Information Theory

Chapter IV: Error control coding

What is error control coding?

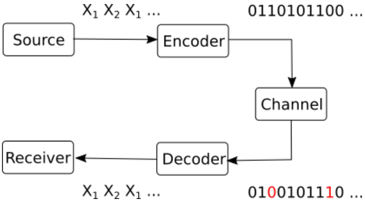

Figure 1: Communication system

- The second main task of coding: error control
- Protect information against channel errors

Mutual information and error control

- Mutual information $I(X, Y)$ = the information transmitted on the channel
- Why do we still need error control?
- Example: consider the following BSC channel (p = 0.01, $p(x_1) = 0.5$, $p(x_2) = 0.5$):

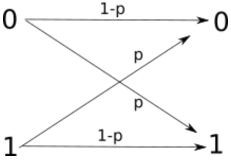

Figure 2: Binary symmetric channel (BSC)

- The receiver would like to know the source messages
  - In absence of communication, the uncertainty is $H(X) = 1$ bit/msg
  - With communication, the uncertainty is $H(X|Y) \approx 0.081$ bit/msg

Mutual information and error control

- The reduction in uncertainty due to communication = mutual information
  - $I(X, Y) = H(X) - H(X|Y) = \approx 0.919$ bit/msg
- Even though we have large I(X,Y), we still lose some information
  - Imagine downloading a file, but having 1% wrong bits

Why is error control needed?

- In most communications it is required that *all* bits are received correctly
    - Not 1% errors, not 0.1%, not 0.0001%. **None!**
- But that is not possible unless the channel is ideal.
- So what do to? **Error control coding**

Modelling the errors on the channel

- We consider only binary channels (symbols $= \{0, 1\}$
- An error $=$ a bit is changed from 0 to 1 or viceversa
- Errors can appear:
  - **independently**: each bit on its own
  - in **packets of errors**: groups of errors

Modelling the errors on the channel

- Changing the value of a bit = modulo-2 sum with 1
- Value of a bit remains the same = modulo-2 sum with 0

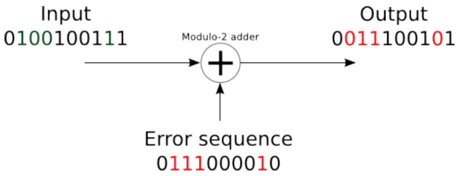

Figure 3: Channel error model

- Channel model we use (simple):
  - The transmitted sequence is summed modulo-2 with an **error sequence**
  - Where the error sequence is 1, there is a bit error
  - Where the error sequence is 0, there is no error

$$\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$$

Mathematical properties of modulo-2 arithmetic

- Product is the same as for normal arithmetic
- Multiplication is distributive just like in normal case

$$a(b \oplus c) = ab \oplus ac$$

- Subtraction = addition. There is no negativation. Each number is its own negative

$$a \oplus a = 0$$

Error detection vs correction

What can we do about errors?

- **Error detection**: find out if there is any error in the received sequence
  - don't know exactly where, so cannot correct the bits, but can discard whole sequence
  - perhaps ask the sender to retransmit (examples: TCP/IP, internet communication etc)
  - easier to do
- **Error correction**: find out exactly which bits have errors, if any
  - locating the error = correcting error (for binary channels)
  - can correct all errored bits by inverting them
  - useful when can't retransmit (data is stored: on HDD, AudioCD etc.)
  - harder to do than mere detection

Overview of error control coding process

The process of error control:

1. Want to send a sequence of k bits = **information word**

$$\mathbf{i} = i_1 i_2 ... i_k$$

2. For each possible information word, the coder assigns a **codeword** of length $n > k$:

$$\mathbf{c} = c_1 c_2 ... c_n$$

3. The codeword is sent on the channel instead of the original information word
4. The receiver receives a sequence $\hat{\mathbf{c}} \approx \mathbf{c}$, with possible errors:

$$\hat{\mathbf{c}} = \hat{c}_1 \hat{c}_2 ... \hat{c}_n$$

5. The decoding algorithm detects/corrects the errors in $\hat{\mathbf{c}}$

Definitions

- An **error correcting code** is an association between the set of all possible information words to a set of codewords
  - Each possible information word $\mathbf{i}$ has a certain codeword $\mathbf{c}$
- The association can be done:
  - randomly: codewords are selected and associated randomly to the information words
  - based on a certain rule: the codeword is computed with some algorithm from the information word
- A code is a **block code** if it operates with words of *fixed size*
  - Size of information word $\mathbf{i} = k$, size of codeword $\mathbf{c} = n$, $n > k$
  - Otherwise it is a *non-block code*
- A code is **linear** if any linear combination of codewords is also a codeword
- The **coding rate** of a code is:

$$R = k/n$$

Definitions

- A code C is an t-**error-detecting** code if it is able to *detect* t or less errors
- A code C is an t-**error-correcting** code if it is able to *correct* t or less errors
- Examples: at blackboard

