Golay codes onderzocht

met behulp van

Gröbner bases

door Loes Paffen

Golay codes onderzocht

met behulp van

Gröbner bases

door Loes Paffen

29 augustus 2005
Scriptiebegeleider: W. Bosma
Radboud Universiteit Nijmegen
Faculteit Natuurwetenschappen, Wiskunde en Informatica
Subfaculteit Wiskunde
Toernooiveld 1
6525 ED Nijmegen

Voorwoord

In deze scriptie ga ik mij in de codetheorie verdiepen. Codetheorie houdt zich op meerdere manieren bezig met communicatie. Eén van deze manieren behelst het op een slimme manier versturen van boodschappen over een luidruchtig kanaal. Hier zal ik verder op ingaan.

Ik zal de Golay codes, $\mathbf{G}_{11}$ en $\mathbf{G}_{23}$, nader onderzoeken. Er is al veel bekend over deze codes. Ik zal echter uitgaan van minimale voorkennis en zelf op zoek gaan naar meer gegevens. Zo zal ik een algoritme geven om de minimumafstand van beide codes te bepalen. Dit algoritme is overigens bruikbaar voor alle cyclische codes. Met de gevonden minimumafstand van beide codes kan ik bewijzen dat de codes perfect zijn. Met deze extra gegevens (minimumafstand en daarmee bewijs dat de codes perfect zijn) ga ik twee decodeeralgoritmes implementeren. Beide decodeeralgoritmes zijn te gebruiken voor alle cyclische codes, alhoewel ik bij het tweede algoritme dan nog wel wat aanpassingen zou moeten maken. Tenslotte zal ik de functionaliteit en snelheid van de twee decodeeralgoritmes voor $\mathbf{G}_{11}$ en $\mathbf{G}_{23}$ vergelijken.

De definities en notaties die ik gebruik komen overeen met die uit het boek *Some tapas of computer algebra* [1]. Dit boek ligt tevens ten grondslag aan de methodes die ik in mijn scriptie beschrijf en gebruik (Hoofdstuk 10 en 11 en Project 7). Om deze methodes toe te passen zoals ik het wil heb ik flink wat aanpassingen moeten maken. Ook heb ik de theorie van de Gröbner bases bestudeerd. Het gebruik van Gröbner bases bleek namelijk erg handig in de verschillende algoritmes die ik heb geïmplementeerd. In elk algoritme moet namelijk een stelsel niet-lineaire vergelijkingen worden opgelost. Het stelsel als een ideaal in een polynoomring beschouwen en daar vervolgens de Gröbner basis van bepalen is een snelle en gemakkelijke manier om het stelsel op te lossen. Er kan namelijk gebruik worden gemaakt van enkele handige eigenschapopen van Gröbner bases.

Voor elk algoritme zal ik de theorie bespreken die nodig is bij die methode. Alle belangrijke stellingen die gebruikt worden zal ik bewijzen. Bij enkele minder belangrijke stellingen verwijs ik naar een boek waar je het bewijs kan vinden.

De computertaal, waarin ik mijn algoritmes heb geïmplementeerd is die van het computer algebra systeem *Magma* [4]. Grote programma's voeg ik aan de bijlagen toe. Daar zal ik ook input en output van enkele programma's afdrukken. De kortere programma's zal ik samen met input en output gewoon in de tekst voegen. Bij de output zal ik ook telkens de tijd afdrukken die het programma nodig had voor de berekeningen. Deze tijd is in seconden weergegeven.

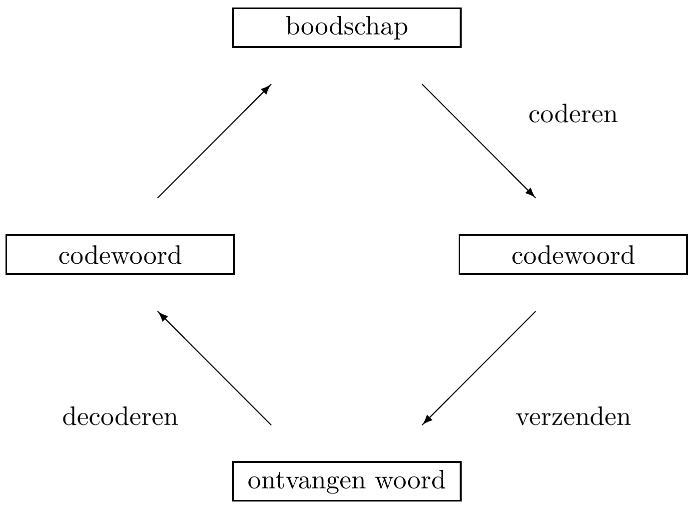
Tenslotte wens ik u veel plezier met het lezen van mijn scriptie.

Inhoudsopgave

1 Inleiding codetheorie

1.1 Foutcorrigerende codes

Ik zal mij in deze scriptie bezighouden met *foutcorrigerende codes*. Dit zijn codes die aan een boodschap op een slimme manier extra informatie toevoegen. Hierdoor kan na het verzenden van het codewoord de boodschap weer hervonden worden ook al is er tijdens het versturen iets niet helemaal goed doorgekomen. Ik bedoel daarmee dat als je bijvoorbeeld de boodschap $(1, 1, 0, 0)$ verzendt over een luidruchtig kanaal er een fout kan optreden. Zo kan het gebeuren dat een andere boodschap $(0, 1, 0, 0)$ aan het einde van het kanaal ontvangen wordt.



1.2 Basisgegevens coderen

De boodschap die verstuurd moet worden zal een woord van k letters over een alfabet van q elementen zijn. We zullen de boodschap noteren als een rijvector van lengte k over het eindige lichaam $\mathbf{F}_q$ met q elementen. Het *coderen* van de boodschap wordt gedaan door een matrixvermenigvuldiging van rechts:

$$M : \mathbf{F}_q^k \to \mathbf{F}_q^n,$$

$$M(w) = wG = c.$$

De afbeelding M voegt aan de boodschap $x \in \mathbf{F}_q^k$ het codewoord $c \in \mathbf{F}_q^n$ toe en wordt met betrekking tot de standaardbases van $\mathbf{F}_{q^k}$ en $\mathbf{F}_{q^n}$ door de $k \times n$-matrix G van rang k beschreven. Deze matrix wordt een *generatormatrix* van de code genoemd. Het is geen unieke matrix. Er zijn verschillende afbeeldingen M en dus verschillende matrices G die dezelfde lineaire deelruimte opspannen. De lineaire deelruimte van $\mathbf{F}_q^n$ die we met afbeelding M krijgen heet een *lineaire code van dimensie k en lengte n*. De codewoorden zijn rijvectoren van lengte n.

Om woorden met elkaar te vergelijken gebruiken we de *Hamming afstand*:

$$d(x, y) = \#\{i \mid x_i \neq y_i\},$$

voor $x, y \in \mathbf{F}_q^n$. De *minimumafstand* van een code C wordt als volgt gedefiniëerd:

$$d = d(C) = \min\{d(x, y) \mid x, y \in C,\ x \neq y\}.$$

Het *gewicht* van een codewoord c, genoteerd met $\mathrm{wt}(c)$, is het aantal coördinaten van de vector c die ongelijk aan 0 zijn. Het *minimumgewicht* van een code is het minimum van de gewichten van alle codewoorden behalve het codewoord dat gelijk is aan de nulvector. Omdat een code lineair is, is de minimumafstand gelijk aan het minimumgewicht:

$$d(C) = \min\{\mathrm{wt}(c) \mid c \in C,\ c \neq 0\}.$$

De parameters van een lineaire code C van dimensie k, lengte n en minimumafstand d over het lichaam $\mathbf{F}_q$ worden als volgt genoteerd: $[n, k, d]_q$.

