|  |  |
| --- | --- |
| A picture of a winding road and trees  **Intelligent LC/MS Feature Detection** | Abstract  This algorithm helps to analyze Liquid chromatography (LC/MS) data. This data is generally about compounds from very complex biological samples such as serum, urine, or body tissue.  Shah, Dharak |

Contents

[Algorithm Scope 2](#_Toc491262764)

[Aim of Algorithm 2](#_Toc491262765)

[Work Flow of Algorithm 3](#_Toc491262766)

[Algorithm Execution Steps 4](#_Toc491262767)

[Brief Description of Classes Used in Code 4](#_Toc491262768)

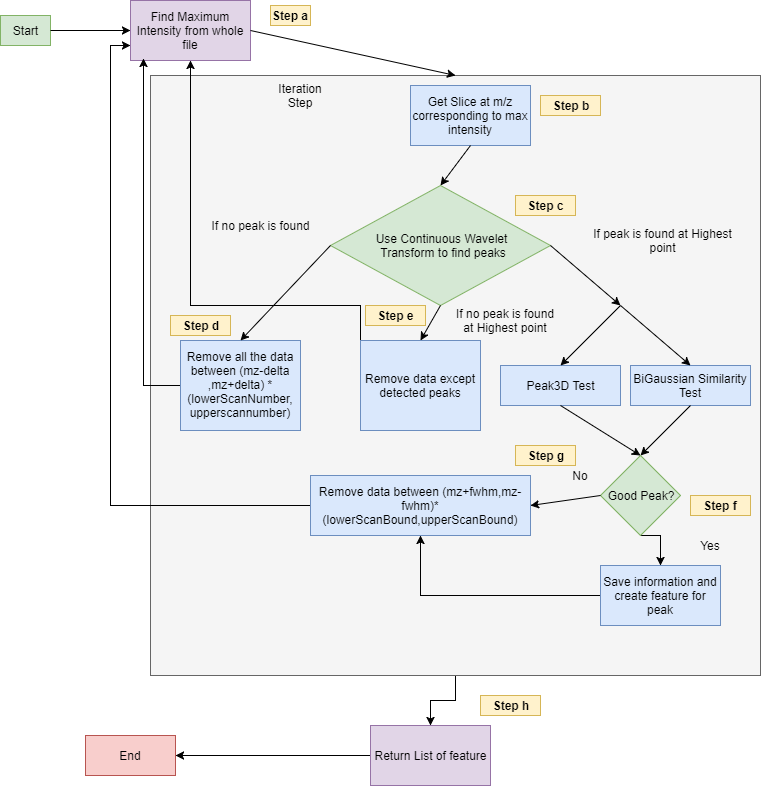
# Algorithm Scope

Liquid chromatography- or gas chromatography-mass spectrometry (LC/MS or GC/MS) are analytical chemistry techniques that combine the separation capabilities of chromatography with the mass measurement capabilities of mass spectrometry and can detect compounds from very complex biological samples such as serum, urine, or body tissue. Modern day high throughput experiments require measurement (identification and quantitation) of thousands of chemical compounds in a sample simultaneously. Many compounds might have close or even the same masses, e.g. molecules having the same elemental composition but different structures. Liquid (LC) or gas (GC) chromatography is used to separate such compounds in time (retention time in chromatographic column, RT). After this separation mass spectrometry provides structural identity of the individual components with high molecular specificity and detection sensitivity. The data from LC/MS and GC/MS experiments is 3 dimensional: m/z (mass to charge ratio), intensity, and retention time. To process this data various algorithms have been developed, such as centWave (in XCMS software), GridMass (in MZmine 2 software), FeatureFinderMetabo (in OpenMS software), or others.Most algorithms process this 3D data in three separate steps for detecting analyte-relevant features. Each of these three steps processes a projection of the 3D data into a 2D space - intensity vs m/z, intensity vs. RT, and m/z vs RT. This way of feature detection contributes partially to the non-negligible detection rate of false positive features, which is a widespread problem for all widely used algorithms. In addition, all current algorithms require users to manually specify some parameters like noise threshold, minimum signal-to-noise, expected chromatographic peak width, etc. The process of parameter selection is, of course, highly algorithm and data dependent and requires both the knowledge of the algorithm’s inner workings and the characteristics of the dataset it’s being applied to. The problem is further aggravated by the fact that many mass spectrometry users are not actually experts in mass spectrometry, but rather use it as an analytical tool that they don’t understand, and often have little to no knowledge about the critical issues of mass spectrometry data processing.

