Chapter 6

Singular Value Decomposition of Spectroscopic Data

6.1 Introduction

If you've spent any time talking to Jahan you will surely hear him praise the usefulness of the Fourier transform. I would argue that a close runner up in terms of utility for data analysis is singular value decomposition (SVD). It is an algorithm that can be used to decompose a two-dimensional data matrix which contains signals from multiple species, e.g. chemicals with overlapping absorption spectra, into individual spectra. I have used it in my analyses of the pKas of quinoline compounds (chapter 3) from pH titration data and for the extraction of time constants from from ultrafast transient absorption data.

First I will introduce the algorithm with a minimal example, a small matrix of arbitrary column vectors. Next, an example of using SVD for image compression is given, which I believe helps to illustrate how it works to find redundancies in data sets. Next, I will construct a fake, but realistic looking data set and demonstrate how to extract chemically meaningful information from it. Finally, I will outline a scenario where SVD is an inappropriate method of analysis that may

lead to confusing results.

A basic understanding of linear algebra is required to implement SVD. An excellent refresher on the fundamentals of linear algebra is covered here (CITE) and is highly recommended reading. Below, I will demonstrate the utility of SVD by guiding the reader through a minimal example and then through a simple, simulated spectroscopic data set. These examples are done with the MATLAB software package. Any text that is in the monospaced font may be copied and pasted into the MATLAB environment for experimentation.

Please see this chapter's bibliography for a list of recommended reading on all things SVD.

6.2 Minimal example

Consider the column vectors

```
v1 = [1 ; 1 ; 1 ; 1 ; 1];
v2 = [0 ; 0 ; 0 ; 0 ; 1];
v3 = [1 ; 2 ; 2 ; 2 ; 2];
```

and a matrix constructed from these vectors

```
A = [v1 v1*2 v1*3 v1*4 v2 v2+v1 v3]
```

```
A =

     1     2     3     4     0     1     1
     1     2     3     4     0     1     2
     1     2     3     4     0     1     2
     1     2     3     4     0     1     2
     1     2     3     4     1     2     2
```

These vectors are **linearly independent** from each other, which you may verify on your own. The matrix, A, has dimensions

```
size(A)

ans =

     5     7
```

and rank

```
rank(A)

ans =

     3
```

The rank of a matrix is the number of linearly independent vectors needed to represent it. It should be relatively clear that in this case it should be 3 beause of the way A was constructed.

Matrices that have a rank lower than their smallest dimension are said to be **rank deficient**. Matrices with a rank equal to their smallest dimension are said to be **full rank**. Another way to think about rank deficient matrices is that they contain redundant information. Vectors resulting from multiplication by a constant, or from addition of two vectors don't contain new information. E.g. instead of sending someone all of the *numbers* in A, you can send them the three *linearly independent vectors* in A, and the *rules* for consructing A. The essence of SVD is to find these rules, when only A is known.

6.3 Addition of noise

Now let's define matrix with small random numbers. This is a simulation of noise, which is present in any experimental data set.

```
% Seed rng with a constant so the numbers come out the same each time the
% code is run.
rng(1);
N = (rand(size(A))-0.5).*0.2
```

```
N =

   -0.0166   -0.0815   -0.0162    0.0341    0.0601    0.0789   -0.0803
    0.0441   -0.0627    0.0370   -0.0165    0.0937   -0.0830   -0.0158
   -0.1000   -0.0309   -0.0591    0.0117   -0.0373   -0.0922    0.0916
   -0.0395   -0.0206    0.0756   -0.0719    0.0385   -0.0660    0.0066
   -0.0706    0.0078   -0.0945   -0.0604    0.0753    0.0756    0.0384
```

What happens when a small amount of noise is added to a rank deficient matrix?

```
rank(A+N)
```

```
ans =

     5
```

We see that **addition of noise makes the matrix full rank.** This fact is important when analysing a real experimental data set, which will always contain noise.

6.4 Rank approximation

Is there a way to approximate the rank of a matrix, which we know to be rank deficient, but contains noise? Singular value decomposition is one method to achieve this.

SVD is easily applied in MATLAB using the command

```
[U,S,V] = svd(A+N);
```

SVD breaks the input matrix into three matries such that $A = U * S * V^{\dagger}$

V contains information about the row space of A, and U contains information about the column space of A. Since our matrix A was defined somewhat arbitrarily, the columns of U and V do not have much meaning. The meaning of these vectors will be more apparent in the following section when we examine a simulated spectral data set.

6.5 Singular values

S is a diagonal matrix which contains the weighted importance of the vectors in U and V, known as singular values.

```
S
```

```
S =
```

```
   12.8947         0         0         0         0         0         0
         0    2.9327         0         0         0         0         0
         0         0    0.9315         0         0         0         0
         0         0         0    0.1562         0         0         0
         0         0         0         0    0.0855         0         0
```

