Monografías de la Real Academia de Ciencias de Zaragoza **33**, 39–53, (2010).

Non-negative Matrix Factorisation for Network Reordering

Clare M. Lee, Desmond J. Higham

Department of Mathematics and Statistics, University of Strathclyde, Glasgow G1 1XH.

and

Daniel Crowther, J. Keith Vass

Translational Medicine Research Collaboration, University of Dundee, Dundee DD1 5EH

Abstract

Non-negative matrix factorisation covers a variety of algorithms that attempt to represent a given, large, data matrix as a sum of low rank matrices with a prescribed sign pattern. There are intuitave advantages to this approach, but also theoretical and computational challenges. In this exploratory paper we investigate the use of non-negative matrix factorisation algorithms as a means to reorder the nodes in a large network. This gives a set of alternatives to the more traditional approach of using the singular value decomposition. We describe and implement a range of recently proposed algorithms and evaluate their performance on synthetically constructed test data and on a real data set arising in cancer research.

1 Introduction

Many large, complex networks contain hidden substructures that can be revealed using a range of post-processing algorithms. In particular, reordering the network nodes appropriately may help to summarise key properties by exposing significant clusters, or more generally sets of neighbours with similar features. This work focuses on the use of matrix factorisation methods to derive network reorderings.

Non-negative matrix factorisation (NMF) is a relatively new matrix computation tool that has been applied to problems in data compression and feature extraction [1, 2, 6]. We aim here to study the potential for NMF in network reordering. Our target applications concerns the behaviour of genes and proteins in cells. Microarray data produces large non-square matrices of information recording the behaviour of many genes across a small number of samples. This data is by its very nature non-negative. A common aim is then to cluster or order the genes/samples into groups where members behave similarly to each other and differently to those in other groups. This allows us to find sets of genes whose behaviour distinguishes different sample types. As we are particularly interested in using the technique to classify clusters of samples from microarray data we typically talk about the rows of a matrix being genes and the columns as samples. Currently this is often done using the singular value decomposition. [3, 4].

In Section 2 we introduce non-negative matrix factorisation, before giving specific algorithms in Section 3. In this section we also describe how we use the factorisation for reordering. In Section 4 we test all the algorithms on synthetic data before looking at a real data set in Section 5.

2 Non-negative Matrix Factorisation

In general, given an input matrix A with non-negative entries, $a_{i,j} \geq 0$, NMF produces lower rank non-negative factors W and H, so that

$$A \approx WH, \tag{1}$$

with $A \in \mathbb{R}^{m\times n}$, $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{k\times n}$, for $k \ll \min(m, n)$. We often express the two factors in terms of their rows or columns, that is, $W = [w_1, \dots, w_k]$, and $H = [h_1, \dots, h_k]^{\mathrm{T}}$. NMF techniques, which are relatively new in the context of network reordering, have the intuitive advantage of respecting the non-negative nature of the original data. It has been shown in other areas of research that the outer product of a column of the first factor w_i and a row of the second h_i may pick out a feature of the original data [6]. For network reordering this has the potential to generate bases of “eigen-genes” or “eigen-samples” that combine non-negatively to represent the expression of a particular gene across the samples or the gene expression for a particular sample type. In this setting the (i, j)th entry of $W_{i,j}$ is the level of expression of gene i in eigen-sample j, and $H_{i,j}$, the (i, j)th entry of H, is the importance of eigen-gene i in sample j.

Although NMFs take advantage of the non-negative nature of the original data, they have various theoretical and practical drawbacks that are yet to be fully explored in the network reordering context. Different NMF variations can produce very different results, and the iterative nature of the underlying computations makes them highly sensitive to the choice of target rank and initial starting guess.

3 Specific NMF Algorithms

There are many different NMF algorithms. We look at five variants [7].

Multiplicative Update Here both factors need to be initialised with starting guesses

W^0 and H^0, typically random matrices. The algorithm is then:

$$\left.\begin{aligned} H^{i+1} &= H^i .* \frac{{W^i}^{\mathrm{T}} A}{{W^i}^{\mathrm{T}} W^i H^i + 10^{-9}} \\[2ex] W^{i+1} &= W^i .* \frac{A {H^i}^{\mathrm{T}}}{W^i H^i {H^i}^{\mathrm{T}} + 10^{-9}} \end{aligned}\right\} \tag{2}$$

where .∗ denotes point-wise multiplication.

Alternating Least Squares Only W^0 needs to be initialised. The algorithm is

$$\left.\begin{array}{l} \text{Solve for } H^{i+1} \text{ in } W^i H^{i+1} = A \\ \text{Set all negative entries in } H^{i+1} \text{ to } 0 \\ \text{Solve for } W^{i+1} \text{ in } W^{i+1} H^{i+1} = A \\ \text{Set all negative entries in } W^{i+1} \text{ to } 0 \end{array}\right\} \tag{3}$$

Tri-factorisation Here A is factorised into three factors so that $A \approx WSH$ with $W \in \mathbb{R}^{m\times k}$ $S \in \mathbb{R}^{k\times \ell}$ and $H \in \mathbb{R}^{\ell \times n}$, with $k, \ell \ll \min(m, n)$. This allows different numbers of clusters in the rows and columns of the data. After initialising all the factors with random guesses the algorithm is then:

$$\left.\begin{aligned} H^{i+1} &= H^i .* \sqrt{\frac{{S^i}^{\mathrm{T}} {W^i}^{\mathrm{T}} A}{{S^i}^{\mathrm{T}} {W^i}^{\mathrm{T}} A {H^i}^{\mathrm{T}} H^i + 10^{-9}}} \\[2ex] W^{i+1} &= W^i .* \sqrt{\frac{A {H^i}^{\mathrm{T}} {S^i}^{\mathrm{T}}}{W^i {W^i}^{\mathrm{T}} A {H^i}^{\mathrm{T}} {S^i}^{\mathrm{T}} + 10^{-9}}} \\[2ex] S^{i+1} &= S^i .* \sqrt{\frac{{W^i}^{\mathrm{T}} A {H^i}^{\mathrm{T}}}{{W^i}^{\mathrm{T}} W^i S^i H^i {H^i}^{\mathrm{T}} + 10^{-9}}} \end{aligned}\right\} \tag{4}$$

Both the multiplicative update method and the alternating least squares algorithms are included in the MATLAB statistics toolbox[1]. There are also partial-non-negative factorisations, which allow an input matrix of mixed sign and produce factorisations where one of the two factors is non-negative. We will consider the case where $A \approx WH$ as before but with A and W allowed to have negative entries. Two examples of these are;

