

Protein Identification Target/Decoy and False Discover Rate

COMPUTATIONAL BIOLOGY 1

PROF.DR/IBRAHIM AL-AWADY ALSAMMAN



Team Names:

- Omnia Tarek Farghaly Moahemd (sec 3 group 1)
- Afnan Eid Mahmoud (sec 2 group 1)
- Sohaila Essam El-Den Mohamed (sec 4 group 2)
- Arwa Mostafa Kamal Abd El-Wahab (sec 2 group 1)

References

- Fasta File

Uniprot link: <https://www.uniprot.org/uniprot/P48444>

- Raw Data File(mzML)

Pridelink : <https://www.ebi.ac.uk/pride/archive/projects/PXD024124>

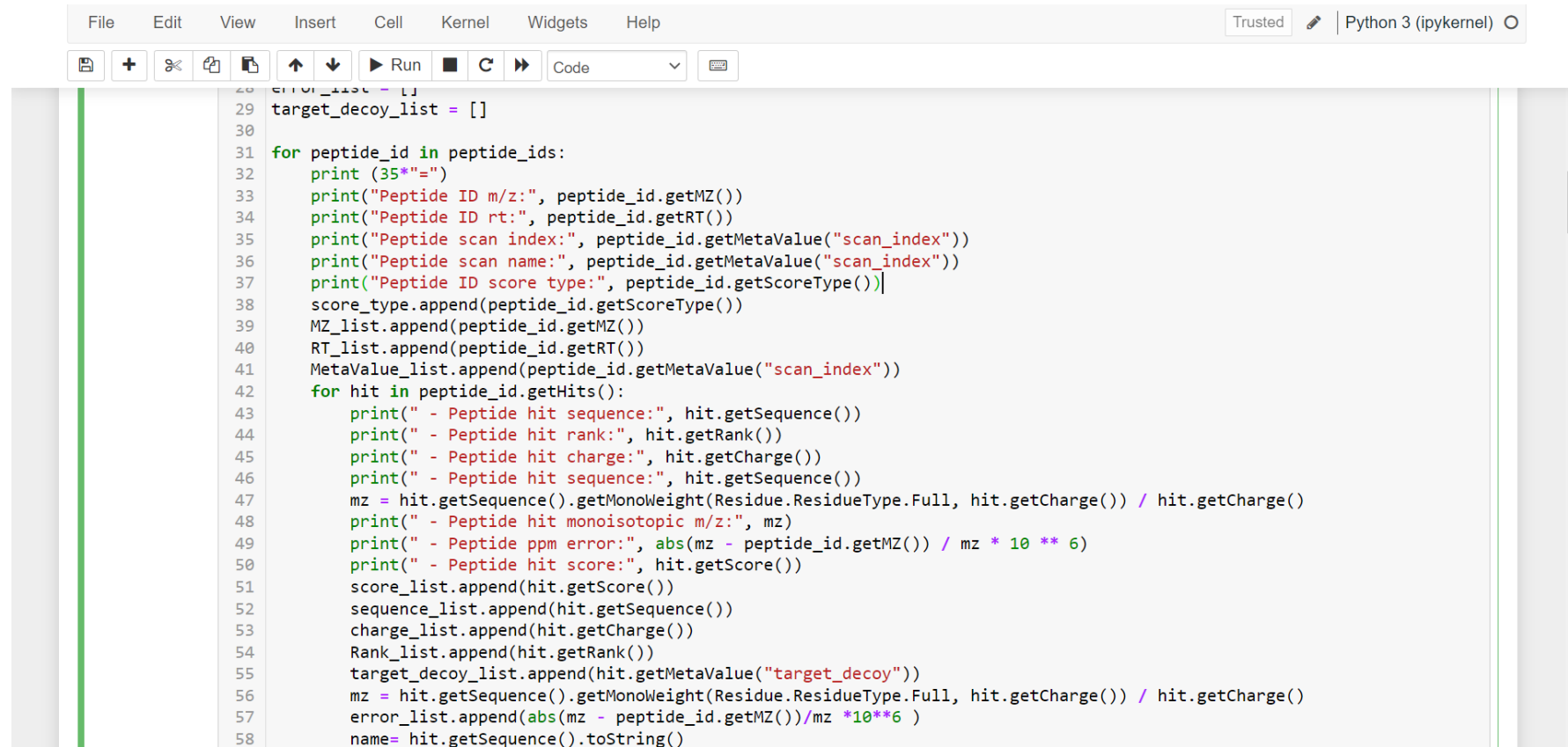
Fasta sequence(Homo sapiens)

- >sp|P48444|COPD_HUMAN Coatmer subunit delta OS=Homo sapiens OX=9606 GN=ARCN1 PE=1 SV=1
MVLLAAAVCTKAGKAIVSRQFVEMTRTRIEGLLAAPKLMNTGKQHTFVETESVRYVYQPM EKLYMV
LITTKNSNILEDLET LRLFSRVIPEYCR ALEENEISEHCFDLIFAFDEIVALGYRENVNLAQIRTFTEMDSHEEKVF
RAVRETQEREAKAEMRRKAKELQQARRDAERQGKKAPGFGGFGSSAVSGGSTAAMITETIIETDKPKVAPA
PARPSGPSKALKLGAKGKEVDNFVDK LKSEGETIMSSSMGKRTSEATKMHAPPINMESVHMKIEEKITLTC
GRDGGLQNMELHGMIMLRISDDKYGRIRLHV ENEDKKGVQLQTHPNVDKKLFTAESLIGLKNPEKSFPV
NSDVGV LKWRLQTTEESFIPLTINCWPSESGNGCDVNIEYELQEDNLELNDVVITIPLPSGVGAPVIGEIDGEY
RHDSRRNTLEWCLPVIDAKNKSGSLEFSIAGQPNDFFPVQVSFVSKKNYCN IQVTKVTQVDGNSPVRFSTE
TTFLVDKYEIL

Identifications Data

- In OpenMS, identifications of peptides, proteins and small molecules are stored in dedicated data structures. These data structures are typically stored to disc as idXML or mzIdentML file. The highest-level structure is ProteinIdentification. It stores all identified proteins of an identification run as ProteinHit objects plus additional metadata (search parameters, etc.). Each ProteinHit contains the actual protein accession, an associated score, and (optionally) the protein sequence
- A PeptideIdentification object stores the data corresponding to a single identified spectrum or feature. It has members for the retention time, m/z, and a vector of PeptideHit objects. Each PeptideHit stores the information of a specific peptide-to-spectrum match or PSM (e.g., the score and the peptide sequence). Each PeptideHit also contains a vector of PeptideEvidence objects which store the reference to one or more (in the case the peptide maps to multiple proteins) proteins and the position therein

Protein and Peptide identification

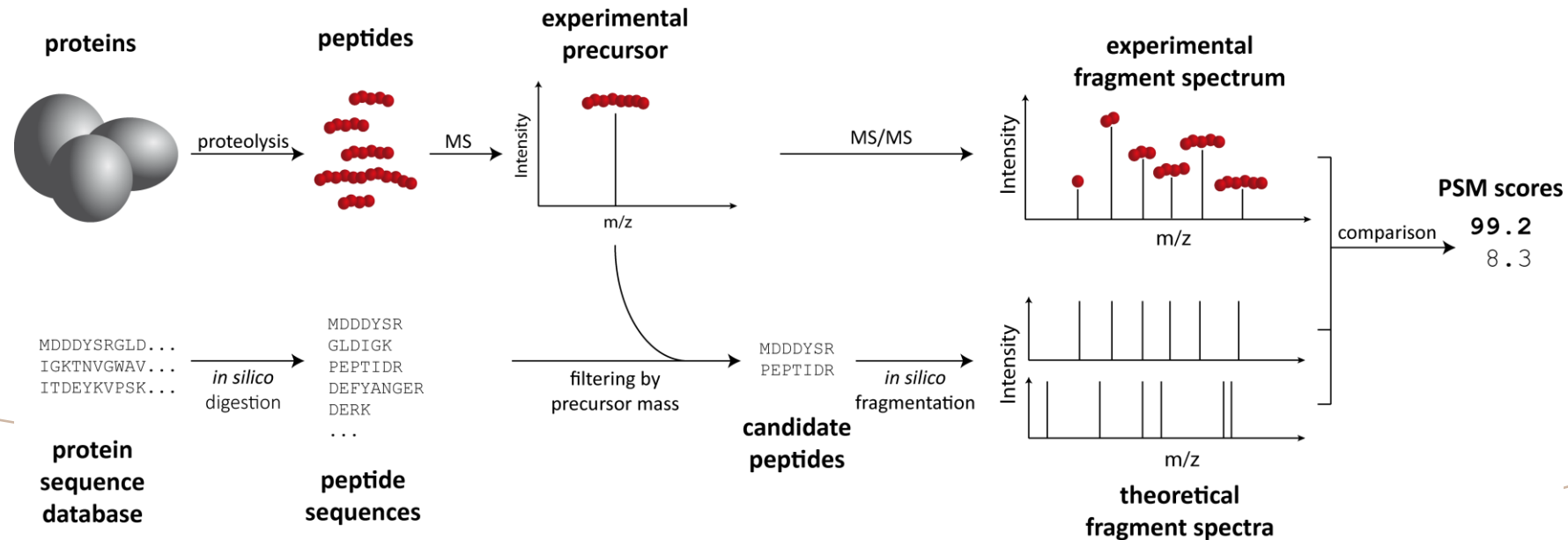


The image shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running code, and other functions. The code is written in Python and processes peptide identification data. It iterates through a list of peptide IDs, printing various attributes like m/z, retention time, scan index, and score type. It also calculates the monoisotopic m/z and ppm error for each hit. The code is as follows:

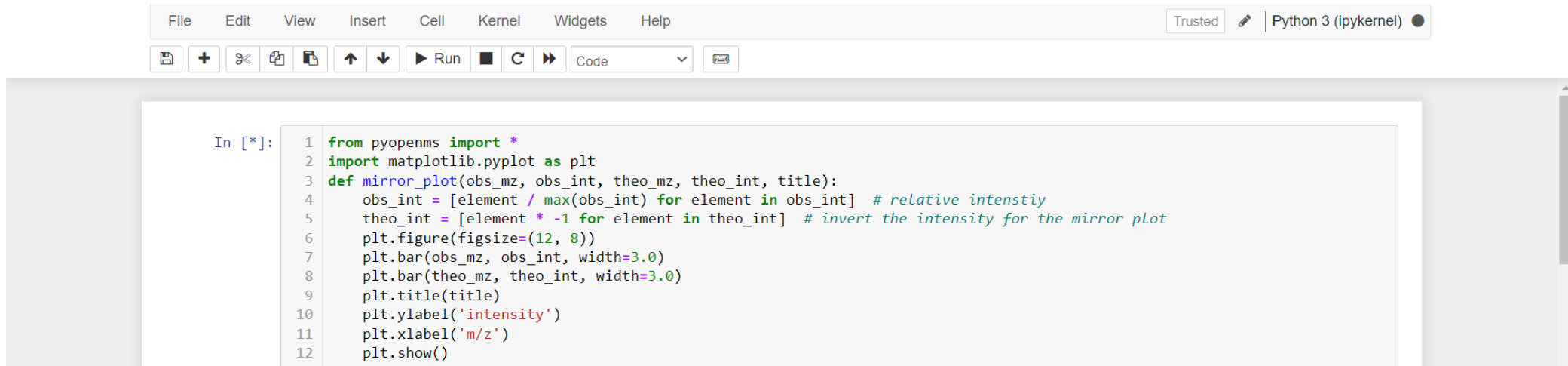
```
28 error_list = []
29 target_decoy_list = []
30
31 for peptide_id in peptide_ids:
32     print (35*"=")
33     print("Peptide ID m/z:", peptide_id.getMZ())
34     print("Peptide ID rt:", peptide_id.getRT())
35     print("Peptide scan index:", peptide_id.getMetaValue("scan_index"))
36     print("Peptide scan name:", peptide_id.getMetaValue("scan_index"))
37     print("Peptide ID score type:", peptide_id.getScoreType())
38     score_type.append(peptide_id.getScoreType())
39     MZ_list.append(peptide_id.getMZ())
40     RT_list.append(peptide_id.getRT())
41     MetaValue_list.append(peptide_id.getMetaValue("scan_index"))
42     for hit in peptide_id.getHits():
43         print(" - Peptide hit sequence:", hit.getSequence())
44         print(" - Peptide hit rank:", hit.getRank())
45         print(" - Peptide hit charge:", hit.getCharge())
46         print(" - Peptide hit sequence:", hit.getSequence())
47         mz = hit.getSequence().getMonoWeight(Residue.ResidueType.Full, hit.getCharge()) / hit.getCharge()
48         print(" - Peptide hit monoisotopic m/z:", mz)
49         print(" - Peptide ppm error:", abs(mz - peptide_id.getMZ()) / mz * 10 ** 6)
50         print(" - Peptide hit score:", hit.getScore())
51         score_list.append(hit.getScore())
52         sequence_list.append(hit.getSequence())
53         charge_list.append(hit.getCharge())
54         Rank_list.append(hit.getRank())
55         target_decoy_list.append(hit.getMetaValue("target_decoy"))
56         mz = hit.getSequence().getMonoWeight(Residue.ResidueType.Full, hit.getCharge()) / hit.getCharge()
57         error_list.append(abs(mz - peptide_id.getMZ())/mz * 10**6)
58         name = hit.getSequence().toString()
```

Peptide Search

- In MS-based proteomics, fragment ion spectra (MS2 spectra) are often interpreted by comparing them against a theoretical set of spectra generated from a FASTA database. OpenMS contains a (simple) implementation of such a "search engine" that compares experimental spectra against theoretical spectra generated from an enzymatic or chemical digest of a proteome (e.g. tryptic digest).

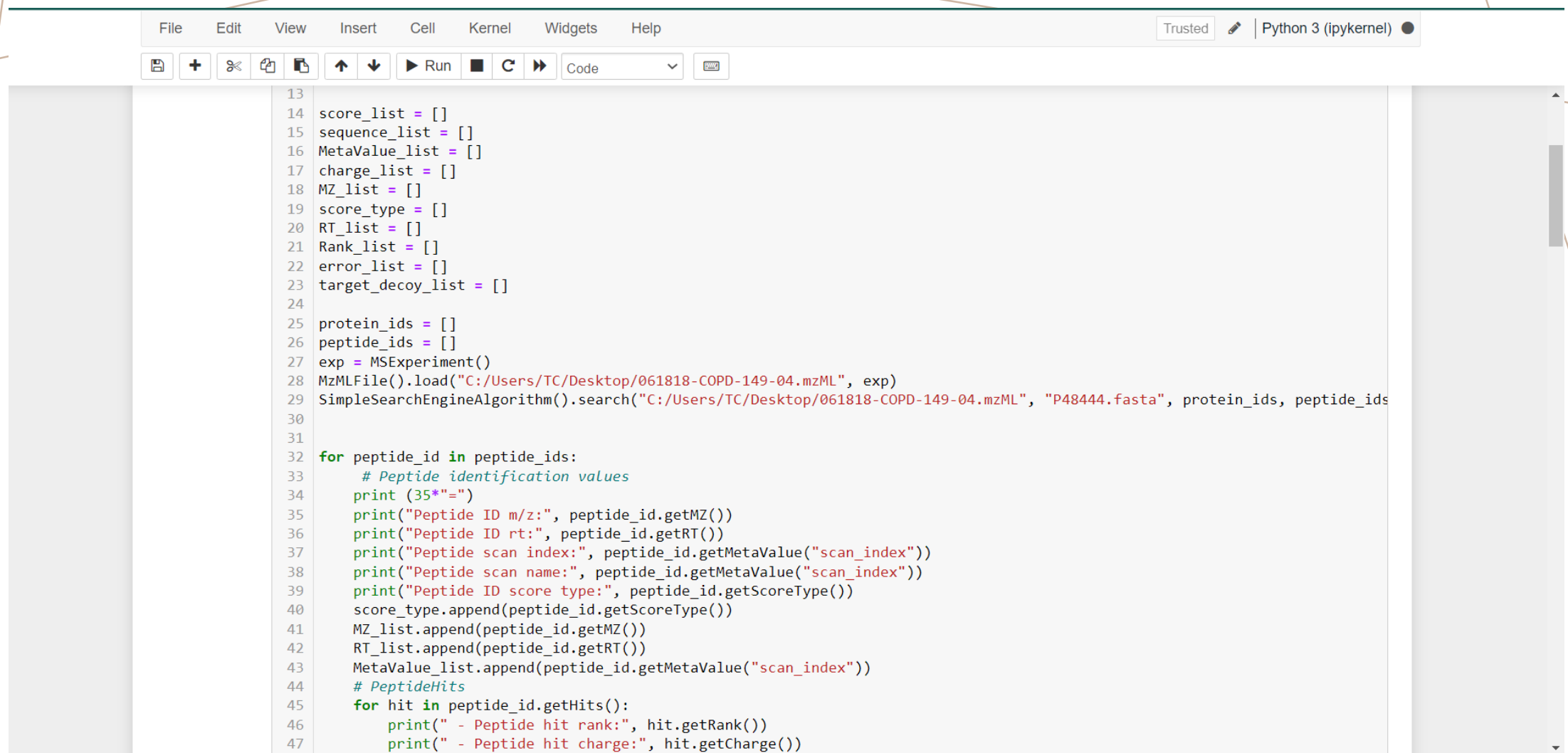


The whole code



```
In [*]: 1 from pyopenms import *
2 import matplotlib.pyplot as plt
3 def mirror_plot(obs_mz, obs_int, theo_mz, theo_int, title):
4     obs_int = [element / max(obs_int) for element in obs_int] # relative intensitiy
5     theo_int = [element * -1 for element in theo_int] # invert the intensity for the mirror plot
6     plt.figure(figsize=(12, 8))
7     plt.bar(obs_mz, obs_int, width=3.0)
8     plt.bar(theo_mz, theo_int, width=3.0)
9     plt.title(title)
10    plt.ylabel('intensity')
11    plt.xlabel('m/z')
12    plt.show()
```

Here we start with writing a function called **mirror_plot** to draw the mirror between the theoretical and experimental data



```
13
14 score_list = []
15 sequence_list = []
16 MetaValue_list = []
17 charge_list = []
18 MZ_list = []
19 score_type = []
20 RT_list = []
21 Rank_list = []
22 error_list = []
23 target_decoy_list = []
24
25 protein_ids = []
26 peptide_ids = []
27 exp = MSEExperiment()
28 MzMLFile().load("C:/Users/TC/Desktop/061818-COPD-149-04.mzML", exp)
29 SimpleSearchEngineAlgorithm().search("C:/Users/TC/Desktop/061818-COPD-149-04.mzML", "P48444.fasta", protein_ids, peptide_ids)
30
31
32 for peptide_id in peptide_ids:
33     # Peptide identification values
34     print (35*"=")
35     print("Peptide ID m/z:", peptide_id.getMZ())
36     print("Peptide ID rt:", peptide_id.getRT())
37     print("Peptide scan index:", peptide_id.getMetaValue("scan_index"))
38     print("Peptide scan name:", peptide_id.getMetaValue("scan_index"))
39     print("Peptide ID score type:", peptide_id.getScoreType())
40     score_type.append(peptide_id.getScoreType())
41     MZ_list.append(peptide_id.getMZ())
42     RT_list.append(peptide_id.getRT())
43     MetaValue_list.append(peptide_id.getMetaValue("scan_index"))
44     # PeptideHits
45     for hit in peptide_id.getHits():
46         print(" - Peptide hit rank:", hit.getRank())
47         print(" - Peptide hit charge:", hit.getCharge())
```