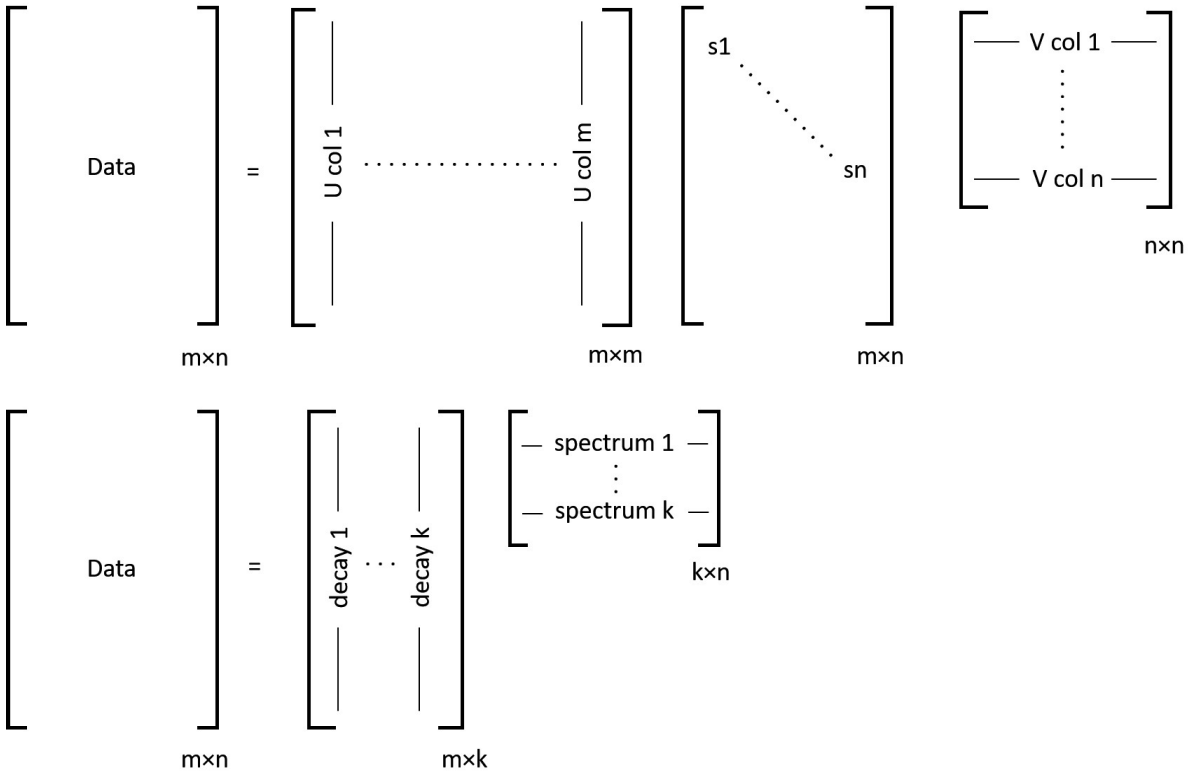
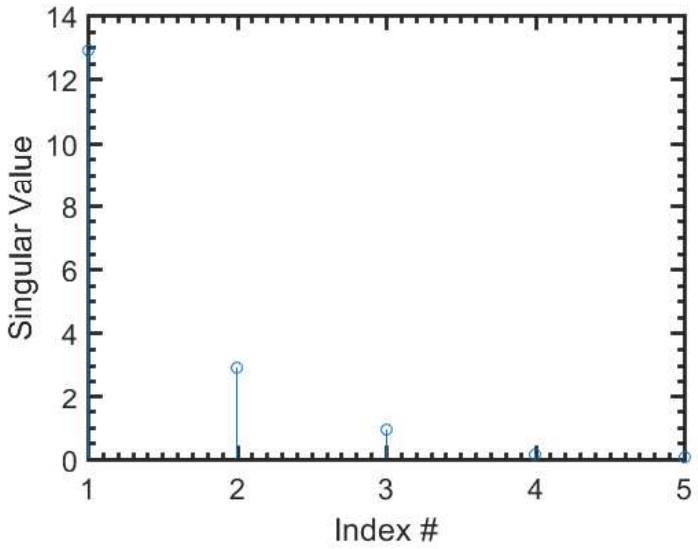


Figure 6.1: Dimensional analysis of data decomposition by SVD. Top: When the SVD algorithm is applied to a data matrix, D, three matrices are returned such that $D = U * S * V^{\dagger}$. U and V contains information about the column space and row space of D, respectively. S is a diagonal matrix which contains the singular values. Bottom: An example data set can be constructed by multiplying a set of spectra and decay profiles, where each row/column represents features of a chemical species. Some numerical work and often times assumptions are needed to turn SVD (top decomposition) into a chemically insightful form (bottom decomposition).

The sigular values can be interpreted as how important each vector is in creating the dataset.

```
stem(diag(S))
ylabel('Singular Value')
xlabel('Index #')
```



We see three large values, and two smaller ones. The three large values indicate that there are three linearly independent vectors which account for most of this dataset.

SVD can be thought of as a generalization of diagonalization for non-square matrices. The singular values, are similar to eigenvalues. We can make a rectangular matrix square by multiplying it by it's transpose, $AA^{\dagger}$, or vice versa, $A^{\dagger}A$. The square root of the eigenvalues of this new matrix are equivalent to the singular values obtained via SVD.

We will do this for the original matrix, A, and compare it's singular values to the noisy matrix, A+N

```
sqrt(eig((A*A')))
```

```
ans =


   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.8224 + 0.0000i
   2.8109 + 0.0000i
  12.9778 + 0.0000i
```

Now we see that, without noise, there are three non-zero, singular values, indicating the data set contains three linearly independent vectors.

How do we know whether a small singular value is due to noise or another linearly independent vector? A good rule of thumb is if it is 2% of the first singular value it is not significant. This approach is sensitive depending on the signal to noise ratio in a real experiment. More sophisticated methods for rejection of singular values are outlines in this paper (CITE)

Using the 2% rule we see that we can eliminate the last two singular values.

```
% Convert singlar values to percentages of the first value
sv_percent = diag(S)./S(1,1).*100


sv_percent =


  100.0000
   22.7434
    7.2240
    1.2116
    0.6629
```

We can now create a rank 3 approximation of our noisy data set by cutting out the last two columns of U and V, and the last two rows and columns of S. This cropping should preserve the original dimensions of the dataset. Alternatively, you can set all of the values in the last two columns to zero.

