:] == '.DSC':
        filename_spc,filename_par = filename.replace('.DSC','.DTA'),filename
    elif filename[-4:] == '.YGF':
        filename_spc,filename_par = filename.replace('.YGF','.DSC'),filename.
↪replace('.YGF','.DTA')
    else:
        raise ValueError(str("When guessing that the filename is a"
                " WinEPR file, the extension must be either .SPC or"
                " .PAR\n"
                "This one is called",filename))
    # check if the file exist
    if not os.path.exists(filename_spc):
        filename_spc = filename_spc[:-4] + filename_spc[-4:].lower()
        filename_par = filename_par[:-4] + filename_par[-4:].lower()
    # load the data
    with open(filename_spc,'rb') as fp:
        data = np.frombuffer(fp.read(),'>f8')
    # load the parameters
    v = xepr_load_acqu(filename_par)
```

```
    # flatten the dictionnary
    new_v = {}
    for k_a,v_a in v.items():
        new_v.update(v_a)
    v = new_v
    # reorganize the data in a complex way separating real part (first column)
    # from imaginary part (second column)
    if v['IKKF'] == 'CPLX':
        data = data.reshape(int(v['XPTS']),int(data.shape[0]/v['XPTS']))
    return data, v
```

[54]:
```
import io
import numpy as np
def xepr_load_acqu(filename):
    '''Load the Xepr acquisition parameter file, which should be a .dsc␣
 ↪extension.

    Returns
    -------
    A dictionary of the relevant results.
    Because of the format of the .dsc files, this is a dictionary of
    dictionaries, where the top-level keys are the hash-block (*i.e.*
    ``#DESC``, *etc.*).
    '''
    def auto_string_convert(x):
        '''genfromtxt is from numpy -- with dtype=None, it does
        automatic type conversion -- note that strings with
        spaces will be returned as a record array it appears to
        need this StringIO function rather than a string because
        it's designed to read directly from a file.  The tolist
        converts the record array to a list.'''
        if len(x):
            try:
                return np.genfromtxt(io.StringIO(x),dtype=None, encoding='str').
 ↪tolist()
            except:
