# **Singular Value Decomposition (SVD) of microarray data.**

In the previous chapter we have learned the SVD decomposition of any *m* x *n* matrix ***X*** as:

*m* x *n* *m* x *m* *m* x *n* *n* x *n*

such that:

with:

An extremely important application of this concept is in *systems biology*. Consider the case of microarray data ***X***, in which *xij* is the expression level of the *i*th gene in the *j*th assay. The elements of the *i*th row of ***X***form the *n*-dimensional vector **g***i*, which we refer to as the *transcriptional response* of the *i*th ***g****ene* in different assays. Alternatively, the elements of the *j*th column of ***X***form the *m*-dimensional vector **a***j*, which we refer to as the genes *expression profile* in the *j*th ***a****ssay*.

**g1** = 1st transcriptional response (expression of the 1st **gene** in all the assays)

*n**assays*

*m**genes*

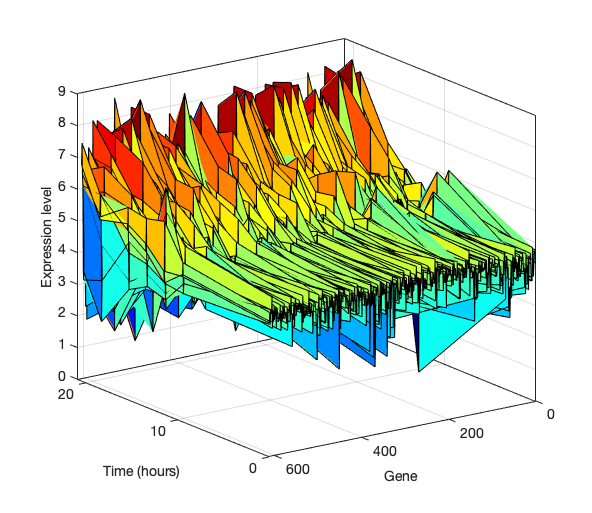
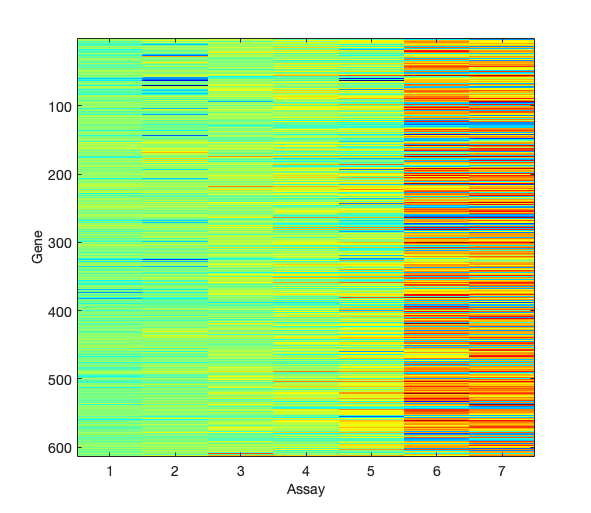
**a1** = **1st** expression profile (expression of all the genes in the 1st **assay**)

If one conditions the data matrix ***X***by *centering* each column, then ***X*T*X***(*n x n*) is proportional to the covariance matrix of the expression profiles). *Eigen* decomposition of ***X*T*X***yields ***V***: so, the right singular vectors ***v1***, ***v2***, ... are the orthonormal basis (= *principal components axes*) of the *row space* (the space of the gene transcriptional responses).

If one conditions the data matrix ***X***by *centering* each row, ***XX*T** (*m x m*) is proportional to the covariance matrix of the genes transcriptional responses. In this case, the left singular vectors ***u1***, ***u2***, ... are the orthonormal basis (= *principal components axes*) of the *column space*, the space of the assay expression profiles.

Thus, if we have a gene expression data matrix ***X***with *n* columns corresponding to assays, and *m* rows corresponding to genes, the **SVD** of ***X***produces two orthonormal bases of right and left singular vectors. It is a widely used convention to refer to the left singular vectors ***u****k* as *eigenassays* (or *eigenprofiles*) and to the right singular vectors ***v****k* as *eigengenes*. Furthermore, when the number of genes *m* is larger than the number of assays *n* (the most common case), usually the '*economy*' SVD is calculated in which the first *n*columns of ***U*** are included (regardless of the matrix rank) and therefore ***Σ*** and ***V***  are both *n* x *n*.

*m* x *n* *m* x *n* *n* x *n* *n* x *n*



Let's see how this works in practice. As an example, we will consider microarray data ***X*** in which the expression of *m*= 614 yeast genes was monitored for *n**=*7 different times (up to 20 hours) after changing the main nutrient in the culture media at time 0.

load filteredyeastdata

data = yeastvalues; [m,n] = size(data);

figure;imagesc(data);colormap jet

xlabel('Assay');ylabel('Gene');

[XI,YI]=meshgrid(times,1:m);

figure;surf(XI,YI,data);

box('on'); ylim([0 614]); xlim([0 21]);

xlabel('Time (hours)'); ylabel('Gene')

zlabel('Expression level')

We can see how the expression of some genes goes up and that of others goes down. We start by calculating the *economy* SVD of the ***X*** matrix.

Microarray\_SVD = figure

[U,S,V] = svd(data,'econ');

set(gcf,'Unit','Normalized','Position',[0.1 0.0 0.6 1.0])

pos1 = [0.05 0.08 .2 .87];subplot('Position',pos1);imagesc(U\*S\*V');colormap jet

title('X \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',24)

xlabel('Assay','FontWeight','Bold','FontSize',24,'Color','Blue'),

ylabel('Gene','FontWeight','Bold','FontSize',24,'Color','Red')

pos2 = [0.29 0.08 .2 .87];subplot('Position',pos2);imagesc(U);colormap jet

title('U \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',24,'Color','Blue')

xlabel('Eigen Assay','FontWeight','Bold','FontSize',24,'Color','Blue')

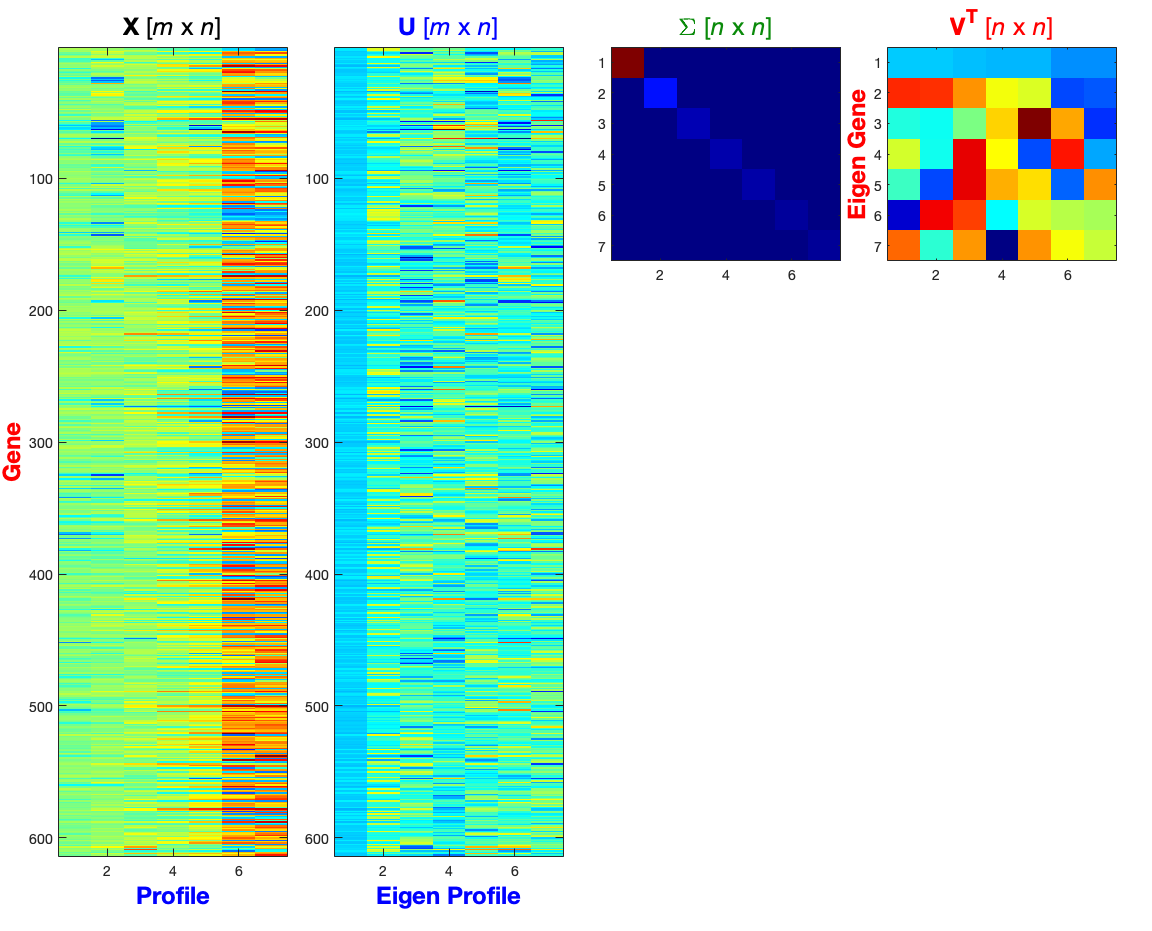
pos3 = [0.53 0.72 .2 .23];subplot('Position',pos3);imagesc(S);colormap jet

title('\Sigma \rm[\itn \rmx \itn\rm]','FontWeight','Bold','FontSize',24,'Color',[0.04 0.54,0.04])

pos4 = [0.77 0.72 .2 .23];subplot('Position',pos4);imagesc(V');colormap jet

title('V^T \rm[\itn \rmx \itn\rm]','FontWeight','Bold','FontSize',24,'Color','Red')

ylabel('Eigen Gene','FontWeight','Bold','FontSize',24,'Color','Red')



The *systems biology* interpretation of the 'economy' **SVD** decomposition of microarray data views the columns of the ***U*** matrix as the *unscaled* *eigenprofiles*, and the rows of the ***VT*** matrix as the *eigengenes*.