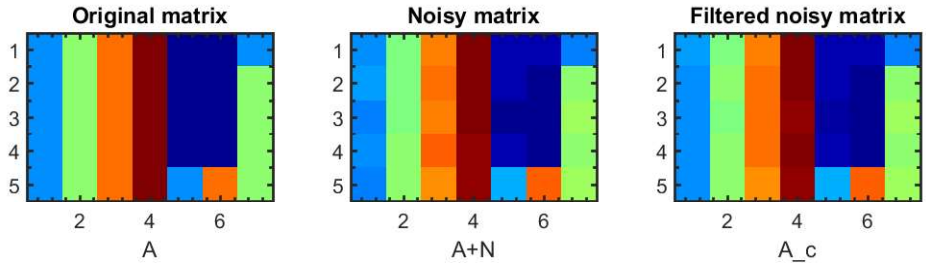
Crop the matrices at index n, for an n'th rank approximation.

```
n = 3;
U_c = U(:,1:n);
V_c = V(:,1:n);
S_c = S(1:n,1:n);

% Reconstruct the dataset with only the important singular values/vectors
% Don't forget to transpose V
A_c = U_c*S_c*V_c'

A_c =

    0.9907    1.9164    2.9893    4.0292    0.0668    0.0770    0.9195
    0.9709    1.9686    3.0269    3.9951    0.0339   -0.0647    1.9827
    0.9602    1.9565    3.0050    3.9554    0.0204   -0.1089    2.0888
    0.9663    1.9626    3.0160    3.9779    0.0338   -0.0657    2.0112
    0.9299    2.0077    2.9061    3.9391    1.0758    3.0755    2.0383
```

We find that the reconstructed matrix, A_c, has singificantly less noise than the original noisy matrix, A+N. They can be compared visually to see that some of the noise has been filtered out.

```
h = figure();
h = figure('Position', [100, 100, 1200, 275]);
```

```
subplot(131);
imagesc(A)
title('Original matrix')
xlabel('A')
subplot(132)
imagesc(A+N)
title('Noisy matrix')
xlabel('A+N')
subplot(133);
imagesc(A_c)
title('Filtered noisy matrix')
xlabel('A\_c')
```



6.6 Image Compression

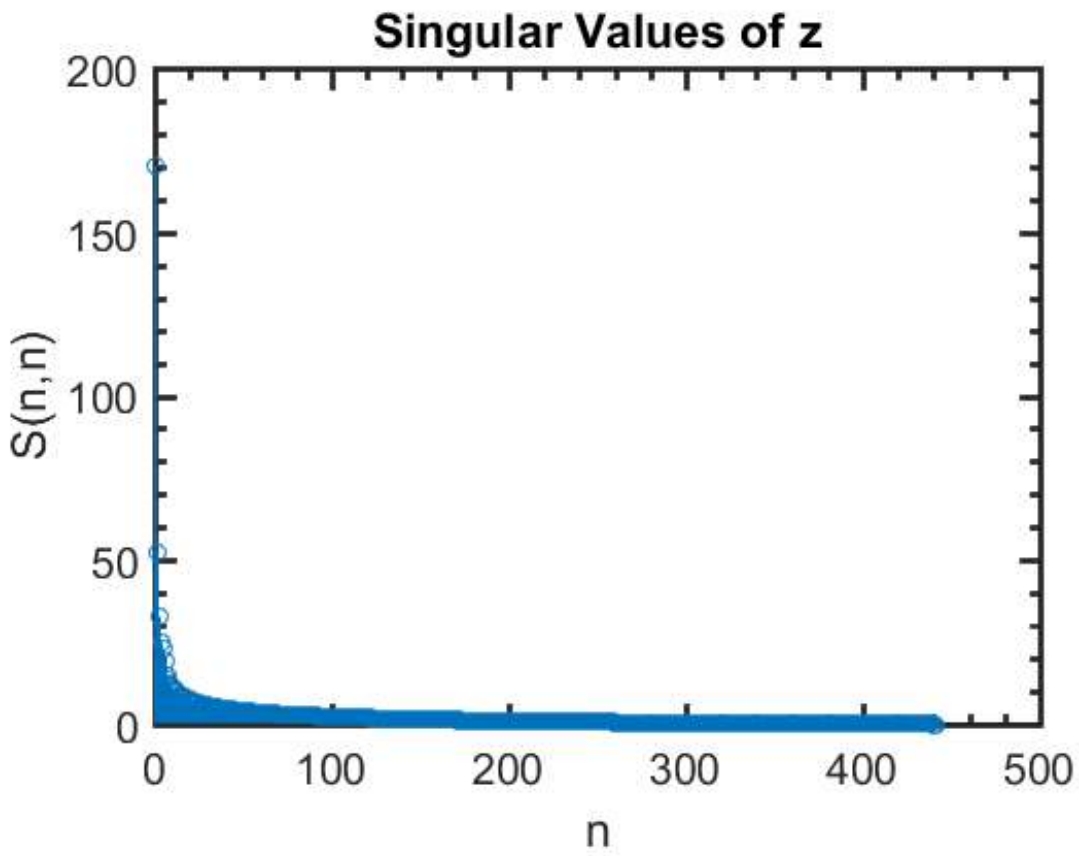
SVD can be used as a form of image compression. The idea is to find redundancies in an image, keep the most significant ones and toss out the less important onces. Similar to the minimal example above, we can toss out the "noise" in an image, which may contribute significantly to it's file size, but have a very small impact on the percieved quality of the image.

```
% Load and convert original image to grayscale
z = imread('myimg.png');
z = rgb2gray(z); % convert rgb image to grayscale
```

```
z = im2double(z); % convert image to double
figure();
imshow(z);
xlabel('Original grayscale image of z');
```