Semi-NMF Only H^0 is initialised with the algorithm being:

$$\left.\begin{aligned} W^{i+1} &= A {H^i}^{\mathrm{T}} (H^i {H^i}^{\mathrm{T}})^{-1} \\[2ex] H^{i+1} &= H^i .* \sqrt{\frac{({W^i}^{\mathrm{T}} A)^+ + ({W^i}^{\mathrm{T}} W^i)^- H^i}{({W^i}^{\mathrm{T}} A)^- + ({W^i}^{\mathrm{T}} W^i)^+ H^i}} \end{aligned}\right\} \tag{5}$$

[1]`http://www.mathworks.com/`

Convex-NMF In this factorisation the columns of W lie in the space spanned by the columns of A, i.e. $W = AR$ so $A \approx ARH$. The algorithm is;

$$\left.\begin{aligned} H^{i+1} &= H^i.*\sqrt{\frac{{R^i}^{\mathrm{T}}(A^{\mathrm{T}}A)^+ + {R^i}^{\mathrm{T}}(A^{\mathrm{T}}A)^- R^i H^i}{{R^i}^{\mathrm{T}}(A^{\mathrm{T}}A)^- + {R^i}^{\mathrm{T}}(A^{\mathrm{T}}A)^+ R^i H^i}} \\ R^{i+1} &= R^i.*\sqrt{\frac{(A^{\mathrm{T}}A)^+ {H^i}^{\mathrm{T}} + (A^{\mathrm{T}}A)^- R^i H^i {H^i}^{\mathrm{T}}}{(A^{\mathrm{T}}A)^- {H^i}^{\mathrm{T}} + (A^{\mathrm{T}}A)^+ R^i H^i {H^i}^{\mathrm{T}}}} \end{aligned}\right\} \tag{6}$$

where $(M)^+ = \dfrac{(|M| + M)}{2}$ and $(M)^- = \dfrac{(|M| - M)}{2}$, the positive and negative parts of the matrix.

In all cases the aim is that a row/column of the factorisation matrices W and H conveys information on a single feature of the data. For $k = 1$ all the algorithms produce the same result. In this case W is the first left singular vector of A and H is the first right singular vector. However, for $k > 1$ the algorithms all differ.

3.1 Permutation

We can show that one positive feature as of all these algorithms is that they are impervious to permutation of the original matrix. This means that the initial ordering of the nodes in the network has no influence on the final reordering produced by the algorithm.

This can be verified for all algorithms using the properties of permutation matrices. By way of example we show the calculations for the method of multiplicative update. Letting P_1 and P_2 denote arbitrary permutation matrices and starting with $\widetilde{W^0} = P_1 W^0$ and $\widetilde{H^0} = H^0 P_2^{\mathrm{T}}$, the factorisation of $P_1 A P_2^{\mathrm{T}}$ gives that;

$$\begin{aligned} \widetilde{W^1} &= P_1 W^0.*\frac{P_1 A P_2^{\mathrm{T}}\left(H^0 P_2^{\mathrm{T}}\right)^{\mathrm{T}}}{P_1 W^0 H^0 P_2^{\mathrm{T}}\left(H^0 P_2^{\mathrm{T}}\right)^{\mathrm{T}} + 10^{-9}} \\ &= P_1 W^0.*\frac{P_1 A {H^0}^{\mathrm{T}}}{P_1 W^0 H^0 {H^0}^{\mathrm{T}} + 10^{-9}} \\ &= P_1\left(W^0.*\frac{A {H^0}^{\mathrm{T}}}{W^0 H^0 {H^0}^{\mathrm{T}} + 10^{-9}}\right) \\ &= P_1 W^1 \end{aligned}$$

and

$$\begin{aligned}\widetilde{H^1} &= H^0 P_2^{\mathrm{T}} .* \frac{\left(P_1 W^0\right)^{\mathrm{T}} P_1 A P_2^{\mathrm{T}}}{\left(P_1 W^0\right)^{\mathrm{T}} P_1 W^0 H^0 P_2^{\mathrm{T}} + 10^{-9}} \\ &= H^0 P_2^{\mathrm{T}} .* \frac{{W^0}^{\mathrm{T}} A P_2^{\mathrm{T}}}{{W^0}^{\mathrm{T}} W^0 H^0 P_2^{\mathrm{T}} + 10^{-9}} \\ &= \left(H^0 .* \frac{{W^0}^{\mathrm{T}} A}{{W^0}^{\mathrm{T}} W^0 H^0 + 10^{-9}} \right) P_2^{\mathrm{T}} \\ &= H^1 P_2^{\mathrm{T}}.\end{aligned}$$

Therefore the result follows by induction.

3.2 Ordering and Clustering

Using ideas from [2] we put the rows of W and the columns of H into k clusters. Each row of W is placed into a cluster according to the most highly expressed column, e.g. row i of W belongs to cluster j if $W_{i,j}$ is the maximum over $W_{i,\{1,\dots,k\}}$. Similarly for the columns of H, column j of H belongs to cluster i if $H_{i,j}$ is the maximum over $H_{\{1,\dots,k\},j}$. Each cluster is then sorted independently by size of element in that column/row. So the rows of W in cluster j are sorted in order of the jth column of W, and the columns of H in cluster i are sorted in order of the ith row of H. The orderings are then put together into a single reordering vector with the reordered cluster 1 appearing before the reordered cluster 2 and so on. The MATLAB code for clustering and reordering W reads:

```
% finding the clusters
for i=1:m
    [b,clustW(i)]=max(W(i,:));
end

% sorting the clusters
for i=1:k
    k_ind=find(clustW==i);
    [b,tind]=sort(W(k_ind,i));
    indW(kind)=kind(tind);
end

% sorting the ordering so we have cluster 1
% then cluster 2 etc...
[d,cind]=sort(clustW);
indW=indW(cind);
clustW=d;
```

