# **Principal component analysis.**

We now ask the question: how do we use *eigenvalues* and *eigenvector*? There are countless applications in various field of science, but here we will focus on some of particular importance for biochemists. The first one is the technique of statistical analysis known as '*Principal Component Analysis*' or '*PCA*'.

This technique can be conveniently used to reduce the dimensionality of data. For example, if the data is spread over 3 dimensions, but most of the information is contained in only 2 dimensions, we can use PCA to represent the data using only the most informative dimensions. This is possible because the different dimensions are effectively decoupled from each other, such that only some of them can be taken and the others discarded. There are various way to carry out a PCA analysis: here we will discuss the algorithm that makes use of the *covariance matrix*.

Let's consider a *random vector*: this is a *column* vector containing *m* variables X1, X2, X3, ...Xm. Furthermore, we have *n* observations of this vector collected in a matrix ***X***:

*n**observations*

*m**variables*

If we have a sufficiently large number of observations and the different observed values of each variable are normally distributed around the mean, the *mean* of each row can be considered as the *expected value* ***E*** of each variable. If this is true, and we remove the mean of each row from every element of the row, we obtain a matrix of *errors* (deviations of the observed values from the expected value); however in other cases the multiple observations may represent a *time series* or changes of a physico-chemical quantity in response to a different environmental factor (e.g., the spectral changes elicited by the addition of increasing quantities of a ligand to a protein). Thus, in the more general case, rather than defining the matrix originating from the subtraction of the mean as a matrix of *errors*, we refer to it as a matrix ***cX*** of *centered data.*

nvars = 10;

nobs = 10;

X = rand(nvars,1)

X\_mat = zeros(nvars,nobs)

for i = 1:nobs

X\_mat(:,i) = normrnd(X,0.05);

end

X\_mean = mean(X\_mat,2)

cX\_mat = X\_mat - X\_mean

We recall here the definition for the *variance* of vector X1 of observations:

where are the *degrees of freedom* (= number of dimensions) of the system. In matrix form we have:

X1\_var = cX\_mat(1,:)\*cX\_mat(1,:)'/(nobs-1)

X1\_var = var(cX\_mat(1,:))

Likewise, we define the *covariance* between variable X1 (row 1) and X2 (row 2) as:

or in matrix form:

X12\_var = cX\_mat(1,:)\*cX\_mat(2,:)'/(nobs-1)

Then, we can produce a new *m* x *m*matrix ***C*** =(***cXcXT****)/nobs*in which all the different dot products between the rows of the centered matrix ***cX*** are included:

cov\_X = cX\_mat\*cX\_mat'/(nobs-1)

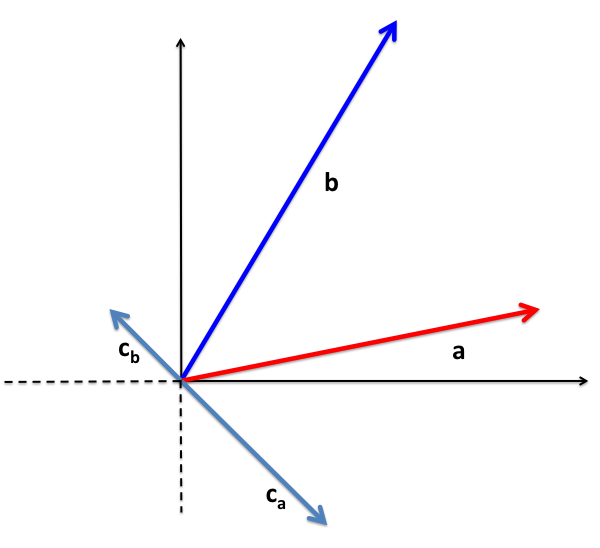
In MATLAB we can calculate directly the covariance matrix using the function *cov*, but since MATLAB uses the convention of keeping the observations (rather than the variables) in different rows, we have to take the transpose first:

cov\_X = cov(cX\_mat') using the *cov* function

cov\_X = cov(cX\_mat',1) using the *cov* function with nobs = n instead of nobs = n-1

Notice how in the *covariance* matrix all the elements in the diagonal represent the *variance* of each variable (that is the *covariance* of a variable with itself; e.g. ), while the off-diagonal terms represent the *covariance* between two variables (e.g. ).

We can use a simple 2D example to understand better the geometric meaning of *variance* and *covariance*. Let's consider 2 vectors:



We can calculate the angle θ between them as:

a = [5 3]', b = [1 5]'

a'\*a, sqrt(a'\*a)

norm(a), norm(b)

cos\_theta = (a'\*b)/(norm(a)\*norm(b))

theta = (180/pi)\*acos(cos\_theta)

theta = acosd(cos\_theta)

and we can calculate the centered vectors:

and the cosine of the angle θc between the centered vectors:

ca = a - mean(a);

cb = b - mean(b);

cos\_theta = (ca'\*cb)/(norm(ca)\*norm(cb))

corr\_ab = corr(a,b)

theta = acosd(cos\_theta)

this value of *cos(θc)* represents the *correlation* between two column vectors, ***a*** and ***b***, while the simple dot product without normalization is proportional (by 1/*nobs*) to either the *variance*:

or the *covariance*:

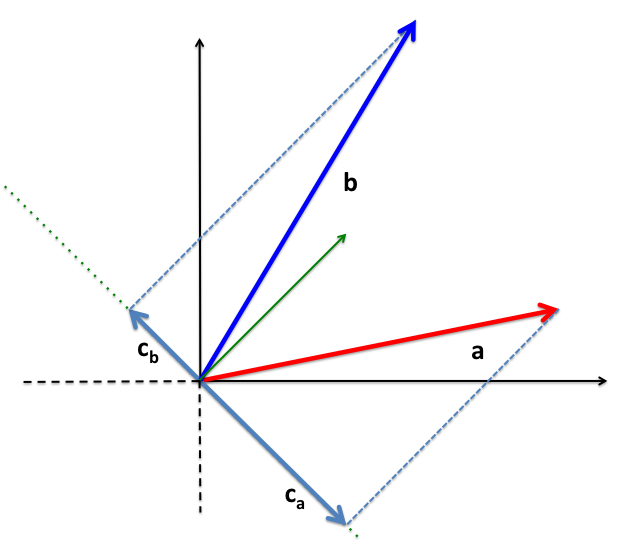
cov([a a],1) covariance

var(a,1) variance

(ca'\*ca)/2 variance

We can see that while the *variance* tells us about the '*spread*' of the observed components of a vector with respect to the expected value, the *covariance* tells us how two vectors change from one component to the next. For example:

Thus, since in moving from the 1st to the 2nd term the ***a*** vector goes down in value and the ***b*** vector goes up in value, they are *anticorrelated* (*corr(a,b)* = -1), and their *covariance* is negative. Notice how covariance and correlation are simply related by a normalization operation. In fact the statistical definition of correlation is:

which is exactly what we defined above as the cosine of the angle between the two centered vectors.

Finally notice how the centered vectors ***ca*** and ***cb*** represent the projection of ***a*** and ***b*** on a line perpendicular to any vector that bisects the 1st quadrant of the plane (for example ,shown in green in the drawing). Therefore, that line is in the *left null space* of the column vector , which we can easily identify as . The *projection* matrix onto this subspace is:

from which we derive:

Since projection matrices are *symmetric* and *idempotent* we get:

These alternative expressions for variance, covariance, and correlation can be useful in applications in which the vectors ***a*** and ***b*** are themselves derived from projection operations. It is important to remember that the basis for the subspace onto which the original vectors are projected to obtain the *centered* vectorsis always the *left null space* (of dimensions *m-1*)of the column vector that *equisect* the positive ortant of the original space. Examples of the centering subspaces if the original vectors belong to **R*3***, or **R*4*** are given in the Table below:

|  |  |  |  |
| --- | --- | --- | --- |
| original space  dim = *m* | equisecting vector  (*m x 1***)** | rank | basis of the centering subspace  (*m x m-1***)**,dim = *m-1* |
|  |  |  |  |
| **R*3*** |  | 1 |  |
|  |  |  |  |
| **R*4*** |  | 1 |  |
|  |  |  |  |

Thus, the operation of *centering* is itself a type of *dimensionality reduction*. It is important to understand that it is the reduction in the dimensions of the *row space* of the *m* x *n* data matrix (*,* that provides the basis for decreasing by 1 the number of degrees of freedom used to calculate values of variances and covariances from the matrix of centered data. For example, consider the dataset ***A*** of observations and its conversion to centered data ***cA***:

***A*** *mean(****A****)* ***cA***

A = [2 4 6 8;2 1 1 4; 2 3 2 5;2 2 4 4 ;2 4 2 4];

meanA = mean(A,2);

cA = A - meanA;

rank(A)

rank(cA)

Notice that *rank(****A****)* = 4, but *rank(****cA****)* = 3, because the *row space* of ***cA*** has only 3 dimensions. In this case, since the centered matrix ***cA*** has rank 3 and *n-1<m*the covariance matrix ***cAcAT*** (*mxm* ***=*** *5x5*) has also rank 3 and therefore is only *positive semidefinite*. However, in most cases the number of observations is much larger than the number of variables (*n-1>>m*), in which case most likely *r* = *m* and the covariance matrix (*mxm* ***=*** *5x5*) is *positive definite*. We can verify with the data matrix used in the initial example:

rank(X\_mat)

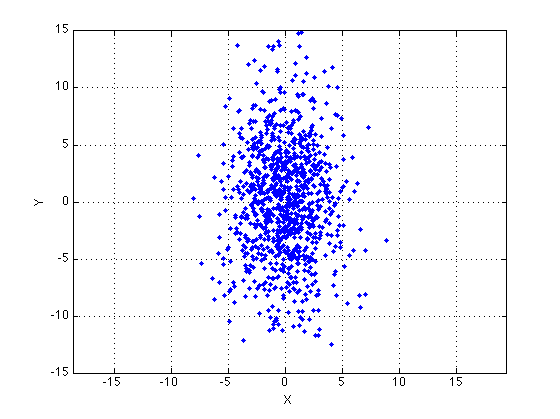
rank(cX\_mat)

rank(cov\_X)

The Choleski factorization produces an upper triangular matrix ***R*** satisfying the equation ***RTR*** = ***A***, provided that ***A*** is *positive definite*. For this reason, this type of factorization is often used to check for the positive definiteness of a covariance matrix:

R = chol(cov\_X); R’\*R

However, it is important to keep in mind that a matrix may *appear* positive definite only because one or more eigenvalues are *numerically* strictly different from 0, but in practice equal to 0. In this case, *effective* rank determination (typically carried out by SVD, CHAPTER 11) is always the best way (although not the fastest) to decide about positive definiteness. Another option is to determine the *condition* or *reciprocal condition number* of the matrix (see SPECIAL TOPIC: condition number).

To understand the use of the covariance matrix and its eigen factorization, we start with another example in 2 dimensions. We generate some data points spread out along the x and y axis. The spread is much larger along the y direction.

x\_norm = normrnd(zeros(1,1000),2.5);

y\_norm = normrnd(zeros(1,1000),5.0);

data = [x\_norm;y\_norm];

figure;plot(data(1,:),data(2,:),'.',...

'Linewidth',0.5,'MarkerEdgeColor','b',...

'MarkerSize',15,'MarkerFaceColor','g')

xlabel('X');ylabel('Y');grid on; box on; axis equal; set(gca,'Ylim',[-15 15],'XLim',[-5 +5])

The covariance matrix shows very little correlation between the two coordinates (off-diagonal value):

C = cov(data')

For example, if x21>x20 there is no reason to believe that y21>y20. Notice how the diagonal of the covariance matrix has the variances (the squares of the standard deviations, σ's, used to generate the X and Y coordinates).

Since any covariance matrix ***C*** is *symmetric*, either positive definite or semidefinite, its eigen factorization as , provides a basis ***S*** (of orthogonal *eigenvectors*) in which the *similar* covariance matrix is *diagonal* (no covariation).

[S,D] = eig(C)

It is IMPORTANT to remember that if we calculate a *scaled* covariance matrix as ***C*** *=* ***AAT*** without dividing by the the number of observations *nobs* the eigenvalues will be also be scaled accordingly (multiplied by *nobs*). The MATLAB function *eig* typically lists the eigenvalues in ascending order. If we are interested in the eigenvalues in descending order, they can be obtained by reordering the matrices:

k = size(S,2);

[~,i] = sort(diag(D),'descend');

W = S(:,i);

R = D(i,i);

Where ***W*** and ***R*** are the reordered ***S*** and ***D***. Furthermore, the sign of the eigenvectors is arbitrary so if we want to have a consistent result we can change the sign so that it is positive in the direction with the most weight.

for n=1:k

W(:,n)=sign(mean(W(:,n)))\*W(:,n);

end

Alternatively, we can use the MATLAB function *pcacov* to obtain an eigen factorization with eigenvalues and eigenvectors in decreasing order.

[S,D] = pcacov(C)

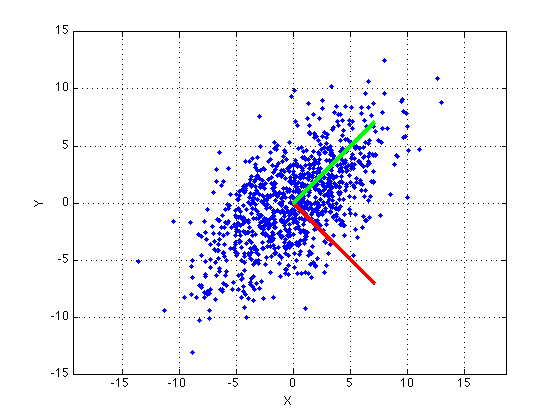
In the language of Principal Component Analysis the elements of each eigenvector are called the *loadings*. You will notice that in this case the *eigenvector* basis is only sligthly tilted with respect to the X and Y axes. The diagonal matrix of *eigenvalues* provides the variance of the data in the *eigenvector* basis. We can check if the covariance is removed by representing the data in the eigenvector basis. The coordinates of the data (more often the *centered* data) in the eigenvector basis ***S*** are usually called the *scores*:

scores = S'\*data;

Notice that to obtain a change of basis we should normally use the inverse of ***S***. However, since the covariance matrix is symmetric, its eigenvalues are real and its eigenvectors are (more properly 'can be chosen') orthogonal to each other. In the general case of an *m* x *n* matrix ***A*** with orthogonal vectors, ***ATA*** = ***I***. If ***A*** is also square then ***AT*** is also ***A-1***. For this reason we can simply take the transpose of ***S*** instead of its inverse.

cov(scores')

this means the covariance between X and Y is completely removed!

Let's take a look at a different data set: this time we generate data points aligned along two 45° axes (shown in red and green, respectively). Each point in the 2D space is going to be a linear combination of the new axes ***v1*** and ***v2***:

v1 = [1;-1]; v2 = [1;1];

v1 = v1/norm(v1); v2 = v2/norm(v2);

data = [v1 v2]\*[x\_norm;y\_norm]

In contrast to the previous example, the covariance matrix shows now very strong correlation between the X and Y coordinates:

C = cov(data')

This is clearly because since the axes along which the points are spread are inclined it is more likely that when X increases or decreases Y also increases or decreases. For the same reason the variance along X and Y is similar despite the fact that the variance is clearly different along the tilted axes.

When we factorize the covariance matrix in its *eigenvectors* and *eigenvalues*, we identify a basis, the *eigenvectors*, in which the covariance matrix becomes diagonal, such that the *coupling* between X and Y disappears and only the variance of the data remains on the diagonal.

[S,D] = pcacov(C)

We can see that the *eigenvectors* are inclined by ~45°; the variance of the data along those axes is now given by the *eigenvalues*.

In fact, we can can carry out a change of basis of the data (or the centered data) from the standard to the *eigenvector* basis and calculate the covariance matrix of the data represented in this basis.