We can ask the question: **are there genes that have a common pattern of activation or inhibition?** In other words, we are looking at the *row space*, the space of the gene transcriptional responses.

Since the rank of ***X*** is *r* = 7,intuitively we can see how the transcriptional response of each gene can be considered as a linear combination of 7 *eigengenes*, each *eigengene* being a vector of 7 elements, and each element representing some level of transcriptional response at that time. Therefore, we want to find the representation of ***X*** (the *scores*) in terms of *eigengenes* (the orthogonal basis ***V*** for the *row space* of ***X***). In order to do this we apply a change of basis to ***X*** into ***V*** space. Notice that in this case we are considering the columns of the microarrays data as the variables (=coordinates), and the rows as the observations. Thus, the centered data ***cX*** is obtained by subtracting from each row of ***X*** the mean of all the rows:

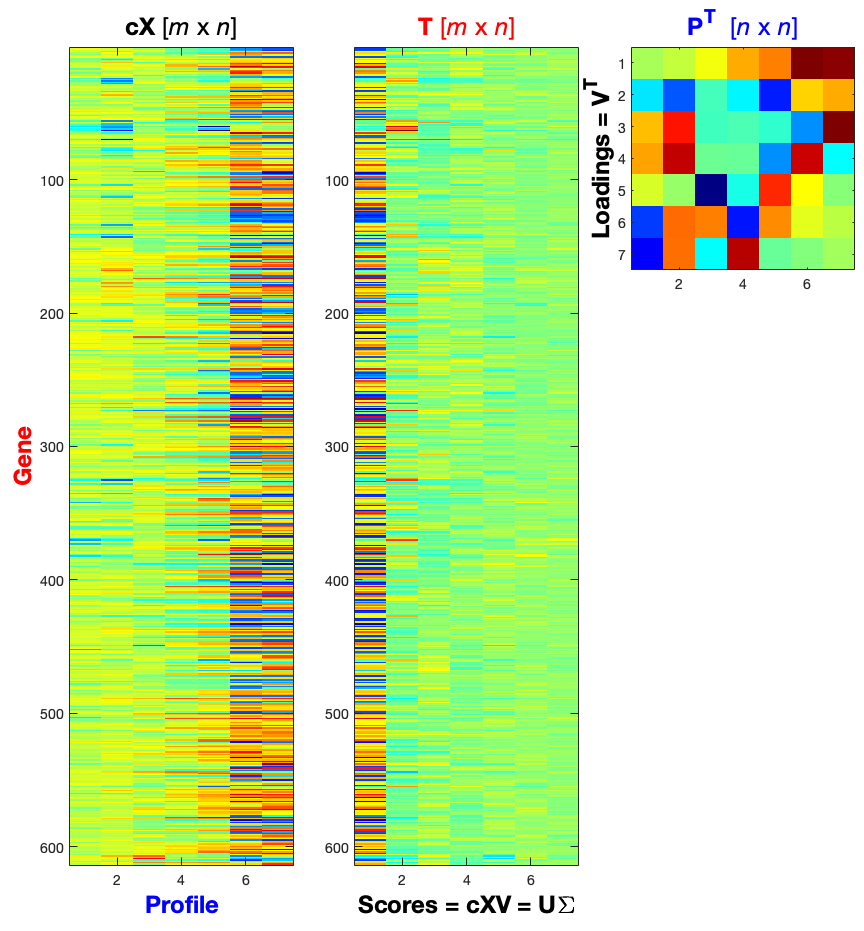
This is nothing other than a PCA of **c*X***:

***T*** = score matrix

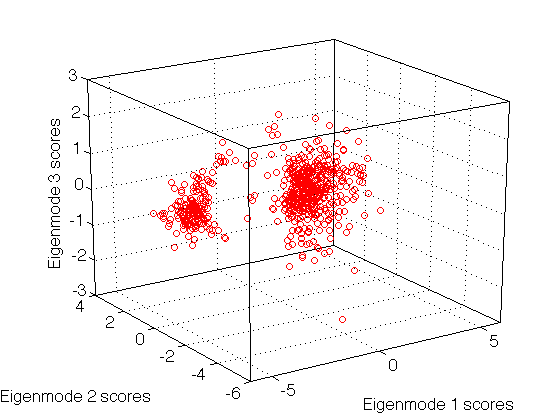
***P*** = loading matrix

Since:

multiplying both side by ***V*** on the right:

Thus, the rows of the product matrix are the coordinates (*scores*)of the transcriptional responses of each gene in the 7 principal components (*eigengenes)* ***v****k* axes of the *row space* of ***X***.

It follows that if we plot the first 3 columns of the matrix against each other, genes that have similar eigen transcriptional responses (similar contributions from the top 3 *eigengenes*) will have similar coordinates and thus will appear clustered. This type of plot can be considered as a projection of the entire genome transcriptional response onto the first 3 *eigengenes* (out of a total of 7 *eigengenes*).

 [U,S,V] = svd(data,'econ');

XV = data\*V;US = U\*S;

figure; plot3(XV(:,1),XV(:,2),...

XV(:,3),'ro')

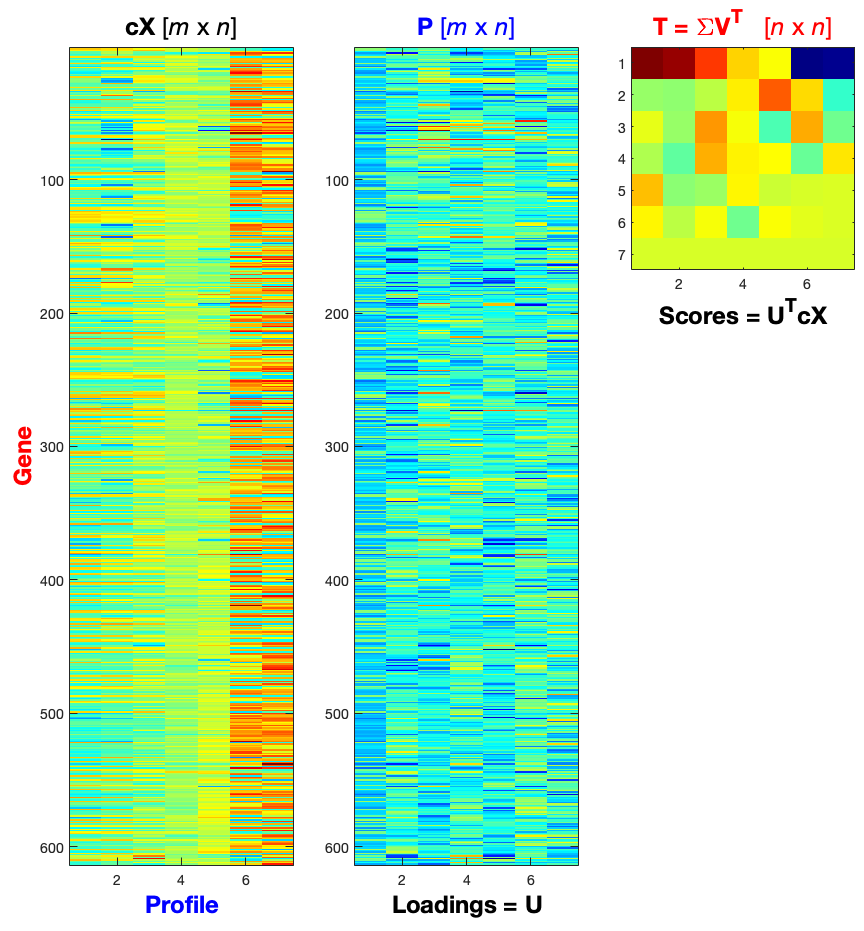
xlabel('Eigengene 1 scores ')

ylabel('Eigengene 2 scores ')

zlabel('Eigengene 3 scores ')

xlim([-6 6]),grid('on'),box('on')

Alternatively we can consider the rows of the microarrays data as the variables (=coordinates), and the columns as the observations. In this case we want to generate a model of gene expression based on the assumption that the expression profiles (the columns in **c*X***) are determined by a combination of different *expression normal modes* (the basis of the *column space* of **c*X***), each mode revealing groups of genes that are activated or inhibited in a coordinated way, possibly because they belong to a metabolic pathway or are involved in a specific cell function. Thus, we want to represent **c*X*** in the column space of **c*X*** (the columns of ***U***).

Upon change of basis, the combinations of modes giving origin to each column of **c*X*** are in the corresponding column of the matrix (*the scores*).