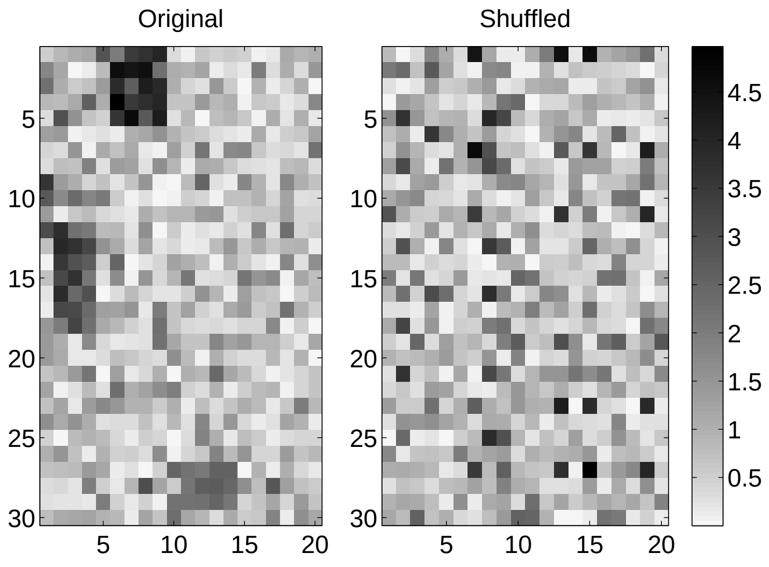

Figure 1.— The left panel has the original test data with three hidden blocks. The panel on the right shows this matrix with the rows and columns shuffled independently; this is the matrix that is factorised.

where `indW` contains the ordering indices for the rows and `clustW` has the reordered cluster numbers. The same method is used to sort the columns of H.

4 Performance on test data

The first test for the algorithms was whether they could find significant blocks placed in a random matrix. The test matrix, generated by a pseudo-random number generator, has three blocks with higher values, as shown in the left panel of Figure 1. The highest block is referred to as block 1 and the lowest is block 3. The rows and columns of this matrix are then shuffled independently to produce the picture on the right of Figure 1. This is the matrix the algorithms are tested on. In all the figures the darker colours represent the higher values in the matrix as shown by the colourbar in this figure. Each algorithm is run with 10 different random initial conditions and the factorisation that produces the the lowest approximation error in the Frobenius norm $\|A - WH\|_F$ is chosen to represent the factorisation method.

The results are shown in Figures 2–7. The first three panels of the top row show the matrix reshuffled using the relative sizes of the first, second and third rows of W and the first second and third columns of H respectively. The fourth shows them clustered and ordered as in Section 3.2. The bottom row shows the ordering and clusters in the clustered version with $*$ denoting the original block 1, $\circ$ block 2, and $+$ block 3; the remaining rows and columns are shown by dots. The vertical axis shows the cluster number with the reordered (as in Section 3.2) position on the horizontal axis. The graph on the left shows

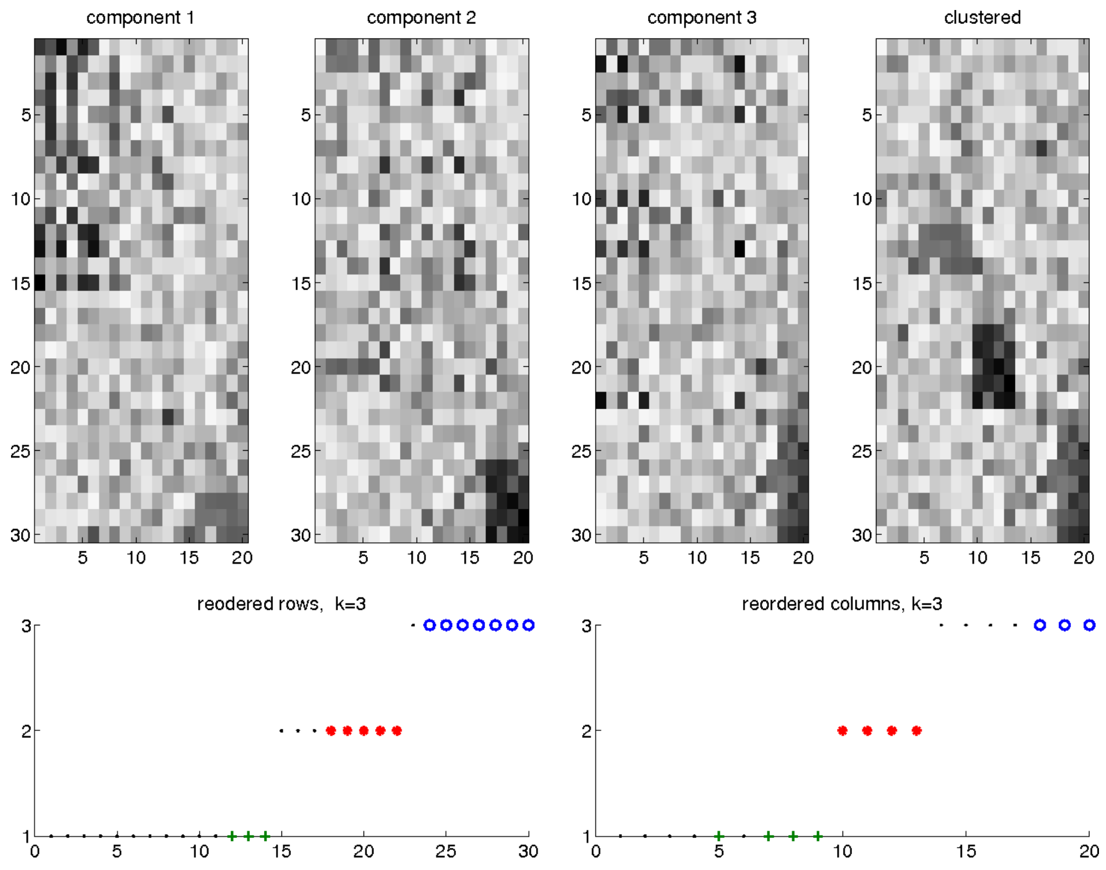

Figure 2.— Results using multiplicative update with $k = 3$.

the reordering of the rows and the graph on the right that for the columns.

Figure 2 shows the results using multiplicative update (2) for $k = 3$. We see that ordering purely on the magnitude of the first column of W and first row of H finds one of the blocks in the original matrix, as do the second and third components. Using the clustering and ordering given in Section 3.2 we see all the blocks appearing with one block in each cluster. For this algorithm we get that $\|A - WH\|_F = 0.55737$.

The results from the alternating least squares method (3) are in Figure 3 for $k = 3$. The algorithm converges to a rank one solution rather than a rank 3 solution. All but the first column of W and the first row of H are zeros. There is only one "cluster" found, but the row reordering nearly finds two of the three blocks. The column ordering is not as good. This algorithm is also slower than the multiplicative update, and the approximation isn't as close to the original matrix with $\|A - WH\|_F = 0.83243$.