Definitie 1 *Een $[n, k, d]_q$-code heet perfect als de Hamming bollen met straal $\frac{(d-1)}{2}$ en middelpunt een codewoord de vectorruimte $\mathbf{F}_q^n$ volledig overdekken.*

Zij $v \cdot w$ het standaard inproduct tussen de vectoren v en w. Voor de code C van lengte n over $\mathbf{F}_q$ definiëren we de *duale code*:

$$C^\perp = \{x \in \mathbf{F}_q^n \mid c \cdot x = 0 \text{ voor alle } c \in C\}.$$

In het bijzonder geldt dus: $(C^\perp)^\perp = C$.

Stelling 1 *Zij C een lineaire code van lengte n en dimensie k. Dan heeft $C^\perp$ dimensie $n - k$. Zij H de generatormatrix van code $C^\perp$. Dan:*

$$C = \{c \in \mathbf{F}_q^n \mid Hc^\top = 0\ \}.$$

Bewijs: Zij G de generatormatrix van C, waarbij de rijvector G_i de i-de rij van G is. Dan:

$$\begin{aligned} C &= \{aG \mid a \in \mathbf{F}_q^k\ \} \\ &= \{\lambda_1 G_1 + \ldots + \lambda_k G_k \mid \text{met } \lambda_i \in \mathbf{F}_q\}. \end{aligned}$$

Dus voor elke $c \in C$ zijn er $\lambda_1, \ldots, \lambda_k \in \mathbf{F}_q$ met: $c = \lambda_1 G_1 + \ldots + \lambda_k G_k$. Andersom is er voor alle $\lambda_1, \ldots, \lambda_k \in \mathbf{F}_q$ een $c \in C$ met:
$\lambda_1 G_1 + \ldots + \lambda_k G_k = c$. Dus:

$$\begin{aligned} C^\perp &= \{x \in \mathbf{F}_q^n \mid c \cdot x = 0 \text{ voor alle } c \in C\} \\ &= \{x \in \mathbf{F}_q^n \mid (\lambda_1 G_1 + \ldots + \lambda_k G_k) \cdot x = 0 \text{ voor alle } \lambda_i \in F_q\} \\ &= \{x \in \mathbf{F}_q^n \mid G_1 \cdot x = 0, \ldots, G_k \cdot x = 0\} \\ &= \{x \in \mathbf{F}_q^n \mid Gx^\top = 0\}. \end{aligned}$$

Aangezien G dimensie k heeft en $x \in \mathbf{F}_q^n$ weten we:

$$\dim(C^\perp) = \dim(\ker(G)) = n - \dim(G) = n - k.$$

Zoals we net lieten zien dat:

$$C^\perp = \{x \in \mathbf{F}_q^n \mid Gx^\top = 0\},$$

geldt natuurlijk ook:

$$C = \{c \in \mathbf{F}_q^n \mid Hc^\top = 0\}.$$

QED

Een matrix H waarvoor geldt $C = \{c \in \mathbf{F}_q^n \mid Hc^\top = 0\}$ wordt ook wel een *parity check* matrix van C genoemd.

1.3 Cyclische codes

Een bijzondere afdeling binnen de lineaire codes vormen de *cyclische codes*. Een code C is cyclisch als C voldoet aan de volgende eigenschap:

$$c = (c_0, c_1, \ldots, c_{n-1}) \in C \implies (c_{n-1}, c_0, \ldots, c_{n-2}) \in C.$$

Het is handig om de index i van een woord dus als een element van $\mathbf{Z}/n\mathbf{Z}$ te beschouwen. Verder zullen we de woorden van C uit $\mathbf{F}_q^n$ ook wel schrijven als polynomen uit $\mathbf{F}_q[X]/(X^n - 1)$. Dit kan met gebruik van de bijectie ϕ:

$$\begin{aligned} &\phi : \mathbf{F}_q^n \to \mathbf{F}_q[X]/(X^n - 1), \\ &\phi(c) = c_0 + c_1X + c_2X^2 + \ldots + c_{n-1}X^{n-1}. \end{aligned}$$

Stelling 2 *De bijectie ϕ induceert een bijectie tussen de idealen in de ring $\mathbf{F}_q[X]/(X^n - 1)$ enerzijds en de cyclische codes in $\mathbf{F}_q^n$ anderzijds.*

Bewijs: Ik kan bij dit bewijs volstaan door het volgende op te merken:
Voor een ideaal geldt:

$$\begin{aligned} &(c_0 + c_1X + c_2X^2 + \ldots + c_{n-1}X^{n-1}) \in I \iff \\ &(b_0 + b_1X + b_2X^2 + \ldots + b_{n-1}X^{n-1})(c_0 + c_1X + c_2X^2 + \ldots + c_{n-1}X^{n-1}) \in I \\ &\text{voor alle } b_0, b_1, \ldots, b_{n-1} \in \mathbf{F}_q. \end{aligned}$$

Nu gebeuren er eigenlijk herhaaldelijk drie dingen bij deze vermenigvuldiging:
Ten eerste wordt er met X vermenigvuldigd. Dit correspondeert met een cyclische opschuiving en het verkregen woord zit dus in de code.
Ten tweede wordt er met een element uit het lichaam $\mathbf{F}_q$ vermenigvuldigd. Dit correspondeert met de scalaire vermenigvuldiging binnen een lineaire code.
Ten slotte wordt er opgeteld. Dat correspondeert met optellen binnen een lineaire code. QED

Aangezien polynoomringen over lichamen zelfs Euclidische ringen zijn, is de ring $\mathbf{F}_q[X]/(X^n - 1)$ een hoofdideaalring. Hieruit volgt direct dat een cyclische code

C door één codewoord wordt voortgebracht. Het monische polynoom $g(X)$ dat correspondeert met het voortbrengende codewoord heet het *generatorpolynoom*. Het generatorpolynoom heeft maximaal graad $n-1$. Er geldt:

$$C = \{c(X) \mid c(X) = r(X)g(X) \bmod (X^n - 1),\ r(X) \in \mathbf{F}_q[X]\}.$$

We kunnen een cyclische code ook bestuderen aan de hand van de nulpunten van $g(X)$ in een uitbreidingslichaam van $\mathbf{F}_q$. Hiervoor hebben we echter nodig dat q en n relatief priem zijn. Dit veronderstellen we dus vanaf nu altijd. Zij $\mathbf{F}_{q^e}$ het kleinste uitbreidingslichaam van $\mathbf{F}_q$ dat een primitieve n-de eenheidswortel bevat. Hierbij is e de multiplicatieve orde van q in de ring $\mathbf{Z}/n\mathbf{Z}$. Dat wil zeggen: e is het kleinste getal met $e > 0$ en $q^e = 1 \bmod n$.

Zij nu α een primitieve n-de eenheidswortel in het uitbreidingslichaam $\mathbf{F}_{q^e}$. Dan heet $J \subseteq \mathbf{Z}/n\mathbf{Z}$ een *definiërende verzameling* voor C als:

$$C = \{c(X) \in \mathbf{F}_q[X]/(X^n - 1) \mid c(\alpha^j) = 0 \text{ voor alle } j \in J\}.$$

De *volledige definiërende verzameling* $J(C)$ is:

$$J(C) = \{j \in \mathbf{Z}/n\mathbf{Z} \mid c(\alpha^j) = 0 \text{ voor alle } c \in C\}.$$

Het is zo niet meteen duidelijk dat de (volledige) definiërende verzameling van een code C überhaubt bestaat. We gaan dit dus verder bekijken.