This is nothing other than the PCA of **c*X***:

***T*** = score matrix

***P*** = loading matrix

The biological meaning of the *left singular vectors* ***U*** (*the loadings*) is clarified by considering the SVD decomposition as a two terms product:

data = yeastvalues;

data = data - min(data(:));

[m,n] = size(data);

mean\_data = mean(data,2);

cdata = data - mean\_data;

[U,S,V] = svd(cdata,'econ');

UtX = U'\*U\*S\*V';

SVt = S\*V';

PT = figure;

set(gcf,'Unit','Normalized','Position',[0.1 0.1 0.45 0.9])

pos1 = [0.08 0.07 .26 .88]

subplot('Position',pos1)

imagesc(U\*SVt);colormap jet

title('cX \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',24')

xlabel('Profile','FontWeight','Bold','FontSize',24,'Color','Blue'),

ylabel('Gene','FontWeight','Bold','FontSize',24,'Color','Red')

pos2 = [0.41 0.07 .26 .88]

subplot('Position',pos2)

imagesc(U);colormap jet

title('P \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',24','Color','Blue')

xlabel('Loadings = U','FontWeight','Bold','FontSize',24,'Color','Black')

pos3 = [0.73 0.71 .26 .24]

subplot('Position',pos3)

imagesc(SVt);colormap jet

title('T = \SigmaV^T \rm[\itn \rmx \itn\rm]','FontWeight','Bold','FontSize',24,'Color','Red')

xlabel('Scores = U^TcX','FontWeight','Bold','FontSize',24,'Color','Black')

In conclusion, the rationale behind using the SVD remains very simple:

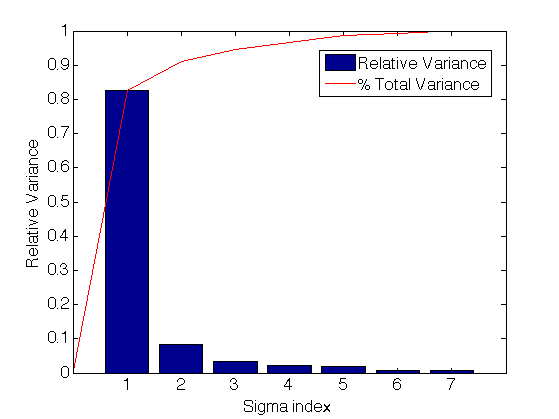
***U*** provides an orthonormal basis for the column space of ***X***.

***V*** provides an orthonormal basis for the row space of ***X***.

Therefore, to represent:

***X*** in an orthonormal basis of its column space calculate (*nxn*)*.* Optionally center *the* ***rows*** of ***X***as for calculating cov = ***XXT***

***X*** in an orthonormal basis of its row space calculate (*mxn*)*.* Optionally center *the* ***columns*** of ***X***as for calculating cov = ***XTX****.*

******There is additional information that can be obtained from the **SVD** of the data matrix ***X***. We can look at a 'scree' plot of the Σ2 values to determine the percentage of variance *explained* by the top *singular vectors*.

D = diag(S).^2;

sumD = cumsum(D);

E = sumD/sumD(end);

relD = D/sumD(end);

figure;bar(relD);hold on

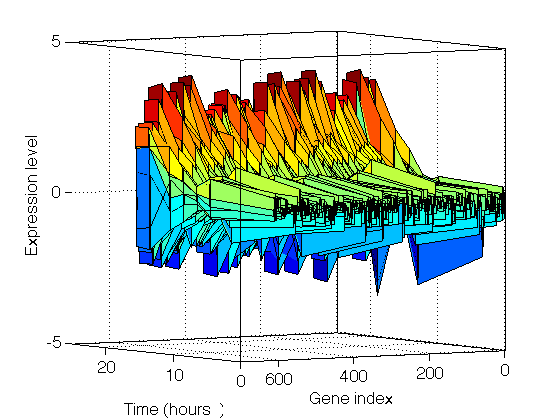
plot([0:7],[0 ; E],'-r')

xlabel('Sigma index')

ylabel('Relative Variance ')

legend('Relative Variance ','% Total Variance','Location','Best')

In this case over 90% of the total variance in trascriptional response/expression profile is explained by the top 2 eigengene/eigenprofile combinations. Therefore we could re-represent our microarray data using only those combinations:



.

data\_red = U(:,1:2)\*S(1:2,1:2)\*V(:,1:2)';

[XI,YI]=meshgrid(times,1:m);

figure;surf(XI,YI,data\_red);

box('on'); xlabel('Time (hours )')

ylabel('Gene index ')

zlabel('Expression level ')

xlim([0 25]),ylim([0 700]),zlim([-5 5])

***Xred(1,2)*** is the best rank-2 approximation of ***X***, which was originally a rank-7 matrix.

A final type of analysis afforded by the SVD is the generation of *correlation plots*. First we calculate the matrix of Pearson correlation coefficients (the correlation matrix) of each gene’s transcriptional response with each of the *eigengenes*: we recall that this is the cosine of the angle between the centered transcriptional responses and the centered *eigengenes*. The correlation matrix has dimensions **614** x **7**, and each column holds the correlation of all the genes transcriptional responses with a given *eigengene*.

mean\_data = mean(data,1);

cdata = data - mean\_data;

[U,S,V] = svd(cdata,'econ');

Since the sign of the singular vectors is arbitrary, if we want to have a consistent result we can change the sign so that the mean is higher than the median:

for n=1:n

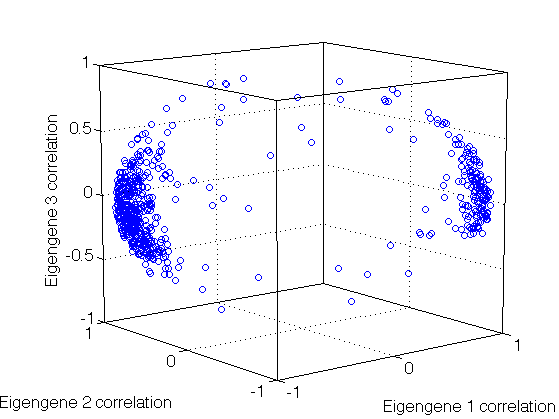
vec\_sign = sign(mean(U(:,n))-median(U(:,n)));

U(:,n)=vec\_sign\*U(:,n);V(:,n)=vec\_sign\*V(:,n);

end

corrGV = corr(data',V);

figure;plot3(corrGV(:,1),corrGV(:,2),corrGV(:,3),'ob')

xlabel('Eigengene 1 correlation ')

ylabel('Eigengene 2 correlation ')

zlabel('Eigengene 3 correlation ')

grid('on')

box('on')

Next, like we did for the projection analysis, we plot the first 3 column vectors against each other: genes that have similar transcriptional response correlations to all three *eigengenes* will appear clustered.

Both the projection plots and the correlation plot show clearly that the 614 yeast genes surveyed in this microarray study cluster in two distinct groups with respect to their transcriptional response. The correlation plot shows clearly that the two types of response are anticorrelated: genes with strong eigengene 1 component have very small eigengene 2 component, and viceversa. The correlation plot also show that both groups of genes spread further out with respect to the 3rd *eigengene* without separating into additional new clusters.

**SPECIAL TOPIC: Independent Component Analysis (ICA) of Microarray Data.**

We have seen how SVD can be used to decompose the space of the transcriptional responses {**g***i*} into linear combinations of *eigengenes*, and the space of the expression profiles {**a***j*} into linear combinations of *eigenassays*. Each *eigenassay* is an *'expression normal mode'* of the genome, with all modes orthogonal to each other and decorrelated.However, we have seen that decorrelation between variables does not automatically guarantee their independence. The identification of modes that are not only decorrelated, but also independent is the goal of *Independent Component Analysis* (**ICA**). ICA is widely used for blind source separation and denoising: in the case of microarray data ICA is used to transform linearly the expression profiles into components with minimal statistical dependencies between them. The interpretation of this analysis is similar to that afforded by SVD: the independent components reflect the *influence* of *unobserved variables* that control the expression of all the genes. Thus, each component defines particular levels of induction or repression of individual genes as *'independent modes of expression*' of the genome. Very often the dominant modes can be related to the activation or repression of a particular cellular function.

In standard ICA analysis the *observed components* ***X*** can be derived from the *independent components* ***S*** by means of a *mixing matrix* ***A***. Traditionally in ICA both the observed and the independent components are represented as *row vectors*, and the ICA solution has the form:

However, microarray data is typically in the form of *m*genes and *n*samples, and thus the *observed components* ***X*** are usually represented as *column vectors*. Therefore, the ICA analysis typically starts with *transposing* the microarray data ***X*** to obtain the solution:

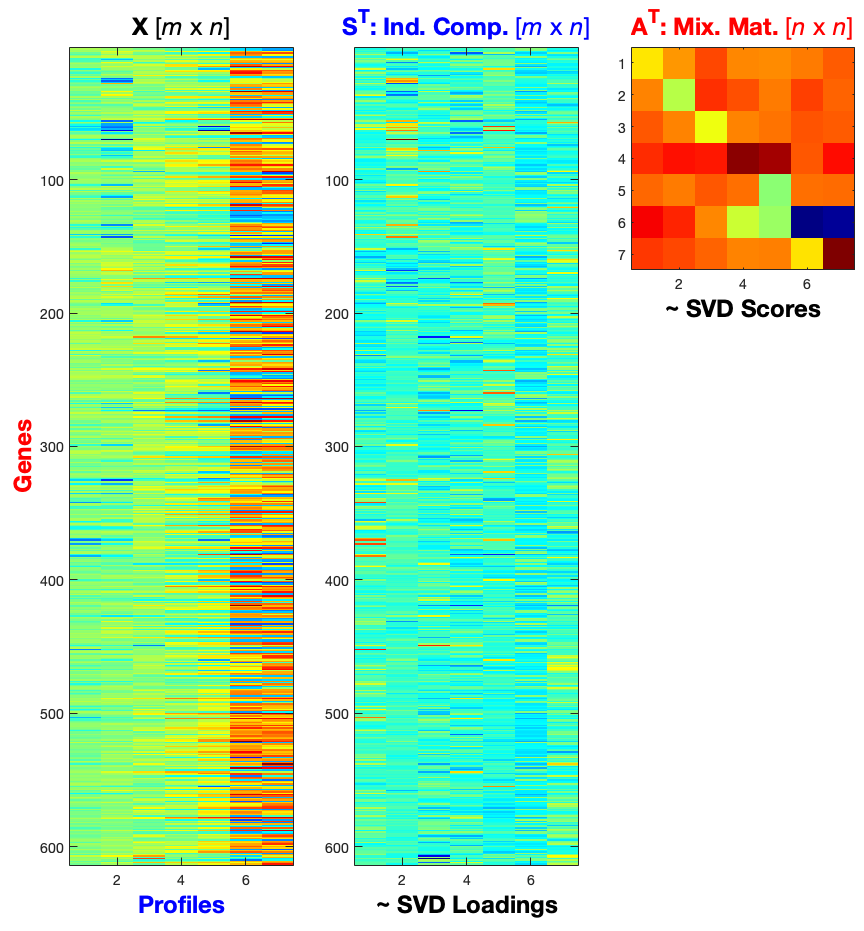
The solution is then converted to standard microarray format by *transposing* again:

This format has the independent components ***S*** and the mixing matrix ***A*** switched in positions and transposed with respect to the traditional ICA expression (***X = AS***):

load filteredyeastdata

data = yeastvalues;

[m,n] = size(data);

% [S1, A1, W1] = fastica(data')

[S1,W1] = RADICAL(data');

A1 = inv(W1);

covS = cov(S’)

Since the sign of the independent components is arbitrary, if we want to have a consistent result we can change the sign so that the mean is higher than the median:

for n = 1:n

vec\_sign = sign(mean(S(n,:),2)-median(S(n,:)))

S(n,:) = vec\_sign\*S(n,:);

A(:,n) = vec\_sign\*A(:,n);

end

X = S’\*A’;

St = S';

At = A';

X = St\*At

This linear decomposition represents a model of gene expression based on the assumption that the ***X*** expression profiles (samples) are determined by a combination of hidden regulatory variables producing different *‘expression independent modes'* (the columns of ***ST***). The *coefficients* for the combinations of these modes giving origin to each ***k*** column in ***X*** are in the corresponding ***k*** column of ***AT***. Each expression mode may reveal a particularly active state of groups of genes belonging to a metabolic pathway or specific cell function. This is very reminiscent of the SVD linear decomposition:

with ICA ***ST*** and ***AT*** corresponding to SVD ***U*** and ***UTX***. The fundamental difference between the two decompositions is that while the SVD ***U*** modes are *orthogonal* and therefore uncorrelated, the ICA modes are also *independent* in addition to being uncorrelated, with the known difference between *uncorrelatedness* and *independence*.

However, while in PCA/SVD *eigen/singular* vectors are ranked based on the magnitude of the corresponding *eigen/singular* values, the independent components identified by ICA are in no specific order. Thus, in order to select which ones to retain and which to discard to filter out noise in the data, two criteria are usually adopted:

It's worth recalling here that the specific contribution of the ***k*** independent component to the entire data set can be calculated as the *dyadic* (outer) product (for which we use here the symbol to distinguish it from the*Kronecker tensor* product):

X\_1 = St(:,1)\*At(1,:);

so that the entire set can be recovered as the sum of all the *dyadic* products:

X\_all = zeros(m,n,n);

for i = 1:n

X\_all(:,:,i) = St(:,i)\*At(i,:);

end

figure;imagesc(X\_all(:,:,3))

Summation along the 3rd dimension

figure;imagesc(sum(X\_all,3))

1. the fraction of total variance in the data associated with a ***k*** independent component (*column*) of ***ST***. This is calculated as the variance ***JA*** of the corresponding ***k*** *row* of the mixing matrix ***A***.

J\_a = var(At)’

2. the *contrast* value ***JS*** of each independent component in ***ST***.Thisis becausewhen compared to noise, the biological components should not only capture a higher amount of the data variance, but also be more *informative* and *less random*. A convenient measure of the information content of a variable is given by its *entropy* ***H***.

According to Shannon’s information theory the *entropy* ***H*** of a discrete random variable ***X*** with possible values {*x*1, …, *xn*} can be calculated from the expression:

where *b* is the base of the logarithm used. Common values of *b* are 2, *e*, and 10. When *b* = 2, the units of entropy are commonly referred to as *bits*. For most applications, the probability of observation is replaced with the observed frequency of or of a *bin* of values centered around in ***X***.

The calculation of entropy for continuous variables is not defined due to the presence of an infinite component, and instead a different measure of entropy is used, called the *differential entropy*, which discards the infinite term that would otherwise appears in using the standard formulation. A fundamental result of information theory is that a continuous unbounded *gaussian* variable has the largest differential entropy among all unbounded random variables of equal variance. Thus, it is customary to use a *contrast* function based on the concept of *negative entropy* ***JS***, defined as the difference between the differential entropy of a purely *gaussian* variable and that of the variable under study:

The differential entropy of a Gaussian distribution, , with mean and standard deviation ( = variance ), is only dependent on :

The differential entropy of the columns of ***ST*** is best calculated using the *kernel density* estimation of differential entropy. The MATLAB function *ksdensity* returns a probability density estimate, *f,* for vector *x*. The estimate is evaluated at equally-spaced points, *xi*, that cover the range of the data in *x*. By default, *ksdensity* estimates the density at 100 points for univariate data. Then, the standard expression for reported above is used to calculate the differential entropy of *x*.

J\_a = var(At)

St\_var = var(St)

All S vectors have variance 1

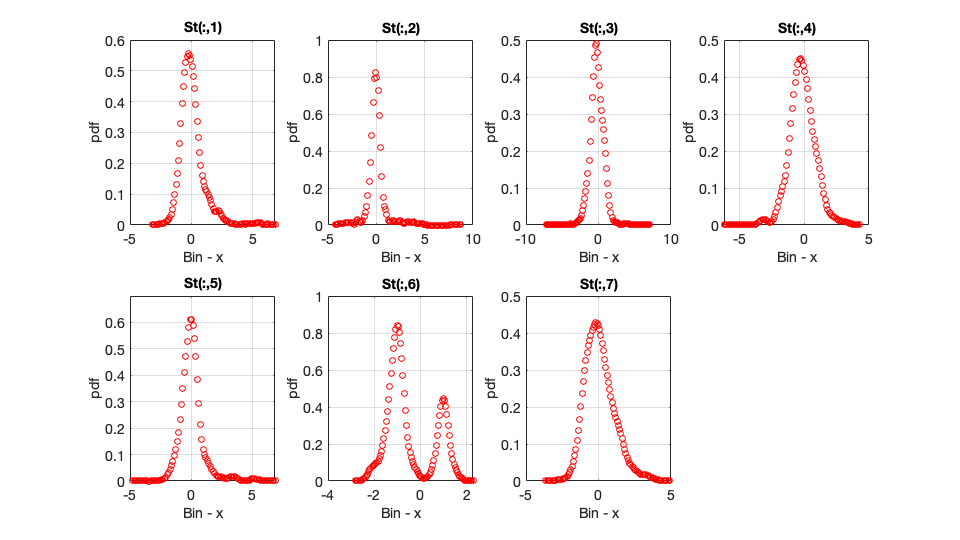
% Entropy of a Gaussian variable only dependent on sigma

H\_gaussian = 0.5\*log2(2\*pi\*exp(1)) + log2(1);

Entropy of ***ST*** columns

Kernel\_Density\_St = figure

set(gcf,'Unit','Normalized','Position',[0 0.5 0.5 0.5]);

for i = 1:size(St,2)

[fi,xi] = ksdensity(St(:,i));

subplot(2,4,i)

plot(xi,fi,'or');grid on

xlabel('Bin - x');ylabel('pdf')

title(['St(:,' num2str(i) ')'])

bw\_i = mean(diff(xi));

ind\_fi = find(fi);

H\_St(i) = - (bw\_i\*fi(ind\_fi))\*log2(fi(ind\_fi)');

end

J\_s = H\_gaussian - H\_St;

Then, to take both properties into account, and without considering a biological meaning behind the exact order, we sort the independent components according to a linear combination of both quantities, scaled by their mean values, with some arbitrary constant ***c*** between 0 and 1:

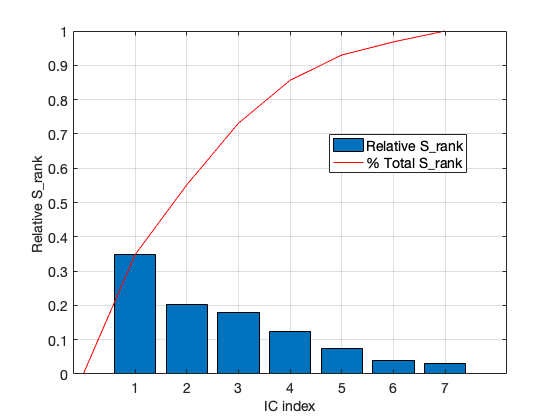
mean\_J\_s = mean(J\_s)

mean\_J\_a = mean(J\_a)

c = 0.5;

