Learning Enhancement Project 2019-2020

Final Report

An Introduction to MAGMA for Coding Theory

Eimear Byrne Giuseppe Cotardo

An interactive workshop on the computer algebra system MAGMA to complement the mathematical theory of error correcting codes.

School of Mathematics and Statistics

University College Dublin

June 14, 2021

Table of Contents

Acknowledgements

We would like to thank Aleksandar Gubic for the IT support during the workshop. This project was made possible through funding provided in the form of a Learning Enhancement Project grant, awarded by Teaching & Learning at University College Dublin and funded by the HEA and the National Forum for Teaching and Learning Enhancement.

Chapter 1: **Introduction**

Coding theory is a major component of Applied Algebra. It has many applications in the mathematics of communications, but its primary application is in designing efficient and reliable data transmission methods. Therefore, it has many links with other scientific disciplines, such as electrical engineering and computer science.

The material of the module MATH30180 *An Introduction to Coding Theory* requires students to work with objects from linear algebra and combinatorics and to study mathematical models and algorithms. In order gain a concrete understanding of the theoretical content, examples and implementations are essential. To this end, the computer algebra package Magma provides powerful tools that allow students to create and compile computer programs and to apply the methods they learned in class to problems with realistic parameters. It is also a tool that educators can use to create class examples, problem sheets and check any sort of algebraic computations.

Coding theory is often taught in mathematics departments with an emphasis on its theoretical aspects. Associating the teaching of coding theory with a MAGMA workshop made this module unique and more complete.

Magma is widely used by researchers of coding theory, so that having this basic knowledge enhances the students' further studies and enable them to apply Magma in other areas of algebra.

Overall feedback from the students was that working with Magma did help them gain a better insight into coding theory and improved their learning of the course material. The workshop was implemented by the tutor, Giuseppe Cotardo, who also oversaw the project work and handled the assessment.

Chapter 2: **Design and Intended Outcomes**

The learning enhancement project took the form of a 12-hour workshop, delivered over 3 days during the study period (March 9-11,2020). Assessment comprised short daily projects along with a longer group project to be submitted later in the semester and a presentation to be given at the end of the semester.

This initiative aimed to introduce the students to the computational algebra system Magma and to enhance student learning of algebraic coding theory. The software Magma is a high-level programming language and it is designed for computations in algebra, number theory, algebraic geometry and algebraic combinatorics and provides a mathematically rigorous environment for defining and working with structures such as groups, rings, fields, modules, algebras, schemes, curves, graphs, designs, codes and many others.