We gaan $X^n - 1$ ontbinden in irreducibele factoren over $\mathbf{F}_q$. Zij $X^n - 1 = f_1 \cdots f_r$ deze factorisatie in irreducibele factoren over $\mathbf{F}_q$. Omdat q en n relatief priem zijn, heeft $X^n - 1$ geen dubbele factoren en zijn alle f_i verschillend.
De factoren f_i zijn juist de minimumveeltermen van de machten α^j van α. Omdat het generatorpolynoom $g(X)$ een deler is van $X^n - 1$, is het een produkt van een aantal van de factoren f_i. De definiërende verzameling is nu die verzameling die voor iedere f_i die $X^n - 1$ deelt minstens een j bevat zodat α^j minimumveelterm f_i heeft. De volledige definiërende verzameling bevat alle j waarvoor α^j minimumveelterm f_i heeft voor alle factoren f_i die $g(X)$ delen. Hieruit volgt dat beide verzamelingen voor een code C bestaan.

Ter verduidelijking zal ik nog een triviaal voorbeeld bekijken. Zij $J = \{0\}$. De minimumveelterm van $\alpha^0 = 1$ is $X - 1$. Aangezien 0 het enige element van J is, is het generatorpolynoom gelijk aan deze minimumveelterm, $g(X) = X - 1$. Dus is code C een $(n-1)$-dimensionale code en is de duale code 1-dimensionaal: $C^\perp = < (1, \ldots, 1) >$.

Een stelling waar we veel gebruik van zullen maken is:

Stelling 3 *De minimumafstand van een cyclische code is minstens m als $J(C)$ $m-1$ opeenvolgende elementen bevat.*

Bewijs: Stel C is een cyclische code met $J(C) = \{j_1, \ldots, j_r\}$. Stel $J(C)$ heeft $m-1$ opeenvolgende elementen; zeg $i, i+1, \ldots, i+m-2$. Hierbij zijn $i, i+1, \ldots, i+m-2$ elementen in $\mathbf{Z}/n\mathbf{Z}$. Eerst zal ik de stelling bewijzen voor $i = 1$. Vervolgens zal ik het bewijs generaliseren voor i.

1. Zij $i = 1$. Zij α een primitieve n-de eenheidswortel in $\mathbf{F}_{q^e}$, dan geldt voor alle $j \in J(C)$:
$$c(\alpha^j) = 0 \text{ voor alle } c \in C.$$
Laten we nu matrix H bekijken:
$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \alpha^{2(2)} & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{(m-1)} & \alpha^{(m-1)2} & \dots & \alpha^{(m-1)(n-1)} \end{pmatrix}.$$
Voor deze matrix geldt dus:
$$cH^\top = 0 \text{ voor alle } c \in C.$$
Stel nu er is een $c \in C$ met $\mathrm{wt}(c) < m$. Het codewoord c heeft op minder dan m plaatsen een element ongelijk aan 0 staan. Zoals we zagen geldt: $cH^\top = 0$. Dat betekent dat een lineaire combinatie van $m-1$ rijen van $H^\top$ de nulvector oplevert. Dus er zijn $m-1$ kolommen in H die afhankelijk zijn. Laten we nu de matrix bekijken die bestaat uit deze $m-1$ kolommen:
$$\hat{H} = \begin{pmatrix} \alpha^{l_1} & \alpha^{l_2} & \dots & \alpha^{l_{m-1}} \\ \alpha^{2l_1} & \alpha^{2l_2} & \dots & \alpha^{2l_{m-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{(m-1)l_1} & \alpha^{(m-1)l_2} & \dots & \alpha^{(m-1)l_{m-1}} \end{pmatrix},$$
met $l_1, \dots, l_{m-1} \in \mathbf{Z}/n\mathbf{Z}$. De determinant van matrix $\hat{H}$ is gelijk aan de Vandermonde determinant:
$$\det(\hat{H}) = \prod_{0 \le k < i \le m-1} (\alpha^{l_i} - \alpha^{l_k}).$$
Aangezien $\mathbf{F}_{q^e}$ een lichaam is heeft het geen nuldelers. Dus is deze determinant alleen gelijk aan 0 als een van de termen van het produkt gelijk aan 0 is. Nu is α een primitieve n-de eenheidswortel en dus is geen van de produkttermen gelijk aan 0. De determinant is dus ongelijk aan 0 en de $m-1$ kolommen van $\hat{H}$ zijn onafhankelijk.
Tegenspraak!! Ik heb namelijk net verondersteld dat de $m-1$ kolommen van H, waarmee ik $\hat{H}$ heb gemaakt afhankelijk zijn. Dus:
$$\forall c \in C \backslash \{0\} \ : \ wt(c) \ge m.$$
Dus is de minimumafstand van C minstens m.

2. Zij i nu willekeurig. We gaan analoog te werk als voor $i = 1$. Zij α weer een primitieve n-de eenheidswortel in $\mathbf{F}_{q^e}$, dan geldt voor alle $j \in J(C)$:
$$c(\alpha^j) = 0 \text{ voor alle } c \in C.$$
Laten we nu matrix B bekijken:
$$B = \begin{pmatrix} 1 & \alpha^i & \alpha^{i2} & \dots & \alpha^{i(n-1)} \\ 1 & \alpha^{i+1} & \alpha^{(i+1)2} & \dots & \alpha^{(i+1)(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{(i+m-2)} & \alpha^{(i+m-2)2} & \dots & \alpha^{(i+m-2)(n-1)} \end{pmatrix}.$$

Voor deze matrix geldt dus:

$$cB^\top = 0 \text{ voor alle } c \in C.$$

Zij c weer een codewoord met gewicht kleiner dan m. Dan zijn er $m-1$ kolommen in B die afhankelijk zijn. Laten we nu de matrix bekijken die bestaat uit deze $m-1$ kolommen:

$$\hat{B} = \begin{pmatrix} \alpha^{il_1} & \alpha^{il_2} & \dots & \alpha^{il_{m-1}} \\ \alpha^{(i+1)l_1} & \alpha^{(i+1)l_2} & \dots & \alpha^{(i+1)l_{m-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{(i+m-2)l_1} & \alpha^{(i+m-2)l_2} & \dots & \alpha^{(i+m-2)l_{m-1}} \end{pmatrix},$$

met $l_1, \dots, l_{m-1} \in \mathbf{Z}/n\mathbf{Z}$. We gaan de determinant van $\hat{B}$ berekenen:

$$\begin{aligned} \det(\hat{B}) &= \det \begin{pmatrix} \alpha^{il_1} & \alpha^{il_2} & \dots & \alpha^{il_{m-1}} \\ \alpha^{(i+1)l_1} & \alpha^{(i+1)l_2} & \dots & \alpha^{(i+1)l_{m-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{(i+m-2)l_1} & \alpha^{(i+m-2)l_2} & \dots & \alpha^{(i+m-2)l_{m-1}} \end{pmatrix} \\ &= \alpha^{(i-1)l_1} \det \begin{pmatrix} \alpha^{l_1} & \alpha^{il_2} & \dots & \alpha^{il_{m-1}} \\ \alpha^{2l_1} & \alpha^{(i+1)l_2} & \dots & \alpha^{(i+1)l_{m-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{(m-1)l_1} & \alpha^{(i+m-2)l_2} & \dots & \alpha^{(i+m-2)l_{m-1}} \end{pmatrix}. \end{aligned}$$

