An algorithm to fit a simplex to a set of multidimensional points

PJ Wirapati TP Speed

January 15, 2002

1 Introduction

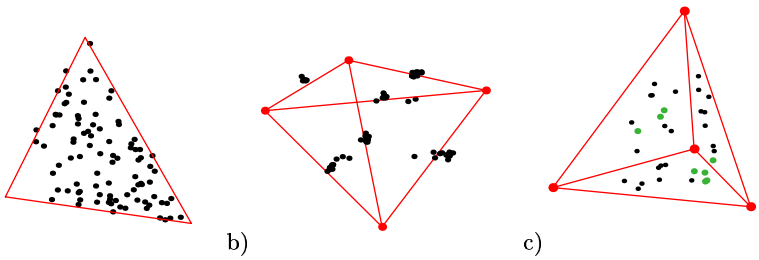
Let $\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_n \end{bmatrix}$ a set of points, $\boldsymbol{y}_j \in \mathbb{R}^m$. We would like to approximate them with a simplex $\boldsymbol{M} = \begin{bmatrix} \boldsymbol{\mu}_1 & \cdots & \boldsymbol{\mu}_k \end{bmatrix}$, where $\boldsymbol{\mu}_i \in \mathbb{R}^m$ is a vertex and the column vectors of $\boldsymbol{M}$ are affinely independent. Usually $m \gg k$, and at least $m = k - 1$ for $\boldsymbol{M}$ to be affinely independent.

The most natural approach is to approximate each data point by a convex combination of the vertices. Let $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_j \end{bmatrix}$ the coefficients of the convex combination of each point, $\boldsymbol{x}_j \in \mathbb{R}^k$. The problem can be considered as an (approximate) decomposition:

$$\boldsymbol{Y} \approx \boldsymbol{M}\,\boldsymbol{X} \qquad x_{ij} \geq 0,\ \sum_i x_{ij} = 1 \tag{1}$$

that minimizes the sum of, say, least-square distance from a point $\boldsymbol{y}_j$ to its nearest-point map $\boldsymbol{M}\,\boldsymbol{x}_j$.

Without additional constraints, any simplex large enough will minimize the errors. To make the solution less arbitrary, archetypal analysis (Cutler 1994) limits the vertices to be inside the convex hull of the data, making the solution more meaningful but may be inappropriate for some problems [e.g. Figure 1].



a) b) c)

Figure 1: Problems with missing vertices. (a) A set of point sampled from a triangle, but with an area around a vertex truncated. (b) A set of points sampled from the midpoint of the edges of a tetrahedron. (c) A set of points are sampled from the interior each facet of a tetrahedron. The vertex in the middle are "behind" the drawing plane, and the points from the facet closest to the drawing plane is colored green.

The "shrink-wrap" algorithm (Fuhrmann 1999) allows the vertices to be outside the convex hull, using the minimum simplex volume as the constraint. However, this consequently requires all points to be strictly inside the simplex, making it susceptible to noise and outliers. There are several other works somewhat related to simplex fitting (Renner 1993, Lee and Seung 1999, Parra 2000, Venet 2001), but they need to make assumptions on the vertices (such as non-negativity and sparseness, or known prior probability of $\boldsymbol{\mu}_i$).

In summary, the followings are the issues in simplex fitting that have only been partially addressed by existing algorithms:

1. We need to be able to handle cases where $m > k - 1$. This means simultaneously finding the best affine hull of $\boldsymbol{M}$ in $\mathbb{R}^m$ and the placement of the vertices within the affine hull. Some algorithms rely on separate preprocessing to make m equal $k - 1$, such as using PCA, which may discard some important information leaking to the unused subspace.

2. Robustness against noise and outliers. This means allowing some points to be outside the simplex, otherwise the solution may hinge on (literally speaking) a few bad points.

3. Because we are interested in the boundaries of the simplex, the algorithm should not assume too much about the distribution of points inside the simplex (compare Figure 1a, 1b, and 1c). We only need to specify the noise level, which corresponds to how fuzzy the boundary of the simplex and where we would like to draw the line.

4. There might be a lack of sparseness in $\boldsymbol{X}$, which pulls the points away from the vertices and edges toward the interior. Of course, for simplex-fitting to be meaningful, there should be some points lying on the facets[1] (thus having at least one zero-valued coefficient). However, only "weak sparseness" of the coefficients is needed[2]. If there are enough points on each facet to define its hyperplane, then the vertices are determined, as the intersections of the hyperplanes [see Figure 1c].

5. The simplex can be anywhere in the data space ($\mathbb{R}^m$), and not necessarily confined to the positive orthant. Of course, when desired, it should be possible to restrict the vertices to be within a bounded set. This is application-specific and should be achieved through a "plugged-in" gradient projection function, instead of a part of the core algorithm.

6. The computation should be as efficient as possible. Some of the existing algorithms requires computationally expensive non-negative or convex least-squares solution at each iteration cycle.

This report is organized as follows. In Section 2, the proposed algorithm is outlined. In Section 3, various examples using simulated data are shown to illustrate the behavior of the algorithm. Possible improvements to the algorithm and open questions are discussed in Section 4. Applications to real data are presented in a separate report. Our focus here is on proposing a fairly generic simplex-fitting algorithm.

[1] See Appendix A for the definition of a *facet*

[2] cf. "strong sparseness" required by some unsupervised learning methods, where most coefficients are zeroes, except one.

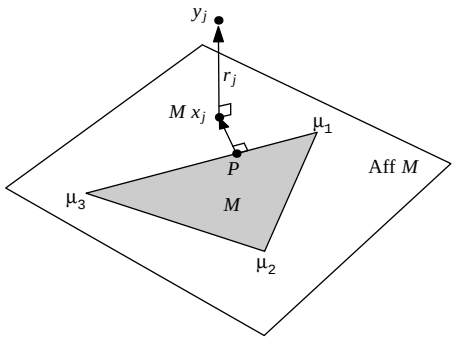


Figure 2: An illustration for equation 2

2 The Algorithm

Our algorithm is similar in spirit to k-mean clustering and mixture model fitting using EM algorithm. It is iterative, and involves "partitioning" each data point to each vertex. That is, a weight (analogous to the posterior probability in EM algorithm, but without formal probabilistic meaning) is assigned to every pair of point $\boldsymbol{y}_j$ and vertex $\boldsymbol{\mu}_i$. Based on the weighted data points, each vertex is updated independently. However, instead of maximizing the vertices toward a central location (e.g., the mean), we "extremize" them outward. To have a notion of "outward" and "inward", we perform the operations in the coefficient world ($\mathbb{R}^k$), where the signs of the coefficients indicates the inside-outside polarity. Briefly, the algorithm consists of three main steps: regression, expectation-like, and "extremization" steps.

Regression step Let's reformulate the approximation as follows:

$$\boldsymbol{Y} = \boldsymbol{M}\,\boldsymbol{X} + \boldsymbol{R} \qquad \sum_i x_{ij} = 1 \tag{2}$$

where $\boldsymbol{R} = \begin{bmatrix}\boldsymbol{r}_1 & \cdots & \boldsymbol{r}_j\end{bmatrix}$, $\boldsymbol{r}_j \in \mathbb{R}^m$, are the residual vectors normal to affine hull of $\boldsymbol{M}$ (see Figure 2). Notice that we relax the non-negativity condition. As we will see, this is fine if what we want to do is to optimize the vertices. The negative coefficients are in fact the main driving force of our algorithm. If the convex solution is required (the point P on Figure 2), it can always be done separately after the best-fit simplex is found.

To find the affine coefficients $\boldsymbol{x}_j$, an arbitrary vertex can be chosen as the origin (say $\boldsymbol{\mu}_i$). Let $\boldsymbol{y}_j^i = \boldsymbol{y}_j - \boldsymbol{\mu}_i$ and $\boldsymbol{M}^i = \begin{bmatrix}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_i & \cdots & \boldsymbol{\mu}_k - \boldsymbol{\mu}_i\end{bmatrix}$. We simply solve the linear equation:

$$\min_{\boldsymbol{x}_j} \|\boldsymbol{y}_j^i - \boldsymbol{M}^i\,\boldsymbol{x}_j\|_2 \ . \tag{3}$$

Obviously the i-th column of $\boldsymbol{M}$ is $\boldsymbol{0}$ and ignored. The corresponding coefficient is set to:

$$x_{ij} = 1 - \sum_{h \neq i} x_{hj} \tag{4}$$

so that the affine condition is satisfied. Note that the answer is the same regardless of which vertex is chosen as the origin.

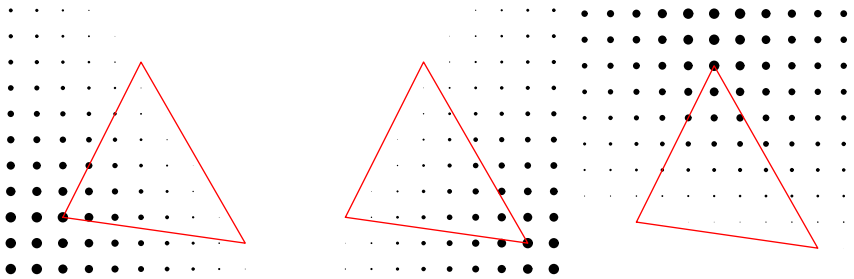


Figure 3: The weight τ_{ij} as a function of $\boldsymbol{y}_j$, computed using equation 6 on the affine coordinate $\boldsymbol{x}_j$. The size of the dots is proportional to τ_{ij}. The three panels correspond to the the lower-left, lower-right and top vertices, respectively.

Expectation-like step Here, each point is partitioned to all vertices by assigning a weight $\tau_{ij} \geq 0$, $\sum_i \tau_{ij} = 1$ to each pair of data point j and vertex i. The closer a point j to a vertex i, the higher the weight should be. The proximity is based on the coefficient x_{ij}, which corresponds to the location of the point along an axis normal to the facet opposite the vertex. The weight can be computed as follows:

$$\tau_{ij} = \frac{\phi(x_{ij})}{\sum_{h=0}^{k} \phi(x_{hj})} \tag{5}$$

where $\phi(x)$ is a monotone increasing and nonnegative function. We will use a simple one:

$$\phi(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x < 1 \\ 1 & x \geq 1 \end{cases} \tag{6}$$

which works well for a wide range of problems[3]. Figure 3 illustrates τ_{ij} as a function of the location in the affine hull.

Extremization step Each vertex is updated by a sum of two orthogonal gradients: (i) perpendicular to the affine hull, as the function of the residual $\boldsymbol{r}_j$, and (ii) within the affine hull, as the function of the affine coefficients:

$$\Delta\boldsymbol{\mu}_i = \Delta\boldsymbol{\mu}_i^r + \Delta\boldsymbol{\mu}_i^x \ . \tag{7}$$

The first term is quite simple. Let $w_1, \ldots, w_n$ the weights of the points (given as input; or set to $1/n$ if not available). It's reasonable to require that at convergence the following is locally minimized (for each vertex i):

$$\sum_{j=1}^{n} \tau_{ij}\, w_j\, \|\boldsymbol{r}_j\|_2^2 \ . \tag{8}$$

[3]It is possible to use smoother function such as sigmoidal (logistic) functions. In general, any measure of "grades of membership" can be used for τ_{ij}.

At the fixed point, this should hold:

$$\sum_{j=1}^{n} \tau_{ij}\, w_j\, \boldsymbol{r}_j = \mathbf{0}\ , \tag{9}$$

and hence, this updating formula:

$$\Delta\boldsymbol{\mu}_i^r = \frac{\sum_{j=1}^{n} \tau_{ij}\, w_j\, \boldsymbol{r}_j}{\sum_{j=1}^{n} \tau_{ij} w_j}\ . \tag{10}$$

Before defining the gradient within the affine hull, $\Delta\boldsymbol{\mu}_i^x$, we first look at a geometric interpretation of the affine coefficients x_{ij}. Consider any vertex $\boldsymbol{\mu}_i$ as the origin of a coordinate system, with the edges $\boldsymbol{\mu}_h - \boldsymbol{\mu}_i$, $h \neq i$, as the basis vectors. See Figure 4a, each point can be represented as:

$$\boldsymbol{M}\, \boldsymbol{x}_j = \boldsymbol{\mu}_i + \sum_{h \neq i} x_{hj}(\boldsymbol{\mu}_h - \boldsymbol{\mu}_i)\ . \tag{11}$$

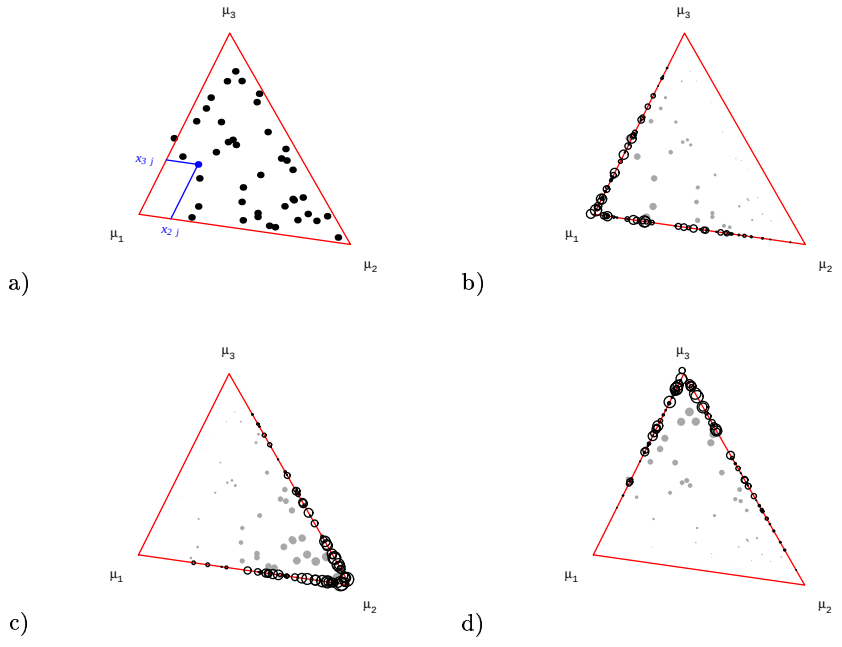


Figure 4: Decomposition of the data points w.r.t. each vertex. (a) The original data points. A point is decomposed w.r.t $\boldsymbol{\mu}_1$ into x_{2j} and x_{3j} (blue lines). (b) Decomposition of all points w.r.t. μ_1. The gray dots are the original points weighted by τ_{ij} (shown by the size of the dots). The black circles are $\boldsymbol{\mu}_1 + x_{hj}(\boldsymbol{\mu}_h - \boldsymbol{\mu}_1)$, for $h = 2$ and $h = 3$, along the respective edges. Similar pictures for $\boldsymbol{\mu}_2$ and $\boldsymbol{\mu}_3$ are shown in (c) and (d), respectively.

