MAT: 150 -- Lab 7

Date: 12/2/2017

Singular Value Decomposition

The Singular Value Decomposition (SVD) is one of the most important factorizations in all of linear algebra. For any mxn matrix A there exists orthonormal U (mxm) and V (nxn) and diagonal matrix S (mxn) such that $A = U\ S\ V^T$. The diagonal entries of S are the singular values of A: [σ_1, ... , σ_r, 0, ... 0], where there are (min(m,n) - r) trailing zeros and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. The column vectors of V are the right singular vectors of A, and the column vectors of U are the left singular vectors of A.

Maximizing Magnification under A

It is possible to define the singular values of A via the maximization of magnification under A. To this end, view A as a linear transformation that maps vectors in R^n to vectors in R^m. We are interested in maximizing ||Ax|| subject to $x^T x = 1$ (i.e. ||x||=1). In class we noted that this maximum is denoted by ||A|| (the norm of the matrix A) and is equal to σ_1 (the largest singular value).

We can also view the maximization of ||Ax|| through the lens of maximizing || Ax (|| ^2) = < Ax, Ax > = $x^T \left(A^T A\right) x$, which is the quadratic induced by the symmetric matrix $A^T A$. We know that the maximum of this quadratic form subject to the condition $x^T x = 1$ is the maximum eigenvalue of $A^T A$ and occurs at the corresponding eigenvector.

It follows that $\sigma_1 = (\lambda_1)^{1/2}$ and the corresponding right singular vector of A, v_1, is equal to the corresponding eigenvector of $A^T A$. Similarly, the second largest singular value σ_2 is the maximum of ||Ax|| subject to $x^T x = 1$ and $x^T v_1 = 0$. Following Theorem 7 of Section 7.3 of Lay, it follows that $\sigma_2 = (\lambda_2)^{1/2}$ and the corresponding right singular vector of A, v_2, is equal to the corresponding eigenvector of $A^T A$. In this fashion, using Theorem 8 of Section 7.3 of Lay, we know that the singular values of A are the square root of the corresponding eigenvalues of $A^T A$, and the corresponding right singular vectors of A are equal to the corresponding eigenvector of $A^T A$.

Example

We will compute the SVD of the following matrix.

In[1]:=
```
A={{1,2,0},{2,0,2}};
Print["A = ", MatrixForm[A]];
```

A = $\begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 2 \end{pmatrix}$

First, we form the matrix $A^T A$ and denote it B.

In[3]:=
```
B=Transpose[A].A;
Print["B = ", MatrixForm[B]];
```

B = $\begin{pmatrix} 5 & 2 & 4 \\ 2 & 4 & 0 \\ 4 & 0 & 4 \end{pmatrix}$

Next, we compute the eigenvalues and eigenvectors of B using the built in Eigensystem function. The eigenvalues are stored in eigval and the eigenvectors are stored in eigvec.

In[5]:=
```
{eigval,eigvec}=Eigensystem[B];
Print["Eignevalues: ",MatrixForm[eigval], ", Eigenvectors: ",MatrixForm[eigvec]];
```

Eignevalues: $\begin{pmatrix} 9 \\ 4 \\ 0 \end{pmatrix}$, Eigenvectors: $\begin{pmatrix} 5 & 2 & 4 \\ 0 & -2 & 1 \\ -2 & 1 & 2 \end{pmatrix}$

Note that there are 3 eigenvalues, but we only have 2 singular values since the dimension of A is 2x3. The eigenvectors are in row-form, they are clearly orthogonal, which we would expect since B is symmetric, but they are not normalized. We have to take all of this into account when forming the SVD. Below we store the matrix S and the matrix V.

In[7]:=
```
(* Singular Values *)
S=ConstantArray[0,{2,3}];
S[[1,1]]=Sqrt[eigval[[1]]];
S[[2,2]]=Sqrt[eigval[[2]]];
(* Right Singular Vectors *)
V=ConstantArray[0,{3,3}];
V[[All,1]]=eigvec[[1]]/Norm[eigvec[[1]]];
V[[All,2]]=eigvec[[2]]/Norm[eigvec[[2]]];
V[[All,3]]=eigvec[[3]]/Norm[eigvec[[3]]];
(* Print Results *)
Print["S =", MatrixForm[S],", V = ", MatrixForm[V]];
```

S = $\begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix}$, V = $\begin{pmatrix} \frac{\sqrt{5}}{3} & 0 & -\frac{2}{3} \\ \frac{2}{3\sqrt{5}} & -\frac{2}{\sqrt{5}} & \frac{1}{3} \\ \frac{4}{3\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{2}{3} \end{pmatrix}$

In general, we can compute the left singular vectors by computing the eigenvectors of AA^T. In this case, since both singular values are nonzero, we can use the geometric interpretation of the SVD discussed in class. To this end, $Av_1 = \sigma_1 u_1$ and $Av_2 = \sigma_2 u_2$. Below we store the matrix U.