Zo haal ik $m-1$ keer een constante $\alpha^{(i-1)l_t}$ naar voren en ik vind:

$$\det(\hat{B}) = \alpha^{(i-1)l_1} \cdots \alpha^{(i-1)l_{m-1}} \det \begin{pmatrix} \alpha^{l_1} & \dots & \alpha^{l_{m-1}} \\ \alpha^{2l_1} & \dots & \alpha^{2l_{m-1}} \\ \vdots & \ddots & \vdots \\ \alpha^{(m-1)l_1} & \dots & \alpha^{(m-1)l_{m-1}} \end{pmatrix}.$$

Dus:

$$\det(\hat{B}) = \alpha^{(i-1)l_1} \cdots \alpha^{(i-1)l_{m-1}} \det(\hat{H}) \neq 0.$$

Hieruit volgt dat de kolommen van $\hat{B}$ onafhankelijk zijn.
Tegenspraak!! Ik heb namelijk net verondersteld dat de $m-1$ kolommen van B, waarmee ik $\hat{B}$ heb gemaakt afhankelijk zijn. Dus:

$$\forall c \in C \backslash \{0\} \;\; : \;\; wt(c) \geq m.$$

Dus is de minimumafstand van C minstens m.

QED

Deze minimumgrens, het aantal opeenvolgende elementen in $J(C)$ vermeerderd met 1, voor de minimumafstand van een code wordt de *BCH grens* genoemd. Deze naam stamt af van de *BCH codes*. Dit is een belangrijke verzameling van codes die ontdekt werd door R.C. Bose, D.K. Ray-Chaudhuri en A. Hocquenghem. De BCH codes hebben per definitie als generatorpolynoom het kleinste gemene veelvoud van de minimumpolynomen van $\alpha^i, \ldots, \alpha^{i+m-2}$ voor een zekere i, m en met α een primitieve n-de eenheidswortel in een uitbreidingslichaam van $\mathbf{F}_q$. De BCH codes zijn dus zo gemaakt dat de minimumafstand minstens m is.

1.4 Basisgegevens decoderen

Bij het versturen van een codewoord kan een fout optreden. Zij c het codewoord dat verstuurd wordt en y het *ontvangen woord*. Dan is bij het zenden een *foutvector* e opgetreden met:

$$y = c + e.$$

Wij zullen gebruik maken van *minimumafstand decoderen*. Hierbij ga je ervan uit dat de kans dat er veel fouten in een ontvangen vector zitten kleiner is dan de kans dat er geen of weinig fouten in een ontvangen vector zitten. Tot een aantal van $t = \lfloor (d-1)/2 \rfloor$ fouten in een ontvangen vector kun je bepalen wat het codewoord was; namelijk het unieke codewoord met de kleinste afstand tot de ontvangen vector. Minimumafstand decoderen geeft bij een ontvangen vector met hoogstens t fouten het unieke dichtstbijzijnde codewoord als output. Als er meer dan t fouten in de ontvangen vector zitten zijn er twee mogelijkheden. Het kan zijn dat de ontvangen vector hoogstens t fouten verwijderd is van een codewoord c' dat ongelijk is aan het verzonden codewoord c. Dan geeft minimumafstand decoderen dit 'foute' codewoord c' als output. Als er geen codewoord met afstand $\leq t$ tot de ontvangen vector bestaat, dan wordt een foutmelding gegeven. Dit kan alleen maar gebeuren bij codes die niet perfect zijn.

Zij c een codewoord dat verzonden wordt en y de ontvangen vector. Zij nu e de bijbehorende foutvector met een gewicht dat maximaal gelijk is aan $\lfloor (d-1)/2 \rfloor$. Zij y' het gedecodeerde woord. Omdat c het unieke woord is met de kortste afstand tot y is het gedecodeerde woord y' dus gelijk aan c. De waarde $\lfloor (d-1)/2 \rfloor$ heet de *foutcorrigerende capaciteit* van de code. De twee decodeermethodes die ik ga bekijken kunnen foutvectoren tot een maximumgewicht van $\lfloor (d-1)/2 \rfloor$ corrigeren.

Zij H een parity check matrix van C. Dan weten we nog:

$$cH^\top = 0 \text{ voor alle } c \in C.$$

De *syndroomvector* is de vector s met:

$$s = yH^\top.$$

De componenten van de syndroomvector heten *syndromen*. Voor deze vector geldt ook:

$$s = yH^\top = (c+e)H^\top = cH^\top + eH^\top = eH^\top.$$

Het decodeerprobleem wordt dus vereenvoudigd tot het vinden van een foutvector e met minimaal gewicht bij een gegeven s. Daarbij is de volgende stelling nuttig:

Stelling 4 *Zij C een lineaire code in $\mathbf{F}_q^n$ met parity check matrix H en minimumafstand d. Zij y een ontvangen woord met foutvector e en J een verzameling met hoogstens $d-1$ elementen, die de verzameling van niet 0 posities van de foutvector omvat. Dan is de foutvector e de unieke oplossing voor x in het volgende stelsel lineaire vergelijkingen:*

$$xH^\top = yH^\top \quad \wedge \quad x_j = 0 \text{ voor alle } j \notin J$$

Bewijs: Het bewijs bestaat uit twee delen.
Ten eerste:
zij $x = e$, dan

$$xH^\top = eH^\top = eH^\top + cH^\top = (e+c)H^\top = yH^\top.$$

Ook bevat J alle posities, waarop e een element ongelijk aan 0 heeft, dus:

$$e_j = 0 \text{ voor alle } j \notin J.$$

Dus e is een oplossing van het stelsel lineaire vergelijkingen.
Ten tweede:
zij f ook een oplossing van het stelsel, dan

$$fH^\top = yH^\top = eH^\top.$$

Nu:

$$fH^\top = eH^\top \iff fH^\top - eH^\top = 0 \iff (f-e)H^\top = 0 \iff f-e \in C.$$

Aangezien $e_j = 0 \wedge f_j = 0$ voor alle $j \notin J$ geldt ook $(f-e)_j = 0$ voor alle $j \notin J$. Dus heeft $f-e$ op maximaal $d-1$ posities een element ongelijk aan 0 staan. Maar het minimumgewicht is d, dus:

$$f - e = 0 \ (\in C).$$

Dus $f = e$ en e is de unieke oplossing van het stelsel lineaire vergelijkingen.

QED

2 Gröbner bases

Ik zal gebruik maken van *Gröbner bases* om zekere codes te onderzoeken. Om te definiëren wat een Gröbner basis is en hoe je een Gröbner basis kan berekenen, zal ik eerst enkele begrippen invoeren (zie ook [1], Hoofdstuk 1).

