Example of processing FT-ICR data

Marc-André Delsuc - Joensuu August 2018

First load some libraries

In [1]:
```
from __future__ import print_function, division
import array
import os.path as op
import numpy as np

import matplotlib as mpl
import matplotlib.pylab as plt
%matplotlib inline
```

locate the data-set which should be in the repository

In [2]:
```
ls files/FTICR-Files/080617-insulin_2M_MS_000001.d
```

```
Wesley_nanoLCMS04042012.m/ analysis.baf_xtr*
analysis.baf*              desktop.ini*
analysis.baf_idx*          fid*
```

so now we can read it

In [3]:
```
BASE = 'files/FTICR-Files/ESI_pos_Ubiquitin_000006.d'
```

In [4]:
```
F = open(  op.join(BASE, 'fid'), 'rb')
tbuf = F.read()
fid = np.array(array.array('i',tbuf))
F.close()
```

In [5]:
```
len(fid)
```

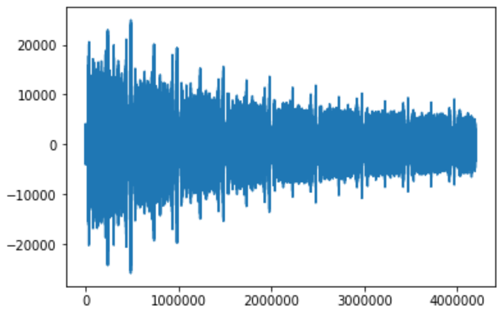
Out[5]:
```
4194304
```

In [6]:
```
fid[1000:1010]
```

Out[6]:
```
array([ 1137,   125,  1623,   952,  -138,  -366,  -857, -1464, -1604,
       -1247], dtype=int32)
```

In [7]:
```
plt.plot(fid)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x10a8f95c0>]
```



Fourier processing

small complication

There are 2 fft packages available

- `numpy.fft`
  - simple, basic, efficient
- `scipy.fftpack`
  - slightly faster, more complex

we are going to use `numpy.fft`

In [8]:
```
sp = np.fft.fft(fid)
```

In [9]:
```
print (sp[10])
len(sp)
```

```
(-70742.64832249525+217578.94774257578j)
```

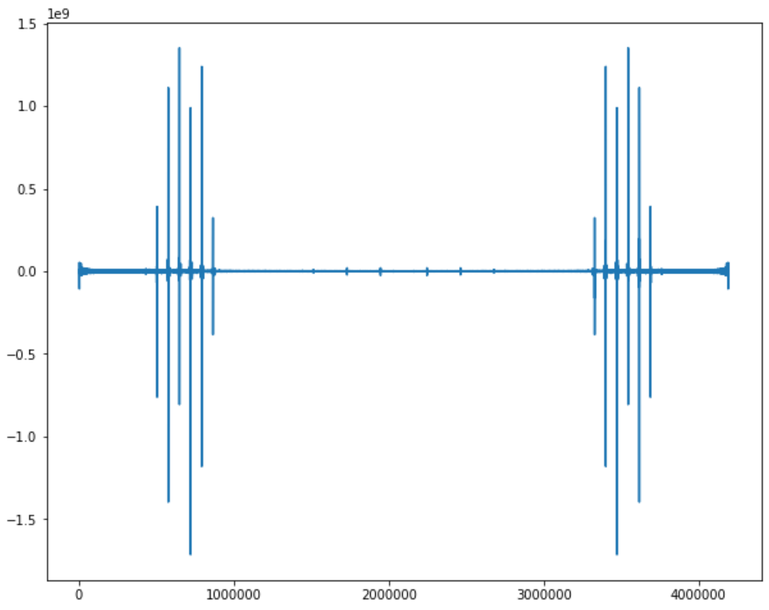
Out[9]:
```
4194304
```

In [10]:
```python
# default size of the figures
mpl.rcParams['figure.figsize'] = [10.0,8.0]
# this helps matplotlib to draw the huge vectors we're using (4-8Mpoints)
mpl.rcParams['agg.path.chunksize'] = 100000
```

In [11]:
```python
plt.plot(sp)
```

```
/Users/mad/anaconda3/lib/python3.6/site-packages/numpy/core/numeric.py:492: ComplexWarning: Casting complex value
s to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x10ad7d7b8>]
```



3 remarks

- imaginary discarded (eventually silently)
- phase
- symetry

1 syntax remark

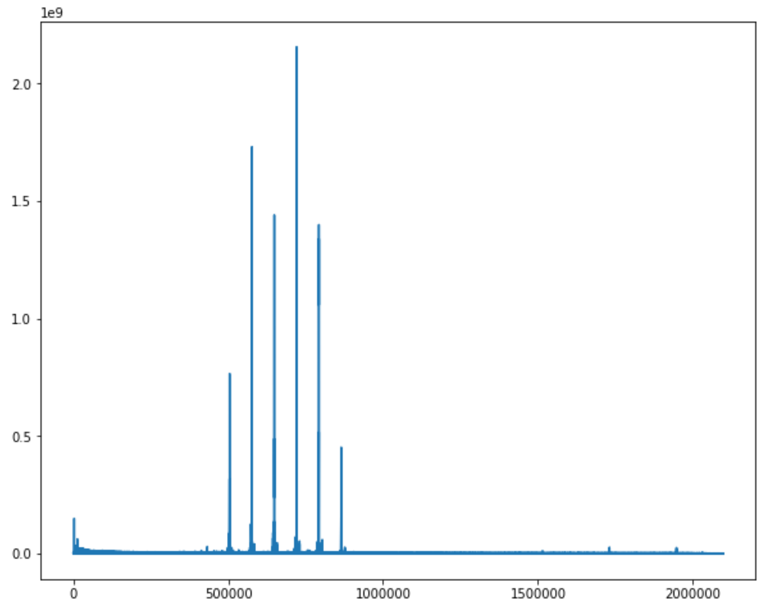
In [12]:
```python
from numpy.fft import fft, rfft
sp = rfft(fid)
print (sp[0])
len(sp)
```

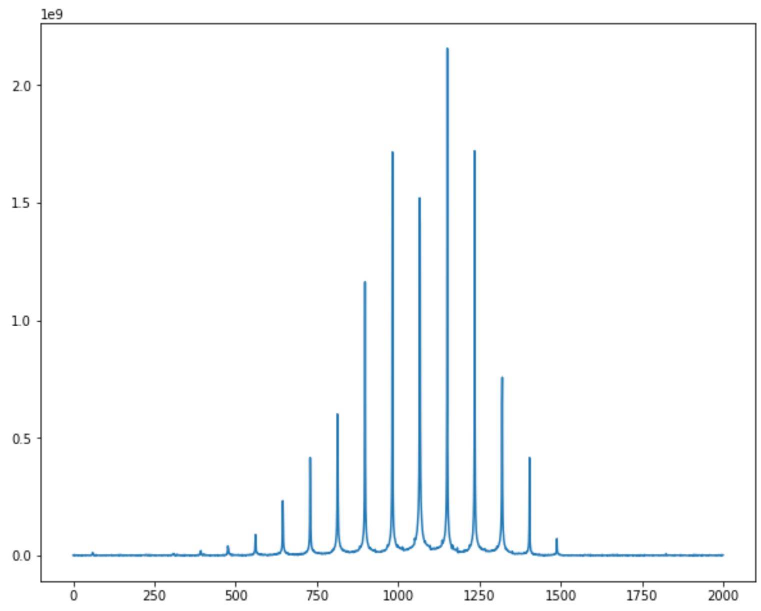
```
(-2681443+0j)
```

Out[12]:
```
2097153
```

In [13]:
```python
plt.plot(abs(sp))      # absolute value is modulus
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x10b00d6a0>]
```

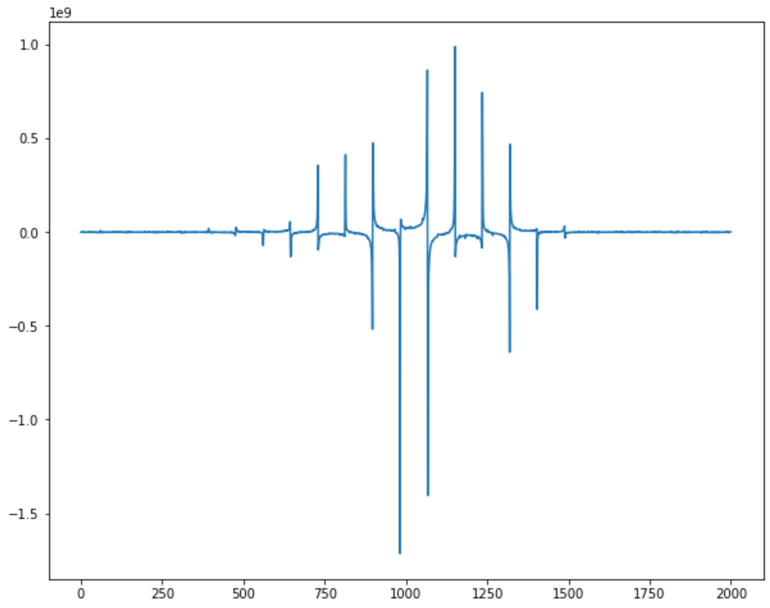


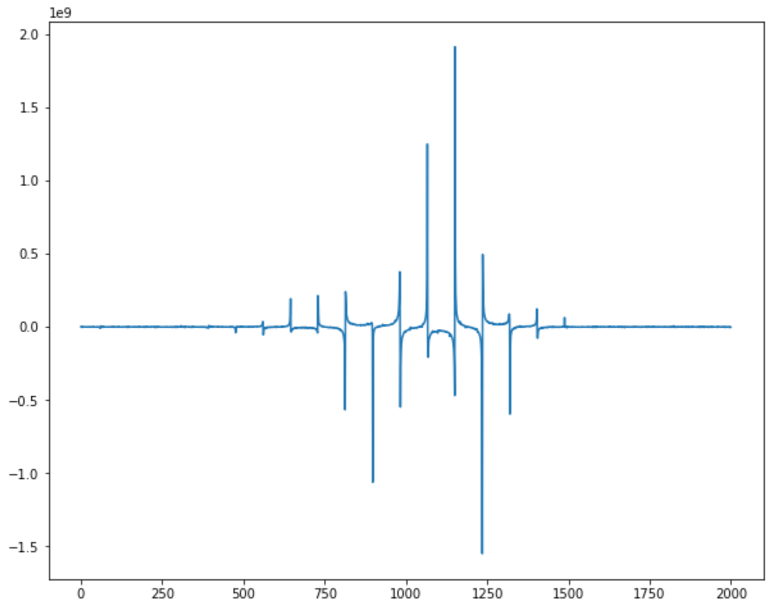
In [14]:
```
# let's zoom in
plt.plot(abs(sp[720000:722000]))
```
Out[14]:
```
[<matplotlib.lines.Line2D at 0x10b062400>]
```


In [15]:
```
# without the abs() we see the phase rotating
plt.plot(sp[720000:722000])
plt.figure()
# we rotate by 90° the phase
plt.plot(1j*sp[720000:722000])

```

```
/Users/mad/anaconda3/lib/python3.6/site-packages/numpy/core/numeric.py:492: ComplexWarning: Casting complex value
s to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x14cf54978>]
```

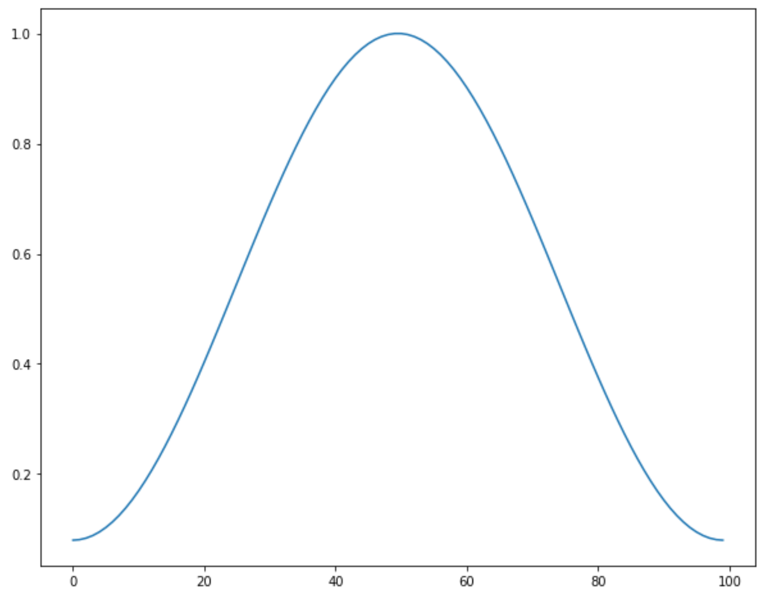




2 possible improvements

- improved resolution by apodisation
- recovery of the full resolution by zerofilling

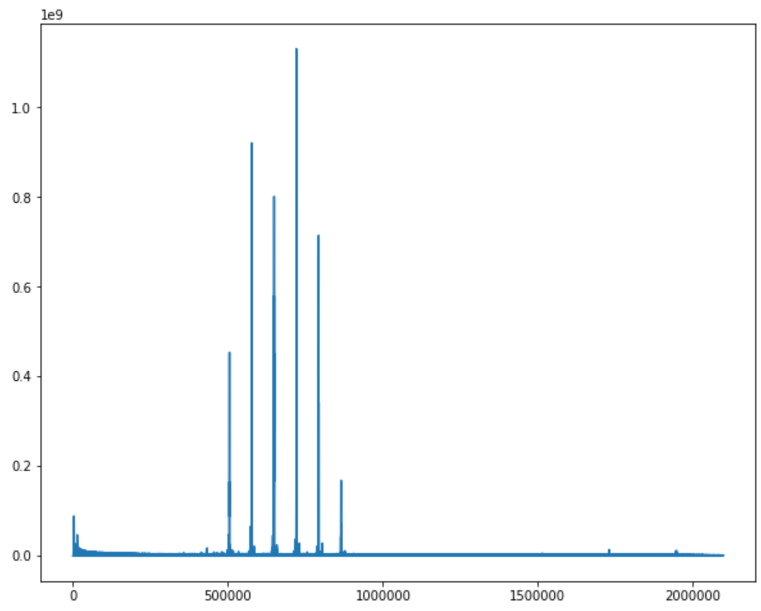
apodisation

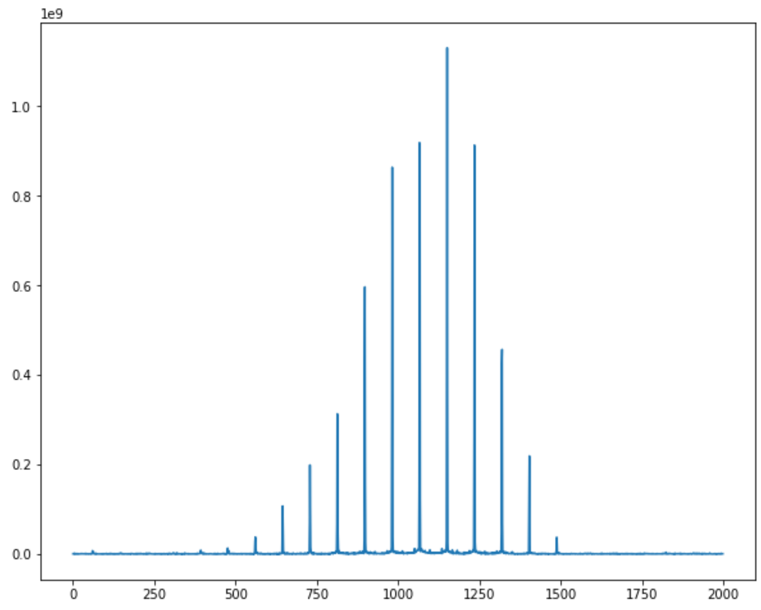
In [16]:
```
plt.plot(np.hamming(100))

```
Out[16]:
```
[<matplotlib.lines.Line2D at 0x14cde40b8>]
```


there is also `blackman` `hanning` `bartlett` ...

In [17]:
```
sp2 = rfft( np.hamming(len(fid))*fid )
```

In [18]:
```
plt.plot(abs(sp2))
plt.figure()
plt.plot(abs(sp2[720000:722000]))
```
Out[18]:
```
[<matplotlib.lines.Line2D at 0x1512c6278>]
```





Notice how lineshape is improved, and the isotopic pattern intensities restored

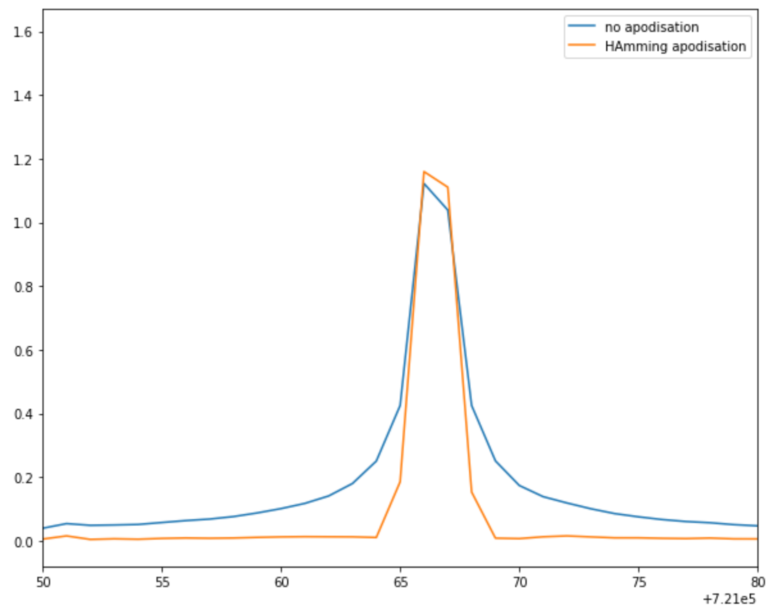
We can superimpose both processing, zooming on a line you can see that FWHM is sligthly worse when apodized.

Theory is 1.5 for hamming window, but does not really show here.

In [19]:
```
plt.plot(abs(sp)/max(sp), label="no apodisation")
plt.plot(abs(sp2)/max(sp2), label="HAmming apodisation")
plt.xlim(721050,721080)
plt.legend()
```

```
/Users/mad/anaconda3/lib/python3.6/site-packages/numpy/core/numeric.py:492: ComplexWarning: Casting complex value
s to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

Out[19]:
```
<matplotlib.legend.Legend at 0x1512c6c50>
```



zerofilling

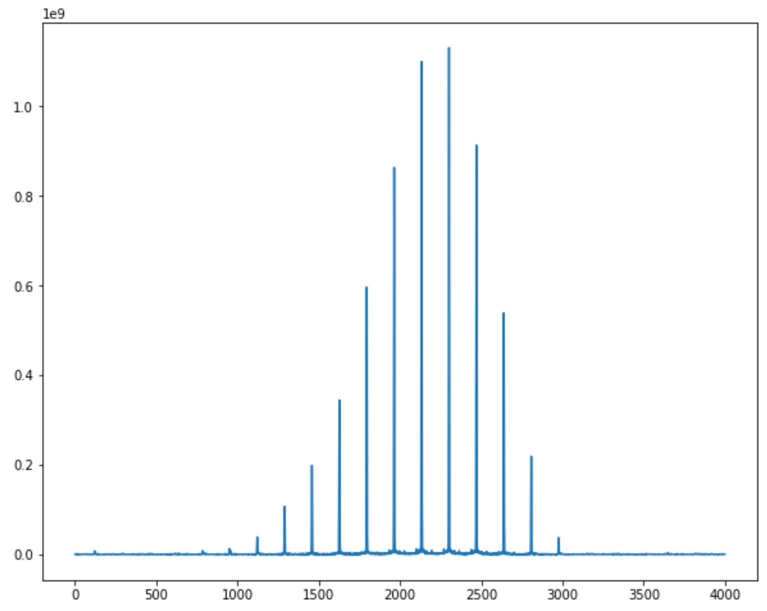
zerofilling consists in adding zero at the end of the FID to interpolate points in the spectrum and smooth the result

(and an example of slighlty different numpy arithmetics)

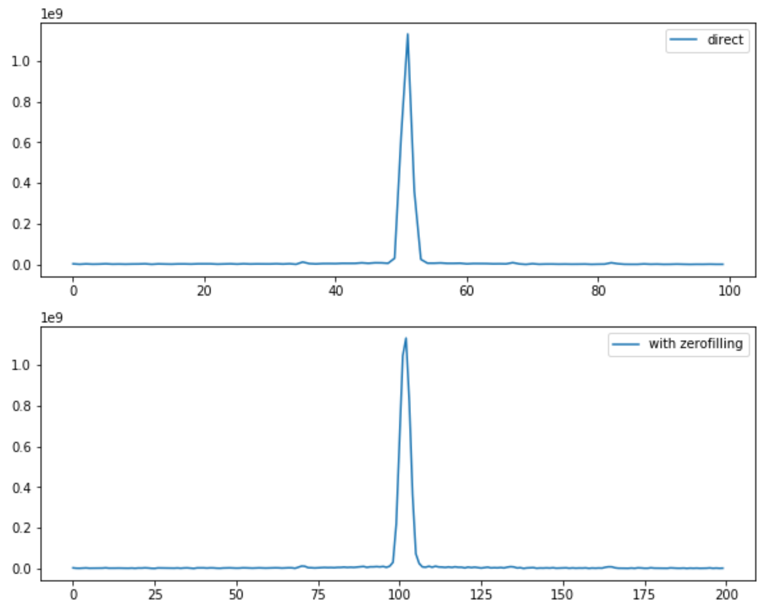
In [20]:
```
n = len(fid)
fidzf = np.zeros(2*n)

```

In [21]:
```
fidzf[:n] = fid[:]
fidzf[:n] *= np.hamming(n)
spzf = abs(rfft( fidzf ))
plt.plot(abs(spzf[2*720000:2*722000]))
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x166f37cc0>]
```



lineshape and isotopic pattern intensities further improved !

In [22]:
```
plt.subplot(211)
plt.plot(abs(sp2[721100:721200]), label='direct')
plt.legend()
plt.subplot(212)
plt.plot(abs(spzf[2*721100:2*721200]), label='with zerofilling')
plt.legend()
```
Out[22]:
```
<matplotlib.legend.Legend at 0x178fd45c0>
```


m/z Calibration

In FT-ICR, the frequency f depends on the magnetic field B_o the charge z and the mass m :

$$f = \frac{B_o z}{m} \quad \Rightarrow \quad m/z = \frac{B_o}{f}$$

We are going to use a sligthly extended equation which takes cares of experimental imperfections:

$$m/z = \frac{M_1}{M_2 + f}$$

where M_1 and M_2 are constants determined precisely by a calibration procedure

parameter files

FT-ICR parameters are stored in the .method files

In [23]:
```
ls 'files/FTICR-Files/ESI_pos_Ubiquitin_000006.d/ESI_pos_150_3000.m/apexAcquisition.method'
```
```
files/FTICR-Files/ESI_pos_Ubiquitin_000006.d/ESI_pos_150_3000.m/apexAcquisition.method*
```

In [24]:
```
cat 'files/FTICR-Files/ESI_pos_Ubiquitin_000006.d/ESI_pos_150_3000.m/apexAcquisition.method'
```
```
<?xml version='1.0' encoding='UTF-8'?>
<method version="Apex_4_alpha">

<methodmetadata>
<primarykey>
  <methodfilepath>D:\Data\Training_June_2010\ESI_pos_Ubiquitin_000006.d\ESI_pos_150_3000.m\apexAcquisition.method
</methodfilepath>
  <!-- Actual file that method was originally loaded from -->
  <methodloadfromfile>D:\Data\Training_June_2010\ESI_pos_Ubiquitin_000005.d\ESI_pos_150_3000.m\apexAcquisition.me
thod</methodloadfromfile>
  <datarootpath>D:\Data</datarootpath>
  <methodname>ESI_pos_150_3000</methodname>
  <username>Administrator</username>
  <samplename>100 fmol</samplename>
  <sampledescription></sampledescription>
  <date>Jun_29_2010 11:07:53.234</date>
  <userlevel>EXPERT</userlevel>
  <!--  HystarName = [] -->
</primarykey>
<version><versionname>ApexControl 1.5.0.42.8 (May 21  2010)</versionname></version>
```

utility

parameter are stored in a XML file, we can look through the file and get the parameters manually, but it is also possible to simply build a dictionary from the entries.

Here a simple mini parser, using standard python library

In [25]:
```python
from xml.dom import minidom
def read_param(filename):
    """
        Open the given file and retrieve all parameters from apexAcquisition.method
        NC is written when no value for value is found

        structure : <param name = "AMS_ActiveExclusion"><value>0</value></param>

        read_param returns  values in a dictionnary
    """
    xmldoc = minidom.parse(filename)

    x = xmldoc.documentElement
    pp = {}
    children = x.childNodes
    for child in children:
        if (child.nodeName == 'paramlist'):
            params = child.childNodes
            for param in params:
                if (param.nodeName == 'param'):
                    k = str(param.getAttribute('name'))
                    for element in param.childNodes:
                        if element.nodeName == "value":
                            try:
                                v = str(element.firstChild.toxml())
                                #print v
                            except:
                                v = "NC"
                    pp[k] = v
    return pp
```

In [26]:
```python
pfile = op.join(BASE, 'ESI_pos_150_3000.m', 'apexAcquisition.method')
param = read_param(pfile)
print (param['TD'])
```
```
4194304
```

In [27]:
```python
# print the 10 first entries
len(list(param.keys()))
```
Out[27]:
```
781
```

In [28]:
```python
param['ML1']
```
Out[28]:
```
'1.8427001783361462E8'
```

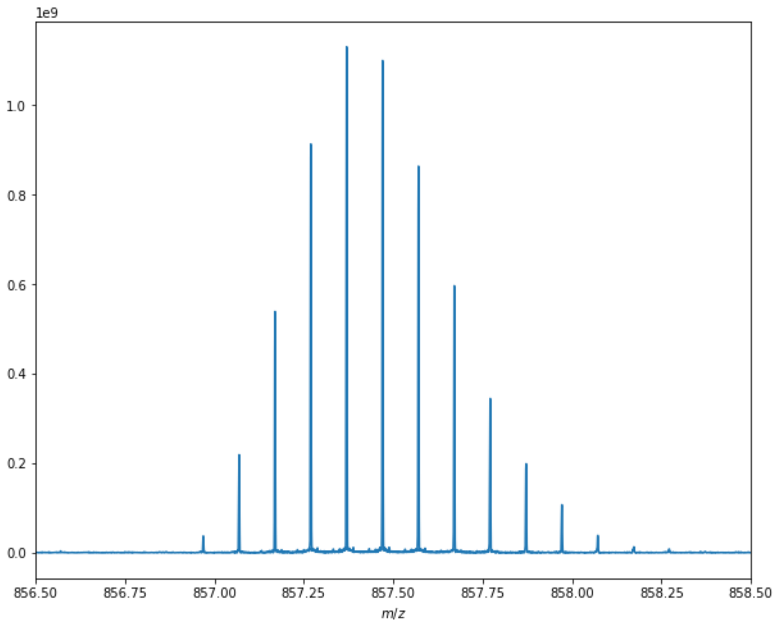
In [29]:
```python
# we find the parameters in the parameter file
sw = float(param['SW_h_Broadband'])
ml1 = float(param['ML1'])
ml2 = float(param['ML2'])
print('Spectral width: {}'.format(sw) )
print('constant M_1: {}'.format(ml1))
print('constant M_2: {}'.format(ml2))
```
```
Spectral width: 625000.0
constant M_1: 184270017.83361462
constant M_2: 5.039161102310875
```

In [30]:
```python
faxis = np.linspace(0, sw, len(spzf))    # the freq axis from 0 to sw
```

In [31]:
```python
mzaxis = ml1/(ml2+faxis)    # and the mzaxis
```

In [32]:

```python
plt.plot(mzaxis, spzf)
plt.xlim(856.5, 858.5)
plt.xlabel("$m/z$");
```

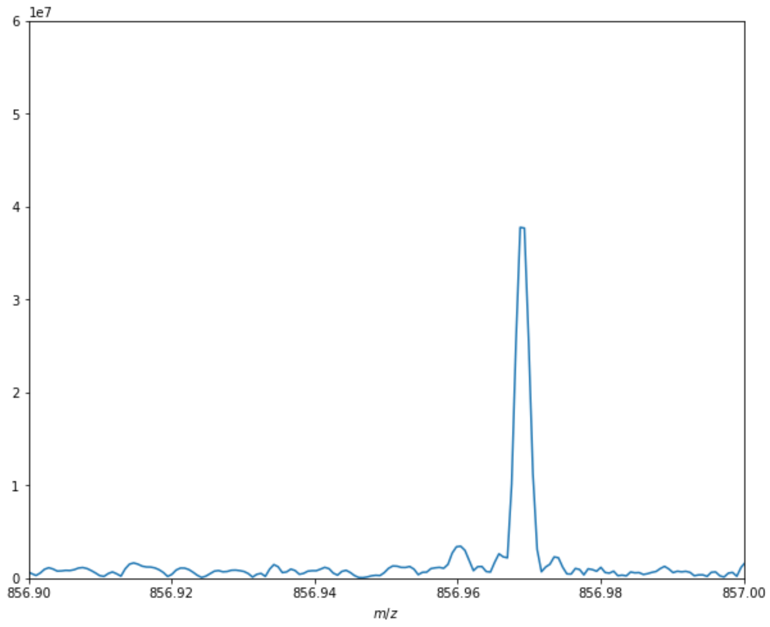


we can zoom on the monoisotopic peak, and try to determine precisely its value.

We know the theorical mass is 856.969496 (for the $z = 10$ state).

In [33]:

```python
plt.plot(mzaxis, spzf)
plt.xlim(856.9, 857.0)
plt.ylim(0,6E7)
plt.xlabel("$m/z$");
```



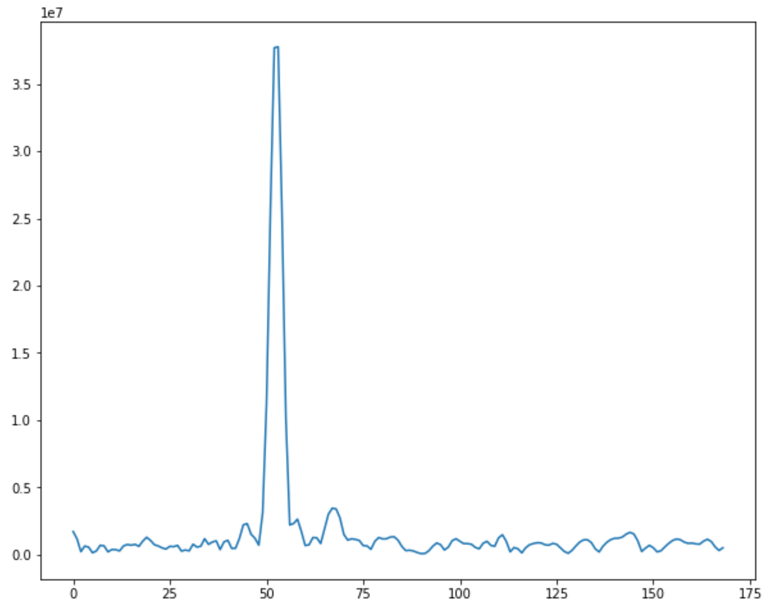
In [34]:

```python
# these functions convert back and forth from index to m/z
def itomz(val, N):
    """transform index to m/z for N points,
    using current ml1 ml2 and sw
    """
    f = sw*val/N
    return ml1/(ml2+f)
def mztoi(m, N):
    """transform m/z to index for N points,
    using current ml1 ml2 and sw
    """
    f = ml1/m - ml2
    return N*f/sw
theo = 856.9694962104804
def ppm(theorical, measured):
    return 1E6*(measured-theorical)/measured
```

we can compute the vector coordinates of the previous zoom window

In [35]:
```python
start = int( mztoi(857.0, len(spzf)))
end = int( mztoi(856.9, len(spzf)))
print("start: {}  - end: {}".format(start,end))
plt.plot(spzf[start:end])
top = spzf[start:end].argmax() + start
meas1 = itomz(top,len(spzf))
print("maximum is at: {}, for m/z: {}".format(top,meas1))
print("For an error of {:.3f} ppm".format(ppm(theo, meas1)))
```

```
start: 1442924  - end: 1443093
maximum is at: 1442977, for m/z: 856.9689406761887
For an error of -0.648 ppm
```



In [36]:
```python
# peak barycenter
bary = 0.0
s = 0
for i in range(-3, +4):
    bary += i*spzf[i+top]
    s += spzf[i+top]
mbary = itomz(bary/s+top, len(spzf))
print ("peak barycenter is at {}".format(mbary))
print("For an error of {:.3f} ppm".format(ppm(theo, mbary)))
```

```
peak barycenter is at 856.9692194653925
For an error of -0.323 ppm
```

packing it up

Now we can simply build a function doing all this at once :

In [37]:

```python
import glob
def read_fticr(folder):
    """
    load and process the data Solarix Apex FTICR file found in folder
    uses the calibration from the parameter file

    eg:
    spectrum,axis = read_fticr('FTICR/Files/bruker ubiquitin file/ESI_pos_Ubiquitin_000006.d')
    """
    # find and load parameters
    L = glob.glob(op.join(folder,"*","apexAcquisition.method"))
    if len(L)>1:
        raise Exception( "You have more than 1 apexAcquisition.method file in the %s folder, using the first on
    elif len(L) == 0:
        raise Exception( "You don't have any apexAcquisition.method file in the  %s folder, please double check
    param = read_param(L[0])

    # load data
    n = int(param['TD'])
    fidzf = np.zeros(2*n)
    with  open(  op.join(BASE, 'fid'), 'rb') as F:  # 'with' a better way of reading a file
        tbuf = F.read(4*int(param['TD']))
        fidzf[:n] = np.array(array.array('i',tbuf)) # [:] is less memory intensive

    # process
    fidzf[:n] *= np.hamming(n)
    spectrum = abs( rfft( fidzf ) )

    # calibrate
    sw = float(param['SW_h_Broadband'])
    ml1 = float(param['ML1'])
    ml2 = float(param['ML2'])
    faxis = np.linspace(0, sw, len(spectrum))    # the freq axis from 0 to sw
    mzaxis = ml1/(ml2+faxis)    # and the mzaxis
    return spectrum, mzaxis

```

In [38]:

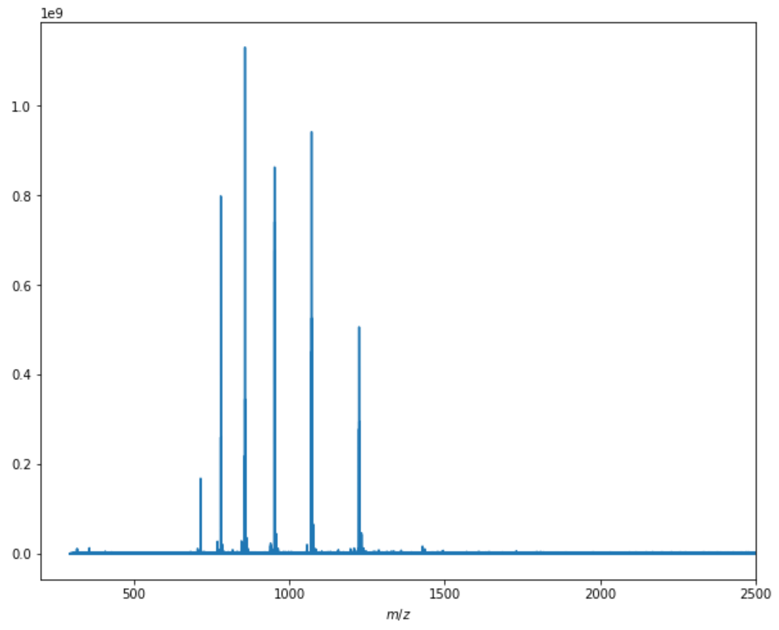
```python
spectrum,axis = read_fticr('files/FTICR-Files/ESI_pos_Ubiquitin_000006.d')
```

In [39]:

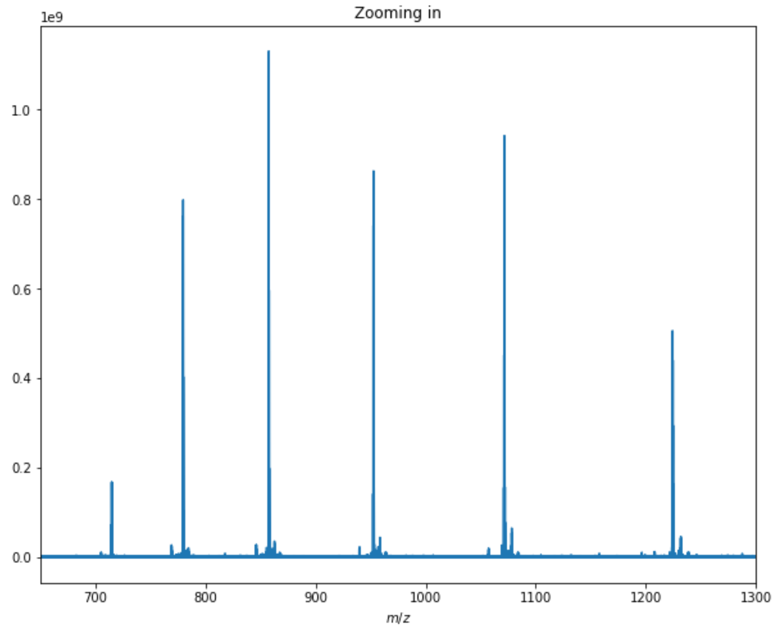
```python
# %matplotlib
plt.plot(axis, spectrum)
plt.xlabel("$m/z$")
plt.xlim(200,2500)
```

Out[39]:

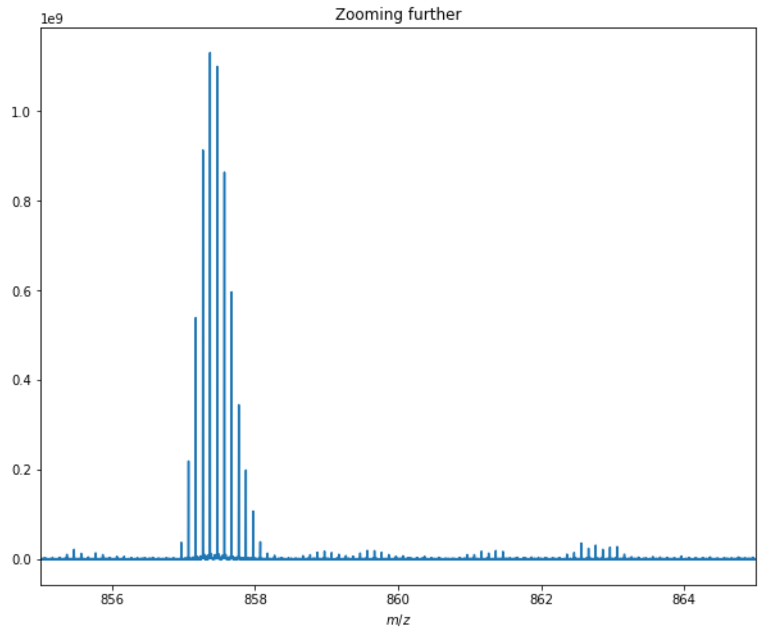
```
(200, 2500)
```



In [40]:
```python
plt.plot(axis, spectrum)
plt.xlabel("$m/z$")
plt.xlim(650,1300)
plt.title("Zooming in")
```
Out[40]:
```
Text(0.5,1,'Zooming in')
```



In [41]:
```python
plt.plot(axis, spectrum)
plt.xlabel("$m/z$")
plt.xlim(855,865)
plt.title("Zooming further");
```



Doing the same thing with SPIKE

SPIKE is a complete software suite we're building to do this, *and much more*

It is distributed here : https://bitbucket.org/delsuc/spike (https://bitbucket.org/delsuc/spike)

However, the program is still in development, stay tuned !

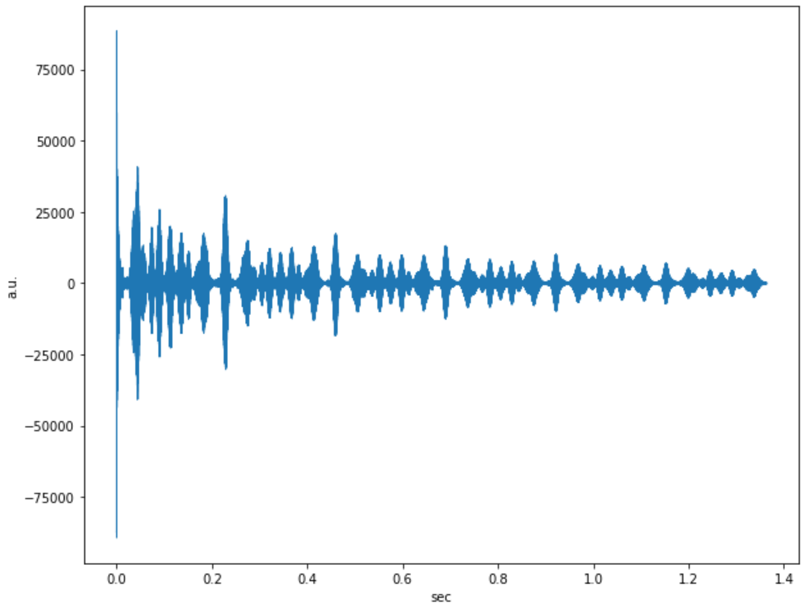
In [42]:
```python
import sys
sys.path.append('/Users/mad/')  # adapt this to your own set-up
import spike
from spike.File import Apex
```

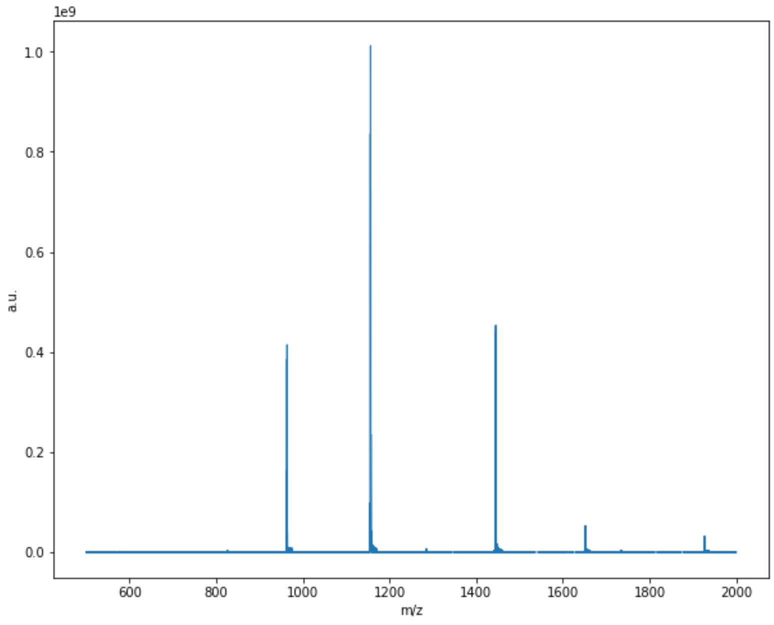
```
    ========================
          SPIKE
    ========================
    Version     : 0.99.0
    Date        : 06-03-2018
    Revision Id : 369
    ========================
*** Importing  << zoom3D >> Failed ***
plugins loaded:
Bruker_NMR_FT,  Bucketing,  FTMS_calib,  Fitter,  Linear_prediction,  PALMA,  Peaks,  apmin,  bcorr,  fastclean,
pg_sane,  rem_ridge,  sane,  sg,  test,  urQRd,  wavelet,
type spike.plugins.report() for a short description
and spike.plugins.report('module_name') for complete documentation on one plugin
```

In [43]:
```python
File = 'files/FTICR-Files/080617-insulin_2M_MS_000001.d'
dd = Apex.Import_1D(File)
```

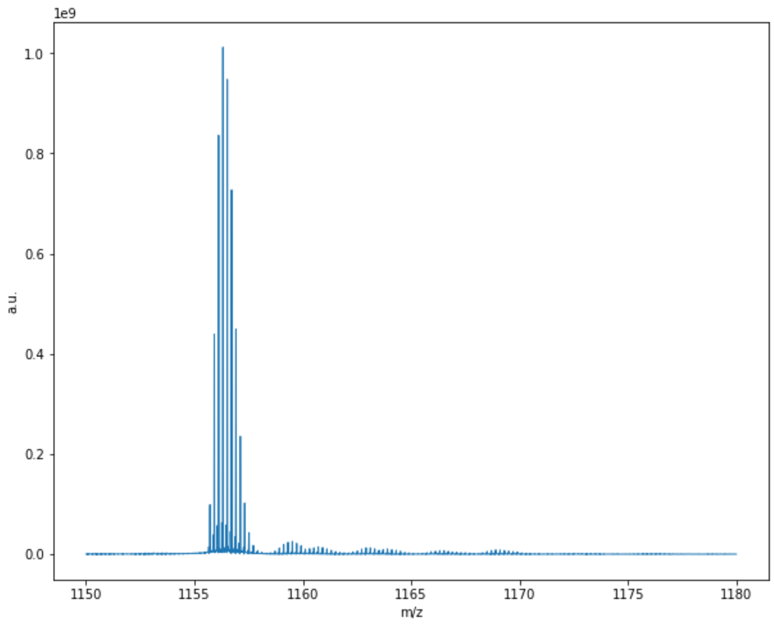
In [44]:
```python
dd.unit = "sec"
dd.display()
```

Out[44]:
```
1D data-set
Axis F1 :FT-ICR report axis at 769.230769 kHz,  2097152 real points,  from physical mz =   187.692   to m/z = 5000
.000  R max (M=400) = 984074
data-set is real
```



In [45]:
```
dd.hamming().zf(4).rfft().modulus()
dd.unit = "m/z"
dd.display(zoom=(500,2000))
```
Out[45]:
```
1D data-set
Axis F1 :FT-ICR report axis at 769.230769 kHz,  4194304 real points,  from physical mz =  187.692   to m/z = 5000
.000  R max (M=400) = 1968148
data-set is real
```


In [46]:
```
dd.pp(threshold=1E7)
print ("detected %d peaks"%len(dd.peaks))
```
```
PP Threshold: 10000000.0
detected 146 peaks
```

In [47]:
```
dd.display(zoom=(1150,1180))
dd.display_peaks(zoom=(856.5, 858.5))
```


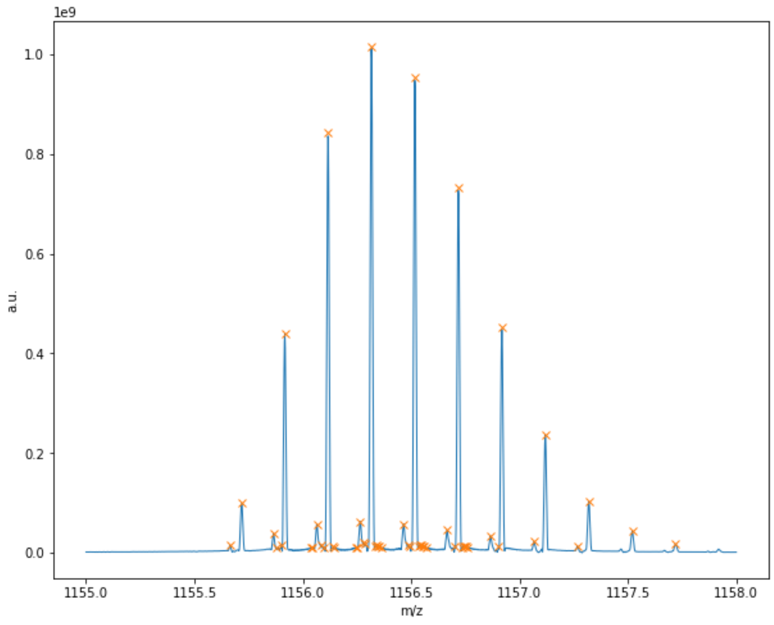
In [48]:
```
p = dd.peaks[40]
```

In [49]:
```
p.report()
```
Out[49]:
```
'40, 40, 544737.00, 340973582.28'
```

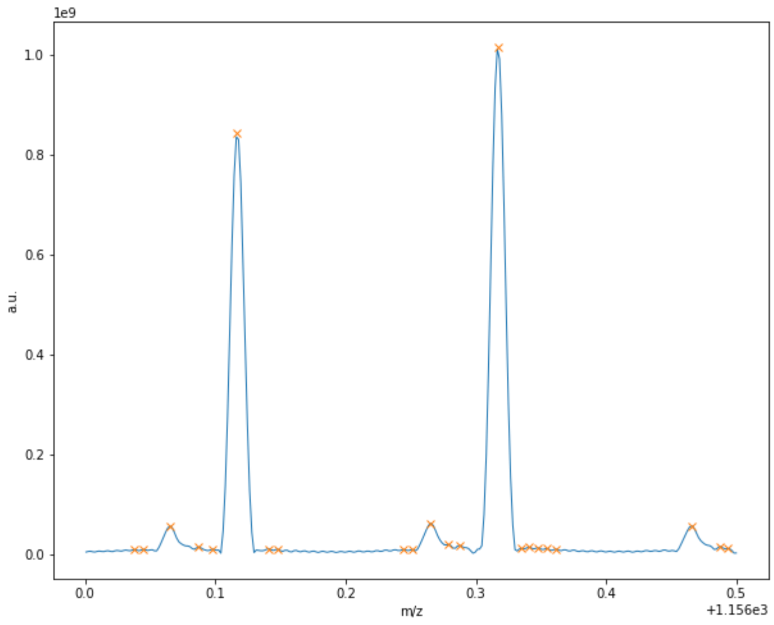
In [50]:
```
p.label = "Peak 40"
p.report()  # in index
p.report(f=dd.axis1.itomz)  # provides in m/z
# string is :
# id, label, m/z, intensity, width (0 if not determined)
```
Out[50]:
```
'40, Peak 40, 1444.89, 340973582.28'
```

In [51]:
```
dd.centroid()
for p in dd.peaks:
    p.label = "%.5f"%(dd.axis1.itomz(p.pos))
```
```
/Users/mad/anaconda3/lib/python3.6/site-packages/scipy/optimize/minpack.py:785: OptimizeWarning: Covariance of th
e parameters could not be estimated
  category=OptimizeWarning)
```

In [52]:
```
# this switches to a different display mode
z = (1155,1158)
dd.display(zoom=z)
dd.display_peaks(zoom=z) #, peak_label=True)
```



In [53]:
```
# this switches to a different display mode
z = (1156,1156.5)
dd.display(zoom=z)
dd.display_peaks(zoom=z) #, peak_label=True)
```



1e9
a.u.
m/z
+1.156e3