In[15]:=
```
(* Left Singular Vectors *)
U=ConstantArray[0,{2,2}];
U[[All,1]]=A.V[[All,1]]/S[[1,1]];
U[[All,2]]=A.V[[All,2]]/S[[2,2]];
(* Simplify Result (cleans up radical arithmatic) *)
U=Simplify[U];
(* Print Results *)
Print["U =", MatrixForm[U]];
(* Check SVD *)
Print["USV^T = ", MatrixForm[U.S.Transpose[V]]];
```

$$U = \begin{pmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}$$

$$USV^T = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 2 \end{pmatrix}$$

We can see that our work is correct from the above result. Fortunately, there is a built in Mathematica function that will perform the same decomposition. Below is the code for calling this function.

In[21]:=
```
{u,s,v}=SingularValueDecomposition[A];
Print["u = ", MatrixForm[u],", s = ", MatrixForm[s],", v = ", MatrixForm[v]];
```

$$u = \begin{pmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}, \ s = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix}, \ v = \begin{pmatrix} \frac{\sqrt{5}}{3} & 0 & -\frac{2}{3} \\ \frac{2}{3\sqrt{5}} & -\frac{2}{\sqrt{5}} & \frac{1}{3} \\ \frac{4}{3\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{2}{3} \end{pmatrix}$$

Applications of the SVD

As noted in the previous section, the SVD is one of the most important factorizations in all of linear algebra. Much of this has to do with the large array of applications of the SVD. In this section we will review some of the major applications associated with the SVD.

Condition Number

Let A be an nxn matrix. The condition number of A is a measurement of how accurate solutions to the matrix equation Ax=b can be in floating point arithmetic. In essence it measures how close the columns of A are to being linearly dependent. The condition number of A is denoted by k(A) and is defined by the ratio k(A) = σ_1 / σ_n. The best condition number is 1 and the worst is ∞. Below is a function for computing the condition number of a nxn matrix A.

In[23]:=
```
Cond[A_]:=
Module[{n,u,s,v},
n=Dimensions[A][[1]];
{u,s,v}=SingularValueDecomposition[N[A]];
s[[1,1]]/s[[n,n]]];
```

Below is a table of comparisons of the condition number of some famous matrices that are readily available in Mathematica and a random Orthonormal matrix. Note that the condition number of the orthonormal matrix is 1. This is always the case and is easy to prove, I recommend doing so as a review.

In[24]:=
```
A1=HilbertMatrix[10];
A2=ToeplitzMatrix[10];
A3=HankelMatrix[10];
A4=RandomVariate[NormalDistribution[0,1],{10,10}];
A4=QRDecomposition[A4][[1]];
T={{Cond[A1]},{Cond[A2]},{Cond[A3]},{Cond[A4]}};
Print[TableForm[T,TableHeadings→{{"Hilbert","Toeplitz","Hankel","Orthonormal"},{"Conditio
```

	Condition Number
Hilbert	1.60249×10^{13}
Toeplitz	85.9953
Hankel	8.18017
Orthonormal	1.

Bases for Fundamental Subspaces

Think of A as a linear transformation mapping from R^n to R^m and let $A = USV^T$ be the SVD. Then, the columns of V and U form an orthonormal basis for R^n and R^m, respectively. The action of A with respect to these basis is incredibly simple, see the geometric interpretation from your notes or Figure 4 of Section 7.4 of Lay. Furthermore, these basis provide a basis for the 4 fundamental subspaces associated with A. Specifically

$$\text{RowA} = \text{span}\{v_1, v_2, \ldots, v_r\}$$
$$\text{ColA} = \text{span}\{u_1, u_2, \ldots, u_r\}$$
$$\text{NulA} = \text{span}\{v_{r+1}, v_{r+2}, \ldots, v_n\}$$
$$\text{NulA}^T = \text{span}\{u_{r+1}, u_{r+2}, \ldots, u_m\}$$

From these fundamental spaces we gain a lot of information about a matrix and its potential inverse. For Instance, the nxn matrix A is invertible if and only if NulA = {0} which is true if and only if r=n which is equivalent to A not having any zero singular values. See the Invertible Matrix Theorem in Section 7.4 of Lay.

Pseudo-Inverse

There are plenty of important matrix equations Ax=b where A does not have an inverse, or the system does not have a solution; for instance, when solving for the line of best fit. The SVD provides a way to consistently solve for the best approximation using what is known as the pseudo-inverse.