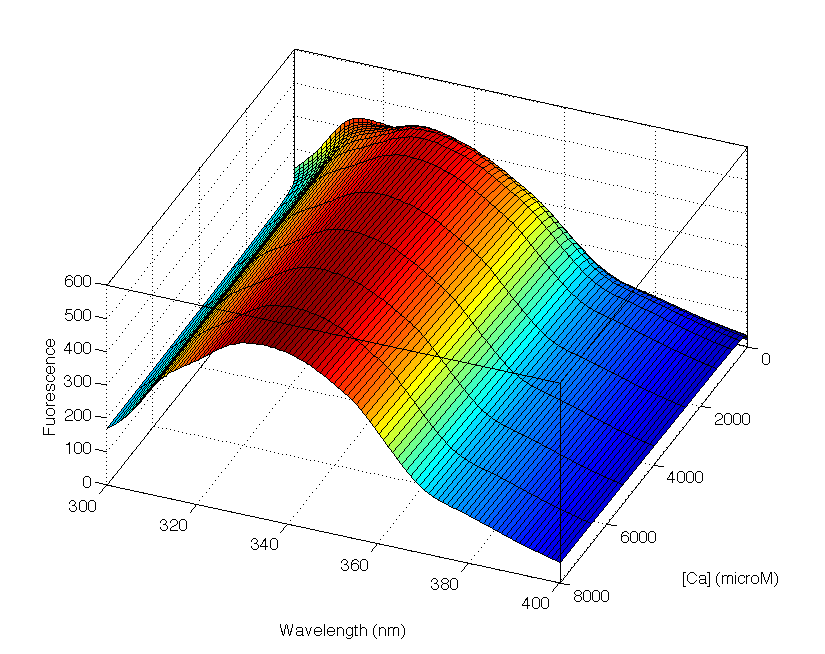
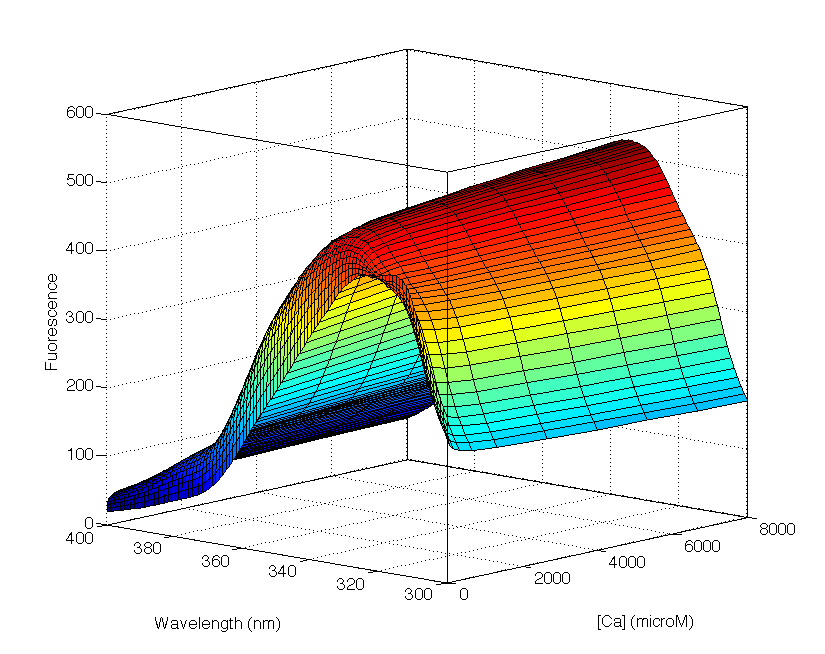
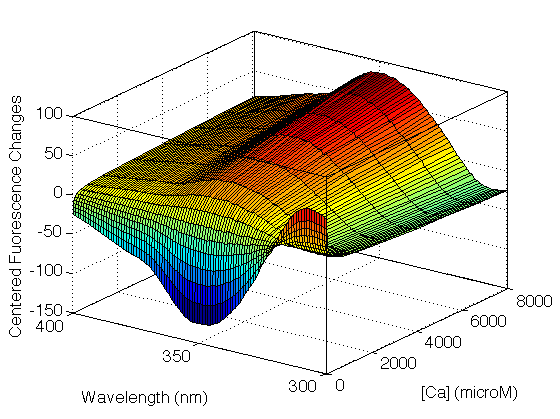
scores = S'\*data

cov(scores')

Indeed after the change to the *eigenvector* basis, the covariance matrix of the data shows no correlation between the X and Y coordinates. In that basis, each coordinate of a data point is effectively *decoupled* from the other coordinates of the same point. We conclude that the two vectors of the *eigenvector* basis ***S*** provide the '*principal components axes*’ along which the intrinsic variance of the data is revealed and the covariance is eliminated.

The elements of each *eigenvector* (the *coefficients* or *loadings*) are the coordinates of each eigenvector in standard space. The , the product of the transposed *eigenvectors* times the centered data ***cA***, are the coordinates of the centered data in the *eigenvector* basis, and are commonly referred to as the '*principal components*’ of the centered data ***cA***. Again, since the vectors in ***S*** are orthogonal, and the matrix is square the inverse is the same as the transpose. We have carried out a '*Principal Component Analysis*' (*PCA*) of our data.

Since PCA identifies consecutive orthogonal directions (dimensions) along which the variance of the data becomes progressively smaller, it can be conveniently used to *reduce* the dimensionality of the data to only those dimensions that hold most of the meaningful variance, while removing those dimensions that contain mostly noise.



We will now consider an example of such an application taken from a real biochemical experiment: the spectroscopic determination of the binding dissociation constant (*Kd*)of Ca2+ to a protein.

The determination was conducted by measuring in triplicate the fluorescence changes produced by increasing concentrations of calcium (a total of 15 concentrations ranging from 0 to 8 mM) in the solution of the protein Atp11p, a chaperone for the assembly of mitochondrial F1FO ATPase. The three independent assays were averaged to produce a single data matrix ***A*** containing the fluorescence changes (shown from two angles in the top two insets of the figures on the side).

importfile('DATABASE/Fluorescence\_data.txt');

A = Fluorescence\_data;

ca\_conc = [0 2.6 7.8 17.8 38.8 80.8 163.8 …

331 664 1331 2664 3997 5330 6663 7996];

wl =[300:400];[XI,YI] = meshgrid(ca\_conc,wl);

figure;surf(XI,YI,A);

In the ***A*** matrix (*m* x *n* = 101 x 15) the variables (1 to *m* = 101) are the fluorescence changes at the different wavelengths, while the observations (1 to *n* = 15) are the different concentrations of Ca2+ (ranging from 0 to 8 mM). We want to determine whether the observed spectral changes are due to a single or multiple components, and also remove as much as possible of the noise in order to obtain a clean correlation between the spectral changes and the concentration of Ca2+ added.

[nwls,nobs] = size(A);

mean\_A = mean(A,2);

cA = A - mean\_A;

The centered matrix, ***cA***, represents *changes* from the mean: therefore, values at different wavelengths are both positive and negative (bottom inset of the figure). The covariance matrix ***C***, ***cAcAT****/(nobs-1)*, has dimensions *m* x *m* = 101 x 101. E*igen decomposition* of the covariance matrix produces a matrix ***S*** of *eigenvectors* and a diagonal matrix ***D*** of *eigenvalues*. The 3rd output ('percent') of the function *pcacov*gives the percentage of the variance associated with each eigenvalues.Next, we represent the centered data ***cA***inthe *eigenvector* basis as .

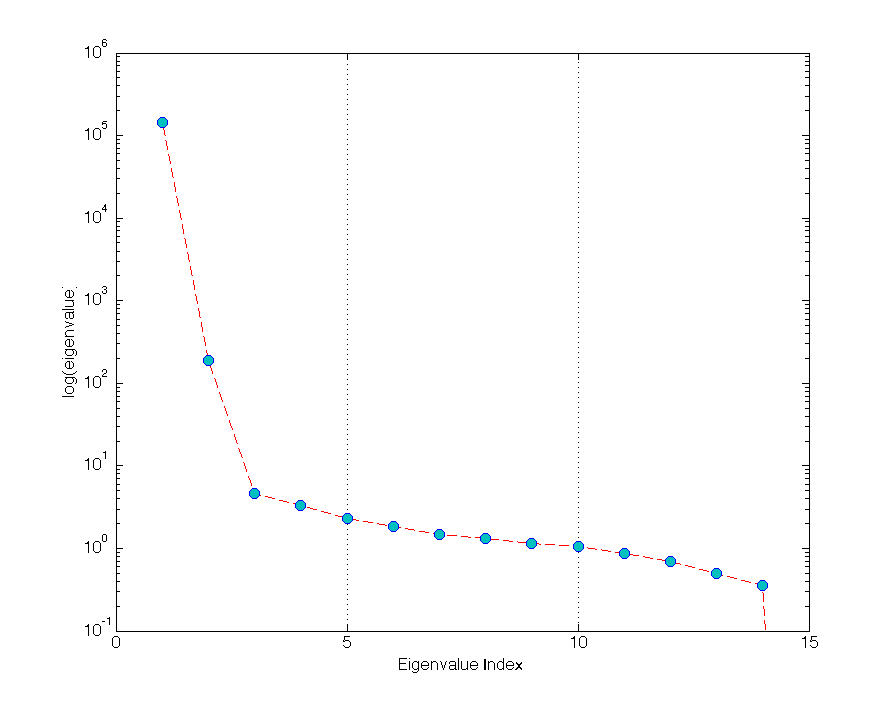
C = (cA\*cA')/(nobs-1);

[S,D,percent] = pcacov(C);

cA\_s = S' \* cA;

**IMPORTANT**: each column of represents a spectrum (properly a spectral change from the mean) at a different concentration of Ca. Therefore, when we carry out a change of basis to represent each spectrum in eigenvector basis ***S***, each column of the *scores* contains the coefficients of the linear combination of eigenvectors (the columns of *loadings*) that produces :

Thus, the eigenvectors are the 'normal modes' or 'individual spectral components': in other words, each *eigenvector* can be considered as a 'pure' spectral change (with respect to the '*mean*' of all the spectra), and the combination of these different 'pure' spectral changes gives origin to the 'total' spectral change that is observed at each Ca2+ concentration.

We also notice that the top *eigenvalue* represents 98.86% of all the variance of the data ('*scree*' plot on the right. 'Scree' is the rubble at the bottom of a cliff; in our case the rubble is all the *eigenvalues* we want to discard).

Scree = figure;

semilogy(D(1:14),'--ro','Linewidth',1.0,...

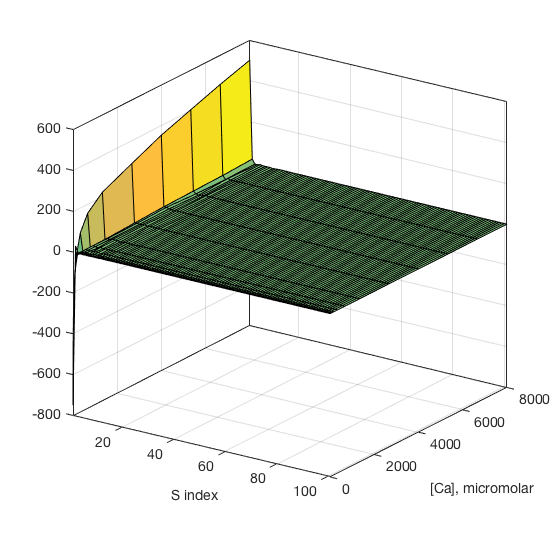
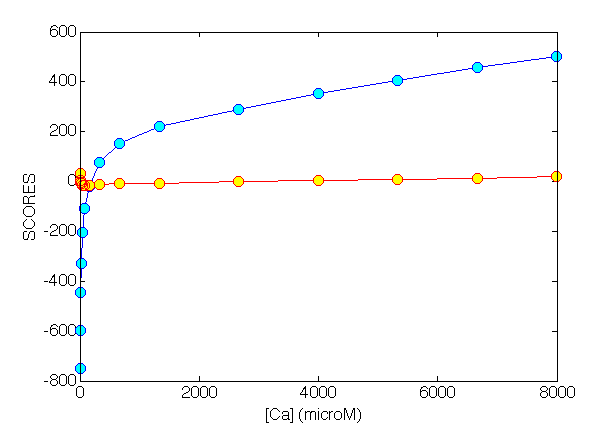
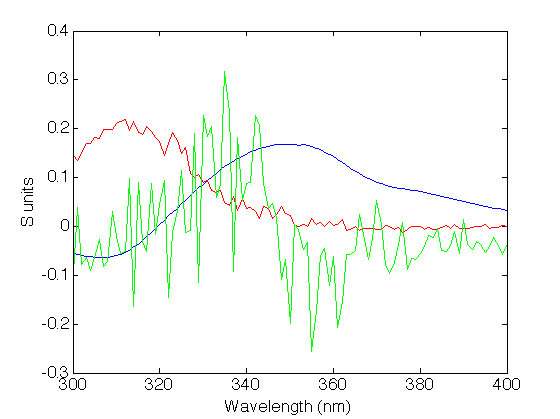
'MarkerEdgeColor','m','MarkerFaceColor','c');

ylabel('log(eigenvalue) '),xlabel('eigenvalue index ')

xlim([0 15]),ylim([1e-1,1e6])

Likewise, a plot the top 3 *eigenvectors* (normal modes) reveals only the 1st (blue line) to hold any significant information, while the 2nd (red line) and 3rd one (green line) already contain much of the noise in the data.

Normal\_modes = figure;



plot(wl,S(:,1),'b');

hold on

plot(wl,S(:,2),'r');

plot(wl,S(:,3),'g');

xlabel('Wavelength (nm) '),ylabel('S units' )

We can look at the representation of the centered data in eigenvector space:

cA\_data\_s = figure;

axes1 = axes('Parent',cA\_data\_s,'YDir','reverse');

view(axes1,[-77.5 12]);

box(axes1,'on');

grid(axes1,'on');

hold(axes1,'all');

nS = nwls; % eigenvector indices

[XS,YS] = meshgrid(ca\_conc,[1:nS]);

surf(XS,YS,cA\_s,'Parent',axes1);

xlabel('[Ca], micromolar'), ylabel('Eigen\_wl');box on

ylim([1,nS])

We can see how only the 1st *principal component* (1st row of the *scores* matrix) contributes significantly to this representation. In this plot the first 2 axes are the *Ca concentration* and the indices of each principal component: we could call these indices the *eigenwavelenghts*: the 3rd axis has the *scores*.An even better view of this result can be obtained by plotting only the changes in the contribution of the 1st (blu line) and 2nd (red line) spectral components (eigenvectors) upon increasing the Ca2+ concentration (these are the 1st two rows of the scores matrix).

Scores = figure;

plot(ca\_conc, cA\_s(1,:),'-ob', 'Linewidth',1.0,...

'MarkerEdgeColor','b','MarkerFaceColor','c');

hold on

plot(ca\_conc, cA\_s(2,:),'-or', 'Linewidth',1.0,...

'MarkerEdgeColor','r','MarkerFaceColor','y');

xlabel(['[Ca] (' texlabel('mu') 'M) '])

ylabel('SCORES' )

hold off

This plot clearly shows that only the 1st component contributes appreciably to the fluorescence changes. Therefore we can use this component to study the equilibrium binding of Ca to Atp11p, by first converting it to fractional saturation (for this purpose we guess the saturation point), and knowing that the fractional saturation Θ of a receptor by a ligand can be expressed as the *hyperbolic binding function*:

Θ = [LR]/[Rtot] = [L]/([L]+*Kd*)

where L is the ligand, R the receptor and LR the complex Ligand:Receptor.