                raise ValueError("genfromtxt chokes on "+repr(x))
        else:
            return None
    which_block = None
    block_re = re.compile(r'^ *#(\w+)')
    comment_re = re.compile(r'^ *\*')
    variable_re = re.compile(r'^ *([^\s]*)\s+(.*?) *$')
    comma_re = re.compile(r'\s*,\s*')
    with open(filename,'r') as fp:
        blocks = {}
        # {{{ read lines and assign to the appropriate block
```

```
        for line in fp:
            m = comment_re.search(line)
            if m:
                pass
            else:
                m = block_re.search(line)
                if m:
                    if which_block is not None:
                        blocks.update({which_block:dict(block_list)})
                    which_block = m.groups()[0]
                    block_list = []
                else:
                    if which_block is None:
                        raise ValueError("Appears to be stuff outside the first␣
↪hashed block which, as far as I know, should not be allowed.  The first␣
↪non-comment line I see is: "+repr(line))
                    else:
                        m = variable_re.search(line)
                        if m:
                            if ',' in m.groups()[1]:
                                # {{{ break into lists
                                block_list.append((m.groups()[0],
                                        map(auto_string_convert,
                                            comma_re.split(
                                                m.groups()[1]))))
                                # }}}
                            else:
                                block_list.append((m.groups()[0],
                                        auto_string_convert(
                                            m.groups()[1])))
                        else:
                            raise ValueError("I don't know what to do with the␣
↪line:\n"+line)
        blocks.update({which_block:dict(block_list)})
        # }}}
    return blocks
```

[100]:
```
filename2 = '\\20200305_NOSLwT_QBand_FSE_FirstDeriv.DSC'
full_filename = foldername+filename2
