DECONVOLUTION IN GC/MS-LIKE SITUATIONS

DREW JOHNSON

1. Introduction

Gas chromatography-mass spectrometry (GC/MS) is a powerful method for identifying chemical substances in complex mixtures. The use of this method introduces some interesting mathematical problems. A mass spectrometer works by fragmenting a compound into ions. An ion detector can then count the intensity of ions with distinct mass to charge ratios, giving a *mass spectrum*. This mass spectrum is a fingerprint that can be matched against a library of known compounds. Mass spectrometry is useful for identifying uncontaminated, pure samples. A gas chromatograph forces a sample through a column. Depending on the properties of the column, different compounds are retained in the column for different amounts of time. The amount of elution at the end of the column can be detected and recorded. Gas chromatographs are useful for separating compounds, but the retention time of a compound alone is not always enough to uniquely identify it. We can combine these techniques into a powerful "hyphenated method" by running the mass spectrometer multiple times as the sample elutes (comes out) from the column. However, especially when a mixture contains several similar compounds or when the sample is run through the column quickly, the elution times of some compounds may overlap. In this case, we may hope to deconvolve them by mathematical means.

2. Set up

We have a $M \times T$ data matrix X, where M is the number of distinct mass to charge ratios that the scanner is set to detect, and T is the number of scans or observations. Each column of X is a linear combination of an unknown number, n, of unknown spectra of the actual components $s_1, s_2, \ldots s_n$. (In this paper, the word "spectra" is referring to mass spectra, a chemistry concept, rather than the mathematical concept with the same name.) Each pure component has an (also unknown) concentration profile (or elution profile), $c_1, c_2, \ldots, c_n$, which is a vector of length T that represents how much of the component eluted at any time. In other words, if c_i are the rows of C and s_i are the columns of S,

$$X = SC$$

Each entry of all three of these matrices is positive. Our goal is to recover the vectors s_i.

In general, factoring a matrix into two matrices this way is non-unique. By using additional assumptions or regularization criteria, we can attempt to identify good solutions. In the methods we tried, we assumed that the number of components n was known or could be estimated accurately.

3. Directly Fitting

One approach is to attempt to directly fit a model to the situation. We assume that the elution profiles have shapes $C_{it} = f_{\theta_i}(t)$, where f is a family of functions with parameters θ. Then, we must solve the optimization problem

$$\min_{\{\hat{\theta}_k\},\hat{S}} \left|\left| X - \hat{S}\hat{C}(\{\hat{\theta}_k\}) \right|\right|$$

where $\hat{C}(\{\hat{\theta}_k\})$ is the estimate for C using the parameters $\{\hat{\theta}_k\}$. If we use the Frobenius norm (and square the objective), this problem is linear in $\hat{S}$. However, it may be quite non-linear in $\{\theta_k\}$.

One model I have used for f_θ is the family $A \exp\left(-\left(\frac{t-\mu}{\sigma}\right)^2\right)$ with parameters A, μ, and σ. This seems to be a reasonable approximation of shapes for concentration profiles.

The Nelder-Mead simplex method [4] was used to solve the optimization problem. The linearity in $\hat{S}$ makes this problem tractable; however, it seems to have many local minima, so a good starting point is critical. The most effective method we found was to get a starting point for $\hat{\theta}$ using an estimate for C from another method such as NNMF or AR (discussed in Section 5). Several variations of this "Direct Fit" algorithm were tested in Section 8. There is a two parameter method where the coefficient A was omitted, and the family $\exp\left(-\left(\frac{t-\mu}{\sigma}\right)^2\right)$ was used for the fit, even the the randomly generated examples used $A \exp\left(-\left(\frac{t-\mu}{\sigma}\right)^2\right)$. In some situations, this variation seems to perform better. We hypothesize that this is the case because when these coefficients A were similar or the same, the algorithm would benefit by having fewer variables to optimize over and not be hurt too much by the loss of flexibility when fitting.

4. Using convexity

4.1. **Normalizing.** One very interesting way to approach this problem is to exploit a normalization trick that turns the linear combinations into convex combinations. This trick was suggested by Grande and Manne in [1].

Let x_t be the tth column of X. Take a unit vector p, with the properties that $x_t^T p > 0$ for all x_t, and p has no negative entries.

Now, we normalize by assigning

$$y_t = \frac{x_t}{p^T x_t}$$

Geometrically, this amounts to truncating or extending each vector so that it lies in the hyperplane supported by p.

Now, that means the normalized spectra of the pure components are

$$e_i = \frac{s_i}{p^T s_i}.$$

Now any x_t, the tth column of X, is a linear combination of the pure spectra $x_t = \sum C_{it} s_i$. Thus

$$y_t = \frac{x_t}{p^T x_t} \tag{1}$$

$$= \sum_i \frac{C_{it} s_i}{p^T x_t} \tag{2}$$

$$= \sum_i \frac{C_{it} s_i}{p^T x_t} \frac{p^T s_i}{p^T s_i} \tag{3}$$

$$= \sum_i \frac{s_i}{p^T s_i} \left(\frac{C_{it} p^T s_i}{p^T x_t} \right) \tag{4}$$

$$= \sum_i e_i \frac{C_{it} p^T s_i}{p^T x_t}. \tag{5}$$

Now, $\sum_i C_{it} p^T s_i = p^T x_t$, so $\sum_i \frac{C_{it} p^T s_i}{p^T x_t} = 1$. Since $\frac{C_{it} p^T s_i}{p^T x_t} \geq 0$ for all i, we have that y_t is a convex combination of $e_1, e_2, \ldots e_n$.

4.2. **Finding S.** Now, if each observation is a convex combination of a finite set of points, then each observation lies in a convex polytope with the pure spectra as vertices. Since X is of rank n (ignoring noise), and the normalization eliminated one degree of freedom, we can represent these as points in $n-1$ dimensional space, and now our polytope is a simplex. Figure 1 shows a graphical representation of a hypothetical data matrix of a sample with three components with each ion (row of X) plotted in a different color. Figure 2 shows a representation of these same data (with loss of information about total intensity) as convex combinations of the vertices of a simplex (a triangle in this case).

We can examine the normalized observations y_i, and then estimate where the vertices of the containing simplex lie. This problem is of course quite ill posed, since there are infinitely many simplices that contain a given set of points.

Looking at Figure 2, we may be tempted to choose the two "endpoints" as two of our estimates for actual spectra, and then do two linear extrapolations using the first few points on each side, and calculating their intersection. This is the method tested in Section 8 as "Convex Extrapolate". However, for more difficult problems where the peaks are closely overlapping, the shape is not as "nice" as that seen in 2, and the intersection of the extrapolations can be inside the convex hull of the data, or in other unreasonable places. In addition, there is no sound theoretical justification for this method; it just seems natural geometrically.