We try a non-linear least squares fit with a single hyperbola (magenta dashed line) and with two hyperbolas (cyan line):

xvec = ca\_conc';

yvec = (cA\_s(1,:)' - min(cA\_s(1,:)))/1200;

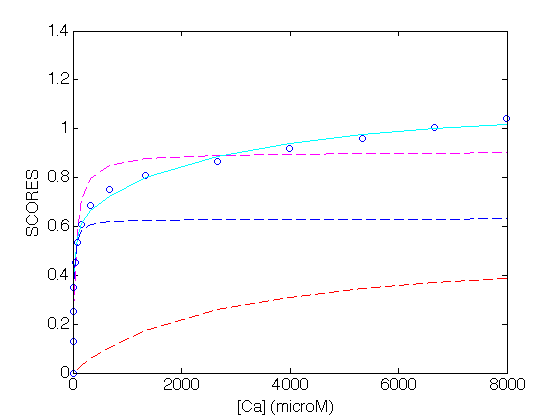
Equil\_bind = figure;

plot(xvec,yvec,'ob'); hold on

f = fittype('b\*(x/(a + x))');

[Hyperb\_1,GOF\_1] = fit(xvec,yvec,f,...

'StartPoint',[40 1]);

u1 = coeffvalues(Hyperb\_1)

lsq\_1 = u1(2)\*xvec./(u1(1)+xvec);

plot(xvec,lsq\_1,'--m');

f = fittype('b\*(x/(a + x)) + d\*(x/(c + x))');

[Hyperb\_2,GOF\_2] = fit(xvec,yvec,f,...

'StartPoint',[20 0.5 2000 0.5]);

u2 = coeffvalues(Hyperb\_2)

lsq\_21 = u2(2)\*xvec./(u2(1)+xvec);

lsq\_22 = u2(4)\*xvec./(u2(3)+xvec);

plot(xvec,lsq\_21,'--b',xvec,lsq\_22,'--r')

lsq\_2 = lsq\_21 + lsq\_22;

plot(xvec,yvec,'ob',xvec,lsq\_2,'-c')

xlabel(['[Ca] (' texlabel('mu') 'M) '])

ylabel('Fractional Saturation')

The fit with two hyperbolas is clearly superior and allows the identification of two different binding sites (contributing almost equally) with dissociation binding constants:

*Kd1* = 13.6 μM (blue dashed line)

*Kd2* = 2.61 mM (red dashed line).

We can also use PCA to remove noise from our fluorescence data. In order to do this we bring back the data from ***S*** space in the original space keeping only the contribution from the 1st eigenvector. Addition of the mean to the reduced centered data completes the operation of noise filtering:

fA = S(:,1)\*cA\_s(1,:) + mean\_A(:,ones(1,nobs));

Filtered\_A = figure;

axes3 = axes('Parent',Filtered\_A,'YDir','reverse');

view(axes3,[-77.5 12]);

box(axes3,'on');grid(axes3,'on');hold(axes3,'all');

surf(XI,YI,fA,'Parent',axes3);

xlabel('[Ca], micromolar'), ylabel('wavelength, nm');box on

It is important to recognize that this operation of *noise filtering* is effectively a *projection* operation of the centered data onto the subspace whose basis is the 1st eigenvector. In fact:

S1 = S(:,1)

P = S1/(S1'\*S1)\*S1';

P = S1\*inv(S1'\*S1)\*S1';

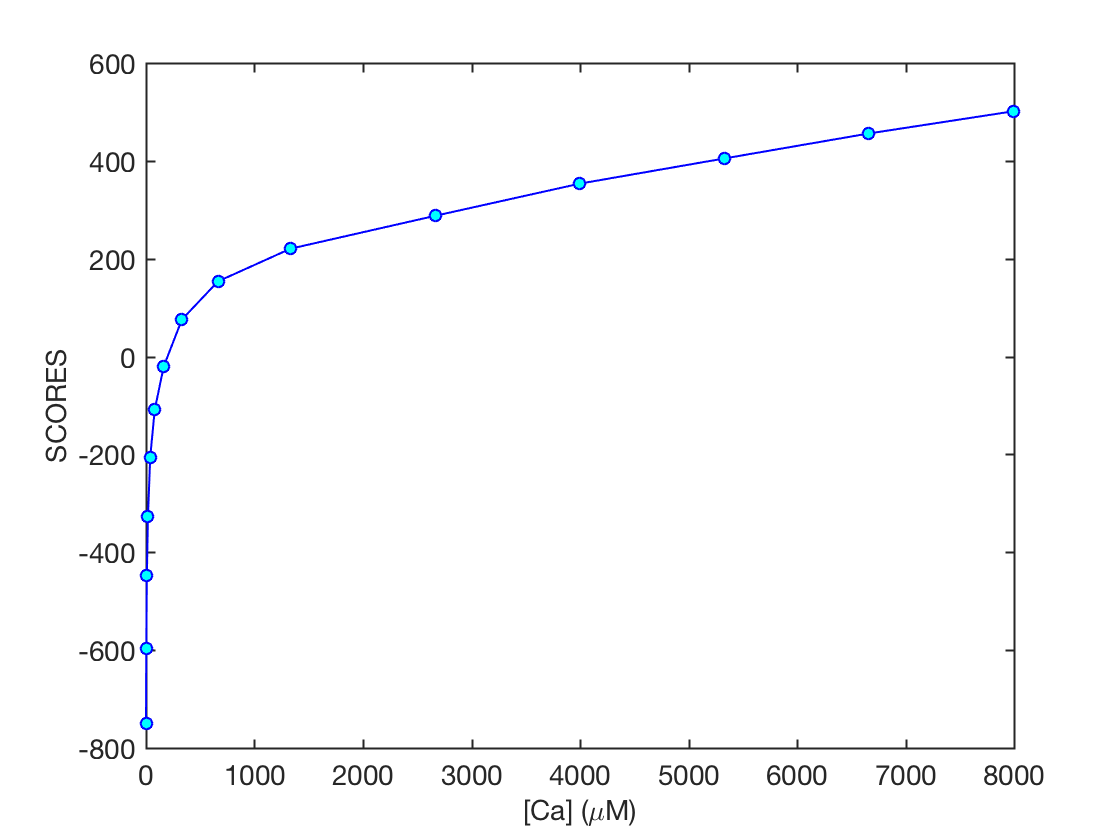
P = S1\*S1';

The projection of ***cA*** on ***S1*** is for every column in ***cA*** the 'part' of ***S1*** that looks most like that column:

fA = P\*cA;

This is the same as recognizing that there is no exact solution for:

where is a column vector and is a row vector, because none of the columns of is in the *column space* of , and therefore there exists only a *least squares* solution:



where is the vector containing all the multipliers of that produce a *least squares fit* of to each column of . Clearly corresponds to the 1st row of the *scores* matrix.

fA = S1\*inv(S1'\*S1)\*S1'\*cA;

x = inv(S1'\*S1)\*S1'\*cA

x = S1'\*cA

LS\_Scores = figure;

plot(ca\_conc, x,'-ob', 'Linewidth',1.0,...

'MarkerEdgeColor','b','MarkerFaceColor','c');

xlabel(['[Ca] (' texlabel('mu') 'M) '])

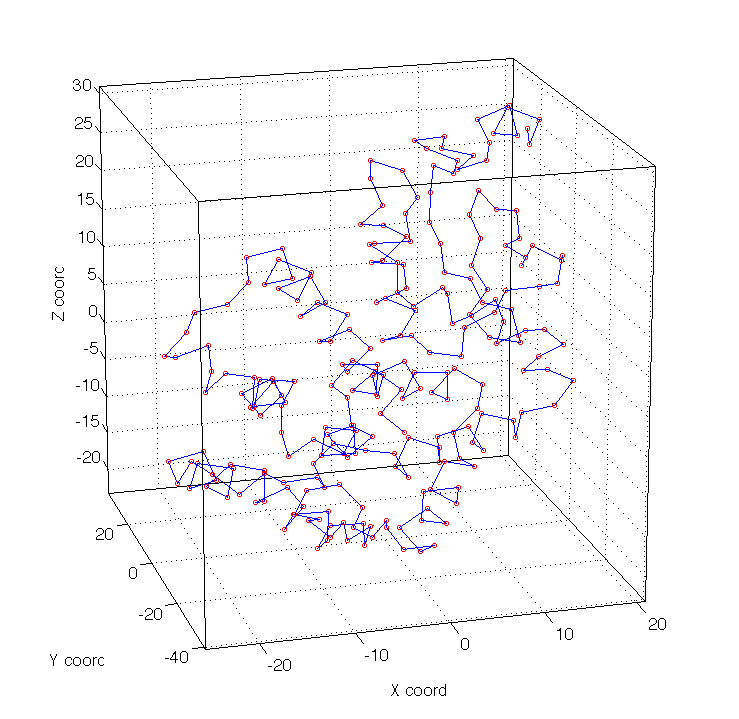
xlabel('[Ca] \muM)'); ylabel('SCORES' )

As a final example of applications of PCA we will discuss the identification of the conformational changes occurring during the simulation of a protein in solution by Molecular Dynamics (MD): these conformational changes are often referred to as the *'Essential Dynamics'* of a protein.

We start by loading the 20 ns MD trajectory of the simulation of the protein ATP12p, another chaperone necessary for the assembly of mitochondrial F1FO ATPase. The original trajectory matrix contains a total of ***m =*** 417 rows (each row corresponding to a frame, an observation), and ***n* =** 735 columns corresponding to the X, Y, and Zcoordinates(the variables) of 245Cα atoms.

trj\_mat = dlmread('DATABASE/md\_trj\_matrix.txt');

savefile = 'MD\_trj\_pca';

[nframes,ncoords] = size(trj\_mat);

natoms = ncoords/3;

Several MATLAB Toolboxes are freely available to carry out a large variety of computational tasks. In our course, we will use several toolboxes designed for specific applications; they are stored in the folder 'TOOLBOXES'. In particular, the MDTOOLBOX developed by Yasuhiro Matsunaga (<https://github.com/ymatsunaga/mdtoolbox>) contains a very sophisticated set of tools for the analysis of MD simulations. After downloading the toolbox and installing it in our TOOLBOXES directory, we use its program *meanstructure* to superimpose all the frames to the structure that is the best possible average of all the structures in the trajectory.

addpath(genpath('TOOLBOXES/MDTOOLBOX'))

[Aref,A] = meanstructure(trj\_mat);

Aref = Aref'; A = A';

[nvar,nobs] = size(A);

At the end of this process the trajectory data is in the form of a ***m*** x ***n*** (*735* x *417*) matrix ***A*** in which for every atom there are three rows representing the x, y, and z coordinates, respectively. As an example, we display the trace of the protein at time 0 (the 1st frame).

a1 = A(:,1);

a1\_x = a1(1:3:end);

a1\_y = a1(2:3:end);

a1\_z = a1(3:3:end);

A1\_Frame = figure;

plot3(a1\_x,a1\_y,a1\_z,'-bo','Linewidth',1.0,...

'MarkerEdgeColor','r','MarkerFaceColor','y')

box('on'); grid('on')

xlabel('X coord')

ylabel('Y coord')

zlabel('Z coord')

Upon centering (***A⇒cA***) the covariance matrix ***C*** of ***A*** is calculated.

mean\_A = mean(A,2);

cA = A - mean\_A(:,ones(1,nobs));

C = (cA\*cA')/(nobs-1);

MSSF = diag((cA'\*cA)/(nvar))

IMPORTANT: The diagonal entries of ***C*** (dimensions ***m*** x ***m,*** 735 x 735) are the variances of the x, y, and z coordinate of each atom along the MD trajectory (in other words they provide information on the fluctuation of each atom about its average position). The off-diagonal elements give information about the correlation or anticorrelation of the fluctuations of different atoms. Notice that each column of ***cA*** containsthe deviation from the meanof the coordinate of all the atoms of each frame. Thus, if we were to calculate the product ***cATcA****/nvar* between the columns of the ***cA*** matrix instead of the rows and divide it by the number of variables, the diagonal would contain the Mean Sum of Square Fluctuations (MSSF) of all the atoms in each frame with respect to the reference frame. This is not the same as calculating the frames covariance matrix, because the centering in ***cA*** still occurs along the direction of the different observations instead of the different variables.

*Eigen decomposition* of the ***C*** matrix leads to:

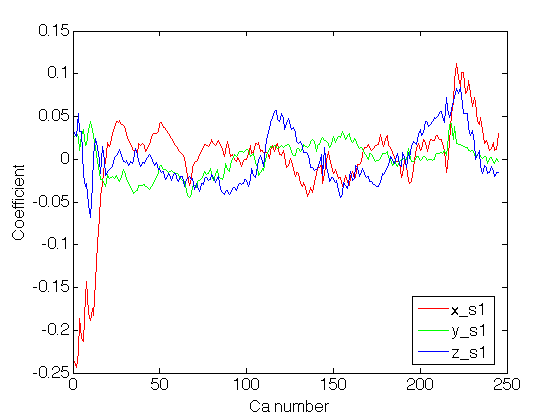
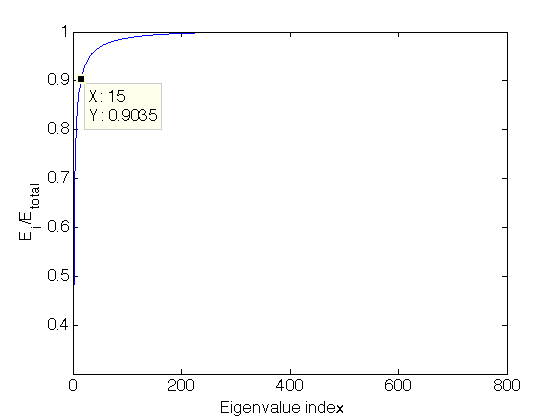
[S,D,percent] = pcacov(C);

The elements of each *eigenvector* in the matrix ***S*** are the principal components '*coefficients*' or '*loadings*'. They are the coordinates (in the standard basis of a *m*-dimensional space - unit vectors containing a single 1 and 734 0's) of ***m*** = 735 orthogonal basis vectors along which the variance of the variables is maximized and their covariance is minimized. Next, we represent the centered data ***cA*** in the ***S*** basis (change of basis).

cA\_s = S' \* cA;

The are again the *'principal components*'. They are the product of the transposed *eigenvectors* times the centered data ***cA***; therefore, they are the coordinates (the representation) of the centered data in the ***S*** basis. Let's think for a moment about the physical meaning of this fact. The ***scores*** matrix has the same dimension ***m*** x ***n*** (735 X 417) as the centered data ***cA***; each column ***cAj*** of the matrix represents the coordinate changes of the protein from the mean structure in each frame of the simulation. However, since the basis now is ***S***, these changes are represented by coefficients *aij* (the *scores*) that tell us how much of each *eigenvector* ***s*** we have to take in order to obtain the change ***cAj*** (the ***j*** column of ***cA****,* with ***j*** going from 1 to the number of frames ***n***):

Thus, each *eigenvector* represents a vibrational '*normal mode*' containing information on the displacements of all the atoms of the protein from their mean positions, and is the linear combination of these normal modes (vibrations) that gives origin to the difference in structure observed in each frame of the trajectory. We can recognize this clearly by plotting separately the x, y, and z components of each *eigenvector*.



x\_S1 = S(1:3:end,1);

y\_S1 = S(2:3:end,1);

z\_S1 = S(3:3:end,1);

Vibration = figure;

plot(x\_S1,'r')

hold on

plot(y\_S1,'g')

plot(z\_S1,'b')

legend('x\\_s1','y\\_s1','z\\_s1','Location','Best')

xlabel('Ca number')

ylabel('Coefficient')

Notice the oscillatory behaviour of the entire Cα chain along its length, as if it was a linear elastic string.

