Another, very in-depth linear algebra review from CS229 is available here:
http://cs229.stanford.edu/section/cs229-linalg.pdf
And a video discussion of linear algebra from EE263 is here (lectures 3 and 4):
http://see.stanford.edu/see/lecturelist.aspx?coll=17005383-19c6-49ed-9497-2ba8bfcfe5f6
Outline

• **Vectors and matrices**
  – Basic Matrix Operations
  – Special Matrices
• **Transformation Matrices**
  – Homogeneous coordinates
  – Translation
• **Matrix inverse**
• **Matrix rank**
• **Singular Value Decomposition (SVD)**
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm
Vectors and matrices are just collections of ordered numbers that represent something: movements in space, scaling factors, pixel brightnesses, etc. We’ll define some common uses and standard operations on them.
Vector

• A column vector \( \mathbf{v} \in \mathbb{R}^{n \times 1} \) where

\[
\mathbf{v} = \begin{bmatrix}
v_1 \\
v_2 \\
\vdots \\
v_n
\end{bmatrix}
\]

• A row vector \( \mathbf{v}^T \in \mathbb{R}^{1 \times n} \) where

\[
\mathbf{v}^T = \begin{bmatrix}
v_1 & v_2 & \ldots & v_n
\end{bmatrix}
\]

\( T \) denotes the transpose operation
Vector

• We’ll default to column vectors in this class

\[
\mathbf{v} = \begin{bmatrix}
  v_1 \\
  v_2 \\
  \vdots \\
  v_n
\end{bmatrix}
\]

• You’ll want to keep track of the orientation of your vectors when programming in MATLAB

• You can transpose a vector \( \mathbf{V} \) in MATLAB by writing \( \mathbf{V}' \). (But in class materials, we will always use \( \mathbf{V}^T \) to indicate transpose, and we will use \( \mathbf{V}' \) to mean “V prime”)
Vectors have two main uses

- Vectors can represent an offset in 2D or 3D space
- Points are just vectors from the origin
- Data (pixels, gradients at an image keypoint, etc) can also be treated as a vector
- Such vectors don’t have a geometric interpretation, but calculations like “distance” can still have value
Matrix

• A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an array of numbers with size $m \downarrow$ by $n \rightarrow$, i.e. $m$ rows and $n$ columns.

$$\mathbf{A} = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\
  a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & a_{m3} & \ldots & a_{mn}
\end{bmatrix}$$

• If $m = n$, we say that $\mathbf{A}$ is square.
Images

- MATLAB represents an image as a matrix of pixel brightnesses.
- Note that matrix coordinates are NOT Cartesian coordinates. The upper left corner is \([y,x] = (1,1)\).
Color Images

• Grayscale images have one number per pixel, and are stored as an $m \times n$ matrix.
• Color images have 3 numbers per pixel – red, green, and blue brightnesses
• Stored as an $m \times n \times 3$ matrix
Basic Matrix Operations

• We will discuss:
  – Addition
  – Scaling
  – Dot product
  – Multiplication
  – Transpose
  – Inverse / pseudoinverse
  – Determinant / trace
Matrix Operations

• Addition

\[
\begin{bmatrix}
a & b \\
c & d \\
\end{bmatrix}
+ 
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
= 
\begin{bmatrix}
a + 1 & b + 2 \\
c + 3 & d + 4 \\
\end{bmatrix}
\]

— Can only add a matrix with matching dimensions, or a scalar.

\[
\begin{bmatrix}
a & b \\
c & d \\
\end{bmatrix}
+ 7
= 
\begin{bmatrix}
a + 7 & b + 7 \\
c + 7 & d + 7 \\
\end{bmatrix}
\]

• Scaling

\[
\begin{bmatrix}
a & b \\
c & d \\
\end{bmatrix}
\times 3
= 
\begin{bmatrix}
3a & 3b \\
3c & 3d \\
\end{bmatrix}
\]
Matrix Operations

• Inner product (dot product) of vectors
  – Multiply corresponding entries of two vectors and add up the result
  – \( x \cdot y \) is also \( |x| |y| \cos(\text{the angle between } x \text{ and } y) \)

\[
x^T y = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i \quad \text{(scalar)}
\]
Matrix Operations

• Inner product (dot product) of vectors
  – If $B$ is a unit vector, then $A \cdot B$ gives the length of $A$ which lies in the direction of $B$
Matrix Operations

• Multiplication

• The product AB is:

• Each entry in the result is (that row of A) dot product with (that column of B)

• Many uses, which will be covered later
Matrix Operations

• Multiplication example:

\[
\begin{pmatrix}
0 & 2 \\
4 & 6
\end{pmatrix}
\begin{pmatrix}
1 & 3 \\
5 & 7
\end{pmatrix}
\]

- Each entry of the matrix product is made by taking the dot product of the corresponding row in the left matrix, with the corresponding column in the right one.

\[
\begin{pmatrix}
0 & 2 \\
4 & 6
\end{pmatrix}
\begin{pmatrix}
10 & 14 \\
34 & 54
\end{pmatrix}
= 14
\]

\[
0 \cdot 3 + 2 \cdot 7 = 14
\]
Matrix Operations

• Powers
  – By convention, we can refer to the matrix product $AA$ as $A^2$, and $AAA$ as $A^3$, etc.
  – Obviously only square matrices can be multiplied that way
Matrix Operations

• Transpose – flip matrix, so row 1 becomes column 1

\[
\begin{bmatrix}
0 & 1 & \ldots
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 1 \\
2 & 3 \\
4 & 5
\end{bmatrix}^T = \begin{bmatrix}
0 & 2 & 4 \\
1 & 3 & 5
\end{bmatrix}
\]

• A useful identity:

\[
(ABC)^T = C^T B^T A^T
\]
Matrix Operations

• Determinant
  – \( \det(A) \) returns a scalar
  – Represents area (or volume) of the parallelogram described by the vectors in the rows of the matrix
  – For \( A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \), \( \det(A) = ad - bc \)
  – Properties: \( \det(AB) = \det(BA) \)
    \[ \det(A^{-1}) = \frac{1}{\det(A)} \]
    \[ \det(A^T) = \det(A) \]
    \( \det(A) = 0 \iff A \text{ is singular} \)
Matrix Operations

• Trace

\[
\text{tr}(A) = \text{sum of diagonal elements}
\]

\[
\text{tr} \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} = 1 + 7 = 8
\]

– Invariant to a lot of transformations, so it’s used sometimes in proofs. (Rarely in this class though.)

– Properties:

\[
\text{tr}(AB) = \text{tr}(BA)
\]
\[
\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)
\]
Special Matrices

• Identity matrix $\mathbf{I}$
  – Square matrix, 1’s along diagonal, 0’s elsewhere
  – $\mathbf{I} \cdot [\text{another matrix}] = [\text{that matrix}]$

$$
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
$$

• Diagonal matrix
  – Square matrix with numbers along diagonal, 0’s elsewhere
  – A diagonal $\cdot [\text{another matrix}]$ scales the rows of that matrix

$$
\begin{bmatrix}
3 & 0 & 0 \\
0 & 7 & 0 \\
0 & 0 & 2.5
\end{bmatrix}
$$
Special Matrices

• Symmetric matrix
\[ A^T = A \]
\[
\begin{bmatrix}
1 & 2 & 5 \\
2 & 1 & 7 \\
5 & 7 & 1
\end{bmatrix}
\]

• Skew-symmetric matrix
\[ A^T = -A \]
\[
\begin{bmatrix}
1 & -2 & -5 \\
2 & 1 & -7 \\
5 & 7 & 1
\end{bmatrix}
\]
Outline

• Vectors and matrices
  – Basic Matrix Operations
  – Special Matrices
• Transformation Matrices
  – Homogeneous coordinates
  – Translation
• Matrix inverse
• Matrix rank
• Singular Value Decomposition (SVD)
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm

Matrix multiplication can be used to transform vectors. A matrix used in this way is called a transformation matrix.
Transformation

- Matrices can be used to transform vectors in useful ways, through multiplication: \( x' = Ax \)
- Simplest is scaling:

\[
\begin{bmatrix}
s_x & 0 \\
0 & s_y
\end{bmatrix} \times \begin{bmatrix}
x \\
y
\end{bmatrix} = \begin{bmatrix}
s_xx \\
s_yy
\end{bmatrix}
\]

(Verify to yourself that the matrix multiplication works out this way)
Rotation

• How can you convert a vector represented in frame “0” to a new, rotated coordinate frame “1”?

• Remember what a vector is: [component in direction of the frame’s x axis, component in direction of y axis]
Rotation

• So to rotate it we must produce this vector: [component in direction of **new** x axis, component in direction of **new** y axis]
• We can do this easily with dot products!
• New x coordinate is [original vector] \textbf{dot} [the new x axis]
• New y coordinate is [original vector] \textbf{dot} [the new y axis]
Rotation

• Insight: this is what happens in a matrix*vector multiplication
  – Result x coordinate is [original vector] **dot** [matrix row 1]
  – So matrix multiplication can rotate a vector \( p \):

\[
R \times p = \text{rotated } p'
\]

\[
\begin{bmatrix}
.707 & .707 \\
-.707 & .707
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y
\end{bmatrix}
\rightarrow
\begin{bmatrix}
p_{x'} \\
p_{y'}
\end{bmatrix}
\]
Rotation

• Suppose we express a point in a coordinate system which is rotated left
• If we use the result in the same coordinate system, we have rotated the point right

> Thus, rotation matrices can be used to rotate vectors. We’ll usually think of them in that sense-- as operators to rotate vectors
2D Rotation Matrix Formula

Counter-clockwise rotation by an angle $\theta$

$$x' = \cos \theta \ x - \sin \theta \ y$$
$$y' = \cos \theta \ y + \sin \theta \ x$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = R \ P$$
Transformation Matrices

- Multiple transformation matrices can be used to transform a point:
  \[ p' = R_2 R_1 S \ p \]
- The effect of this is to apply their transformations one after the other, from right to left.
- In the example above, the result is \( (R_2 (R_1 (S \ p))) \)
- The result is exactly the same if we multiply the matrices first, to form a single transformation matrix:
  \[ p' = (R_2 R_1 S) \ p \]
Homogeneous system

• In general, a matrix multiplication lets us linearly combine components of a vector

\[
\begin{bmatrix}
a & b \\
c & d \\
\end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}
\]

– This is sufficient for scale, rotate, skew transformations.

– But notice, we can’t add a constant! 😞
Homogeneous system

– The (somewhat hacky) solution? Stick a “1” at the end of every vector:

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1 \\
\end{bmatrix}
= 
\begin{bmatrix}
ax + by + c \\
dx + ey + f \\
1 \\
\end{bmatrix}
\]

– Now we can rotate, scale, and skew like before, AND translate (note how the multiplication works out, above)

– This is called “homogeneous coordinates”
Homogeneous system

– In homogeneous coordinates, the multiplication works out so the rightmost column of the matrix is a vector that gets added.

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
0 & 0 & 1 \\
\end{bmatrix}
\times
\begin{bmatrix}
x \\
y \\
1 \\
\end{bmatrix}
=
\begin{bmatrix}
ax + by + c \\
dx + ey + f \\
1 \\
\end{bmatrix}
\]

– Generally, a homogeneous transformation matrix will have a bottom row of \([0 \ 0 \ 1]\), so that the result has a “1” at the bottom too.
Homogeneous system

- One more thing we might want: to divide the result by something
  - For example, we may want to divide by a coordinate, to make things scale down as they get farther away in a camera image
  - Matrix multiplication can’t actually divide
  - So, by convention, in homogeneous coordinates, we’ll divide the result by its last coordinate after doing a matrix multiplication

\[
\begin{bmatrix}
  x \\
  y \\
  7
\end{bmatrix} \Rightarrow \begin{bmatrix}
  x/7 \\
  y/7 \\
  1
\end{bmatrix}
\]
2D Translation using Homogeneous Coordinates

\[ P = (x, y) \rightarrow (x, y, 1) \]

\[ t = (t_x, t_y) \rightarrow (t_x, t_y, 1) \]

\[
P' \rightarrow \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{y} \\ 1 \end{bmatrix}
\]

\[
= \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \cdot P = T \cdot P
\]
Scaling
Scaling Equation

\[ P = (x, y) \rightarrow P' = (s_x x, s_y y) \]

\[ P = (x, y) \rightarrow (x, y, 1) \]

\[ P' = (s_x x, s_y y) \rightarrow (s_x x, s_y y, 1) \]

\[ P' \rightarrow \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S' & 0 \\ 0 & 1 \end{bmatrix} \cdot P = S \cdot P \]
Scaling & Translating

\[ P'' = T \cdot P' \]

\[ P' = S \cdot P \]

\[ P'' = T \cdot (S \cdot P) = T \cdot S \cdot P = A \cdot P \]
Scaling & Translating

\[ P'' = T \cdot S \cdot P = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} sx \cdot x + tx \\ sy \cdot y + ty \\ 1 \end{bmatrix} = \begin{bmatrix} S \cdot x \\ t \cdot y \\ 1 \end{bmatrix} \]
Translating & Scaling

\[ P''' = T \cdot S \cdot P = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix} \]

\[ P''' = S \cdot T \cdot P = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & s_x t_x \\ 0 & s_y & s_y t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + s_x t_x \\ s_y y + s_y t_y \\ 1 \end{bmatrix} \]
Rotation
Rotation Equations

Counter-clockwise rotation by an angle $\theta$

$x' = \cos \theta \ x - \sin \theta \ y$
$y' = \cos \theta \ y + \sin \theta \ x$

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

$P' = R \ P$
Rotation Matrix Properties

• Transpose of a rotation matrix produces a rotation in the opposite direction

\[ \mathbf{R} \cdot \mathbf{R}^T = \mathbf{R}^T \cdot \mathbf{R} = \mathbf{I} \]
\[ \det(\mathbf{R}) = 1 \]

• The rows of a rotation matrix are always mutually perpendicular (a.k.a. orthogonal) unit vectors
  – (and so are its columns)
Properties

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

A 2D rotation matrix is 2x2

Note: \( R \) belongs to the category of normal matrices and satisfies many interesting properties:

\[
R \cdot R^T = R^T \cdot R = I
\]

\[
\text{det}(R) = 1
\]
Rotation + Scaling + Translation

\[ P' = (T \ R \ S) \ P \]

\[
P' = T \cdot R \cdot S \cdot P = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \]

\[
= \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \]

\[
= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} R \ S & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \]

This is the form of the general-purpose transformation matrix.
Outline

• Vectors and matrices
  – Basic Matrix Operations
  – Special Matrices
• Transformation Matrices
  – Homogeneous coordinates
  – Translation
• **Matrix inverse**
• Matrix rank
• Singular Value Decomposition (SVD)
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm

The inverse of a transformation matrix reverses its effect
Inverse

• Given a matrix $A$, its inverse $A^{-1}$ is a matrix such that $AA^{-1} = A^{-1}A = I$

• E.g. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$

• Inverse does not always exist. If $A^{-1}$ exists, $A$ is invertible or non-singular. Otherwise, it’s singular.

• Useful identities, for matrices that are invertible:

$$(A^{-1})^{-1} = A$$

$$(AB)^{-1} = B^{-1}A^{-1}$$

$$A^{-T} \triangleq (A^T)^{-1} = (A^{-1})^T$$
Matrix Operations

• Pseudoinverse
  – Say you have the matrix equation $AX = B$, where $A$ and $B$ are known, and you want to solve for $X$
  – You could use MATLAB to calculate the inverse and premultiply by it: $A^{-1}AX = A^{-1}B \rightarrow X = A^{-1}B$
  – MATLAB command would be `inv(A)*B`
  – But calculating the inverse for large matrices often brings problems with computer floating-point resolution (because it involves working with very small and very large numbers together).
  – Or, your matrix might not even have an inverse.
Matrix Operations

• Pseudoinverse
  – Fortunately, there are workarounds to solve $AX=B$ in these situations. And MATLAB can do them!
  – Instead of taking an inverse, directly ask MATLAB to solve for $X$ in $AX=B$, by typing `A\B`
  – MATLAB will try several appropriate numerical methods (including the pseudoinverse if the inverse doesn’t exist)
  – MATLAB will return the value of $X$ which solves the equation
    • If there is no exact solution, it will return the closest one
    • If there are many solutions, it will return the smallest one
Matrix Operations

- MATLAB example:

\[
AX = B
\]

\[
A = \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\]

\[
\gg \ x = A\backslash B
\]

\[
x =
\begin{bmatrix}
1.0000 \\
-0.5000
\end{bmatrix}
\]
Outline

• Vectors and matrices
  – Basic Matrix Operations
  – Special Matrices
• Transformation Matrices
  – Homogeneous coordinates
  – Translation
• Matrix inverse
• **Matrix rank**
• Singular Value Decomposition (SVD)
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm

The rank of a transformation matrix tells you how many dimensions it transforms a vector to.
Linear independence

• Suppose we have a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$

• If we can express $\mathbf{v}_1$ as a linear combination of the other vectors $\mathbf{v}_2 \ldots \mathbf{v}_n$, then $\mathbf{v}_1$ is linearly dependent on the other vectors.
  – The direction $\mathbf{v}_1$ can be expressed as a combination of the directions $\mathbf{v}_2 \ldots \mathbf{v}_n$. (E.g. $\mathbf{v}_1 = .7 \mathbf{v}_2 - .7 \mathbf{v}_4$)

• If no vector is linearly dependent on the rest of the set, the set is linearly independent.
  – Common case: a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is always linearly independent if each vector is perpendicular to every other vector (and non-zero)
Linear independence

Linearly independent set  Not linearly independent
Matrix rank

• Column/row rank

\[
\text{col-rank}(\mathbf{A}) = \text{the maximum number of linearly independent column vectors of } \mathbf{A} \\
\text{row-rank}(\mathbf{A}) = \text{the maximum number of linearly independent row vectors of } \mathbf{A}
\]

– Column rank always equals row rank

• Matrix rank

\[
\text{rank}(\mathbf{A}) \triangleq \text{col-rank}(\mathbf{A}) = \text{row-rank}(\mathbf{A})
\]
Matrix rank

- For transformation matrices, the rank tells you the dimensions of the output
- E.g. if rank of $A$ is 1, then the transformation $p' = Ap$
  maps points onto a line.
- Here’s a matrix with rank 1:

$$
\begin{bmatrix}
1 & 1 \\
2 & 2 \\
\end{bmatrix} \times \begin{bmatrix}
x \\
y \\
\end{bmatrix} = \begin{bmatrix}
x + y \\
2x + 2y \\
\end{bmatrix}
$$

All points get mapped to the line $y=2x$.
Matrix rank

- If an $m \times m$ matrix is rank $m$, we say it’s “full rank”
  - Maps an $m \times 1$ vector uniquely to another $m \times 1$ vector
  - An inverse matrix can be found
- If rank $< m$, we say it’s “singular”
  - At least one dimension is getting collapsed. No way to look at the result and tell what the input was
  - Inverse does not exist
- Inverse also doesn’t exist for non-square matrices
Outline

• Vectors and matrices
  – Basic Matrix Operations
  – Special Matrices
• Transformation Matrices
  – Homogeneous coordinates
  – Translation
• Matrix inverse
• Matrix rank
• Singular Value Decomposition (SVD)
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm

SVD is an algorithm that represents any matrix as the product of 3 matrices. It is used to discover interesting structure in a matrix.
Singular Value Decomposition (SVD)

• There are several computer algorithms that can “factor” a matrix, representing it as the product of some other matrices.

• The most useful of these is the Singular Value Decomposition.

• Represents any matrix $A$ as a product of three matrices: $UΣV^T$

• MATLAB command: $[U,S,V]=\text{svd}(A)$
Singular Value Decomposition (SVD)

\[ U \Sigma V^T = A \]

- Where \( U \) and \( V \) are rotation matrices, and \( \Sigma \) is a scaling matrix. For example:

\[
\begin{bmatrix}
-0.40 & 0.916 \\
0.916 & 0.40
\end{bmatrix}
\begin{bmatrix}
5.39 & 0 \\
0 & 3.154
\end{bmatrix}
\begin{bmatrix}
-0.05 & 0.999 \\
0.999 & 0.05
\end{bmatrix}
= \begin{bmatrix}
3 & -2 \\
1 & 5
\end{bmatrix}
\]
Singular Value Decomposition (SVD)

• Beyond 2D:
  – In general, if $A$ is $m \times n$, then $U$ will be $m \times m$, $\Sigma$ will be $m \times n$, and $V^T$ will be $n \times n$.
  – (Note the dimensions work out to produce $m \times n$ after multiplication)

\[
U = \begin{bmatrix}
-.39 & -.92 \\
-.92 & .39
\end{bmatrix}
\times \begin{bmatrix}
9.51 & 0 & 0 \\
0 & .77 & 0
\end{bmatrix}
\times \begin{bmatrix}
-.42 & -57 & -70 \\
.81 & .11 & -58 \\
.41 & -.82 & .41
\end{bmatrix}
= \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]
Singular Value Decomposition (SVD)

- **U** and **V** are always rotation matrices.
  - Geometric rotation may not be an applicable concept, depending on the matrix. So we call them “unitary” matrices – each column is a unit vector.

- **Σ** is a diagonal matrix
  - The number of nonzero entries = rank of **A**
  - The algorithm always sorts the entries high to low

\[
U \begin{bmatrix}
-.39 & -.92 \\
-.92 & .39
\end{bmatrix}
\times \begin{bmatrix}
9.51 & 0 & 0 \\
0 & .77 & 0
\end{bmatrix}
\times
\begin{bmatrix}
-.42 & -.57 & -.70 \\
.81 & .11 & -.58 \\
.41 & -.82 & .41
\end{bmatrix}
= \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]
SVD Applications

• We’ve discussed SVD in terms of geometric transformation matrices
• But SVD of an image matrix can also be very useful
• To understand this, we’ll look at a less geometric interpretation of what SVD is doing
SVD Applications

\[
U = \begin{bmatrix}
-0.39 & -0.92 \\
-0.92 & 0.39
\end{bmatrix} \times \begin{bmatrix}
9.51 & 0 & 0 \\
0 & 0.77 & 0
\end{bmatrix} \times \begin{bmatrix}
-0.42 & -0.57 & -0.70 \\
-0.81 & 0.11 & 0.58 \\
0.41 & -0.82 & 0.41
\end{bmatrix} = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]

- Look at how the multiplication works out, left to right:
- Column 1 of \( U \) gets scaled by the first value from \( \Sigma \).

\[
U \Sigma = \begin{bmatrix}
-3.67 & -0.71 & 0 \\
-8.8 & 0.30 & 0
\end{bmatrix} \times \begin{bmatrix}
-0.42 & -0.57 & -0.70 \\
-0.81 & 0.11 & 0.58 \\
0.41 & -0.82 & 0.41
\end{bmatrix}
\]

- The resulting vector gets scaled by row 1 of \( V^T \) to produce a contribution to the columns of \( A \).
SVD Applications

\[
\begin{bmatrix}
-3.67 & -0.71 & 0 \\
-8.8 & 0.30 & 0
\end{bmatrix}
\begin{bmatrix}
-0.42 & -0.57 & -0.70 \\
0.81 & 0.11 & -0.58 \\
0.41 & -0.82 & 0.41
\end{bmatrix}
\begin{bmatrix}
1.6 & 2.1 & 2.6 \\
3.8 & 5.0 & 6.2
\end{bmatrix}
\]

\[
\begin{bmatrix}
-3.67 & -0.71 & 0 \\
-8.8 & 0.30 & 0
\end{bmatrix}
\begin{bmatrix}
-0.42 & -0.57 & -0.70 \\
0.81 & 0.11 & -0.58 \\
0.41 & -0.82 & 0.41
\end{bmatrix}
\begin{bmatrix}
-0.6 & -1 & 0.4 \\
0.2 & 0 & -0.2
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]

• Each product of \((column \ i \ of \ U) \cdot (value \ i \ from \ \Sigma) \cdot (row \ i \ of \ V^T)\) produces a component of the final matrix \(A\).
SVD Applications

\[
\begin{bmatrix}
-3.67 & -0.71 & 0 \\
-8.8 & 0.30 & 0
\end{bmatrix}
\begin{bmatrix}
\Sigma
\end{bmatrix}
\begin{bmatrix}
V^T
\end{bmatrix}
\begin{bmatrix}
A_{\text{partial}}
\end{bmatrix}
\begin{bmatrix}
A
\end{bmatrix}
\]

- We’re building \( A \) as a linear combination of the columns of \( U \)
- Using all columns of \( U \), we’ll rebuild the original matrix perfectly
- But, in real-world data, often we can just use the first few columns of \( U \) and we’ll get something close (e.g. the first \( A_{\text{partial}} \), above)
SVD Applications

\[
\begin{bmatrix}
-3.67 & -.71 & 0 \\
-8.8 & .30 & 0
\end{bmatrix}
\begin{bmatrix}
V^T \\
A_{\text{partial}}
\end{bmatrix}
\begin{bmatrix}
1.6 & 2.1 & 2.6 \\
3.8 & 5.0 & 6.2
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]

- We can call those first few columns of \( U \) the \textit{Principal Components} of the data.
- They show the major patterns that can be added to produce the columns of the original matrix.
- The rows of \( V^T \) show how the \textit{principal components} are mixed to produce the columns of the matrix.
SVD Applications

We can look at $\Sigma$ to see that the first column has a large effect while the second column has a much smaller effect in this example.
SVD Applications

- For this image, using only the first 10 of 300 principal components produces a recognizable reconstruction
- So, SVD can be used for image compression
Principal Component Analysis

\[
[U\Sigma \begin{array}{ccc} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{array} ] \times [V^T \begin{array}{ccc} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{array} ] = \begin{array}{c} \text{partial} \\ 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{array}
\]

- Remember, columns of \( U \) are the *Principal Components* of the data: the major patterns that can be added to produce the columns of the original matrix.
- One use of this is to construct a matrix where each column is a separate data sample.
- Run SVD on that matrix, and look at the first few columns of \( U \) to see patterns that are common among the columns.
- This is called *Principal Component Analysis* (or PCA) of the data samples.
Principal Component Analysis

\[
\begin{bmatrix}
3.67 & -0.71 & 0 \\
-8.8 & 0.30 & 0
\end{bmatrix}
\begin{bmatrix}
-0.42 & -0.57 & -0.70 \\
0.81 & 0.11 & -0.58 \\
0.41 & -0.82 & 0.41
\end{bmatrix}
\begin{bmatrix}
1.6 & 2.1 & 2.6 \\
3.8 & 5.0 & 6.2
\end{bmatrix}
\]

- Often, raw data samples have a lot of redundancy and patterns.
- PCA can allow you to represent data samples as weights on the principal components, rather than using the original raw form of the data.
- By representing each sample as just those weights, you can represent just the “meat” of what’s different between samples.
- This minimal representation makes machine learning and other algorithms much more efficient.
Outline

• Vectors and matrices
  – Basic Matrix Operations
  – Special Matrices
• Transformation Matrices
  – Homogeneous coordinates
  – Translation
• Matrix inverse
• Matrix rank
• Singular Value Decomposition (SVD)
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm

Computers can compute SVD very quickly. We’ll briefly discuss the algorithm, for those who are interested.
Addendum: How is SVD computed?

• For this class: tell MATLAB to do it. Use the result.

• But, if you’re interested, one computer algorithm to do it makes use of Eigenvectors
  – The following material is presented to make SVD less of a “magical black box.” But you will do fine in this class if you treat SVD as a magical black box, as long as you remember its properties from the previous slides.
Eigenvector definition

• Suppose we have a square matrix $A$. We can solve for vector $x$ and scalar $\lambda$ such that $Ax = \lambda x$

• In other words, find vectors where, if we transform them with $A$, the only effect is to scale them with no change in direction.

• These vectors are called eigenvectors (German for “self vector” of the matrix), and the scaling factors $\lambda$ are called eigenvalues

• An $m \times m$ matrix will have $\leq m$ eigenvectors where $\lambda$ is nonzero
Finding eigenvectors

• Computers can find an $x$ such that $Ax = \lambda x$ using this iterative algorithm:
  
  – $x=$ random unit vector
  – while($x$ hasn’t converged)
    • $x=Ax$
    • normalize $x$

• $x$ will quickly converge to an eigenvector
• Some simple modifications will let this algorithm find all eigenvectors
Finding SVD

• Eigenvectors are for square matrices, but SVD is for all matrices

• To do svd(A), computers can do this:
  – Take eigenvectors of $AA^T$ (matrix is always square).
    • These eigenvectors are the columns of $U$.
    • Square root of eigenvalues are the singular values (the entries of $\Sigma$).
  – Take eigenvectors of $A^TA$ (matrix is always square).
    • These eigenvectors are columns of $V$ (or rows of $V^T$)
Finding SVD

• Moral of the story: SVD is fast, even for large matrices
• It’s useful for a lot of stuff
• There are also other algorithms to compute SVD or part of the SVD
  – MATLAB’s svd() command has options to efficiently compute only what you need, if performance becomes an issue

A detailed geometric explanation of SVD is here:
http://www.ams.org/samplings/feature-column/fcarc-svd
What we have learned

• **Vectors and matrices**
  – Basic Matrix Operations
  – Special Matrices

• **Transformation Matrices**
  – Homogeneous coordinates
  – Translation

• **Matrix inverse**

• **Matrix rank**

• **Singular Value Decomposition (SVD)**
  – Use for image compression
  – Use for Principal Component Analysis (PCA)
  – Computer algorithm