With the tri-factorisation (4) the clusters are no longer necessarily in the same order in the rows and columns; this can be seen in Figure 4. However looking at the matrix S, shown in Figure 5, we can see that row cluster i corresponds to column cluster j if $S_{i,j} = \max_{p\in\{1,\ldots,\ell\}} S_{i,p}$. Similarly, column cluster j corresponds to row cluster i if $S_{i,j} = \max_{p\in\{1,\ldots,k\}} S_{p,j}$. If $k \neq \ell$ it is possible that there are row or column clusters

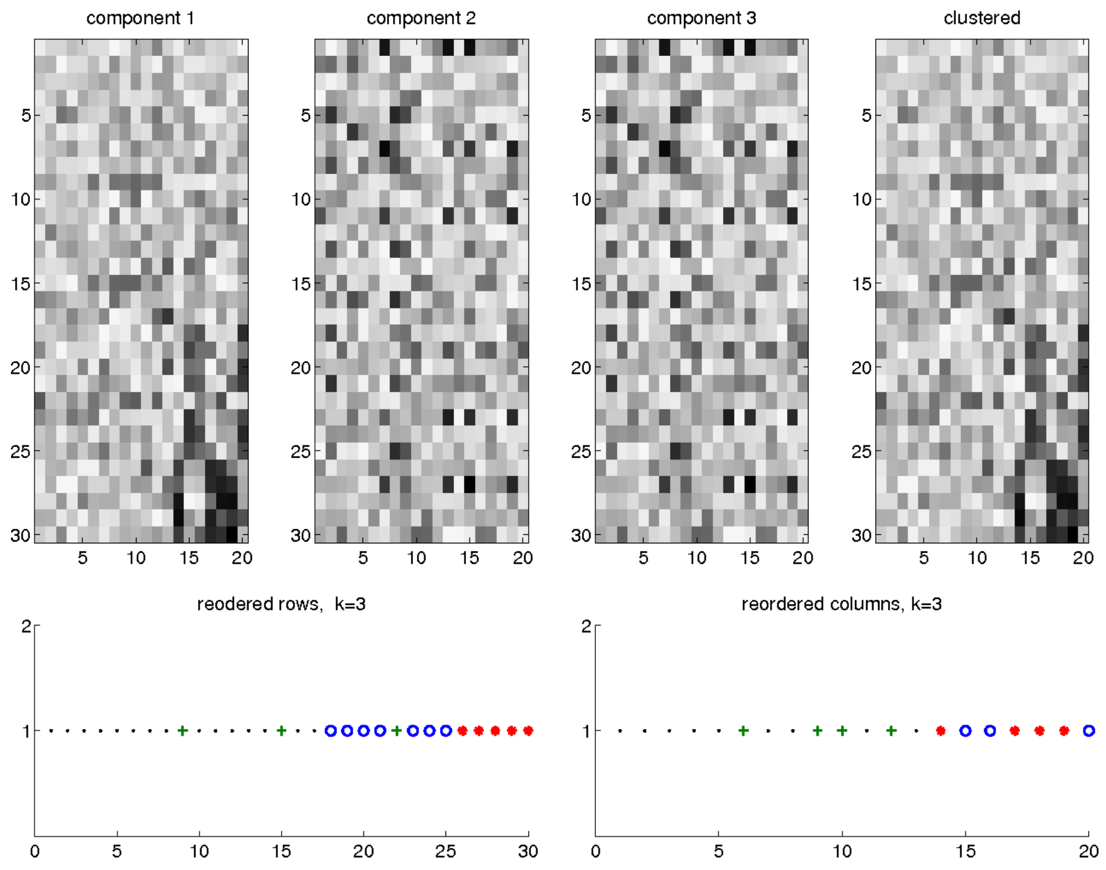

Figure 3.— Results using alternating least squares with $k = 3$.

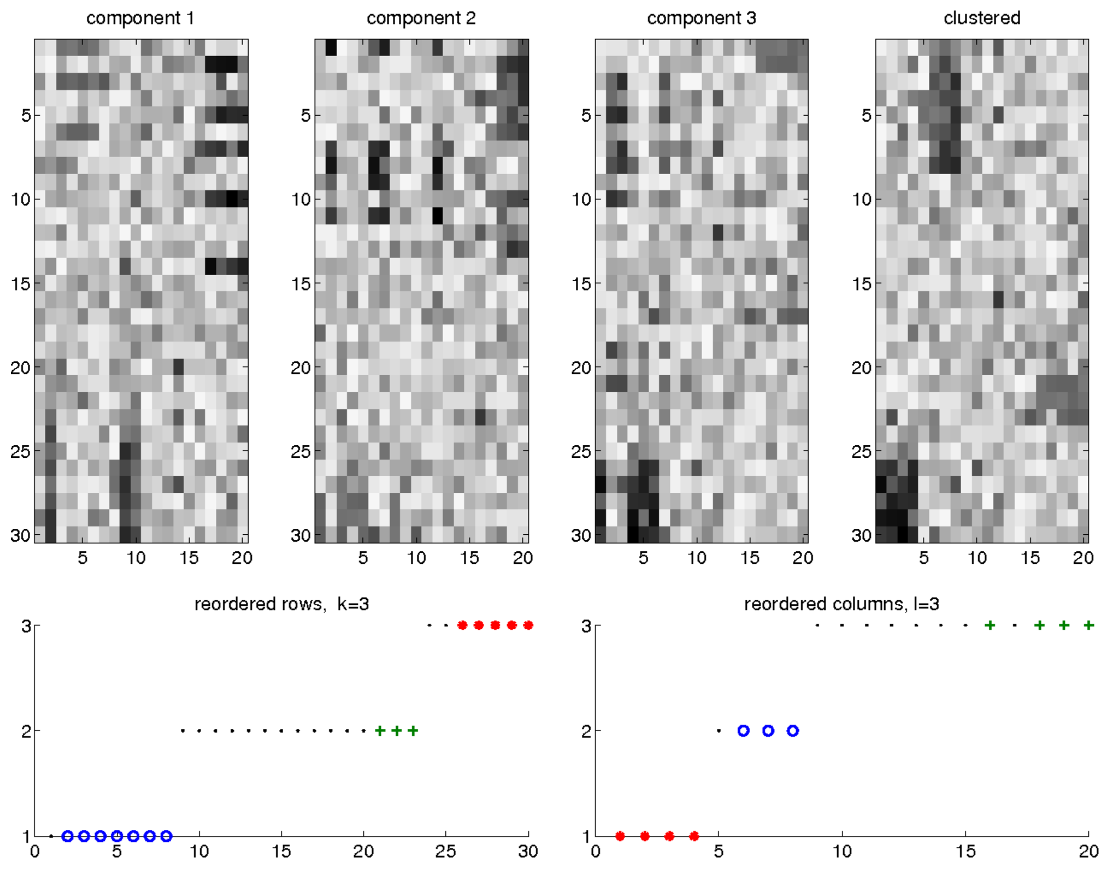

Figure 4.— Results using tri-factorisation with $k = 3$ and $l = 3$.