Original grayscale image of z

```
% Decompose into z = U*S*V'
[U,S,V] = svd(z);

% Examine sigular values
s = diag(S);
figure();
stem(s);
xlabel('n');
ylabel('S(n,n)');
title('Singular Values of z');
```



Unlike the minimal example where we knew there to be three signifiant singular values due to the way the matrix was constructed, there is no obvious repeating pattern in most photographs, with the exception of Shayne's head being cropped onto a copy of my body due to his absence during the time of this group picture.

We see that there are many significant singular values, and no obvious cut off point. As we did above, we will choose to make a rank approximation.

```
% Make k'th rank approximation of the input matrix
k = 100;

Uc = U(:,1:k); % Compressed version of U
```

```
Sc = S(1:k,1:k);
Vc = V(:,1:k);
zc = Uc*Sc*Vc'; % don't forget the prime when reconstructing z

figure();
imshow(zc);
xlabel(['k = ' num2str(k) ' rank approximation of z']);
```

k = 100 rank approximation of z

We see a slightly fuzzier version of the input image results. If scaled down, such as for a thumbnail, the reduction in quality may not be significant.

But wait! It appears that we have just made a matrix that is the same size as in the input, i.e. same file size, but now it looks worse! How does this actually achieve compression? While it is true that the reconstructed matrix zc has the same size as the input (441x618 = 272,538 numbers), the truncated decomposition matrices are much smaller.

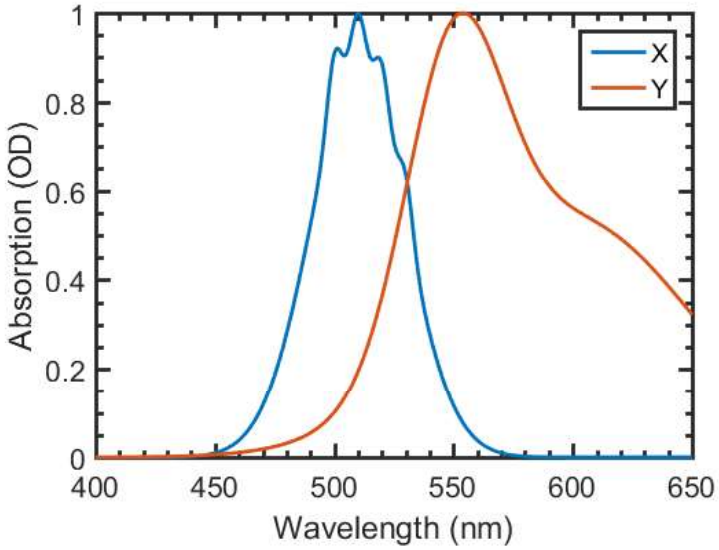
After truncating at k = 100, Uc contains 411x100 = 41,100 numbers, Vc contains 618x100 = 61,800, and while S has the dimensions of the input image, it is diagonal, so we really only need to know the 100 singular values that we kept. The sum off all these matrix sizes is 103,000 numbers, which is approximately half of the data of the orignal input. A user could store Uc, Vc, and diag(Sc) to save on disk space, or transmit these matrices to save on bandwidth. When the image is viewed, the matrix zc is computed from these stored/transmitted values.

6.7 Simulated Spectroscopic Data

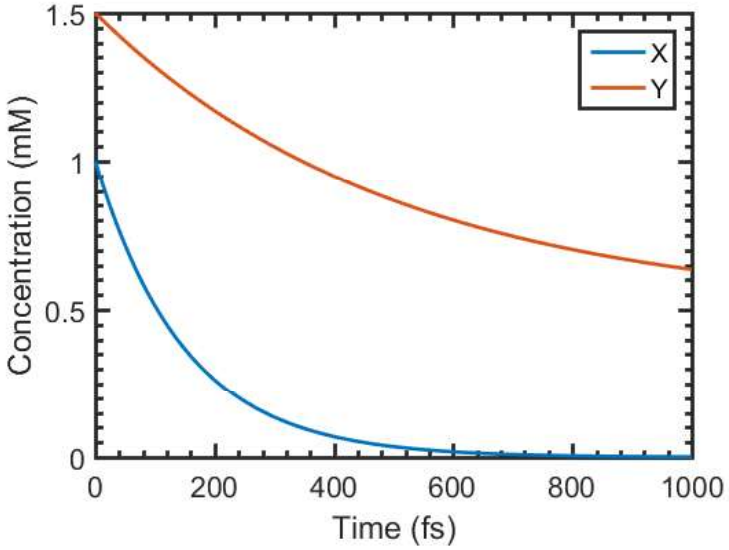
Now I will create a larger, more realistic looking, but still easy to interpret, data set, which resembles a time-resolved spectroscopy experiment.