A first example: parity bit

- Add parity bit to a 8-bit long information word, before sending on a channel
  - coding rate $R = 8/9$
  - can detect 1 error in a 9-bit codeword
  - detection algorithm: check if parity bit matches data
  - fails for 2 errors
  - cannot correct error (don't know where it is located)
- Add more parity bits to be able to locate the error
  - Example at blackboard
  - coding rate $R = 8/12$
  - can detect and correct 1 error in a 9-bit codeword

A second example: repetition code

- Repeat same block of data n times
  - want to send a k-bit information word
  - codeword to send = the information word repeated $n = 5$ times
  - coding rate $R = k/n = 1/5$
  - can detect and correct 2 errors, and maybe even more if they do not affect the same bit
  - error correcting algorithm = majority rule
  - not very efficient

Redundancy

- Because $k < n$, we introduce **redundancy**
  - to transmit k bits of information we actually send more bits (n)
- Error control coding adds redundancy, while source coding aims to reduce redundancy –> Contradiction?
  - but now redundancy is added in a controlled way, with a purpose
- Source coding and error control coding in practice: do sequentially, independently
  1. First perform source coding, eliminating redundancy in representation of data
  2. Then perform error control coding, adding redundancy for protection

Shannon's noisy channel theorem (second theorem, channel coding theorem)

- A coding rate is called **achievable** for a channel if, for that rate, there exists a coding and decoding algorithm guaranteed to correct all possible errors on the channel

Shannon's noisy channel coding theorem (second theorem)

For a given channel, all rates below capacity $R < C$ are achievable. All rates above capacity, $R > C$, are not achievable.

Channel coding theorem explained

In layman terms:

- For all coding rates $R < C$, **there is a way** to recover the transmitted data perfectly (decoding algorithm will detect and correct all errors)
- For all coding rates $R > C$, **there is no way** to recover the transmitted data perfectly

Example:

- Send binary digits on a BSC channel with capacity 0.7 bits/message
- For any coding rate $R < 0.7$ there exist an error correction code that allow perfect recovery
  - $R < 0.7 =$ for every 7 bits of data, coding adds more than 3 bits, on average
- With less than 3 bits for every 7 bits of data $=>$ impossible to recover all data

Ideas behind channel coding theorem

- The rigorous proof of the theorem is too complex to present
- Key ideas of the proof:
  - Use very long information words, $k \to \infty$
  - Use random codes, compute the probability of having error after decoding
  - If $R < C$, *in average for all possible codes*, the probability of error after decoding goes to 0
  - If the average for all codes goes to 0, there exists at least on code better than the average
  - That is the code we should use
- **The theorem does not tell what code to use**, only that some code exists
  - There is no clue of how to actually find the code in practice
  - Only some general principles:
    - using longer information words is better
    - random codewords are generally good
- In practice, cannot use infinitely long codewords, so will only get a *good enough* code

Distance between codewords

Practical ideas for error correcting codes:

- If a codeword $\mathbf{c_1}$ is received with errors and becomes identical to another codeword $\mathbf{c_2}$ $==>$ cannot detect any errors
  - Receiver will think it received a correct codeword c_2 and the information word was $\mathbf{i_2}$, but actually it was $\mathbf{i_1}$
- We want codewords as different as possible from each other
- How to measure this difference? **Hamming distance**

Hamming distance

- The **Hamming distance** of two binary sequences $\mathbf{a}$, $\mathbf{b}$ of length $n =$ the total number of bit differences between them

$$d_H(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{N} a_i \oplus b_i$$

- We need at least $d_H(a, b)$ bit changes to convert one sequence into another
- It satisfies the 3 properties of a metric function:
  1. $d_H(\mathbf{a}, \mathbf{b}) \geq 0 \quad \forall \mathbf{a}, \mathbf{b}$, with $d_H(\mathbf{a}, \mathbf{b}) = 0 \Leftrightarrow \mathbf{a} = \mathbf{b}$
  2. $d_H(\mathbf{a}, \mathbf{b}) = d_H(\mathbf{b}, \mathbf{a}), \forall \mathbf{a}, \mathbf{b}$
  3. $d_H(\mathbf{a}, \mathbf{c}) \leq d_H(\mathbf{a}, \mathbf{b}) + d_H(\mathbf{b}, \mathbf{c}), \forall \mathbf{a}, \mathbf{b}, \mathbf{c}$
- The **minimum Hamming distance of a code**, $d_{H\,min} =$ the minimum Hamming distance between any two codewords $\mathbf{c_1}$ and $\mathbf{c_2}$
- Example at blackboard