S\_rank = c\*J\_s/mean\_J\_s + (1-c)\*J\_a/mean\_J\_a

[~,S\_rank\_ind] = sort(S\_rank,'descend')



We can make a 'scree' plot of the S\_rank to help us decide which IC to retain.

sumJ = cumsum(S\_rank(S\_rank\_ind));

E = sumJ/sumJ(end);

relJ = S\_rank(S\_rank\_ind)/sumJ(end);

figure;bar(relJ);

hold on

plot([0:7],[0 E],'-r')

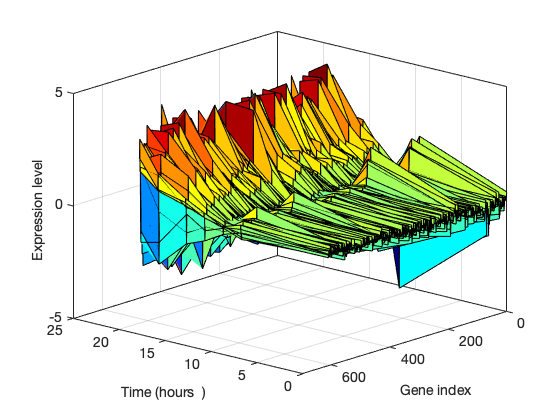
xlabel('IC index')

ylabel('Relative S\\_rank ')

legend('Relative S\\_rank ','% Total S\\_rank','Location','Best')

grid on

In this case the scree plot shows that almost 90% of all the information is contained in the top 4 ranked IC's. Based on this information, we can decide to take only that subset of IC’s to reconstitute the data. For example, to take only components 2,5,6,7 we would sum:

****or we can simply take the top 4 ranked components identified with the scree plot:

data\_recov =...

S(:,S\_rank\_ind(1:4))\*...

A(S\_rank\_ind(1:4),:);

% figure;imagesc(data)

% figure;imagesc(data\_recov)

[XI,YI]=meshgrid(times,1:m);

figure;surf(XI,YI,data\_recov);

box('on'); xlabel('Time (hours )')

ylabel('Gene index ')

zlabel('Expression level ')

xlim([0 25]),ylim([0 700]),zlim([-5 5])

**SPECIAL TOPIC: Non-negative matrix factorization of microarray data**

The non-negative factorization algorithm can be conveniently used to obtain a *lower rank approximation* of the original matrix as , with a diagonal matrix, or constraining . We use our own function *nnmf\_sca* (CHAPTER 21). In this case, since we are looking for a non-negative factorization of the original data we first positivizeit by subtracting the minimal value of the entire data set, and then we carry out a matrix factorization selecting 4 components.

load filteredyeastdata; data = yeastvalues;

data = data - min(data(:)); [m,n] = size(data);

rng(20); k = 4;

dchoice = 'ident'; achoice = 'nneg'; asparse = 0.; schoice = 'random'; maxiter = 10000;

[ A,D,Bt,X\_hat,niter,sse,sse\_diff,tot\_sparseness,sparseness] = ...

nnmf\_sca(data,k,dchoice,achoice,asparse,schoice,maxiter);

ADBt = figure;

set(gcf,'Unit','Normalized','Position',[0.1 0.1 0.4 0.9])

pos0 = [0.07 0.05 .19 .9];subplot('Position',pos0);imagesc(data);colormap jet

title('X \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',22,'Color','Black')

xlabel('Profiles','FontWeight','Bold','FontSize',22,'Color','Blue'),

ylabel('Genes','FontWeight','Bold','FontSize',22,'Color','Red')

pos1 = [0.31 0.05 .19 .9];subplot('Position',pos1);imagesc(X\_hat);colormap jet

title('X\\_hat \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',22,'Color','Black')

xlabel('Profiles','FontWeight','Bold','FontSize',22,'Color','Blue'),

pos2 = [0.55 0.05 .19 .9];subplot('Position',pos2);imagesc(A);colormap jet

title('A \rm[\itm \rmx \itn\rm]','FontWeight','Bold','FontSize',22,'Color','Red')

xlabel('~ SVD Loadings','FontWeight','Bold','FontSize',20,'Color','Black')

pos3 = [0.79 0.8 .19 .15];subplot('Position',pos3);imagesc(Bt);colormap jet

title('Bt \rm[\itn \rmx \itn\rm]','FontWeight','Bold','FontSize',22','Color','Blue')

xlabel('~ SVD Scores','FontWeight','Bold','FontSize',20,'Color','Black')

varA = var(A)

ABt = A\*Bt;

res = data(:)-ABt(:);

sse = res'\*res

% 3D plot

[XI,YI]=meshgrid(times,1:m);

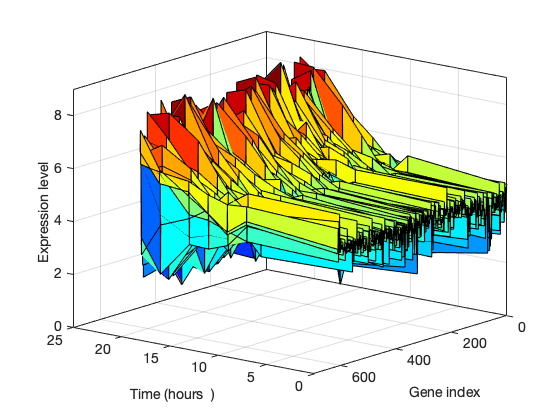
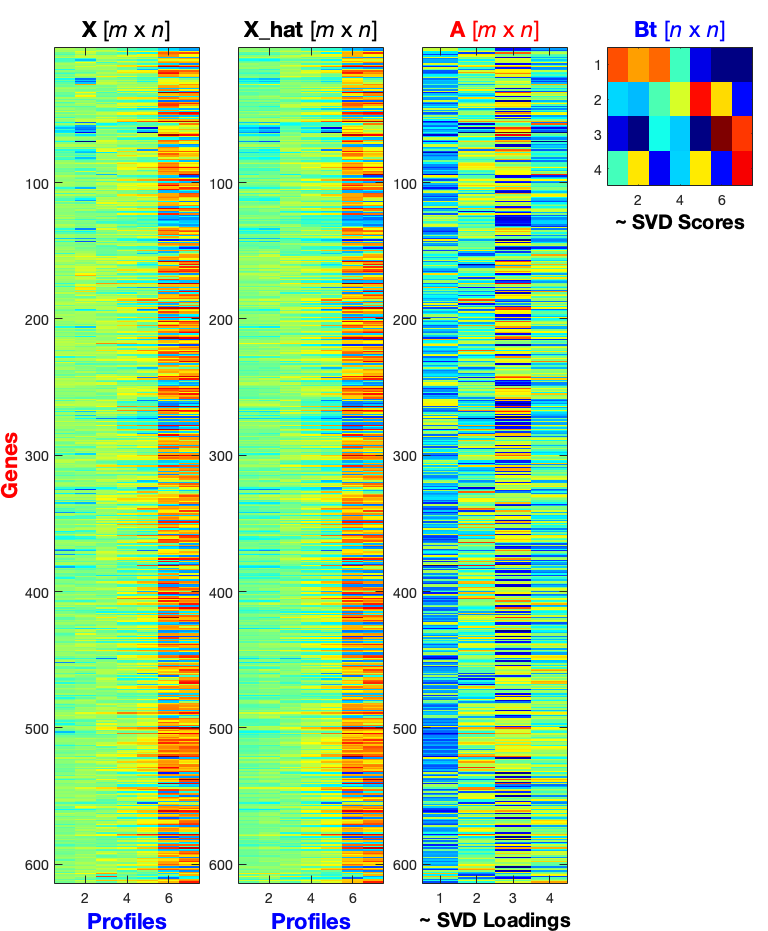
figure;surf(XI,YI,X\_hat);colormap jet

box('on'); xlabel('Time (hours )')

ylabel('Gene index ')

zlabel('Expression level ')

xlim([0 25]),ylim([0 700])



**SPECIAL TOPIC: Principal Component Analysis by Alternating Least Squares (ALS).**

What happens if the microarray data is incomplete (sparse)? In this case both SVD and ICA will fail unless the columns or rows of the data containing missing values are removed from the dataset.

Consider again the *f* component PCA of an *m* x *n* *data matrix* ***Y*** with *n* variables as different columns and *m* observations as different rows. If ***P*** is the *n* x *n loading matrix* and ***T*** is the *m* x *n* *score matrix* , we have:

***Y*** = ***T P****T*

Then, in order to make an *f-components* PCA model ***Yf***of this matrix with *f* < *n*, we calculate the approximation:

***Yf*** = ***Tf Pf****T*

This model depends on identifying the matrices ***T***and ***P*** corresponding to the minimum of the *least squares error* cost function *J*:

where is an optional regularization term (see *Ridge Regression, CHAPTER 6, Special Topic:* Multiple Linear regression and regularization techniques), and the subscript *F* indicates the *Frobenius norm* (2-norm of the vectorized matrix). Because of the term, *J* is not convex and thus not amenable to minimization by an iterative procedure using gradient descent. However, if and are *alternatively* fixed at each iteration, the function becomes convex.