```

[104]:
```
data2,par2 = xepr(full_filename)
```

[105]:
```
data2.shape
```

[105]:
```
(512, 2)
```

[87]:
```
par2
```

```
[87]: {'DSRC': 'MAN',
       'BSEQ': 'BIG',
       'IKKF': 'CPLX',
       'XTYP': 'IDX',
       'YTYP': 'NODATA',
       'ZTYP': 'NODATA',
       'IRFMT': 'D',
       'IIFMT': 'D',
       'XPTS': 512,
       'XMIN': 11948.0,
       'XWID': 200.0,
       'TITL': "'20200305_NOSLwT_QBand_FSE_FirstDeriv.DSC'",
       'IRNAM': "'Intensity'",
       'IINAM': "'Intensity'",
       'XNAM': "'Field'",
       'IRUNI': "''",
       'IIUNI': "''",
       'XUNI': "'G'",
       'OPER': 'xuser',
       'DATE': '03/05/20',
       'TIME': '16:02:28',
       'CMNT': None,
       'SAMP': None,
       'SFOR': None,
       'STAG': 'C',
       'EXPT': 'PLS',
       'OXS1': 'TADC',
       'AXS1': 'BOVL',
       'AXS2': 'NONE',
       'AXS3': None,
       'MWPW': 0.2123,
       'A1CT': 1.2048,
       'BOVL': 1.2048,
       'A1SW': 0.02,
       'MWFQ': 33497770000.0,
       'AVGS': 1,
       '.DVC': <map at 0x9c79d88>,
       'AcqFineTuning': 'Never',
       'AcqScanFTuning': 'Off',
       'AcqSliceFTuning': 'Off',
       'Power': (212.3, 'mW'),
       'PowerAtten': (0, 'dB'),
       'QValue': 300,
       'EIEENDORFreq': (14.90218, 'MHz/3.5', 'kG'),
       'EIEIsotope': 'H1',
       'EIERFSweepDir': 'Same',
       'EIEStaticField': (12140.0, 'G'),
```

```
'EIEStaticRF': (38.0, 'MHz'),
'ENDORType': 'EIF',
'RF1Atten': (60.0, 'dB'),
'RF1FreqPos': (38.0, 'MHz'),
'RF1StartFreq': (38.0, 'MHz'),
'RF1SweepWidth': (30.0, 'MHz'),
'RF2Atten': (80.0, 'dB'),
'RF2FreqPos': (38.0, 'MHz'),
'RF2StartFreq': (38.0, 'MHz'),
'RF2SweepWidth': (30.0, 'MHz'),
'RFSrcMixing': 'Add',
'SumAtten': (10.0, 'dB'),
'SumAttenStart': (0.0, 'dB'),
'SumAttenWidth': (31.0, 'dB'),
'AllegroMode': True,
'CenterField': (12048.0, 'G'),
'Delay': (0.0, 's'),
'FieldFlyback': 'On',
'FieldPosition': (12048.0, 'G'),
'FieldWait': ['Wait', 'LED', 'off'],
'GFactor': 2.0,
'MeasuringHall': False,
'SetToSampleG': False,
'StaticFieldMon': (12140.0, 'G'),
'SweepDirection': 'Up',
'SweepWidth': (200.0, 'G'),
'WidthTM': (200.0, 'G'),
'FrequencyMon': (33.497765, 'GHz'),
'QMonitBridge': 'Off',
'Attenuation': (60.0, 'dB'),
'ELDORAtt': (30, 'dB'),
'FrequencyA': (33.5, 'GHz'),
'VideoBW': (200, 'MHz'),
'VideoGain': (33, 'dB'),
'AWGPhaseShift': (90.0, 'deg.'),
'AWGPrg': <map at 0x9c05508>,
'AutoTimeOut': True,
'AveragesPerScan': 256,
'ELDORFreqStart': (33.067385, 'GHz'),
'ELDORFreqWidth': (0.8, 'GHz'),
'FTAcqModeSlct': ['Run', 'from', 'Tables'],
'FTEzAWGELDORa': (100.0, '%'),
'FTEzAWGELDORf': (0.0, 'MHz'),
'FTEzAWGELDORw': (0.0, 'MHz'),
'FieldIsStatic': False,
'GradIntPulse': False,
'GrdEnable': False,
```

```
'LastXAxis': ['Magnetic', 'Field'],
'LastYAxis': '?',
'MDigPrg': <map at 0x9c05c48>,
'MMWaveLOFreq': (26.88, 'GHz'),
'MicroImgPrg': <map at 0x9c36748>,
'OnlyAWGChans': False,
'PCycleAllowed': True,
'PCycleOn': True,
'PPExtTrg': False,
'PPExtTrgSlope': 'Rising',
'PlsSPELEXPSlct': ['Set', 'Up', 'Echo'],
'PlsSPELGlbTxt': '\\',
';\\n\\': None,
';': ['QUAD', 'detection', 'with', 'Integration\\n\\'],
'begin': ['exp6',
 '"Hole',
 'Burning',
 'Inv.',
 'Rec.',
 'Pulse',
 'log',
 'scale"',
 '[TRANS',
 'QUAD]\\n\\'],
'\\n\\': None,
'p0': ['[awg1]', ';', 'hard', 'pi/2', 'pulse', 'length\\n\\'],
'p1': ['[awg2]', ';', 'hard', 'pi', 'pulse', 'length\\n\\'],
'p2': ['[awg0]', ';', 'selective', 'pi', 'pulse\\n\\'],
'p3': ('=', 40, ';\\n\\'),
'pg': ('=', 200, ';', 'Integrator', 'Gate', 'Width', '(RESERVED)\\n\\'),
'd0': ';\\n\\',
'd1': ';\\n\\',
'd2': [';', 'delay', 'between', 'inversion', 'and', 'detection\\n\\'],
'd3': ('=', 300, ';\\n\\'),
'd4': ('=', 400, ';\\n\\'),
'd5': ('=', 400, ';\\n\\'),
'd9': [';', 'DAF', '(Delay', 'After', 'Flash)\\n\\'],
'd30': ('=', 2, ';', '1st', 'time', 'scale', 'increment\\n\\'),
'd31': ['=', 't*d30', ';\\n\\'],
'a': ('=',
 256,
 ';',
 '(RESERVED)',
 'number',
 'of',
 'transient',
 'averages',
```

```
 '(TRANS)\\n\\'),
'h': ('=',
 10,
 ';',
 '(CONVENTION)',
 'number',
 'of',
 'shots',
 'per',
 'point',
 '(INTG',
 'and',
 'SPT)\\n\\'),
'n': ('=',
 4,
 ';',
 '(CONVENTION)',
 'number',
 'of',
 'sweeps',
 'to',
 'accumulate\\n\\'),
'srt': ('=',
 1000,
 '*',
 'srtu',
 ';',
 '(RESERVED)',
 'SRT',
 '-',
 'Shot',
 'Repetition',
 'Time',
 '(srtu=1.02',
 'us)\\n\\'),
'b': ('=', 1, ';', 'SRT', 'increment', 'for', 'SRT', 'Recovery\\n\\'),
'r': ('=',
 10,
 ';',
 'Number',
 'of',
 180,
 'pulses',
 'for',
 'CP-GM',
 'sequence\\n\\'),
's': ('=', 1, ';\\n\\'),
```

```
't': ['=', 't+1', ';\\n\\'],
'w': ('=',
 800,
 ';',
 '(CONVENTION)',
 'Split',
 'point',
 'of',
 'sweeps',
 '(points)\\n\\'),
'dx': [';', 'increment', 'x-axis\\n\\'],
'dy': [';', 'evolution', 'time', 'axis\\n\\'],
'aa0': ('=', 50, '\\n\\'),
'aa1': ('=', 20, '\\n\\'),
'af0': ('=', 0, ';\\n\\'),
'af1': ('=', -80000, ';\\n\\'),
'af2': ('=', 0, ';\\n\\'),
'as1': ('=', 1, ';\\n\\'),
'as0': ('=', 0, ';\\n\\'),
'df1': ('=', 250, 'kHz', '\\n\\'),
'end': 'exp6\\n\\',
'\\n': None,
'PlsSPELLISTSlct': '2-step',
'PlsSPELPhPrgEx': 'Normal',
'PlsSPELPrg': 'PulseDefinitionType2/hole_burning_AWG2.exp',
'PlsSPELPrgTxt': '\\',
'dim': ['s[512]', ';', 'for', 'set-up', '2pulse', 'echo\\n\\'],
'dim1': ['s[256]',
 ';',
 'for',
 'nutation',
 'experiment',
 'to',
 'adjust',
 'the',
 'selective',
 'pi/2',
 'pi',
 'dur',
 'length\\n\\'],
'dim2': ['s[4096]', ';', 'for', 'set-up', '3pulse', 'echo\\n\\'],
'dim3': ['s[256]',
 ';',
 'for',
 'nutation',
 'experiment',
 'to',
```

```
 'adjust',
 'the',
 'selective',
 'pi/2',
 'pi',
 'dur',
 'length\\n\\'],
'dim4': <map at 0x9c46388>,
'dim5': <map at 0x9c46408>,
'dim6': <map at 0x9c46548>,
'': [';', 'QUAD', 'detection', 'with', 'Integration\\n\\'],
'asg1': ['+a', '-a', '+a', '-a\\n\\'],
'bsg1': ['+b', '-b', '+b', '-b\\n\\'],
'shot': ['i=1', 'to', 'a\\n\\'],
'd1\\n\\': None,
'dig': ['[sg1]', ';', 'acquisition\\n\\'],
'next': 'k\\n\\',
'for': ['y=1',
 'to',
 'sy',
 ';',
 'loop',
 'on',
 'the',
 'selective',
 'pi',
 'pulse\\n\\'],
'totscans(n)': [';', 'output', 'of', 'total', 'number', 'of', 'scans\\n\\'],
'p0=p20': [';', 'recall', 'original', 'pulse', 'length\\n\\'],
'dx=p20': [';', 'assignment', 'of', 'x-axis\\n\\'],
'sweep': ['x=1', 'to', 'sx', ';', 'pulse', 'length', 'sweep', 'loop\\n\\'],
'acq': ['[sg1]', ';', 'acquisition\\n\\'],
'p0=p0+d30': [';', 'increment', 'pulse', 'length\\n\\'],
'dx=dx+d30': [';', 'increment', 'delay', 'time\\n\\'],
'scansdone(k)': [';', 'output', 'of', 'scans', 'done\\n\\'],
'p2=p20': [';', 'start', 'inversion', 'pulse', 'length\\n\\'],
'p2=p2+d30\\n\\': None,
'dy=p20': [';', 'assignment', 'of', 'time', 'axis\\n\\'],
'dy=dy+d30': [';', 'increment', 'delay', 'time\\n\\'],
'dy=d2': [';', 'initialize', 'evolution', 'time\\n\\'],
'dy=dy+d31': [';', 'increment', 'delay', 'time\\n\\'],
'PlsSPELShpTxt': None,
'Psd1': <map at 0x9c5b588>,
'Psd10': <map at 0x9c5bac8>,
'Psd11': <map at 0x9c5bd48>,
'Psd12': <map at 0x9c5bfc8>,
'Psd13': <map at 0x9c58148>,
```

```
'Psd14': <map at 0x9c58308>,
'Psd15': <map at 0x9c58448>,
'Psd16': <map at 0x9c585c8>,
'Psd17': <map at 0x9c58748>,
'Psd18': <map at 0x9c58908>,
'Psd19': <map at 0x9c58a48>,
'Psd2': <map at 0x9c58bc8>,
'Psd20': <map at 0x9c58d88>,
'Psd21': <map at 0x9c58ec8>,
'Psd22': <map at 0x9c4e088>,
'Psd23': <map at 0x9c4e288>,
'Psd24': <map at 0x9c4e408>,
'Psd25': <map at 0x9c4e608>,
'Psd26': <map at 0x9c4e848>,
'Psd27': <map at 0x9c4ea48>,
'Psd28': <map at 0x9c4ebc8>,
'Psd29': <map at 0x9c4ee08>,
'Psd3': <map at 0x9c3b0c8>,
'Psd30': <map at 0x9c3b248>,
'Psd31': <map at 0x9c3b408>,
'Psd32': <map at 0x9c3b6c8>,
'Psd33': <map at 0x9c3b848>,
'Psd34': <map at 0x9c3ba08>,
'Psd35': <map at 0x9c3bbc8>,
'Psd36': <map at 0x9c3bdc8>,
'Psd4': <map at 0x9c3bf88>,
'Psd5': <map at 0x9c53208>,
'Psd6': <map at 0x9c53448>,
'Psd7': <map at 0x9c53608>,
'Psd8': <map at 0x9c53808>,
'Psd9': <map at 0x9c53a48>,
'QuadDetect': True,
'RF1Prg': <map at 0x9c6a488>,
'RF2Prg': <map at 0x9c78b88>,
'ReplaceMode': 'Off',
'ShotRepTime': (1020.0, 'us'),
'ShotsPLoop': 100,
'SmoothAllowed': False,
'SmoothPoints': 1,
'SptProgress': (100, '%'),
'StochMode': False,
'SweepsPExp': 1,
'TriggerTimeOut': (67, 's'),
'XAxisChan': 'AWG1',
'XAxisParam': 'Frequency',
'XAxisPls': 1,
'XAxisQuant': ['Magnetic', 'Field'],
```

```
 'XSpecRes': 512,
 'YAxisChan': 'AWG1',
 'YAxisParam': 'Frequency',
 'YAxisPls': 1,
 'YAxisQuant': ['Magnetic', 'Field'],
 'YSpecRes': 1,
 'BaselineCorr': 'Off',
 'NbScansAcc': 1,
 'NbScansDone': 1,
 'NbScansToDo': 1,
 'SmoothMode': 'Manual',
 'SOURCE': None,
 "'20200305_NOSLwT_QBand_FSE.DSC'": None,
 'END_SOURCE': None,
 'PROCESS': "'prPseudoMod'",
 'PAR_STR': ('Harmonic', '=', 1),
 'MDATE': ['03/10/20', '13:48:13']}
```

[80]:
```
def define_xaxis_xepr(parameter):
    npoint = int(parameter['XPTS'])
    xmin = float(parameter['XMIN'])
    xwid = float(parameter['XWID'])
    xmax = xmin + xwid
    xaxis = np.linspace(xmin,xmax,npoint)
    return xaxis
```

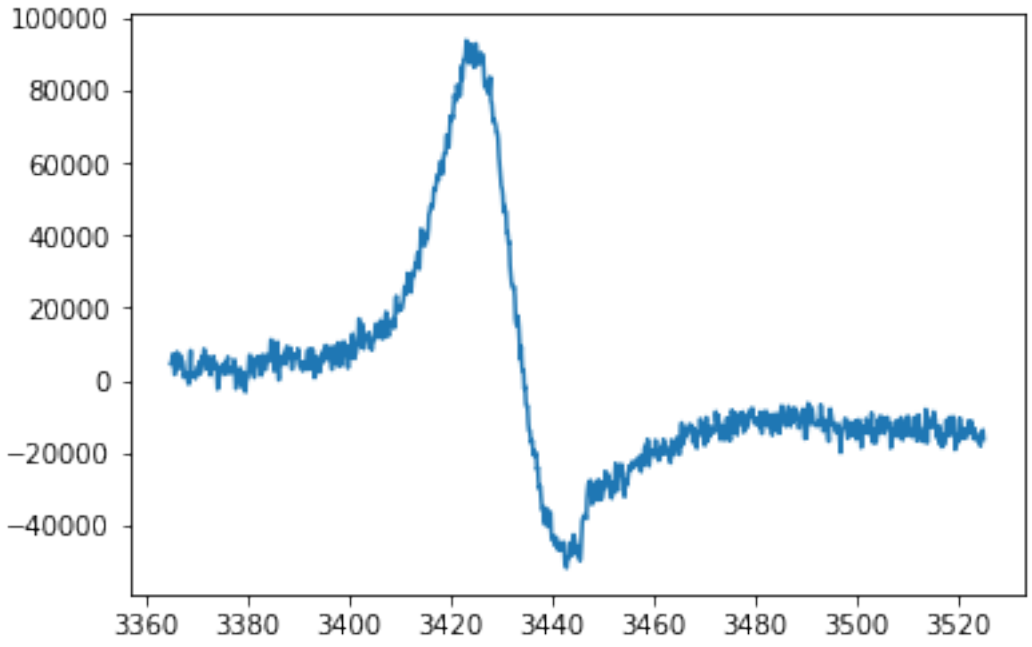
[42]:
```
def define_xaxis_winepr(parameter):
    npoint = int(parameter['ANZ'])
    xmin = float(parameter['GST'])
    xwid = float(parameter['HSW'])
    xmax = xmin + xwid
    xaxis = np.linspace(xmin,xmax,npoint)
    return xaxis
```

[44]:
```
magneticfield1 = define_xaxis_winepr(par)
```

[46]:
```
import matplotlib.pyplot as plt
```

[48]:
```
plt.plot(magneticfield1,data)
```

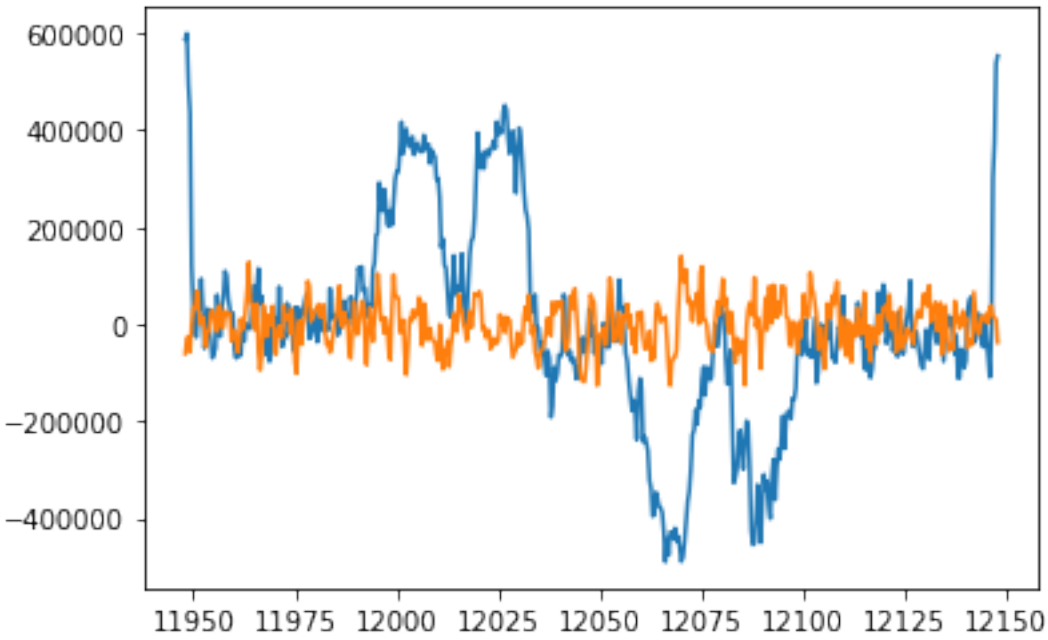
[48]:
```
[<matplotlib.lines.Line2D at 0x92fd408>]
```



[107]:
```
magneticfield2 = define_xaxis_xepr(par2)
```

[108]:
```
plt.plot(magneticfield2,data2)
```

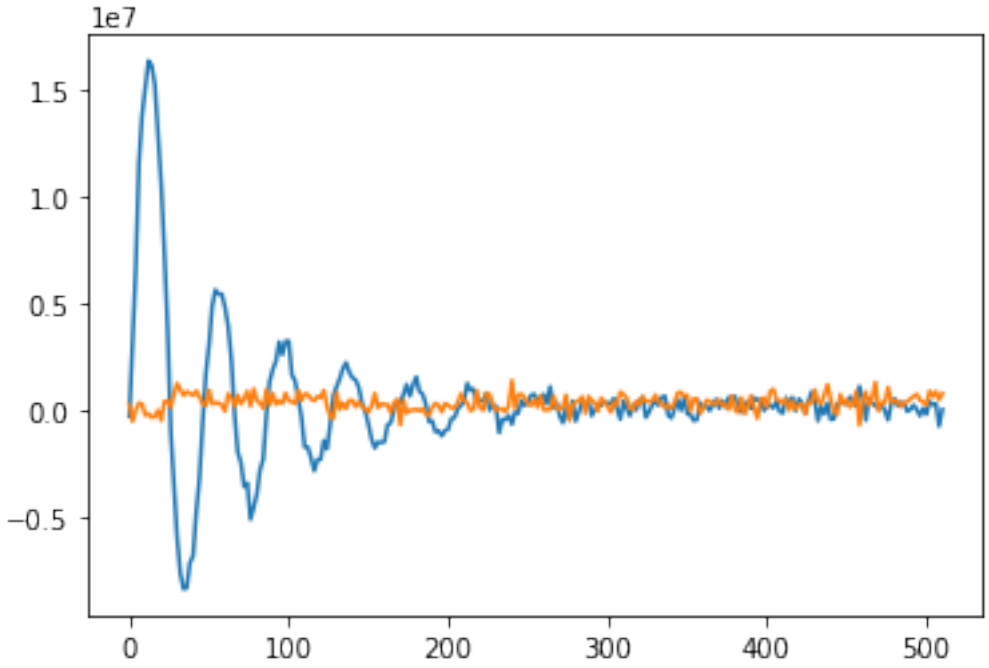
[108]:
```
[<matplotlib.lines.Line2D at 0x9887cc8>,
 <matplotlib.lines.Line2D at 0x9b04a08>]
```



[110]:
```
filename3 = '\\20200305_NOSLwT_QBand_EchoNut_aa0_50.DSC'
full_filename3 = foldername+filename3
data3,par3 = xepr(full_filename3)
magneticfield3 = define_xaxis_xepr(par3)
```

[111]:
```
plt.plot(magneticfield3,data3)
```

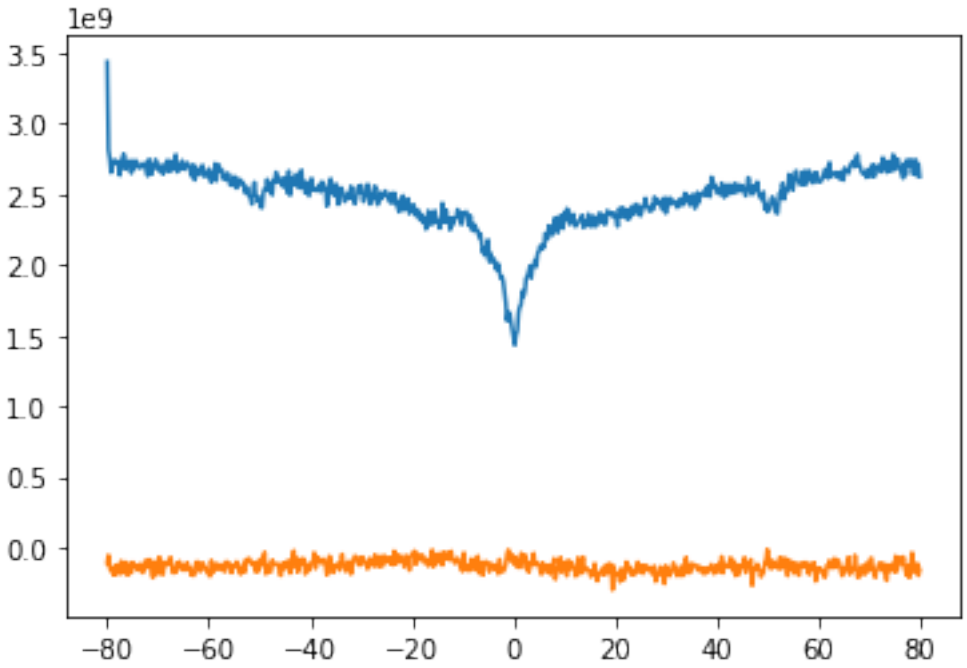
[111]:
```
[<matplotlib.lines.Line2D at 0xba7d0c8>,
 <matplotlib.lines.Line2D at 0x997ebc8>]
```



[112]:
```
filename4 = '\\20200305_NOSLwT_EDNMR_1.DSC'
full_filename4 = foldername+filename4
data4,par4 = xepr(full_filename4)
magneticfield4 = define_xaxis_xepr(par4)
```

[113]:
```
plt.plot(magneticfield4,data4)
```

[113]:
```
[<matplotlib.lines.Line2D at 0xbd561c8>,
 <matplotlib.lines.Line2D at 0xbd4ccc8>]
```



[114]:
```
filename5 = '\\20200207_20180308_NOSLwt+DT+W_InvRecTrans_aa0_50_aa1_2_30K.DSC'
full_filename5 = foldername+filename5
data5,par5 = xepr(full_filename5)
magneticfield5 = define_xaxis_xepr(par5)
```

[115]:
```
data5.shape
```

[115]:
```
(2048, 80)
```

[ ]:
```

```