Math 312: Class meeting 10

Some applications

Jeff Jauregui

September 26, 2012

Errors in transmission

- Situation: transmitting a binary data signal (0's and 1's) across some (noisy) channel.

Errors in transmission

- Situation: transmitting a binary data signal (0's and 1's) across some (noisy) channel.
- Errors (0s and 1s being randomly swapped) will occur.

Errors in transmission

- Situation: transmitting a binary data signal (0's and 1's) across some (noisy) channel.
- Errors (0s and 1s being randomly swapped) will occur.
- Examples: WiFi, cell phones, lasers reading DVDs, etc.

Errors in transmission

- Situation: transmitting a binary data signal (0's and 1's) across some (noisy) channel.
- Errors (0s and 1s being randomly swapped) will occur.
- Examples: WiFi, cell phones, lasers reading DVDs, etc.
- Problem: devise a scheme for errors to be detected, corrected automatically by receiver.

Naive solution 1

- One idea: sender repeats each bit (0 or 1) in segments of two.

Naive solution 1

- One idea: sender repeats each bit (0 or 1) in segments of two.
- Data `[0 1 0]` would be encoded as `[0 0 : 1 1 : 0 0]`, etc.

Naive solution 1

- One idea: sender repeats each bit (0 or 1) in segments of two.
- Data [0 1 0] would be encoded as [0 0 : 1 1 : 0 0], etc.
- If recipient gets [0 0 : 1 1 : 1 0], for instance, they know an error occurred in third segment of message.

Naive solution 1

- One idea: sender repeats each bit (0 or 1) in segments of two.
- Data `[0 1 0]` would be encoded as `[0 0 : 1 1 : 0 0]`, etc.
- If recipient gets `[0 0 : 1 1 : 1 0]`, for instance, they know an error occurred in third segment of message.
- But they can't correct the error!

Naive solution 1

- One idea: sender repeats each bit (0 or 1) in segments of two.
- Data `[0 1 0]` would be encoded as `[0 0 : 1 1 : 0 0]`, etc.
- If recipient gets `[0 0 : 1 1 : 1 0]`, for instance, they know an error occurred in third segment of message.
- But they can't correct the error!
- We'll ignore possibility of multiple errors.

Naive solution 2

- Next idea: sender repeats each bit (0 or 1) in segments of three.

Naive solution 2

- Next idea: sender repeats each bit (0 or 1) in segments of three.
- Data [0 1 0] would be encoded as [0 0 0 : 1 1 1 : 0 0 0], etc., and sent.

Naive solution 2

- Next idea: sender repeats each bit (0 or 1) in segments of three.
- Data [0 1 0] would be encoded as [0 0 0 : 1 1 1 : 0 0 0], etc., and sent.
- If recipient gets [0 0 0 : 1 1 0 : 0 0 0], for instance, they know an error occurred in second segment.

Naive solution 2

- Next idea: sender repeats each bit (0 or 1) in segments of three.
- Data [0 1 0] would be encoded as [0 0 0 : 1 1 1 : 0 0 0], etc., and sent.
- If recipient gets [0 0 0 : 1 1 0 : 0 0 0], for instance, they know an error occurred in second segment.
- Second segment should have been 1 1 1, so message can be corrected.

Naive solution 2

- Next idea: sender repeats each bit (0 or 1) in segments of three.
- Data [0 1 0] would be encoded as [0 0 0 : 1 1 1 : 0 0 0], etc., and sent.
- If recipient gets [0 0 0 : 1 1 0 : 0 0 0], for instance, they know an error occurred in second segment.
- Second segment should have been 1 1 1, so message can be corrected.
- The cost: requires 3 times as much data to send.

$(7, 4)$ Hamming code

- Hamming codes: Richard Hamming 1950

$(7, 4)$ Hamming code

- Hamming codes: Richard Hamming 1950
- "$(7, 4)$ Hamming code" takes a 4 bit message, encodes it as 7 bits, requiring only $7/4 = 1.75$ times as much data.

$(7,4)$ Hamming code

- Hamming codes: Richard Hamming 1950
- "$(7,4)$ Hamming code" takes a 4 bit message, encodes it as 7 bits, requiring only $7/4 = 1.75$ times as much data.
- Can detect and correct single errors.

$(7,4)$ Hamming code

- Hamming codes: Richard Hamming 1950
- "$(7,4)$ Hamming code" takes a 4 bit message, encodes it as 7 bits, requiring only $7/4 = 1.75$ times as much data.
- Can detect and correct single errors.
- Setup: say the senders message is x_1, x_2, x_3, x_4 (each 0 or 1). Denote by the vector:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}.$$

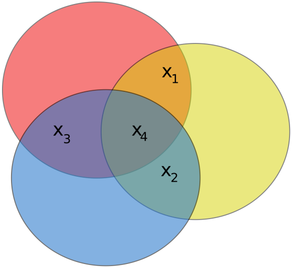
Parity

- We'll use addition "mod 2", meaning:

$$
\begin{aligned}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 0,
\end{aligned}
$$

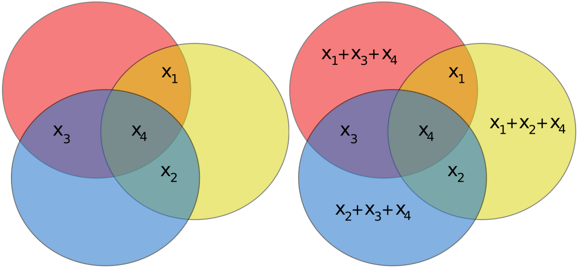
How it works

- Write the bits in a Venn diagram:



How it works

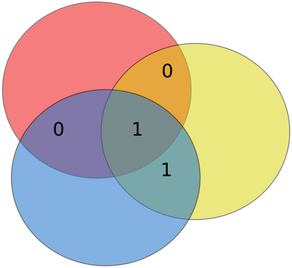
- Write the bits in a Venn diagram:



- ... then fill in the 3 circles with the sum of the bits inside that circle, taken mod 2.

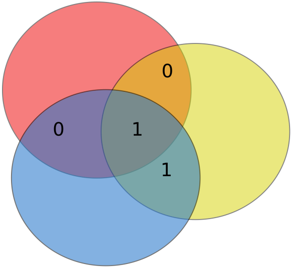
Example

- If original message is [0 1 0 1], we have



Example

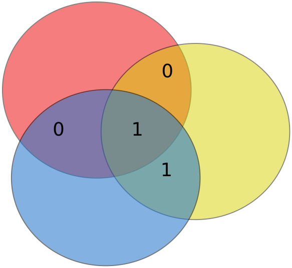
- If original message is `[0 1 0 1]`, we have



- ... filled in as?

Example

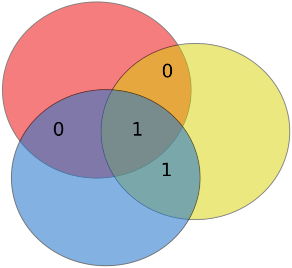
- If original message is `[0 1 0 1]`, we have



- ... filled in as?
- The sender sends 7 bits: the original four, plus the additional 3 "parity bits" (starting at top left, moving clockwise):

Example

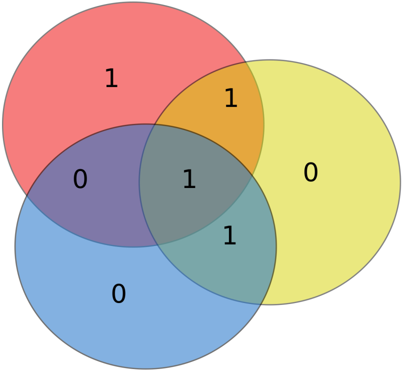
- If original message is [0 1 0 1], we have



- ... filled in as?
- The sender sends 7 bits: the original four, plus the additional 3 “parity bits” (starting at top left, moving clockwise):
- [0 1 0 1 1 0 0] gets sent.

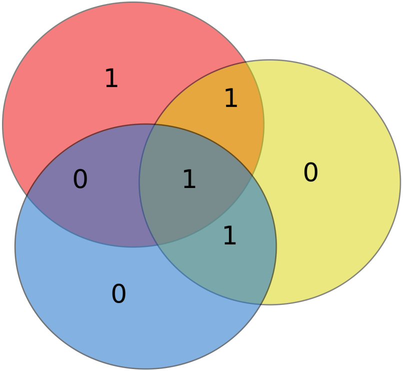
Correcting errors

- Now, recipient gets 7 bits, and fills out the picture in order. Example:



Correcting errors

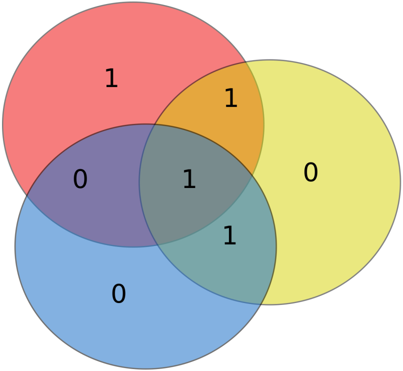
- Now, recipient gets 7 bits, and fills out the picture in order. Example:



- They check the validity of each parity bit to locate errors.