A plot of the cumulative sum of the *eigenvectors energy*:

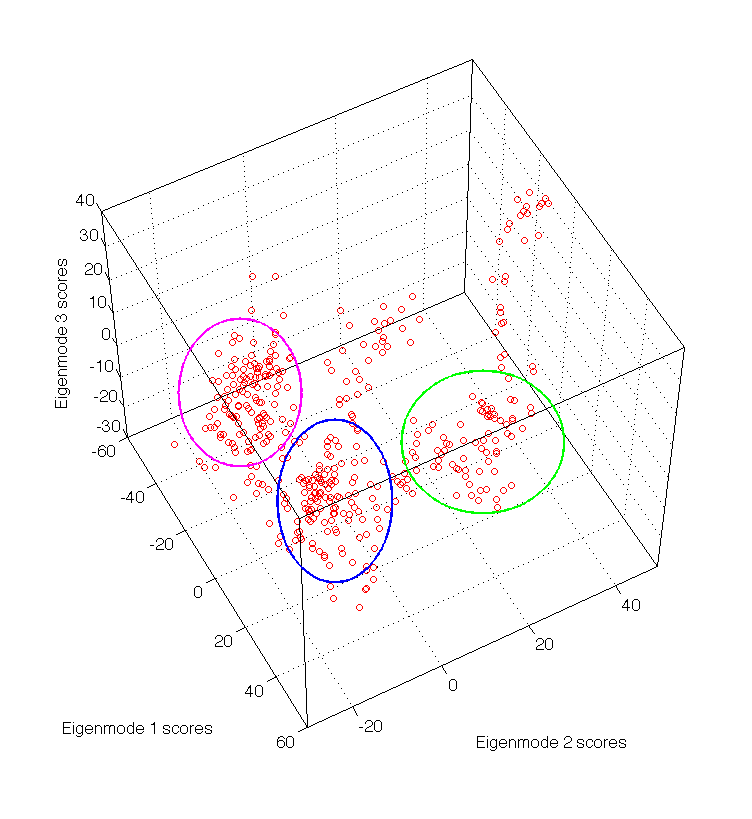
energy = cumsum(D);

Energy = figure; plot(energy/energy(end))

xlabel('Eigenvalue index ')

ylabel('E\_j/E\_t\_o\_t\_a\_l ')

shows that 90% of all the vibrational energy is contained in the first 15 *eigenmodes*, and ~70% of it in just the first 3 *eigenmodes*. In light of this observation, we can look at the contributions of the top 3 *eigenmodes* along the entire recorded trajectory. We recall here that these contributions are contained in each row of the ***scores*** matrix : for example the contributions of the different *eigenmodes* to the ***n*** frames (often called the *'projections*'of the MD trajectory onto the different *eigenmodes*) are:



...

It is intuitously evident that if different frames have similar contributions from the top *eigenmodes*, then it means they represent similar conformations. Therefore, if we plot the top 3 rows of the *scores* matrix (the top 3 *principal components* of the MD) against each other, frames that have similar contributions from the top 3 *eigenmodes* will have similar coordinates and thus will appear clustered. In our case, 3 major clusters of coordinates can be clearly identified, suggesting that 3 major conformations of ATP12p appear during the MD simulation.

CL = clusterdata(cA\_s(1:3,:)','linkage', 'ward','savememory','on','maxclust',5);

Clusters = figure; plot3(cA\_s(1,:),cA\_s(2,:),cA\_s(3,:),'ro'); hold on

scatter3(cA\_s(1,:),cA\_s(2,:),cA\_s(3,:),20,CL,'fill')

xlabel('Eigenmode 1 scores '); ylabel('Eigenmode 2 scores ');

zlabel('Eigenmode 3 scores ')

grid('on');box('on')

We can easily find out which frames correspond to which clusters and the size of each cluster:

CL1 = CL==1; CL2 = CL==2; CL3 = CL==3;CL4 = CL==4; CL5 = CL==5;

CL1\_size = sum(CL1); CL2\_size = sum(CL2); CL3\_size = sum(CL3); CL4\_size = sum(CL4);

CL5\_size = sum(CL5);

Since we have established that the important information about the conformational changes occurring along the MD trajectory are provided by the top 15 *eigenmodes*, we can ignore the contribution of the remaining *eigenmodes* in the matrix and bring it back into Cartesian coordinates to obtain a *reduced* representation ***rcA*** of the ***cA*** matrix:

rcA = S(:,1:15)\*cA\_s(1:15,:);

Finally, we add back the *mean* that we had subtracted to obtain the centered data:

rA = rcA + mean\_A(:,ones(1,nobs));

The resulting matrix ***rA*** is a new MD trajectory in which only the contribution from the top 15 *eigenmodes* is included. This result corresponds to a selective removal of all the atoms small movements due to thermal fluctuation, while retaining only the large fluctuations associated with true conformational changes. This operation of *dimensionality reduction* can be done even for just 1 *eigenmode*: as an example, we plot here together a sampling (14 evenly spaced frames) of the proteins motions produced by just the top vibrational mode (*1st essential dynamics*) during the entire MD simulation.

rcA\_ed1 = S(:,1)\*cA\_s(1,:);

rA\_ed1 = rcA\_ed1 + mean\_A(:,ones(1,nobs));

ED1 = figure;

for i = 1:29:nobs

F = rA\_ed1(:,i);

x = F (1:3:end);

y = F (2:3:end);

z = F (3:3:end);

plot3(x,y,z,'-bo','Linewidth',1.0,...

'MarkerEdgeColor','r',...

'MarkerFaceColor','y')

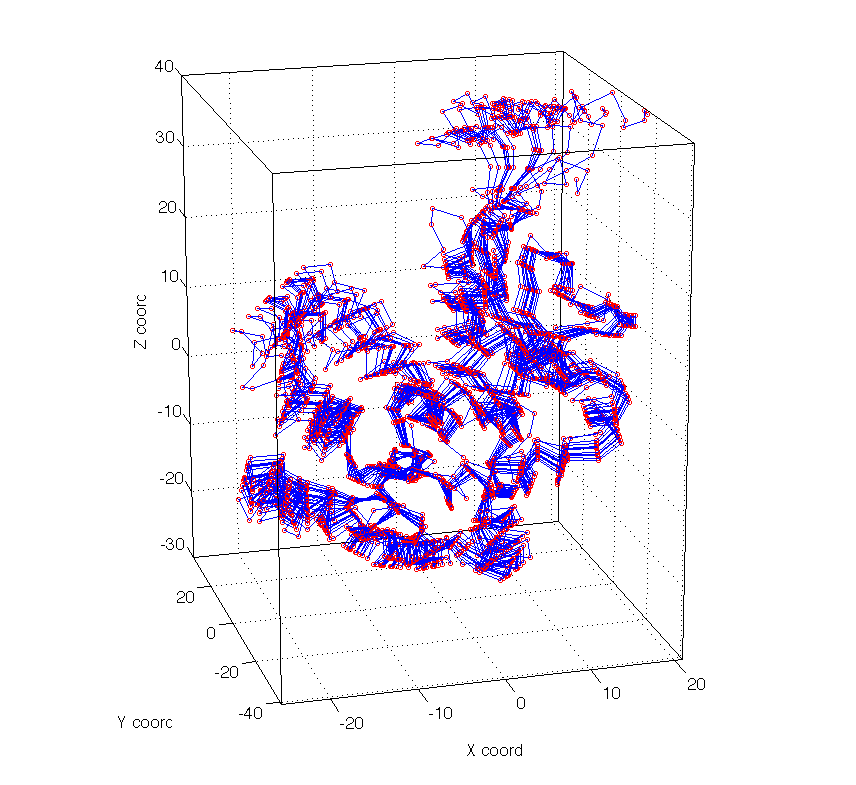
hold on;

drawnow expose; pause(.1)

end

box('on');grid('on')

xlabel('X coord');ylabel('Y coord');zlabel('Z coord')



Additional information on the protein fluctuation is provided by the *eigenvalues* of the covariance matrix. In fact, each *eigenvalue* represents the variance of each row of ***cA\_s*** (the MD trajectory represented in the *eigenvector* space), and each row of ***cA\_s*** is the contribution of a single vibrational *eigenmode* of all the atoms of the structure to the various frames*.* This means that each *eigenvalue* is the mean of squared positional fluctuations of all the atoms due to the corresponding eigenmode: therefore the total **RMSF** (Root Mean Square Fluctuation) for all the atoms in the protein due to a given *eigenmode* can be calculated very easily by taking the square root of the corresponding *eigenvalue*; likewise the total RMSF for all the atoms due to the combination of different *eigenmodes* (or even all *eigenmodes*) is derived as the square root of the sum of the corresponding *eigenvalues*.

==

We can easily check that this is true. First, we determine the RMSF for each atom of the structure either directly from the known formula or from the diagonal of the covariance matrix:

C\_diag = diag(C);

MSF = zeros(natoms,1);

RMSF = zeros(natoms,1);

dRMSF = zeros(natoms,1); % where d stands for direct

first = 1;

for i = 1:natoms

second = first+1;

last = first+2;

% Direct determination from the RMSF formula

dRMSF(i) = sqrt(((cA(first,:)\*cA(first,:)' + ...

cA(second,:)\*cA(second,:)' + ...

cA(last,:)\*cA(last,:)')/nobs));

% Determination from the diagonal of the covariance matrix:

MSF(i) = sum(C\_diag(first:last));

RMSF(i) = sqrt(sum(C\_diag(first:last)));

first = first+3;

end

From which we derive the RMSF for the entire structure including all *eigenmodes* as:

total\_RMSF = sqrt(sum(MSF)); total\_RMSF = sqrt(sum(diag(C)));

total\_RMSF = sqrt(sum(D))

We can check that this result is valid also for the trajectory corresponding to the *1st* *essential dynamics* that we have calculated before.

C\_ed1 = (rcA\_ed1\*rcA\_ed1')/(nobs-1)

total\_RMSF\_ed1 = sqrt(sum(diag(C\_ed1)))

total\_RMSF\_ed1 = sqrt(D(1))

**SPECIAL TOPIC: Rotational ambiguity of eigenvector basis in PCA.**

In developing our derivation of PCA we have adopted the convention that in a data set variables are distributed down the columns and that each row contains all the different observations of each variable. This convention treats the variables (fluorescence at a given wavelength) as coordinates in a column vector, and thus representing the data in the eigenvector space of the covariance matrix is achieved by premultiplying the data by the inverse of the eigenvector basis. We rapidly recapitulate this for the fluorescence data set we have used in CHAPTER 10 as an example:

Load fluorescence data:

load('DATABASE/Fluorescence\_data.txt');

X = Fluorescence\_data;

ca\_conc = [0 2.6 7.8 17.8 38.8 80.8 163.8 331 664 1331 2664 3997 5330 6663 7996];

wl =[300:400];

[XI,YI] = meshgrid(ca\_conc,wl);

figure;surf(XI,YI,X);

Center the data along each row and calculate the covariance matrix

nobs = size(X,2)

mX = mean(X,2)

cX = X - mX

covX = cX\*cX'/nobs

Here we calculate *loadings* (coefficients) and *scores* given the variables along the columns of X. The scores are the coordinates of the (centered) data in eigenvector space, which means they tell us how much of each eigenvector we have to take in order to generate the original data set in standard space. The eigenvalues are the variance of the scores.

[S,D] = pcacov(covX)

scores = S'\*cX;

var\_scores = var(scores')

In this sense the eigenvectors ***S*** represent *pure spectral components* that when summed up produce the original fluorescence data. In this particular example we know already that only the first two eigenvectors (spectral components) contribute significantly to the variance of the fluorescence at each wavelength. We can plot the 1st spectral component:

figure;plot(wl,S(:,1))

Thus, we can reconstitute a noise filtered data set as:

fcX = S(:,1:2)\*scores(1:2,:)

figure;surf(XI,YI,fcX);

This is the same as:

S\_red = S;

scores\_red = scores;

S\_red(:,3:4) = 0;

scores\_red(3:4,:)= 0;

fcX\_red = S\_red\*scores\_red;

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

However, it would be misleading to think that the pure spectral components derived from this analysis are *unique*. In fact, we can introduce an orthogonal transformation matrix that will rotate the basis of the eigenvector space, for example by 30 degrees counterclockwise:

theta = pi/6

Q = [cos(theta) sin(theta);-sin(theta) cos(theta)]

Srot1 = S(:,1:2)\*Q';

Since we used an orthogonal transformation matrix the new basis is still orthogonal:

Srot1'\*Srot1

Here we recalculate the scores based on the new basis

scoresrot1 = Srot1'\*cX;

Here we calculate the variance explained by both the old and the new basis:

var\_scores(1:2)

var\_scoresrot1 = var(scoresrot1')

sum\_var\_scores = sum(var\_scores,2)

sum\_var\_scoresrot1 = sum(var\_scoresrot1,2)

Here we produce a new noise filtered data set, which is identical to the previous one

fcXrot1 = Srot1\*scoresrot1

figure;surf(XI,YI,fcXrot1);

This result is not limited to an orthogonal transformation matrix, but it applies to any transformation represented by an invertible matrix;

T = rand(2)

Srot2 = S(:,1:2)\*inv(T);

For example, the new basis is not orthogonal

Srot2'\*Srot2

Next, we recalculate scores and noise filtered data set. Since the new basis is not orthogonal we can’t simply take the transpose, but we need to calculate a pseudoinverse (see *SVD*, CHAPTER 11):

scoresrot2 = pinv(Srot2)\*cX;

Variance explained:

var\_scoresrot2 = var(scoresrot2')

sum\_var\_scoresrot2 = sum(var\_scoresrot2,2)

Noise filtered data:

fcXrot2 = Srot2\*scoresrot2

figure;surf(XI,YI,fcXrot2);

Here we plot the spectral components derived in the three cases:

figure;

subplot(2,1,1);plot(wl,[S(:,1) Srot1(:,1) Srot2(:,1)]),

legend('original basis','rotated orthog. basis ','rotated non-orthog. basis')

ylabel('Fluorescence')

title('1st spectral component')

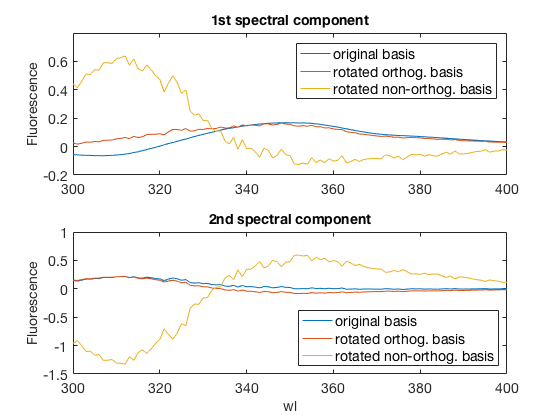
subplot(2,1,2);plot(wl,[S(:,2) Srot1(:,2) Srot2(:,2)]), legend show

legend('original basis','rotated orthog. basis ','rotated non-orthog. basis','Location','Best')

xlabel('wl')

ylabel('Fluorescence')

title('2nd spectral component')

This result shows clearly that the spectral components provided by the basis of the eigenvector space are not unique (*rotational ambiguity*), for the simple reason that the choice of basis for this space is arbitrary as long as the basis vectors are independent (orthogonal rotation), and even if they are correlated (oblique rotation – convex basis).

In MATLAB, the rotational ambiguity of the eigenvector basis can be explored using the function *rotatefactors*. In order to use this function, it is important to recall that MATLAB (like many other statistical packages that provide a PCA analysis expects the variable to be distributed along rows, rather than columns, and thus our original dataset must be transposed before further action:

Xt = X';

mXt = mean(Xt);

cXt = Xt - mXt;

covXt = cXt'\*cXt/nobs;

[S2,D2] = pcacov(covXt)

Notice that this is the more general way in which Principal Component Analysis is described in most textbooks of Statistics.

Let’s assume we have a *m* x *n* *data matrix* ***X*** with *n* variables as different columns and *m* observations as different rows. In order to make an *f-components* PCA model of this matrix we calculate the approximation:

***X*** = ***TP****T*

where ***T*** is the *m* x *f* *score matrix* and ***P*** is the *n* x *f* *loading matrix*.

As a consequence, the *scores* are also transposed. Notice that if we transpose ***X***, then ***STX*** becomes ***XTS*** , and thus to obtain the scores we multiply *on the right* the data (centered down the columns) by the eigenvector basis:

scores2 = cXt\*S2;

Likewise, to bring the noise filtered data back into standard space we multiply the new scores *on the right* by the inverse (transpose if orthogonal) of the eigenvector basis:

cfXt = scores2\*S2';

figure;surf(XI,YI,cfXt');

*Rotatefactors* offers various options to rotate the loading matrix. The most popular rotation (the default with *rotatefactors*)is the so called *varimax rotation* . The issue here is that the eigenvector space found with PCA is represented by a dense basis with many non-zero weights, which makes it hard to interpret. Varimax is so called because it maximizes the sum of the variances of the squared loadings while preserving orthogonality. Intuitively, this is achieved if any given variable has a high loading on a single score column but near-zero loadings on the remaining score columns. If these condition holds, the loading matrix is said to have a *simple structure*. From the variables perspective, varimax seeks a basis such that each variable can be described by a linear combination of the minimum number of basis functions.