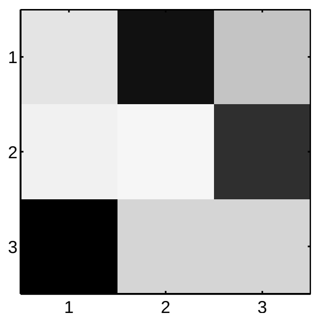

Figure 5.— A heat map of the middle factor S of the factorisation.

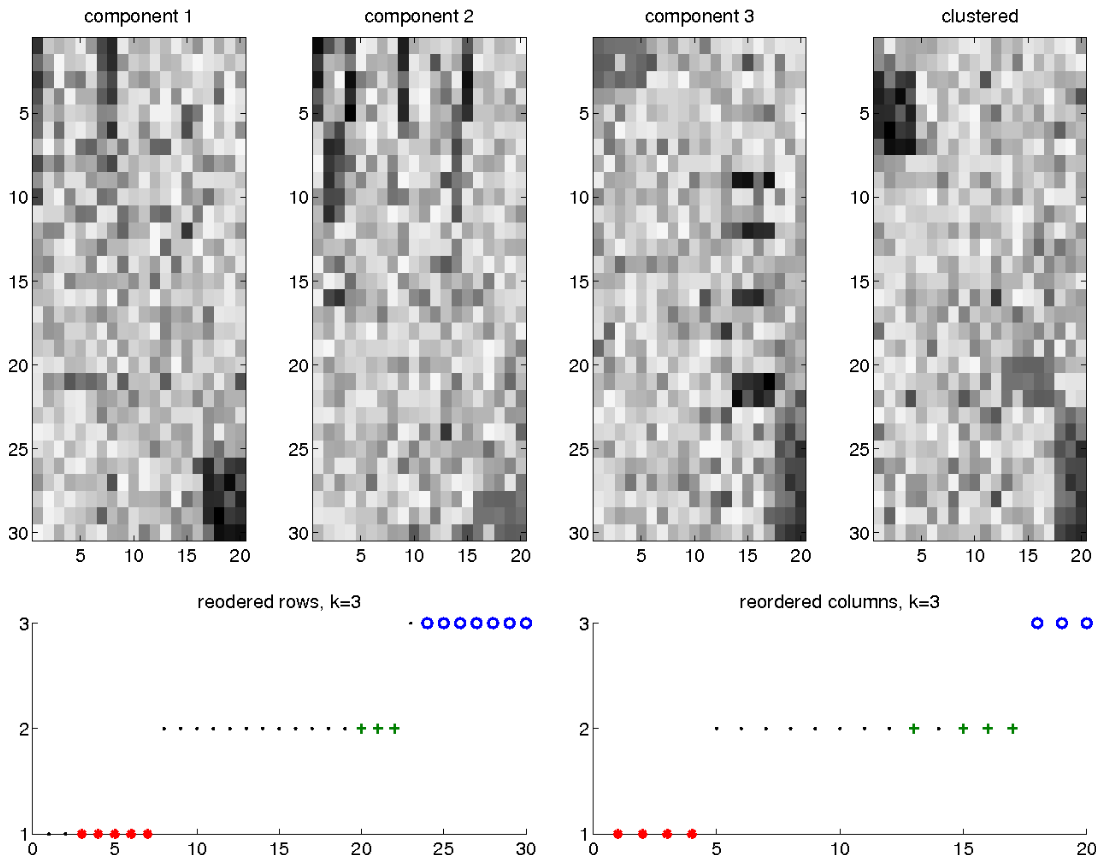

Figure 6.— Results using semi-NMF with $k = 3$.

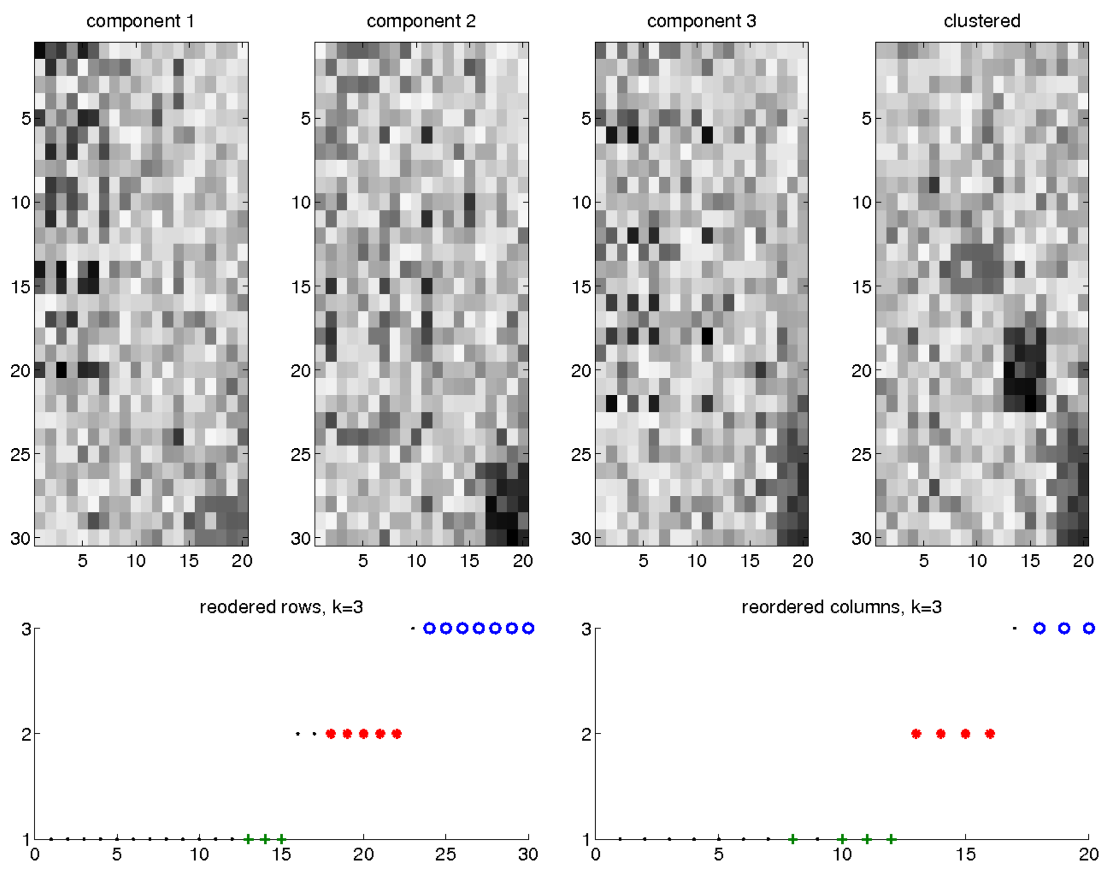

Figure 7.— Results using convex-NMF with $k = 3$.