Thus, at each iteration starting from we fix to calculate :

and we fix to calculate :

Both expressions represent applications of the *normal equation* with and being the *unknown* in each expression, and can be augmented with a *ridge* regularization term inside the inverse as:

Then, we simply iterate updating *J* for a set number of cycles or until *J* falls below a threshold value. In the following we use as an example the same data from the previous *Special Topic*.

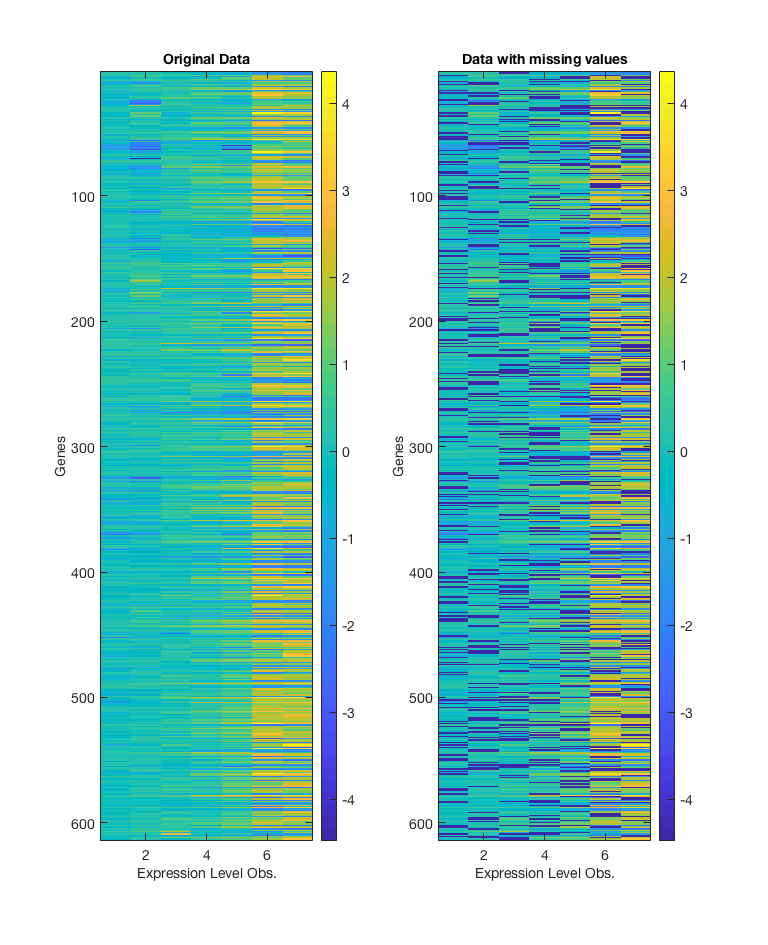
Load microarray data.

load filteredyeastdata

orig\_data = yeastvalues;

[m,n] = size(orig\_data);

r = rank(orig\_data);



Here we remove randomly 20% of the data.

rng(3)

sparsity = 0.2;

D = logical(binornd(1,sparsity,m,n));

sparse\_data = orig\_data;

sparse\_data(D) = NaN;

Y = sparse\_data;

We display both the original data and the data with missing values.

Sparse\_PCA\_1 = figure

set(gcf,'Unit','Normalized','Position',[0 0 0.4 1])

subplot(1,2,1)

imagesc(orig\_data);colorbar;

xlabel('Expression Level Obs.');ylabel('Genes');

title('Original Data')

subplot(1,2,2)

imagesc(Y);colorbar;

xlabel('Expression Level Obs.');ylabel('Genes');

title('Data with missing values')

Random initialization of the scores and coefficients matrices. The number of features (components) selected can't exceed the rank.

[num\_obs,num\_vars] = size(Y);

num\_features = 3;

T = randn(num\_obs, num\_features);

P = randn(num\_vars, num\_features);

I = eye(num\_features);

We center the sparse data, then for convenience in carrying out matrix multiplication in the sparse PCA algorithm we convert all the NaN to 0, keeping track of their positions.

Y\_mean = nanmean(Y);

Y = Y - Y\_mean;

Y(D) = 0;

R = ~D;

Initialization by svd.

[U,S,V] = svd(Y);

T = U\*S;

T = T(:,1:num\_features);

P = V(:,1:num\_features);

I = eye(num\_features);

***Y*** is a *m*x*n* matrix, containing *n* expression levels of *m* genes. ***R*** is a *m*x*n* matrix, where ***R***(i,j) = 1 if and only if the j level of expression of gene i was measured.

Set Regularization lambda.

lambda = 0.2;

niter = 500;

Y\_iter = Y;

Y\_iter\_all = zeros(num\_obs,num\_vars,niter);

P\_all = zeros(num\_vars, num\_features, niter);

T\_all = zeros(num\_obs, num\_features, niter);

for i = 1:niter

T = Y\_iter\*P/(P'\*P + lambda\*I);

P = Y\_iter'\*T/(T'\*T + lambda\*I);

P\_all(:,:,i) = P;

T\_all(:,:,i) = T;

Y\_iter = T\*P';

Y\_iter\_all(:,:,i) = Y\_iter;

% Update U and V as orthogonal matrices

[U,S,V] = svd(Y\_iter);

T = U(:,1:num\_features)\*S(1:num\_features,1:num\_features);

P = V(:,1:num\_features);

E = (Y - Y\_iter).\*R;

e = E(:);

t = T(:);

p = P(:);

J(i) = e'\*e + lambda\*(t'\*t + p'\*p);

end

figure;plot(J(1:end));grid on

xlabel('No. of iterations');ylabel('J')

[~,best\_iter\_ind] = min(J)

T = T\_all(:,:,best\_iter\_ind);

P = P\_all(:,:,best\_iter\_ind);

Y\_iter = Y\_iter\_all(:,:,best\_iter\_ind);

Data reconstruction using non-orthonormal coefficients

P'\*P

Y\_cen\_rec = T \* P';

Y\_rec = Y\_cen\_rec + Y\_mean;

We display both the original data and the reconstructed data

Y\_reconstruct\_1 = figure

set(gcf,'Unit','Normalized','Position',[0 0 0.4 1])

subplot(1,2,1)

cmin = min(orig\_data(:));

cmax = max(orig\_data(:));

imagesc(orig\_data);colorbar;

xlabel('Expression Level Obs.');ylabel('Genes');

title('Original Data')

subplot(1,2,2)

imagesc(Y\_rec,[cmin cmax]);colorbar;

xlabel('Expression Level Obs.');ylabel('Genes');

title('Data with missing values replaced')

Correlation between original and reconstructed data for the missing

entries.

corr\_orig\_rec\_obs = corr(orig\_data(R),Y\_rec(R))

corr\_orig\_rec\_mis = corr(orig\_data(D),Y\_rec(D))

Data reconstruction using orthonormal coefficients. First, we obtain and orthogonal basis for the coefficients matrix:

P\_orth = orth(P);

Given that ***TP****T*= ***Y*** transposing both side we obtain ***PT****T*= ***Y****T*. Applying the normal equation we derive:

***(P****T****P)T****T*= ***P****T****Y****T*

***T****T*= ***(P****T****P)-1P****T****Y****T*

***T*** = ***YP(P****T****P)-1***

Since ***P*** is now orthogonal ***P****T****P*** is just the identity matrix:

***T*** = ***YP***

T\_orth = Y\_cen\_rec\*P\_orth;

Y\_orth\_rec = T\_orth \* P\_orth' + Y\_mean;

Correlation between original and reconstructed data for the missing entries.

corr\_orig\_rec\_obs = corr(orig\_data(R),Y\_orth\_rec(R))

corr\_orig\_rec\_mis = corr(orig\_data(D),Y\_orth\_rec(D))

MATLAB has its own program for matrix completion by ALS.

[P\_matlab,T\_matlab,latent\_matlab,~,~,mu\_matlab] = pca(sparse\_data,'algorithm','als');

Y\_matlab = T\_matlab(:,1:num\_features)\*P\_matlab(:,1:num\_features)' + mu\_matlab;

corr\_local\_als\_matlab\_als = diag(corr(Y\_orth\_rec,Y\_matlab))

corr\_orig\_matlab\_als = corr(orig\_data(R),Y\_matlab(R))

corr\_orig\_matlab\_als\_mis = corr(orig\_data(D),Y\_matlab(D))

**SPECIAL TOPIC: Principal Component Analysis of Sparse Data by Collaborative Filtering.**

*Collaborative filtering* with reconstruction of missing data can be considered as a variation of *Alternating Least Squares* using the minimization of a cost function *J*.

In this case the strategy to handle the missing data in is based on an optimization procedure in which we derive both a *scores matrix* and a *loading matrix* (where is the number of *components* or *features* selected) simultaneously as refined parameters based on the definition of a *loss matrix* :

where such that if and only if the *j* level of expression of gene *i* was measured. We define the *cost function* *J* to be minimized as:

where , , and , are the column vectors derived from linearizing , , and , respectively, and is a *Lagrange multiplier* for the regularization term (see CHAPTER 6, Special Topics: Multiple Linear regression and regularization techniques).