- In the above code we make lists to store the values we mentioned to retrieve it in a data frame after we get the values
- Construction of decoy databases



Code



```
51     score_list.append(hit.getScore())
52     sequence_list.append(hit.getSequence())
53     charge_list.append(hit.getCharge())
54     Rank_list.append(hit.getRank())
55     target_decoy_list.append(hit.getMetaValue("target_decoy"))
56     mz = hit.getSequence().getMonoWeight(Residue.ResidueType.Full, hit.getCharge()) / hit.getCharge()
57     error_list.append(abs(mz - peptide_id.getMZ())/mz *10**6 )
58     name= hit.getSequence().toString()
59     tsg = TheoreticalSpectrumGenerator()
60     theo_spec = MSSpectrum()
61     p = Param()
62     p.setValue("add_y_ions", "true")
63     p.setValue("add_b_ions", "true")
64     p.setValue("add_metainfo", "true")
65     tsg.setParameters(p)
66     peptide = AASequence.fromString(hit.getSequence().toString())
67     tsg.getSpectrum(theo_spec, peptide, 1, 2)
68     # Iterate over annotated ions and their masses
69     spectrum = exp.getSpectrum(peptide_id.getMetaValue("scan_index"))
70     alignment = []
71     spa = SpectrumAlignment()
72     p = spa.getParameters()
73     # use 0.5 Da tolerance (Note: for high-resolution data we could also use ppm by setting the is_relative_tolerance va
74     p.setValue("tolerance", 0.5)
75     p.setValue("is_relative_tolerance", "false")
76     spa.setParameters(p)
77     # align both spectra
78     spa.getSpectrumAlignment(alignment, theo_spec, spectrum)
79     # Print matching ions and mz from theoretical spectrum
80     print("Number of matched peaks: " + str(len(alignment)))
```

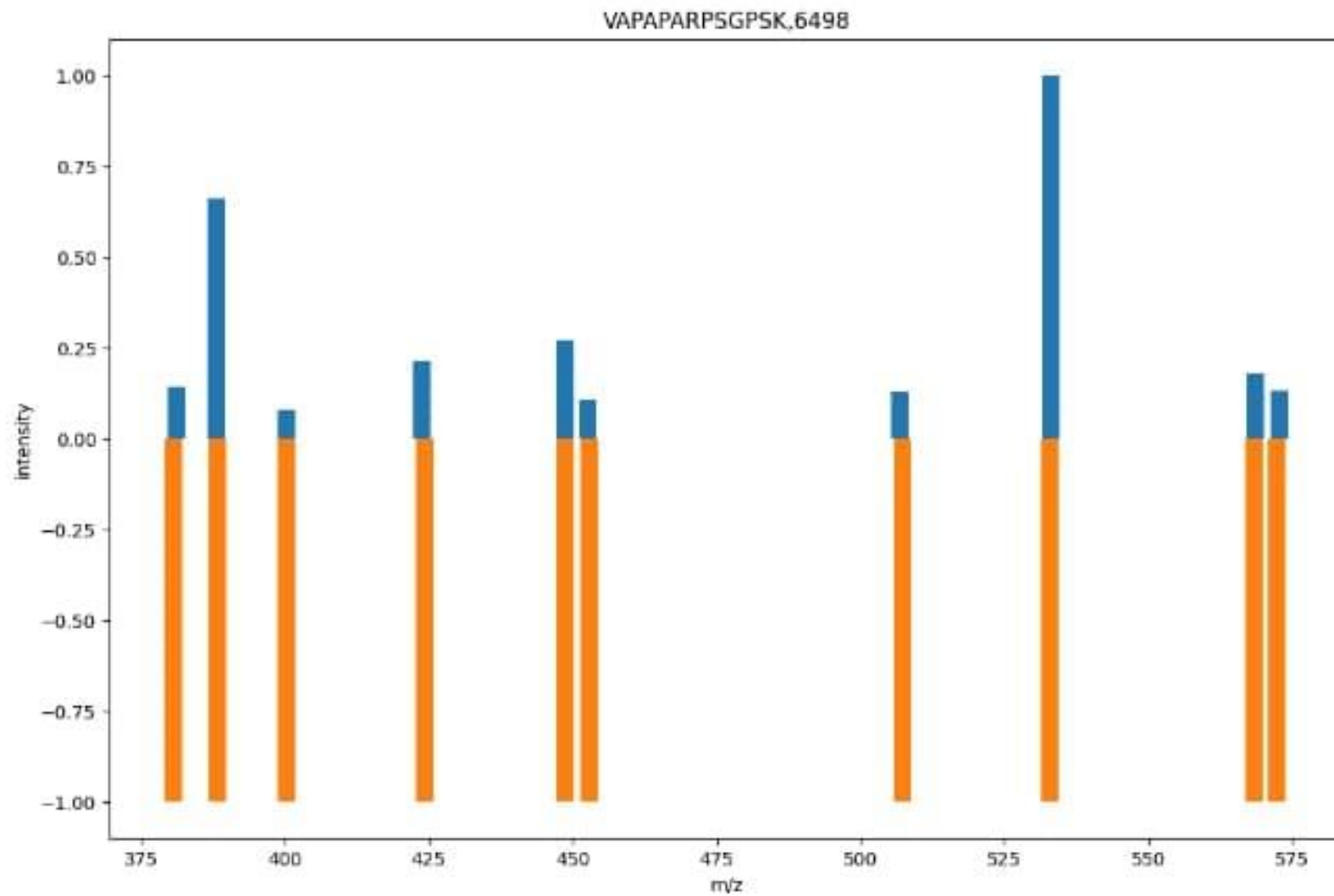


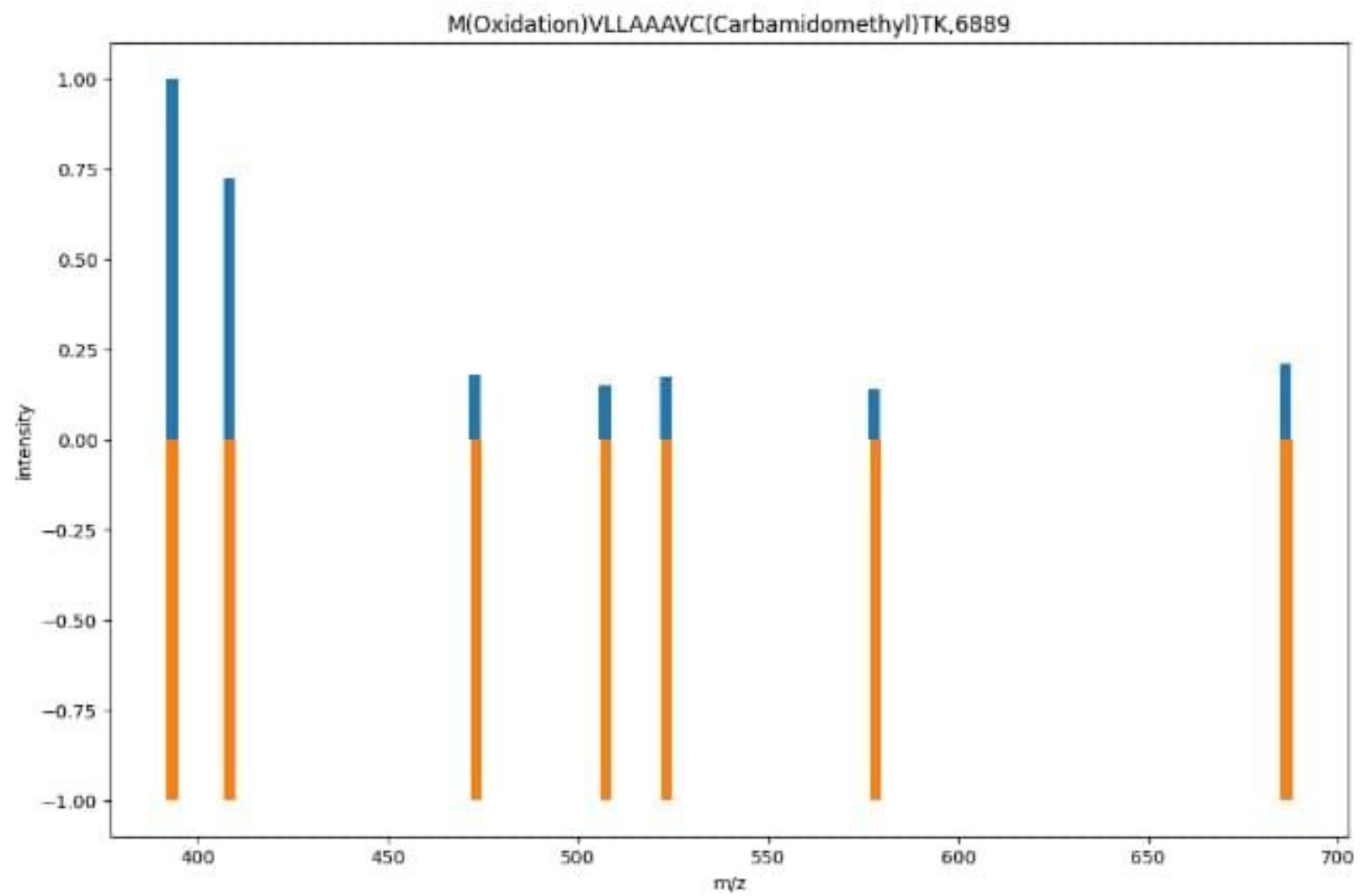
Code

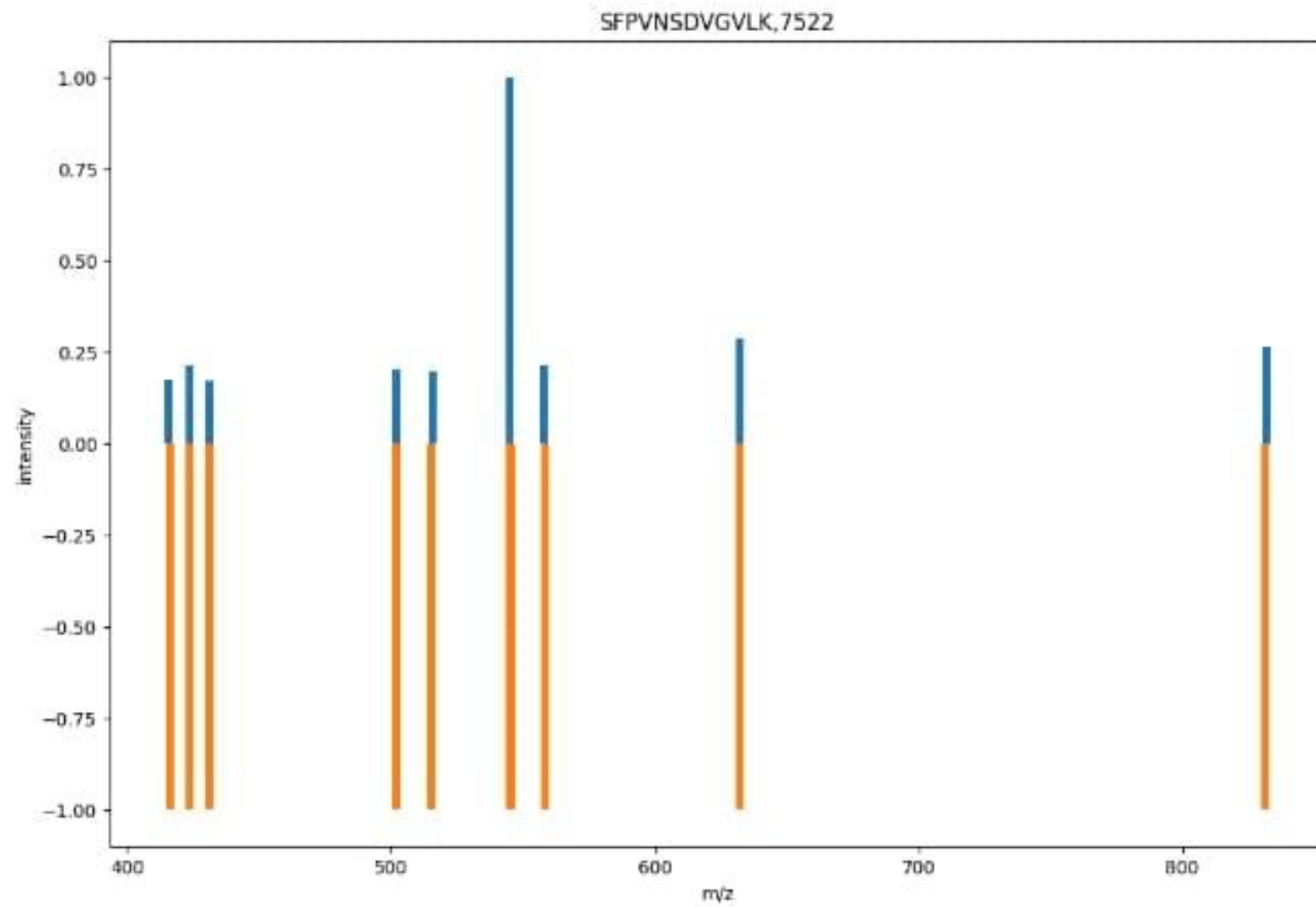


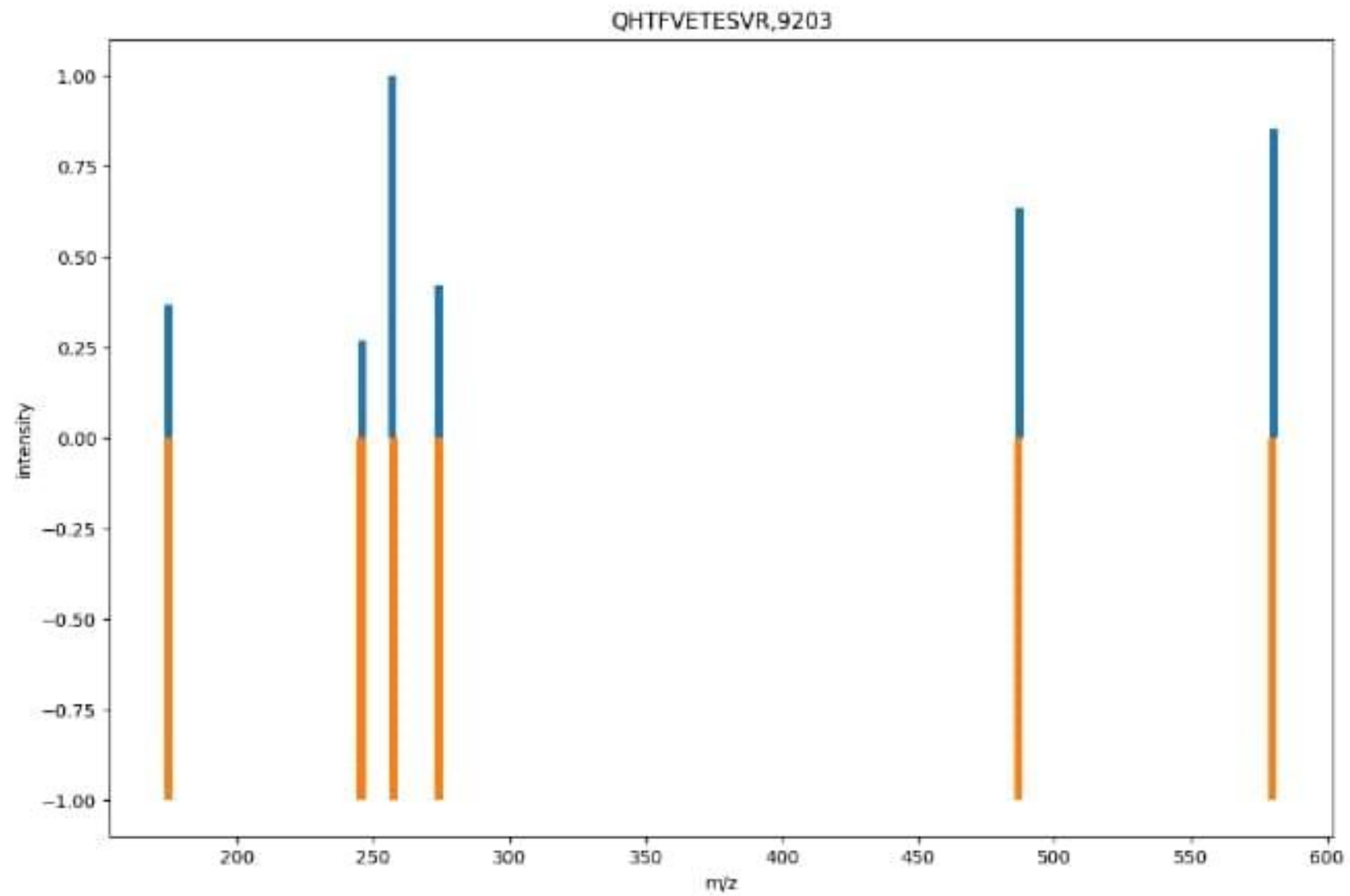
```
75     p.setValue("is_relative_tolerance", "false")
76     spa.setParameters(p)
77     # align both spectra
78     spa.getSpectrumAlignment(alignment, theo_spec, spectrum)
79     # Print matching ions and mz from theoretical spectrum
80     print("Number of matched peaks: " + str(len(alignment)))
81     print("ion\ttheo. m/z\tobserved m/z")
82     for theo_idx, obs_idx in alignment:
83         ion_name = theo_spec.getStringDataArrays()[0][theo_idx].decode()
84         ion_charge = theo_spec.getIntegerDataArrays()[0][theo_idx]
85         print(ion_name + "\t" + str(ion_charge) + "\t"
86               + str(theo_spec[theo_idx].getMZ())
87               + "\t" + str(spectrum[obs_idx].getMZ()))
88
89     theo_mz, theo_int, obs_mz, obs_int = [], [], [], []
90     for theo_idx, obs_idx in alignment:
91         theo_mz.append(theo_spec[theo_idx].getMZ())
92         theo_int.append(theo_spec[theo_idx].getIntensity())
93         obs_mz.append(spectrum[obs_idx].getMZ())
94         obs_int.append(spectrum[obs_idx].getIntensity())
95     title = f'{name},{peptide_id.getMetaValue("scan_index")}'
96     flag=False
97     for intensity in obs_int:
98         if(intensity>0):
99             flag=True
100             break
101     if(flag==True):
102         mirror_plot(obs_mz, obs_int, theo_mz, theo_int, title)
103     else:
104         flag=False
```