We can make some improvements if we have some knowledge about the shape of C. Since rows of C are "continuously" changing with respect to time, it seems natural to use a model similar to that used in Bezier curves. This kind of model uses a partition of unity, which is a set of functions $\{m_i(t)\}$ where $\sum_i m_i(t) = 1$. The curve B is then defined by a set of control points, P_i, by

$$B(t) = \sum_i m_i P_i.$$

A standard nth degree polynomial Bezier curve uses the partition of unity defined by the Bernstein polynomials $\{b_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}\}$. The algorithm "Convex, Bezier Fit" (see Section 8 works by fitting a curve of this form to the data, and then

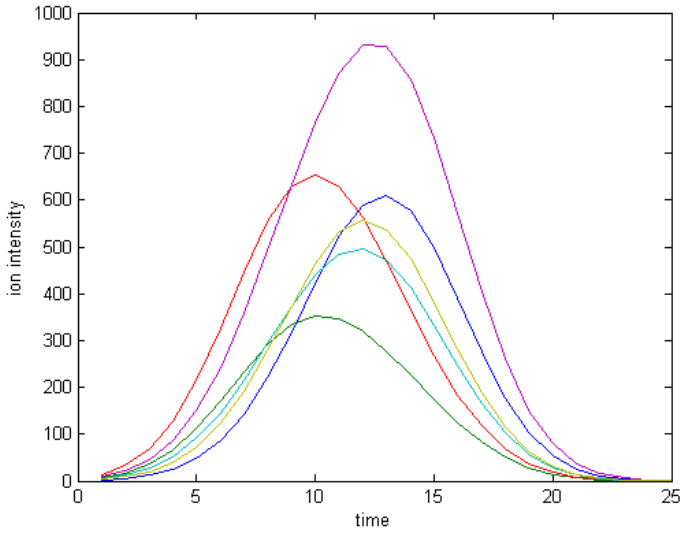

FIGURE 1. A plot of a hypothetical data matrix X.

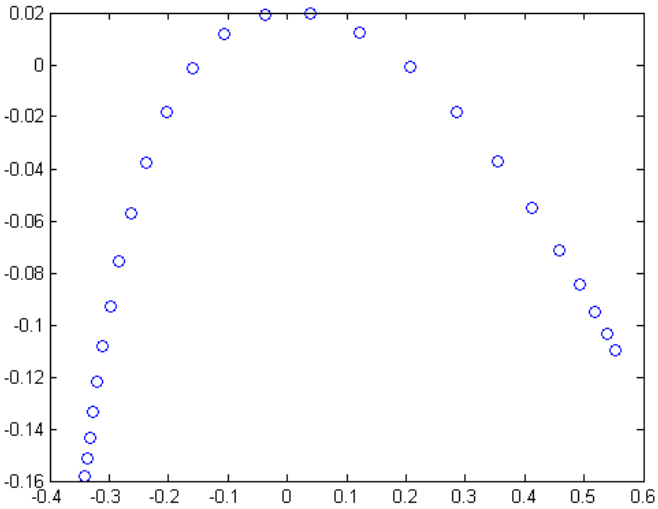

FIGURE 2. A convex representation of Figure 1.

using the control points (transformed back into the original space) as estimates for the actual spectra. This tends to produce better estimates than the extrapolation method. We can do better, however, if we use more assumptions about the shapes of the concentration profiles.

In general, if we assume as in Section 3 that the components have shapes $C_{it} = f_{\Theta_i}(t)$, it seems reasonable to use the partition of unity

$$\left\{ \frac{f_{\theta_k}(t)}{\sum_{j=1}^{n} f_{\theta_j}(t)} \right\}_{k=1}^{n}.$$

This means that again we must solve an optimization problem involving minimizing a residual between the data and the fit, over the variables $\{\hat{\theta}_k\}$ and the control points. This is still a fairly complex problem with local minima. Although the representation of the data as a set of convex combinations is appealing, it is not yet completely clear what advantage this has over more direct methods like those in Section 3. However, empirical evidence suggests that it does perform well in certain situations.

4.3. **The Choice of p.** The properties noted in Section 4.1 ($p \geq 0$ and not orthogonal to any x_i) are sufficient to give the convexity property. The original paper [1] suggested using the first singular vector (the first column of U from the SVD). This seems reasonable because the span of the first n columns of the SVD is a good estimate for the span of s_i.

One property that seems like it would be desirable would be to choose a p so that $p^T s_i$ is the same for all i. This will give (from equation (5))

$$\begin{aligned} y_t &= \frac{p^T s_1}{p^T x_t} \sum_i f_{\Theta_i}(t) e_i \\ &= \frac{p^T s_1}{p^T \sum_i f_{\Theta_i}(t) s_i} \sum_i f_{\Theta_i}(t) e_i \\ &= \frac{p^T s_1}{\sum_i f_{\Theta_i}(t) p^T s_i} \sum_i f_{\Theta_i}(t) e_i \\ &= \frac{1}{n \sum_i f_{\Theta_i}(t)} \sum_i f_{\Theta_i}(t) e_i. \end{aligned}$$

This is exactly the partition of unity (within a constant factor) that we suggested in Section 4.2. Thus, choosing this p ensures that, assuming the actual profiles really do come from the family f_θ, our partition of unity model is accurate. A geometric way to describe this situation is that the simplex and its containing points are determined, within an affine transformation, only by C and not by S.

Vectors with the property that $p^T s_i$ is constant are not unique. Any will do, as long as $p^T s_i \neq 0$. To avoid this and encourage numerical stability, we want to choose p so that $p^T s_1$ is at a maximum. To find this p, if we assume we know s_i, we can find the subspace $P = \{p : S^T p = k\mathbf{1}\}$ which is equal to $\text{span}\{\mathcal{N}(S^T) \cup \{p_0\}\}$, where p_0 is a particular solution to $S^T p = \mathbf{1}$. Then, the best solution is s_1 projected onto this subspace, and normalized.

Numerical experiments suggest that this method is slightly better than the first singular vector method in some cases. However, testing this scheme requires us to "cheat" by using the actual values of s_i to calculate p. It is not clear whether there is a good way to estimate this p, especially since the whole problem in the first place is to estimate s_i. We can use an estimate obtained by another method, such as non-negative matrix factorization, as an estimate for s_i, and use this estimate to find p. This is the approach used in the "special" variations in Section 8.

Another option is to take the vector that supports the n-dimensional hyperplane that passes through each of the s_i and normalize it. This seems to be geometrically satisfying. Numerical experiments again suggest improvement over the first singular vector method, but we have the same problem of needing to know s_i in order to calculate p.

In random tests the vectors p selected by both of these methods are quite similar to the first singular vector, especially when all the concentration profiles are approximately the same size and shape.

All of these methods involve finding the intersections of rays with an affine space. Although the first alternative method preserves the ratios of the intensities of the compounds, there is still some distortion, as two pairs of vectors with the same angle between them will have different Euclidean distances between them in the space where the simplex lives — the pair of vectors which are almost normal to the plane may be closer than the pair which is less close to normal. This may be a problem because when our optimization routine measures error, the error may be measured inconsistently. This issue could be resolved if we project the points onto the unit sphere. This will still preserve convexity, and would cause none of this kind of distortion, and also preserve the relative values of coefficients. However, it seems that it may complicate the computations in curve fitting and destroy what linearity we have.

In Section 8 these algorithms are referred to as "ConEx", an acronym for "Normalization to produce *Con*vexity followed by *Ex*ponential Fit", with each combination of the "special" and 2 and 3 parameter variations included.

5. Non-negative matrix factorization

The problem of factoring a positive matrix into a product of two positive matrices has applications in other areas as well, and several methods have been proposed. These methods are sometimes effective for the type of deconvolution we are attempting.

5.1. **Alternating Regression.**

The method described in Algorithm 1 was suggested specifically for the type of deconvolution we are discussing [2].

Algorithm 1 Alternating Regression [2]

Require: Data matrix X
  Fill S' with random positive entries.
  repeat
    $S \leftarrow S'$
    Calculate a least squares fit for C in $X = SC$: $C \leftarrow (S^T S)^{-1} S^T X$.
    Set the negative entries of C to zero.
    Force C to be unimodal by setting secondary humps to zero.
    Calculate a least squares fit for S' in $X = S'C$: $S' = (CC^T)^{-1} C X^T$.
    Set the negative entries of S' to zero.
  until $||S - S'|| < tol$, where we use the Frobenius norm

The idea of this algorithm is to produce a factorization that "looks good" based on some assumptions about the shape of the concentration profiles. The assumptions suggested by [2] are non-negativity and unimodality. This method may seem

ad hoc, but it works surprisingly well in many situations. Notice that the algorithm is non-deterministic, as it uses a random starting point, so one variation is to run the algorithm several times with different starting points. However, as the algorithm always produces outputs that "look good" it may be tricky to choose which output is the desirable one.