Minimization of this cost function is facilitated by the simple form of the two partial derivatives of with respect to the *scores* and the *loadings*:

function [J, grad] = sPCA\_CostFunc(params, Y, R, num\_obs, num\_vars, ...

num\_features, lambda)

% Unfold the scores and loading matrices from params

scores = reshape(params(1:num\_obs\*num\_features), num\_obs, num\_features);

coeffs = reshape(params(num\_obs\*num\_features+1:end), ...

num\_vars, num\_features);

J = 0;

scores\_grad = zeros(size(scores));

coeffs\_grad = zeros(size(coeffs));

res = (scores\*coeffs' - Y).\*R;

J = (res(:)'\*res(:) + lambda\*(coeffs(:)'\*coeffs(:) + scores(:)'\*scores(:)))/2;

scores\_grad = res\*coeffs + lambda\*scores;

coeffs\_grad = res'\*scores + lambda\*coeffs;

% Return the gradients

grad = [scores\_grad(:); coeffs\_grad(:)];

end

As an example, we use the same microarray data from the previous *Special Topic.*

load filteredyeastdata

orig\_data = yeastvalues;

[m,n] = size(orig\_data);

Here we remove randomly 20% of the data

D = logical(binornd(1,0.2,m,n));

Y = orig\_data;

Y(D) = NaN;

Like in standard PCA we can center the data removing the mean of all the observations. However, identical results are usually obtained without centering.

Y\_mean = nanmean(Y);

Y\_cent = Y - Y\_mean;

For convenience in carrying out the matrix multiplications used in the optimization of the cost function we convert all the *NaN* to zero keeping track of their positions.

Y\_cent(D) = 0;

R = Y\_cent ~= 0;

Again, here contains *n* expression levels of *m* genes, and is such that if and only if the *j* level of expression of gene *i* was measured.

We start by initializing randomly the *scores* and *loadings* matrices, and by selecting the number of principal components (features) we want to use. We recall here that this number cannot be higher than the rank of the data.

[num\_obs,num\_vars] = size(Y\_cent);

r = rank(Y\_cent);

num\_features = r;

Set Initial Parameters (scores, coefficients)

scores = randn(num\_obs, num\_features);

coeffs = randn(num\_vars, num\_features);

The scores and loading matrices are vectorized and concatenated for use in the unconstrained minimization of the cost function (see CHAPTER 17: Unconstrained minimization) using either *fminunc* (MATLAB) or *fmincg*, a highly optimized *conjugate gradients* minimizer originally written by Carl Edward Rasmussen (as *minimize* in the [gpml matlab/octave toolbox](http://www.gaussianprocess.org/gpml/code)) and further modified by Andrew Ng.

initial\_parameters = [scores(:); coeffs(:)];

In this case the amount of missing data is relatively small and we set the regularization to 0.

lambda = 0;

Set options for MATLAB *fminunc*

options = optimoptions('fminunc','SpecifyObjectiveGradient',true, ...

'MaxIterations', 10000,'Display','iter','Algorithm','quasi-newton');

[parameters, J, exit\_flag] = fminunc(@(t)(sPCA\_CostFunc(t,Y\_cent,R, num\_obs,num\_vars, ...

num\_features, lambda)), parameters, options);

Or alternatively, set options for Rasmussen/Ng *fmincg*

options = optimset('GradObj', 'on', 'MaxIter', 10000);

parameters = fmincg (@(t)( sPCA\_CostFunc(t, Y\_cent, R, num\_obs, num\_vars, ...

num\_features, lambda)), ...

initial\_parameters, options);

Unfold the returned parameters back into scores and coefficient matrices.

scores = reshape(parameters(1:num\_obs\*num\_features), num\_obs, num\_features);

coeffs = reshape(parameters(num\_obs\*num\_features+1:end), ...

num\_vars, num\_features);

The data can now be reconstructed from the refined *scores* and *loadings* matrices:

Y\_rec\_cent = scores \* coeffs';

Y\_rec = Y\_rec\_cent + Y\_mean;

Notice that in this case the *loadings* (coefficients) matrix is not orthonormal as that we would get from standard PCA. However, we can orthogonalize the matrix and then derive the corresponding *scores* matrix:

Orthonormal *loadings* matrix:

coeffs\_orth = orth(coeffs);

coeffs\_orth'\*coeffs\_orth

ans = 1.0000 0.0000 0.0000 -0.0000 0.0000 -0.0000 -0.0000

0.0000 1.0000 0.0000 0.0000 -0.0000 -0.0000 -0.0000

0.0000 0.0000 1.0000 -0.0000 0.0000 0.0000 -0.0000

0.0000 0.0000 -0.0000 1.0000 -0.0000 -0.0000 0.0000

0.0000 -0.0000 0.0000 -0.0000 1.0000 0.0000 0.0000

0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 -0.0000

0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000

Given that ***TP'*** = ***Y*** , transposing both side we obtain ***PT'*** = ***Y'***. Applying the normal equation we derive (***P'P***)***T'*** = ***P'Y'*** from which ***T'*** = (***P'P***)\***P'Y'***. Since is now orthogonal, is the identity matrix and ***T*** = ***YP****.*

scores\_orth = Y\_rec\_cent \* coeffs\_orth;

We can check that reconstruction using an orthonormal coefficient matrix gives the exact same result (within machine numerical precision) as the reconstruction using a non-orthonormal coefficient matrix:

Y\_orth\_rec = scores\_orth \* coeffs\_orth' + Y\_mean;

res = sum(Y\_rec(:) - Y\_orth\_rec(:))

res = 4.3507e-15

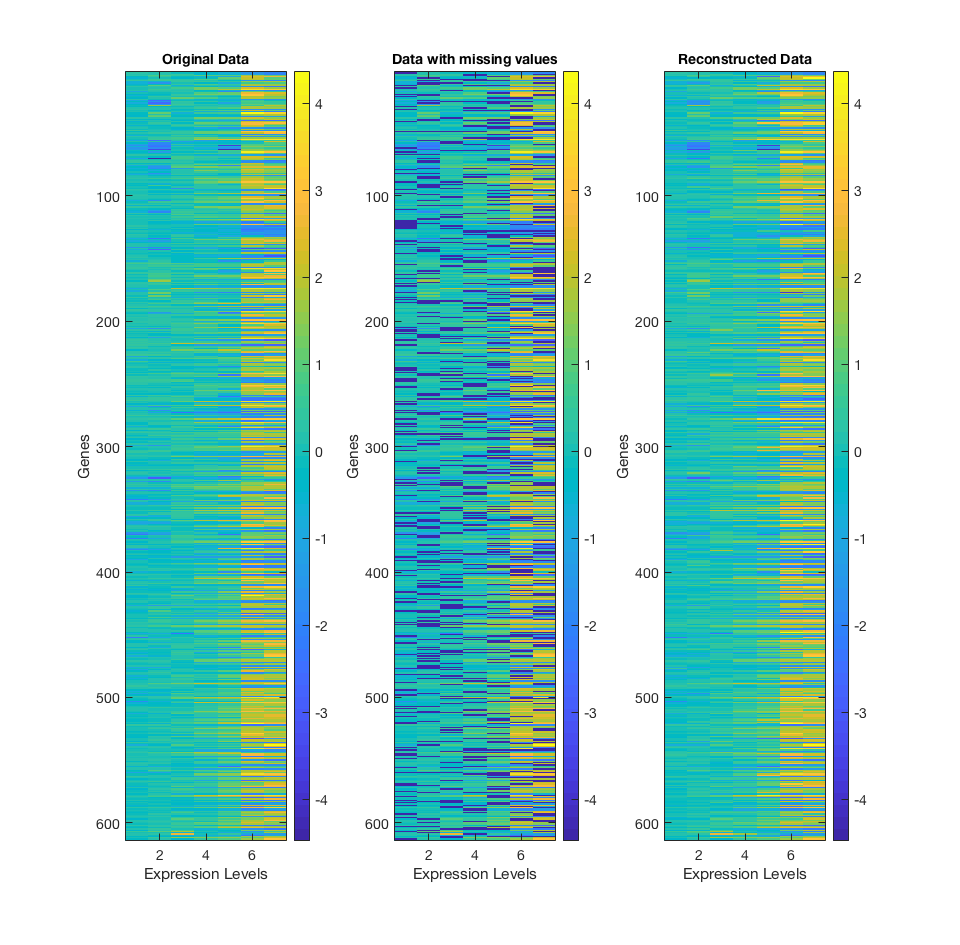
Also notice how the variance of the new *scores* matrix corresponding to the orthogonalized *loadings* matrix is the same (also within machine numerical precision) as the *eigenvalues* of the covariance matrix of the reconstructed data using a non-orthogonal *loadings* matrix:

scores\_orth\_var = var(scores\_orth)

scores\_orth\_var = 6.8001 0.9055 0.3569 0.2588 0.2176 0.0582 0.0419

[Coeff,Latent] = pcacov(cov(Y\_rec));

Latent = 6.8020 0.9058 0.3572 0.2586 0.2160 0.0579 0.0414

We can now compare the original data, the data with missing values, and the reconstructed data:

Sparse\_PCA\_2 = figure

set(gcf,'Unit','Normalized','Position',[.1 0 0.5 1])

subplot(1,3,1)

imagesc(orig\_data);colorbar;

xlabel('Expression Levels');ylabel('Genes');

title('Original Data')

cmin = min(orig\_data(:))

cmax = max(orig\_data(:))

subplot(1,3,2)

imagesc(Y,[cmin cmax]);colorbar;