Consider two chemical species, X and Y. Each has a representative absorption spectrum and concentration profile as a function of time.

```
% Define wavelength axis and absorption spectra of X and Y
lambda = 400:650;
xa = 1.3*normpdf(lambda,500,3)+1.2*normpdf(lambda,510,3)...
    +1.1*normpdf(lambda,520,3)+normpdf(lambda,530,3)+50*normpdf(lambda,510,20);
xa = xa./max(xa);

ya = normpdf(lambda,550,20)+2*normpdf(lambda,600,50);
ya = ya./max(ya);

figure(); plot(lambda,xa,lambda,ya);
xlabel('Wavelength (nm)');
ylabel('Absorption (OD)');
legend(['X' ; 'Y'])
```




```
% Define concentration profiles for X and Y
t = 0:1000;

% Single exponential decays with 150 and 500 fs time constants.
xc = exp(-t/150);
yc = exp(-t/500)+0.5;

figure(); plot(t,xc,t,yc);
xlabel('Time (fs)');
ylabel('Concentration (mM)');
legend(['X' ; 'Y'])
```



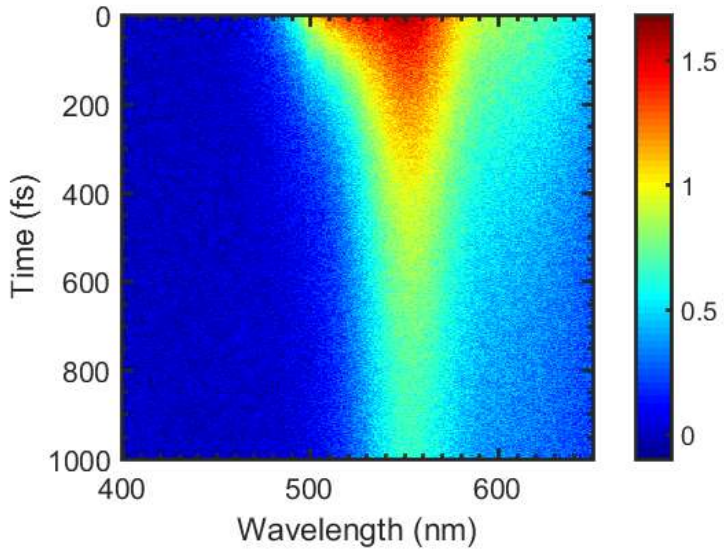
This data is meant to simulate a kinetics experiment. That is, time t=0 is defined by some impulsive event, such as a laser pulse being absorbed. After this event the concentrations of the species X and Y changes. We can use their absortion spectra to measure their concentration as a function of time. A simulated transient absorption data set can be constructed as follows.

```
% The "absorption spectra" matrix
A = [xa' ya'];
% and the "concentration" matrix
C = [xc' yc'];

% the data matrix then can be constructed
D = C*A';
rank(D)

ans =

     2
```

```
% Make some noise!!!
D = D + (rand(size(D))-0.5).*0.2;
rank(D)

ans =

   251
```

We see that the rank of the simulated data is 2 and, as before, the addition of noise makes the matrix full rank.

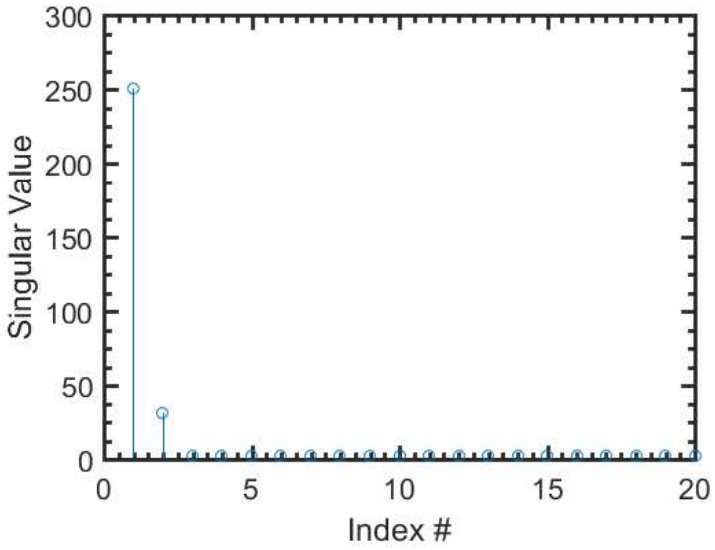
```
figure(); imagesc(lambda,t,D); colorbar;
xlabel('Wavelength (nm)');
ylabel('Time (fs)');
```



Now the task is, given the noisy data matrix, extract the original spectra and/or concentration profiles. In many cases, especially with transient absorption, you may not know *a priori* how many

species are present in the mixture. SVD should first be used to approximate the rank of the data matrix.

```
[U,S,V] = svd(D);
stem(diag(S))
ylabel('Singular Value')
xlabel('Index #')
xlim([0 20]);
```

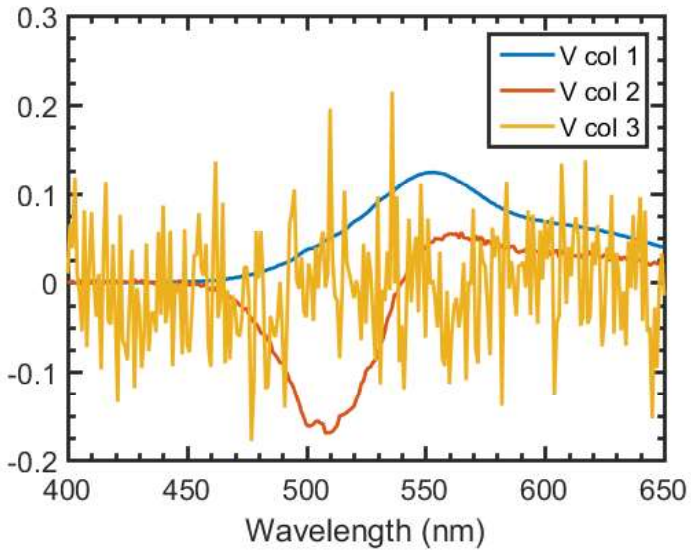


I have truncated the stem plot for the purpose of clarity, but it extends out to the full dimension of the matrix. We see that only two of the singular values are significant by visual inspection. The other's are non-zero, but small, and due to noise. Their values are the so-called **noise floor**.