Recall that the SVD of A can be written in condensed form A = $\hat{U}\,\hat{S}\,\hat{V}^T$, where $\hat{U}$ is an mxr orthonormal matrix, $\hat{S}$ is an rxr diagonal matrix (diagonal entries are the nonzero singular values), and $\hat{V}$ is an nxr orthonormal matrix. We define the pseudo-inverse of A by $A^\dagger = \hat{V}\,\hat{S}^{-1}\,\hat{U}^T$. As a review, you should show that $A^\dagger A = I_n$, the nxn identity, and . It follows that to solve Ax=b one can just multiply both sides on the left by $A^\dagger$, to get $x = A^\dagger b$ as a solution. Consider the code below.

In[31]:=
```
(* Matrix A and vector b *)
A={{1,1},{2,1},{3,1}};
b={{2},{3.1},{3.95}};
(* SVD of A *)
{u,s,v}=SingularValueDecomposition[A];
(* pseudo-inverse of A *)
A†=v[[All,1;;2]].Inverse[s[[1;;2,1;;2]]].Transpose[u[[All,1;;2]]];
(* best approximation *)
x=A†.b;
(* print results *)
Print["Best Approximation: ", x];
```

```
Best Approximation: {{0.975}, {1.06667}}
```

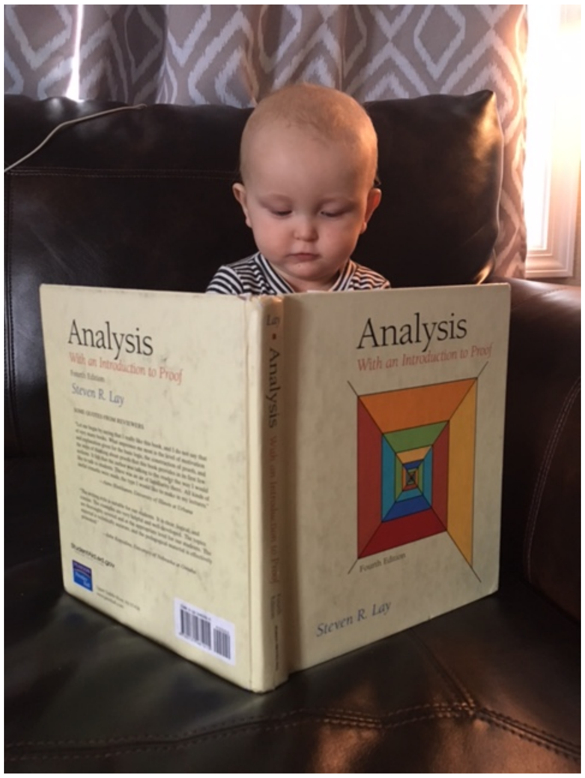
Image Compression

The condensed SVD allows us to write the matrix A as the sum of rank-1 matrices. Specifically, $A = \sum_{i=1}^{r} \sigma_i\, u_i\, v_i^T$. This has huge applications in image compression. Recall that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$; in addition, it is often that case that $\sigma_1 >> \sigma_r$ (σ_1 is much larger than σ_r), and it follows that the most important components are associated with the first relatively few singular values. Therefore, we can store a good approximation to A using the first few (principal) components. This is the essence of Principle Component Analysis (PCA), here we will use it to compress an image. The code below loads an image, stores it as a matrix A, displays the dimensions of A, and displays the image.

In[37]:=
```
A=ImageData[Import["/Users/thcameron/Documents/Wolfram\ Mathematica/hud.JPG"]];
Print["Dimensions: ", Dimensions[A]];
Print["Original Image: ", Image[A,ColorSpace→"RGB",ImageSize→300]];
```

```
Dimensions: {640, 480, 3}
```

```
Original Image:
```