# Aim of Algorithm

The major aim of implementing this algorithm is to produce more accurate feature detection results by removing user-defined parameters and processing the 3D data directly. In addition, mass spectrometry processing software tools like MZmine can quickly adopt this new algorithm, and thus provide more robust, user friendly way to extract biomedical knowledge from MS datasets.

# Work Flow of Algorithm



**Please refer below section for description of steps**

# Algorithm Execution Steps

* 1. Read raw data file and convert it into sparse matrix. Find the maximum intensity point from whole sparse matrix.
  2. Take the slice of m/z value corresponding to highest intensity.
  3. Use the wavelet transform to determine peak boundaries.
  4. If we don’t find any peak from the wavelet transform remove all the data between range of (mz-delta,mz+delta) and (lowerScanBound, upperScanBound). Find next maximum from the sparse matrix and continue the process.
  5. If we find peak but it’s not at Highest intensity point then remove data except detected peak. Find next maximum from the sparse matrix and continue the process.
  6. If we find the peak at Highest intensity point then test that peak Peak3D test and BiGaussian Similarity test. If both returns true then it’s good peak. Then store the peak data and remove the data between (mz+fwhm, mz- fwhm) and (loweScanBound,UpperScanBound). Find next maximum from the sparse matrix and continue the process.
  7. If peak is not good then also remove the data between (mz+fwhm, mz- fwhm) and (loweScanBound,UpperScanBound). Find next maximum from the sparse matrix and continue the process.
  8. Create features for good peaks.

# Brief Description of Classes Used in Code

a. SliceSparseMatrix Class

* + 1. This class converts raw data file into sparse matrix.
    2. Column indices are scan indices and row indices are m/z values and elements are intensity value.
    3. In the constructor of this class we’ve sorted list of data points in 3 diverse ways. According intensity, according scan index and according scan index and m/z both to support slicing horizontally and vertically.
    4. For fast retrieval of element from matrix binary search has been used.
    5. This class contains methods for vertical slicing, horizontal slicing, removing and restoring data points, finding maximum intensity value, get retention time values from scan numbers, creating data point list for Continuous Wavelet Transform.

1. ContinuousTransformWavelet Class
2. This class is used to determine peak boundaries from list of intensities and retention time provided by SliceSparseMatrix class getCWTDataPoint method.
3. CurveTool Class
   * 1. This class is used to determine Full Height Half Max(fwhm) of whole raw data file.
     2. This class also contains methods for normalizing EIC with different parameters.
4. Peak3D Class
   * 1. This class is used to run peak similarity test for peaks determine by ContinuousWaveletTransform. This is first of two tests for finding good peak.
     2. Peak is tested by comparing similarity between adjacent m/z slices. Let mzValues be a sorted list of all m/z values in the profile data. Let index be an integer such that mzValues[index] == mz.
     3. We find similarities between the EIC corresponding to m/z value mzValues[index] and adjacent EICs corresponding to ..., mzValues[index-2], mzValues[index-1], mzValues[index+1], mzValues[index+2], ... if those similarities are higher than the similarity threshold.
     4. First, we check each m/z-value higher than mzValues[index], stop when the current similarity becomes lower than the similarity threshold, and save the last m/z-value (variable upperMZbound). Next, we check each m/z-value lower than mzValues[index], stop when the current similarity becomes lower than the similarity threshold, and save the last m/z-value (variable lowerMZbound)
     5. Peak is good if the differences upperMZbound - mzValues[index], mzValues[index] - lowerMZbound, upperMZbound - lowerMZbound exceed certain thresholds, which depend on FWHM-value.
5. BiGaussian Class
   * 1. BiGaussian Class is used for fitting BiGaussian on EIC. BiGaussian is composed of 2 halves of Gaussian with different standard deviations. It depends on 4 parameters (height, mu, sigmaLeft, sigmaRight) and computed by the formula
6. BiGaussianSimilarityTest Class

This is second of two tests. BiGaussianSimilarityTest class is used for determining true or false peak by comparing BiGaussian values with intensity values of given m/z and left and right bounds (variables leftBound and rightBound).

1. ADAP3DPeakDetectionAlgorithm Class
   * 1. ADAP3DPeakDetectionAlgorithm is used to run iteration step of the algorithm which determines whether peak is good or bad and removes data accordingly.
     2. It has two execute methods to execute iteration step. One determines the first 20 peaks with default parameters and other determines rest of the peaks with newly estimated parameters.
2. ADAP3DFeatureDetectionMethod Class
   * 1. This is the main class to run this algorithm. It calls execute methods of ADAP3DPeakDetectionAlgorithm class. It creates feature for each good peak and returns the list of features.