Now let's inspect the first three columns of V, which contains spectral (row) information

```
vc1 = V(:,1);
vc2 = V(:,2);
vc3 = V(:,3);
figure();
```

```
plot(lambda,vc1,lambda,vc2,lambda,vc3);
legend(['V col 1' ; 'V col 2' ; 'V col 3']); xlabel('Wavelength (nm)');
```



The first thing to notice is that the third column of V looks like noise. This means that it is not a significant contribution to the original dataset. Accordingly, the third sigular value is very small.

Next, we see that columns 1 and 2 have some resemblance to spectra, e.g. they have gaussian shapes. However, they are distinctly **not** the original absorption spectra that we defined above. In fact, the "spectra" in the columns of V have negative values in them, which is non-physical for an absorption spectrum. Note: this is only non-physical in this example, typically transient absorption *difference* spectra often have negative features in them due to stimulated emission and ground state bleach.

6.8 Orthogonal Basis Spectra Versus Species Associated Spectra

The "spectra" that the SVD algorithm returns are not associated with any species. They are, however, orthonormal vectors, i.e. their inner product with themselves is 1, and the inne rproduct between different vectors is 0.

```
vc1'*vc1

ans =

    1.0000


vc2'*vc2

ans =

    1.0000


vc1'*vc2

ans =

   6.4510e-17
```

I will refer to these vectors as **basis spectra**. In contrast, the chemically meaningful spectra, which we have defined in this example, are often referred to as **species associated spectra (SAS)**.

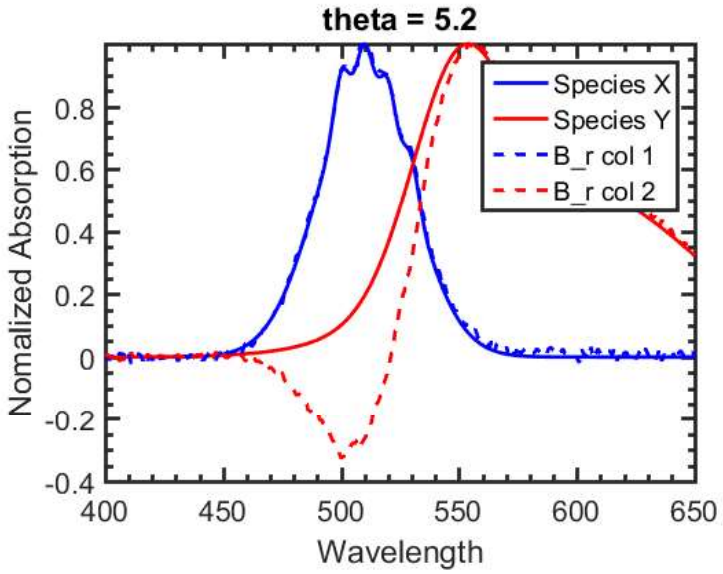
The task for the experimenter analysing the data is to transform the basis spectra into species associated spectra. The species associated spectra can be reconstructed by a **linear combination of basis spectra**.

In the case where there are two species present this transformation can be done by multiplying the basis spectra by a rotation matrix and comparing the result to the original species spectra. If the original species spectra are already known, this is simple. However, as is the case with transient absorption spectroscopy, you may not know what species are present. You may have to make assumptions or place contraints on the rotated spectra, e.g. rotate until a certain region is non-negative, or rotate until it can be fit with a single gaussian. These assumptions will be system dependent and need to be considered carefully.

6.9 Rotation of Basis Spectra

```
%  Group the basis spectra
B = [vc1 vc2];

% Define an angle and 2D rotation matrix
theta = 5.2;



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

% Rotate basis spectra
B_r = B*R;

% Normalize rotated specta for comparison to species spectra
br1 = B_r(:,1); br1 = br1./max(abs(br1));
```

```
br2 = B_r(:,2); br2 = br2./max(abs(br2));


% Plot and compare to species spectra
figure();
plot(lambda,xa,'b',lambda,ya,'r',lambda,br1,'b--',lambda,br2,'r--');
xlabel('Wavelength');
ylabel('Nomalized Absorption');
legend(['Species X ' ; 'Species Y ' ; 'B\_r col 1'; 'B\_r col 2']);
title(['theta = ' num2str(theta)]);
```



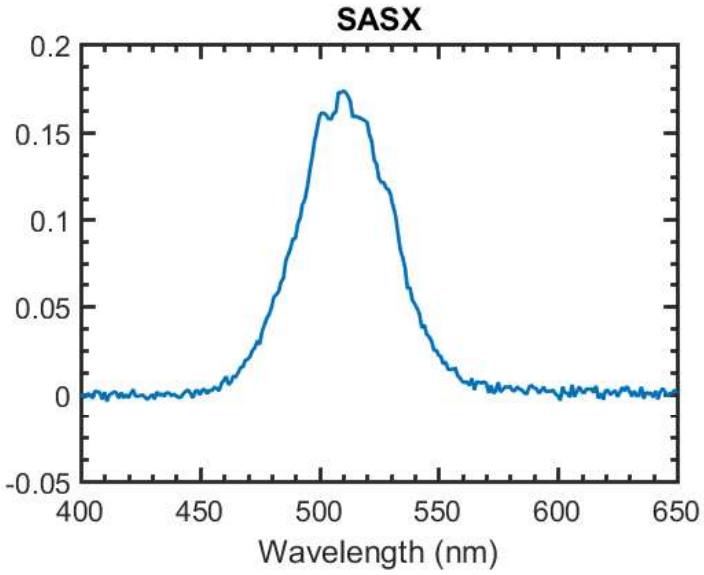
This angle of 5.2 was found by creating a loop around the above block of code and changing the angle slowly while inspecting the graphs.

```
% We can convert this rotation to a complex number via
Z = 1*exp(i*theta)


Z =
```

```
0.4685 - 0.8835i
```