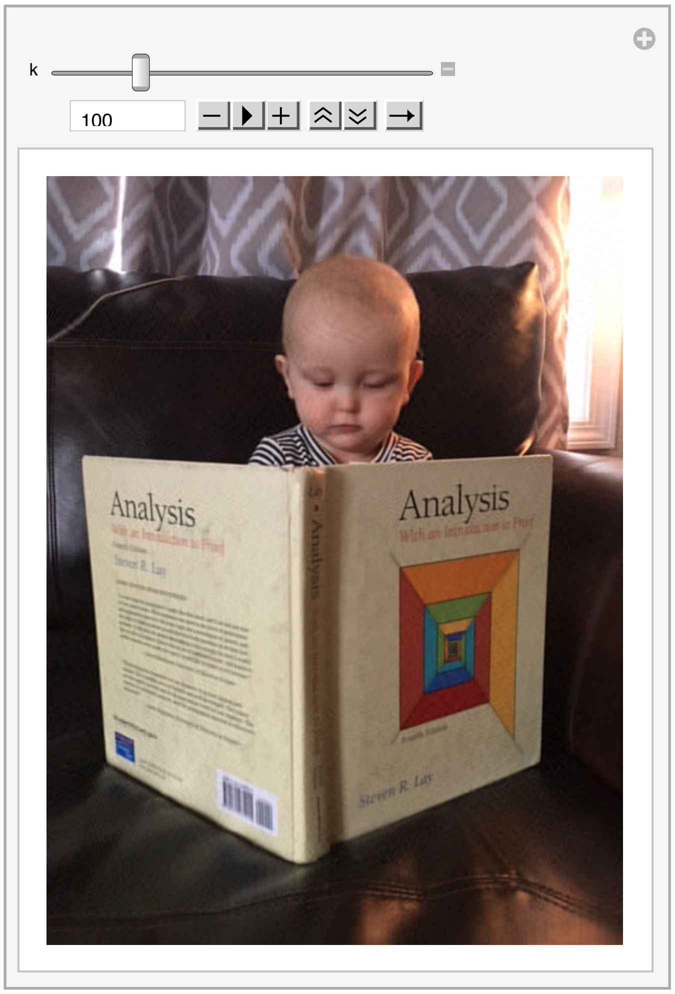
The function below will perform an SVD on each RGB matrix and return the compressed image using only k singular values.

```
In[40]:= CImage[A_,k_]:=
Module[{M1,M2,M3,u,s,v},
{u,s,v}=SingularValueDecomposition[A[[All,All,1]],k];
M1=u.s.Transpose[v];
{u,s,v}=SingularValueDecomposition[A[[All,All,2]],k];
M2=u.s.Transpose[v];
{u,s,v}=SingularValueDecomposition[A[[All,All,3]],k];
M3=u.s.Transpose[v];
u=ConstantArray[0,{640,480,3}];
u[[All,All,1]]=M1;
u[[All,All,2]]=M2;
u[[All,All,3]]=M3;
u];
```

Now, we use the Manipulate function to see how our image quality changes given more principal components.

```
In[41]:= Print["Compressed Image: ", Manipulate[Image[CImage[A,k],ColorSpace→"RGB",ImageSize→300],
```

```
Compressed Image:
```



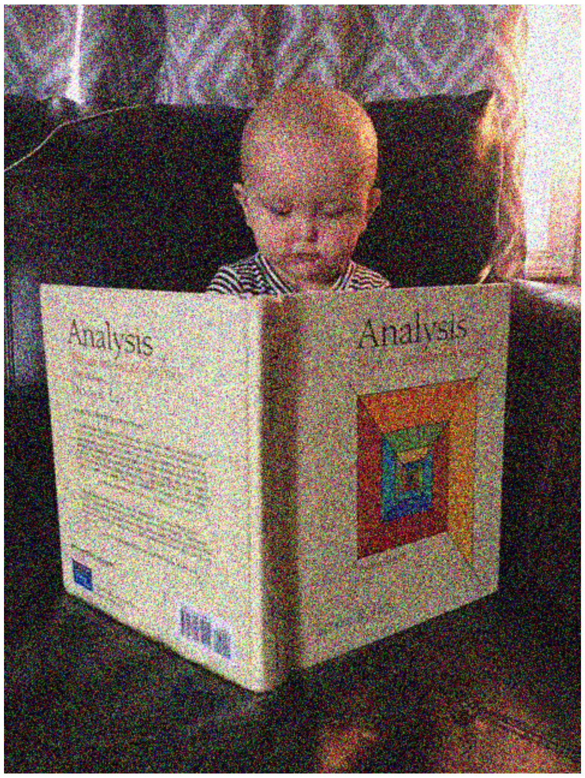
By the time we get to 100 principal components the image looks pretty good. What’s more important, we can store this matrix using 100 singular values, right-singular vectors, and left-singular vectors. Thus, the storage requirements is 3*(100+100*(640+480))=336300, where as the original matrix requires 3*640*480=921600. Thus, we are using 36% of the original storage requirements.

Image Noise

Suppose that we received our image in a process that is prone to introducing noise. Image noise is random variation of brightness or color information. This noise can be produced by sensors, scanners, digital cameras, film grain, or shot noise. Here we will add a random amount of small noise to the matrix A introduced in the previous section, we store the new matrix as B.

In[42]:=
```
B=A+RandomVariate[NormalDistribution[0,0.2],Dimensions[A]];
Print["Noisy Image: ", Image[B,ColorSpace→"RGB",ImageSize→300]];
```

Noisy Image:



While the SVD is involved in removing the noise in the above image, there are many other concepts that would take us well outside of the scope of our course. Fortunately, there is a built in Mathematica function TotalVariationFilter that allows us to remove the noise in an image. Consider the code below, where the method used in Gaussian. You have options of Gaussian, Laplacian, and Poisson. Play around with adding noise to your image and removing it with different methods.

In[44]:=

```
Print["Denoised Image: ", Image[TotalVariationFilter[B,Method->"Gaussian"],ImageSize→300]
```

Denoised Image: