Lecture 4: Singular Value Decomposition

Motivation

- Image Compression by Using MATLAB: 
  http://www.owlnet.rice.edu/elec539/Projects99/JAMK/proj1/
- Image Compression by Using SVD(Singular Value Decomposition): 
  http://fourier.eng.hmc.edu/e161/lectures/svdcompression.html
- Key word: SVD(Singular Value Decomposition), PCA(Principal Component Analysis)

Review of Matrix Transformation

Let’s start by reviewing the matrix transformations.

- Ex. 1
  Let \( A_1 = \begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix}, u = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \text{and } v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}. \) We get

  \[ A_1u = \begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -5 \\ -1 \end{bmatrix} \text{ and } A_1v = \begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 1 \end{bmatrix}. \]

  For \( u \), matrix \( A_1 \) yields the usual linear transformation. But, for \( v \), since the resulting vector is a product of the original vector, we can write \( A_1v = 2v \). This is very interesting, as this implies the matrix \( A_1 \) stretches vectors of the form \( v \).

  This can be shown in other cases. For example, try \( A_1w \) for \( w = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \) and see.

- Ex. 2 Let \( A_2 = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}, u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \text{and } v = \begin{bmatrix} 3 \\ -2 \end{bmatrix}. \) Applying \( A_2 \) to our vectors, we get

  \[ A_2u = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \end{bmatrix} = 7 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } A_2v = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ -2 \end{bmatrix} = \begin{bmatrix} -9 \\ 11 \end{bmatrix}. \]

  Now \( A_2v \) is just a general transformation of \( v \), but again \( A_2 \) stretches the vector \( u \) out by seven times.

It seems like for a matrix \( A \), there is a vector \( u \) such that \( Au \) is the certain times of \( u \), i.e. \( Au = \lambda u \) for some scalar \( \lambda \). We can collect these kinds of vectors and give them names.
Eigenvalues and Eigenvectors

- Definition

An **eigenvector** of an \( n \times n \) matrix \( A \) is a nonzero vector \( x \) such that \( Ax = \lambda x \) for some scalar \( \lambda \). A scalar \( \lambda \) is called an **eigenvalue** of \( A \) if there is a nontrivial solution \( x \) of \( Ax = \lambda x \); such an \( x \) is called an eigenvector corresponding to \( \lambda \).

- Ex. 3

So in the first example, recall we have \( A_1 v = 2v \), as shown above. Thus, \( v \) is an eigenvector of \( A \) with a corresponding eigenvalue \( \lambda = 2 \). If you also try \( A_1 w \), you will find out \( A_1 w = w \) so \( w \) is an eigenvector of \( A \) with a corresponding eigenvalues of 1.

Obviously, only certain vectors can be the eigenvectors of a matrix. How about eigenvalues?

- Ex. 4

We can show that \(-4\) is an eigenvalue of matrix \( A_2 \) in Ex. 2 and find the corresponding eigenvector. Based on the definition, the scalar \(-4\) is an eigenvalue of matrix \( A_2 \) if and only if we can find a vector \( x \) such that

\[
A_2 x = -4 x
\]

which is equivalent to \( A_2 x + 4x = 0 \) or

\[
(A_2 + 4I)x = 0.
\]

where \( I \) is the identity matrix. To solve this equation, form the matrix

\[
A_2 + 4I = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix} + \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 6 \\ 5 & 6 \end{bmatrix}.
\]

We can let \( x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \) then we are solving the equation

\[
\begin{bmatrix} 5 & 6 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.
\]

Since the first and second row of \( \begin{bmatrix} 5 & 6 \\ 5 & 6 \end{bmatrix} \) are the same, we are just solving \( 5x_1 + 6x_2 = 0 \).

Then we get \( x_1 = 6, x_2 = -5 \) and \( x = \begin{bmatrix} 6 \\ -5 \end{bmatrix} \).

We are lucky to guess \(-4\) as the other eigenvalue of \( A_2 \). How could this be solved in general? (See notes on Linear Algebra 2133).
Ex. 5

Could other numbers be an eigenvalue of $A_2$? Let’s try 3 and check if it is an eigenvector of $A_2$. We need to find a vector $x$ such that

$$(A_2 - 3I)x = 0.$$ 

We have

$$(A_2 - 3I)x = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix} - \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -2 & 6 \\ 5 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$ 

But, besides $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ there is no nontrivial $x$ such that $(A_2 - 3I)x = 0$, so 3 is not an eigenvalue of $A_2$ and has no associated eigenvector.

Eigenvalues and Eigenvectors in MATLAB

- The function `eig()`

MATLAB is particularly useful for computing eigenvalues of matrices. As an example, let’s set matrix $A_2$ into MATLAB, which is defined above, and try the command out:

```matlab
>> A2 = [ 1 , 6 ; 5 , 2 ] ;
>> eig (A2)
```

```plaintext
ans =
-4
 7
```

As you can see, it’s printed out the two eigenvalues we found in Ex. 2 and Ex. 4. In fact, these are ALL of the eigenvalues of the matrix $A_2$. Further, we can actually use this same function to compute the eigenvectors, which are normalized (discussed below), in the same way. However, we need to give MATLAB two variables to store these in. So, again try

```matlab
>> A2 = [ 1 , 6 ; 5 , 2 ] ;
>> [U,E] = eig (A2)
```

```
U =
-0.7682   -0.7071
 0.6402   -0.7071

E =
-4   0
 0   7
```
We see now a couple things. First, $E$ is now a matrix of the eigenvalues, not a vector. We also have the eigenvectors that correspond to each eigenvalue, given by each column of $U$. Notice that they aren’t integers like we had before. That’s because eigenvectors aren’t unique. They can be rescaled.

Small proof: Consider an eigenvector $u$ and a rescaling parameter $a$. Then,

$$Au = \lambda u \implies aAu = a\lambda u \implies A(au) = \lambda(au)$$

So, $au$ is also an eigenvector. MATLAB will scale these eigenvectors to their normalized versions by default. Normalized eigenvectors, however, are unique.

- **Eigen-decomposition**
  The reason MATLAB puts these into matrices is because it is tied to something called the eigenvalue decomposition. This, without putting in too much detail, is idea that a nonsingular matrix $A$ can be decomposed into its eigenvectors and eigenvalues, such that $A = UEU^{-1}$, where $U$ is the matrix of eigenvectors and $E$ is a matrix of the eigenvalues on the diagonal. For example, try

```matlab
>> U*E*inv(U)
ans =
    1.0000    6.0000
    5.0000    2.0000
```

This should match up with $A_2$. Sometimes this isn’t as accurate as you see here, part of which depends on the matrix. But, that’s something that can be discussed in a later course. For now, you can check using this idea.

**Singular Value Decomposition**

- **Norm and Unit Vector**
  In Ex. 4, we can find the eigenvector of $A_2$ corresponding the eigenvector $-4$ by solving

$$5x_1 + 6x_2 = 0$$

and find out $x = \begin{bmatrix} 6 \\ -5 \end{bmatrix}$. But, solutions such as $x = \begin{bmatrix} 6 \\ -5 \end{bmatrix}$, not $\begin{bmatrix} 6 \\ -1 \end{bmatrix}$, or $\begin{bmatrix} -1 \\ 5 \end{bmatrix}$? also work. For this, sometimes we would like to find an **unit vector** to be the answer. An unit vector means that the length of unit vector is equal to 1.

How to define the length of a vector? In mathemetic, the length of a vector $x = (x_1, x_2, x_3, \cdots, x_n)$ is called **norm** and defined by

$$\|x\| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2}.$$
For example, the vectors $e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ are unit vectors, but $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is not. If the length of a vector $x$ is not 1, we can normalize $x$ by divided the norm $||x||$:

$$\frac{x}{||x||}.$$

For $x = \begin{bmatrix} 6 \\ -5 \end{bmatrix}$ in Ex. 2, we can find the norm of it and normalize it:

$$||x|| = \sqrt{6^2 + (-5)^2} = \sqrt{61} \text{ and } \frac{x}{||x||} = \begin{bmatrix} \frac{6}{\sqrt{61}} \\ \frac{-5}{\sqrt{61}} \end{bmatrix}.$$ Compare this to the first column of $U$. Does this match? Why or why not?

• **Inner Product, Orthogonal and Orthonormal**

We’ve said the multiplication between two objects a lot.

For two vectors $u = (u_1, u_2, u_3, \cdots, u_n), v = (v_1, v_2, v_3, \cdots, v_n)$, we define an inner product between $u$ and $v$

$$u \cdot v = (u_1, u_2, u_3, \cdots, u_n) \cdot (v_1, v_2, v_3, \cdots, v_n) = u_1v_1 + u_2v_2 + \cdots + u_nv_n.$$

If the inner product of two vectors is equal to 0, we say the two vectors are orthogonal. For example, let $u = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. we have

$$u \cdot v = 1 \times 1 + (-1) \times 1 = 0.$$

Then we say $u$ and $v$ are orthogonal.

Furthermore, if two unit vectors are orthogonal, we call this two unit vectors are orthonormal. For example, we have $u = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, ||u|| = \sqrt{2}$ and $||v|| = \sqrt{2}$.

If we normalize $u$ and $v$ to get

$$\frac{u}{||u||} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}, \frac{v}{||v||} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

then $\frac{u}{||u||}$ and $\frac{v}{||v||}$ are orthonormal.

If we have a $2 \times 2$ matrix $A$ whose first and second rows orthonormal, then $A* A^T = I$. 5
For example, let \( A = \begin{bmatrix} 1 & 1 \\ \sqrt{2} & \sqrt{2} \\ -1 & -1 \end{bmatrix} \). then we have

\[
AA^T = \begin{bmatrix} 1 & 1 \\ \sqrt{2} & \sqrt{2} \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ \sqrt{2} & \sqrt{2} \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.
\]

This kind of martices is called **orthogonal matrices**.

- **Singular Value Decomposition (SVD)**

Singular value decomposition (or SVD) is a factorization of a matrix. In fact, is a generalized version of eigenvalue decomposition. Before, for eigenvalue decomposition, we needed to have square matrices. So, a size \( n \times n \) matrix would have at most \( n \) distinct eigenvalues (possibly less if numbers repeated). This is no longer the case.

Given an \( m \times n \) matrix \( A \) with \( m > n \), \( A \) can be factorized by SVD into three matrices:

- \( U \) is an \( m \times n \) orthogonal matrix that satisfies \( U^TU = I_n \),
- \( S \) is a \( n \times n \) diagonal matrix,
- \( V \) is an \( n \times n \) orthogonal matrix satisfying \( VV^T = V^TV = I_n \),

such that \( A = USV^T \). The entries in the diagonal matrix \( S \) are known as the singular values of \( A \). They turn out to be the square roots of the eigenvalues of the square matrix \( A^TA \). So, if \( A \) is a real symmetric matrix with positive eigenvalues, then the singular values and eigenvalues are the same. However, this is not true in general. It is important to realize they are related, but distinct factorizations.

**SVD in MATLAB**

- **svd()**

In MATLAB, we can use command `svd()` to get these three matrices.

Given an \( 4 \times 2 \) matrix \( A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \) and use this command.

```matlab
>> A = [1 2; 3 4; 5 6; 7 8];
>> [U,S,V] = svd(A)
```

We will get
\[ U = \]
\[
\begin{bmatrix}
-0.1525 & -0.8226 & -0.3945 & -0.3800 \\
-0.3499 & -0.4214 & 0.2428 & 0.8007 \\
-0.5474 & -0.0201 & 0.6979 & -0.4614 \\
-0.7448 & 0.3812 & -0.5462 & 0.0407
\end{bmatrix}
\]

\[ S = \]
\[
\begin{bmatrix}
14.2691 & 0 & 0 & 0 \\
0 & 0.6268 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[ V = \]
\[
\begin{bmatrix}
-0.6414 & 0.7672 \\
-0.7672 & -0.6414
\end{bmatrix}
\]