Thus, in the coefficient space, x_{hj} is the projection of the point to the edges incident on $\boldsymbol{\mu}_i$. Notice in Figure 4b how the projections along both axes are distributed. There is a 'mass' of decomposed points around $\boldsymbol{\mu}_1$ although there are no points sampled around it in the original data (Figure 4a). This is the heart of our "extremization" procedure:

> The optimal vertex corresponds to lower quantiles of the decomposed points along the incident edges.

This is how, we think, problems exhibiting "weak sparseness" can be solved.

Let θ_{hi} an extreme location statistics along the $\boldsymbol{\mu}_h - \boldsymbol{\mu}_i$ axis for a given $\boldsymbol{M}$ (which might not be the optimal simplex). The location of the vertex can be improved by:

$$\Delta\boldsymbol{\mu}_i^x = \sum_{h \neq i} \theta_{hi}(\boldsymbol{\mu}_h - \boldsymbol{\mu}_i) \tag{12}$$

It's convenient to consider θ_{hi} as a component of a vector $\boldsymbol{\theta}_i \in \mathbb{R}^k$. If we set $\theta_{ii} = 1 - \sum_{h \neq i} \theta_{hi}$, then $\boldsymbol{\theta}_i$ becomes the affine coordinate of $\Delta\boldsymbol{\mu}_i^x + \boldsymbol{\mu}_i$. The new vertex is then:

$$\boldsymbol{\mu}_i^{\text{new}} = \Delta\boldsymbol{\mu}_i^r + \boldsymbol{M}\,\boldsymbol{\theta}_i \tag{13}$$

This completes one cycle of the iterative algorithm. Note that when a problem-specific constraints need to be imposed on $\boldsymbol{\mu}_i$, a gradient projection function can be inserted here. It should return a "chopped" gradient given $\boldsymbol{\mu}_i^{\text{new}}$ and the current $\boldsymbol{\mu}_i$. Obviously, the initial simplex has to be in the feasible set.

The choice of extreme location statistics There are many ways to specify θ_{hi}, that corresponds more or less to the lower boundaries of the decomposed points in Figure 4. Any quantile-like statistics might be used, with a parameter α correspond roughly to a location along the empirical distribution (for empirical quantile, θ_{hi} is the αth quantile). Note that we are not trying to estimate a "true" parameter in a formal way, or to have a probabilistic interpretation of θ_{hi}, although it should correspond somehow to the noise variance (in the noiseless case, $\theta_{hi} = \min_j x_{hj}$ at convergence).

We choose an M-estimator because it can be computed without sorting, and the influence function can be flexibly modified to achieve a particular robustness. For each distinct (ordered) pairs i and h, we want to minimize:

$$\sum_{j=1}^{n} \tau_{ij}\, w_j\, \rho_\alpha(x_{hj} - \theta_{hi}) \tag{14}$$

where ρ_α is an asymmetric cost function. α determines how extreme we want to be. The interpretation of θ_{hi} depends on the choice of ρ and the sample distribution (Green and Kozek 2001).

A simple choice for ρ is an asymmetric least-square estimator called "expectile" (Newey and Powell 1987), defined as:

$$\rho_\alpha(t) = |\alpha - \mathbf{1}_{\{x \leq 0\}}|t^2 = \begin{cases} (1-\alpha)t^2 & x \leq 0 \\ \alpha t^2 & x > 0 \end{cases} \qquad 0 < \alpha < 1\,. \tag{15}$$

Equation 14 can be minimized by finding θ_{hi} that satisfies:

$$\sum_{j=1}^{n} \tau_{ij}\, w_j\, \psi_\alpha(x_{hj} - \theta_{hi}) = 0 \tag{16}$$

where

$$\psi_\alpha(x-\theta) = |\alpha - \mathbf{1}_{\{x \le \theta\}}|(x-\theta) \quad \propto \quad \partial\rho_\alpha(x-\theta)/\partial\theta\ . \tag{17}$$

Obviously, smaller α means the balance is shifted further to the negative direction. Its precise meaning is difficult and depends on the distribution of the points (Abdous and Remillard 1995). For our purpose, it is a parameter to be tuned in an *ad hoc* way for each problem domain.

The solution can be found using iterative reweighting (Huber 1981). However, in our case we want to couple the M-estimator iteration with the overall algorithm. We think it's better not to iterate until convergence for a given $\boldsymbol{M}$, because θ_{hi} is conditional on the current values of τ_{ij} (a function of $\boldsymbol{M}$, which in turn, a function of previous θ_{hi}). Instead, only one update is performed. This allows the weights τ_{ij} to be readjusted first by the affine regression and partitioning step, possibly leading to smoother and faster convergence (or at least cheaper computation). There are other advantages: (i) we don't have to worry about the scale; x_{hj} are always relative to the length of the edges of the simplex, and (ii) the current θ_{hi} is always zero, because the new vertex becomes the new origin.

With a slight abuse of notation, we redefine θ_{hi} to be the next estimate (instead of the solution of [16]). The update can then be computed as:

$$\theta_{hi} = \frac{\sum_{j=1}^{n} \tau_{ij}\, w_j\, |\alpha - \mathbf{1}_{\{x \le 0\}}|\, x_{hj}}{\sum_{j=1}^{n} \tau_{ij}\, w_j\, |\alpha - \mathbf{1}_{\{x \le 0\}}|}\ . \tag{18}$$

At the fixed point, $\theta_{hi} = 0$.

The effect of α on the convergence We found that the convergence is problematic for small α. The positive coefficients contribute very little to the gradient, and the simplex expanded quickly outward. If the current simplex is not correctly oriented, the improved simplex will enclose all points, but incorrectly oriented (see Figure 5). When α is larger, the mass of points in the optimal direction can pull the vertices and twist the simplex into the correct orientation. However, we often want to have a small α to fully extremize the simplex (e.g. when we are sure that the noise level is small).

Our solution is to start with a relatively large α, iterate until convergence (that is, $\|\Delta\boldsymbol{M}\|_F \approx 0$) for that value of α, and then gradually decreasing the value of α until the desired α is reached, iterating until convergence at each given value. In practice, it's sufficient to start with $\alpha = 0.5$ and then decreasing it exponentially in 5 steps or so to the final value. This directs the simplex to grow from the inside, and roughly orients the simplex before pushing it to the boundaries. Note that when we use the expectile (equ. 15), $\alpha = 0.5$ corresponds to the mean.

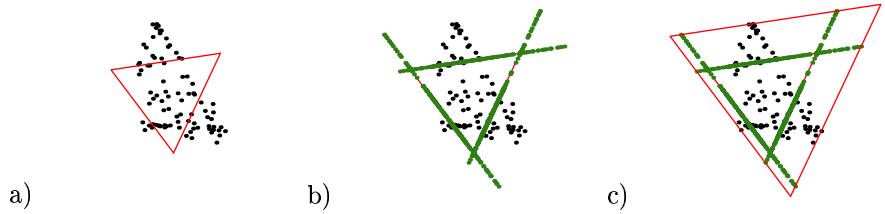


Figure 5: The effect of very small α on the convergence. (a) A cloud of points to be fitted, the red triange is the current simplex, (b) the green dots are the decomposed points based on the current simplex, (c) the new simplex resulting from using $\theta_{hi} = \min_j x_{hj}$ (or $\alpha \to 0$). This new simplex can not be improved in the next cycle.