Nearest-neighbor decoding

Coding:

- Design a code with large $d_{H\,min}$
- Send a codeword **c** of the code

Decoding:

- Receive a word **r**, that may have errors
- Error detecting:
  - check if r is part of the codewords of the code C:
  - if r is part of the code, decide that there have been no errors
  - if r is not a codeword, decide that there have been errors
- Error correcting:
  - if **r** is a codeword, decide there are no errors
  - else, choose codeword **nearest** to the received **r**, in terms of Hamming distance
  - this is known as **nearest-neighbor decoding**

Performance of nearest neighbor decoding

Theorem:

- If the minimum Hamming distance of a code is $d_{H\,min}$, then:
  1. the code can *detect* up to $d_{H\,min} - 1$ errors
  2. the code can *correct* up to $\left\lfloor \frac{d_{H\,min}-1}{2} \right\rfloor$ errors using nearest-neighbor decoding

Consequence:

- It is good to have $d_{H\,min}$ as large as possible
  - This implies longer codewords, i.e. smaller coding rate, i.e. more redundancy

Performance of nearest neighbor decoding

Proof:

1. at least $d_{H\,min}$ binary changes are needed to change one codeword into another, $d_{H\,min} - 1$ is not enough $=>$ the errors are detected
2. the received word $\mathbf{r}$ is closer to the original codeword than to any other codeword $=>$ nearest-neighbor algorithm will find the correct one
   - because $\left\lfloor \frac{d_{H\,min}-1}{2} \right\rfloor$ = less than half the distance to another codeword

Note: if the number of errors is higher, can fail:

- Detection failure: decide that there were no errors, even if they were (more than $d_{H\,min} - 1$)
- Correction failure: choose a wrong codeword

Example: blackboard

Linear block codes

- A code is a **block code** if it operates with words of *fixed size*
  - Size of information word $\mathbf{i} = k$, size of codeword $\mathbf{c} = n$, $n > k$
  - Otherwise it is a *non-block code*
- A code is **linear** if any linear combination of codewords is also a codeword
- A code is called **systematic** if the codeword contains all the information bits explicitly, unaltered
  - coding merely adds supplementary bits besides the information bits
  - codeword has two parts: the information bits and the parity bits
  - example: parity bit added after the information bits
- Otherwise the code is called **non-systematic**
  - the information bits are not explicitly visible in the codeword
- Example: at blackboard

Generator matrix

- All codewords for a linear block code can be generated via a **matrix multiplication**:

$$\mathbf{i} \cdot [G] = \mathbf{c}$$

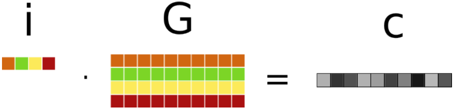

Figure 4: Codeword construction with generator matrix

- $[G]$ = **generator matrix** of size $k \times n$
- Row-wise interpretation:
  - $\mathbf{c}$ = a linear combination of rows in $[G]$
  - The rows of $[G]$ = a *basis* for the linear code
- All operations are done in modulo-2 arithmetic

Parity check matrix

- Every generator matrix $[G]$ has a complementary **parity-check matrix** $[H]$ such that
$$0 = [H] \cdot [G]^T$$
- How to check if a binary word is a codeword or not?
- For every codeword $\mathbf{c}$ generated with $[G]$:
$$\boxed{0 = [H] \cdot \mathbf{c}^T}$$
- Proof:
$$\mathbf{i} \cdot [G] = \mathbf{c}$$
$$[G]^T \cdot \mathbf{i}^T = \mathbf{c}^T$$
$$[H] \cdot \mathbf{c}^T = [H] \cdot [G]^T \cdot \mathbf{i}^T = 0$$