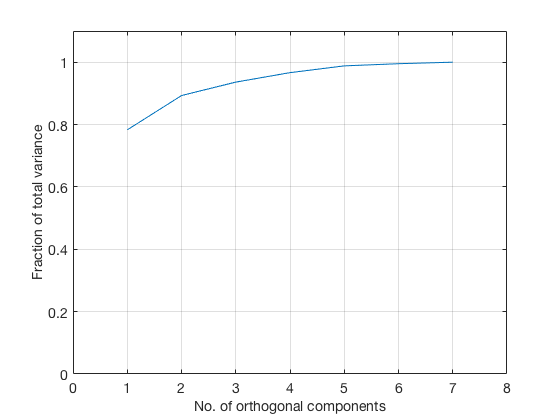
xlabel('Expression Levels');ylabel('Genes');

title('Data with missing values')

subplot(1,3,3)

imagesc(Y\_rec,[cmin cmax]);colorbar;

xlabel('Expression Levels');ylabel('Genes');

title('Reconstructed Data')

We can also calculate the contribution of the various components to the total variance of the data:

fractional\_var = …

cumsum(scores\_orth\_var')/sum(scores\_orth\_var')

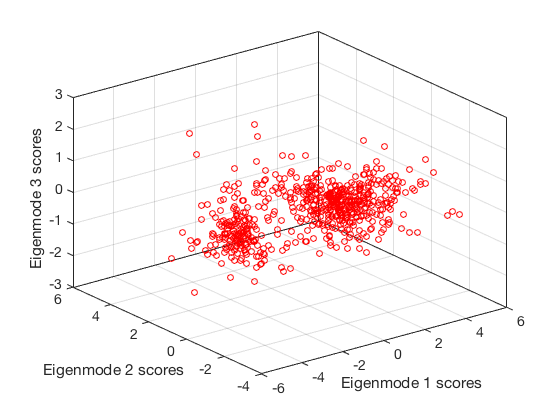
plot(fractional\_var);grid on

xlim([-1 num\_features+1]);ylim([0 1.1])

xlabel('No. of orthogonal components')

ylabel('Fraction of total variance')

From which it is apparent that the first 3 components account for almost 95 % of all the variance. Using those 3 components we can obtain starting from incomplete data a plot of the clustering of the different genes essentially identical to the one with obtained with the SVD of complete data:



figure; plot3(scores\_orth(:,1),scores\_orth(:,2),...

scores\_orth(:,3),'ro')

xlabel('Eigenmode 1 scores ')

ylabel('Eigenmode 2 scores ')

zlabel('Eigenmode 3 scores ')

xlim([-6 6]),grid('on'),box('on')

Finally, we can calculate the correlation between original and reconstructed data for both the observed and the missing entries.

corr\_orig\_rec\_obs = corr(orig\_data(R),Y\_rec(R))

corr\_orig\_rec\_mis = corr(orig\_data(D),Y\_rec(D))

corr\_orig\_rec\_obs = 1.0

corr\_orig\_rec\_mis = 0.7669

Increasing the regularization parameter will have the effect of decreasing the correlation between original and reconstructed data for the observed entries and increasing the correlation between original and reconstructed data for the missing entries.

Sparse PCA reconstruction can be carried out also selecting only the desired components from the start. For example, in the following code we decide to use only 3 components in the reconstruction:

Number of features (principal components)

num\_features = 3;

Initial Parameters

scores = randn(num\_obs, num\_features);

coeffs = randn(num\_vars, num\_features);

initial\_parameters = [scores(:); coeffs(:)];

Regularization lambda

lambda = 0.3;

Options for *fmincg*

options = optimset('GradObj', 'on', 'MaxIter', 10000);

parameters = fmincg (@(t)(sPCA\_CostFunc(t, Y\_cent, R, num\_obs, num\_vars, ...

num\_features, lambda)), ...

initial\_parameters, options);

Unfold the returned parameters back into scores and coefficient matrices

scores\_f = reshape(parameters(1:num\_obs\*num\_features), num\_obs, num\_features);

coeffs\_f = reshape(parameters(num\_obs\*num\_features+1:end), ...

num\_vars, num\_features);

Reconstruction using non-orthonormal coefficient matrix

Y\_rec\_f = scores\_f \* coeffs\_f' + Y\_mean;

Reconstruction using orthonormal coefficient matrix

Y\_rec\_cent\_f = scores\_f \* coeffs\_f';

coeffs\_orth\_f = orth(coeffs\_f);

scores\_t\_f = (coeffs\_orth\_f'\*coeffs\_orth\_f)\coeffs\_orth\_f'\*Y\_rec\_cent\_f';

scores\_orth\_f = scores\_t\_f';

scores\_orth\_f\_var = var(scores\_orth\_f)

Y\_orth\_rec\_f = scores\_orth\_f \* coeffs\_orth\_f' + Y\_mean;

Display reconstruction

Sparse\_PCA\_3 = figure

set(gcf,'Unit','Normalized','Position',[.1 0 0.5 1])

subplot(1,3,1)

imagesc(orig\_data);colorbar;

xlabel('Expression Levels');ylabel('Genes');

title('Original Data')

cmin = min(orig\_data(:))

cmax = max(orig\_data(:))

subplot(1,3,2)

imagesc(Y,[cmin cmax]);colorbar;

xlabel('Expression Levels');ylabel('Genes');

title('Data with missing values')

subplot(1,3,3)

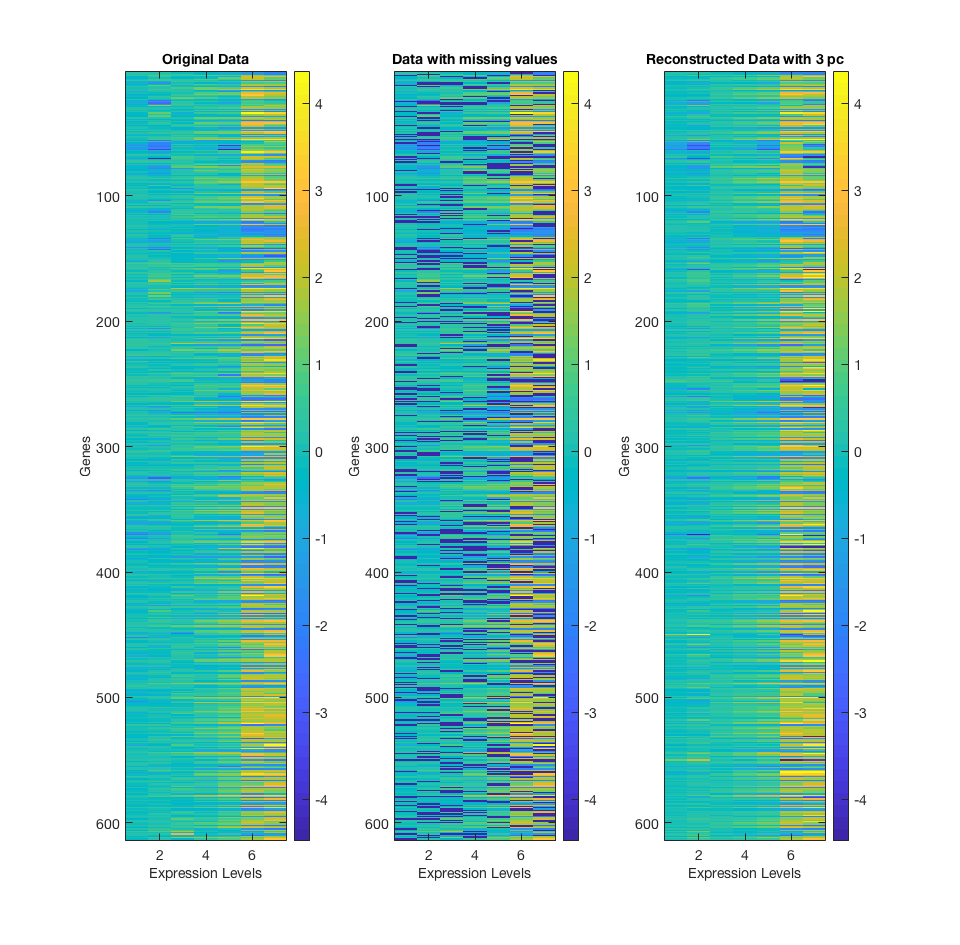
imagesc(Y\_rec\_f,[cmin cmax]);colorbar;

xlabel('Expression Levels');ylabel('Genes');

title('Reconstructed Data with 3 pc')

Alternatively, and perhaps more effectively, completion and low rank approximation of a matrix can be achieved by first using all the features up to the rank to carry out completion, and then by using SVD for low rank approximation.

Finally, it should be noted that while *collaborative filtering* was shown in this section as applied to microarray data, it can be used in all cases in which a data matrix is sparse. A famous application is the *recommender* algorithm used by movie channels to recommend movies to a viewer. In this case the data matrix has the movies as columns, the viewers as rows, and the ranking given by each viewer to individual movies as the entries in the (very sparse) matrix. After completing the data matrix, a ranking for each movies (not yet seen) is assigned to each viewer. Movies with the highest ranking are recommended to each viewer.



**PRACTICE**

**1.** Load the yeast microarray data described in this chapter:

a. Use traditional PCA to carry out the cluster analysis described in this chapter.

b. Using SVD identify which genes belong to each cluster of transcriptional responses (use the tools we used to cluster frames of a molecular dynamics trajectory).

c. Produce reduced data sets containing the transcriptional responses of the genes in each cluster.

**2.** Save the results from the SVD run. Repeat the analysis, this time using ICA:

a. Identify the genes corresponding to the top 20% expression levels in both types of analysis using only the 1st left singular vector and the IC vector with the best Srank.

b. Find which genes are in both sets of top expression levels (Hint: use the *'find'* and *'intersect'* functions or work with logical arrays).