Zij $\mathbf{F}$ een lichaam. We gaan de polynoomring $\mathbf{F}[X_1, \ldots, X_n]$ bekijken. Zij B een eindige deelverzameling van $\mathbf{F}[X_1, \ldots, X_n]$ en (B) het ideaal voortgebracht door B. Op de verzameling van monomen, $\mathbf{M}$, in $\mathbf{F}[X_1, \ldots, X_n]$ kunnen meerdere ordeningen worden aangebracht. Ik ga de *lexicografische ordening* $\prec_L$ gebruiken. Meestal wordt bij de lexicografische ordening gekozen voor deze ordening van de variabelen:

$$X_n \prec_L X_{n-1} \prec_L \cdots \prec_L X_2 \prec_L X_1.$$

Er geldt dan:

$$X_1^{a_1} X_2^{a_2} \cdots X_n^{a_n} \prec_L X_1^{b_1} X_2^{b_2} \cdots X_n^{b_n} \iff \begin{array}{l} \exists k \leq n, \text{ met} \\ a_k < b_k \text{ en } a_i = b_i \text{ voor alle } i < k. \end{array}$$

Ik zal bij toepassing van de lexicografische ordening niet altijd de gebruikelijke ordening der variabelen kiezen. Bij het berekenen van een Gröbner basis zal ik daarom altijd aangeven welke ordening van elementen ik gebruik.
We willen nu van een polynoom $f = \sum_{m \in \mathbf{M}} f_m m \in \mathbf{F}[X_1, \ldots, X_n]$ het grootste monoom en bijbehorende coëfficiënt in de gekozen ordening bepalen. Hiervoor definiëren we:

1. $\mathrm{lm}(f) = \max\{m \in \mathbf{M} | f_m \neq 0\}, \text{ met } \mathrm{lm}(0) = -\infty.$
2. $\mathrm{lc}(f) = \text{de coëfficiënt van } \mathrm{lm}(f), \text{ met } \mathrm{lc}(0) = 0.$
3. $\mathrm{lt}(f) = \mathrm{lc}(f)\mathrm{lm}(f).$

Om een Gröbner basis van (B) te berekenen gebruiken we een recursief algoritme, R, dat polynomen reduceert:

$R(B, f) =$

$J := \{b \in B| \ \mathrm{lm}(b) \text{ deelt } \mathrm{lm}(f)\};$
als $J \neq \emptyset$, dan kies $b \in J$; en geef $\mathrm{R}(B, f - (\frac{\mathrm{lt}(f)}{\mathrm{lt}(b)})b)$;
anders geef f;
einde.

Zolang $\mathrm{lm}(f)$ een veelvoud van een $\mathrm{lm}(b)$ is voor een $b \in B$, zoekt (en vindt) het algoritme een representant van de restklasse $f + (B)$, die een kleinere leading monoom heeft dan f. Dit algoritme werkt dus analoog aan het vegen in stelsels lineaire vergelijkingen. Het polynoom $R(B, f)$ heet een *gereduceerde vorm van* f mod B.

Ten slotte heb ik nog één ding nodig voordat ik het algoritme kan tonen, waarmee een Gröbner basis berekend wordt:

Het *S-polynoom* van f en g is:

$$S(f,g) = \frac{\mathrm{kgv}(\mathrm{lm}(f),\mathrm{lm}(g))}{\mathrm{lt}(f)} f - \frac{\mathrm{kgv}(\mathrm{lm}(f),\mathrm{lm}(g))}{\mathrm{lt}(g)} g.$$

Hierbij is *kgv* het *kleinste gemene veelvoud*.

Een klein voorbeeld om het idee achter S-polynomen te illustreren:
Zij $f = X^2Y + 1$ en $g = 2Y^3 - X$ met betrekking tot lexicografische ordening $X \prec_L Y$, dan is het S-polynoom:

$$S(f,g) = \frac{X^2Y^3}{X^2Y} f - \frac{X^2Y^3}{2Y^3} g = Y^2 f - \frac{1}{2}X^2 g = X^2Y^3 + 1 - X^2Y^3 + \frac{1}{2}X^3 = \frac{1}{2}X^3 + 1.$$

Je vermenigvuldigt f en g dus met monomen zodat de producten dezelfde leading monoom hebben en het verschil van de twee producten een kleinere leading monoom heeft dan $\mathrm{kgv}(\mathrm{lm}(f), \mathrm{lm}(g))$.

Definitie 2 *Een eindige deelverzameling B van $\mathbf{F}[X_1, \ldots, X_n]$ heet een Gröbner basis als voor elk polynoom $f \in (B)$ het nulpolynoom een gereduceerde vorm van $f \bmod B$ is. Zij $I = (B)$, dan heet B een Gröbner basis van I.*

We zijn nu klaar om het *Buchberger algoritme* te beschrijven. Dit algoritme rekent voor een verzameling B van polynomen een Gröbner basis van (B) uit. Het algoritme gebruikt het S-polynoom om nieuwe polynomen van (B) te vinden die $\mathrm{mod} B$ niet tot 0 reduceren. Als het gereduceerde resultaat ongelijk aan 0 is wordt het aan B toegevoegd. Het Buchberger algoritme:

Gröbner(B) =
  $P := \{\{f,g\}|\ \ f,g \in B\}$;
  zolang $P \neq 0$ doe:
  kies $\{f,g\} \in P$; $P := P\backslash\{f,g\}$; $c := \mathrm{R}(B, \mathrm{S}(f,g))$;
  als $c \neq 0$
  dan $B := B \cup \{c\}$; $P := P \cup \{\{b,c\}|\ \ b \in B\}$;
  einde; geef B.

De basis B' die met dit algoritme gevonden wordt is een Gröbner basis van (B) met $(B) = (B')$. Aangezien aan B alleen maar elementen van (B) worden toegevoegd om B' te vinden is het duidelijk dat $(B) = (B')$. De basis B' is een Gröbner basis omdat aan het einde van het algoritme aan het tweede punt van Stelling 5 (op pagina 10) wordt voldaan.

Het Buchberger algoritme heeft helaas een groot nadeel. Voordat de juiste basis gevonden is kunnen B en P namelijk explosief groeien. Bij elke stap, waarin het gereduceerde S-polynoom ongelijk aan 0 is wordt het toegevoegd aan B. Daarnaast worden aan P alle paren van het gereduceerde S-polynoom en een polynoom uit B toegevoegd. Deze explosie van gegevens binnen het algoritme kan tot problemen leiden. Hier zal ik in Hoofdstuk 4.3.2 nog op terug komen.

Stelling 5 *De volgende eigenschappen van een eindige deelverzameling B van $\mathbf{F}[X_1, \ldots, X_n]$ met betrekking tot een lexicografische ordening $\prec_L$ in $\mathbf{M}$ zijn equivalent:*

1. *B is een Gröbner basis.*
2. *$S(b,c)$ reduceert tot* $0 \bmod B$ *voor alle $b, c \in B$.*
3. $\{lm(f) \mid f \in (B)\}\mathbf{F}[X_1, \ldots, X_n] = \{lm(b) \mid b \in B\}\mathbf{F}[X_1, \ldots, X_n]$.
4. $\{lt(f) \mid f \in (B)\}\mathbf{F}[X_1, \ldots, X_n] = \{lt(b) \mid b \in B\}\mathbf{F}[X_1, \ldots, X_n]$.

Bewijs: Zie [1], pagina 11.

Een stelling die later erg nuttig zal zijn:

Stelling 6 *Zij $\prec_L$ de lexicografische ordening met $X_1 \prec_L \cdots \prec_L X_w$ en I een ideaal in $\mathbf{F}[X_1, X_2, \ldots, X_w]$. Zij G een Gröbner basis van I met betrekking tot $\prec_L$. Dan is $G \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$ een Gröbner basis van $I \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$.*

Bewijs: Ik ga eigenschap 4 van Stelling 5 (op pagina 10) gebruiken. Volgens deze eigenschap hoef ik alleen te bewijzen:

$$\begin{aligned}
&\forall f \in I \cap \mathbf{F}[X_1, X_2, \ldots, X_i] : \\
&\exists g \in G \cap \mathbf{F}[X_1, X_2, \ldots, X_i],\ \exists s \in \mathbf{F}[X_1, X_2, \ldots, X_i] \text{ met} : \\
&\mathrm{lt}(f) = \mathrm{lt}(g)s.
\end{aligned}$$

Zij $f \in I \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$. Omdat $f \in I$ zijn er $g \in G$ en $s \in \mathbf{F}[X_1, X_2, \ldots, X_w]$ met $\mathrm{lt}(f) = \mathrm{lt}(g)s$. Maar ook: $f \in \mathbf{F}[X_1, X_2, \ldots, X_i]$, dus $s \in \mathbf{F}[X_1, X_2, \ldots, X_i]$ en $\mathrm{lt}(g) \in \mathbf{F}[X_1, X_2, \ldots, X_i]$. Omdat $g \in G \cap \mathbf{F}[X_1, X_2, \ldots, X_w]$ en $X_1 \prec_L \cdots \prec_L X_i \prec_L \cdots \prec_L X_w$ geldt er: $g \in G \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$. Zo heb ik dus een g en s gevonden die aan de voorwaarden voldoen. Dus is $G \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$ een Gröbner basis van $I \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$. QED

Zij G de Gröbner basis van een ideaal I in $\mathbf{F}[X_1, X_2, \ldots, X_w]$ met betrekking tot lexicografische ordening $\prec_L$. Als $G \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$ een Gröbner basis is van $I \cap \mathbf{F}[X_1, X_2, \ldots, X_i]$ met betrekking tot $\prec_L$ dan noemen we zo'n ordening $\prec_L$ een *eliminatie ordening*.

Een Gröbner basis B kan verfijnd worden tot een *gereduceerde Gröbner basis* B'. Hiervoor geldt per definitie onder andere: $\mathrm{lc}(b') = 1$ voor alle $b' \in B'$. Ik ga niet verder in op de exacte definitie van zo'n gereduceerde basis en hoe je eraan komt; zie hiervoor [1], pagina 13. Wel wil ik opmerken dat als ik in mijn programma's het commando *GroebnerBasis* aanroep Magma de gereduceerde Gröbner basis berekent. Een erg mooie eigenschap van gereduceerde Gröbner bases is:

Stelling 7 *Voor een gegeven lexicografische ordening heeft ieder ideaal in $\mathbf{F}[X_1, \ldots, X_n]$ een unieke gereduceerde Gröbner basis.*

Bewijs: Zie [1], pagina 14.

Omdat het zo'n bijzondere eigenschap is wilde ik het even noemen, maar ik zal er verder geen gebruik van maken.

3 De Golay codes

De twee codes die ik ga onderzoeken behoren tot de mooiste objecten in de codetheorie; de *Golay codes.* De Golay codes werden in 1949 door M.E.J. Golay ontdekt. Voor deze codes gelden erg mooie en uitzonderlijke eigenschappen. In de theorie van dit hoofdstuk ga ik ervan uit dat de parameters van beide codes bekend zijn. Hiermee zal ik enkele eigenschappen van de codes bespreken en eventueel bewijzen. In de rest van mijn scriptie ga ik echter uit van zeer beperkte informatie over de twee codes. Van beide codes zijn dan alleen de volgende gegevens bekend: de lengte van de code, de definiërende verzameling en het lichaam waarover de code gedefiniëerd is.

3.1 De theorie

De eerste Golay code die ik bespreek is de tertiaire, cyclische code $\mathbf{G}_{11}$. Deze code bestaat en wordt gedefiniëerd door de generatormatrix die ik in Bijlage 1 op pagina 46 heb afgedrukt. De parameters van $\mathbf{G}_{11}$ zijn $[11, 6, 5]_3$. $\mathbf{G}_{11}$ is de unieke code met deze parameters.

Stelling 8 *De code* $\mathbf{G}_{11}$ *is perfect.*

Bewijs: De dimensie van de code is 6. Dus het totale aantal codewoorden is: 3^6. Het aantal vectoren over $\mathbf{F}_3$ binnen een bol met straal $\frac{(d-1)}{2} = 2$ is:

$$1 + 2\binom{11}{1} + 2 \cdot 2\binom{11}{2} = 243 = 3^5.$$

Dus overdekken alle bollen met straal 2 en middelpunt $c \in \mathbf{G}_{11}$ een verzameling met $3^6 \cdot 3^5 = 3^{11}$ vectoren. Nu is dit exact gelijk aan het aantal vectoren in de vectorruimte $\mathbf{F}_3^{11}$. QED

Er bestaat een unieke uitbreiding van $\mathbf{G}_{11}$. Dit is de Golay code $\mathbf{G}_{12}$ met parameters $[12, 6, 6]_3$.

De tweede Golay code die ik bespreek is de binaire, cyclische code $\mathbf{G}_{23}$. Ook deze code bestaat echt en wordt gedefiniëerd door de generatormatrix die ik in Bijlage 1 op pagina 46 heb afgedrukt. De parameters van $\mathbf{G}_{23}$ zijn $[23, 12, 7]_2$. Net zoals $\mathbf{G}_{11}$ is $\mathbf{G}_{23}$ de unieke code met deze parameters.

Stelling 9 *De code* $\mathbf{G}_{23}$ *is perfect.*

Bewijs: De dimensie van de binaire Golay code is 12. Dus het totale aantal codewoorden is: 2^{12}. Het aantal vectoren over $\mathbf{F}_2$ binnen een bol met straal $\frac{(d-1)}{2} = 3$ is:

$$1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2048 = 2^{11}.$$

Dus overdekken alle bollen met straal 3 en middelpunt $c \in \mathbf{G}_{23}$ een verzameling met $2^{12} \cdot 2^{11} = 2^{23}$ vectoren. Nu is dit exact gelijk aan het aantal vectoren in de vectorruimte $\mathbf{F}_2^{23}$. QED

Ook van de binaire Golay code bestaat een unieke uitbreiding. Dit is de Golay code $\mathbf{G}_{24}$ met parameters $[24, 12, 8]_3$.

Voor beide codes geldt nog een eigenschap. Voordat ik deze kan noemen heb ik echter wat extra informatie nodig.
Een *t-design* met parameters (v, k, λ) is een paar $(\mathbf{X}, \mathbf{B})$. Hierbij is $\mathbf{X}$ een verzameling van v elementen, die punten genoemd worden, en $\mathbf{B}$ een verzameling van deelverzamelingen van $\mathbf{X}$ met k elementen. Deze deelverzamelingen heten *blokken* en hebben de eigenschap dat elke deelverzameling van t punten van X in precies λ blokken voorkomt. Een t-design met $\lambda = 1$ heet een *Steiner systeem.* Voor Steiner systemen wordt de notatie $S(t, k, v)$ gebruikt.

Oorspronkelijk werden t-designs gebruikt bij experimenten in de landbouw. De naam design komt dan ook af van *experimental design.* Als je bijvoorbeeld het effect van verschillende typen mest op v soorten aardappels wilt testen, is het misschien te duur om elke soort mest op iedere soort aardappel te testen. In plaats daarvan neem je de blokken van een t-design en behandel je de aardappelen van een blok met dezelfde soort mest. Uit de eigenschappen van een t-design volgt dat de blokken aardappelen goed gebalanceerd zijn. Op de resultaten van het experiment kunnen nu dus goed statistische analyse-methodes worden toegepast.

Nu zijn we klaar voor de laatste eigenschap:

1. De rijtjes met niet 0 posities van de codewoorden met minimumgewicht van $\mathbf{G}_{11}$ vormen de blokken van een 4-design;
   het Steiner systeem $S(4, 5, 11)$.

2. De rijtjes met niet 0 posities van de codewoorden met minimumgewicht van $\mathbf{G}_{23}$ vormen de blokken van een 4-design;
   het Steiner systeem $S(4, 7, 23)$.