There is no scaling after rotation so the magnitude is 1. Now we can see that this rotation is equivalent to linear combination of basis spectra as follows

```
% Species associated spectrum for X, as a linear combination of basis spectra
SASX = real(Z).*vc1 + imag(Z).*vc2;

figure(); plot(lambda,SASX);
xlabel('Wavelength (nm)'); title('SASX');
```



6.10 Rotation In 3D

In principle, a similar process can be done when there are three contributing species, and thus the SAS will be a linear combination three basis vectors. You will have to use three, 3D rotation matrices however and iterate through all of their angles. I believe that **quaternions**, the extension

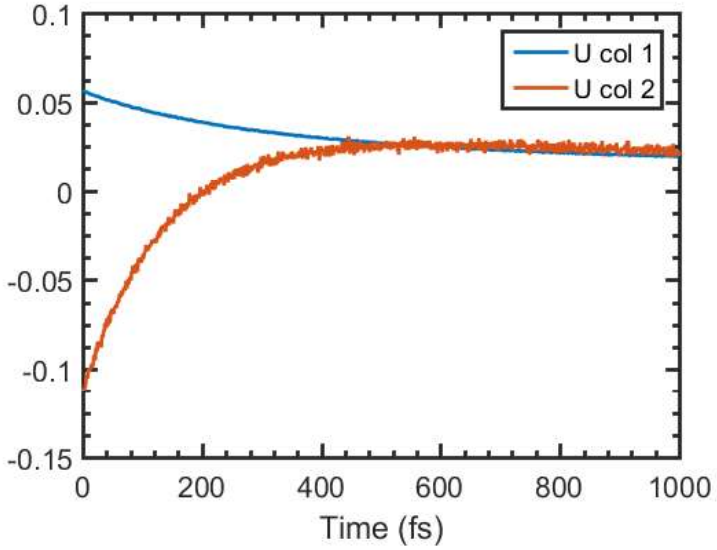
of complex numbers to a 3rd dimension, are a more convenient way to make these 3D rotation, but I have not thought about how to actually do it in great enough detail to provide a tutorial on it.

6.11 Extraction of Dynamics from SVD

The same discussion above applied to the column vectors contained in the U matrix. In this example, this matrix represents the time dependence of the concentration of the species. However, as is the case for spectra, U contains orthonormal basis concentration profiles, rather than the species associated profiles that have chemical significants.

First, let's examine the first two columns of U.

```
uc1 = U(:,1);
uc2 = U(:,2);

figure(); plot(t,uc1,t,uc2);
xlabel('Time (fs)');
legend(['U col 1' ; 'U col 2']);
```



Similar to the basis spectra, they look like exponential decays, but are non-physical if they are meant to represent population due to the negative values.

Since we know that the **species associated decays (SAD)** can be expressed as a linear combination of the basis decays, then each basis decay is also a linear combination of SADs. In transient absorption spectroscopy the usual result is first order exponential kinetics. This means that we can simply fit one of the basis spectra with a sum of exponential functions to extract time constants. Fitting can be done with the MATLAB curve fitting application.

```
General model:
f(x) = a1*exp(-x/tau1)+a2*exp(-x/tau2)+c
Coefficients (with 95% confidence bounds):
       a1 =     0.00996  (0.007225, 0.01269)
       a2 =     0.03076  (0.0285, 0.03301)
       c =     0.01537  (0.01482, 0.01593)
       tau1 =         152  (129.4, 174.5)
       tau2 =       495.6  (448.2, 543)
```

```
Goodness of fit:
  SSE: 5.475e-05
  R-square: 0.9994
  Adjusted R-square: 0.9994
  RMSE: 0.0002344
```

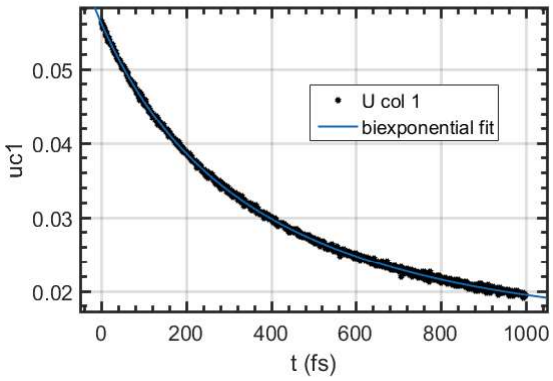


Figure 6.2: The first column of U contains information about the dynamics of the data set. It is an orthonormal basis vector of U, which by itself is not chemically meaningful. However, it can also be expressed as a linear combination of the decay profiles we defined earlier. This is demonstrated by fitting the orthonormal vector with a sum of exponential functions.

We see that the 150 and 500 fs time constants are recovered.