5.2. **Other non-negative matrix factorizations (NNMF).** More well known non-negative matrix factorizations include a multiplicative update method, such as that described by Lee and Seung [3]. Traditional alternating least squares methods also exist. They differ from Algorithm 1 in that they solve the least squares problem with a positivity constraint, and omit the coercion towards unimodality. MATLAB provides two implementations of NNMF — a multiplicative one, and an alternating least squares. We empirically found the alternating least squares implementation to be more effective for our application.

6. Peak Maximization Methods

In real applications in GC/MS, the spectra of actual compounds are usually sparse in the sense that they have many zero entries. This introduces the possibility of using methods which attempt to find ions which are unique to each component. This is the basic premise of AMDIS, one of the standards in the industry. The method of AMDIS is described in a paper by Stein [5]. Here, we describe a simple method which uses essentially the same ideas.

First, the MATLAB curve fitting toolbox function `fit` is run on each row of the data, using the `smoothingspline` option. Other types of fitting may also be appropriate, but the important point is to have a polynomial model (or another type of model that can be evaluated quickly) so we can interpolate between the data and simulate a higher resolution. We find the times of local maxima in the model and record them. We then use a clustering algorithm to group them into n groups. We then take the median of each group, and any ions that maximize "close" to it, and add them up and use that for an estimate of the concentration profile. We then do a non-negative least squares fit to these profiles to estimate S. This method seems to work quite well in practice when the data is sparse. Of course it fails completely in the non-sparse case.

Two variations of this idea are seen in Sections 8. They differ in the setting of parameters. The first is the increase in resolution when evaluating the fit, and the second is a parameter that controls how close an ions maximization has to be to the median in order for it to be considered as part of the estimation of the profile shape.

7. The Denizen Method

The Denizen method was developed by James Oliphant and others at Torion. The author has done some analysis to explain why it works.

The first round of the "Denizen" algorithm is given in Algorithm 2. We hope that spectra of the original components will be among the extracted vectors v_i.

The following observations will help us justify the use of this algorithm.

Proposition 1. *Let $v, x \geq 0$ (entry-wise), with $||v|| = 1$. Let $D = I - vv^T$. Then, either $Dx = 0$, or the entries of the vector Dx have at least one positive and one*

Algorithm 2 The Basic Denizen Method

Input: Data matrix X
  $X^{(1)} \leftarrow X$
  $i \leftarrow 1$
  while $\min_j ||x_j|| > \epsilon$ **do**
    $k \leftarrow \arg\max_j \left|\left|x_j^{(i)}\right|\right|$
    $v_i \leftarrow \frac{x_k^{(i)}}{\left|\left|x_k^{(i)}\right|\right|}$
    $D \leftarrow I - v_i v_i^T$
    $X^{(i+1)} \leftarrow DX^{(i)}$
    Set all negative entries of $X^{(i+1)}$ to zero.
    $i \leftarrow i + 1$.
  end while

non-positive entry among them. $Dx = 0$ iff x is a multiple of v. Furthermore, Dx has no more positive entries than does x.

Proof. Recall that D is projection onto the orthogonal complement of the space spanned by the vector v. Thus, $Dx \perp v$, and $Dx = 0$ iff x is a multiple of v.

Now, assume for contradiction that Dx has only positive entries. Then $v^T Dx > 0$, a contradiction.

Next, assume that Dx contains no positive entries. Since $v^T Dx = 0$, we must have that $v_i \neq 0$ implies $(Dx)_i = 0$. Also, when $v_i = 0$, then $(Dx)_i = x_i - (v^T x)v_i = x_i \geq 0$. We conclude $Dx \geq 0$, and since Dx was assumed non-positive, $Dx = 0$.

To see the last claim in the conclusion, note that if $x_i = 0$, then $(Dx)_i = x_i - (v^T x)v_i = -(v^T x)v_i \leq 0$. □

Corollary 1. *Let $\{u_i\}_{i=1}^n$ be an orthonormal set and let $v = \sum_{i=1}^n m_i u_i$ with $m_i \geq 0$ and $||v|| = 1$. Let $D = I - vv^T$. Then, if $x = \sum_i a_i u_i$ with $a_i \geq 0$, then $Dx = \sum_{i=1}^n b_i u_i$, where b_i are either all zeros or contain at least one positive and one non-positive number. The b_i are all zero iff x is a multiple of v. Moreover, there are no more positive numbers among the b_i than among the a_i.*

Proof. Simply write v and x in the basis of $\{u_i\}_{i=1}^n$ and apply Proposition 1. □

So how does the Denizen method work? Assume for now that n pure spectra $\{s_i\}$ form an orthonormal set. On the first iteration, the algorithm selects the column of X with the maximum norm to extract. This vector is of course in the positive span of $\{s_i\}$, as we assumed. All other columns of X are also in the span of $\{s_i\}$, so the Corollary applies, so before truncating the negatives, each column of $X^{(1)}$ is a linear combination of $\{s_i\}$ with some positive and some non-positive coefficients. But since we throw out the negative coefficients, each column of $X^{(1)}$ is a positive combination of $\{s_i\}$ with at most $n - 1$ non-zero coefficients!

We now repeat this process. There may be different regions where different sets of $n - 1$ components are present. The next vector to extract is a positive linear combination of the $n - 1$ components in that region, and so now each column of $X^{(2)}$ in that region is a linear combination of at most $n - 2$ components from $\{s_i\}$. For the other regions, we apply the Corollary with the original s_i, and thus be

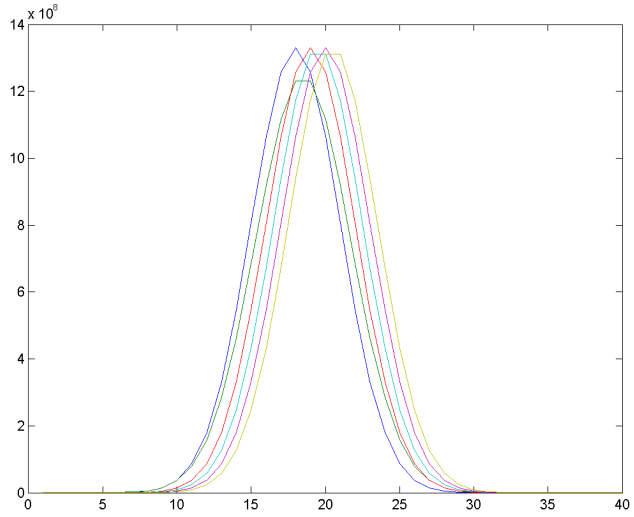

FIGURE 3. An Example of the Denizen Algorithm: The original distributions.