In practice, given a *d* x *m* loading matrix *rotatefactors* finds the orthogonal rotation that maximizes the objective function (and is thus called the *orthomax rotation*):

sum(d\*sum(B.^4,1) - GAMMA\*sum(B.^2,1).^2)

where *d* is the number of variables and GAMMA is:

1 – *varimax* rotation

0 – *quartimax* rotation

*m*/2 – *equamax* rotation

*d*(*m*-1)/(*d*+*m*-2) – *parsimax* rotation

Here we define the loading matrix as the first two columns of the eigenvector matrix, and we recalculate the scores accordingly:

loadings2 = S2(:,1:2);

scores2 = cXt\*loadings2;

cfXt = scores2\*loadings2';

figure;surf(XI,YI,cfXt');

Here we use *rotatefactors* to find the *varimax* rotation of the loadings:

[loadings2rot1, T1] = rotatefactors(loadings2 ,'Method','varimax')

which also returns the rotation matrix ***T1*** used to generate ***loadings2rot1*** according to:

loading2rot1 = loadings2\*T1;

Here we make the table of the rotated loadings:

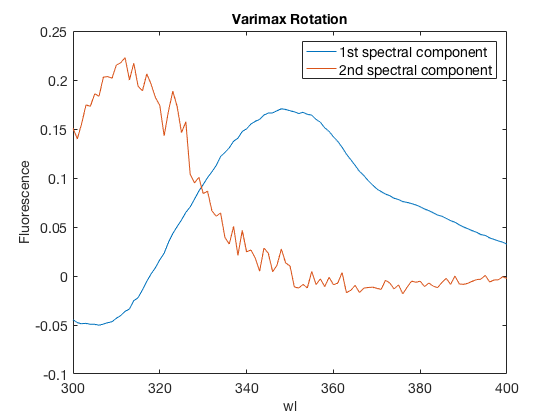
cnames = {'1st spectr. comp.','2nd spectr. comp.','1st spectr. comp.','2nd spectr. comp.','1st spectr. comp.','2nd spectr. comp.',};

Rotated\_loadings = figure('Position',[300 300 700 700]);

t = uitable(Rotated\_loadings,'Data',[loadings2rot1(1:34,:) loadings2rot1(35:68,:) [loadings2rot1(69:101,:);0 0]],'ColumnName',cnames,'FontWeight','bold','FontSize',9,'RowStriping','off','ColumnEditable',true);

t.Position(3) = t.Extent(3);

t.Position(4) = t.Extent(4);

Here we plot the rotated loadings:

figure;

plot(wl,[loadings2rot1(:,1) loadings2rot2(:,2)])

title('Varimax Rotation')

legend('1st spectral component','2nd spectral component')

xlabel('wl') ; ylabel('Fluorescence')

We can clearly see how the weight is now distributed between the two loading columns. When one column has a high value the other one has a small value.

Here we recalculate the scores based on the rotated loadings:

scores2rot1 = cXt\*loadings2rot1;

Here we generate the filterted data:

cfXtrot1 = scores2rot1\*loadings2rot1';

figure;surf(XI,YI,cfXtrot1');

The correlation matrix between the back-rotated scores is:

corr\_scores2rot1 = corr(scores2rot1\*T1')

or also:

corr\_scores2rot1 = inv(T1'\*T1)

**Norms and Condition Number.**

In CHAPTER 1 we have introduced the concept of the 2-norm of vectors and matrices. One common property of 2-norms is that they must fulfill the rule of *triangle inequality*:

a = [2 3 5 7]'

b = [5 1 0 3]'

norm(a+b)

norm(a) + norm(b)

A = randi(10,4)

B = randi(10,4)

norm(A+B,2)

norm(A,2) + norm(B,2)

A second requirement for the 2-norm of a matrix is related to products:

norm(A\*B,2)

norm(A,2)\*norm(B,2)

and considering any vector ***x*** the inequality means that the 2-norm of a matrix is the largest possible value of :

For *symmetric* matrices the 2-norm can be calculated by taking as the *eigenvector* corresponding to the largest *eigenvalue.* In fact, if is symmetric, then and since orthogonal matrices leave the length of a vector unchanged, the ratio to maximize is really :

C = [-53 45 53 0;

45 100 16 -4;

53 16 -108 44;

0 -4 44 -46];

eig(C)

norm(C,2)

is often referred to as the *spectral radius* of a square matrix ***A***, where the *spectrum* of ***A*** is the set of its *eigenvalues*. The spectral radius is a valid 2-norm of ***A*** only if ***A*** is *symmetric*.

For a symmetric matrix ***A*** and a non-zero vector ***x***, the ratio:

is called the *Rayleigh quotient*. This ratio reaches its maximum value and its minimum value when is the corresponding *eigenvector* of those two *eigenvalues*. Thus, if any other vector is substituted into the Rayleigh quotient the result is guaranteed to be between and .

For *unsymmetric* matrices only :

therefore:

In CHAPTER 11 we will see that the *singular values* of a general (square or rectangular) matrix derived from are the square roots of the *eigenvalues* of . It follows that for a general matrix the 2-norm is:

Now, consider the general matrix equation:

If the righ-hand side is changed to because of measurement errors, the solution becomes , and the error equation is:

Notice how becomes progressively larger the closer it is to be singular:

A = randi(10,4);

A1 = A; A1(:,4) = A(:,3)+ ones(4,1)\*1e-6;

inv(A1)

A2 = A; A2(:,4) = A(:,3)+ ones(4,1)\*1e-10;

inv(A2)

A3 = A; A3(:,4) = A(:,3)+ ones(4,1)\*1e-14;

inv(A3)

Consequently, the closer is to singular the larger the error . From triangle inequality we derive the largest error:

where:

To bring all these results together we apply the triangle inequality of matrix norms to the original and the error equations:

**,**

**,**

We multiply the two left sides by each other and the two right sides by each other:

then we divide both sides by **:**

where is the *condition number*. Clearly, larger values lead to larger relative error.

Error, , can also be associated with the matrix itself, in which case it usually originates from numerical errors in the Gaussian elimination, as it often happens when the pivots are very small numbers. In this case both and the *condition number*  control the magnitude of :

In conclusion, the *condition number*  of a matrix, *cond*(***A***), measures the sensitivity of the solution of a system of linear equations to errors in the data. It gives an indication of the accuracy of the results from matrix inversion and the linear equation solution. Values of *cond*(***A***) near 1 indicate a well-conditioned matrix. The norms in the product are most often calculated as the 2-norm, but it is also possible to use the 1-norm, the Frobenius norm, or the infinity norm.

c\_1 = cond(A,1); c\_1 = norm(A,1)\*norm(inv(A),1)

c\_2 = cond(A,2)

c\_fro = cond(A,'fro')

c\_inf = cond(A,inf)

How large should a condition number be to suggest a matrix is poorly conditioned? In order to have a narrow range of value on the basis of which to decide whether a matrix is well conditioned, most often the *reciprocal of the condition number*, *rcond*, calculated with 1-norms is used instead of the condition number. If ***A*** is well conditioned, *rcond*(***A***) is near 1.0. If ***A*** is badly conditioned, *rcond*(***A***) is near 0. Any *rcond*(***A***)<10-14 should raise some concern about the invertibility of ***A***.

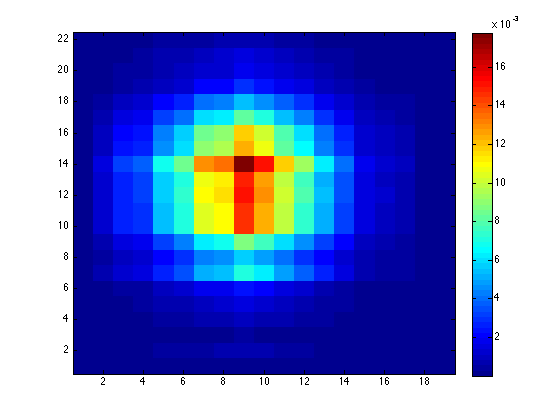
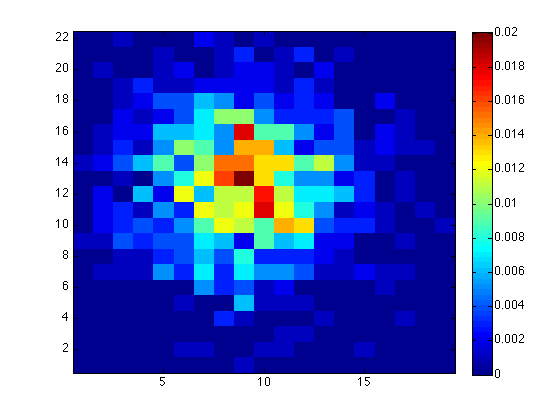
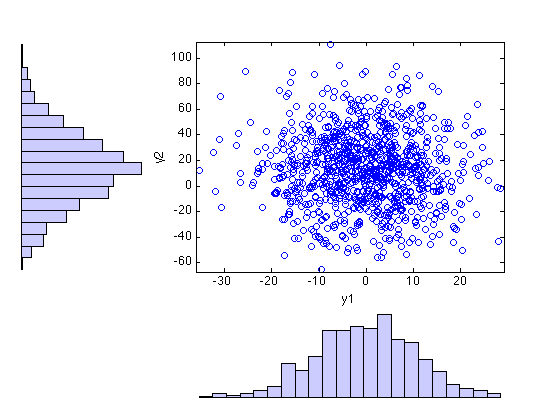
rc\_1 = 1/cond(A,1)

rc\_1 = rcond(A)

**Independent Component Analysis.**

The core of Principal Component Analysis (**PCA**) is the identification of a space in which the variance of each variable is maximized and the covariance between variables is eliminated. In this space the variables are uncorrelated. However, if the distribution of the different values (observations) of each variable is *not normal* (that is, Gaussian) the orthogonal directions identified by PCA will not provide the correct result. This task is instead accomplished by Independent Component Analysis (**ICA**), which identifies a space in which the variables are not just *uncorrelated*, but also *independent*.

It is worth establishing the difference between *uncorrelatedness* and *independence*. To define the concept of independence, consider two scalar-valued random variables ***y*1** and ***y*2**. Basically, the variables ***y*1** and ***y*2** are said to be independent if *information* on the value of ***y*1** does not give any *information* on the value of ***y*2**, and vice versa. Independence can be properly defined by the *probability density functions* (**pdf**). Let us denote by *p(y1,y2)* the joint probability density function (pdf) of ***y*1** and ***y*2**. Let us further denote by *p*1(*y*1) the *marginal* *pdf* of ***y*1**, i.e. the pdf of ***y*1** when it is considered alone, and similarly for ***y*2**. Then we define that ***y*1** and ***y*2** are **independent** if and only if the joint pdf is *factorizable* in the following way:



*p*(*y*1,*y*2) = *p*1(*y*1)*p*2(*y*2).

We can use MATLAB 'hist' and 'scatterhist' functions to visualize the *pdf*'s for two random variables with normal distribution:

Random variables:

y1 = random('normal',0, 10,[1,1000]);

y2 = random('normal',15, 30,[1,1000]);

Best bin number for the distribution according to Scott's rule (to find this number we use a *hidden* MATLAB function):

y1\_bins = internal.stats.histbins(y1);

y2\_bins = internal.stats.histbins(y2);

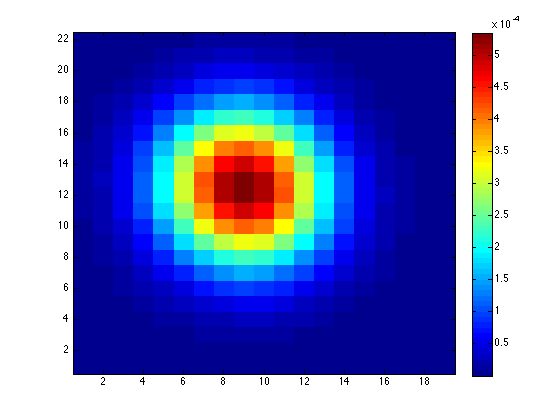
y1\_nbins = length(y1\_bins);

y2\_nbins = length(y2\_bins);

2D scatterplot with empirical marginal densities as histograms (top):

scatterhist(y1,y2,'nbins',[y1\_nbins,y2\_nbins])

y12 = [y1;y2];

y1\_pdf = hist(y1,y1\_nbins)

y2\_pdf = hist(y2,y2\_nbins)

Empirical joint density (middle):

y12\_pdf\_joint = hist3(y12',[y1\_nbins,y2\_nbins])

Normalize the density so that the P integral is 1:

y12\_pdf\_joint = ...

y12\_pdf\_joint/trapz(trapz(y12\_pdf\_joint),2);

Joint\_pdf\_1 = figure; imagesc(y12\_pdf\_joint)

Calculated joint density from product of empirical marginal densities (bottom):

y12\_pdf\_prod = y1\_pdf'\*y2\_pdf % outer product

y12\_pdf\_prod = ...

y12\_pdf\_prod/trapz(trapz(y12\_pdf\_prod),2);

Prod\_pdf\_1 = figure; imagesc(y12\_pdf\_prod);

Theoretical joint density from theoretical marginal densities:

mu = [0 15]; Sigma = cov(y12');

[Y1,Y2] = meshgrid(y1\_bins,y2\_bins);

F = mvnpdf([Y1(:) Y2(:)],mu,Sigma);

F = reshape(F,y2\_nbins,y1\_nbins);

Prod\_pdf\_surf = figure;

h = surf(y1\_bins,y2\_bins,F);

xlabel('x1'); ylabel('x2'); zlabel('Probability Density');

Prod\_pdf\_calc = figure;imagesc(get(h,'ZData')');set(gca,'Ydir','normal')

If we account for an error in sampling due to the small number of elements in the distributions, it becomes clear that in this case the *joint density* is very similar to the product of the *marginal densities*. Thus, the two random variables ***y*1** and ***y*2** are indeed *independent*, and therefore also *uncorrelated* as shown by the very small value of their empirical covariance calculated with the MATLAB function *cov*, or from the expression:

where *E(****y****)* is the *Expected* value (the mean) of ***y***:

Ey1y2 = y12(1,:)\*y12(2,:)'/nobs

Ey1Ey2 = sum(y12(1,:),2)/nobs\*sum(y12(2,:),2)/nobs

emp\_cov = Ey1y2 - Ey1Ey2

cov(y12,1)

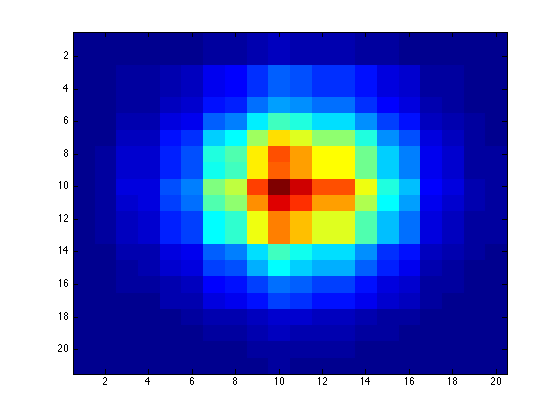
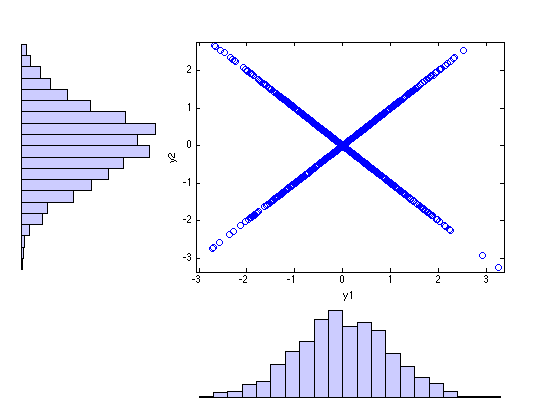
However, there are cases in which ***y*1** and ***y*2** are *uncorrelated*, but not *independent*. Consider the two distributions:

1st normal distribution:

y1 = random('normal',0, 1,[1,1000]);

chsign = binornd(1,0.5,1,1000);

chsign(chsign==0)=-1;



2nd normal distribution:

y2 = chsign.\*y1;

y12 = [y1;y2];

cov(y12',1)

Scatter + marginal densities plot:

Dep\_distr = figure; scatterhist(y1,y2)

y1\_nbins = length(internal.stats.histbins(y1));

y2\_nbins = length(internal.stats.histbins(y2));

Empirical joint density:

y12\_pdf\_joint = hist3(y12',[y1\_nbins,y2\_nbins])

y12\_pdf\_joint = ...

y12\_pdf\_joint/trapz(trapz(y12\_pdf\_joint),2);

Joint\_pdf\_1 = figure; imagesc(y12\_pdf\_joint)

Product of marginal densities:

y1\_pdf = hist(y1,y1\_nbins)

y2\_pdf = hist(y2,y2\_nbins)

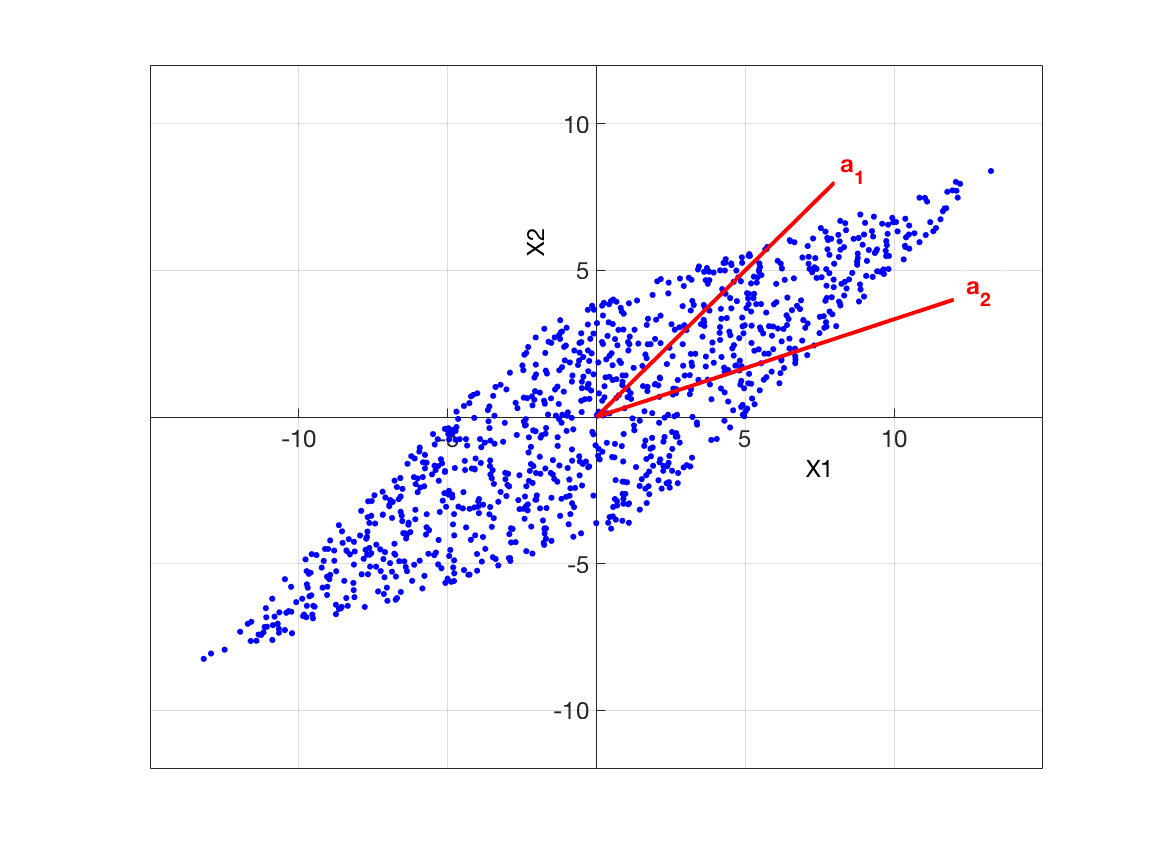
y12\_pdf\_prod = y1\_pdf'\*y2\_pdf

y12\_pdf\_prod = ...

y12\_pdf\_prod/trapz(trapz(y12\_pdf\_prod),2);

Prod\_pdf\_1 = figure; imagesc(y12\_pdf\_prod)

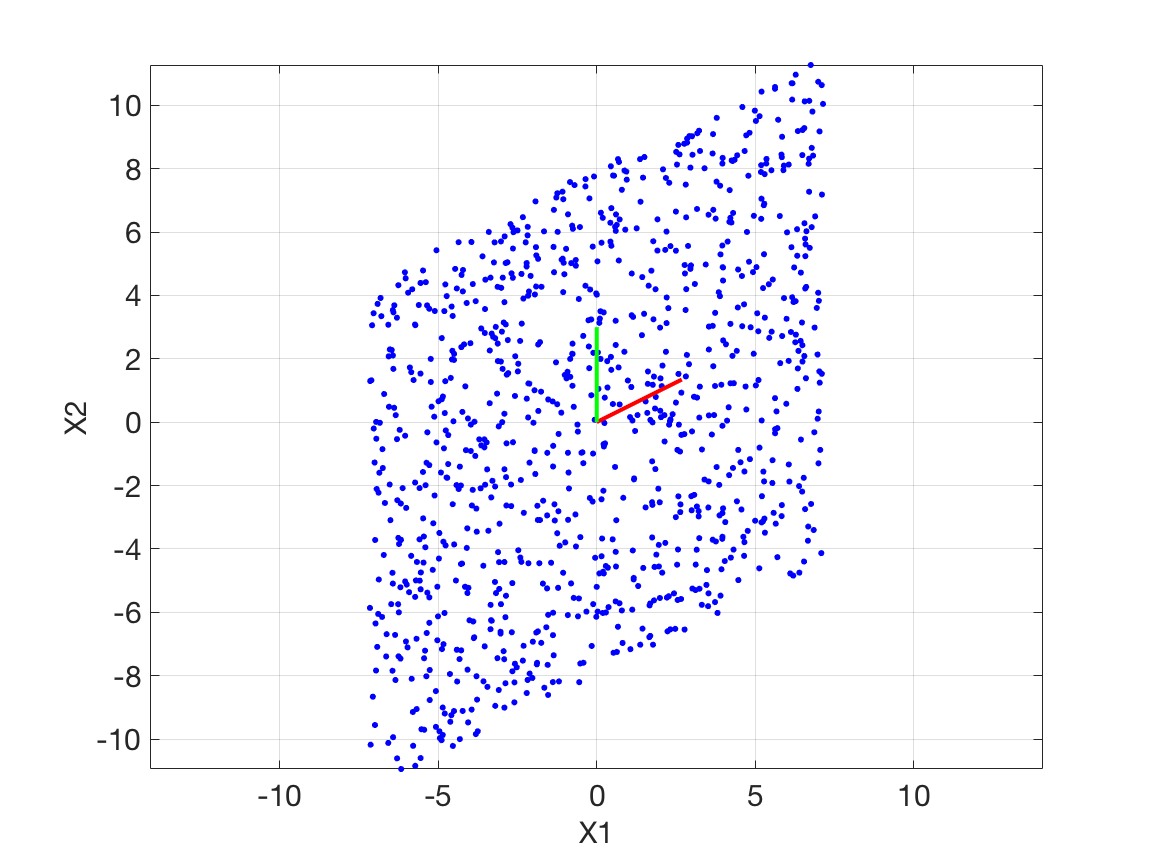
In this case the *joint density* is NOT equal to the product of the *marginal densities*. Thus, the two random variables ***y*1** and ***y*2** are NOT *independent*, despite having the same normal distribution and being *uncorrelated* as shown by the very small value of their empirical covariance. More generally, the variables ***y*1** and ***y*2** are said to be *independent* if information on the value of ***y*1** does not give any information on the value of ***y*2**, and vice versa.

Independent Component Analysis (**ICA**) is an *optimization* technique that (i.e, in this simple case) aims to identify a space with basis ***A*** =[***a1,a2***]**,** in which the variables ***x1***, ***x2*** are represented (*change of basis*) as variable ***s1***, ***s2*** that are maximally *independent*. ***s*** is obtained via a change of basis into ***A*** space as:

and the way back from ***A*** space to standard space is of course given by:

For example, it is intuitively clear why the two variables ***x*1** and ***x*2** in the left panel above are dependent: an easy way to see this is to consider whether it is possible to predict some value of one of them, say ***x*2**, from some value of the other. Clearly if ***x*1** attains one of its maximum or minimum values, then this completely determines the value of ***x*2**. They are therefore NOT independent. For variables ***s1*** and ***s2*** the situation is going to be very different: knowing the values of ***s1*** does not in any way help in guessing the values of ***s2***.

Here, we will first introduce the **ICA** method with some simple examples and then describe some general applications.



We start by creating an ensemble of points with ***x1*** and ***x2*** coordinates represented by linear combinations of random *uniform* distributions.

a1 = [1;0.5]; a2 = [0;1.0];

a1 = a1/norm(a1); a2 = a2/norm(a2);

cos\_theta = …

a1'\*a2;acos(cos\_theta)\*180/pi

s1 = random('uniform',-8 , 8,[1,1000]);

s2 = random('uniform',-8 , 8,[1,1000]);

Linear combinations of the uniform vectors:

X = [a1 a2]\*[s1;s2];

ICA\_1 = figure;

set(ICA\_1,'Unit','Normalized','Position',[0 0.2 0.6 0.8])

plot(X(1,:),X(2,:),'.','Linewidth',0.5,'MarkerEdgeColor','b',...

'MarkerSize',15,'MarkerFaceColor','g')

line(1\*[0 3\*a1(1)],1\*[0 3\*a1(2)],'LineWidth',4,'Color','red');

line(1\*[0 3\*a2(1)],1\*[0 3\*a2(2)],'LineWidth',4,'Color','green');

xlabel('X1');ylabel('X2');grid on;

box on;axis equal;hold on

PCA:

[E,~,D] = pca(X');

line(1\*[0 E(1,1)],1\*[0 E(2,1)],'LineWidth',2.5,'Color','yellow');

line(1\*[0 E(1,2)],1\*[0 E(2,2)],'LineWidth',2.5,'Color','magenta');

Notice that each pair of *x1*,*x2* (or x,y) coordinates for a point of the ensemble derives from the linear combination of two vectors (***a1*** = [1 0.5] and ***a2*** = [0 1]) pointing in the directions represented by the red and green line, with the coefficients for the combination provided by the two uniform distributions ***s1*** and ***s2***:

***X A S***

These two directions may be very different from the *principal directions* of the ensemble (vectors ***e1*** and ***e2***, as identified by PCA) represented by the yellow and magenta lines.

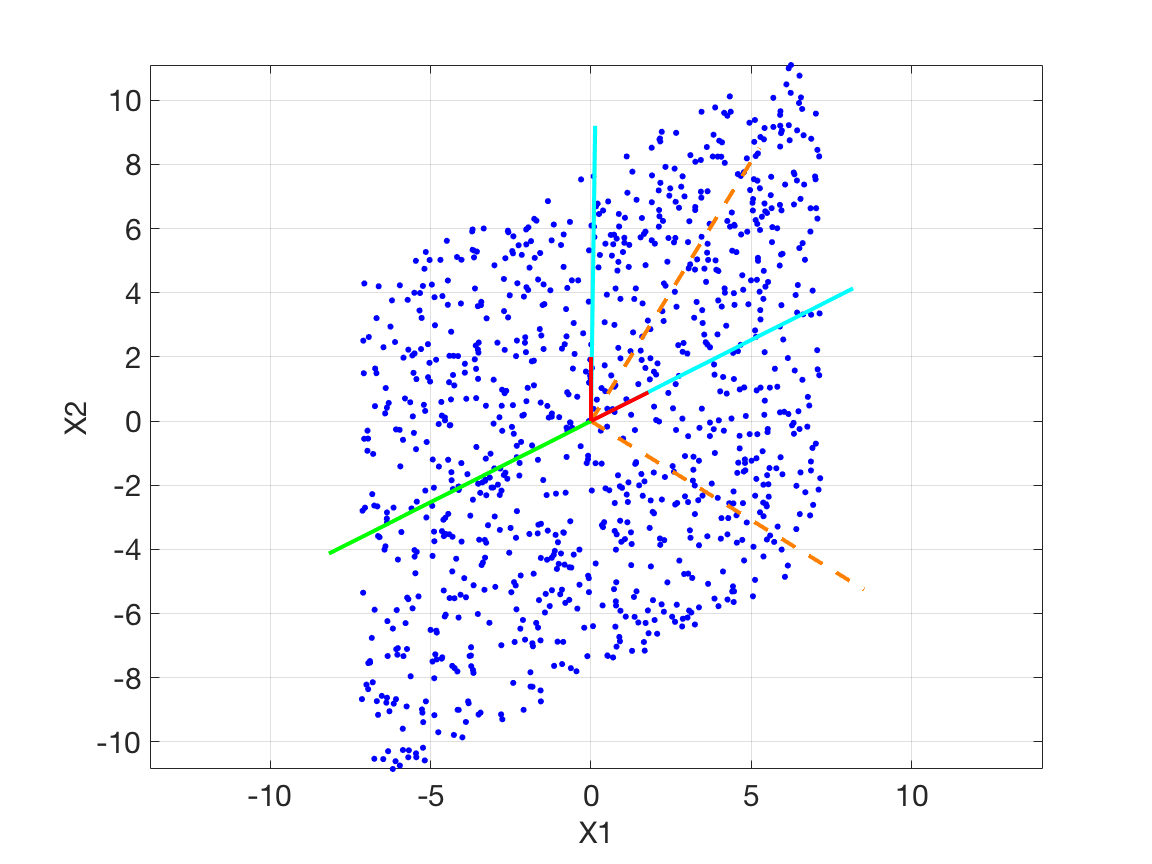
Suppose we are only given the final coordinates of the generated ensemble of points, and we are interested in finding out the (possibly non-orthogonal) vectors whose linear combination produces all the points of the ensemble, and the two independent (possibly non-orthogonal) vectors that provide all the coefficients for the linear combinations. This is the same as solving the general matrix equation ***As*** = ***x***, with the additional complication that in this case we only know ***x***, and both ***A*** and ***s*** are unknown. It is worth noting that an alternative way of interpreting the matrix equation shown above is to consider the *rows* of ***X*** as originating from a linear combinationof the *rows* of ***S***, with the coefficients of the combination provided by the *rows* of ***A***.For this reason, in the languange of *independent component analysis* (**ICA**), ***A*** (*m* x *m*) is called the *mixing* matrix, and the *m* rows of ***S*** (*m* x *n*) are the *independent components* whose mixing generates the *experimental data* ***X*** (*m* x *n*):

***X A S***

It follows that ***S*** can be obtained by left-multiplying *both sides* by*a separating* matrix ***W*** = ***A-1***:

***S W*** *=* ***A-1 X***

Again, conceptually this is just a simple change of basis to represent ***X*** in ***A*** space. Several algorithms (many of which are listed in http://perso.telecom-paristech.fr/~cardoso/icacentral/) have been developed to identify both the separating matrix and the independent components in different applications. As examples of ICA methods, here we use the toolboxes ***FASTICA*** (<http://research.ics.aalto.fi/ica/fastica/>) and ***RADICAL*** (<http://people.cs.umass.edu/~elm/ICA/)>, (without this representing any endorsement of these toolboxes as superior to others). The following is their application to the analysis of our ensemble of points:



FASTICA:

[S1, A1, W1] = fastica(X)

RADICAL:

[S3, W3] = RADICAL(X)

A3 = inv(W3)

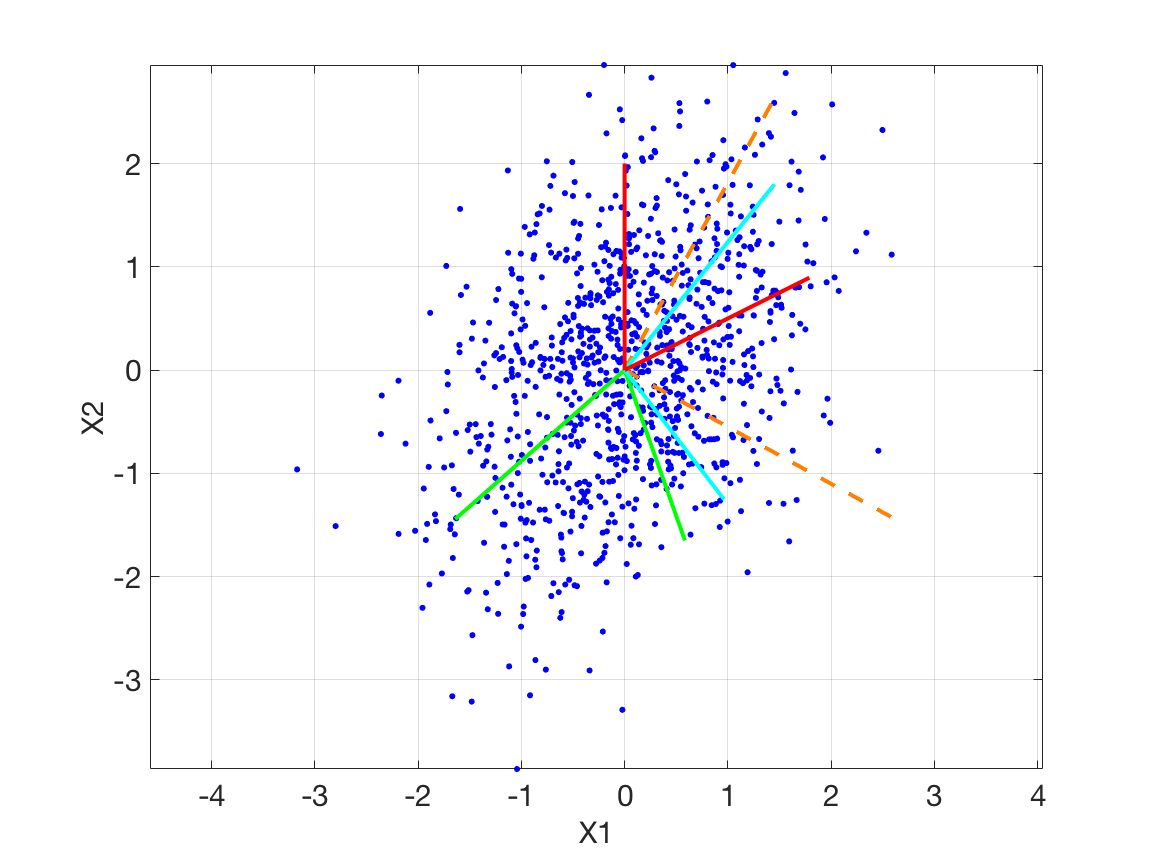
The vectors of the mixing matrix ***A*** identified by ICA (green lines) or RADICAL (cyan lines) correspond almost exactly to the columns of original mixing matrix (red lines) used to generate the ensemble. We will not describe in detail any of the ICA algorithms here, but will only mention that the fundamental restriction in ICA is that the *independent components* (the row vectors ***si*** of ***S***) must be non-gaussian for the method to be successful, which is why we used *uniform* rather than *normal* distributions to generate the ensemble. To see why this is so, we can generate another ensemble of points, in which the vectors ***si*** are gaussian, uncorrelated, and of unit variance.

Normal distributions:

s1 = random('normal',zeros(1,1000),1.0);s2 = random('normal',zeros(1,1000),1.0);

cov(s1',s2'),corr(s1',s2')

Linear combinations of the gaussian vectors:

X = [a1 a2]\*[s1;s2];

PCA:

[E,~,D] = pca(X');

ICA:

[S,W] = RADICAL(X)

A = inv(W)

As shown on the side, the vectors of the mixing matrix identified by ICA (cyan and green lines) do not match well the columns of the original mixing matrix (red lines) used to generate the ensemble. In other words, the true matrix ***A*** (and therefore the true matrix ***S****)* is not uniquely identifiable for gaussian independent components.

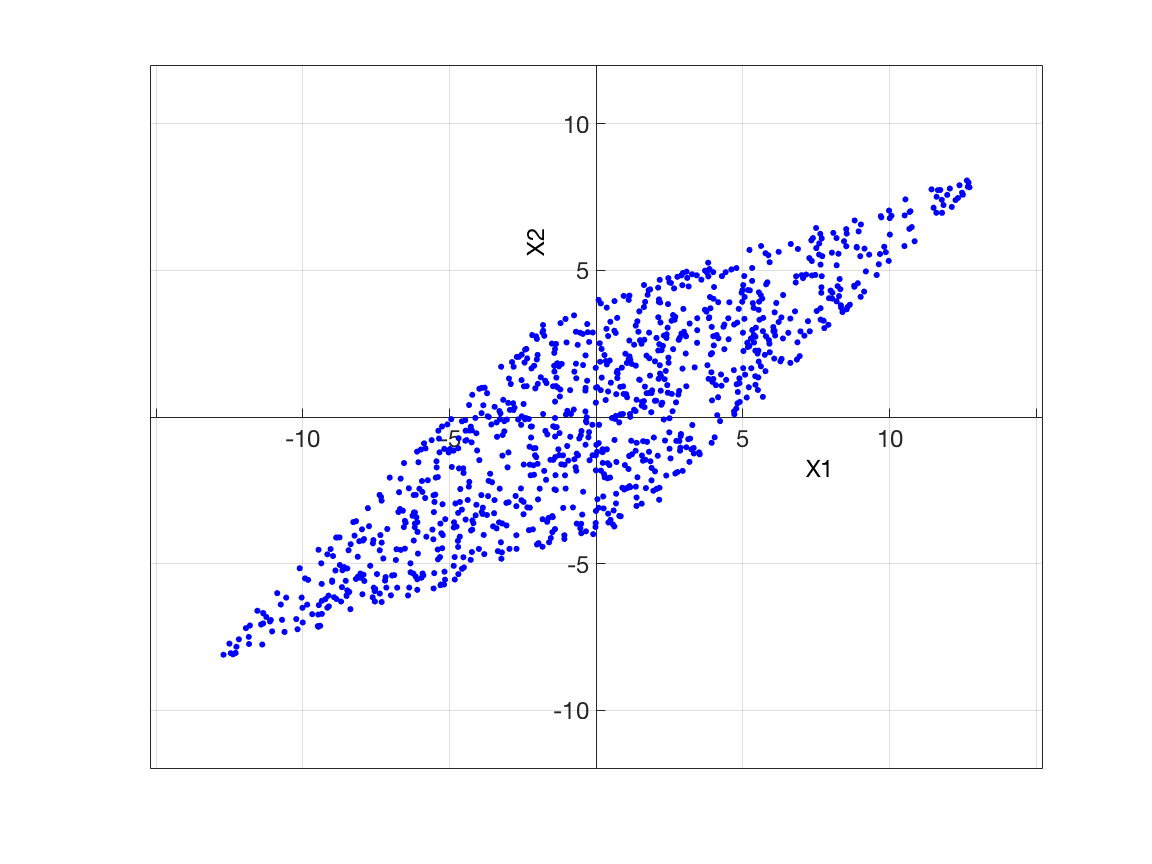
This result tell us something important about the operating principle of ICA:

Searching for maximally *independent* components is equivalent to searching for maximally *non-gaussian* components!

A very important measure of non-gaussianity is given by *negentropy*. Negentropy is based on the information theoretic quantity of entropy, the basic concept of information theory (CHAPTER 20). The entropy of a random variable can be interpreted as the degree of information that the observation of the variable gives. A fundamental result of information theory is that *a gaussian variable has the largest entropy among all random variables of equal variance*. In fact, entropy is small for distributions that are clearly concentrated on certain values, i.e., when the variable is clearly clustered, or has a pdf that is very “spiky”. To obtain a measure of nongaussianity that is zero for a gaussian variable and always nonnegative, a slightly modified version of the definition of entropy, called negentropy, *J*, is defined as follows:

*J*(***y***) = *H*(***ygauss***)−*H*(***y***)

where ***ygauss*** is a Gaussian variable with the same variance as ***y*** . Due to the above-mentioned properties, negentropy is always non-negative, and it is zero if and only if ***y*** has a Gaussian distribution.

The following is a step by step demonstration of how a typical ICA algorithm works in a simple case with only two variables. We start as before by generating two uniform distributions between a lower and an upper bound.

pd1 = makedist('uniform','lower',-3,'upper',3);

s1 = random(pd1,1,1000);

pd2 = …

makedist('uniform','lower',-2.5,'upper',2.5);

s2 = random(pd2,1,1000);

S = [s1;s2];

A = [2 3;2 1];

X = A\*S;

ICA\_5 = figure;

set(ICA\_5,'Unit','Normalized','Position',…

[0 0.2 0.6 0.8])

plot(X(1,:),X(2,:),'.','Linewidth',0.5,…

'MarkerEdgeColor','b','MarkerSize',15,…

'MarkerFaceColor','g')

xlabel('X1');ylabel('X2');

xlim([-15 15]);ylim([-12 12]);

ax = gca;

ax.XAxisLocation = 'origin';

ax.YAxisLocation = 'origin';

box on, grid on

The first step in ICA is to center ***X***, i.e. subtract its mean vector ***mX*** = *E*(***X***) so as to make each ***xi*** a zero-mean variable. This implies:

mX = mean(X,2);

cX = X - mX(:,ones(1,1000));

After estimating the separating matrix ***W*** = ***A****−1*with centered data, we will complete the estimation by adding the mean vector of ***S*** back to the centered estimates of ***S, cS***. The mean vector of ***S*** is given by ***mS*** = ***A****−1****mX***:

The next step in ICA is to *whiten* the centered array . This means to transform into a new array whose components have unit variance and 0 covariance. In other words, the covariance matrix of equals the identity matrix: . One popular method for whitening is to use the *eigen-decomposition* of the covariance matrix , where is the orthogonal matrix of eigenvectors and is the diagonal matrix of eigenvalues. Whitening can be done by . It is easy to check that now .

In general, it may be useful to reduce the dimension of the data before whitening. We look at the eigenvalues of *E*(**c*XcXT***) and discard those that are too small, as is done in PCA. This has the effect of reducing noise. In this particular example, there are only two variables, and PCA is really not necessary. Notice that we also set the whitened data as global variable so we can pass it to the minimizer.

[E,~,D] = pca(cX');

white = (E\*diag(D.^-0.5)\*E')

global X\_white

X\_white = white\*cX;

cov(X\_white')

ICA\_6 = figure;

set(ICA\_6,'Unit','Normalized','Position',[0 0.2 0.6 0.8])

plot(X\_white(1,:),X\_white(2,:),'.','Linewidth',0.5,'MarkerEdgeColor','b',...

'MarkerSize',15,'MarkerFaceColor','g')

xlabel('X1');ylabel('X2');

xlim([-2.5 2.5]);ylim([-2.5 2.5]);

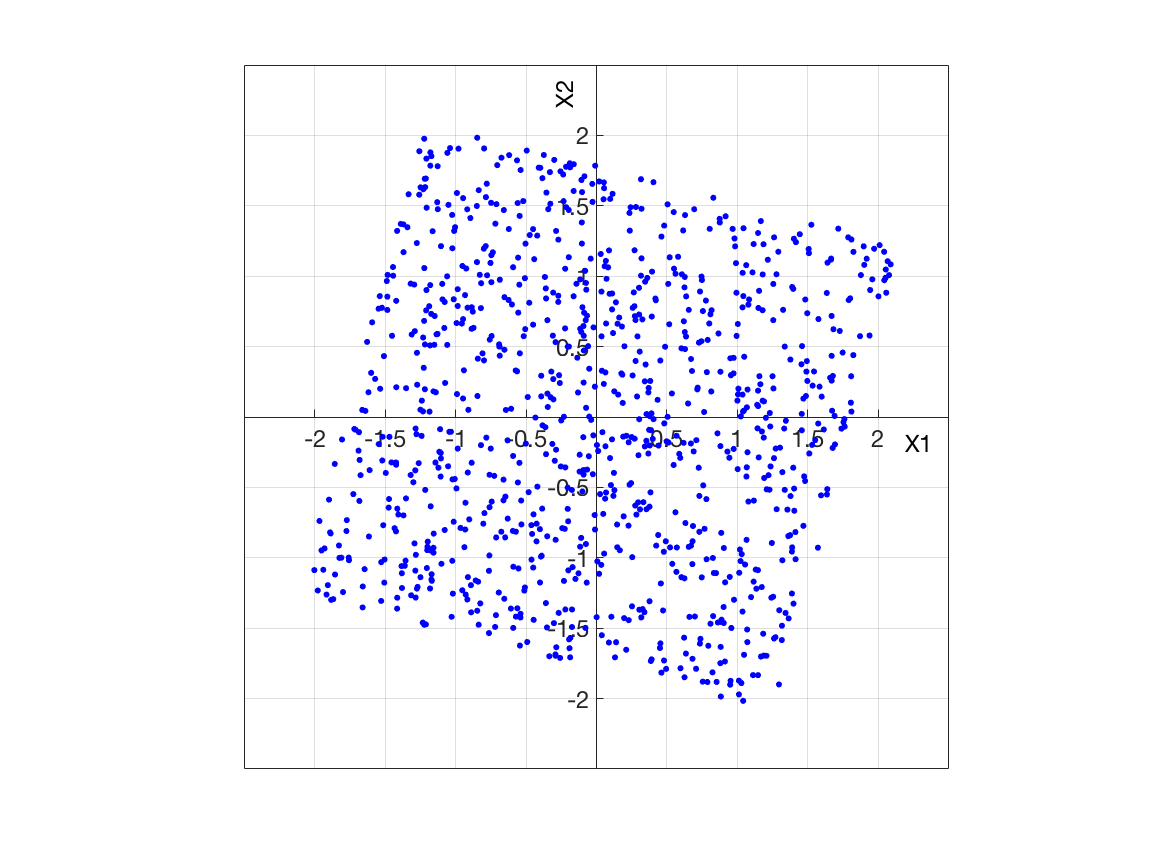
ax = gca;

ax.XAxisLocation = 'origin';

ax.YAxisLocation = 'origin';

box on, grid on

axis equal

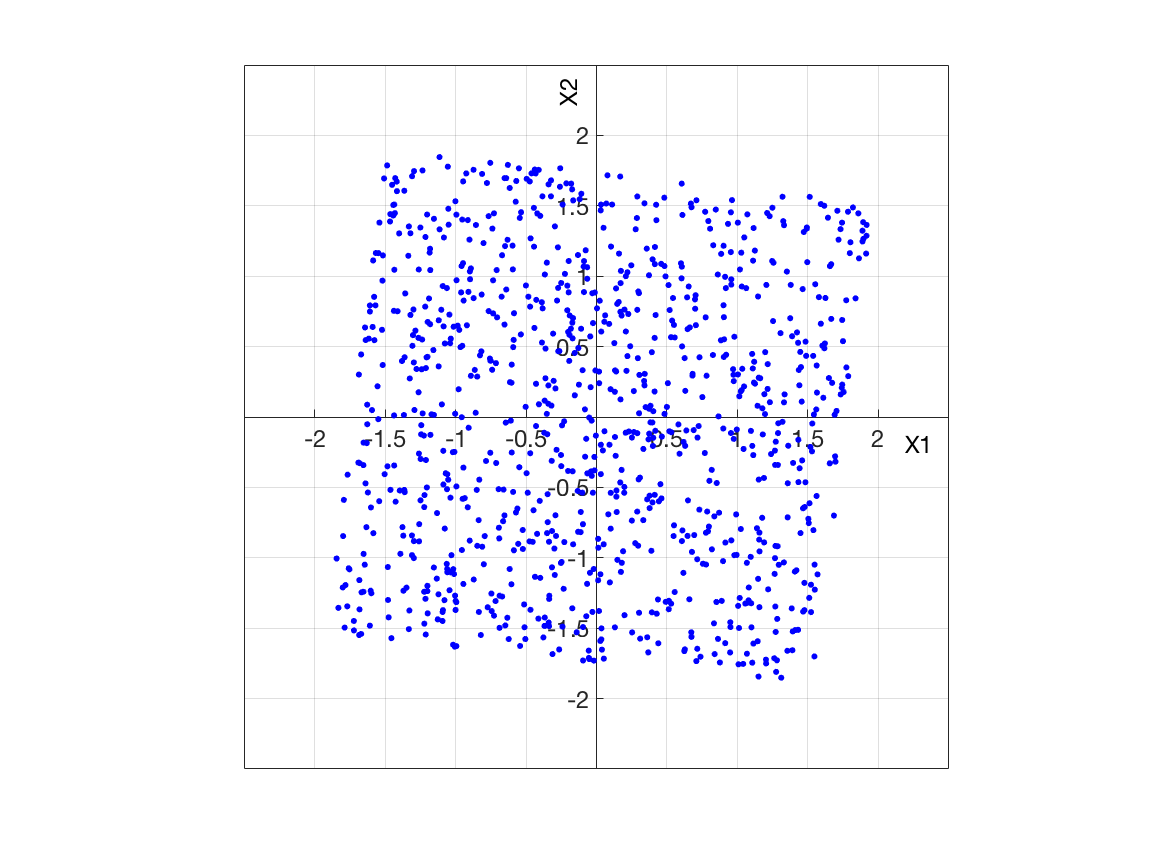
Upon whitening, the square defining the distribution is at this point a rotated version of the final square defined by the independent components. All that is left to do is the estimation of a single angle that gives the rotation.