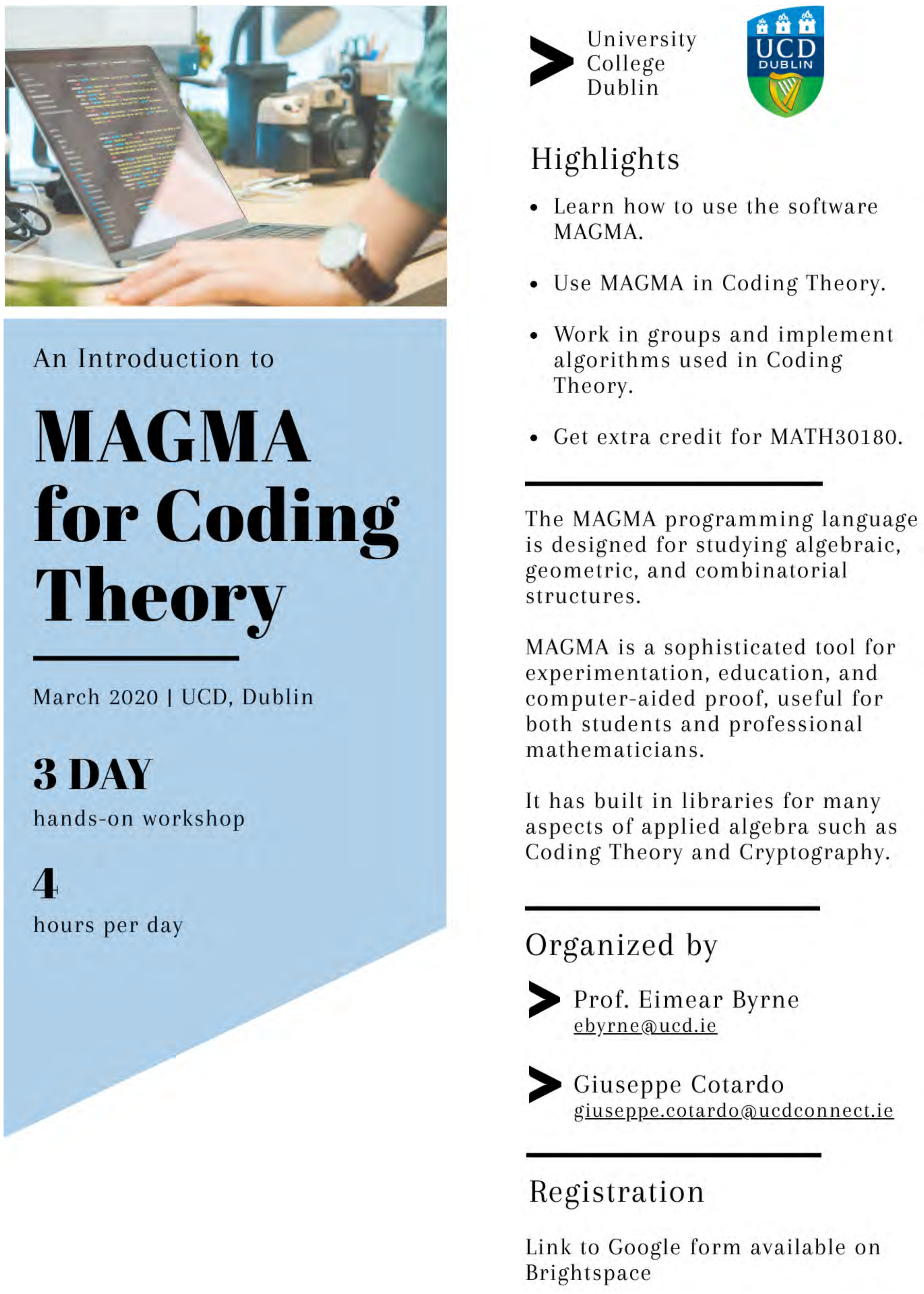
The School of Mathematics and Statistics has a license for Magma installed on the server Magnet. Participants were provided with an account on the server in order to use the software. Magma also has a free online calculator that anyone can use, which was also sufficient for the computational requirements of the workshop.

The workshop was intended to introduce students to basic programming concepts and to the libraries of Magma relevant to coding theory and linear algebra. The Coding Theory and Linear Algebra libraries allow users to construct finite fields and different types of families of codes, to investigate their properties, and furthermore to implement decoding algorithms.

Making connections between theory and computation is a valuable skill in coding theory, which is itself an application of linear algebra. Using this software, students were challenged to explore computations to gain a deeper level of understanding of the topic that would not be feasible without a computer. The goal was that the use of this digital resource would act as a scaffold for student learning, enabling them to visualise and grasp difficult abstract concepts.

Another projected outcome of this initiative was that students would increase their proficiency in programming in Magma . This would be acquired by hands-on activity in the labs and via project work. By the end of the semester the students would have developed skills in designing and writing computer programs and have gained an insight into fundamental concepts and terminology.

Finally, the initiative intended to allow the tutor to apply his extensive experience in using Magma in a teaching context in which he played a major role. In addition, the developed materials can be reused for future teaching.



An Introduction to

MAGMA for Coding Theory

March 2020 | UCD, Dublin

3 DAY

hands-on workshop

4

hours per day

University College Dublin

UCD DUBLIN

Highlights

- Learn how to use the software MAGMA.
- Use MAGMA in Coding Theory.
- Work in groups and implement algorithms used in Coding Theory.
- Get extra credit for MATH30180.

The MAGMA programming language is designed for studying algebraic, geometric, and combinatorial structures.

MAGMA is a sophisticated tool for experimentation, education, and computer-aided proof, useful for both students and professional mathematicians.

It has built in libraries for many aspects of applied algebra such as Coding Theory and Cryptography.

Organized by

Prof. Eimear Byrne
ebyrne@ucd.ie

Giuseppe Cotardo
giuseppe.cotardo@ucdconnect.ie

Registration

Link to Google form available on Brightspace

Figure 2.1: Workshop Poster

Chapter 3: **The Participants**

About twenty students registered to Math30180 Coding Theory participated in the workshop. Some graduate students also attended out of personal interest. Most of the participants had little or no prior programming experience. The workshop was completed just the day before the Covid-19 restrictions came into effect. After that, project work was carried out remotely during the period March-May 2020. While the restrictions had an impact on the group project work that participants were to carry out autonomously, they were extremely resourceful in achieving their aims and they all submitted their projects. The final presentations were made via Zoom in late May 2020.

Chapter 4: **Delivered Outcomes**

The content of the workshop was coordinated with the theoretical content of MATH30180 and delivered according to the schedule in Table 4.1. In this way, students were able to make links between their theoretical understanding of Coding Theory, which they acquired in lectures and tutorials and concrete implementations using Magma .

Date	Time	Topics
9 March 2020	10:00 - 11:50	Overview of Magma and Aggregate Structures
	11:50 - 13:00	Lunch Break
	13:00 - 14:50	Conditional and Iterative Statements, and Functions
10 March 2020	10:00 - 11:50	Ring and Fields
	11:50 - 13:00	Lunch Break
	13:00 - 14:50	Vector and Matrix Spaces
11 March 2020	10:00 - 11:50	Error-Correcting Codes
	11:50 - 13:00	Lunch Break
	13:00 - 14:50	Error-Correcting Codes and Projects Presentation

Table 4.1: Schedule of the workshop.

The first part of the lessons were devoted to the basic programming using the software. The tutor, Giuseppe Cotardo, introduced the topic, the relevant Magma functions and explained how to use them. In the second part, students worked to implement code themselves, and build up their own collection of programmes and examples. These small projects were submitted daily and form part of the assessment. Participants were then given longer group projects to do autonomously, which were submitted on May 29th, 2020.

4.1 In-Workshop Projects

The students were asked to implement the following 5 small projects during the workshop. For each task, we also include an example of implementation provided by the students.

Exercise 1. Compute the first twenty lines of Pascal's triangle.

Implementation:

```
Pascal := [[1]];
for x in [1..19] do
    new := [1];
    for y in [1.. x-1] do
        new := Append(new, Pascal[x][y] + Pascal[x][y+1]);
    end for;
    new := Append(new, 1);
    Pascal := Append(Pascal, new);
end for;
Pascal;
```

Exercise 2. Write a function `IsFermatPseudoPrime(n)` that returns `true` if and only if we have $a^{n-1} \equiv 1 \mod n$ for all $0 < a < n$ such that $\mathrm{GCD}(a, n) = 1$.

```
Implementation:
function IsFermatPseudoPrime(n)
    for a in [1..n-1] do
        if GCD(a,n) eq 1 then
            if (a^(n-1) mod n) ne 1 then
                return false;
            end if;
        end if;
    end for;
    return true;
end function;
```

Exercise 3. Write a function `Factorize(p)`, p prime, that returns the factorization of the polynomial $f(x) = x^p - x - 1 \in \mathbb{F}_p[x]$ of the form $\prod_{a \in \mathbb{F}_p} (\theta + a)$ where θ is a root of $f(x)$ in some extension of $\mathbb{F}_p$.

```
Implementation:
function Factorize(p)
    if IsPrime(p) eq false then
        return "argument must be prime";
    else
        P<x>:=PolynomialRing(GF(p));
        f := x^p-x-1;
        G<a>:=ext<GF(p)|f>;
        root := Roots(f,G)[1][1];
        S:=[];
        for i in GF(p) do
            S:= S cat [root+i];
        end for;
    end if;
    D<x>:=PolynomialRing(G);
    T:=[x-r:r in S];
    return T;
end function;
```

Exercise 4. Write a function `RotatePolynomial(f,k)` that, for a given polynomial $f(x) \in \mathbb{F}_q[x]$ of degree d, return the polynomial $g(x) \in \mathbb{F}_q[x]$ of degree at most d obtained by rotating the coefficient of $f(x)$ by k.

```
Implementation:
function RotatePolynomial(f,k)
    c:= Rotate(Coefficients(f),k);
    g:=Parent(f)!c;
    return g;
end function;
```

Exercise 5. Implement the function `VarshamovBound(K,n,d)` that returns the Varshamov Bound for a finite field K and integers n, d, such that $0 \leq d \leq n$.

Implementation:

```
function VarshamovBound(K,n,d)
    if not (Type(K) eq FldFin) then
        return "The first input must be a finite field";
    elif not (Type(d) eq RngIntElt) then
        return "The second input must be an integer";
    elif not (Type(n) eq RngIntElt) then
        return "The third input must be an integer";
    elif (Type(d) eq RngIntElt) and (Type(n) eq RngIntElt) and (n lt d)
    ↪  then
        return "The third input must be less than or equal the third";
    elif (Type(d) eq RngIntElt) and (Type(n) eq RngIntElt) and not (d
    ↪  ge 2) then
        return "The third input must be greater than or equal 2";
    elif (Type(K) eq FldFin) and (Type(d) eq RngIntElt) and (Type(n) eq
    ↪  RngIntElt) and (n ge d) and d gt 1 then
        q := #K;
        sum := 0;
        for i in [0..(d-2)] do
            sum := sum + Binomial(n-1,i) * (q-1)^i;
        end for;
        frac := (q^n)/(sum);
        return frac;
    else
        return "Unknown Error";
    end if;
end function;
```

4.2 Final Projects

As part of the assessment, the participants worked in groups on the following projects, which involved implementation of functions for the analysis and the decoding of some codes. The maximum score of each project was related to its level of difficulty, from a minimum of 2 points to a maximum of 5 points. The students chose their preferred project and worked with their group on its implementation, which was then submitted as a report containing any code used as well as a description of the underlying theory. The report also included comments on the code, any difficulties encountered during implementation and the solutions they found to these issues.

During the workshop, we agreed with the students to have an extra meeting, after the submission in order to give the groups the opportunity to briefly present their work to the other participants. Due to the pandemic and the consequent impossibility to physically meet in the university, the final presentations were done virtually. This was included as part of the assessment.

4.2.1 Project 1 [2pt]

Implement a function which, for a given a code C, returns some properties of the code, i.e. Is C MDS? Is C perfect? Is C self-dual? Is C a divisible code? Etc.

Compare the outputs with the function already implemented in MAGMA, like `IsMDS`, `IsPerfect`, `IsSelfDual`, etc.

A *divisible code* C is a code such that there exist a constant c which divides all the weights of C.

4.2.2 Project 2 [2pt]

Implement the following functions.

1. `Puncture(C,i)`:

INPUT:	An $\mathbb{F}_q - [n, k, d]$ linear code C and an integer $1 \leq i \leq n$.
OUTPUT:	The code C' obtained by puncturing the i-th coordinate from each codeword of C.

2. `Shorten(C,i)`:

INPUT:	An $\mathbb{F}_q - [n, k, d]$ linear code C and an integer $1 \leq i \leq n$.
OUTPUT:	The code $\overline{C}$ obtained by shortening the i-th coordinate from each codeword of C.

Compare the outputs of your functions with them of the functions `PunctureCode(C,i)` and `ShorthenCode(C,i)`, already present in MAGMA.

4.2.3 Project 3 [2pt]

Implement the following functions.

1. `Sum(C,D)`:

INPUT:	An $\mathbb{F}_q - [n, k_1, d_1]$ linear code C and an $\mathbb{F}_q - [n, k_2, d_2]$ linear code D.
OUTPUT:	The direct sum code E of C and D.

2. `Plotkin(C,D)`:

INPUT:	An $\mathbb{F}_q - [n, k_1, d_1]$ linear code C and an $\mathbb{F}_q - [n, k_2, d_2]$ linear code D.
OUTPUT:	The Plotkin sum code E of C and D.

Compare the outputs of your functions with them of the functions `DirectSum(C,D)` and `PlotkinSum(C,D)`, already present in MAGMA.

4.2.4 Project 4 [3pt]

Investigate the properties of the $[23, 12, 7]$ Golay code and of the $[24, 12, 8]$ extended Golay code over $\mathbb{F}_2$. Compute their weight distribution. Are they divisible-codes? Compare the properties of the two codes. Do they attain any bound? Etc.

A *divisible code* C is a code such that there exist a constant c which divides all the weights of C.

4.2.5 Project 5 [3pt]

Implement a function that simulate a transmission of a message through a noisy channel.

INPUT:	A message vector $\boldsymbol{m} \in \mathbb{F}_q^k$ and an $\mathbb{F}_q - [n, k, d]$ linear code C.
OUTPUT:	The message vector $\boldsymbol{m}$, the code C, the codeword $\boldsymbol{c}$ associated to $\boldsymbol{m}$, the received vector $\boldsymbol{r}$ and a string which says if the decoding was successful of not. In case of successful decoding, the function should also return the error vector $\boldsymbol{e}$ and the decoded word $\overline{\boldsymbol{c}}$.

Compare the outputs of your functions with them of the function `Decode(C,y)`, already present in MAGMA.

4.2.6 Project 6 [4pt]

An Hadamard matrix H_{2^n} is a $2^n \times 2^n$ square matrix whose entries are either $+1$ or -1 and whose rows are mutually orthogonal. H_{2^n} satisfies $H_{2^n} H_{2^n}^t = 2^n I_{2^n}$. It is possible to construct an Hadamard matrix recursively. Indeed,

$$H_{2^n} = \begin{bmatrix} H_{2^{n-1}} & H_{2^{n-1}} \\ H_{2^{n-1}} & -H_{2^{n-1}} \end{bmatrix}$$

with $H_1 = [1]$.

Implement a function that construct the $2^n \times 2^n$ Hadamard matrix using the recursive approach explained above.

INPUT:	An integer n.
OUTPUT:	The Hadamard matrix H_{2^n}.

It it possible to construct a non-linear $\mathbb{F}_2 - (2^n, 2^{n+1}, 2^{n-1})$ code C using an Hadamard matrix H_{2^n}. The 2^{n+1} codewords of C are the rows of H_{2^n} and the rows of $-H_{2^n}$. Notice that to obtain the binary code C, the mapping $-1 \mapsto 1$, $1 \mapsto 0$ is applied to the matrix elements.

Implement a function that return the $\mathbb{F}_2 - (2^n, 2^{n+1}, 2^{n-1})$ Hadamard code C.

INPUT:	An integer n.
OUTPUT:	The Hadamard code C.

Investigate the properties of these code for some values of n. Are they MDS? Are they perfect? Are they divisible codes? Etc.

4.2.7 Project 7 [5pt]

Implement the algorithm below for decoding the $\mathbb{F}_2 - [24, 12, 8]$ extended Golay code C with generator matrix $G = [I|B]$.

INPUT:	A received vector $\boldsymbol{r} \in \mathbb{F}_2^{24}$.
OUTPUT:	The codeword $\boldsymbol{c}$ obtained by decoding $\boldsymbol{r}$ and the error vector $\boldsymbol{e}$ in case of successful decoding. A request for retransmission, otherwise.

Let $\boldsymbol{e} = (\boldsymbol{e}_L|\boldsymbol{e}_R)$ be the error vector. Notice that, since C is self-dual, G is also a parity-check matrix for C. Therefore, we can easily compute two different syndromes:

$$S_1(\boldsymbol{e}) = \boldsymbol{e}H^t = (\boldsymbol{e}_L|\boldsymbol{e}_R)(B^t|I_{12})^t = \boldsymbol{e}_L B + \boldsymbol{e}_R$$
$$S_2(\boldsymbol{e}) = \boldsymbol{e}G^t = (\boldsymbol{e}_L|\boldsymbol{e}_R)(I_{12}|B)^t = \boldsymbol{e}_L + \boldsymbol{e}_R B^t$$

Notice that $S_2(\boldsymbol{e}) = S_1(\boldsymbol{e})B^t$.

ALGORITHM:

1. Compute the syndrome $S_1(\boldsymbol{r}) = \boldsymbol{r}H^t = \boldsymbol{r}(B^t|I_{12})^t$.
   (a) If $\mathrm{wt}(S_1(\boldsymbol{r})) \leq 3$, then the error vector is $e = (0|S_1(\boldsymbol{r}))$ and you can decode.
   (b) If $\mathrm{wt}(S_1(\boldsymbol{r})) > 3$, then compute $\mathrm{wt}(S_1(\boldsymbol{r}) + B_i)$ for all $i = 1, \ldots, 12$, where B_i is the i-th row of B.
      i. If $\mathrm{wt}(S_1(\boldsymbol{r}) + B_i) \leq 2$ for some i, then the error vector is $e = (S_1(\boldsymbol{r}) + B_i|\boldsymbol{\delta_i})$, where $\boldsymbol{\delta_i}$ is the vector in $\mathbb{F}_2^{24}$ with 1 in position i and 0 elsewhere. You can decode.
      ii. If $\mathrm{wt}(S_1(\boldsymbol{r}) + B_i) \leq 2$ for more than one i, choose the one(s) with smallest Hamming weight and decode as in point (1)(b)(i).
2. If $\mathrm{wt}(S_1(\boldsymbol{r}) + B_i) \geq 3$ for all $i = 1, \ldots, 12$, then compute the syndrome $S_2(\boldsymbol{e})$.
   (a) If $\mathrm{wt}(S_2(\boldsymbol{r})) \leq 3$, then the error vector is $e = (S_2(\boldsymbol{r})|0)$ and you can decode.
   (b) If $\mathrm{wt}(S_2(\boldsymbol{r})) > 3$, then compute $\mathrm{wt}(S_2(\boldsymbol{r}) + B_i)$ for all $i = 1, \ldots, 12$.
      i. If $\mathrm{wt}(S_2(\boldsymbol{r}) + B_i) \leq 2$ for some i, then the error vector is $e = (\boldsymbol{\delta_i}|S_2(\boldsymbol{r}) + B_i)$, and you can decode.
      ii. If $\mathrm{wt}(S_2(\boldsymbol{r}) + B_i) \leq 2$ for more than one i, choose the one(s) with smallest Hamming weight and decode as in point (2)(b)(i).
3. If $\boldsymbol{e}$ is not determined (i.e. if $\mathrm{wt}(S_1(\boldsymbol{r})) > 3$, $\mathrm{wt}(S_2(\boldsymbol{r})) > 3$, $\mathrm{wt}(S_1(\boldsymbol{r}) + B_i) \geq 3$ and $\mathrm{wt}(S_2(\boldsymbol{r}) + B_i) \geq 3$ for all $i = 1, \ldots, 12$), then request retransmission.

4.2.8 Project 8 [5pt]

Let C be an $\mathbb{F}_2 - [2^m, k, 2^{m-r}]$ Reed-Muller code, where $k = \sum_{i=0}^{r} \binom{m}{i}$. Let V the list of the elements of $\mathbb{F}_2^m$ sorted by lexicographic order, $K\langle x_1, \ldots, x_m\rangle$ the multivariate polynomial ring over $\mathbb{F}_2$ with m variables. Every vector $\boldsymbol{y} \in \mathbb{F}_2^{2^m}$ can be represented as

$$\boldsymbol{y} = (f(V_1), f(V_2), \ldots, f(V_{2^m}))$$

for a suitable $f \in K\langle x_1, \ldots, x_m\rangle$. Notice that f is of the form

$$f(x_1, \ldots, x_m) = \sum_{t=0}^{m} \sum_{S \subseteq \{1 \ldots m\}, |S|=t} f_S \prod_{i \in S} x_i \qquad \text{with } f_S \in \mathbb{F}_2 \text{ for all } S \subseteq \{1 \ldots m\}.$$

Implement the Majority Logic Decoder for Reed Muller codes.

INPUT:	The received vector $\boldsymbol{y}$, the parameters r, m of C, the list V defined above.
OUTPUT:	The codeword $\boldsymbol{c}$ obtained by decoding $\boldsymbol{y}$.

For a subset S of $\{1, \ldots, m\}$, a vector $\boldsymbol{a} \in \mathbb{F}_2^t$ and a vector $\boldsymbol{b} \in \mathbb{F}_2^{m-t}$, we define the vector $\boldsymbol{v}_{S,\boldsymbol{a},\boldsymbol{b}}$ of length m whose coordinates in S are given by $\boldsymbol{a}$ and the remaining by $\boldsymbol{b}$.

ALGORITHM:

```
1. Find the polynomial $f \in K\langle x_1, \ldots, x_m\rangle$ such that
       $\boldsymbol{y} = (f(V_1), f(V_2), \ldots, f(V_{2^m}))$
2. Initialize $p \in K\langle x_1, \ldots, x_m\rangle$ to be 0 and $t = r$.
3. Do the following for $t \geq 0$.
   (a) Set $f_t = f - p$
   (b) Do the following for every subset $S$ of $\{1, \ldots, m\}$ with $S = t$.
       i. Create an empty list $L_S$.
       ii. Do the following for every $\boldsymbol{b} \in \mathbb{F}_2^{m-t}$.
           • Compute the vector $\boldsymbol{v}_{S,\boldsymbol{a},\boldsymbol{b}}$.
           • Compute the value
                 $C_{S,\boldsymbol{b}} := \sum_{\boldsymbol{a} \in \mathbb{F}_2^T} f_t(\boldsymbol{v}_{S,\boldsymbol{a},\boldsymbol{b}})$.
           • Store the value $C_{S,\boldsymbol{b}}$ in the list $L_S$.
       iii. Compute the value $C_S$ as following. Set $C_S := 1$ if the number of 1s is greater
            or equal then the number of 0s in $L_S$, set $C_S := 0$ otherwise.
       iv. Set $p := p - C_S \prod_{i \in S} x_i$.
4. Return the vector $\boldsymbol{c} := (p(V_1), p(V_2), \ldots, p(V_m))$ if $\boldsymbol{c} \in C$. Ask for a retransmission
   otherwise.
```

4.2.9 Project 9 [5pt]

Let C be an $\mathbb{F}_{q^m} - [q^m - 1, k, d]$ Reed-Solomon code and let $\{1, \alpha, \alpha^2, \ldots, \alpha^{q^m-2}\}$ a set of evaluation point, where α is a primitive element of $\mathbb{F}_{q^m}/\mathbb{F}_q$. Let $\boldsymbol{r}$ a received vector, then we think of $\boldsymbol{y}$ as the set of ordered pairs $\{(1, r_1), (\alpha, r_2), \ldots, (\alpha^{q^m-2}, r_{q^m-1})\}$

Implement the Welch-Berlekamp algorithm for decoding Reed-Solomon codes, under the assumption that we know the weight of the error vector $\boldsymbol{e}$.

INPUT:	$\mathrm{wt}(e) < t$ and the ordered pairs $\{(\alpha^{i-1}, r_i\}_{i=1}^{q^m-1}$ associated to the received vector $\boldsymbol{r}$.
OUTPUT:	The codeword $\boldsymbol{c}$ obtained by decoding $\boldsymbol{r}$ or a request of retransmission.

<u>**ALGORITHM**</u>:

1. Compute the polynomial $E(x)$ of degree $\mathrm{wt}(\boldsymbol{e})$ and the polynomial $Q(x)$ of degree $\mathrm{wt}(\boldsymbol{e}) + k - 1$ such that
$$y_i E(\alpha_{i-1}) = Q(\alpha_{i-1})$$
for all $i = 1, \ldots, q^m - 1$.
2. If $E(x)$ and $Q(x)$ as above do not exist or $E(x)$ does not divide $Q(x)$, then ask for a retransmission.
3. If $E(x)$ and $Q(x)$ as above exist and $E(x)$ divides $Q(x)$, than set $P(x) := \frac{Q(x)}{E(x)}$.
4. Create the vector $\boldsymbol{p} := (P(1), P(\alpha), \ldots, P(\alpha^{q^m-1}))$.
5. If $d(\boldsymbol{y}, \boldsymbol{p}) \leq \mathrm{wt}(e)$ then return $\boldsymbol{p}$ as the decoded codeword. Otherwise, ask for a retransmission.

4.2.10 Project 10 [5pt]

Let C be an $\mathbb{F}_2 - [2^m - 1, k, d]$ BCH code and α be a primitive element of $\mathbb{F}_{2^m}/\mathbb{F}_2$, i.e. $\mathbb{F}[\alpha] = \mathbb{F}_{2^m}$. Define for every vector $\boldsymbol{v} \in \mathbb{F}_2^{2^m-1}$ the polynomial $f_{\boldsymbol{v}}(x) := \sum_{i=0}^{2^m-1} v_i x^i$. Suppose the received vector $\boldsymbol{r}$, then the syndrome vector if

$$\boldsymbol{s} := (f_{\boldsymbol{r}}(\alpha), f_{\boldsymbol{r}}(\alpha^2), \ldots, f_{\boldsymbol{r}}(\alpha^{2t}))$$

where t is the correction capability of C.

Implement a decoding algorithm for binary BCH codes based on the Berlekamp-Massey algorithm.

INPUT:	The received vector $\boldsymbol{r}$, the correction capability.
OUPUT:	A codeword $\boldsymbol{c}$ obtained by decoding $\boldsymbol{r}$ or a request for retransmission.

Given the syndrome vector $\boldsymbol{s}$, the Berlekamp-Massey algorithm finds the associated **locator polynomial**

$$c(x) := c_0 + c_1 x + \ldots + c_{2^m-1} x^{2^m-1} \in \mathbb{F}_{2^m}[x].$$

The roots of this polynomial give information about the location of errors. In particular, if α^i is a root for $c(x)$ then there is an error in $\boldsymbol{r}$ in position j where $\alpha^j = (\alpha^i)^{-1}$.

<u>**ALGORITHM**</u>:

1. Find the syndrome vector $\boldsymbol{s}$ associated to $\boldsymbol{r}$.

2. Initialize the following parameters:
   - $L = 0$, it represent the length of the LFSR;
   - $c(x) = 1$, it will be the locator polynomial;
   - $p(x) = 1$, it represent the locator polynomial before last length change;
   - $l = 1$, it represent the amount of shift in update;
   - $d_m = 1$, it represent the previous discrepancy.
3. Do the following for $k = 1, \dots, 2t$ in steps of 2 (i.e. $k = 1, 3, \dots$).
   (a) Compute the discrepancy
   $$d := \boldsymbol{s}_k + \sum_{i=1}^{L} c_i \boldsymbol{s}_{k-i}.$$
   (b) If $d = 0$ then increase the shift l by 1.
   (c) If $d \neq 0$ then
      i. If $2L \geq k$ then set $c(x) = c(x) - d d_m^{-1} x^l p(x)$ and increase the shift l by 1.
      ii. Otherwise, temporary store the polynomial $p(x)$, that is define $t(x) = c(x)$, set $c(x) = c(x) - d d_m^{-1} x^l p(x)$ and then $p(x) = t(x)$. Finally, set $L = k - L$, $d_m = d$ and $l = 1$.

   Increase the shift l by 1 and go back to point (3).
4. Find the multiplicative inverse of the roots of $c(x)$.
5. Compute the error vector and decode.
6. If the decoded vector is not a codeword of C, then ask for a retransmission.

Chapter 5: **Conclusions**

This initiative added a concrete dimension to strengthen the teaching of an abstract topic. It gave students the opportunity to gain experience in programming and to develop new skills that can be applied in different areas and in different projects, such as the final year project.

At the end of the semester, students were asked to fill a survey on the quality of the workshop and scope to give suggestions on how to improve it in future. Overall, 72% of participants recommended that the workshop should run again in the future. About half of the students agreed that they gained a better understanding of coding theory via using MAGMA . Some 90% of participants found the group project to be beneficial, while 82% agreed that it gave them an opportunity to explore a coding theoretic topic more deeply. Just 45.5% of students thought that sufficient time was dedicated to each project and the same percentage thought that the overall duration of the workshop was sufficient. Finally, 91% of participants agreed that the teaching in the workshop stimulated their interest in the subject matter. If this initiative was to run again, these suggestions would be taken into consideration and incorporated into its design.