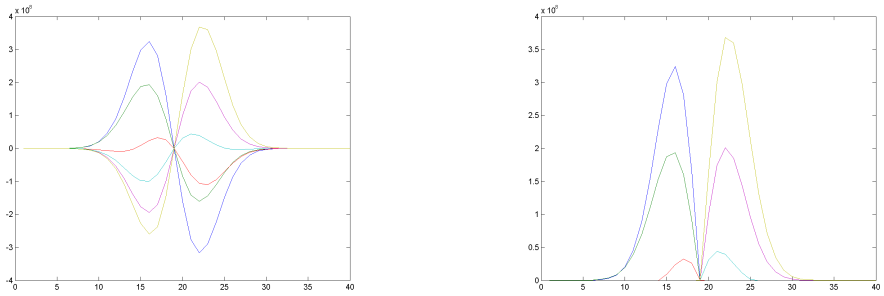

FIGURE 4. An Example of the Denizen Algorithm: After iteration 1

assured that those regions will still be regions with at least one component present, but no more components than they had before.

If we continue this process, eventually we will come to the point where we extract pure vectors (vectors with only one component). The only danger is if one extraction step completely eliminates two components at once.

The plots in Figures 3 through 14 in this document illustrate this step by step for a hypothetical example with 6 components, each of which has one ion. The original data matrix X is shown in Figure 3. The first picture in each subsequent figure shows a plot of the rows of $X^{(i)}$ before truncation, and the second shows $X^{(i)}$ after truncation.

In a real situation, we do not have perfect orthogonality of all the mass spectra, however, given a random pair of actual spectra from the NIST library, it is empirically quite likely that the pair will be *approximately* orthogonal. Thus, the

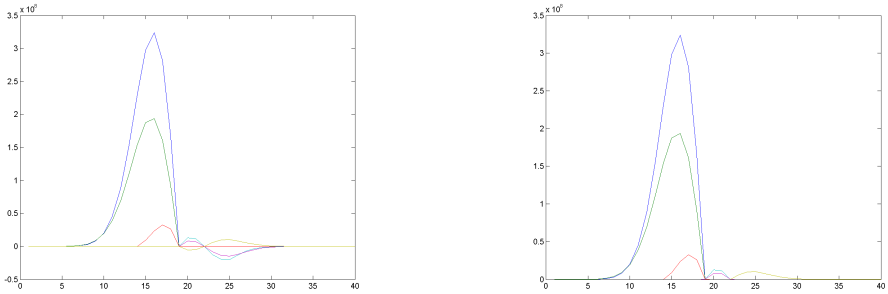

FIGURE 5. An Example of the Denizen Algorithm: After iteration 2

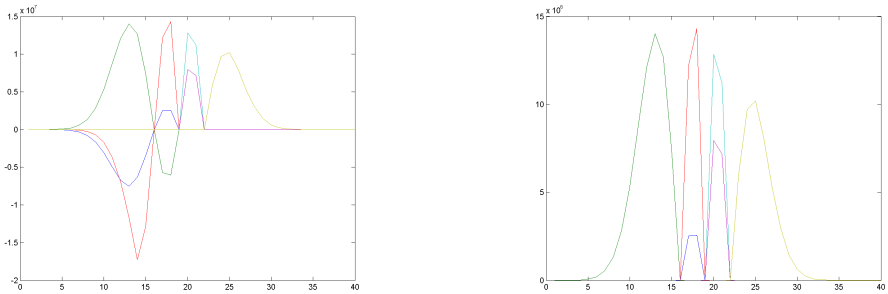

FIGURE 6. An Example of the Denizen Algorithm: After iteration 3

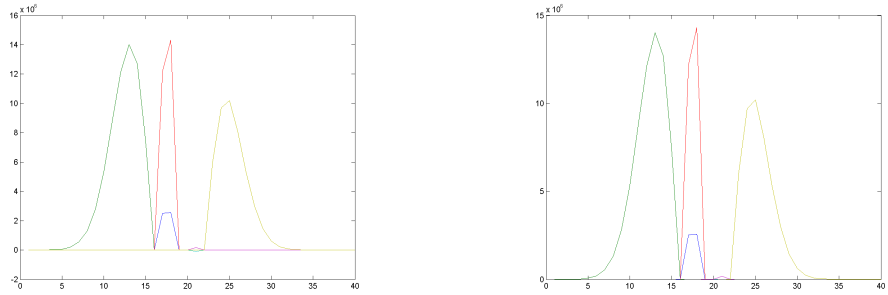

FIGURE 7. An Example of the Denizen Algorithm: After iteration 4

algorithm works quite well in many situations. Some preliminary results bounding the magnitude of this error are found in Theorem 1 in Section 7.1.

Notice in the example that after the first 4 spectra are extracted, although the last two can still theoretically be extracted perfectly, this is a very bad situation numerically, as the maximum intensity drops over 13 orders of magnitude (between Figures 11 and 12). In the extremely noisy real world, recovery in this situation is impossible. Thus, it is preferable to return to the original data matrix and project out the extracted pure spectra.

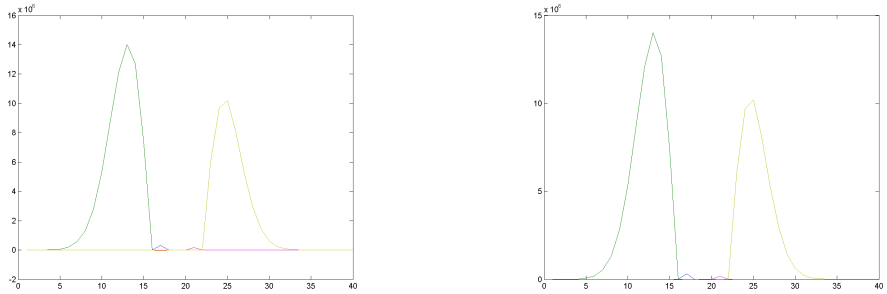

FIGURE 8. An Example of the Denizen Algorithm: After iteration 5

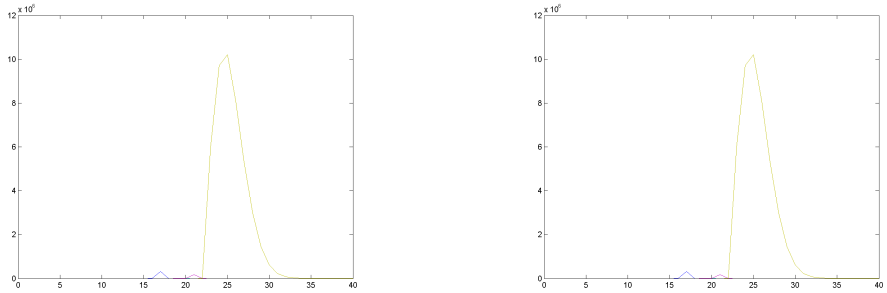

FIGURE 9. An Example of the Denizen Algorithm: After iteration 6

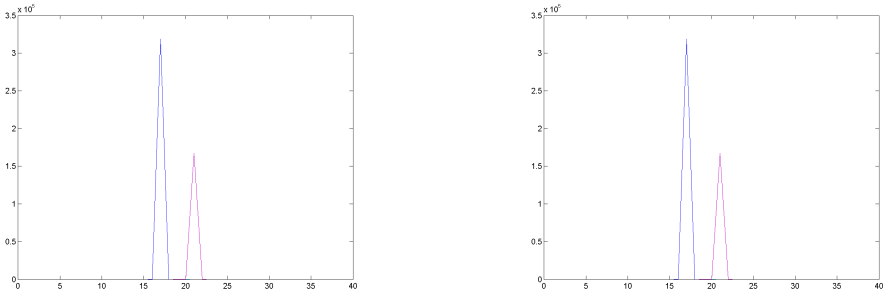

FIGURE 10. An Example of the Denizen Algorithm: After iteration 7

There may be several ways to determine which extracted spectra are pure. For our tests, we use a simple criterion based on empirical observation — we have observed that in our type of tests, the $n-1$ extracted vectors after the first are relatively pure. Thus, we take these $n-1$ vectors $v_2, \ldots, v_n$ and project their span out from the original data matrix. Under the assumptions that these $v_2, \ldots, v_n$ are accurate estimates for $n-1$ of the s_i and that the s_i are orthonormal, only the column from S not estimated by one of $v_2, \ldots, v_n$ will remain in the projected data matrix. Then, we replace v_1 with the column of the projected matrix that has the