Parity check matrix

- $[H]$ is the **parity-check matrix**, size $= (n-k) \times n$
- $[G]$ and $[H]$ are related, one can be deduced from the other
- The resulting vector $z = [H] \cdot [c]^T$ is the **syndrome**
- All codewords generated with $[G]$ will produce 0 when multiplied with $[H]$
- All binary sequences that are not codewords will produce $\neq 0$ when multiplied with $[H]$
- Column-wise interpretation of multiplication:

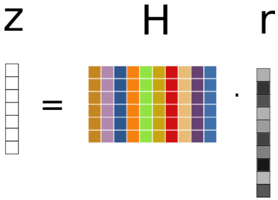

Figure 5: Codeword checking with parity-check matrix

[G] and [H] for systematic codes

- For systematic codes, [G] and [H] have special forms
- Generator matrix
  - first part = identity matrix
  - second part = some matrix Q

$$[G]_{k \times n} = [Q_{k \times (n-k)} \;\; I_{k \times k} s]$$

- Parity-check matrix
  - first part = identity matrix
  - second part = same Q, transposed

$$[H]_{(n-k) \times n} = [I_{(n-k) \times (n-k)} \;\; Q^T_{(n-k) \times k}]$$

- Can easily compute one from the other
- Example at blackboard

Interpretation as parity bits

- The additional bits added by coding are actually just parity bits
  - Proof: write the generation equations (example)
- Generator matrix $[G]$ creates the codeword as:
  - first part = information bits (systematic code, first part of $[G]$ is identity matrix)
  - additional bits = combinations of information bits = *parity bits*
- Parity-check matrix $[H]$ checks if parity bits correspond to information bits
  - Proof: write down the parity check equation (see example)
- If all parity bits match the data, the syndrome $\mathbf{z} = 0$
  - otherwise the syndrome $\mathbf{z} \neq 0$
- **Generator & parity-check matrices are just mathematical tools for easy computation & checking of parity bits**

Syndrome-based error detection

Syndrome-based error **detection** for linear block codes:

1. generate codewords with generator matrix:

$$\mathbf{i} \cdot [G] = \mathbf{c}$$

2. send codeword **c** on the channel
3. a random error word **e** is applied on the channel
4. receive word $\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$
5. compute **syndrome** of **r**:

$$\mathbf{z} = [H] \cdot \mathbf{r}^T$$

6. Decide:
   - If $\mathbf{z} = 0 => \mathbf{r}$ has no errors
   - If $\mathbf{z} \neq 0 => \mathbf{r}$ has errors

Syndrome-based error correction

Syndrome-based error **correction** for linear block codes:

- Syndrome $\mathbf{z} \neq 0 => \mathbf{r}$ has errors, we need to locate them
- The syndrome is the effect only of the error word:

$$\mathbf{z} = [H] \cdot \mathbf{r}^T = [H] \cdot (\mathbf{c}^T \oplus \mathbf{e}^T) = [H] \cdot \mathbf{e}^T$$

7. Create a **syndrome lookup table**:
   - for every possible error word $\mathbf{e}$, compute the syndrome $\mathbf{z} = [H] \cdot \mathbf{e}^T$
   - start with error words with 1 error (most likely), then with 2 errors (less likely), and so on
8. Locate the syndrome $\mathbf{z}$ in the table, read the corresponding error word $\widehat{\mathbf{e}}$
9. Find the correct word:
   - adding the error word again will invert the errored bits back to the originals

$$\widehat{\mathbf{c}} = \mathbf{r} \oplus \widehat{\mathbf{e}}$$

Example

Example: at blackboard

Conditions on [H] for error detection and correction

Conditions for syndrome-based error **detection**:

- To detect a single error: every column of $[H]$ must be non-zero
- To detect two error: sum of any two columns of $[H]$ cannot be zero
  - that means all columns are different
- To detect *n* errors: sum of any *n* or less columns of $[H]$ cannot be zero

Conditions for syndrome-based error **correction**:

- To correct a single error: all columns of $[H]$ are different
  - so the syndromes, for a single error, are all different
- To correct *n* errors: sum of any *n* or less columns of $[H]$ are all different
  - much more difficult to obtain than for decoding

Rearranging the columns of $[H]$ (the order of bits in the codeword) does not affect performance

Hamming codes

- Simple class of linear error-correcting codes
- Definition: a **Hamming code** is a linear block code where the columns of $[H]$ are *the binary representation of all numbers from 1 to* $2^r - 1$, $\forall r \geq 2$
- Example (blackboard): (7,4) Hamming code
- Is systematic (but the columns are shuffled)
  - Could move the r columns with a single 1 to the left side, forming identity matrix