6.12 Generalization to Other Systems

The example given above was tailored for time domain spectroscopy. However, please keep in mind that SVD as an algorithm is completely general. In my research I have used it to work up pH titration for the extraction of pKas. It is analogous to the above example, except absorption spectra

are taken of a solution a many pH values, rather than time delays. Instead of exponential decays in the U matrix, a sigmoidal shaped curve appears, which can be fit to the Henderson-Hasselbalch equation to calculate pKa. If there were multiple protonation states (either on a single molecule like in universal indicator or if there was a mixture of molecules) then, as in the example above, a sum of Henderson-Hasselbalch equations should be used to fit the data. Another potential application in identifying the reduction potentials of species in an electrochemical reaction. Spectra may be acquired as a function of voltage in this case. More generally, it can be used to find redundancies in any data set, even when x and y slices of the data are a bit abstract like in a photograph.

6.13 Potential Pit Falls

While SVD is very broadly applicable to 2D data sets, there are some instances of condensed phase molecular physics which may produce wonky results.

SVD is well suited when there are long-lived (relative to whatever timescale you are working with), well defined chemical species which exchange quickly with each other, e.g. acid base reactions in the ground and excited states. However, continuous processes such as solvent shell relaxation after excitation or wavepacket motion along a potential energy surface that may modulate a fluorescence wavelength will produce strange looking, but mathematically correct, results from SVD.

Here I've constructed a 2D data set by specifying a gaussian spectrum centered at 400 nm. As a function of time, the center frequency shifts to 500 nm in 1000 fs. This could theoretically be a simple model of how the stimulated emission wavelength changes as there is nuclear relaxation in the excited state.

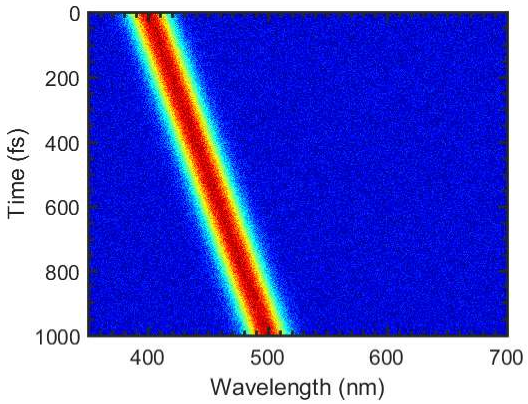


Figure 6.3: A Gaussian function centered at 400 nm that shifts linearly to 500 nm in 1000 fs. Noise is added at 10% of the maximum value. This is a simple simulation of how excited state relaxation may manifest in transient spectroscopy.

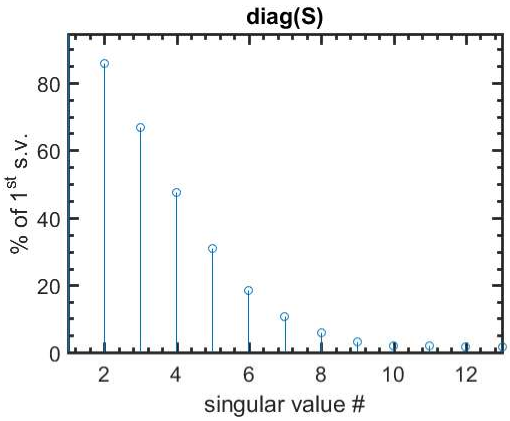


Figure 6.4: The singular values obtained from SVD of the above data, expressed a percent of the first singular value.

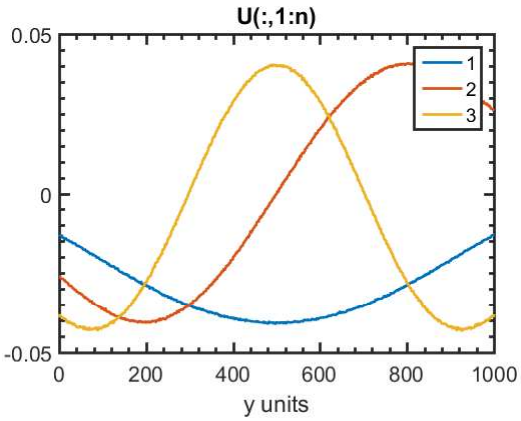


Figure 6.5: The first three columns of the U matrix obtained from SVD

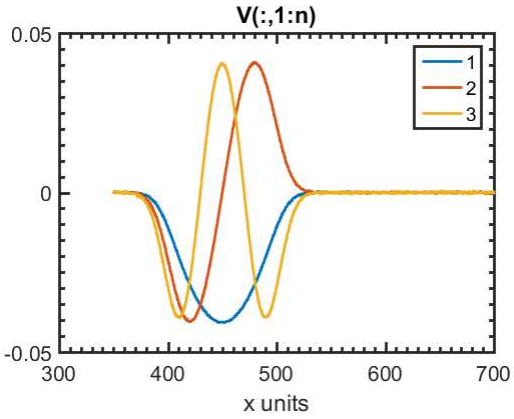


Figure 6.6: The first three columns of the V matrix obtained from SVD

Applying the SVD algorithm to this data set returns nine singular values that are above the 2% limit. Additionally, examination of the spectra and decays shows their odd behavior. These features can be explained in the following way.

When there is a continuous relaxation, many basis vectors are required to model the behavior. SVD is poorly poised to address such a physical system. A more general fitting technique, such as global fitting will be more appropriate

References

[1] Schrager, R. I.; Hendler, R. W. *Analytical Chemistry* **1982**, *54*, 1147–1152.

[2] Henry, E.; Hofrichter, J. *Methods in Enzymology* **1992**, *210*, 129–192.