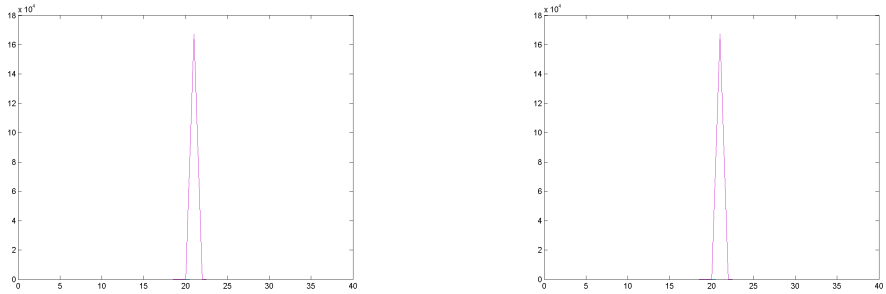

FIGURE 11. An Example of the Denizen Algorithm: After iteration 8

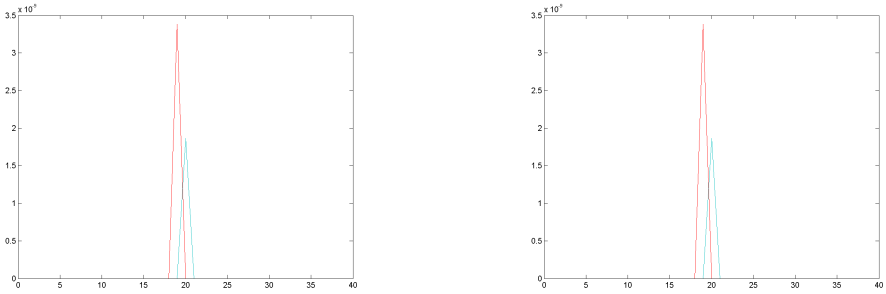

FIGURE 12. An Example of the Denizen Algorithm: After iteration 9

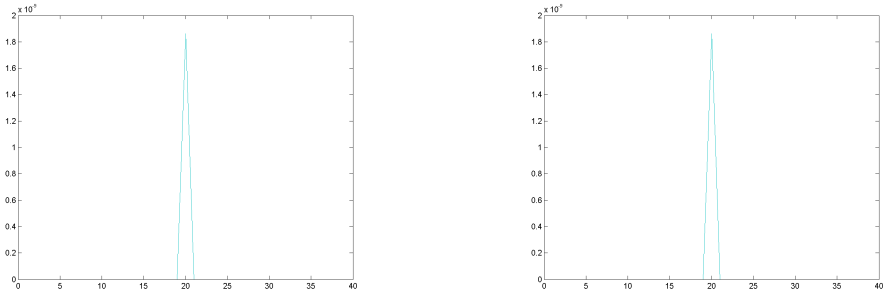

FIGURE 13. An Example of the Denizen Algorithm: After iteration 10

greatest norm. We repeat this procedure for $v_2, \ldots, v_n$, each time projecting out the span of the other $n-1$ vectors. Sometimes, repeating this procedure improves the estimate. In Section 8, Denizen (one iteration) does only one iteration, while Denizen (50 iterations) does 50.

One other variation that was used was to first smooth the data with a LOESS filter before running the algorithm on it. In noisy situations, this provided some improvement in results. This variation is known in Section 8 as "sDenizen".

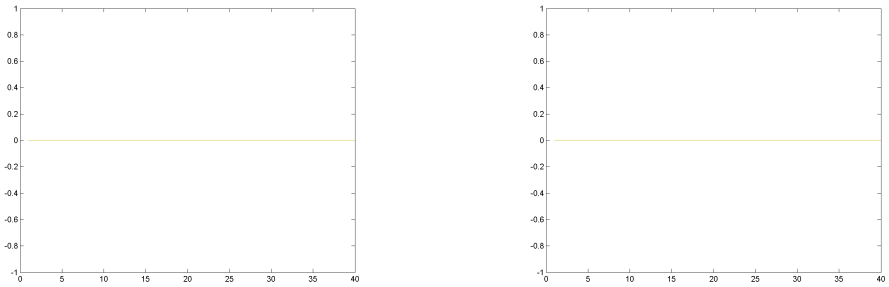

FIGURE 14. An Example of the Denizen Algorithm: After iteration 11

7.1. **Error analysis for deviations from orthogonality.**

Theorem 1. *Assume that* $\{u_i\}_{i=1}^n$ *is a linearly independent set of unit vectors that is "approximately orthogonal" in the sense that* $\max_{i\neq j} |u_i^T u_j| < \epsilon$ *for an* $\epsilon < \frac{1}{n}$. *Then, if* $v = \sum m_i u_i$, $D = I - vv^T$, *and* $x = \sum a_i u_i$, *with* $m_i \geq 0$, *and* $||v||_2 = 1$, *then* $Dx = \sum b_i u_i$ *for some* $\{b_i\}$ *with at least one of the* $\{b_i\}$ *less than* $\epsilon \, ||x||_2 \left(\frac{n}{1+\epsilon(1-n)}\right)^{\frac{3}{2}}$.

Proof. Take an orthonormal set of vectors $\{\hat{u}_i\}_{i=1}^n$. Let $\hat{v} = \sum m_i \hat{u}_i$, let $\hat{D} = I - \hat{v}\hat{v}^T$, and let $\hat{x} = \sum a_i \hat{u}_i$. Now, $Dx = \sum b_i u_i$ for some b_i, and $\hat{D}\hat{x} = \sum \hat{b}_i \hat{u}_i$ for some $\hat{b}_i$. Now, by Corollary 1, we know that at least one of $\hat{b}_i$, say $\hat{b}_k$, is non-positive. Now, we would like to see how far away b_k is from $\hat{b}_k$. First,

$$\begin{aligned}
\sum b_i u_i &= Dx \\
&= x - (v^T x)v \\
&= \sum a_i u_i - (v^T x) \sum m_i u_i \\
&= \sum (a_i - (v^T x) m_i) u_i
\end{aligned}$$

and thus we see that $b_k = a_k - (v^T x) m_k$. Similarly, $\hat{b}_k = a_k - (\hat{v}^T \hat{x}) m_k$. Thus, we see that $|b_k - \hat{b}_k|$ is no more than $m_k |v^T x - \hat{v}^T \hat{x}|$. Now,

$$\begin{aligned}
\left|v^T x - \hat{v}^T \hat{x}\right| &= \left| \left(\sum m_i u_i\right)^T \sum a_i u_i - \left(\sum m_i \hat{u}_i\right)^T \sum (a_i \hat{u}_i) \right| \\
&= \left| \sum m_i a_i + \sum_{i \neq j} m_i a_j u_i^T u_j - \sum m_i a_i \right| \\
&\leq \epsilon \sum_{i \neq j} |m_i a_j| \\
&\leq \epsilon \sum m_i \sum |a_i|
\end{aligned}$$

Thus, the error is bounded by

$$|b_k - \hat{b}_k| \leq \epsilon m_k \sum m_i \sum |a_i|. \tag{6}$$

Next, we need bounds on $\sum |a_i|$ and $\sum m_i$. We can get such bounds by solving the following maximization problem:

$$\begin{aligned} &\max_{\{a_i\},\{u_i\}} \sum a_i \\ &\text{s.t. } ||x||_2^2 = k \\ &|u_i^T u_j| < \epsilon \end{aligned} \tag{7}$$