Let us summarize the algorithm, incorporating the descending α. It has an outer loop for the "annealing schedule" of α, and the inner loop which contains the core steps.

Algorithm SXFIT

Start with an initial simplex

Iterate with decreasing values of $\alpha^{(0)}, \ldots, \alpha^{\text{final}}$:

Iterate until $\|\Delta \boldsymbol{M}_\alpha\|_F \approx 0$:

- R-step: find the affine coefficients
- E-step: partition each point to all vertices
- X-step: extremize the location of the vertices

2.1 Implementation

The algorithm is fairly straightforward and simple to program. The R-step needs safeguards against a collapse in the rank of $\boldsymbol{M}^i$. We use Householder QR decomposition with pivoting (Golub and Van Loan 1996, Algorithm 5.4.1), where the orthogonal vectors are sorted according to their norms. If the ratio of $\|q_i\|_2/\|q_1\|_2$ drops below a certain tolerance, the corresponding coefficients x_{ij} are zeroed, and the vertex i is considered indeterminate (otherwise all weighted average formulas crash due to $\sum_j \tau_{ij} = 0$). $\boldsymbol{\mu}_i$ is not updated in the X-step, but it is not completely removed, in case the rank of $\boldsymbol{M}$ is regained in the following cycles (i.e. if the collapse of the rank is a transient effect of a particular iteration pathway).

If some vertices are still indeterminate after convergence, they are ignored and we consider ourself overspecifying the number of vertices k. This allows us to partially solve the problem of choosing the right k. We may still overfit (due to noise), but not too much. Note that unlike PCA where the non-null basis can be used immediately, we need to rerun the algorithm with smaller value of k (because the basis vectors are not 'nested').

In general, Householder QR is not the cheapest way to solve linear least-square problem. However, in our case usually the number of data points is large (say, several thousands) and k is small. Most of the computation time is on solving $\boldsymbol{R}\boldsymbol{x}_j = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y}_j$, which is even faster than the normal equation method once we have $\boldsymbol{Q}$ and $\boldsymbol{R}$. The decomposition is roughly an mk^2 algorithm. Solving n points requires $n(km+k^2)$. The E-step and X-step are nk and $nmk+nk^2+mk$, respectively.

The numerical stability of Householder QR allows decent computation with faster single-precision floating points, which is reasonable for many problems where the raw data points are small integers coming from analog-to-digital converters. We have implemented the algorithm as a C-library module, suitable for inclusion in high-level analysis software such as R, or a standalone UNIX pipe.

2.2 Choosing an initial simplex

These are some of the requirements for a good initial value:

1. The rank of $\boldsymbol{M}^{(0)}$ has to be $k-1$. Choosing k observations randomly is a bad idea.
2. The vertices should be as mutually distant as possible.
3. If the vertices are already in the affine hull of the data, the convergence might be faster. It is good if they are within the convex hull.
4. When there are constraints on the feasible solution (e.g. nonnegativity), the initial simplex has to be in the feasible set.
5. Choosing the simplex should not be computationally too demanding.

One possible way to choose an initial simplex is to use Householder QR with pivoting on the data, with a slight modification: only the first $k-1$ orthogonal vectors need to be constructed.

First, find a point whose distance from the mean is the largest. Use this as a vertex of the initial simplex:

$$\boldsymbol{\mu}_1 = \arg\max_{\boldsymbol{y}_j} w_j \|\boldsymbol{y}_j - \bar{\boldsymbol{y}}\|_2^2 \,. \tag{19}$$

[w_j is an optional point weight, if available.]

Let $\boldsymbol{Y}' = \begin{bmatrix} w_1(\boldsymbol{y}_1 - \boldsymbol{\mu}_1) & \cdots & w_n(\boldsymbol{y}_n - \boldsymbol{\mu}_1) \end{bmatrix}$ and perform a QR decomposition with pivoting to find the first $k-1$ basis of $\boldsymbol{Y}'$. We need neither $\boldsymbol{Q}$ nor $\boldsymbol{R}$, but only the permutation Π. Let $\Pi_1, \Pi_2, \ldots, \Pi_k$ a set of indices such that $\|\boldsymbol{q}_{\Pi_1}\|_2 \geq \|\boldsymbol{q}_{\Pi_3}\|_2 \geq \ldots \geq \|\boldsymbol{q}_{\Pi_{k-1}}\|_2$. The corresponding data points $\boldsymbol{y}_{\Pi_1}, \ldots, \boldsymbol{y}_{\Pi_{k-1}}$ are used as $\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$. We have to watch $\|q_i\|_2$ and reduce k if $\|q_i\|$ becomes to small.

> A conjecture: when there is no noise and all vertices are present in the data, this procedure finds the true $\boldsymbol{M}$.

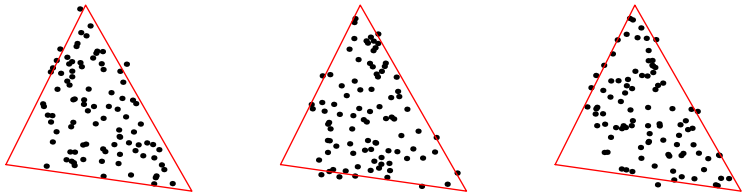


Figure 6: Examples of three data sets generated using Eq. 20. 100 points were sampled, and those located around the lower-left vertex were removed. The noise scale is $\sigma = 0.015$ (the average length of the edges is 0.72), which cause some points to be slightly outside the simplex.

3 Examples on simulated data

3.1 Example 1: Convergence

The data points in this example are in $\mathbb{R}^2$, with $k = 3$. Thus, $m = k - 1$ (not an overdetermined case). The points are generated in the following way:

$$\begin{aligned} &\boldsymbol{y}_j = \boldsymbol{M}\boldsymbol{x}_j + \boldsymbol{\epsilon}_j\,, \qquad \boldsymbol{x}_j = \frac{\boldsymbol{z}_j}{\|\boldsymbol{z}_j\|_1} \\ &z_{ij} \sim \text{Exponential}(1)\,, \quad \epsilon \sim N(0, \sigma) \\ &\boldsymbol{M} = \begin{bmatrix} 0.2 & 0.9 & 0.5 \\ 0.2 & 0.1 & 0.8 \end{bmatrix} \end{aligned} \tag{20}$$

To check the ability to extrapolate outside the convex hull of the data, the points near one vertex are removed (those which are less than a certain Euclidean distance from the vertex). Figure 6 illustrates several samples generate from the model.

First, we compare the convergence from various initial simplices, with α fixed (see Figure 7). At least in this trivial data set, all converges to the same simplex. The final vertices approximates the "true" vertices, including the one located outside the convex hull of the data.

The convergence is faster if the initial simplex has an approximately correct orientation (that is, the vertices are pointing in the same general direction with the true vertices). The convergence is worst if the initial simplex is "inverted" (Figure 7b). Initially, the simplex expand quickly to enclose all points without adjusting the orientation. Very slow adjustment is done later by slight twisting. Figure 7c shows that our method for choosing an initial simplex using Householder QR with pivoting (Section 2.2) works well to find one with a good orientation and size.

The virtue of using descending α is shown in Figure 8. It's clear that it is easier for the algorithm to "negotiate" the simplex orientation at a relatively high value of α, in the interior of the data points. Once good orientation is achieved, reducing α expands the triangle without too much twisting, in less number of iterations.

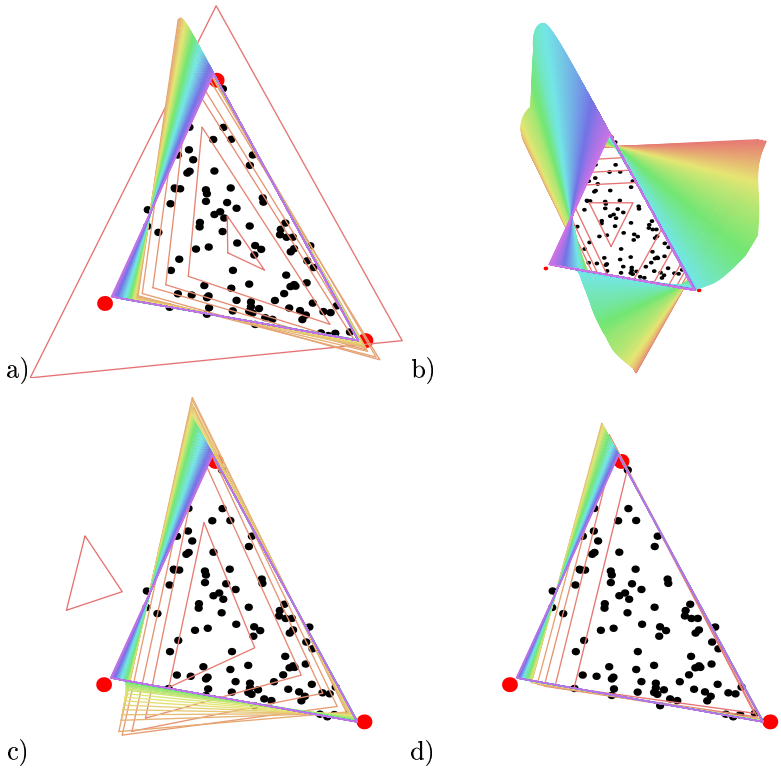


Figure 7: The convergence from various initial simplices. The true vertices are shown as large red dots. The improved simplices from all iteration cycles are drawn as colored triangles, in rainbow coding (red for early cycles, turning to yellow, green, blue and purple progressively). $\alpha = 0.001$ and it is fixed (not descending). The convergence tolerance is $\|\Delta \boldsymbol{M}\|_F / \|\boldsymbol{M}\|_F \leq 0.001$. (a) The largest triangle is the initial value. It converges in 61 steps. Note that its orientation is approximately correct. (b) The small triangle inside is the initial value, the orientation is the opposite of the true simplex (converges in 325 steps). The drawing is zoomed out to show the trajectories of the vertices. (c) The initial simplex does not intersect the true simplex or the convex hull of the data points, but oriented similarly (converges in 51 steps) (d) The initial vertices are three data points selected using QR with pivoting method (converges in 34 steps).

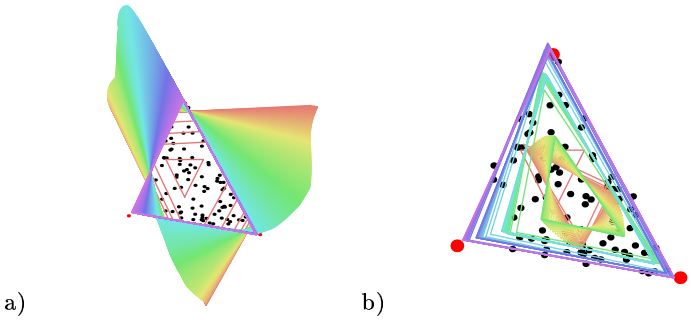


Figure 8: Comparison between fixed and descending α. a) The same example with that in Figure 7c. Convergence at $\alpha = 0.001$ is achieved in 325 steps. b) Convergence to the same simplex from the same initial values in 69 steps using a sequence of $\alpha = 0.4, 0.054, 0.007, 0.001$. The convergence at a given value of α is visible from the closeness of some successive triangles.

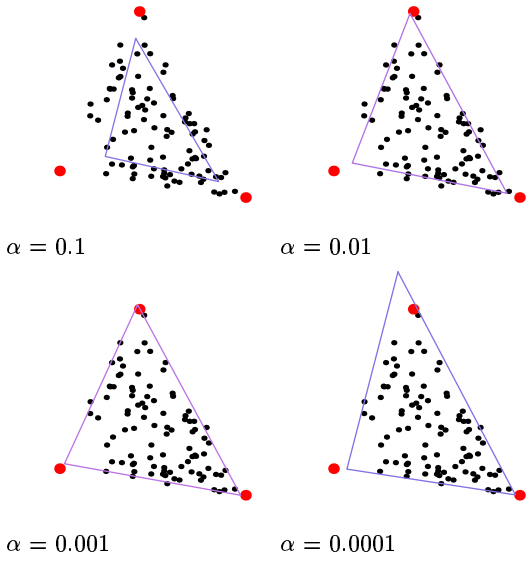


Figure 9: The effect of α on the extremeness of the final simplex. The same data set and parameters is used as the previous example (Figure 7). Only the simplex from the last iteration cycle is shown. The initial values are determined using the QR method.

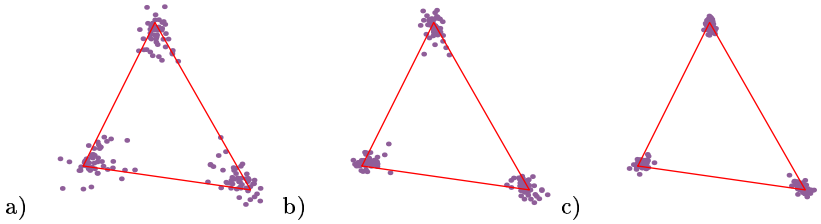


Figure 10: The effect of n on the estimates. a) $n = 20$, b) $n = 50$, and c) $n = 100$. In each case, the data set is generated 50 times, and a simplex is fitted with $\alpha = 0.001$. The true simplex is shown in red triangle. The purple dots are the vertices found in replicate experiments. The noise is $\sigma = 0.015$.

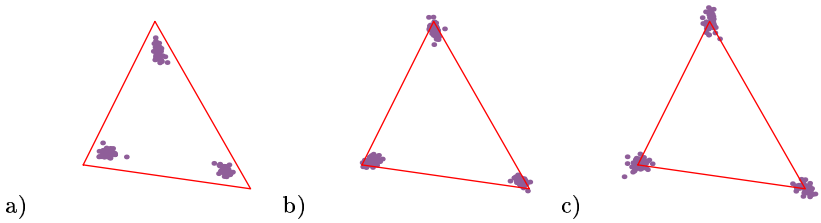


Figure 11: The effect of α on the estimates. The values of α are 0.1 (panel a), 0.01 (panel b), and 0.001 (panel c). $n = 100$, other parameters are same as in Figure 10.

Next, we show the effect final of value of α (Figure 9). As expected, the smaller α, the more extreme the final simplex. Although decreasing α seems to approximate the simplex better, having it too low ($\alpha = 0.0001$) results in sensitivity to the points on the boundaries (Figure 9d). Choosing the appropriate α automatically is a problem still being investigated.

To look at the effect of sampling, we compare the solutions of different realization of Eq. 20 (except that the lower-left corner is not truncated). Figure 10 shows the effect of the sample size n. As expected, the larger n the smaller the variance of the estimates. Figure 11 shows the effect of α. Larger α pulls the estimates vertices toward the interior. Figure 12 shows the effect of the noise variance. When σ is larger, the estimated simplices are larger than the true one (in addition having larger variance). This is understandable. The additive noise convolves the simplex's density. This means α has to be tuned to σ (see Figure 12c, where the results of having α too low and too high are shown). The systematic way to do this is, again, an open question.

What if we try to fit a triangle when the data points do not come a triangle? Figure 13a shows points generated from an equilateral hexagon, and we run the algorithm with $k = 3$. (This is the maximum for $\mathbb{R}^2$, otherwise the affine solution can not be uniquely determined by unconstrained least squares[4]). As expected, there are two possible equally likely solutions, corresponding to triangles that

[4]It might be possible to extend the algorithm to rank-deficient cases, using convex least squares and, for the points inside, minimum norm solutions.

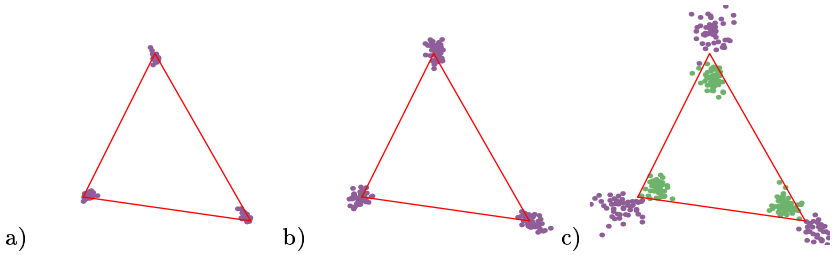


Figure 12: The effect of noise on the estimates. The values of σ are 0 (panel a), 0.02 (panel b), and 0.05 (panel c, purple dots) [Note that the average edge length is 0.72], $n = 100$ and $\alpha = 0.001$. The green dots in panel c are the estimates with $\alpha = 0.1$.

fit three non-adjacent faces of the hexagon. Note that in each sample, a single initial value is used (automatically picked by QR with pivoting). It is interesting to see if both answers are possible for a given sample when different initial values are used (i.e. two local minima).

Finally, a mandatory test for Gaussian data is shown in Figure 13b. It shows the results of the old saying: "trying to force a square peg (or a triangular one) into a round hole".

> Note that in archetypal analysis (Cutler 1994), it is possible to fit a truncated bivariate Gaussian points with a polytope with $k = 4$ (not a simplex), and the vertices tend to be located at the extreme ends of the axes (positive and negative directions of the eigenvectors). This is possible because it uses convex least squares, which can deal with rank-deficient situations.

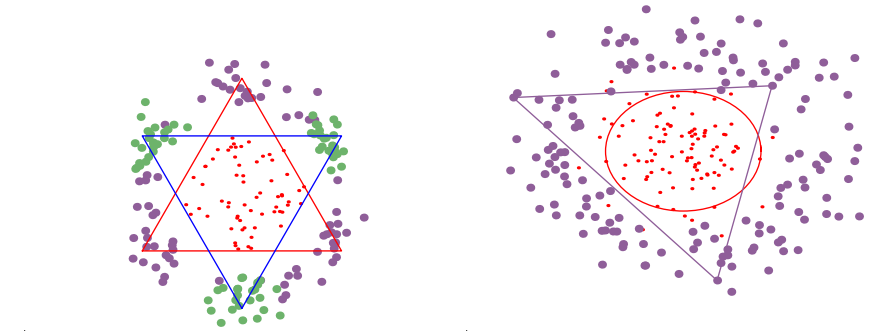
3.2 Example 2: "Weak sparseness"

In the introduction, we conjectured that if there are enough points on each facet of a simplex, then the vertices might be determined, eventhough there are no points sampled near the vertices, edges, or faces with dimensionality lower than the facet.

The problem here is what exactly 'enough points' means. The necessary condition is, of course, there has to be at least $k-1$ affinely independent points on each facet. However, in noisy situations, estimating a hyperplane reliably might require more points. Furthermore, in the absence of information about which point comes from which facet, there is another question whether our algorithm can find the simplex.

We investigate this using the following model. Points are generated from the interior of the facets of a tetrahedron in $\mathbb{R}^3$. They are restricted to be at a certain distance from the edges, by requiring the minimum coefficients (w.r.t the three vertices) to be at least 0.1. One coefficient (corresponding to the vertex not in the facet) is set to zero. A typical data set is shown in Figure 14a, which can be fitted fairly well (Figure 14b). Note that the area around the vertices are devoid of points.

The expected behaviors under replicate samplings are shown in Figure 15, with various numbers of points per facets. It is indeed necessary to have many more points than $k-1$ to have reliable estimates. This is probably because the



a) b)

Figure 13: a) The vertices found when fitting a triangle to a point set sampled from a hexagon, which is the intersection between the red and blue equilateral triangle. The red dots are one realization of the data points. There are 50 resampling, and the results are classified into those that fits the red triangle (purple dots) or the blue triangle (green dots). Either solution is found equally likely. b) The vertices found when fitting a Gaussian data with $\sigma_x = 0.3$ and $\sigma_y = 0.15$. The radii of the ellipse are 2σ in each respective direction. The red dots are one realization, fitted by the purple triangle. The other purple dots are vertices from 50 replicate sampling. $\alpha = 0.01$; higher (or lower α) pulls toward (or pushes away from) the center (results not shown).

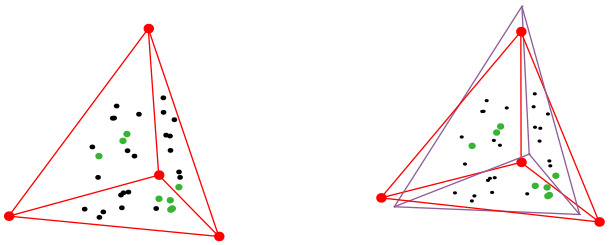
effective dimensionality of $k-1$ points randomly sampled from a hyperplane might be less than $k-2$, if they are too close to rank-deficient. Figure 16 shows the effect of various noise level. Increasing the noise quickly degrades the performance. This might be because the (implicitly) estimated supporting hyperplanes of the facets are sensitive to perturbation in their constituent points.

3.3 Example 3: Overdetermined cases ($m \gg k$)

In the previous examples, $m = k-1$ and thus $\boldsymbol{r}_j = \boldsymbol{0}$ (all points fully described by $\boldsymbol{M}\boldsymbol{x}_j$). Here we test a case where $m \gg k$. We use the same data generated in Example 2, but embedded in $\mathbb{R}^{100}$ using an arbitrary orthogonal vectors, created by performing QR decomposition on a random 100×3 matrix, and use the matrix $\boldsymbol{Q}$ to transform the original data. Symmetric Gaussian noise is then added in this $\mathbb{R}^{100}$ space. The result is shown in Figure 17 (for drawing purpose, the vertices are projected back to $\mathbb{R}^3$ using $\boldsymbol{Q}^{\mathrm{T}}$). Visually, there is no significant difference between the original problem and the one embedded $\mathbb{R}^{100}$, although the noise is added in the larger observation space.

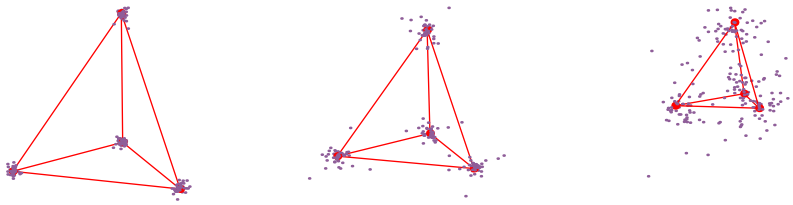
Using PCA to reduce the dimensionality of the data to $\mathbb{R}^3$, and perform the fitting there, seems to be as effective (Figure 17c). The average distance between the estimated vertices and the true one is similar between direct fitting in $\mathbb{R}^{100}$ and fitting in PCA world. It is not clear yet whether there are cases where reducing the dimensionality using PCA is better or worse than direct fitting. In this particular data set, the noise is relatively low and the 3-dimensional subspace of the tetrahedron should be represented faithfully by the first three eigenvectors.

The computation might be faster if dimensionality reduction is performed



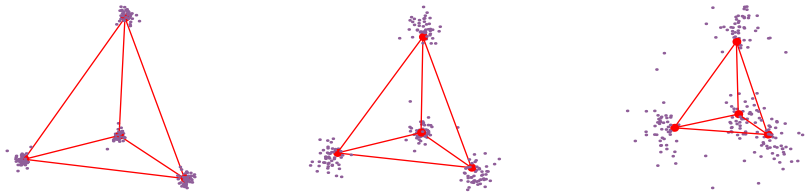
a) b)

Figure 14: a) The red tetrahedron is the true simplex. The vertex in the middle is "behind" the drawing plane. Eight points are sampled from each facet (without noise). The green dots are points on the facet closest to the drawing plane. b) A fitted tetrahedron is shown in purple, with $\alpha = 0.005$.



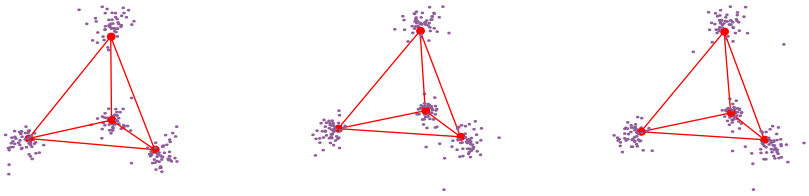
a) b) c)

Figure 15: The effect of the number of points sampled from each facet. The red tetrahedron is the true simplex. The purple dots are the vertices fitted to 50 replicate data set. $\alpha = 0.005$, with no noise. a) 20 points are sampled from each facet. b) 12 points, and c) 5 points.



a) b) c)

Figure 16: The effect of noise. 20 points per facet are used. a) $\sigma = 0.01$. b) $\sigma = 0.025$, and c) $\sigma = 0.05$. Note that the average edge length is 0.95.



a) b) c)

Figure 17: Overdetermined fitting. 20 points per facet are used, $\sigma = 0.05$, $\alpha = 0.005$. a) the original problem in $\mathbb{R}^3$. b) the problem is transformed to $\mathbb{R}^{100}$ by 3 random orthogonal basis vectors, with the noise added in $\mathbb{R}^{100}$. c) same data as b) but fitting is done in the PCA world, spanned by the first 3 eigenvectors.

first, especially when the original m is much larger than k. If there is a worry about important signals leaking to the discarded subspace, more dimensions than $k-1$ can be chosen (but still much less than m). Our algorithm can be combined easily with various dimensionality reduction methods because it does not require $m = k-1$ and the vertices can be anywhere in the input space (not confined to, say, the positive orthant).

3.4 Example 4: Outliers

To see the effect of outliers, we return to the triangle-in-$\mathbb{R}^2$ toy problem. Uniformly generated points are added to the data set, with certain frequency. One example is shown in Figure 18a. The algorithm is expectedly sensitive to points far outside (Figure 18b). This is obvious from Eq. 15, where the square of the negative errors dominate, unless a large value of α is chosen. However, our choice of finding extreme locations using M-estimator allows flexible robustification.

For example, we can modify ψ_α (defined by Eq. 17) to be as follows:

$$\psi_{\alpha,\beta}(x) = \begin{cases} (1-\alpha)\beta & x < \beta \\ (1-\alpha)x & \beta \le x \le 0 \\ \alpha x & x > 0 \end{cases} \tag{21}$$

which basically clamps the value of x if it becomes too negative. The results in Figure 18c is found using $\beta = -3\,\alpha$. The rationale for parameterizing β as a multiple of α is to make it adaptive to the descending values of α throughout the iteration. This way more points are included in the early stage and onle "pruned" later on, when the simplex is already near the boundaries.

There is no reason why the factor of 3 is chosen, other than that it works for this particular example. Admittedly, this is still an anecdotal example, presented to demonstrate the flexibility of the algorithm. More research is necessary to find the best robust function, how to conveniently parameterize it, and how it interacts with the descending α (or if it needs to interact at all).

4 Discussion

In summary, the proposed algorithm seems to be a powerful and generic simplex-fitting method, addressing the issues outlined in the introduction. Of particular

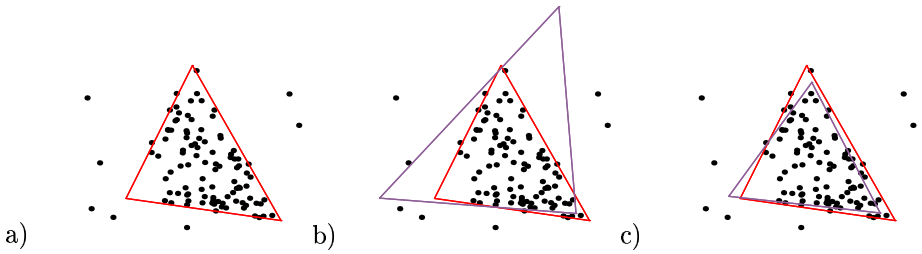


Figure 18: Potential robustification of the algorithm. a) A data set with outliers. The red triangle is the true simplex. b) The purple triangle is a simplex found by unmodified algorithm, with $\alpha = 0.02$. c) The simplex found using a slight modification of the function ψ_α (see text).

importance is its ability to handle data exhibiting "weak sparseness" and the potential robustification.

This is still an early stage in its development. Many questions are still unanswered. We have shown only simulated data set in reports, so that a particular feature of the algorithm can precisely illustrated. Several applications to real data have been tried (with larger k, m and n), and will reported elsewhere.

The following subsections are some of the questions we are currently investigating (in addition to those already mentioned in the examples).

4.1 Is there a better way to specify the desired extremeness?

Let $\boldsymbol{X}^-$ the matrix derived from $\boldsymbol{X}$ where all positive values are zeroed. Recent experiments (still being conducted) suggest that $\|\boldsymbol{X}^-\|_F$ is minimized for given a α. Furthermore, when α is lowered, the minimum value is also reduced. This makes sense considering the loss function $\rho_\alpha(x_{ij}) = |\alpha - \mathbf{1}_{\{x_{ij} \leq 0\}}| \, x_{ij}^2$.

We may therefore define:

$$\sigma = \frac{1}{\sqrt{n}} \|\boldsymbol{X}^-\|_F$$

and interpret σ as the "standard deviation" of the points outside the simplex. This is intuitive because we can consider it as the noise level relative to the simplex size (or maybe the average edge length). It should be resistant to the variation in point density inside the simplex (whereas α might be sensitive).