that do not corrrespond to a cluster the other dimension, usually this row/column of S doesn't have any particularly large values. It is also possible that one row/column cluster corresponds to more than one different column/row clusters, these are still the highest values in S, usually noticeably larger than the other values. For this test example with $k = \ell = 3$ the results are very similar to those using multiplicative update, but with the row/column cluster shuffled. The algorithm also takes about the same time as the multiplicative update and $\|A - WSH\|_F = 0.57826$, similar to that of the multiplicative update. The major difference between the two algorithms is that here we can have different numbers of row clusters and column clusters. In some applications this can be a benefit, however, it does give another parameter that needs to be fixed before the factorisation can begin.

Semi-NMF (5), shown in Figure 6, is the fastest of the algorithms and also produces good results, with $\|A - WH\|_F = 0.55720$.

Initialising both R and H with random matrices the convex-NMF algorithm (6), shown in Figure 7, does as well as any of the other algorithms but it takes a lot longer to run with the error being on a par with the other algorithms at $\|A - WH\|_F = 0.56763$. It is considerably faster if $R^0 = g/(g' * g)$, where g is a random matrix, and the results are similar.

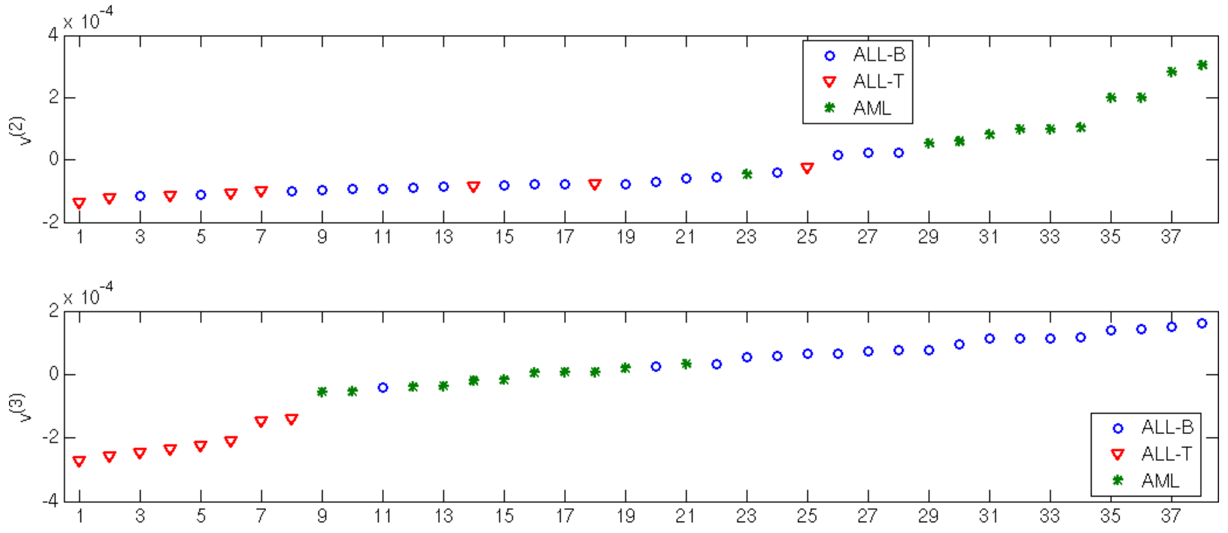

Figure 8.— The ordering as given by the singular value decomposition. The vertical axis has the value of the normalised second or third right singular vector in the top and bottom images respectively.

5 Performance on a cancer microarray data set

We have further tested the algorithms on microarray data from bone marrow samples from patients with acute leukaemia. In the data set the rows represent different genes and the columns are the samples. An individual entry is then the level of gene expression in that sample. The matrix is therefore long and thin with 5000 rows and only 38 columns. There are different types of leukaemia represented here: acute myelogenous leukaemia (AML) and acute lymphoblastic leukaemia (ALL). The second type is further divided into T and B cell subtypes. This data set has been widely used to test algorithms, see for example [2, 5]. It contains two samples that are misclassified by many methods, these are included in our analysis. This time the factorisation with the lowest approximation error is chosen from five different random initial conditions.

For comparison we first show the results achieved by using the singular value decomposition on a normalised data set, see [4] for details. These can be found in Figure 8; where we see the sorted index numbering along the horizontal axis with the value of the right singular vector on the vertical axis. The top graph shows the samples sorted using the second right singular vector, here we see that with one exception the AML type and the ALL types are split. The lower graph has the same information, but sorted by the third singular vector. This time the three different types are split with only two exceptions.