Deze eigenschap zal ik voor $\mathbf{G}_{11}$ in Hoofdstuk 4.3.1 in Stelling 15 (op pagina 24) bewijzen. Aangezien het bewijs voor $\mathbf{G}_{23}$ analoog gaat kom ik daar niet meer op terug.

Tenslotte geeft de volgende tabel aan van welke gewichten er codewoorden zijn en ook hoeveel codewoorden er van dat gewicht zijn (zie [3], Hoofdstuk 20):

Golay code $\mathbf{G}_{11}$		Golay code $\mathbf{G}_{23}$	
Gewicht	Aantal codewoorden	Gewicht	Aantal codewoorden
0	1	0	1
5	132	7	253
6	132	8	506
8	330	11	1288
9	110	12	1288
11	24	15	506
		16	253
		23	1

In de tabel valt meteen de symmetrie van de aantallen codewoorden van $\mathbf{G}_{23}$ op. Voor deze symmetrie is een simpele verklaring. Golay code $\mathbf{G}_{23}$ is gedefiniëerd over het lichaam $\mathbf{F}_2$ met 2 elementen 0 en 1. Het woord $(1, \ldots, 1)$ is een codewoord. Als je nu dit codewoord bij een codewoord met gewicht w optelt krijg je het complementaire codewoord met alle componenten 0 en 1 precies verwisseld. Het gewicht van dit complementaire codewoord is dus $n - w$. Zo krijg je dus bijvoorbeeld uit de 253 codewoorden van gewicht 7 net zo veel codewoorden van gewicht 16.

Ik heb deze tabel alleen ter informatie toegevoegd. Zoals ik al eerder zei maak ik zelf geen gebruik van de gegevens die ik in deze paragraaf heb besproken. Ik gebruik uitsluitend de volgende gegevens:

1. De lengte van code $\mathbf{G}_{11}$ is $n := 11$, de definiërende verzameling is $J := \{1\}$ en het lichaam waarover de code is gedefiniëerd is het lichaam met $q := 3$ elementen.
2. De lengte van code $\mathbf{G}_{23}$ is $n := 23$, de definiërende verzameling is $J := \{1\}$ en het lichaam waarover de code is gedefiniëerd is het lichaam met $q := 2$ elementen.

Tijdens mijn onderzoek zal ik zelf steeds meer gegevens van de twee codes ontdekken. De extra gegevens die ik zelf vind zal ik ook gebruiken om verder te komen en nog meer te ontdekken.

3.2 Het gegevensalgoritme

Ik begin mijn onderzoek naar de Golay codes met het bepalen van enkele basisgegevens. Daarvoor heb ik het *gegevensalgoritme* geschreven. Dit algoritme krijgt als input slechts de lengte, het aantal elementen van het lichaam en de definiërende verzameling van de code. Het gegevensalgoritme geeft als output achtereenvolgens: het kleinste uitbreidingslichaam, het generatorpolynoom, de volledige definiërende verzameling en de BCH grens van de code. Het programma is overigens te gebruiken voor alle cyclische codes.

Ik heb het programma hierna afgedrukt. Aangezien het programma vrij simpel is en gebaseerd is op theorie die ik hiervoor reeds heb besproken geef ik geen verdere uitleg. Ik laat wel voor beide Golay codes de input en de output zien.

```
t:=Cputime(0.00);

//Het kleinste uitbreidingslichaam bepalen;
qe:=q;e:=1;
while qe mod n ne 1 do
        qe:=qe*q; e:=e+1;
end while;
Fqe<b>:=GF(qe);
"\n";"\n";
"Het kleinste uitbreidingslichaam is:";
Fqe;"\n";

//Uiteenvallen X^n-1 bekijken:
m:=(#(Fqe)-1) div n;
a:=b^m;
F<X>:=PolynomialRing(GF(q));
fac:=Factorization(X^n-1);
flag:=false;GenFacRij:=[];H:=J;
while flag eq false do
        j:=Random(H);H:=Exclude(H,j);i:=1;
        while Evaluate(fac[i][1],a^j) ne 0 do
                i:=i+1;
        end while;
        if fac[i][1] notin GenFacRij then
                GenFacRij:=GenFacRij cat [fac[i][1]];
        end if;
        if #H eq 0 then
                flag:=true;
        end if;
end while;
Gen:=1;
for f in GenFacRij do
        Gen:=Gen*f;
end for;
"Het generatorpolynoom is:";
Gen;"\n";

//Hiermee de volledige definierende verzameling bepalen:
JC:=J;
for k in [1..n] do
        if Evaluate(Gen,a^k) eq 0 then
                JC:=Include(JC,k);
        end if;
end for;
"De volledige definierende verzameling is:";
JC;"\n";
```

```
//Hieruit de BCH grens bepalen:
JC:=Sort(SetToSequence(JC));
teller:=1;langstesliert:=1;
for i in [1..(#(JC)-1)] do
        if (JC[i+1]-JC[i]) eq 1 then
                teller:=teller+1;
        else
                if teller ge langstesliert then
                        langstesliert:=teller;
                end if;
                teller:=1;
        end if;
end for;
BCH:=langstesliert+1;
JC:=SequenceToSet(JC);
"De BCH grens is:";
BCH;"\n";

"De totale rekentijd van het programma is:";
print Cputime(t);
```

3.2.1 Gegevensalgoritme voor $\mathbf{G}_{11}$

INPUT

$$n := 11;\ q := 3;\ J := \{1\};$$

OUTPUT

Het kleinste uitbreidingslichaam is: Finite field of size 3^5.
Het generatorpolynoom is: $X^5 + X^4 + 2X^3 + X^2 + 2$.
De volledige definiërende verzameling is: $\{1, 3, 4, 5, 9\}$.
De BCH grens is: 4.
De totale rekentijd van het programma is: 0.050.

3.2.2 Gegevensalgoritme voor $\mathbf{G}_{23}$

INPUT

$$n := 23;\ q := 2;\ J := \{1\};$$

OUTPUT

Het kleinste uitbreidingslichaam is: Finite field of size 2^{11}.
Het generatorpolynoom is: $X^{11} + X^9 + X^7 + X^6 + X^5 + X + 1$.
De volledige definiërende verzameling is: $\{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$.
De BCH grens is: 5.
De totale rekentijd van het programma is: 0.030.

4 Codewoorden van minimumgewicht

We hebben zojuist enkele gegevens over de codes gevonden. In dit deel zal ik een methode bespreken om ook de minimumafstand van de codes te bepalen en alle codewoorden van het overeenkomende minimumgewicht.

4.1 De theorie

Zij $c(X)$ een woord in $\mathbf{F}_q[X]/(X^n-1)$ van een cyclische code C. Zij $\alpha \in \mathbf{F}_{q^e}$ een primitieve n-de eenheidswortel. Dan wordt het *Mattson-Solomon* polynoom $A \in \mathbf{F}_{q^e}[Z]$ van $c(X)$ gedefiniëerd als volgt:

$$A(Z) = \sum_{i=1}^{n} A_i Z^{n-i} \text{ met } A_i = c(\alpha^i) \in \mathbf{F}_{q^e}.$$

Ook hier is het handig om de index i als een element van $\mathbf{Z}/n\mathbf{Z}$ te beschouwen.