Correcting errors

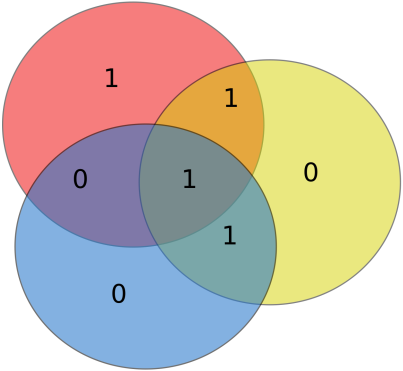
- Now, recipient gets 7 bits, and fills out the picture in order. Example:



- They check the validity of each parity bit to locate errors.
- In which position did error occur?

Correcting errors

- Now, recipient gets 7 bits, and fills out the picture in order. Example:



- They check the validity of each parity bit to locate errors.
- In which position did error occur?
- Corrected message is?

Matrix formalism: encoding

- Say $\vec{x} \in \mathbb{R}^4$ is original message.

Matrix formalism: encoding

- Say $\vec{x} \in \mathbb{R}^4$ is original message. Then $G\vec{x} \in \mathbb{R}^7$ is the 7-bit version, where

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Matrix formalism: encoding

- Say $\vec{x} \in \mathbb{R}^4$ is original message. Then $G\vec{x} \in \mathbb{R}^7$ is the 7-bit version, where

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

- G is "code generator matrix"

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$.

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$. To perform error check, what they do is make sure, that mod 2:

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0 \\ c_1 + c_2 + c_4 + c_6 &= 0 \\ c_2 + c_3 + c_4 + c_7 &= 0. \end{aligned}$$

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$. To perform error check, what they do is make sure, that mod 2:

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0 \\ c_1 + c_2 + c_4 + c_6 &= 0 \\ c_2 + c_3 + c_4 + c_7 &= 0. \end{aligned}$$

- In other words, they check $P\vec{c} = \vec{0}$ in $\mathbb{R}^3$, where

$$P = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$. To perform error check, what they do is make sure, that mod 2:

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0 \\ c_1 + c_2 + c_4 + c_6 &= 0 \\ c_2 + c_3 + c_4 + c_7 &= 0. \end{aligned}$$

- In other words, they check $P\vec{c} = \vec{0}$ in $\mathbb{R}^3$, where

$$P = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

- P is "parity check matrix".

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$. To perform error check, what they do is make sure, that mod 2:

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0 \\ c_1 + c_2 + c_4 + c_6 &= 0 \\ c_2 + c_3 + c_4 + c_7 &= 0. \end{aligned}$$

- In other words, they check $P\vec{c} = \vec{0}$ in $\mathbb{R}^3$, where

$$P = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

- P is "parity check matrix".
- If $P\vec{c} \neq \vec{0}$, recipient can deduce where error occurred.

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$. To perform error check, what they do is make sure, that mod 2:

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0 \\ c_1 + c_2 + c_4 + c_6 &= 0 \\ c_2 + c_3 + c_4 + c_7 &= 0. \end{aligned}$$

- In other words, they check $P\vec{c} = \vec{0}$ in $\mathbb{R}^3$, where

$$P = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

- P is "parity check matrix".
- If $P\vec{c} \neq \vec{0}$, recipient can deduce where error occurred.
- G, P can be viewed as linear transformations.

Matrix formalism: decoding

- Recipient gets some $\vec{c} \in \mathbb{R}^7$. To perform error check, what they do is make sure, that mod 2:

$$\begin{aligned} c_1 + c_3 + c_4 + c_5 &= 0 \\ c_1 + c_2 + c_4 + c_6 &= 0 \\ c_2 + c_3 + c_4 + c_7 &= 0. \end{aligned}$$

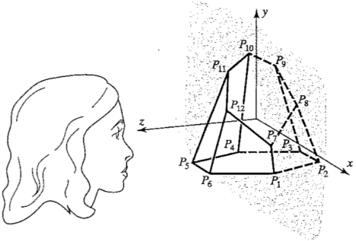
- In other words, they check $P\vec{c} = \vec{0}$ in $\mathbb{R}^3$, where

$$P = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

- P is "parity check matrix".
- If $P\vec{c} \neq \vec{0}$, recipient can deduce where error occurred.
- G, P can be viewed as linear transformations.
- Messages without errors can be thought of as image of G and kernel of P. How so?

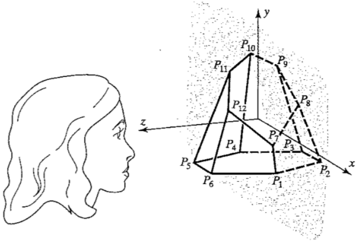
Displaying 3D graphics

- Imagine n points $(x_1, y_1, z_1), \ldots (x_n, y_n, z_n) \in R^3$, connected by various line segments:



Displaying 3D graphics

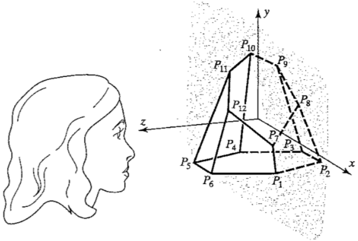
- Imagine n points $(x_1, y_1, z_1), \ldots (x_n, y_n, z_n) \in R^3$, connected by various line segments:



- Viewer located along z-axis.

Displaying 3D graphics

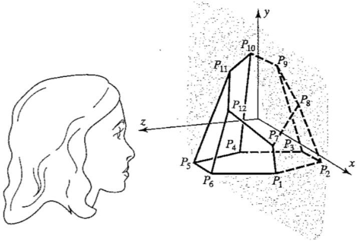
- Imagine n points $(x_1, y_1, z_1), \ldots (x_n, y_n, z_n) \in R^3$, connected by various line segments:



- Viewer located along z-axis.
- Imagine points being projected onto a flat screen in $z = 0$ plane.

Displaying 3D graphics

- Imagine n points $(x_1, y_1, z_1), \ldots (x_n, y_n, z_n) \in R^3$, connected by various line segments:

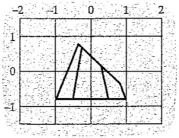


- Viewer located along z-axis.
- Imagine points being projected onto a flat screen in $z = 0$ plane. Store points as a $3 \times n$ matrix:

$$P = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ z_1 & z_2 & \cdots & z_n \end{bmatrix}$$

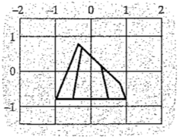
Views

- Straight-on view looks like



Views

- Straight-on view looks like

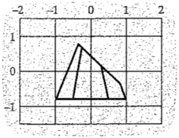


- Can apply various linear transformations of $\mathbb{R}^3$, for instance

$$S = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix}$$

Views

- Straight-on view looks like



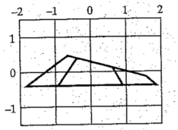
- Can apply various linear transformations of $\mathbb{R}^3$, for instance

$$S = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix}$$

- Matrix product "SP" means apply S to each column.

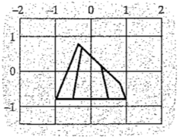
Scaling example

- Here's an example



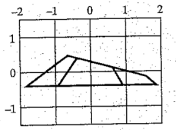
$\alpha = 1.8, \beta = 0.5, \gamma = 3.0.$

- Original view was:



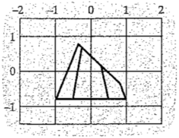
Scaling example

- Here's an example



$\alpha = 1.8, \beta = 0.5, \gamma = 3.0.$

- Original view was:



- In this initial discussion, z coordinate doesn't factor in directly.

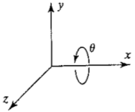
Rotations

- We can also rotate about x, y, and z axes. With $\theta = \pi/2$:

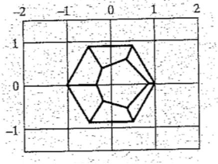
Rotations

- We can also rotate about x, y, and z axes. With $\theta = \pi/2$:

Rotation about the x-axis



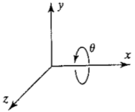
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$



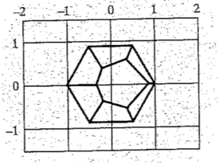
Rotations

- We can also rotate about x, y, and z axes. With $\theta = \pi/2$:

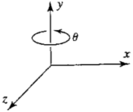
Rotation about the x-axis



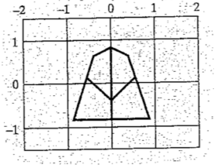
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$



Rotation about the y-axis

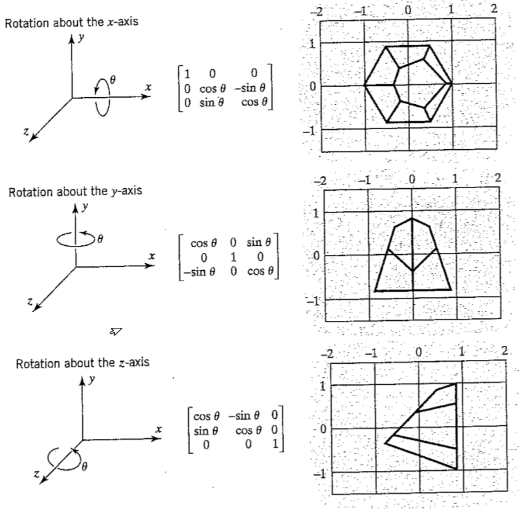


$$\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$



Rotations

- We can also rotate about x, y, and z axes. With $\theta = \pi/2$:



Stereoscopic views

- By composing x, y, and z rotations, we can create oblique views on the object.

Stereoscopic views

- By composing x, y, and z rotations, we can create oblique views on the object.
- See handout.