Properties of Hamming codes

- From definition of $[H]$ it follows:
  1. Codeword has length $n = 2^r - 1$
  2. r bits are parity bits
  3. $k = 2^r - r - 1$ bits are information bits
- Notation: **(n,k) Hamming code**
  - n = codeword length = $2^r - 1$,
  - k = number of information bits = $2^r - r - 1$
  - Example: (7,4) Hamming code, (15,11) Hamming code, $(127, 120)$ Hamming code

Properties of Hamming codes

- Can detect two errors
    - All columns are different => can detect 2 errors
    - Sum of two columns equal to a third => cannot correct 3

OR

- Can correct one error
    - All columns are different => can correct 1 error
    - Sum of two columns equal to a third => cannot correct 2
    - Non-systematic: syndrome = error position
- But not simultaneously!
    - same non-zero syndrome can be obtained with 1 or 2 errors, can't distinguish

SECDED Hamming codes

- Add an additional parity bit to differentiate the two cases
  - = sum of all n bits of the codeword
- Compute syndrome of the received word *without the additional parity bit*
  - If 1 error happened: syndrome is non-zero, parity bit does not match
  - If 2 errors happened: syndrome is non-zero, parity bit matches (the two errors cancel out)
  - If 3 errors happened: same as 1, can't differentiate
- Now can simultaneously differentiate between:
  - 1 error: go ahead and perform correction
  - 2 errors: can detect, but do not perform correction
- If correction is never attempted, can detect up to 3 errors
  - minimum Hamming distance = 4 (no proof)
  - don't know if 1 error or more, so can't try correction
- Known as SECDED Hamming codes
  - **S**ingle **E**rror **C**orrection - **D**ouble **E**rror **D**etection

Decoding SECDED Hamming codes

Decoding with detection and correction

1. Compute syndrome of received word without the additional parity bit
2. If zero, no error
3. If non-zero, check parity bit:
    3.1 If does not match => one error => perform correction
    3.2 If does match => two errors => do not correct

Decoding with detection only

1. Compute syndrome of received word, also check additional parity bit
2. If syndrome is zero and parity bit is ok => no error
3. If syndrome non-zero or parity bit does not match => 1 or 2 or 3 errors have happened, do not correct

Summary until now

- Systematic codes: information bits + parity bits
- Generator matrix: use to generate codeword

$$\mathbf{i} \cdot [G] = \mathbf{c}$$

- Parity-check matrix: use to check if a codeword

$$0 = [H] \cdot \mathbf{c}^T$$

- Syndrome:

$$\mathbf{z} = [H] \cdot \mathbf{r}^T$$

- Syndrome-based error detection: syndrome non-zero
- Syndrome-based error correction: lookup table
- Hamming codes: $[H]$ contains all numbers $1 ... 2^r - 1$
- SECDED Hamming codes: add an extra parity bit

Cyclic codes

Definition: **cyclic codes** are a particular class of linear block codes for which *every cyclic shift of a codeword is also a codeword*

- Cyclic shift: cyclic rotation of a sequence of bits (any direction)
- Are a particular class of linear block codes, so all the theory up to now still applies
  - they have a generator matrix, parity check matrix etc.
- But they can be implemented more efficient than general linear block codes (e.g. Hamming)
- Used **everywhere** under the common name **CRC** (**C**yclic **R**edundancy **C**heck)
  - Network communications (Ethernet), data storage in Flash memory

Binary polynomials

- Every binary sequence corresponds to a polynomial with binary coefficients:

$$10010111 \to X^7 \oplus X^4 \oplus X^2 \oplus X \oplus 1$$

(From now on, by “codeword” we also mean the corresponding polynomial)

- Can perform all operations with these polynomials:
    - addition, multiplication, division etc. (examples)
- There are efficient circuits for performing multiplications