The identification of this angle is an *optimization* process based on *maximizing* the non-gaussianity of the independent components. Various algorithms for ICA differ by the criterion used in evaluating non-gaussianity, and in the optimization method used to converge rapidly on the correct angle. One possible type of optimization is based on *minimizing* the *mutual information* (**MI**) between variables. It suffices to say here that given two random variables, **MI** provides a measure of our capacity to predict the value of one variable knowing the value of the other variable. MI assumes values between 0 (complete *independence*) and 1 (complete *dependence*), and thus, by minimizing it, we search for components that are more *independent*. In particular, it can be shown that finding an invertible transformation ***W*** thatminimizes the mutual information is equivalent to finding directions in which the *negentropy* is maximized .

We check the MI before any rotation is applied (angle = 0 degrees) using the function *mi\_from\_angle*.

MI = mi\_from\_angle(0,X\_white)

> MI = 0.2769



Here we test the effects on MI of a rotation by a small angle.

phi = -8;

Q = [cosd(phi) sind(phi);-sind(phi) cosd(phi)];

X\_q = (Q\*X\_white);

ICA\_7 = figure;

set(ICA\_7,'Unit','Normalized','Position',[0 0.2 0.6 0.8])

plot(X\_q(1,:),X\_q(2,:),'.',…

'Linewidth',0.5,'MarkerEdgeColor','b',...

'MarkerSize',15,'MarkerFaceColor','g')

xlabel('X1');ylabel('X2');

xlim([-2.5 2.5]);ylim([-2.5 2.5]);

ax = gca;

ax.XAxisLocation = 'origin';

ax.YAxisLocation = 'origin';

box on, grid on

MI = mi\_from\_angle(phi,X\_white)

> MI = 0.1819

Here we set up a *multistart* minimization to avoid remaining trapped in local minima: for this purpose we use MATLAB *fminsearch* function, which calls *mi\_from\_angle* at each step of the minimizer.

start\_vec = [-3:-3:-45];

nstart = length(start\_vec);

phi = zeros(1,nstart);

fval = zeros(1,nstart);

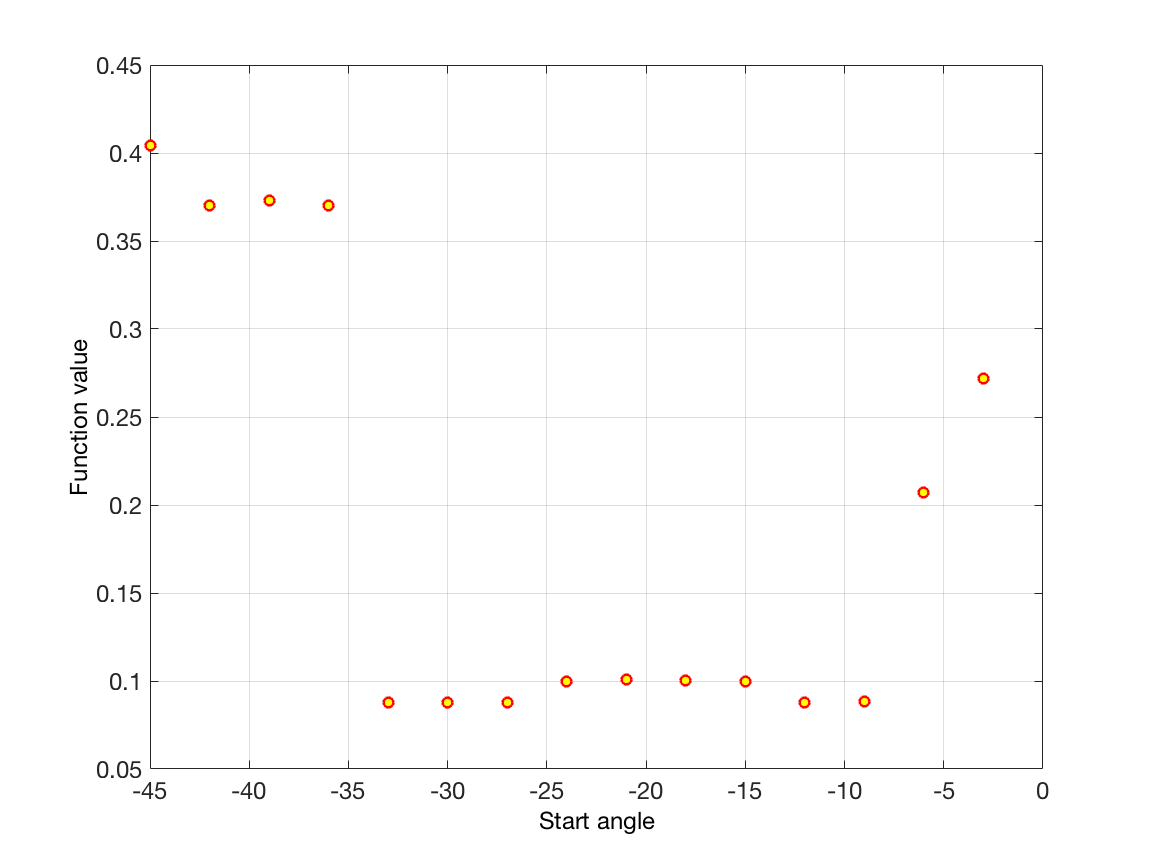
options = optimset('TolFun',1e-8, 'TolX',1e-8);

for i = 1:nstart

start\_point = start\_vec(i);

[phi(i),fval(i),~,~] = fminsearch(@mi\_from\_angle,start\_point,options);

end

Here we plot the multistart search and identify the rotation angle associated with the smallest value of MI.

ICA\_8 = figure;

set(ICA\_8,'Unit','Normalized','Position',[0 0.2 0.6 0.8]);

scatter(start\_vec,fval,100,'r','LineWidth', 2.5,'MarkerFaceColor','y');

box on, grid on

xlabel('Start angle');ylabel('Function value')

[~,min\_ind] = min(fval);

best\_phi = phi(min\_ind);

Best rotation

Q = [cosd(best\_phi) sind(best\_phi);-sind(best\_phi) cosd(best\_phi)];

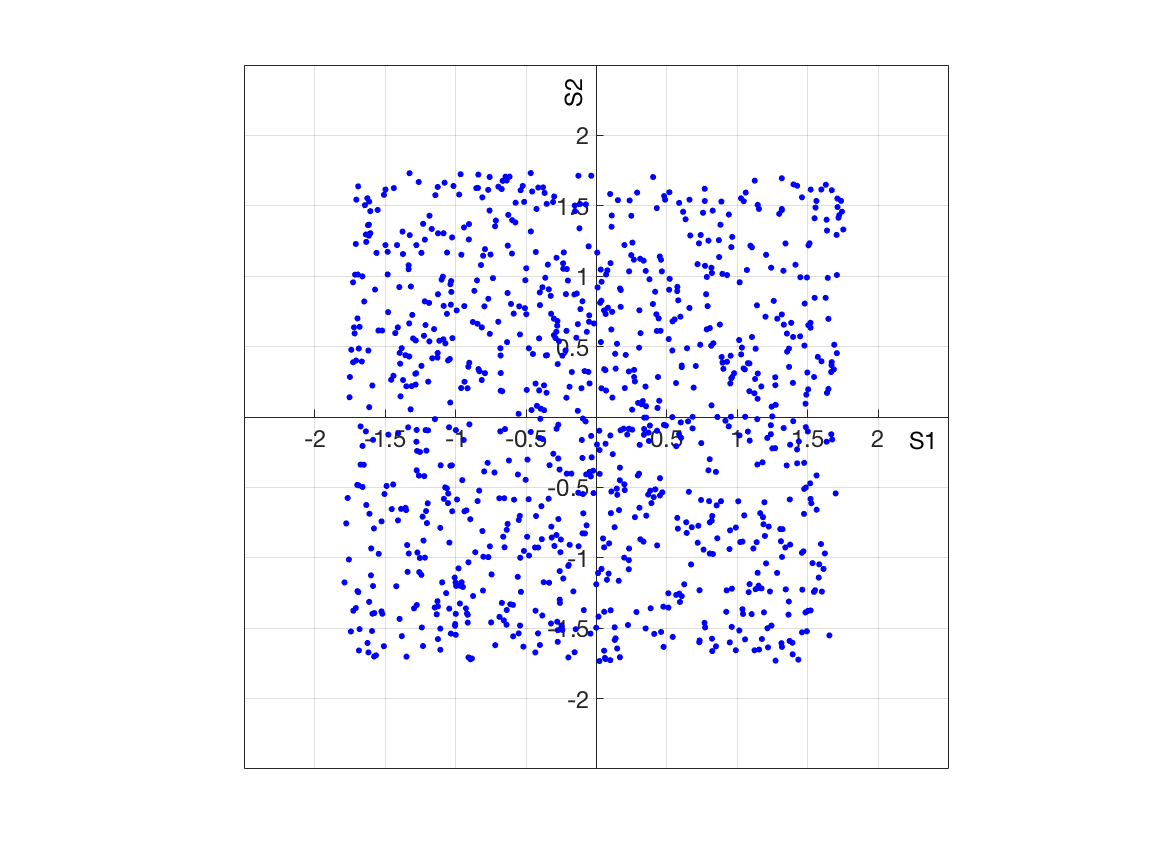
Here we combine whitening and rotation to obtain the *separating matrix* ***W*** = ***A-1***.

W = Q\*white

Independent components and empirical mixing matrix from ICA.

Z = W\*X;

A\_ica = inv(W)

S\_ica = Z + W\*mX(:,ones(1,1000));

Compare this result with that from FASTICA:

[~, A\_fastica, ~] = fastica(X);

ICA\_9 = figure;

set(ICA\_9,'Unit','Normalized','Position',[0 0.2 0.6 0.8])

plot(S\_ica(1,:),S\_ica(2,:),'.',…

'Linewidth',0.5,'MarkerEdgeColor','b',...

'MarkerSize',15,'MarkerFaceColor','g')

xlabel('S1');ylabel('S2');

xlim([-2. 2.]);ylim([-2. 2.]);

ax = gca;

ax.XAxisLocation = 'origin';

ax.YAxisLocation = 'origin';

box on, grid on

axis equal

Finally, since independent components can be estimated only up to a sign, it’s important to remember that there are always 2local maxima for each independent component ***si***, corresponding to ***si***and **−*si***.

What are the situations in which we would want to use ICA rather than PCA? A simple example helps understand this point. Imagine you are at a cocktail party in a room where two of the guests are debating some important issue. Unfortunately, they are speaking on top of each other, so nobody can understand what they are saying. However, in the room there are two microphones, each positioned closer to one of the speakers: the microphones are recording two signals denoted ***x*1**(*t*) and ***x*2**(*t*), with *t as* the time index. Each of these recorded signals is a weighted sum of the speech signals emitted by the two debaters, denoted as ***s*1**(*t*) and ***s*2**(*t*). We could express this as a linear equation:

***x*1**(*t*) = *a*11***s*1** +*a*12***s*2**

***x*2**(*t*) = *a*21***s*1** +*a*22***s*2**

where *a*11, *a*12, *a*21, and *a*22 are some parameters that depend on the distances of the microphones from the speakers. In matrix form this problem becomes:

***X A S***

It would be very useful to derive an estimate of what each individual is really saying (that is, ***s*1**(*t*), ***s*2**(*t*)), using the recorded signals ***x*1**(*t*), ***x*2**(*t*). This is called the *cocktail-party problem*.

Independent component analysis was originally developed to deal with problems that are closely related to the cocktail-party problem, and belongs to the class of algorithms for *blind source separation*. Consider, for example, electrical recordings of brain activity as given by an electroencephalogram (EEG). The EEG data consists of recordings of electrical potentials in many different locations on the scalp. These potentials originate from the mixing of several underlying components of the brain activity. This situation is quite similar to the cocktail-party problem: we would like to find the original components of the brain activity, but we can only observe mixtures of the components. ICA can reveal interesting information on brain activity by giving access to its independent components. EEGLAB, a popular MATLAB Toolbox for ICA analysis of EEG recordings is freely available and can be downloaded from <https://sccn.ucsd.edu/eeglab/downloadtoolbox.php>. We will return in CHAPTER 12 to the ICA techniques of blind source separation when we discuss the analysis of microarray gene expression data.

**PRACTICE**

**1.** We recall that the *correlation* matrix is the same as the *covariance* matrix calculated on normalized (*zscored*) data. A normalized vector is easily obtained from the original unnormalized one using MATLAB *zscore* function. Therefore, PCA can be carried out using a *correlation* matrixinstead of a *covariance* matrix.

a. With this information in mind, write a MATLAB code that will reproduce the PCA analysis of the 24 physico-chemical properties of 20 commonly used drugs described in the article ‘Pharmacological Classification of Drugs by Principal Component Analysis Applying Molecular Modeling Descriptors and HPLC Retention Data’, *J Chromatogr Sci*. 2011 Nov-Dec;49(10):758-63.

b. Explain the physical meaning of the eigenvectors in this case.

**2.** A simple method to *whiten* an array ***X*** of ***m*** variables and ***n*** observations to obtain a new array whose components are uncorrelated and with variance = 1 is to use the *eigen-decomposition* of the covariance matrix *E*{**XX***T*} = ***EDET***, where ***E*** is the orthogonal matrix of eigenvectors of *E*{***XXT***} and ***D*** is the diagonal matrix of its eigenvalues. Whitening is done by:

= ***ED−1/2ETX*** .

a. By replacing ***ED−1/2ETX*** into write an algebraic (non-numeric) proof that now .

b. Generate a 2-dimensional array using 2 pdf's of your choice (i.e. one logistic and one uniform), and present numerical evidence that by whitening the array the variables become uncorrelated and with unit variance.