To achieve a desired value of σ, the outer loop of the algorithm needs to be modified. It is still iterating over α, but the changes should be based on the current value of σ, changing α up or down depending on the difference between the current and desired σ.

4.2 What is the algorithm minimizing?

We have shown that the algorithm seek to minimize a set of "distributed" objective functions, assigned to the vertices and edges:

$$\min_{\boldsymbol{M}} \sum_{j=1}^{n} \tau_{ij} \, w_j \, \|\boldsymbol{r}_j\|_2^2 \qquad \text{for each } i$$

and

$$\min_{\boldsymbol{M}} \sum_{j=1}^{n} \tau_{ij}\, w_j\, \rho_\alpha(x_{hj}) \qquad \text{for each ordered pair } (i,h),\ i \neq h$$

where τ_{ij}, $\boldsymbol{r}_j$ and x_{hj} are functions of $\boldsymbol{M}$.

Is there any single 'global' quantity being optimized implicitly? Can we say that we are also minimizing their sum? Anyway, let's sum them up over i. For the residuals:

$$\begin{aligned}\sum_{i=1}^{k}\sum_{j=1}^{n} \tau_{ij}\, w_j\, \|\boldsymbol{r}_j\|_2^2 &= \sum_{j=1}^{n} w_j\, \|\boldsymbol{r}_j\|_2^2 \sum_{i=1}^{k} \tau_{ij} \\ &= \sum_{j=1}^{n} w_j\, \|\boldsymbol{r}_j\|_2^2 \ .\end{aligned}$$

For each edges ih, $h \neq i$:

$$\begin{aligned}\sum_{i=1}^{k}\sum_{h\neq i}\sum_{j=1}^{n} \tau_{ij}\, w_j\, \rho_\alpha(x_{hj}) &= \sum_{j=1}^{n} w_j \sum_{i=1}^{k} \tau_{ij} \Big\{ \Big[\sum_{h=1}^{k} \rho_\alpha(x_{hj}) \Big] - \rho_\alpha(x_{ij}) \Big\} \\ &= \sum_{j=1}^{n} w_j \Big\{ \Big[\sum_{h=1}^{k} \rho_\alpha(x_{hj}) \Big] - \sum_{i=1}^{k} \tau_{ij}\, \rho_\alpha(x_{ij}) \Big\} \\ &= \sum_{j=1}^{n} w_j \sum_{i=1}^{k} (1-\tau_{ij})\, \rho_\alpha(x_{ij}) \ .\end{aligned}$$

This is interesting. $(1-\tau_{ij})$ is the 'probability' of a point belong to the *facet* opposite $\boldsymbol{\mu}_i$ (cf Figure 3, which shows the complement). x_{ij} is the distance to the facet. So the algorithm is performing what might be called 'k-face clustering' (by analogy with k-mean clustering). The solution depends on the partitioning scheme (definition of τ_{ij}) and the loss function (which can be made asymmetric due to the inside-outside polarity of the face, unlike radial cluster centers).

> This is a nice way to look at the algorithm. We may be able to give it a proper probability interpretation from this angle. For example, we can define a multivariate density with the mode right on hyperplane of a facet (uniform throughout), and decreasing density outside as the function of the perpendicular distance to the face (asymmetrically defined for negative and positive direction). This density can be cosidered a convolution between a positive truncated-tail density (such as exponential) for the "signal" and an isotropic density for the "noise". In this case, the parameter $\sigma = \|\boldsymbol{X}^-\|_F/\sqrt{n}$ is indeed a scale of the noise component. We might be able to cast the algorithm as a type of mixture model fitting using EM algorithm.

Can we say the following quantity is minimized overall?

$$\sum_{j=1}^{n} w_j \Big\{ \|\boldsymbol{r}_j\|_2^2 + \lambda \sum_{i=1}^{k} (1-\tau_{ij})\, \rho_\alpha(x_{ij}) \Big\}$$

$$= \sum_{j=1}^{n} w_j \Big\{ \|\boldsymbol{r}_j\|_2^2 + \lambda \sum_{i=1}^{k} (1-\tau_{ij})\, |\alpha - \mathbf{1}_{\{x_{ij}\leq 0\}}|\, x_{ij}^2 \Big\}$$

with λ related to the size of $\boldsymbol{M}$ (a norm? determinant?), because we need to make the unit of the coefficients comparable with that of $\|\boldsymbol{r}_j\|_2^2$. This makes the algorithm a type of penalized least squares. (We must be implicitly assuming a value of λ somewhere. Where?)

References

Abdous B, Remillard B (1995) Relating quantiles and expectiles under weighted symmetry. *Ann Inst Statist Math* **47**:371–384. `http://citeseer.nj.nec.com`

Cutler A, Breiman L (1994) Archetypal analysis. *Technometrics* **36**:338–347.

Ewald G (1996) Combinatorial convexity and algebraic geometry. *Springer, New York.*

Fuhrmann DR (1999) A simplex shrink-wrap algorithm. *Proc SPIE AeroSense* **3718**:501–511.

Golub GH, Van Loan CF (1996) Matrix computation. 3rd ed. *The John Hopkins Univ. Press, Baltimore.*

Green HM, Kozek AS (2001) Modelling weather data by approximate regression quantiles. `http://conference.maths.uq.edu.au/ctac2001/submission/1001000415.ps`.

Grünbaum B (1967) Convex polytopes. *Wiley, London.*

Huber PJ (1981) Robust Statistics. *Wiley, New York.*

Lee DD, Seung HS (1999) Learning the parts of objects by non-negative matrix factorization. *Nature* **401**:788–791.

Newey WK, Powell JL (1987) Asymmetric least squares estimation and testing. *Econometrica* **55**:819–847.

Parra L, Spence C, Sajda P, Ziehe A, Müller KL (2001) Unmixing hyperspectral data. `http://citeseer.nj.nec.com`.

Renner RM (1993) The resolution of a compositional data set into mixtures of fixed source compositions. *Applied Statistics* **42**:615–631.

Venet D, Pecasse F, Maenhaut C, Bersini H (2001) Separation of samples into their constituents using gene expression data. *Bioinformatics* **17**(Suppl 1):S279–S287.

A Simplex Trivias

References on polytope geometry: Ewald (1996) and Grünbaum (1967).

Faces A *face* of a simplex is the intersection between a supporting and the simplex. If there are k vertices, then there are 2^k faces, including two *improper faces*: $\emptyset$ and the simplex itself. Each face is also a simplex, with the vertices a subset of $\boldsymbol{M}$. Three types of faces have special names:

- A *vertex* is a zero-dimensional face
- An *edge* is a one-dimensional face
- A *facet* is a $(k-2)$-dimensional face

Volume From Fuhrmann (1999). The volume of a simplex $\boldsymbol{M}$ is:

$$V(\boldsymbol{M}) = \frac{|\det \boldsymbol{M}^i|}{(k-1)!}$$

where $\boldsymbol{M}^i = \begin{bmatrix} \boldsymbol{\mu}_1 - \boldsymbol{\mu}_i & \cdots & \boldsymbol{\mu}_k - \boldsymbol{\mu}_i \end{bmatrix}$ and the i-th column removed, for any i, $1 \leq i \leq k$. The determinant can be obtained from the product of the diagonal of $\boldsymbol{R}$ in $\boldsymbol{M}^i = \boldsymbol{Q}\boldsymbol{R}$.