We still can say \( S \) is a diagonal matrix since diagonal elements are the only nonzero elements in \( S \) even though \( S \) is not a square matrix. Then, we can check this decomposition by multiplying \( U \), \( S \), and \( V^T \) together

\[
\gg \ U*S*V';
\]

ans =

\[
\begin{bmatrix}
1.0000 & 2.0000 \\
3.0000 & 4.0000 \\
5.0000 & 6.0000 \\
7.0000 & 8.0000
\end{bmatrix}
\]

We see that we’ve recovered matrix \( A \). It’s also worth trying on a square matrix:

\[
\gg \ A = \text{magic}(5)
\]

\[
\gg \ [U,S,V] = \text{svd}(A)
\]

\[
A =
\begin{bmatrix}
17 & 24 & 1 & 8 & 15 \\
23 & 5 & 7 & 14 & 16 \\
4 & 6 & 13 & 20 & 22 \\
10 & 12 & 19 & 21 & 3 \\
11 & 18 & 25 & 2 & 9
\end{bmatrix}
\]
U =

\[
\begin{bmatrix}
-0.4472 & -0.5456 & 0.5117 & 0.1954 & -0.4498 \\
-0.4472 & -0.4498 & -0.1954 & -0.5117 & 0.5456 \\
-0.4472 & -0.0000 & -0.6325 & 0.6325 & -0.0000 \\
-0.4472 & 0.4498 & -0.1954 & -0.5117 & -0.5456 \\
-0.4472 & 0.5456 & 0.5117 & 0.1954 & 0.4498 \\
\end{bmatrix}
\]

S =

\[
\begin{bmatrix}
65.0000 & 0 & 0 & 0 & 0 \\
0 & 22.5471 & 0 & 0 & 0 \\
0 & 0 & 21.6874 & 0 & 0 \\
0 & 0 & 0 & 13.4036 & 0 \\
0 & 0 & 0 & 0 & 11.9008 \\
\end{bmatrix}
\]

V =

\[
\begin{bmatrix}
-0.4472 & -0.4498 & 0.2466 & -0.6627 & 0.3693 \\
-0.4472 & -0.0056 & 0.6627 & 0.2466 & -0.5477 \\
-0.4472 & 0.8202 & -0.0000 & 0.0000 & 0.3568 \\
-0.4472 & -0.0056 & -0.6627 & -0.2466 & -0.5477 \\
-0.4472 & -0.4045 & -0.2466 & 0.6627 & 0.3693 \\
\end{bmatrix}
\]

How about a 2 x 4 matrix?

```
>> M=A(1:2,1:4)
>>[U,S,V] = svd(M)
```

M =

\[
\begin{bmatrix}
17 & 24 & 1 & 8 \\
23 & 5 & 7 & 14 \\
\end{bmatrix}
\]

U =

\[
\begin{bmatrix}
-0.7428 & -0.6695 \\
-0.6695 & 0.7428 \\
\end{bmatrix}
\]
This time we get a square $U$ and $V$ for $M$.

**Pixel Matrices**

Matrices can be fairly large. For example, grayscale-images can be stored as matrices. For example, we have a big matrix broken into a few parts. first part:

\[
\begin{array}{cccccccccccccccc}
102 & 113 & 84 & 100 & 103 & 85 & 95 & 64 & 41 & 58 & 56 & 77 & 85 \\
124 & 99 & 108 & 115 & 87 & 89 & 62 & 75 & 48 & 60 & 65 & 80 & 86 \\
94 & 109 & 118 & 118 & 129 & 80 & 68 & 74 & 70 & 65 & 60 & 74 & 94 \\
58 & 49 & 59 & 104 & 121 & 69 & 112 & 104 & 73 & 81 & 70 & 85 & 102 \\
91 & 86 & 53 & 41 & 50 & 97 & 83 & 117 & 108 & 126 & 106 & 114 & 114 \\
126 & 95 & 83 & 41 & 22 & 104 & 111 & 110 & 95 & 122 & 129 & 130 & 101 \\
143 & 117 & 93 & 77 & 73 & 56 & 102 & 113 & 158 & 129 & 145 & 143 & 119 \\
132 & 138 & 111 & 97 & 59 & 71 & 149 & 138 & 80 & 71 & 155 & 139 & 72 \\
140 & 126 & 118 & 137 & 155 & 171 & 142 & 155 & 163 & 127 & 150 & 129 & 107 \\
133 & 127 & 120 & 152 & 145 & 155 & 152 & 167 & 134 & 69 & 95 & 85 & 60 \\
123 & 125 & 133 & 158 & 128 & 144 & 154 & 168 & 81 & 190 & 116 & 144 & 143 \\
118 & 109 & 143 & 175 & 146 & 157 & 153 & 163 & 114 & 158 & 202 & 198 & 120 \\
138 & 125 & 158 & 172 & 151 & 164 & 150 & 134 & 97 & 75 & 109 & 100 & 64 \\
151 & 142 & 163 & 165 & 152 & 154 & 162 & 137 & 106 & 95 & 75 & 88 & 108 \\
117 & 111 & 127 & 164 & 155 & 130 & 141 & 171 & 148 & 100 & 128 & 162 & 156 \\
122 & 117 & 122 & 142 & 143 & 128 & 136 & 160 & 154 & 144 & 170 & 170 & 164 \\
130 & 128 & 120 & 121 & 137 & 133 & 136 & 157 & 159 & 158 & 184 & 185 & 185
\end{array}
\]
second part:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>87</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>80</td>
<td>77</td>
<td>82</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>77</td>
<td>84</td>
<td>76</td>
<td>52</td>
<td>76</td>
<td>94</td>
<td>111</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>80</td>
<td>89</td>
<td>59</td>
<td>54</td>
<td>69</td>
<td>71</td>
<td>82</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>90</td>
<td>100</td>
<td>76</td>
<td>49</td>
<td>69</td>
<td>68</td>
<td>71</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>107</td>
<td>100</td>
<td>81</td>
<td>56</td>
<td>110</td>
<td>138</td>
<td>145</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>108</td>
<td>107</td>
<td>93</td>
<td>66</td>
<td>117</td>
<td>132</td>
<td>125</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>146</td>
<td>128</td>
<td>55</td>
<td>81</td>
<td>111</td>
<td>97</td>
<td>78</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>99</td>
<td>133</td>
<td>79</td>
<td>58</td>
<td>101</td>
<td>100</td>
<td>91</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>87</td>
<td>99</td>
<td>66</td>
<td>73</td>
<td>102</td>
<td>106</td>
<td>96</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>163</td>
<td>135</td>
<td>70</td>
<td>74</td>
<td>119</td>
<td>125</td>
<td>110</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>171</td>
<td>143</td>
<td>87</td>
<td>68</td>
<td>122</td>
<td>140</td>
<td>131</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>119</td>
<td>145</td>
<td>135</td>
<td>88</td>
<td>114</td>
<td>133</td>
<td>138</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>122</td>
<td>161</td>
<td>170</td>
<td>124</td>
<td>103</td>
<td>108</td>
<td>122</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>114</td>
<td>168</td>
<td>168</td>
<td>129</td>
<td>126</td>
<td>145</td>
<td>142</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>129</td>
<td>155</td>
<td>159</td>
<td>166</td>
<td>142</td>
<td>132</td>
<td>140</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>133</td>
<td>129</td>
<td>174</td>
<td>175</td>
<td>168</td>
<td>140</td>
<td>126</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>136</td>
<td>158</td>
<td>182</td>
<td>173</td>
<td>168</td>
<td>142</td>
<td>136</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>168</td>
<td>181</td>
<td>176</td>
<td>171</td>
<td>160</td>
<td>135</td>
<td>127</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>181</td>
<td>188</td>
<td>191</td>
<td>171</td>
<td>150</td>
<td>129</td>
<td>116</td>
<td>122</td>
<td></td>
</tr>
</tbody>
</table>

This is a part of the pixel matrix for this graph

![Pixel Matrix Image](image)

This is a grayscale image with size 19 × 22 pixels.