The symbol k is an arbitrary parameter. First, we notice that $||x||_2^2 = \sum a_i^2 + \sum_{i \neq j} a_i a_j u_i^T u_j$. Next, we claim that the $\{a_i\}$ are positive at the optimal point. If one were negative, say a_k, making it positive would increase the the value of the objective function. Of course, this may violate the constraint, but if we simply change u_k to $-u_k$ the constraint is again satisfied. Now, we change the equality constraint to an inequality constraint. We will see that this does not change the optimal value of the objective. Now, knowing that $\{a_i\}$ are positive, and that the choice of $\{u_i\}$ affects only the constraint, we choose u_i so that $u_i^T u_j = -\epsilon$ for all i and j in order to make the constraint as loose as possible. We also transform the problem into a minimization problem. Thus, our problem becomes

$$\begin{aligned} &\min_{\{a_i\}} \quad -\sum a_i \\ &\text{s.t.} \sum a_i^2 - \epsilon \sum_{i \neq j} a_i a_j - k \leq 0 \end{aligned}$$

Now the objective function is linear, and the inequality constraint is convex for $\epsilon \leq \frac{1}{n}$. This is verified in Lemma 1. Thus, a point which satisfies the KKT conditions is in in fact an optimal point. The stationarity conditions for this problem are

$$\frac{\partial}{\partial a_k} \Lambda(\{a_i\}, \mu) = -1 + \mu(2a_k - 2\epsilon \sum_{i \neq k} a_i) = 0$$

Solving for μ gives

$$\mu = \frac{1}{2a_k - 2\epsilon \sum_{i \neq k} a_i}$$

Since this is true for all k, the symmetry inherent in this set of equations implies that $a_k = a_j$ for all k, j. We also see that by complementary slackness, since $\mu \neq 0$, we know the constraint is active as we claimed. We denote the value of a_i by $a^\star$, and then plugging into the original constraint, we get

$$n a^{\star 2} - \epsilon a^{\star 2}(n^2 - n) - k = 0$$

whence

$$a^\star = \pm \frac{\sqrt{k}}{\sqrt{n}\sqrt{1 + \epsilon(1-n)}}.$$

In order to maintain dual feasibility, we choose the positive value for $a^\star$. The value of the objective function for the original problem (7) at $a^\star$ is

$$\sum a^\star = \frac{\sqrt{nk}}{\sqrt{1 + \epsilon(1-n)}}.$$

Applying this to our particular problem, we have

$$\sum |a_i| \leq \frac{\sqrt{n}\,||x||_2}{\sqrt{1+\epsilon(1-n)}}$$
$$\sum m_i \leq \frac{\sqrt{n}}{\sqrt{1+\epsilon(1-n)}}$$
$$m_i \leq \frac{\sqrt{n}}{\sqrt{1+\epsilon(1-n)}}$$

Thus, from (6) we see that the error is bounded by

$$|b_k - \hat{b}_k| \leq \epsilon\,||x||_2 \left(\frac{n}{1+\epsilon(1-n)}\right)^{\frac{3}{2}}.$$

□

Lemma 1. *The function* $\sum_{i=1}^{n} a_i^2 - \epsilon \sum_{i\neq j} a_i a_j$ *is convex for* $0 \leq \epsilon < \frac{1}{n}$.

Proof. The Hessian is a matrix with 2 in every entry on the diagonal and -2ϵ in each other entry:

$$\begin{bmatrix} 2 & -2\epsilon & \dots & -2\epsilon \\ -2\epsilon & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -2\epsilon \\ -2\epsilon & \dots & -2\epsilon & 2 \end{bmatrix}$$

If we can show that the Hessian is positive definite, then we will have shown that the function is convex. Equivalently, we consider the matrix

$$H(n) = \begin{bmatrix} 1 & -\epsilon & \dots & -\epsilon \\ -\epsilon & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\epsilon \\ -\epsilon & \dots & -\epsilon & 1 \end{bmatrix}$$

We use Sylvester's criteria for determining if a matrix is positive definite: if the $m \times m$ submatrix in the top left corner has a positive determinant for every $1 \leq m \leq n$, then the matrix is positive definite.

First, we claim that the $r \times r$ matrix

$$M(r) = \begin{bmatrix} 1 & -\epsilon & \dots & -\epsilon & -\epsilon \\ -\epsilon & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & -\epsilon & \vdots \\ -\epsilon & \dots & -\epsilon & 1 & \vdots \\ -\epsilon & \dots & \dots & \dots & -\epsilon \end{bmatrix}$$

has determinant $-\epsilon(1+\epsilon)^{r-1}$. Applying cofactor expansion along the top row we see that except for the first and last entry of the top row, each submatrix corresponding to an entry of the top row has two columns equal to $-\epsilon\mathbf{1}$, so these have determinant

0. Thus,

$$\det M(r) = \det M(r-1) - (-1)^{r+1}\epsilon \begin{bmatrix} -\epsilon & 1 & -\epsilon & \dots & -\epsilon \\ \vdots & -\epsilon & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & -\epsilon \\ \vdots & -\epsilon & \dots & -\epsilon & 1 \\ -\epsilon & \dots & \dots & \dots & -\epsilon \end{bmatrix}$$

where the matrix shown is $r-1 \times r-1$. Note that this matrix has determinant $(-1)^{r-2}\det M(r-1)$, as it can be transformed into $M(r-1)$ by $r-2$ transpositions of columns. Thus, we get the recurrence relation

$$\begin{aligned} \det M(r) &= \det M(r-1) - (-1)^{2r-1}\epsilon \det M(r-1) \\ &= \det M(r-1)(1+\epsilon). \end{aligned}$$

The formula follows from this recurrence relation and the initial condition $\det M(1) = -\epsilon$.

Now, we consider the determinant of $H(n)$. Again applying cofactor expansion along the top row, note that the submatrix corresponding to the first entry is $H(n-1)$. The submatrix corresponding to the second entry can be transformed into $M(n-1)$ by $2(n-2)$ transpositions— move the first column to the end, and the first row to the bottom. Thus, this submatrix has the same determinant as $M(n-1)$. Now, the submatrix corresponding to the third entry of the first row can be transformed into the the submatrix corresponding to the second by one transposition of rows. Using similar arguments we find the recurrence relation

$$\begin{aligned} \det H(n) &= \det H(n-1) + \sum_{i=2}^{n} (-\epsilon)(-1)^{i-1}(-1)^{i} \det M(n-1) \\ &= \det H(n-1) - (n-1)\epsilon^2(1+\epsilon)^{n-1} \end{aligned}$$

Now, we argue by induction, using this relationship and with initial condition $\det H(2) = 1-\epsilon^2$, that

$$\det H(n) = (1-n)(1+\epsilon)^{n-1}(\epsilon - \frac{1}{n-1}).$$

for $n \geq 2$. Assume the formula holds for $n-1$. Then, by the recurrence relation

$$\begin{aligned} \det H(n) &= (2-n)(1+\epsilon)^{n-2}(\epsilon - \frac{1}{n-2}) - (n-1)\epsilon^2(1+\epsilon)^{n-1} \\ &= (1+\epsilon)^{n-2}\left((2-n)(\epsilon - \frac{1}{n-2}) + (1-n)\epsilon^2\right) \\ &= (1+\epsilon)^{n-2}\left((2-n)\epsilon + 1 + (1-n)\epsilon^2\right) \\ &= (1+\epsilon)^{n-2}(\epsilon+1)((1-n)\epsilon+1) \\ &= (1-n)(1+\epsilon)^{n-1}\left(\epsilon - \frac{1}{n-1}\right) \end{aligned}$$

We thus see that $H(n)$ is positive on $\left(-1, \frac{1}{n-1}\right)$, which proves the lemma. □

The Denizen algorithm assumes that after the first projection and truncation, any column of $X^{(2)}$ is a linear combination of at most $n-1$ of the actual components

s_i. If the orthogonality assumptions are not satisfied, then this may not be true. However, Theorem 1 assures us that any column of $X^{(2)}$ is a linear combination of $n-1$ of the actual components contaminated by no more than $\epsilon \left|\left|x\right|\right|_2 \left(\frac{n}{1+\epsilon(1-n)}\right)^{\frac{3}{2}}$ of another unit vector. For example in a three component system with $\epsilon = .01$, this amounts to 5.36% of $||x_t||$.

8. Numerical Results and Discussion

We designed a MATLAB function that will generate a random example and test several methods, and then report the results. Additionally, a function which calls this function many times and records the average performance of the algorithms was implemented. The first function generates random spectra according to a method specified by the user, and then generates peak profiles with the shape $A\exp\left(-\left(\frac{t-\mu}{\sigma}\right)^2\right)$. It then multiplies these together and truncates regions on the side with very small values to create a simulated data matrix X, and then calls each algorithm and stores its results.

Two methods for generating random spectra were used. First, spectra were generated with entries from a uniform distribution and a given sparsity level (the sparsity level indicates the expected fraction of the entries which are zero) and then normalized. The second method was to randomly select normalized spectra from a NIST library of over 147000 actual spectra. Only mass/charge ratios in the range 50-400 were used, since in real situations ions with mass to charge ratios under 50 are found in nearly everything.

The spacing refers to the distance in scans between the actual profile maximums (the parameters μ). Noise was added to each entry of data matrix from a exponential distribution with mean equal to the value of that entry, and then multiplied by the noise fraction parameter.

Tables 1 to 5 display some results. The "Ave Score" column gives an average of the score given to each estimate produced by the algorithm by taking an inner product of the normalized estimate and the actual spectrum. The "Perfects" column gives the number of trials in which that algorithm achieved "perfect" recovery, which was defined by achieving a total score (the sum of the score for each of the n spectral estimates) greater that $.99n$. The wins column gives a competitive statistic that may not be meaningful. The algorithm with the best score on a trial receives one win, or in the case that several algorithms achieve "perfect"' recovery, the win is shared between them. The "Ave. Time" is the average time used by the algorithm as measured by MATLAB's `tic toc` function.

"Data matrix" was also included — this was scored by taking the maximum score obtained by using a normalized column of the data matrix X as an estimate. This provides some control — if an algorithm cannot give a better estimate than the data we are given, perhaps it is not effective. Of course, however, selecting the best columns of X without prior knowledge of S is an interesting problem.

References

[1] Bjorn-Vidar Grande and Rolf Manne. Use of convexity for finding pure variables in two-way data from mixtures. *Chemometrics and Intelligent Laboratory Systems*, 50(1):19 – 33, 2000.

[2] Errki J. Karjalainen. The spectrum reconstruction problem: use of alternating regression for unexpected spectral compoments in two-dimensional spectroscopies. *Chemometrics and Intelligent Laboratory Systems*, 7(1-2):31–38, 1989.

TABLE 1

Random Spectra, sparsity = 0.2, $n = 3$
Spacing: 1
A = 1, 1, 1
σ = 3, 3.2, 2.8
100 iterations
Noise Fraction: 0

Algorithm	**Ave. Score**	**Perfect**	**Wins**	**Ave. Time**
AR	0.902831	0	0.00	0.056962
NNMF	0.931435	22	6.88	0.010428
Data Matrix (for control)	0.937473	0	0.00	0.000124
Peak Max (res 50, cut .04)	0.880782	0	0.00	1.248300
Peak Max (res 10, cut .1)	0.879961	0	0.00	1.246392
ConEx 2-param	0.917645	0	0.00	0.460377
ConEx 2-param, AR start	0.901869	46	15.18	0.359494
ConEx 2-param, NNMF start	0.965510	63	22.75	0.264661
ConEx 3-param	0.947837	0	0.00	0.443770
ConEx 3-param, AR start	0.885397	1	1.33	0.589327
ConEx 3-param, NNMF start	0.919740	32	10.35	0.395888
ConEx 2-param, special	0.916605	0	0.00	0.507530
ConEx 2-param, AR start, special	0.938628	41	11.95	0.431558
ConEx 2-param, NNMF start, special	0.916224	0	0.00	0.512900
ConEx 3-param, special	0.937298	0	0.00	0.544651
ConEx 3-param, AR start, special	0.872442	3	1.03	0.629233
ConEx 3-param, NNMF start, special	0.943526	0	0.00	0.521438
Direct Fit	0.906852	0	0.00	0.269381
Direct Fit AR	0.900935	22	6.18	0.311498
Direct Fit NNMF	0.947595	40	12.15	0.210221
Direct Fit 2-param AR	0.903016	9	2.48	0.230472
Direct Fit 2-param NNMF	0.938497	30	9.70	0.168406
Denizen 3 (one iteration)	0.736491	0	0.00	0.004815
Denizen 3 (50 iterations)	0.754720	0	0.00	0.028859
sDenizen 3 (one iteration)	0.734683	0	0.00	0.052560
sDenizen 3 (50 iterations)	0.741123	0	0.00	0.076751
Convex, Extrapolate	0.916719	0	0.00	0.004948
Convex, Bezier Fit	0.911797	0	0.00	0.019081

[3] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.

[4] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[5] S. E. Stein. An integrated method for spectrum extraction and compound identification from gas chromatography/mass spectrometry data. *Journal of the American Society for Mass Spectrometry*, 10(8):770 – 781, 1999.

TABLE 2

Random Spectra, sparsity = 0.2, $n = 3$
Spacing: 1
$A = 1, 2, 0.5$
$\sigma = 3, 3.2, 2.8$
100 iterations
Noise Fraction: 0

Algorithm	**Ave. Score**	**Perfect**	**Wins**	**Ave. Time**
AR	0.869267	0	1.00	0.052382
NNMF	0.898402	8	5.58	0.011567
Data Matrix (for control)	0.893941	0	0.00	0.000116
Peak Max (res 50, cut .04)	0.909777	0	5.00	1.105254
Peak Max (res 10, cut .1)	0.923012	0	6.00	1.068473
ConEx 2-param	0.893142	0	0.00	0.446189
ConEx 2-param, AR start	0.751962	0	1.00	0.507717
ConEx 2-param, NNMF start	0.820223	0	2.00	0.297234
ConEx 3-param	0.868796	0	3.00	0.641380
ConEx 3-param, AR start	0.864846	1	1.33	0.563144
ConEx 3-param, NNMF start	0.865431	7	4.08	0.457926
ConEx 2-param, special	0.893171	0	0.00	0.541275
ConEx 2-param, AR start, special	0.740053	0	2.00	0.568785
ConEx 2-param, NNMF start, special	0.892122	0	0.00	0.518609
ConEx 3-param, special	0.880560	0	0.00	0.756957
ConEx 3-param, AR start, special	0.863482	2	1.33	0.661193
ConEx 3-param, NNMF start, special	0.869676	0	0.00	0.742636
Direct Fit	0.888569	1	1.00	0.268436
Direct Fit AR	0.875666	12	9.00	0.310122
Direct Fit NNMF	0.926803	37	26.08	0.241175
Direct Fit 2-param AR	0.863364	3	3.50	0.229655
Direct Fit 2-param NNMF	0.929567	41	28.08	0.198312
Denizen 3 (one iteration)	0.764808	0	0.00	0.004900
Denizen 3 (50 iterations)	0.764724	0	0.00	0.030252
sDenizen 3 (one iteration)	0.760367	0	0.00	0.053685
sDenizen 3 (50 iterations)	0.760377	0	0.00	0.078946
Convex, Extrapolate	0.873789	0	0.00	0.005152
Convex, Bezier Fit	0.879771	0	0.00	0.017912

TABLE 3

Randomly selected spectra from NIST library, $n = 3$
Spacing: 1
$A = 1, 1.2, 0.8$
$\sigma = 3, 3.2, 2.8$
100 trials
Noise Fraction: 0

Algorithm	**Ave. Score**	**Perfect**	**Wins**	**Ave. Time**
AR	0.802218	0	0.00	0.040509
NNMF	0.811569	2	0.61	0.016489
Data Matrix (for control)	0.835624	0	0.00	0.000161
Peak Max (res 50, cut .04)	0.978118	82	22.00	2.213887
Peak Max (res 10, cut .1)	0.995378	89	24.45	2.161557
ConEx 2-param	0.787877	0	0.00	0.525585
ConEx 2-param, AR start	0.740140	0	0.00	0.405827
ConEx 2-param, NNMF start	0.786760	1	1.25	0.347952
ConEx 3-param	0.829842	0	0.00	0.558452
ConEx 3-param, AR start	0.760581	0	0.00	0.672346
ConEx 3-param, NNMF start	0.851573	6	2.14	0.386555
ConEx 2-param, special	0.789884	0	0.00	0.572529
ConEx 2-param, AR start, special	0.750165	2	0.50	0.539195
ConEx 2-param, NNMF start, special	0.789854	0	0.00	0.560172
ConEx 3-param, special	0.832925	0	0.00	0.655680
ConEx 3-param, AR start, special	0.776522	1	0.25	0.599574
ConEx 3-param, NNMF start, special	0.834720	0	0.00	0.655325
Direct Fit	0.800155	0	0.00	0.521194
Direct Fit AR	0.719209	4	0.75	0.479734
Direct Fit NNMF	0.834677	25	7.33	0.349959
Direct Fit 2-param AR	0.727503	3	0.71	0.343539
Direct Fit 2-param NNMF	0.822247	17	4.46	0.292264
Denizen 3 (one iteration)	0.980019	48	7.91	0.032801
Denizen 3 (50 iterations)	0.985876	65	13.36	0.579250
sDenizen 3 (one iteration)	0.977742	39	6.16	0.205430
sDenizen 3 (50 iterations)	0.981147	49	8.11	0.748797
Convex, Extrapolate	0.794396	0	0.00	0.008554
Convex, Bezier Fit	0.770582	0	0.00	0.020120

TABLE 4

Randomly selected spectra from NIST library, $n = 3$
Spacing: 1
$A = 1, 1.2, 0.8$
$\sigma = 3, 3.2, 2.8$
100 trials
Noise Fraction: 0.02

Algorithm	**Ave. Score**	**Perfect**	**Wins**	**Ave. Time**
AR	0.793741	0	0.00	0.060994
NNMF	0.792933	0	0.00	0.019706
Data Matrix (for control)	0.840709	0	0.00	0.000151
Peak Max (res 50, cut .04)	0.957216	30	17.00	2.961989
Peak Max (res 10, cut .1)	0.978449	48	48.17	2.973220
ConEx 2-param	0.784261	0	0.00	0.460602
ConEx 2-param, AR start	0.699160	0	0.00	0.467706
ConEx 2-param, NNMF start	0.715314	0	0.00	0.369713
ConEx 3-param	0.786243	0	0.00	0.578846
ConEx 3-param, AR start	0.749455	0	0.00	0.717692
ConEx 3-param, NNMF start	0.725740	0	0.00	0.650924
ConEx 2-param, special	0.790889	0	0.00	0.545876
ConEx 2-param, AR start, special	0.732991	0	0.00	0.539295
ConEx 2-param, NNMF start, special	0.793790	0	0.00	0.517430
ConEx 3-param, special	0.778571	0	0.00	0.602146
ConEx 3-param, AR start, special	0.745351	0	0.00	0.783509
ConEx 3-param, NNMF start, special	0.766181	0	0.00	0.643778
Direct Fit	0.765841	0	0.00	0.625145
Direct Fit AR	0.705419	0	0.00	0.542293
Direct Fit NNMF	0.762430	0	0.00	0.462369
Direct Fit 2-param AR	0.706536	0	0.00	0.383439
Direct Fit 2-param NNMF	0.765028	0	1.00	0.388841
Denizen 3 (one iteration)	0.973776	24	10.33	0.059611
Denizen 3 (50 iterations)	0.787321	0	0.00	0.613191
sDenizen 3 (one iteration)	0.975293	29	15.83	0.228601
sDenizen 3 (50 iterations)	0.975004	17	7.67	0.769318
Convex, Extrapolate	0.816637	0	0.00	0.008781
Convex, Bezier Fit	0.776020	0	0.00	0.107031

TABLE 5

Random Spectra, sparsity = 0.3, $n = 3$
Spacing: 0.3
$A = 1, 1.2, 0.8$
$\sigma = 3, 3.2, 2.8$
100 trials
Noise Fraction: 0

Algorithm	**Ave. Score**	**Perfect**	**Wins**	**Ave. Time**
AR	0.962262	38	6.46	0.049438
NNMF	0.950839	27	4.60	0.009180
Data Matrix (for control)	0.876044	0	0.00	0.000101
Peak Max (res 50, cut .04)	0.922922	0	0.00	1.045733
Peak Max (res 10, cut .1)	0.930116	0	0.00	1.016095
ConEx 2-param	0.896595	0	0.00	0.216574
ConEx 2-param, AR start	0.922804	0	0.00	0.663570
ConEx 2-param, NNMF start	0.910838	0	0.00	0.590871
ConEx 3-param	0.893699	37	5.73	0.457984
ConEx 3-param, AR start	0.927743	56	9.83	0.404140
ConEx 3-param, NNMF start	0.903325	59	11.09	0.373728
ConEx 2-param, special	0.897821	0	0.00	0.263426
ConEx 2-param, AR start, special	0.923525	0	0.00	0.698692
ConEx 2-param, NNMF start, special	0.895283	0	0.00	0.261475
ConEx 3-param, special	0.869778	34	5.80	0.525514
ConEx 3-param, AR start, special	0.910579	58	10.08	0.509074
ConEx 3-param, NNMF start, special	0.890915	36	6.13	0.475902
Direct Fit	0.841494	6	0.84	0.276488
Direct Fit AR	0.953247	44	7.91	0.194025
Direct Fit NNMF	0.971838	64	11.26	0.138605
Direct Fit 2-param AR	0.953066	41	7.04	0.169689
Direct Fit 2-param NNMF	0.975705	73	13.23	0.118872
Denizen 3 (one iteration)	0.694322	0	0.00	0.004576
Denizen 3 (50 iterations)	0.694322	0	0.00	0.029068
sDenizen 3 (one iteration)	0.702238	0	0.00	0.057344
sDenizen 3 (50 iterations)	0.702238	0	0.00	0.080963
Convex, Extrapolate	0.892109	0	0.00	0.005022
Convex, Bezier Fit	0.867346	0	0.00	0.013306