The results from the multiplicative update are in Figure 9. In this figure, and in those following, the value on the vertical axis is the cluster number. This time we see that choosing $k = 2$, as in the top graph, we correctly split the ALL and AML type with two exceptions. For $k = 3$ in the bottom graph we have split the three different leukaemia types again with two exceptions. As we cluster the results from the non-

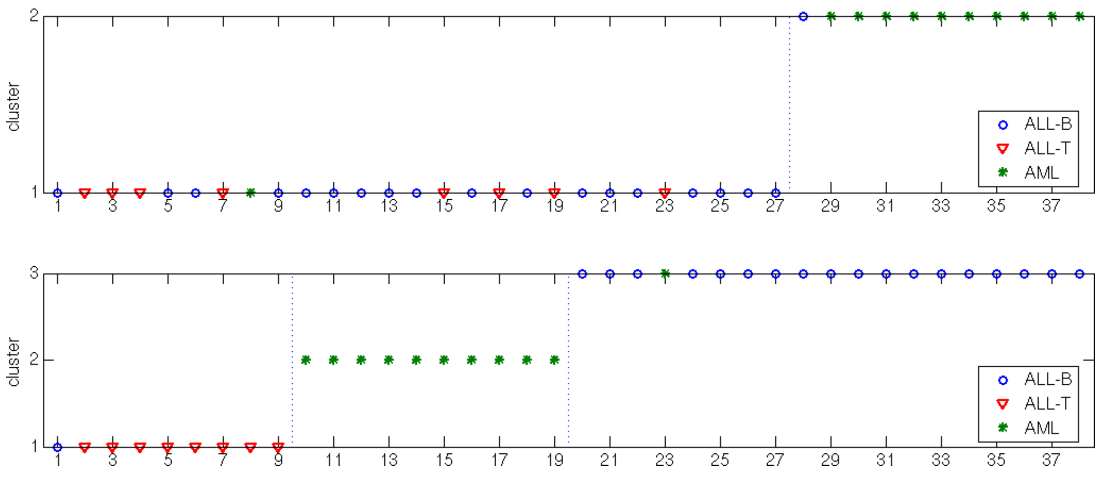

Figure 9.— The ordering as given by the multiplicative update. The vertical axis has the cluster number for $k = 2$, or $k = 3$ in the top and bottom images respectively.

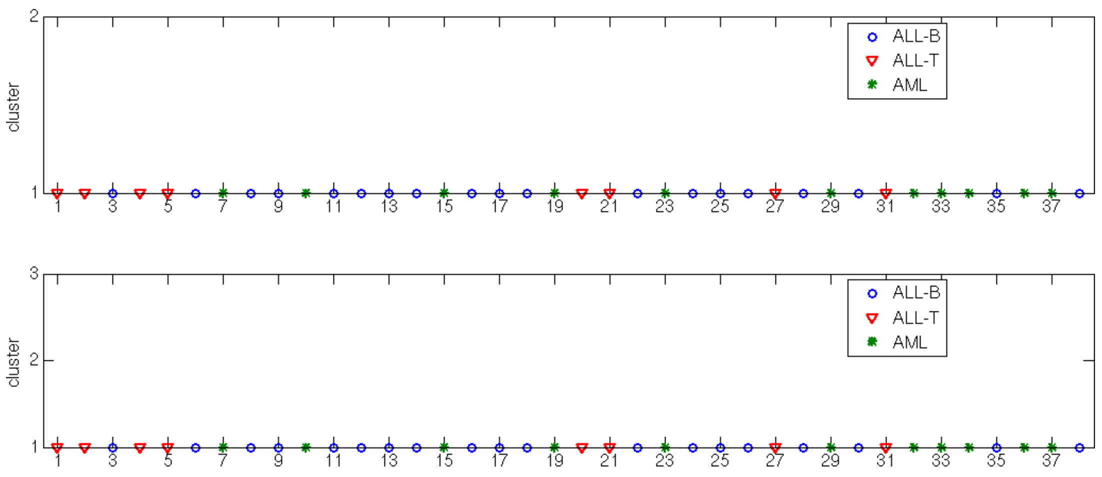

Figure 10.— The ordering as given by the alternating least squares algorithm. The vertical axis has the cluster number for $k = 2$, or $k = 3$ in the top and bottom images respectively.

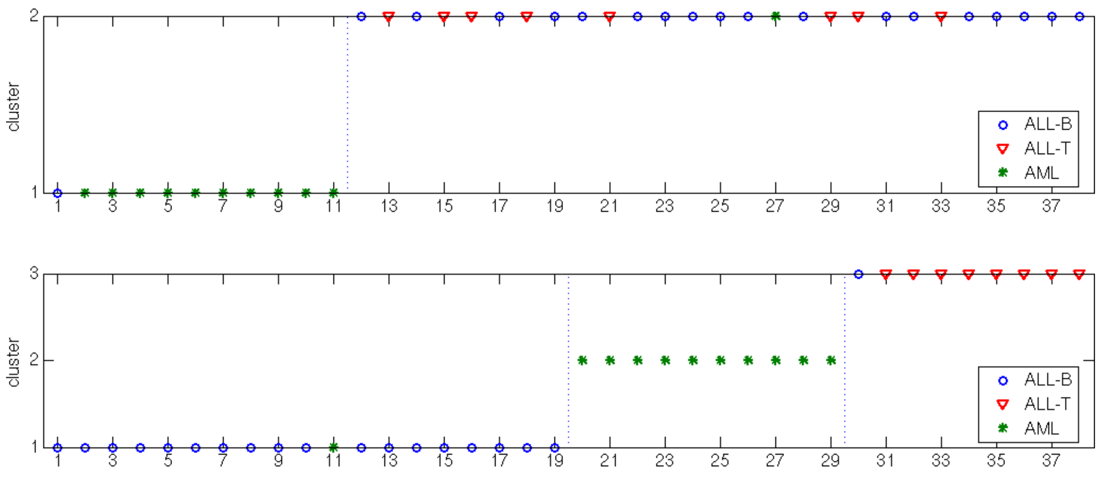

Figure 11.— The ordering as given by tri-factorisation. The vertical axis has the cluster number for $k = 2$, or $k = 3$ in the top and bottom images respectively.

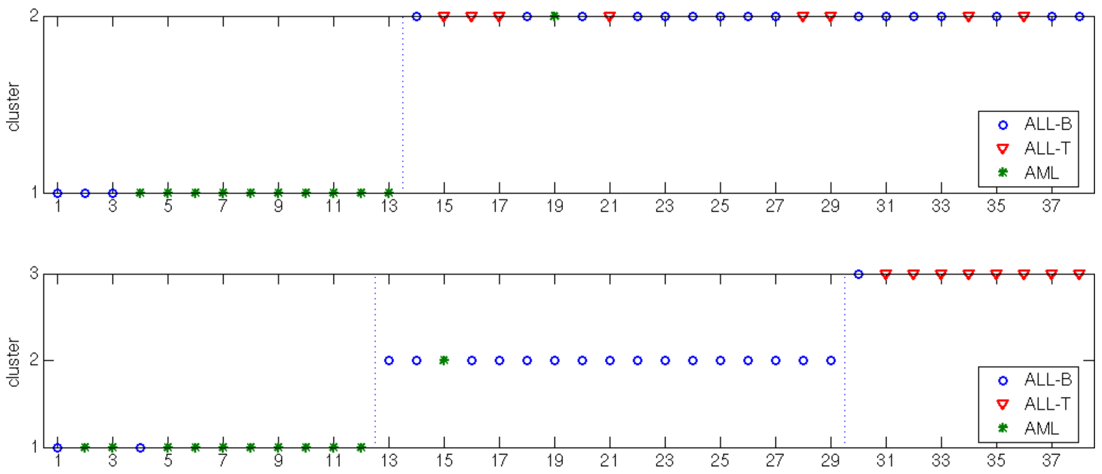

Figure 12.— The ordering as given by semi-NMF. The vertical axis has the cluster number for $k = 2$, or $k = 3$ in the top and bottom images respectively.

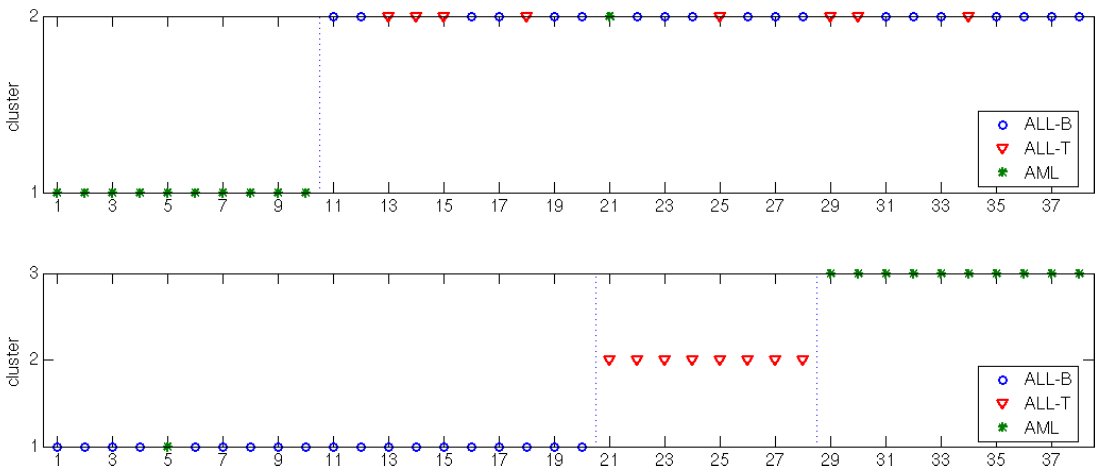

Figure 13.— The ordering as given by convex-NMF. The vertical axis has the cluster number for $k = 2$, or $k = 3$ in the top and bottom images respectively.

negative factorisations, it has the benefit of clear separation of the cell types unlike the singular value decomposition approach where the split is less clear.

The alternating least squares algorithm does a bad job of clustering or ordering the samples, just as it did with the test data in Section 4. The results are shown in Figure 10, where we see that the algorithm produces a rank one solution regardless of the choice of k, and even ordering on the size of the elements of the rank one matrix H does not split the data into the different groups.

As we have mentioned before, it is possible to have different numbers of row and column clusters when using tri-factorisation. However, for ease of comparison we present results for $k = l = 2$, or 3; see Figure 11. For both $k = l = 2$ and $k = l = 3$, we get very similar results to those from the multiplicative update method, though it takes longer. This method is of more benefit in situations where having different numbers of row and column clusters gives a clearer picture.

The semi-NMF algorithm is one of the quicker algorithms, though this test shows that it isn't as accurate in splitting the different leukaemia types. The results in Figure 12 show that with $k = 2$ in the top picture one AML type is placed amongst the ALL types, but three of the ALL types are also placed in the cluster dominated by the AML type. Increasing k to 3 doesn't increase the accuracy with four samples mis-assigned.

The convex-NMF algorithm, despite being able to factorise a mixed sign matrix into the product of a mixed sign and a non-negative matrix, produces clusters that split the different types. For $k = 2$ in the top row of Figure 13 shows that there is a single sample misplaced. When $k = 3$ as in the bottom graph there are only the two samples mis-assigned, these results are on a level with the multiplicative update, and the tri-factorisation.

6 Summary

In this paper we have investgated the use of non-negative matrix factorisation algorithms for reordering data sets and applied them to genetic microarray data. If we look solely at the ability to distinguish the different clusters it would appear that the multiplicative update, tri-factorisation and the convex-NMF algorithms are the best algorithms to use. However, since the tri-factorisation and convex-NMF are much slower, the multiplicative update appears to be the winner in most settings. The other two methods both have added features though, making them possibly more useful in other contexts.

The algorithms have both practical and theoretical challenges that need to be addressed. A basic issue is how to choose the rank of the desired factorisation. How do we choose a value of k (and ℓ) which gives full information from the data without producing misleading results? From a practical point of view it would be ideal to be able to evaluate the data set and then compute one factorisation. However this is probably unrealistic, a more feasible answer would be to find some statistic by which to compare the factorisations for different values of k. A further challenge is how to initialise the algorithms since they produce different results for different initial conditions. Currently we make multiple runs with different initial random matrices and use the factorisation that produces the lowest approximation error. With large data sets this can become costly and time consuming, so some investigation into how best to pick the initial guess would be beneficial. From a theoretical perspective, it would be very helpful to have analytical results that allow us to distinguish between the NMF variants in the network reordering application, and compare them with the more traditional singular value decomposition approach.

Acknowlegments

C.M. Lee and D.J. Higham are supported by EPSRC Grant EP/E049370/1

References

[1] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorisation. *Comput. Stat. Data Anal*, 52:155–173, 2007.

[2] Jean-Philippe Brunet, Pablo Tamayo, Todd R. Golub, and Jill P. Mesirov. Metagenes and molecular pattern discovery using matrix factorisation. *Proc. Nat. Acad. Sci.*, 101(12):4164–4169, 2004.

[3] Desmond J. Higham, Gabriela Kalna, and Milla Kibble. Spectral clustering and its use in bioinformatics. *J. Comput. Appl. Math.*, 204:25–37, 2007.

[4] Desmond J. Higham, Gabriela Kalna, and J. Keith Vass. Analysis of the singular value decomposition as a tool for processing microarray expression data. In *Proceedings of ALGORITMY*, pages 250–259, Slovak Un. of Tech., 2005.

[5] Hyunsoo Kim and Haesun Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12):1495–1502, 2007.

[6] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[7] Tao Li and Chris Ding. The relationship among various nonnegative matrix factorization methods for clustering. In *Proc. IEEE Int'l Conf. on Data Mining (ICDM'06)*, pages 362–371, 2006.