- The function `imread()`
  
  In MATLAB, we can use the function `imread()` to get a pixel matrix. Put the picture `rsz-cougar-gray.jpg` into your MATLAB folder, similar to when we read things from text files. Now, in MATLAB, type
  ```matlab
  >> A=imread('rsz-cougar-gray.jpg')
  ```
where rsz-cougar-gray.jpg is the name of this image. Each pixel from grayscale images can be represented by a number between 0 to 255. 0 means white, 255 means black, and gray colors are in the middle. Based on this resolution, can you guess what this image is?

- The function `imagesc()`
  If we have this data matrix of pixels, we can also recover and plot the image in MATLAB. To get the image back, type:

  ```matlab
  >> imagesc(A); colormap gray;
  ```

  into MATLAB. Since the origin image is a grayscale image, so we use the command `colormap gray`

If we have a color image pixel matrix $A$, we can use `image(A)` or `imagesc(A)` to get the image back.

If we want to deal with color images, we no longer have a single matrix. We have what is called a tensor, but it can be thought of as three different matrix layers, one for each color type - red, green, blue (RGB). Here is the origin picture of the previous image:

![Image of a cougar](rsz-cougar-gray.jpg)

Make sure the file named cougar.jpeg is in your MATLAB folder. If you want to save your pixel matrix as matrix $A$ and to reserve the function that how MATLAB maps your image to a matrix, let’s type

```matlab
>> [A,map]=imread('cougar.jpeg')
```

Here $A$ is that pixel matrix and $map$ is the transformation function from colors to numbers. Again, notice that because this is no longer a grayscale image, it is now a $183 \times 275 \times 3$ array. We can take a look $A$ in workspace and find there are three parameters for matrix $A$ where $A(:,:,1)$ means the color red layer, $A(:,:,2)$ means the color green layer and $A(:,:,3)$ means the color blue layer. Since we can represent rgb color into a three paremeters pair:
(r, g, b), so for example if we want to know the color at the pixel point (100, 200) on origin image then we need to get three numbers

\[ r = A(100, 200, 1), b = A(100, 200, 2), g = A(100, 200, 3) \]

to form a pair of rgb \((r, g, b)\).

We can plot the intensities of RGB by typing

\[
\texttt{>> imagesc(A(:, :, 1)); colormap hot;}
\]

for red, for example. Note that you’ll need to change the colormap to see the different intensities of a certain color. But, some areas should be brighter than others. Please search the key word: “Color map MATLAB”, there are many different options you can play with. Finally, we can also just plot the original image:

\[
\texttt{>> imagesc(A); colormap(map);} 
\]

and we should see the image as above. If you have a particular image you’d like to play with but want it in grayscale, MATLAB can also do that. Type

\[
\texttt{>> Agray = rgb2gray(A);} 
\]

and now the data is stored as a pixel matrix, which can be plotted with \texttt{imagesc()}.

As a final note, your project is going to involve importing an image and manipulating the matrix data associated with it. One way to do that, for example, is to dive into the effects of SVD on the image quality. This is an introductory way to do image compression and we’ll discuss more about it in class and during the online session.
Exercises

1. (Norm and Unit Vector) Given the vector $v_1 = (3, 4, 5)$. Find the norm of $v_1$ and the geometrical meaning of the norm of a vector.

2. (Inner Product, Orthogonal and Orthonormal) What is the geometrical meaning about inner product of two vectors?

3. (Inner Product, Orthogonal and Orthonormal) Given the vector $v_1 = (3, 4, 5)$ and $e_1 = (1, 0, 0)$. Find the inner product $v_1 \cdot e_1$ and the geometrical meaning of the inner product.

4. (Inner Product, Orthogonal and Orthonormal) Given a matrix $A$

$$
A = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0
\end{bmatrix}.
$$

Show that $A$ is an orthogonal matrix.