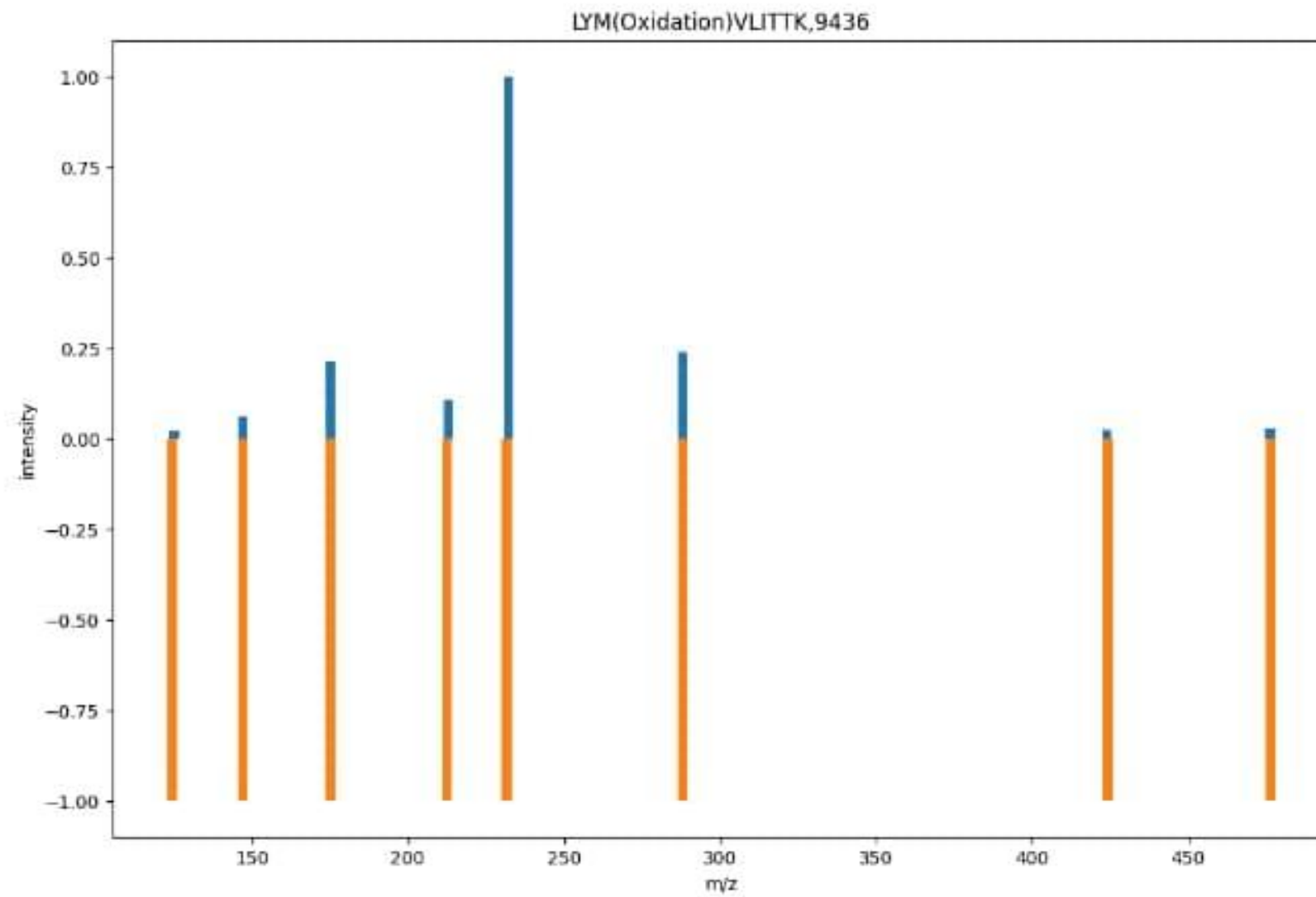
File :061818-COPD-152-02.mzML

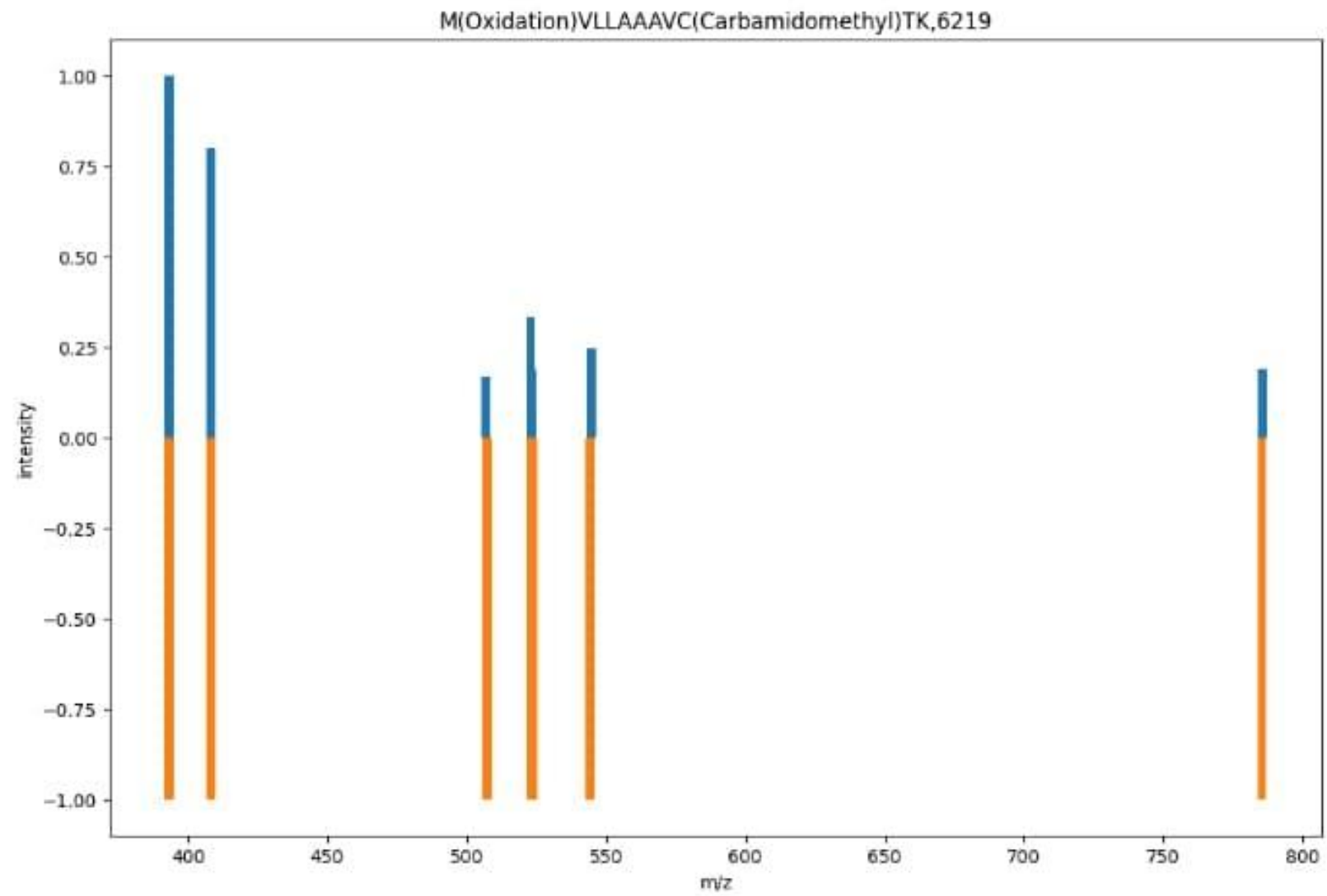




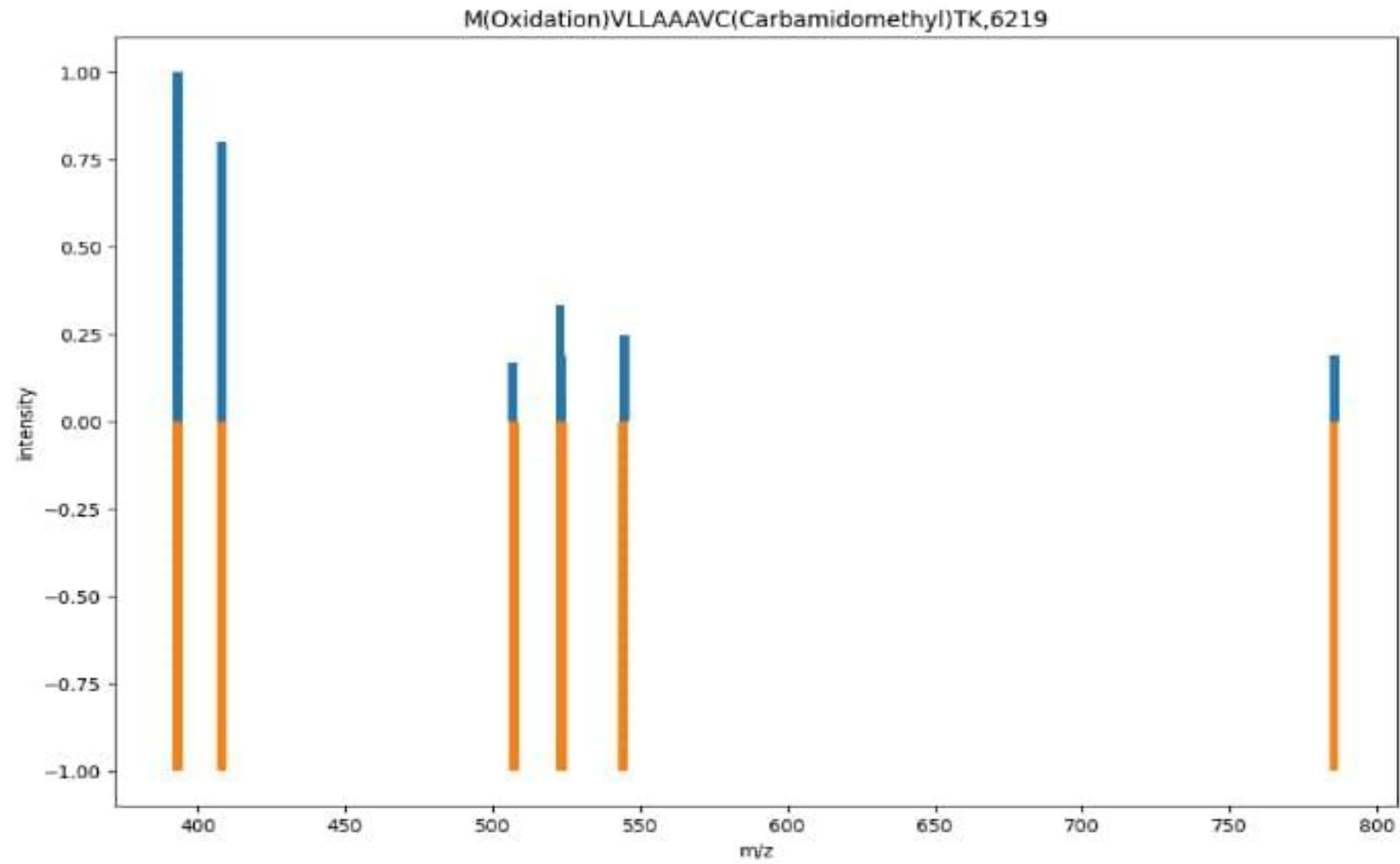


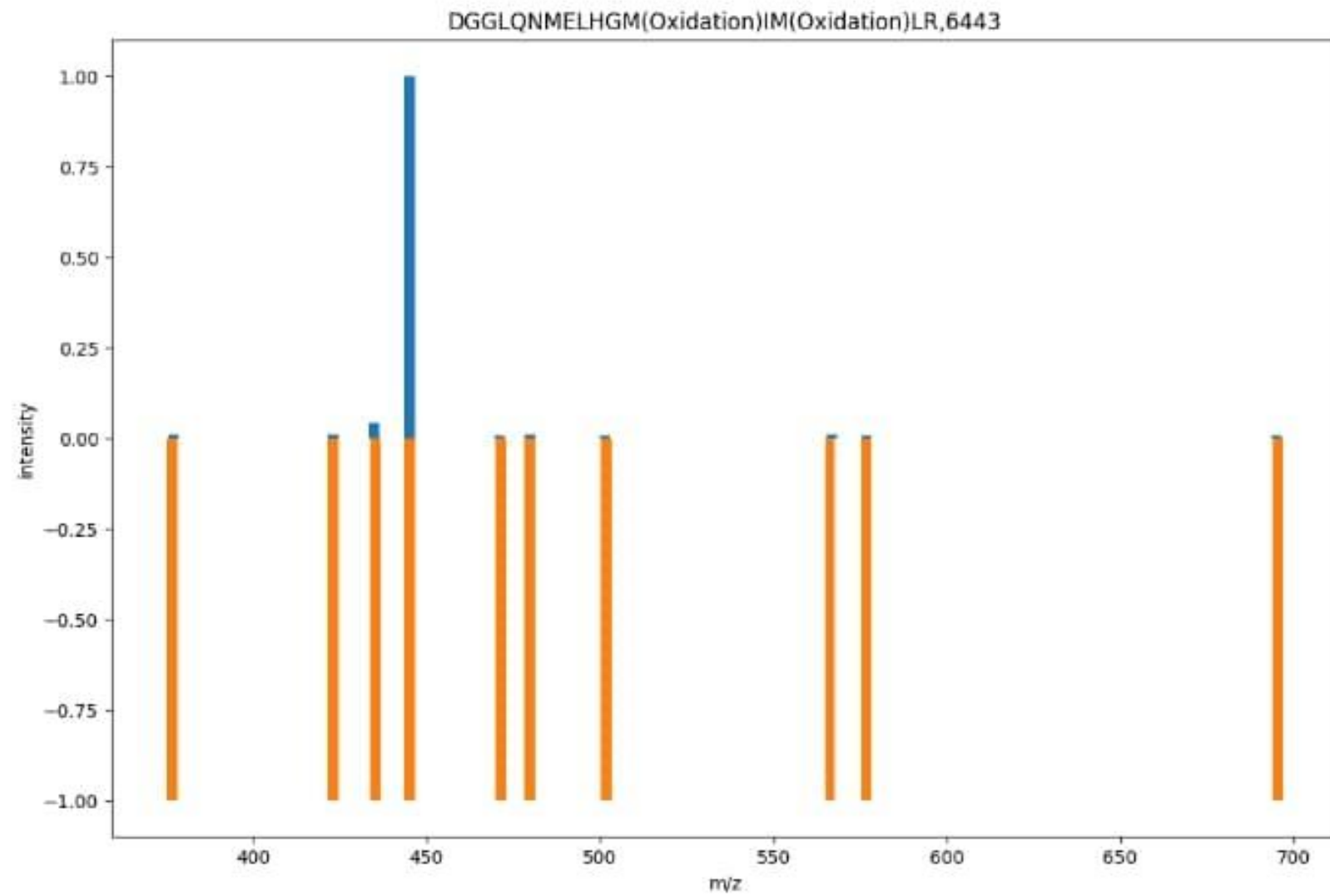


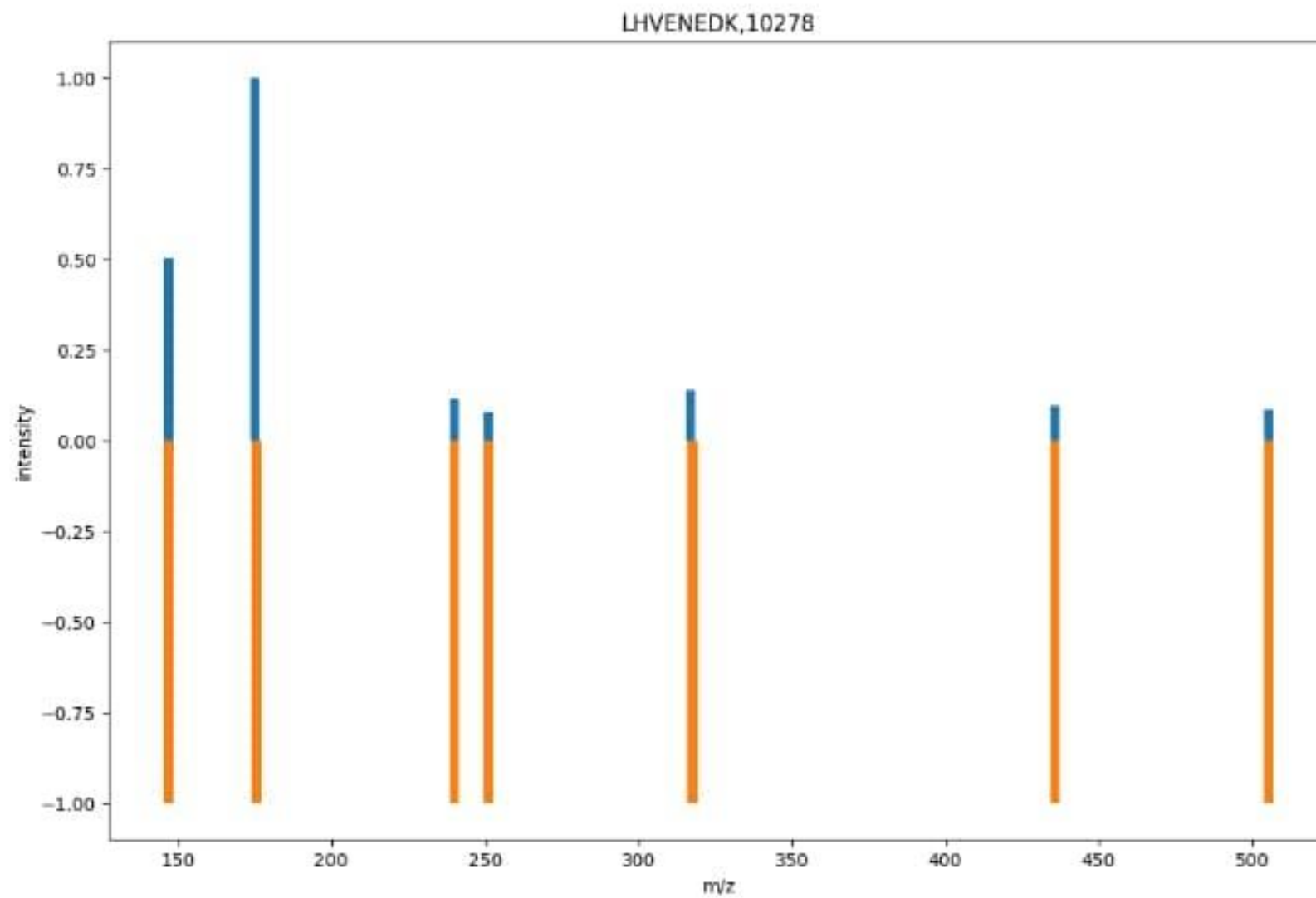


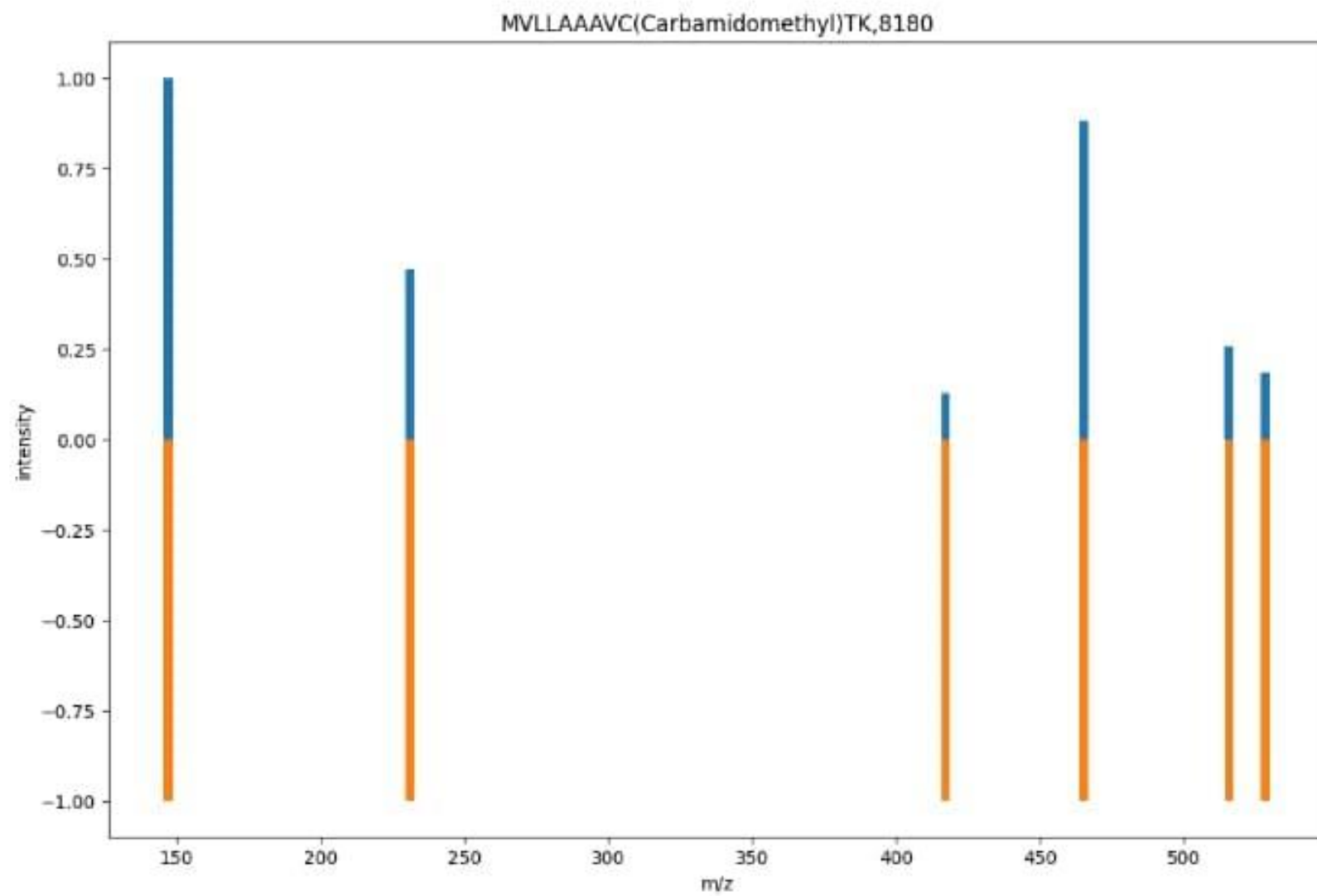


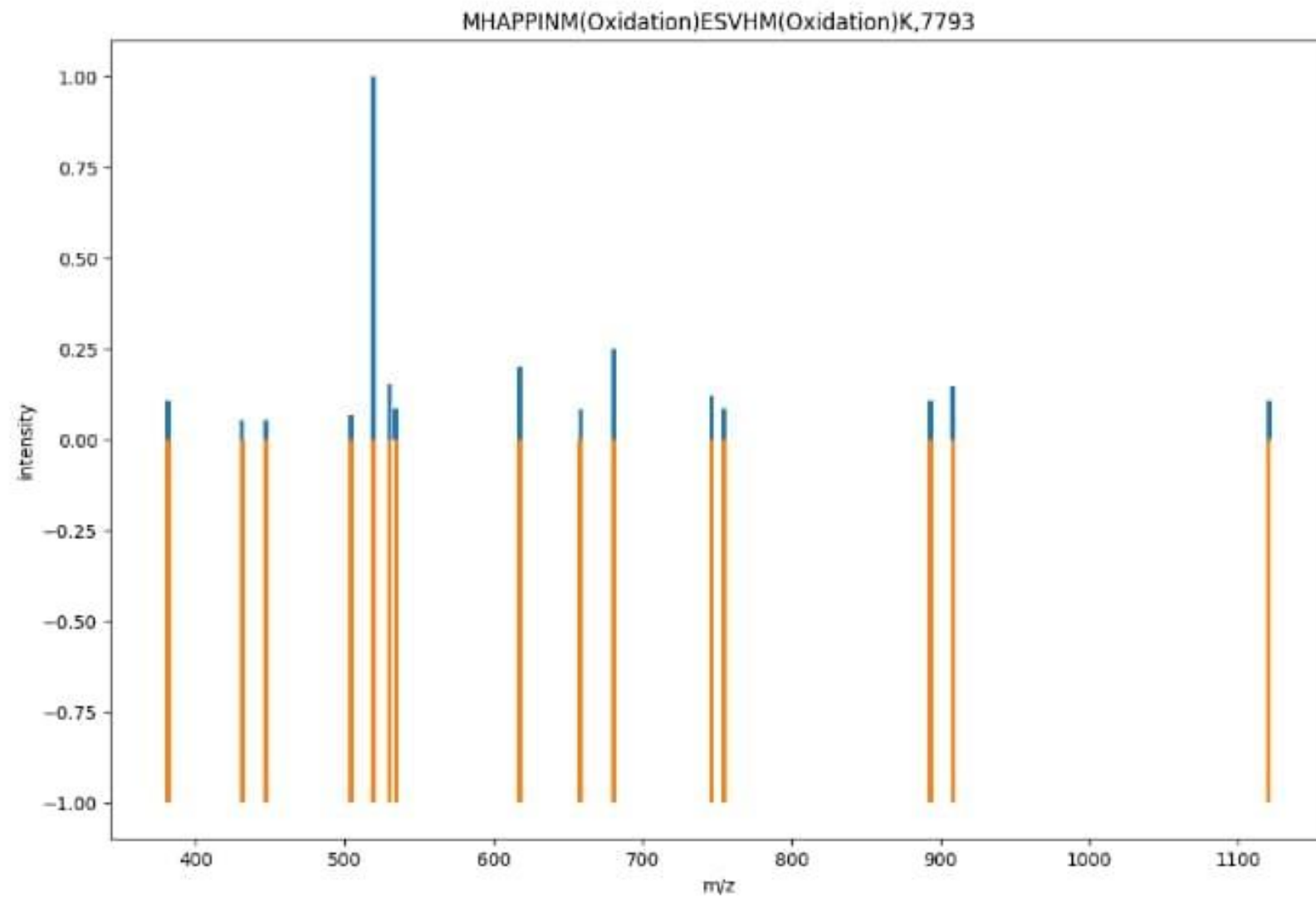
File : 061818-COPD-152-01.mzML



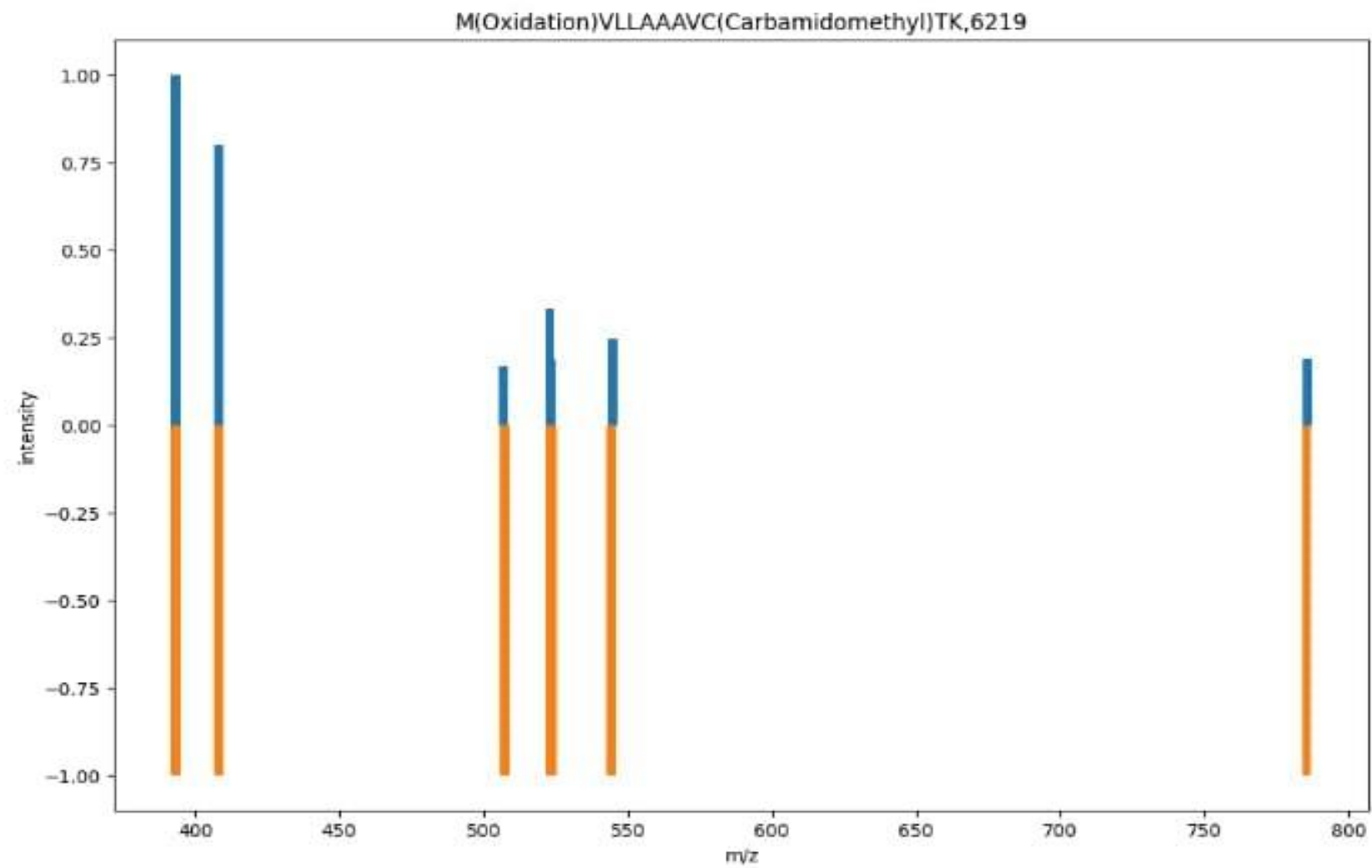


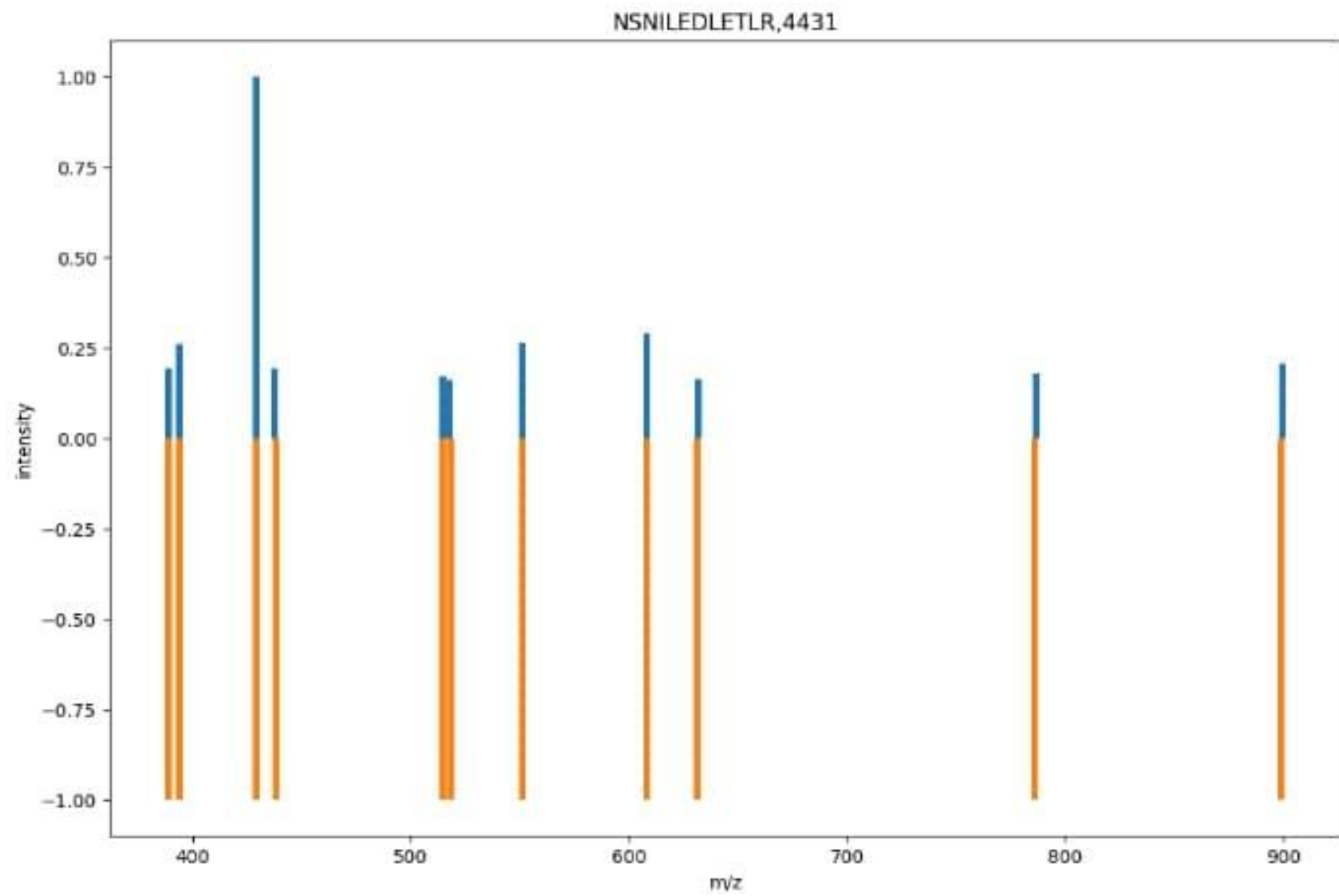


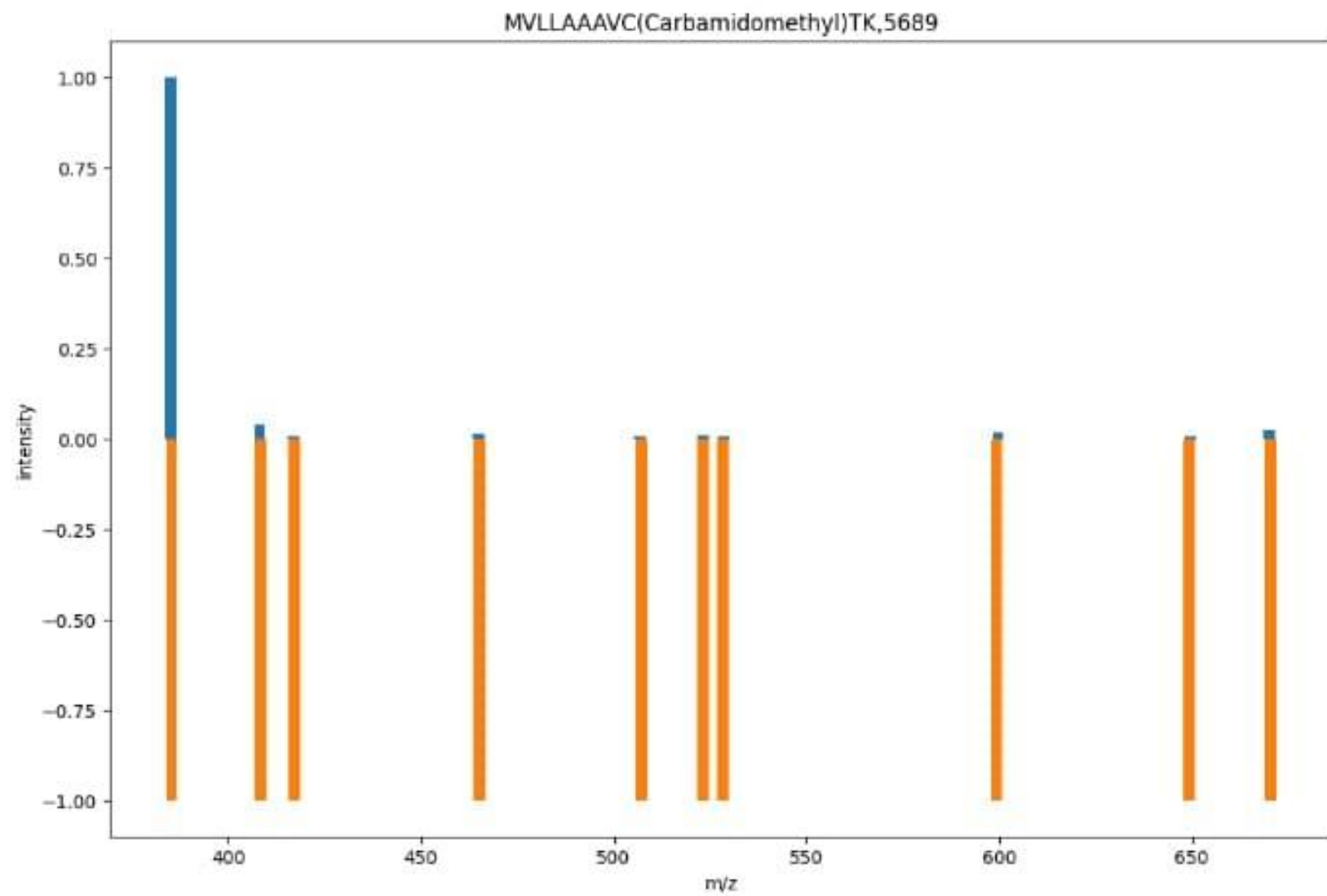


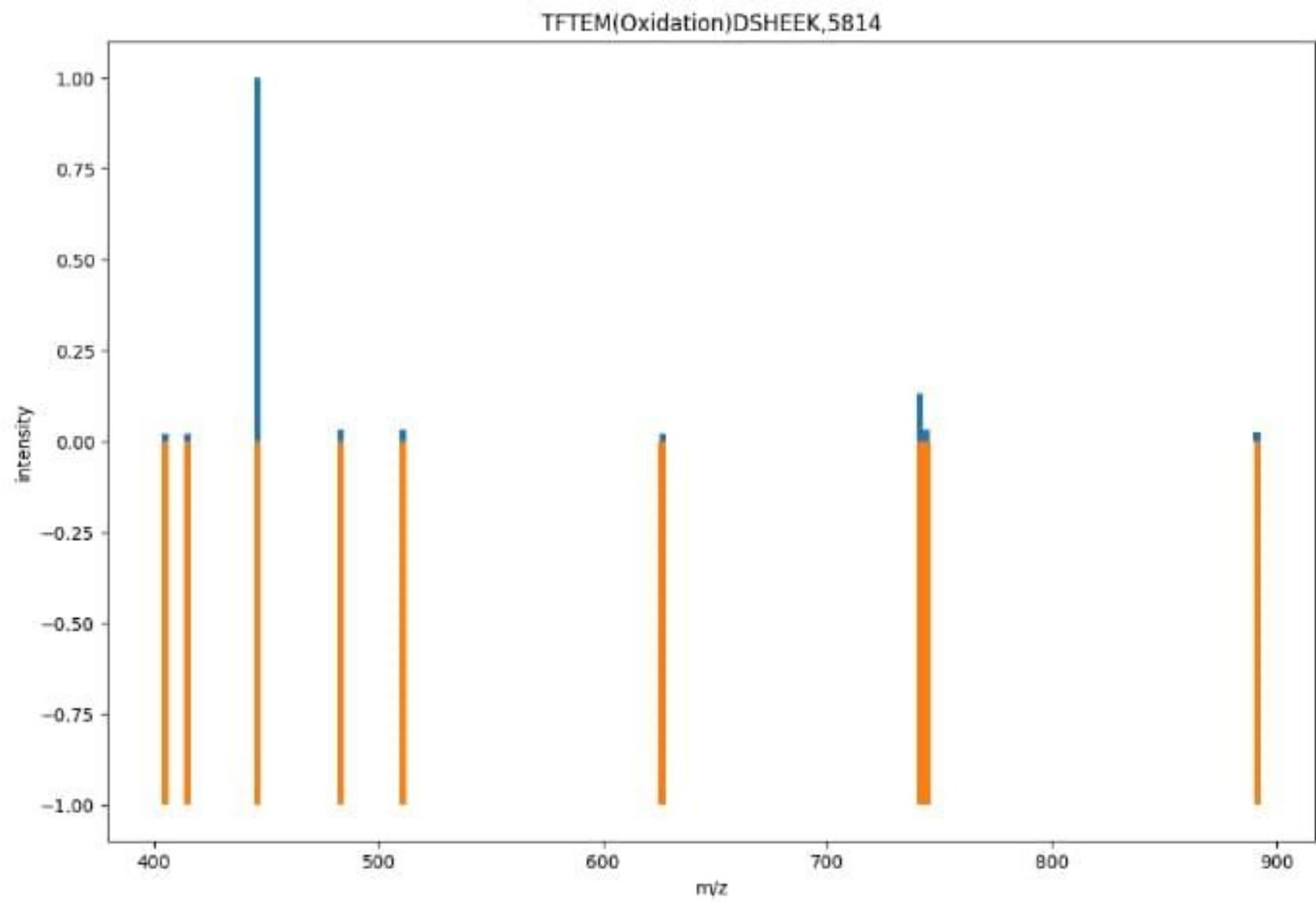


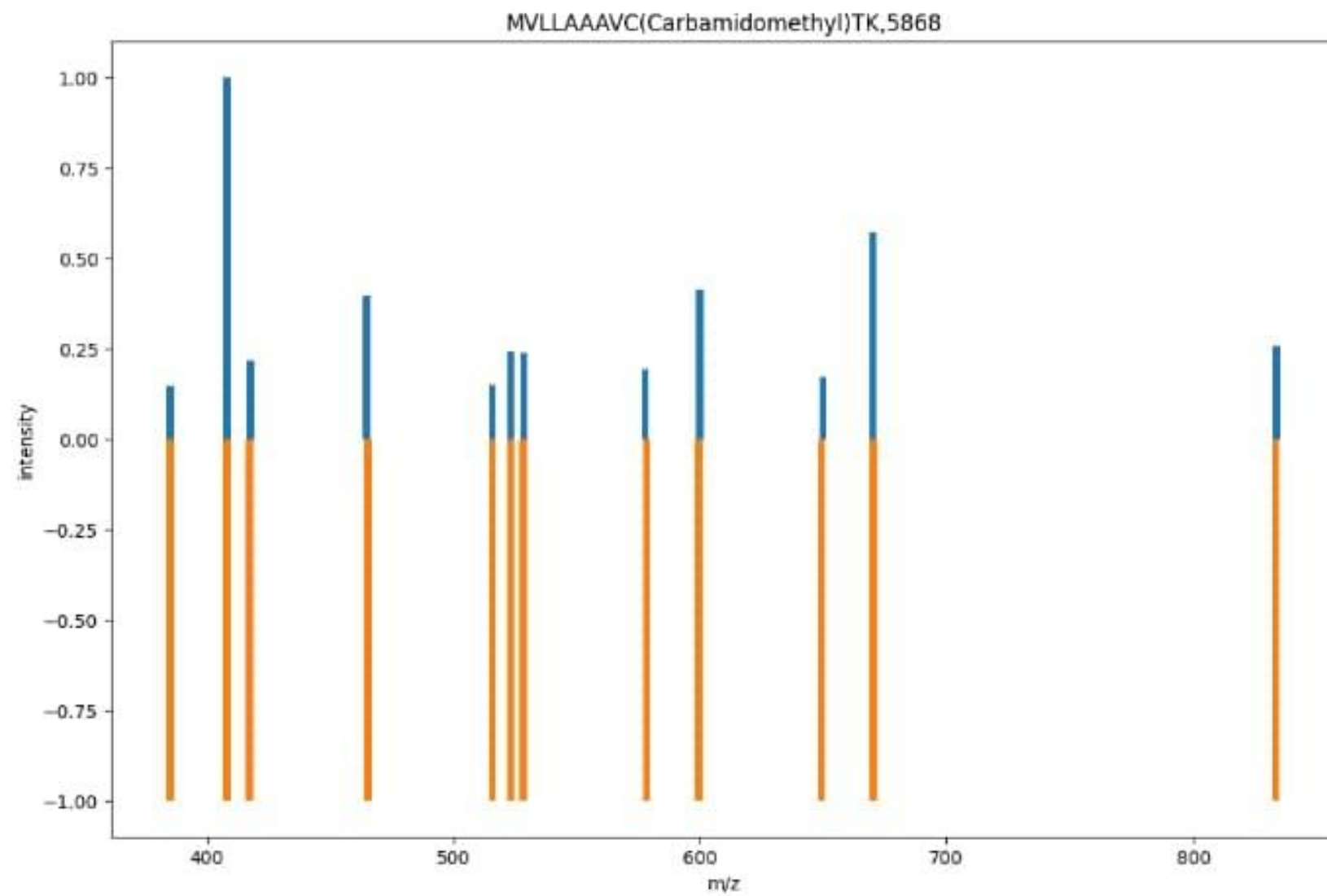