Circuits for multiplication binary polynomials

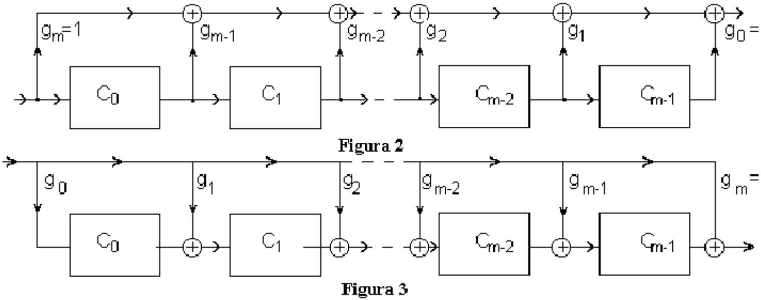

Figura 2

Figura 3

Figure 6: Circuits for polynomial multiplication

Circuits for division binary polynomials

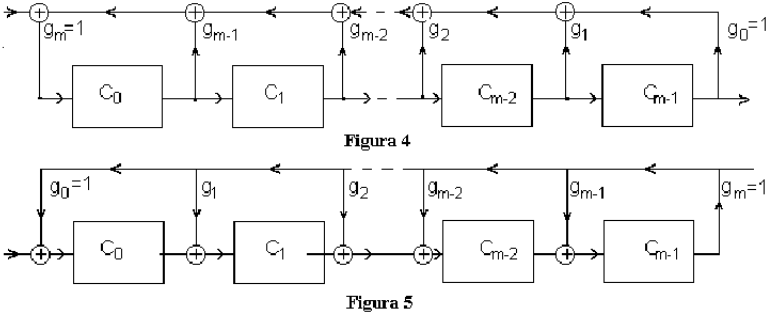

Figure 7: Circuits for polynomial division

Generator polynomial

Theorem:

All the codewords of a cyclic code are multiples of a certain polynomial $g(x)$, known as **generator polynomial**.

Properties of generator polynomial $g(x)$:

- The generator polynomial has first and last coefficient equal to 1.
- The generator polynomial is a factor of $X^n \oplus 1$
- The *degree* of $g(x)$ is $n - k$, where:
  - The codeword = polynomial of degree $n - 1$ (n coefficients)
  - The information polynomial = polynomial of degree $k - 1$ (k coefficients)

$$(k-1) + (n-k) = n - 1$$

Finding a generator polynomial

Theorem:

If $g(x)$ is a polynomial of degree $(n-k)$ and is a factor of $X^n \oplus 1$, then $g(x)$ generates a (n,k) cyclic code

Example:

$$x^7 \oplus 1 = (1 \oplus X)(1 \oplus X + \oplus X^3)(1 \oplus X^2 \oplus X^3)$$

Each factor generates a code:

- $1 \oplus X$ generates a (7,6) cyclic code
- $1 \oplus X \oplus X^3$ generates a (7,4) cyclic code
- $1 \oplus X^2 \oplus X^3$ generates a (7,4) cyclic code

Computing the codewords

Non-systematic codeword generation:

- Codeword = information polynomial * $g(x)$

$$\boxed{c(x) = i(x) \cdot g(x)}$$

Systematic codeword generation:

$$\boxed{c(x) = b(x) \oplus X^{n-k} i(x)}$$

where $b(x)$ is the remainder of dividing $X^{n-k} i(x)$ to $g(x)$:

$$X^{n-k} i(x) = a(x) g(x) \oplus b(x)$$

- (Proof: at blackboard)

Cyclic code encoder circuits

- Coding = based on polynomial multiplications and divisions
- Efficient circuits for multiplication / division exist
- Similar circuit exists for systematic codeword generation (draw on blackboard)

Error detection with cyclic codes

- Like usual for linear codes: check if received word is codeword or not
- Every codeword is multiple of $g(x)$
- Check if received word is actually dividing with $g(x)$
  - Use a circuit for division of polynomials
- If remainder is 0 $=>$ it is a codeword, no error
- If remainder is non-0 $=>$ error detected!
- Cyclic codes have very good error detection capabilities

Error correction capability

Theorem:

Any (n,k) cyclic codes is capable of detecting any error **burst** of length $n - k$ or less.

- A large fraction of longer bursts can also be detected (but not all)
- For non-burst errors (random): more difficult to analyze

Error correction with cyclic codes

- Like usual for linear codes: lookup table based on remainder
- Remainder of division = the effect of the error polynomial
- Create lookup table: for every error word, compute remainder
- Search the table for the remainder of the received word => find error word

Summary of cyclic codes

- Generated using a generator polynomial $g(x)$
- Non-systematic:

$$c(x) = i(x) \cdot g(x)$$

- Systematic:

$$c(x) = b(x) \oplus X^{n-k} i(x)$$

  - $b(x)$ is the remainder of dividing $X^{n-k} i(x)$ to $g(x)$

- Syndrome = remainder of division $r(x)$ to $g(x)$
- Error detection: remainder (syndrome) non-zero
- Error correction: lookup table