File:061818-COPD-151-04.mzML



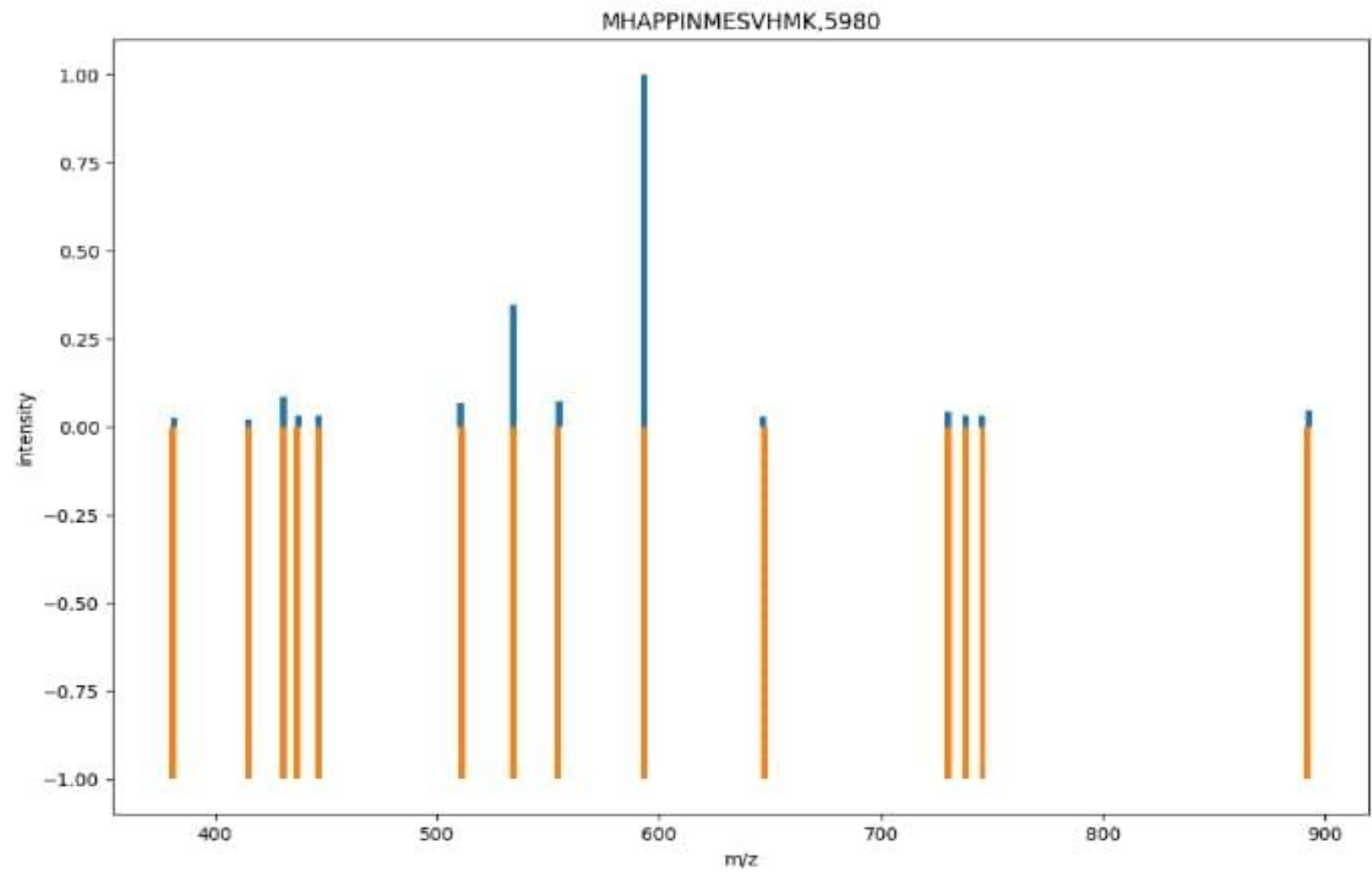


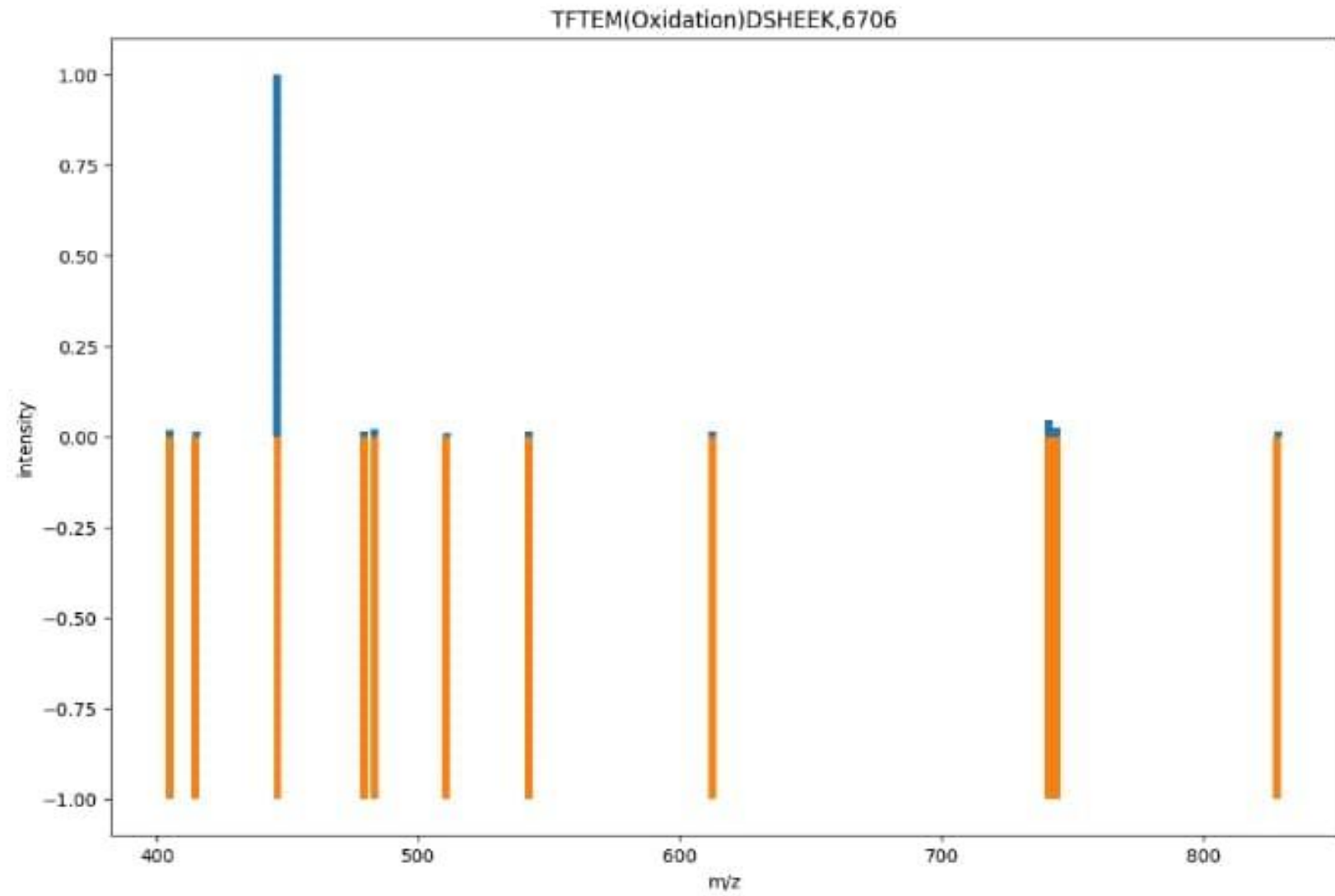


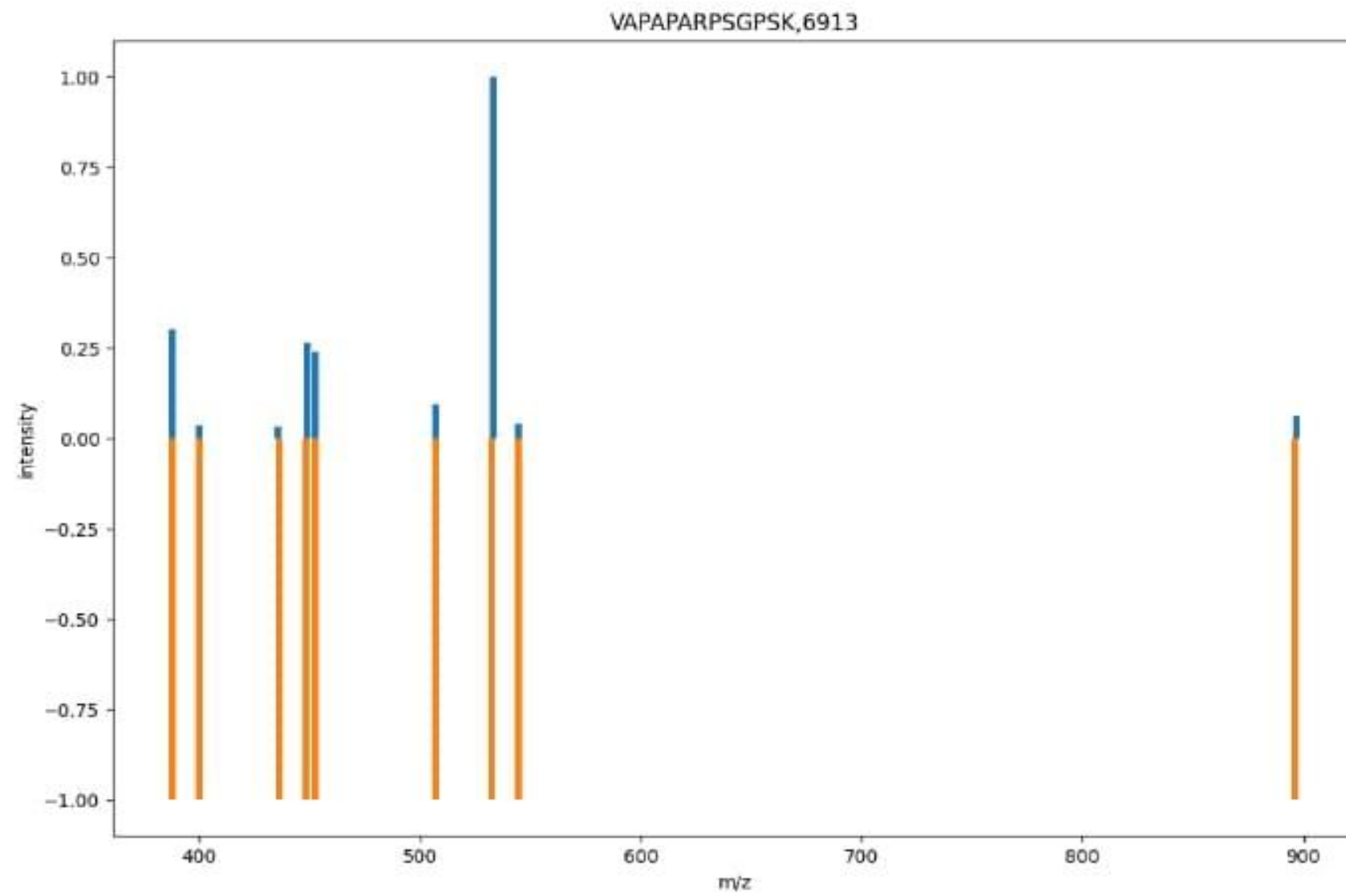




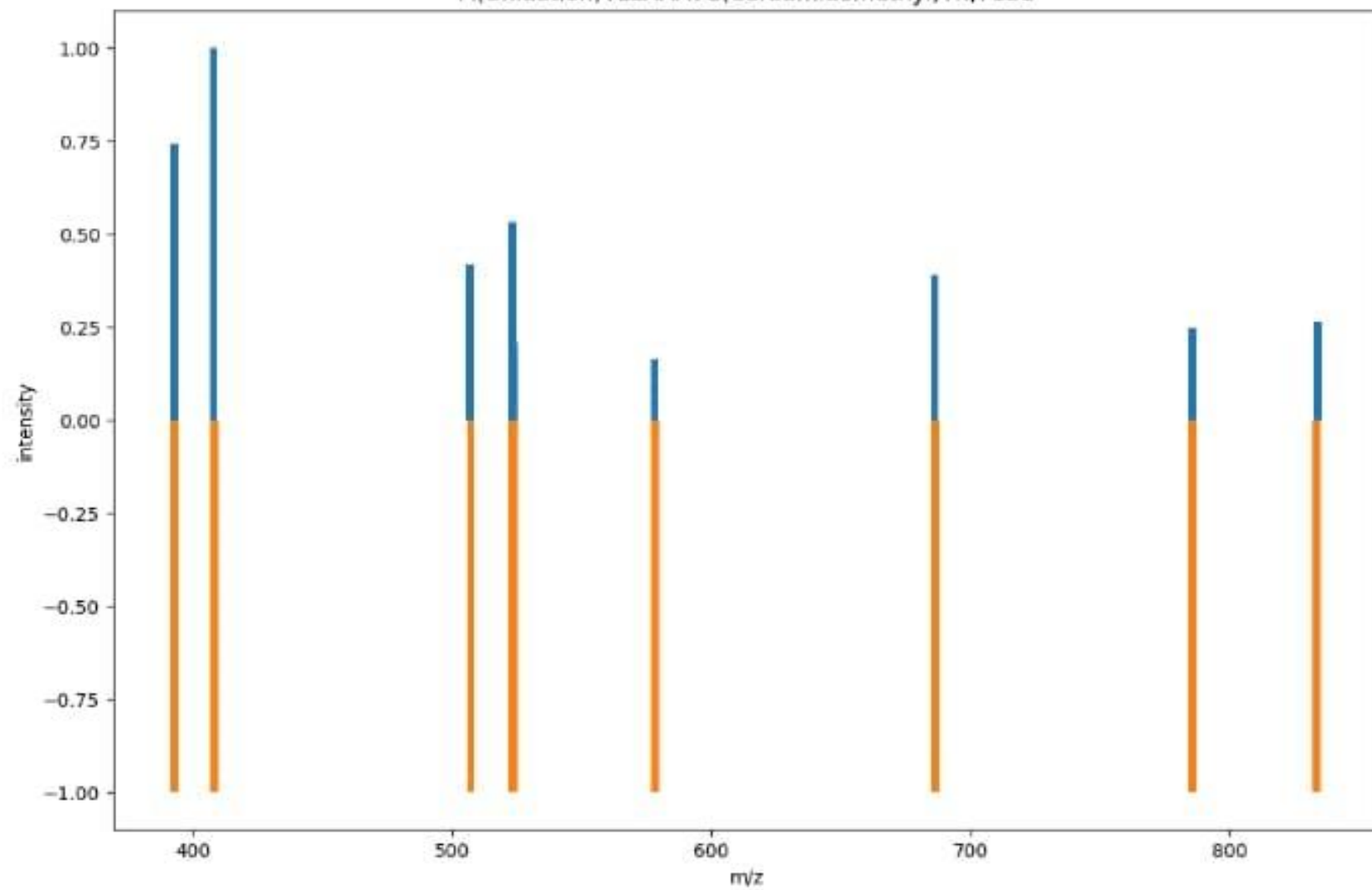
File: 061818-COPD-150-05.mzML

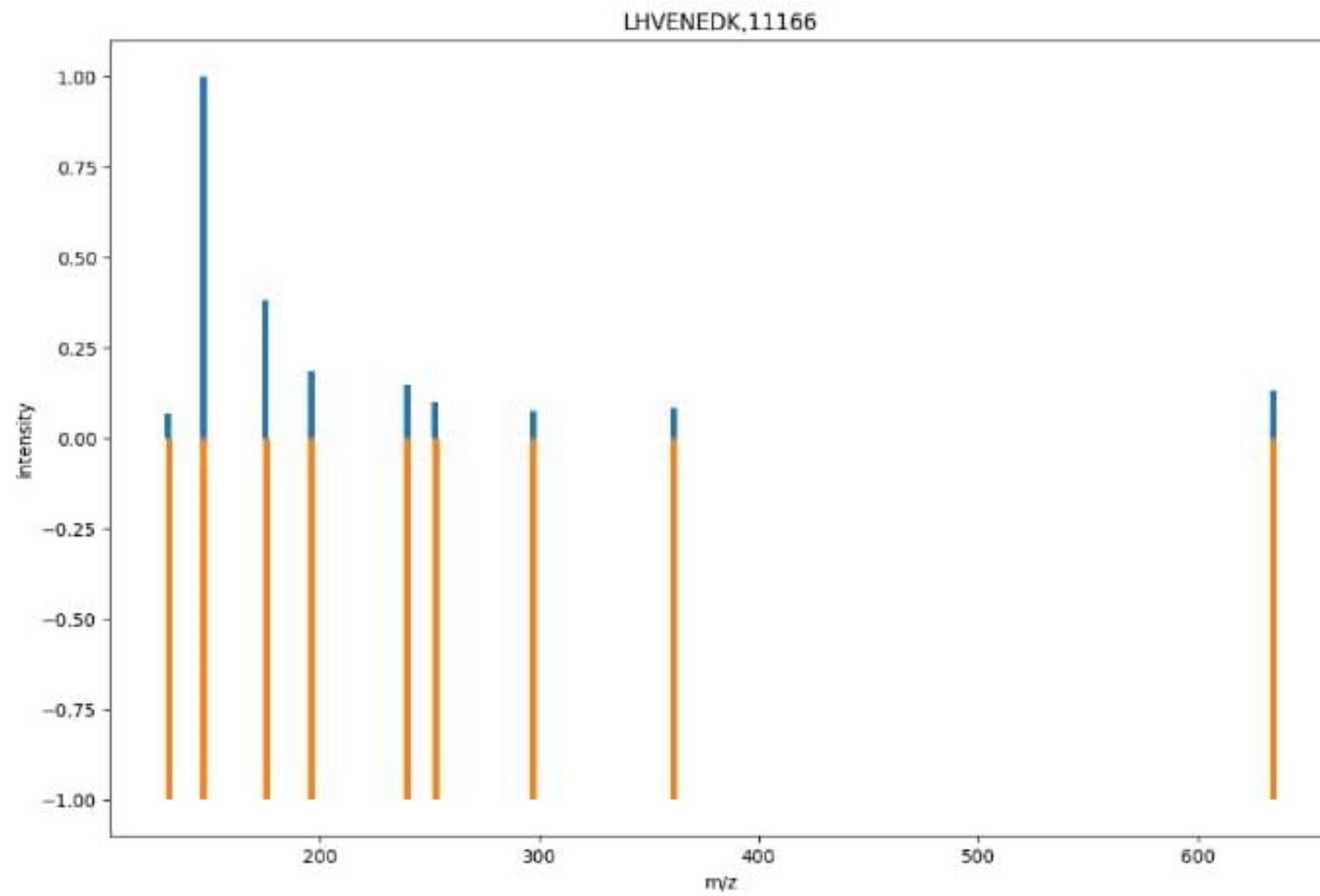




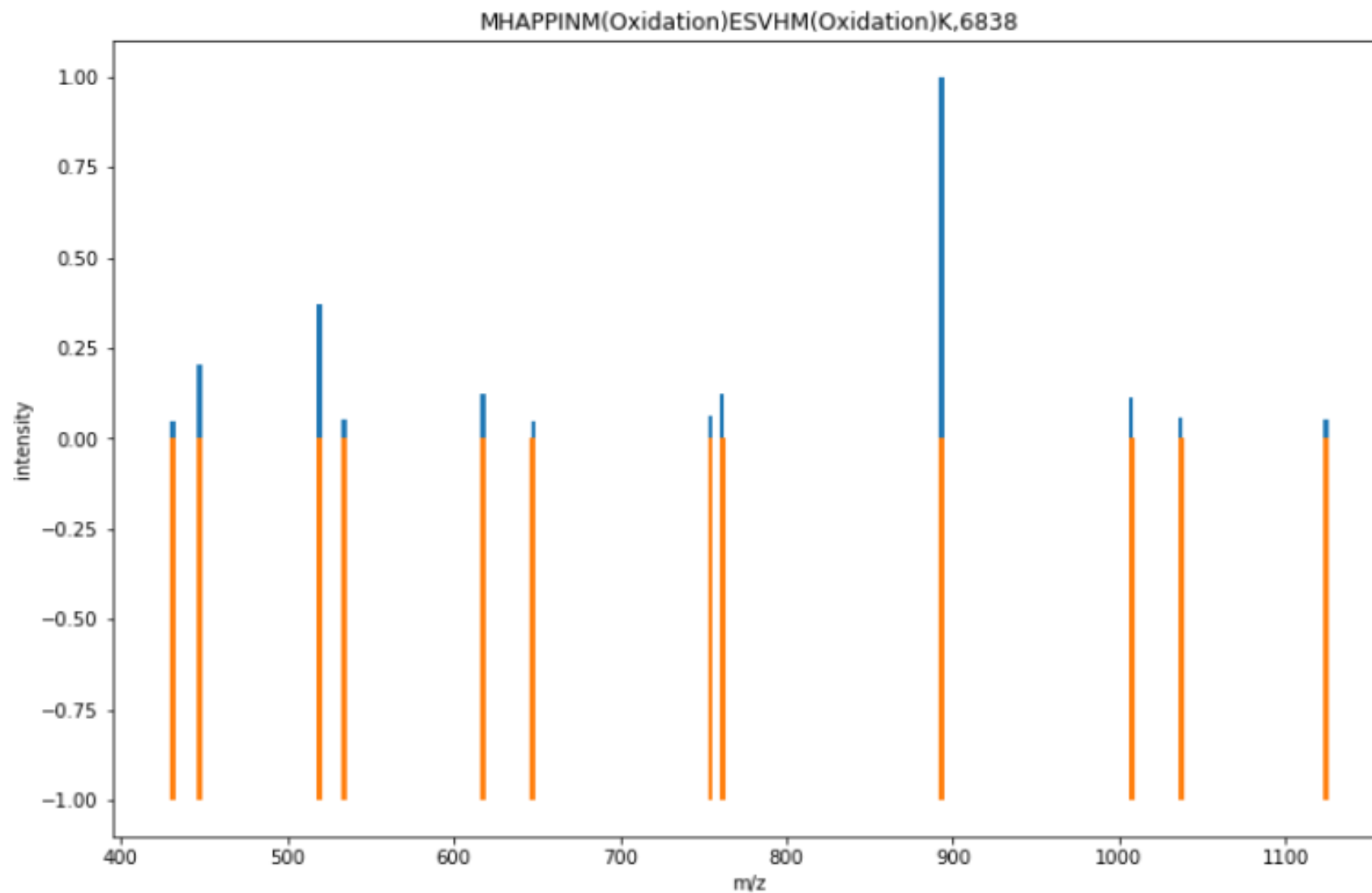


M(Oxidation)VLLAAAVC(Carbamidomethyl)TK,7121

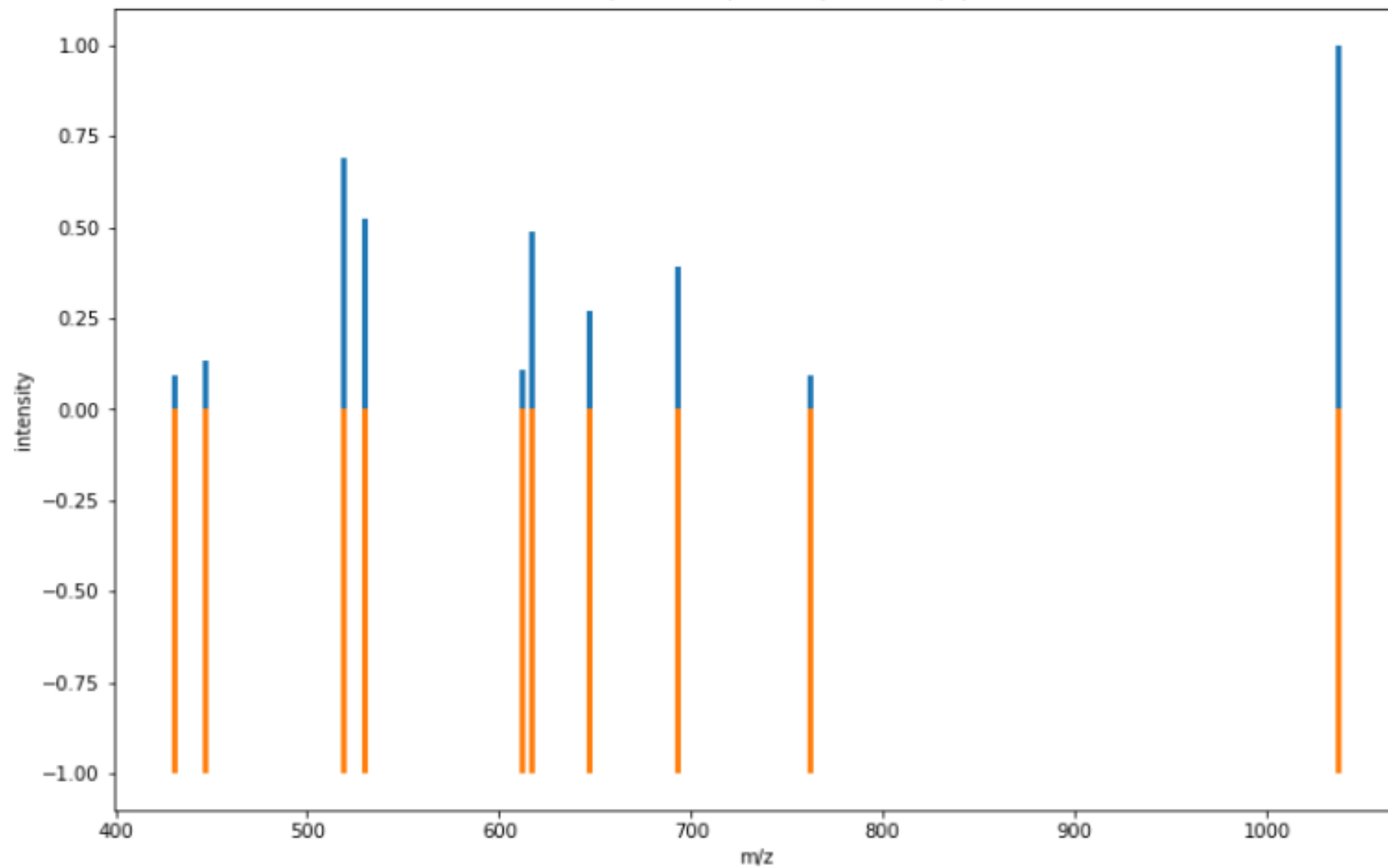




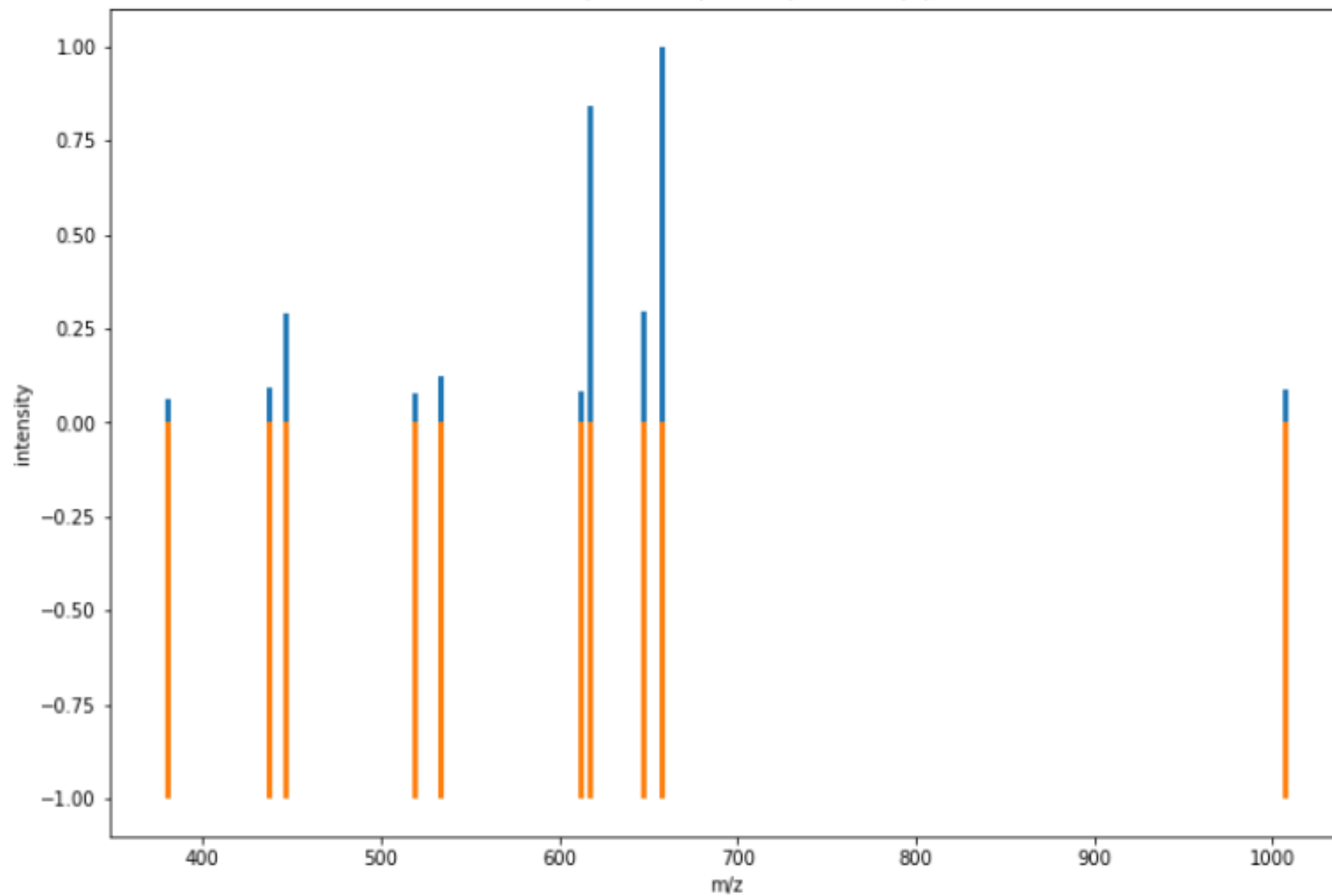
File:061818-COPD-147-02.mzML



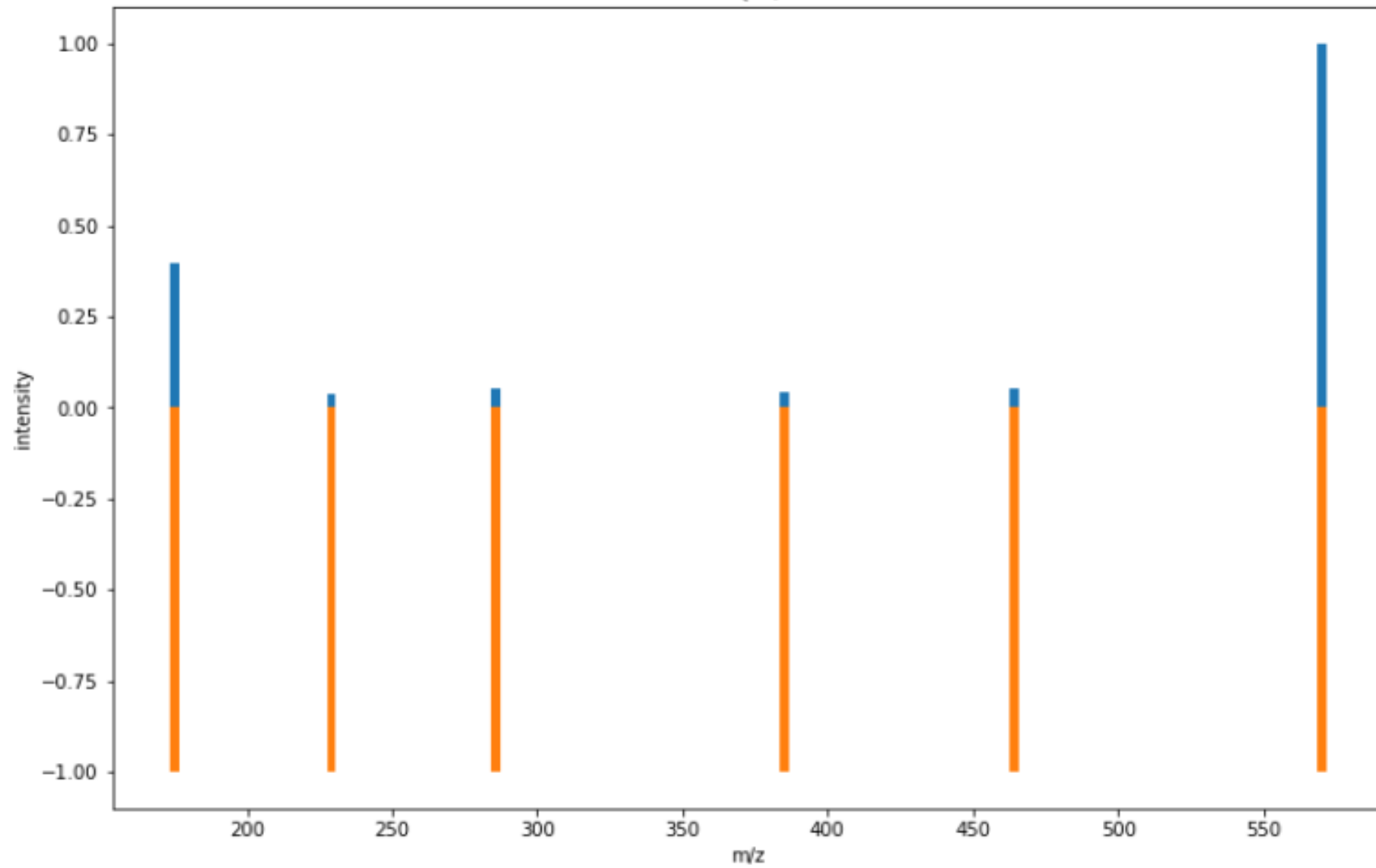
MHAPPINM(Oxidation)ESVHM(Oxidation)K,6939



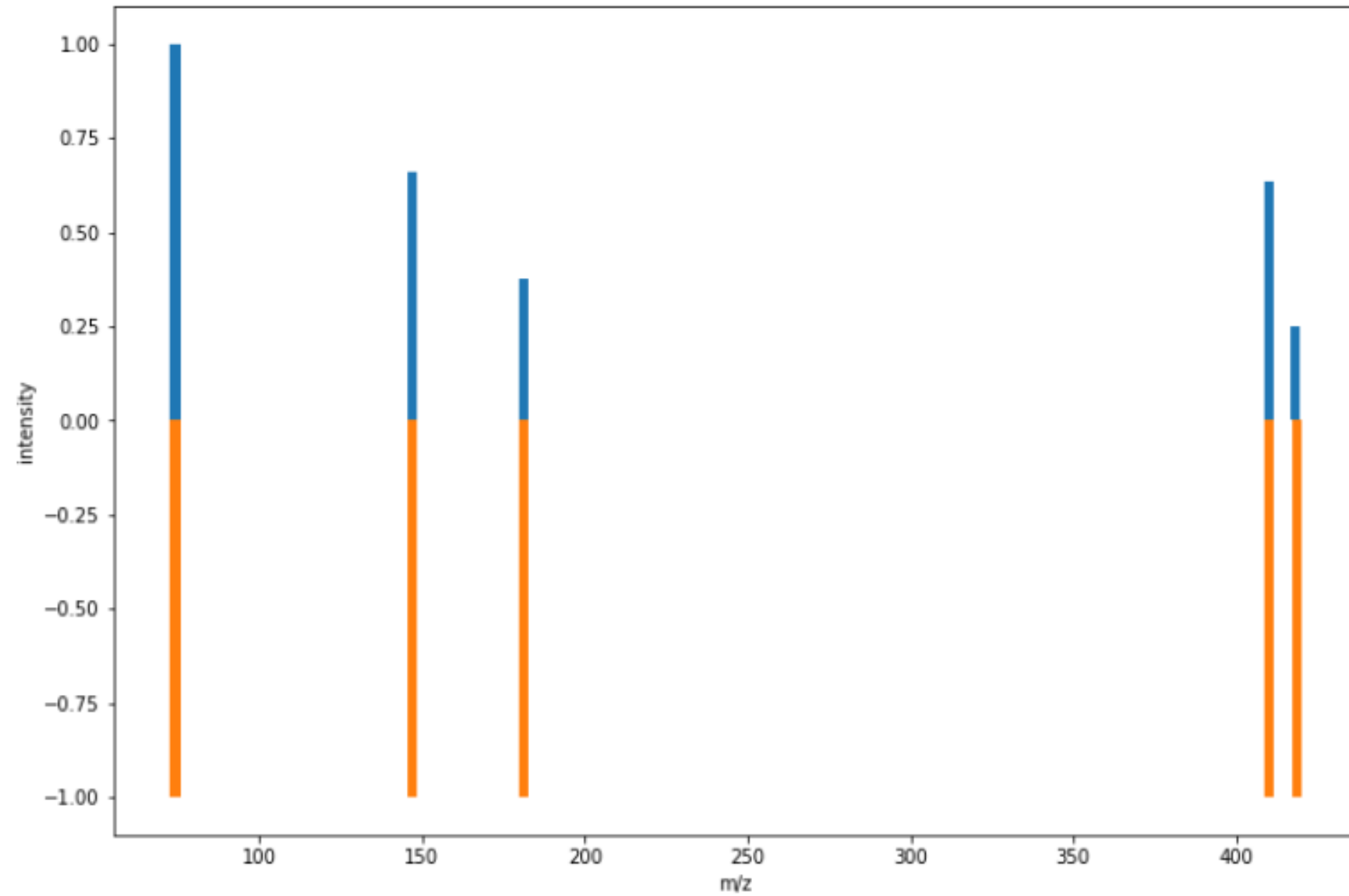
MHAPPINM(Oxidation)ESVHM(Oxidation)K,6997



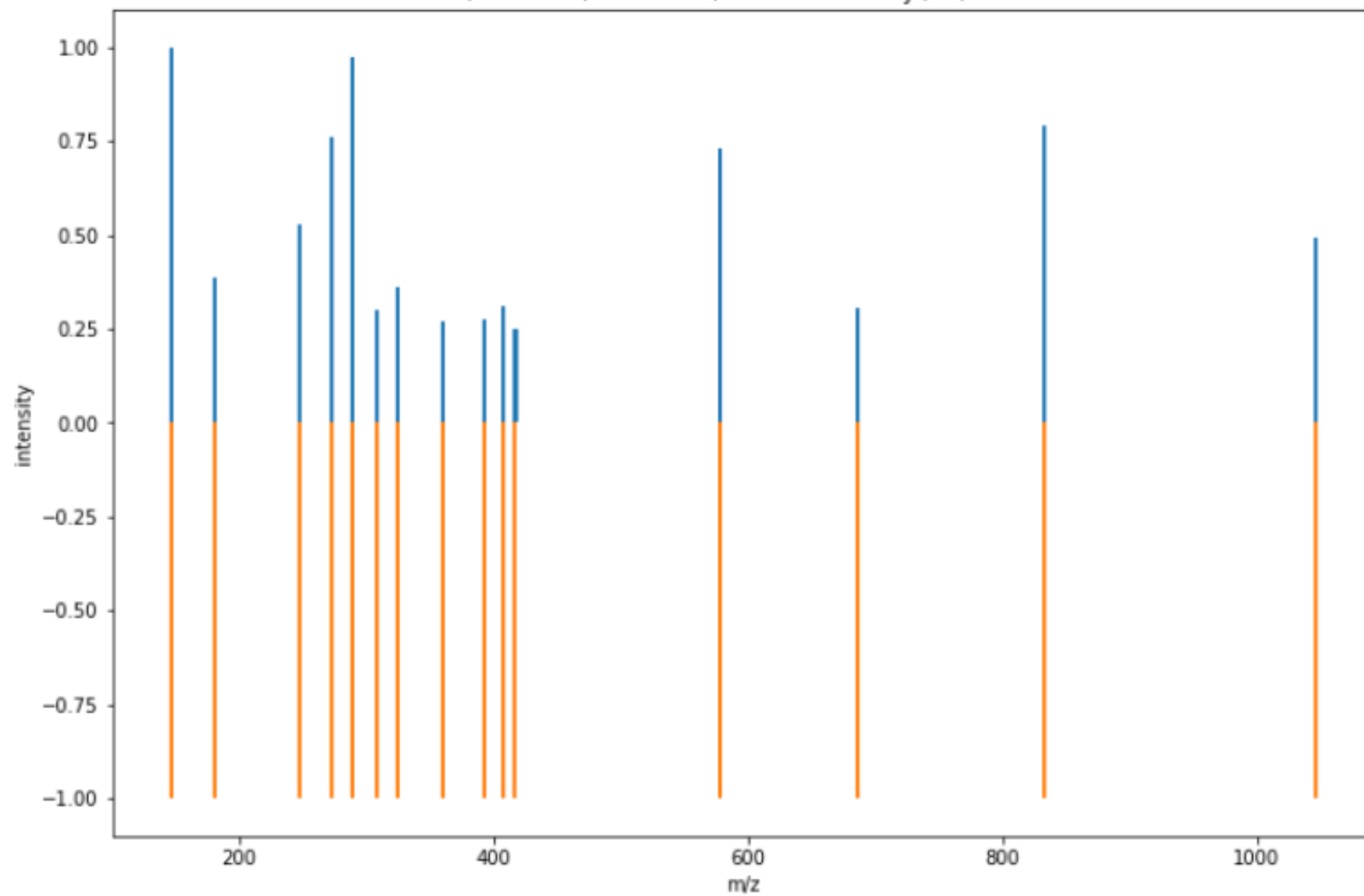
ENVNLAQIR,14772



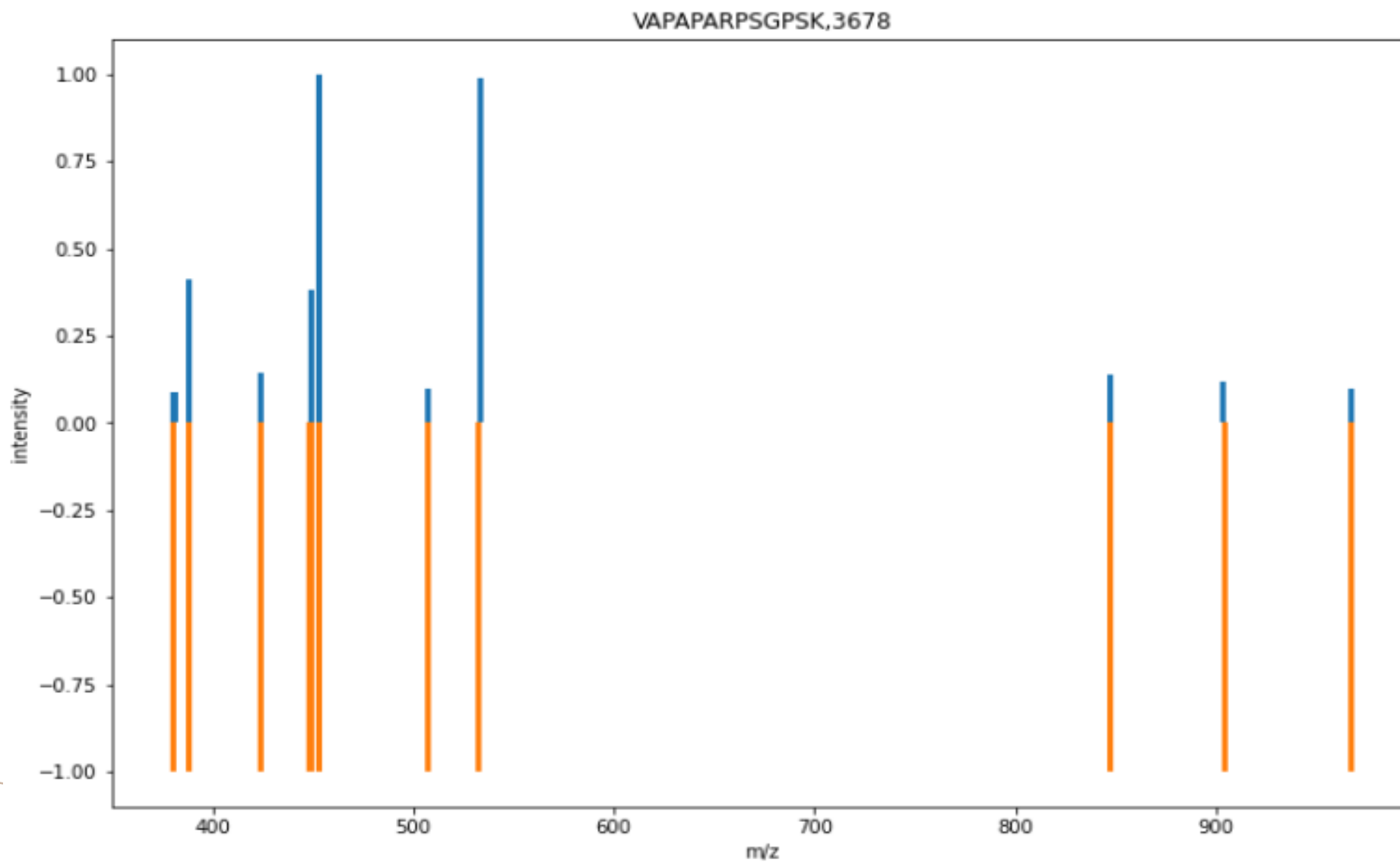
EVDNFVDK,16271



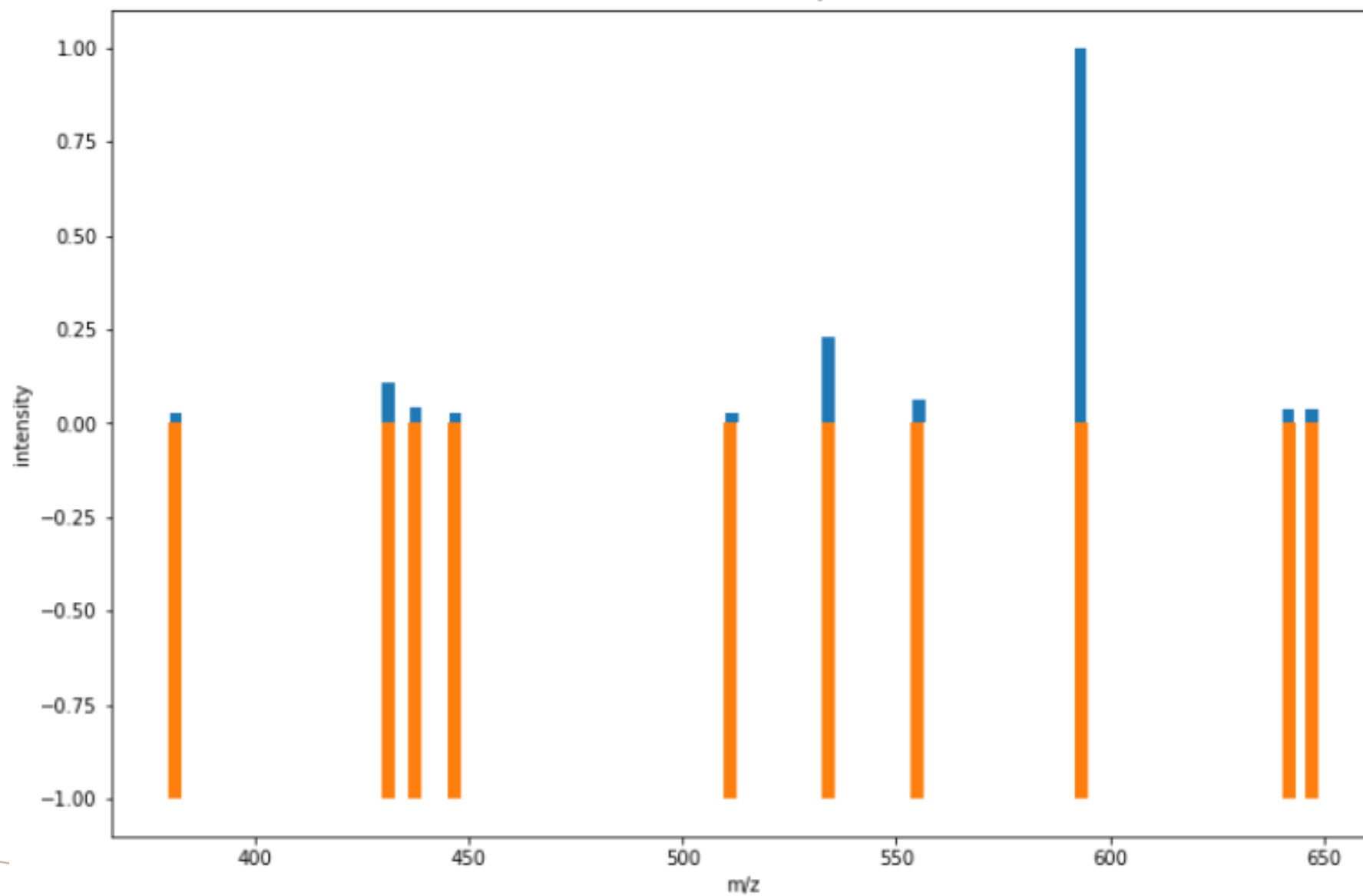
M(Oxidation)VLLAAVC(Carbamidomethyl)TK,16621



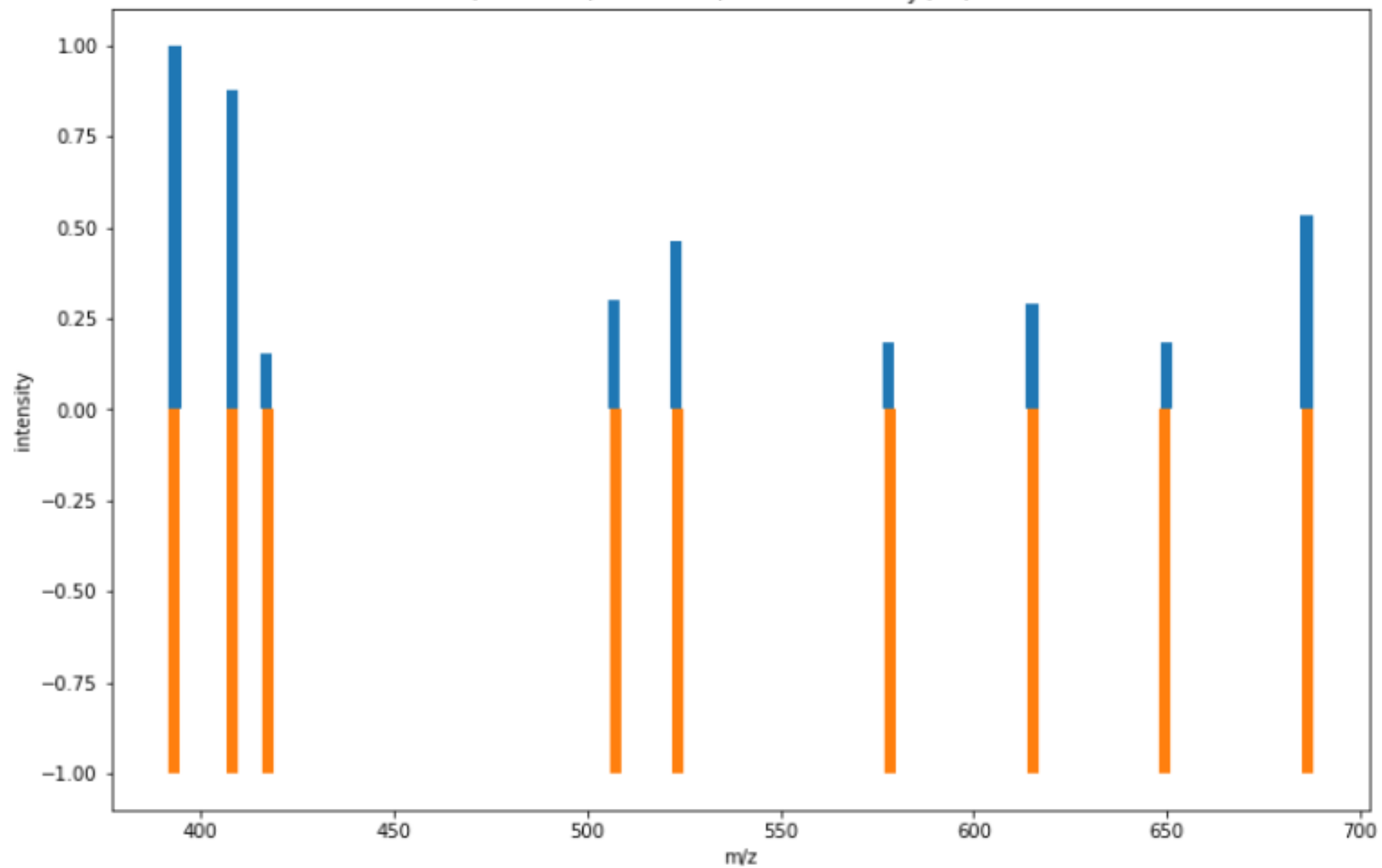
File: a061818-COPD-149-04.mzML



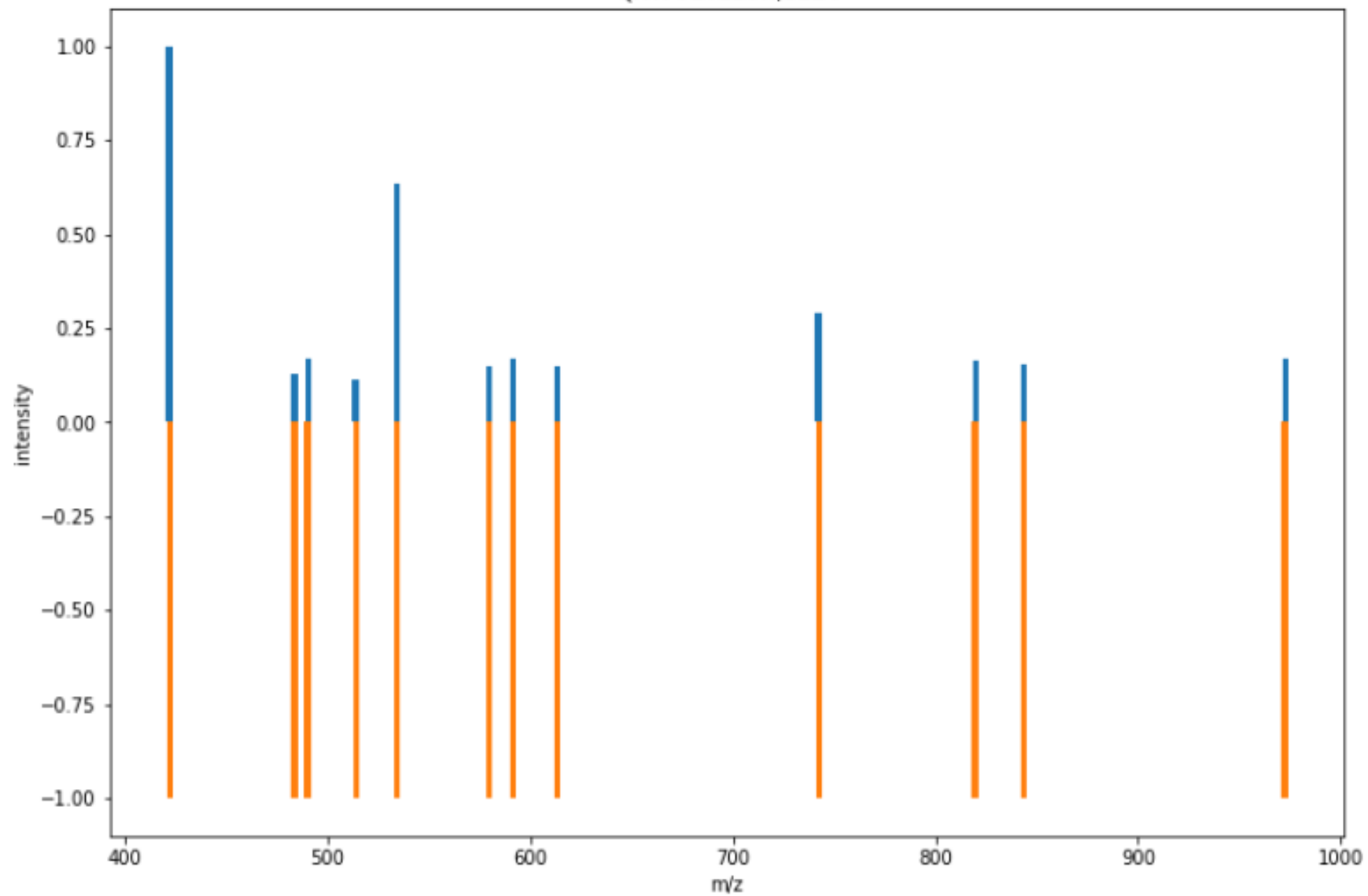
MHAPPINMESVHMK,4249



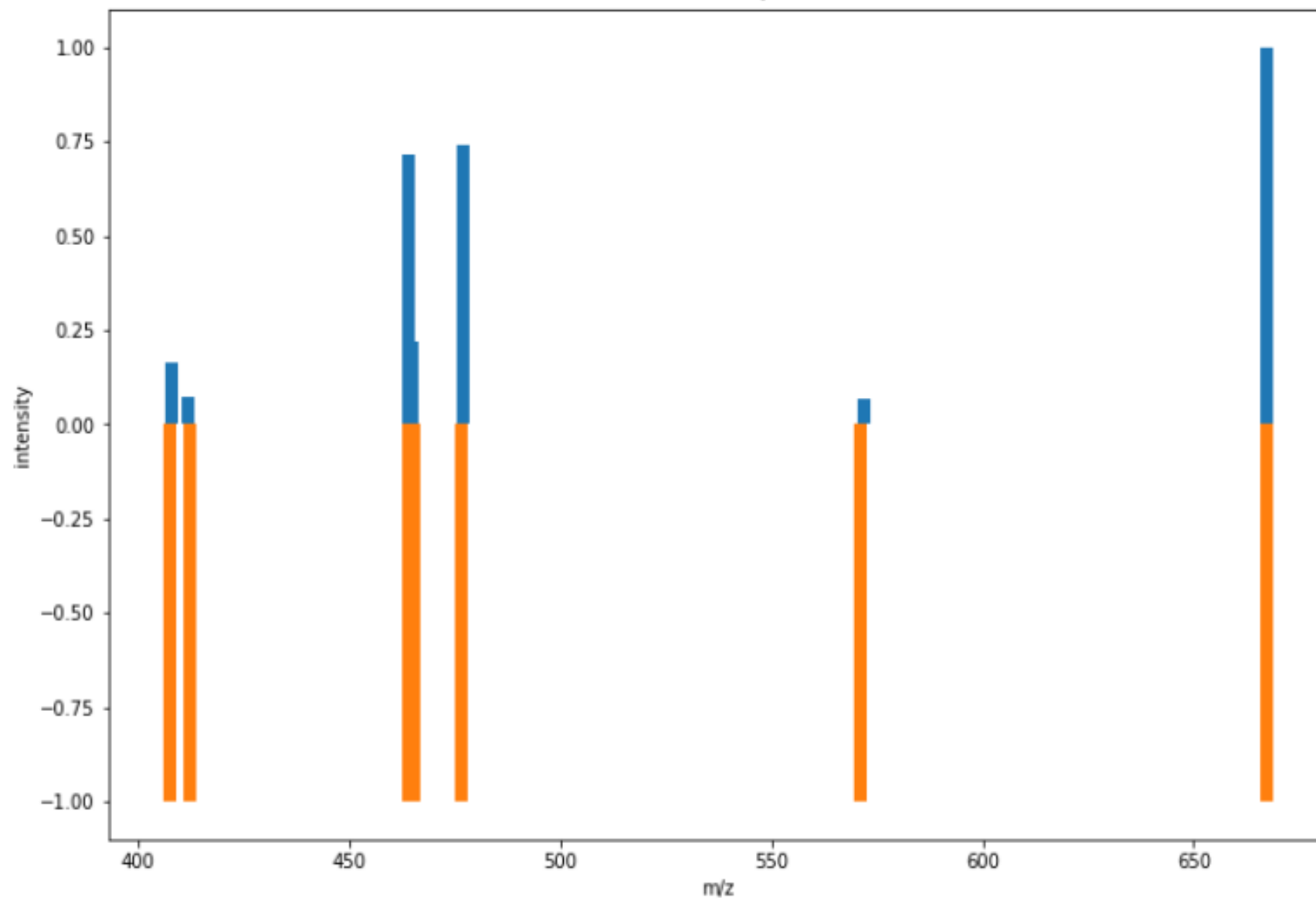
M(Oxidation)VLLAAVC(Carbamidomethyl)TK,5087



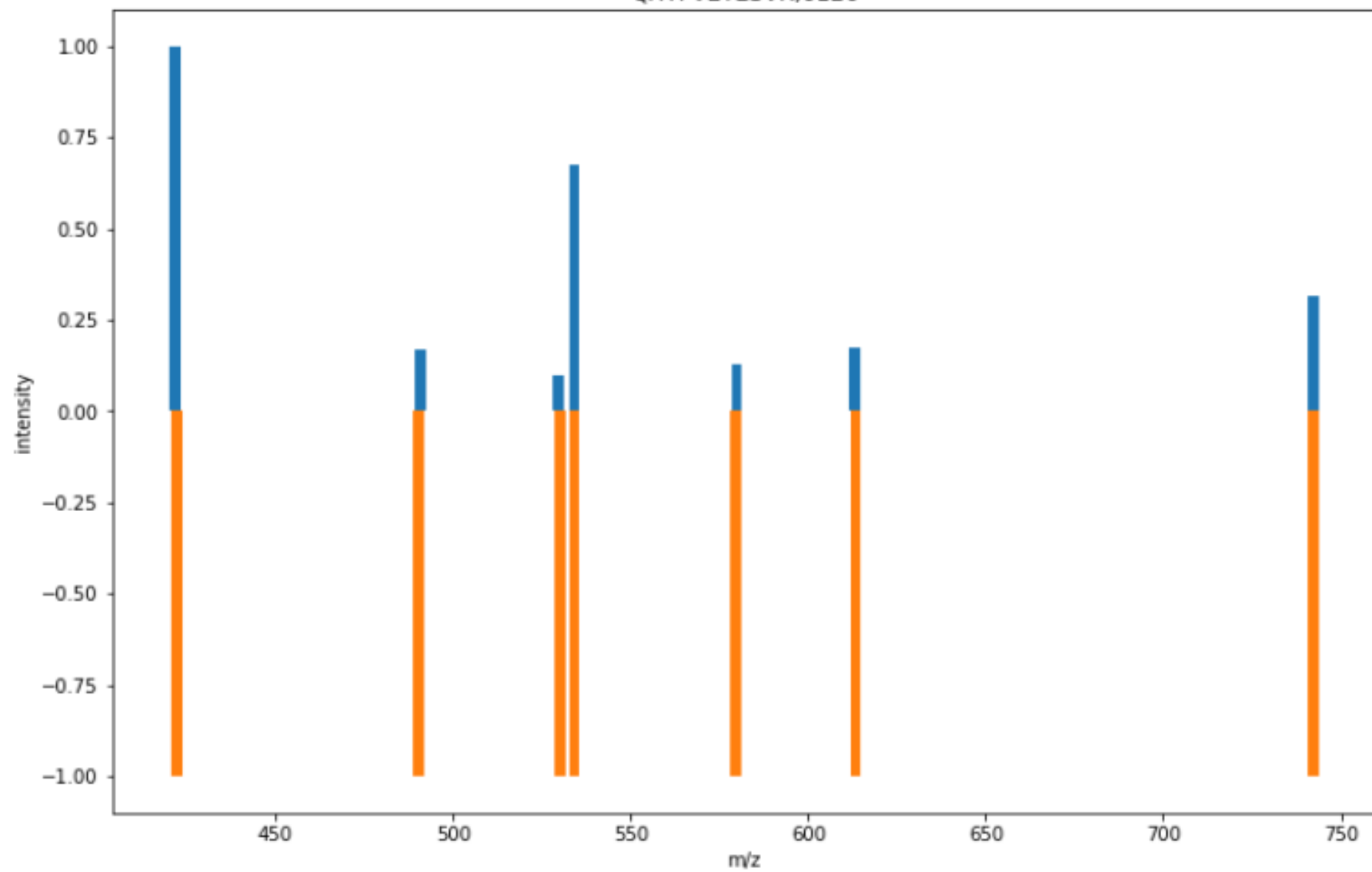
QHTFVETESVR,6124



FSTETTFLVDK,4530



QHTFVETESVR,6126



In [6]:

```
1 searchfile="C:/Users/TC/Desktop/061818-COPD-149-04.mzML"
2 searchdb = "P48444.fasta"
3
4 # generate a protein database with additional decoy sequences
5 targets = list()
6 decoys = list()
7 FASTAFile().load(searchdb, targets) # read FASTA file into a list of FASTAEntries
8 decoy_generator = DecoyGenerator()
9 for entry in targets:
10     rev_entry = FASTAEntry(entry) # copy entry
11     rev_entry.identifier = "DECOY_" + rev_entry.identifier # mark as decoy
12     aas = AASequence().fromString(rev_entry.sequence) # convert string into amino acid sequence
13     rev_entry.sequence = decoy_generator.reverseProtein(aas).toString() # reverse
14     decoys.append(rev_entry)
15
16 target_decoy_database = "search_td.fasta"
17 FASTAFile().store(target_decoy_database, targets + decoys) # store the database with appended decoy sequences
18
19 FASTAFile().store("decoy.fasta", decoys) # store the database with appended decoy sequences
```

In [7]:

```
1 # Run SimpleSearchAlgorithm, store protein and peptide ids
2 protein_ids = []
3 peptide_ids = []
4
5 # set some custom search parameters
6 simplesearch = SimpleSearchEngineAlgorithm()
7 params = simplesearch.getDefault()
8 score_annot = [b'fragment_mz_error_median_ppm', b'precursor_mz_error_ppm']
9 params.setValue(b'annotate:PSM', score_annot)
10 params.setValue(b'peptide:max_size', 30)
11 simplesearch.setParameters(params)
12
13 simplesearch.search(searchfile, target_decoy_database, protein_ids, peptide_ids)
```

```

In [8]: 1 sequence=list()
        2 source=list()
        3 score=list()
        4 for peptide_id in peptide_ids:
        5     for hit in peptide_id.getHits():
        6         sequence.append(hit.getSequence())
        7         source.append(hit.getMetaValue("target_decoy"))
        8         score.append(hit.getScore())
        9 data = {'Sequence':sequence,
       10         'Target_Decoys': source,
       11         'score':score
       12         }
       13
       14 df = pd.DataFrame(data)
       15 df

```

Out[8]:

	Sequence	Target_Decoys	score
0	VAPAPARPSGPSK	target	9.505832
1	VAPAPARPSGPSK	target	9.412774
2	MHAPPINMESVHMK	target	0.642283
3	MHAPPINMESVHMK	target	0.804148
4	VSETEVFTHQK	decoy	0.136526
...
199	MVLLAAAVC(Carbamidomethyl)TK	target	0.031847
200	GTNMLKPFAALLGEIR	decoy	0.061485
201	DGGLQNMELHGMIM(Oxidation)LR	target	0.186863
202	YGLAVIEDFAFILDFC(Carbamidomethyl)HESIENEELAR	decoy	0.234703
203	ENVNLAQIR	target	0.084336

204 rows × 3 columns



Run



Code



202 TGLAVIEDFARILDFQ(Carbamidomethyl)NESIENEELAR decoy 0.234703

203 ENVNLAQIR target 0.084336

204 rows × 3 columns

```
In [11]: 1 # Filter by 1% PSM FDR (q-value < 0.01)
          2 idfilter = IDFilter()
          3 idfilter.filterHitsByScore(peptide_ids, 0.01)
          4
          5 sequence=list()
          6 source=list()
          7 score=list()
          8
          9 for peptide_id in peptide_ids:
          10     for hit in peptide_id.getHits():
          11         sequence.append(hit.getSequence())
          12         source.append(hit.getMetaValue("target_decoy"))
          13         score.append(hit.getScore())
          14 data = {'Sequence':sequence,
          15       'Target_Decoys': source,
          16       'score':score
          17       }
          18
          19 df = pd.DataFrame(data)
          20 df
```

```
17     }
18
19 df = pd.DataFrame(data)
20 df
```

Out[11]:

	Sequence	Target_Decoyscore
0	VAPAPARPSGPSK	target9.505832
1	VAPAPARPSGPSK	target9.412774
2	MHAPPINMESVHMK	target0.642283
3	MHAPPINMESVHMK	target0.804148
4	VSETEVFTHQK	decoy0.136526
...
199	MVLLAAAVC(Carbamidomethyl)TK	target0.031847
200	GTNMLKPFAALLGEIR	decoy0.061485
201	DGGLQNMELHGMIM(Oxidation)LR	target0.186863
202	YGLAVIEDFAFILDFC(Carbamidomethyl)HESIENEELAR	decoy0.234703
203	ENVNLAQIR	target0.084336

204 rows × 3 columns

In [12]:

```
1  #To remove all decoy
2  idfilter.removeDecoyHits(peptide_ids)
3  sequence=list()
4  source=list()
5  score=list()
6
7  for peptide_id in peptide_ids:
8      for hit in peptide_id.getHits():
9          sequence.append(hit.getSequence())
10         source.append(hit.getMetaValue("target_decoy"))
11         score.append(hit.getScore())
12  data = {'Sequence':sequence,
13         'Target_Decoys': source,
14         'score':score
15         }
16
17  df = pd.DataFrame(data)
18  df
```

```
15     }
16
17 df = pd.DataFrame(data)
18 df
```

Out[12]:

	Sequence	Target_Decoys	score
0	VAPAPARPSGPSK	target	9.505832
1	VAPAPARPSGPSK	target	9.412774
2	MHAPPINMESVHMK	target	0.642283
3	MHAPPINMESVHMK	target	0.804148
4	MHAPPINMESVHMK	target	0.076567
...
109	LYM(Oxidation)VLITTK	target	0.034025
110	M(Oxidation)VLLAAAVC(Carbamidomethyl)TK	target	0.781185
111	MVLLAAAVC(Carbamidomethyl)TK	target	0.031847
112	DGGLQNMELHGMIM(Oxidation)LR	target	0.186863
113	ENVNLAQIR	target	0.084336

114 rows × 3 columns



THANK YOU