Lemma 10 *Zij $\beta \in \mathbf{F}_{q^e}$ een nulpunt van $X^n - 1$. Dan:*

$$\sum_{i=0}^{n-1} \beta^i = \begin{cases} n & \textit{als } \beta = 1, \\ 0 & \textit{als } \beta \neq 1. \end{cases}$$

Bewijs: Als $\beta = 1$, dan:

$$\sum_{i=0}^{n-1} \beta^i = 1 + \ldots + 1 = n.$$

Als $\beta \neq 1$, dan:

$$\sum_{i=0}^{n-1} \beta^i = 1 + \beta + \ldots + \beta^{n-1} = \frac{\beta^n - 1}{\beta - 1} = 0.$$

QED

Stelling 11 *Zij C een cyclische code over $\mathbf{F}_q$ en α een primitieve n-de eenheidswortel in $\mathbf{F}_{q^e}$.*

1. *Als $A \in \mathbf{F}_{q^e}[Z]$ het Mattson-Solomon polynoom van een codewoord c is, dan geldt: $c_j = \frac{1}{n}A(\alpha^j)$, voor $j \in \mathbf{Z}/n\mathbf{Z}$.*
2. *Zij $A \in \mathbf{F}_{q^e}[Z]$. Dan: A is het Mattson-Solomon polynoom van een woord $c \in \mathbf{F}_q[X]/(X^n-1)$ van de cyclische code $C \iff \forall j \in \mathbf{Z}/n\mathbf{Z} : \ A_{jq} = A_j^q$ en $\forall j \in J(C) : \ A_j = 0$.*

Bewijs:

1. Voor $A(\alpha^j)$ geldt per definitie:

$$A(\alpha^j) = c(\alpha)(\alpha^j)^{n-1} + c(\alpha^2)(\alpha^j)^{n-2} + \ldots + c(\alpha^{n-1})(\alpha^j) + c(1).$$

Met

$$\begin{array}{lcl} c(\alpha)(\alpha^j)^{n-1} & = & c(\alpha)\alpha^{-j} \\ c(\alpha^2)(\alpha^j)^{n-2} & = & c(\alpha^2)(\alpha^2)^{-j} \\ \vdots & & \vdots \\ c(\alpha^{n-1})(\alpha^j) & = & c(\alpha^{n-1})(\alpha^{n-1})^{-j} \\ c(1) & = & c_0 + \ldots + c_{n-1}, \end{array}$$

en dus

$$\begin{array}{lcccccccc} c(\alpha)(\alpha^j)^{n-1} & = & c_j & + & c_{j+1}\alpha & + & \ldots & + & c_{j-1}\alpha^{n-1} \\ c(\alpha^2)(\alpha^j)^{n-2} & = & c_j & + & c_{j+1}\alpha^2 & + & \ldots & + & c_{j-1}\alpha^{2(n-1)} \\ \vdots & & \vdots & & \vdots & & & & \vdots \\ c(\alpha^{n-1})(\alpha^j) & = & c_j & + & c_{j+1}\alpha^{n-1} & + & \ldots & + & c_{j-1}\alpha^{(n-1)(n-1)} \\ c(1) & = & c_j & + & c_{j+1} & + & \ldots & + & c_{j-1}. \end{array}$$

Met behulp van Lemma 10 (op pagina 17) concluderen we:

$$A(\alpha^j) = c_j n + c_{j+1}\sum_{i=0}^{n-1}(\alpha)^i + \ldots + c_{j-1}\sum_{i=0}^{n-1}(\alpha^{n-1})^i = c_j n.$$

Inderdaad: $c_j = \frac{1}{n}A(\alpha^j)$.

2. $\Longrightarrow$ Zij $A(Z)$ het Mattson-Solomon polynoom van een codewoord $c \in \mathbf{F}_q[X]/(X^n - 1)$ van de cyclische code C, dan:

$$A(Z) = \sum_{i=1}^{n} A_i Z^{n-i} \text{ met } A_i = c(\alpha^i) \in \mathbf{F}_{q^e}.$$

Dus $\forall j \in \mathbf{Z}/n\mathbf{Z}$:

$$A_j = c(\alpha^j) = c_0 + c_1\alpha^j + \ldots + c_{n-1}(\alpha^j)^{n-1},$$

en

$$\begin{array}{rcl} (A_j)^q & = & {c_0}^q + (c_1\alpha^j)^q + \ldots + (c_{n-1}(\alpha^j)^{n-1})^q + q\{\ldots\ldots\} \\ & = & c_0 + c_1\alpha^{jq} + \ldots + c_{n-1}(\alpha^{jq})^{n-1} \\ & = & A_{jq}. \end{array}$$

Uit de definitie van volledige definiërende verzameling volgt direct:

$$\forall j \in J(C): \ A_j = c(\alpha^j) = 0.$$

$\Longleftarrow$ Zij A een polynoom in $\mathbf{F}_{q^e}[Z]$ met:

$$\forall j \in \mathbf{Z}/n\mathbf{Z}: \ A_{jq} = A_j^q \text{ en } \forall j \in J(C): \ A_j = 0.$$

Dan gaan we het polynoom $P(Z)$ maken met

$$P(Z) = \sum_{i=1}^{n} A_i Z^{n-i}.$$

Hierbij maken we een woord $c(X) \in \mathbf{F}_{q^e}[X]/(X^n - 1)$ met

$$c_j = \frac{1}{n} P(\alpha^j).$$

Voor dit woord moeten we nu nog twee dingen bewijzen:

(a) Het Mattson-Solomon polynoom van het woord $c(X)$ is gelijk aan polynoom $P(Z)$.

(b) Het woord $c(X)$ zit in $\mathbf{F}_q[X]/(X^n - 1)$.

Dat doen we als volgt:

(a) Zij $M(Z)$ het Mattson-Solomon polynoom van $c(X)$,

$$M(Z) = \sum_{i=1}^{n} B_i Z^{n-i}, \text{ met } B_i = c(\alpha^i) \in \mathbf{F}_{q^e}.$$

Nu gaan we $c(\alpha^i)$ beter bekijken:

$$\begin{array}{rcccccccc} c(\alpha^i) & = & c_0 & + & c_1(\alpha^i) & + & \ldots & + & c_{n-1}(\alpha^i)^{n-1} \\ & = & \frac{1}{n}P(1) & + & \frac{1}{n}P(\alpha)(\alpha^i) & + & \ldots & + & \frac{1}{n}P(\alpha^{n-1})(\alpha^i)^{n-1}, \end{array}$$

en dat is gelijk aan

$$\begin{array}{rllllllll} = & \frac{1}{n} & (A_1 & + & \ldots & + & A_{n-1} & + & A_n)+ \\ & \frac{1}{n} & (A_1\alpha^{n-1} & + & \ldots & + & A_{n-1}\alpha & + & A_n)\alpha^i+ \\ & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ & \frac{1}{n} & (A_1(\alpha^{n-1})^{n-1} & + & \ldots & + & A_{n-1}(\alpha^{n-1}) & + & A_n)(\alpha^i)^{n-1} \end{array}$$

$$\begin{array}{rll} = & \frac{1}{n} & (A_1 + \ldots + A_i + \ldots + A_{n-1} + A_n + \\ & & A_1\alpha^{i-1} + \ldots + A_i + \ldots + A_{n-1}\alpha^{i+1} + A_n\alpha^i + \ldots \\ & & \\ & \ldots & A_1(\alpha^{i-1})^{n-1} + \ldots + A_i + \ldots + A_{n-1}(\alpha^{i+1})^{n-1} + A_n(\alpha^i)^{n-1}). \end{array}$$

Met Lemma 10 (op pagina 17) weten we nu:

$$\forall i \in \mathbf{Z}/n\mathbf{Z} : \; c(\alpha^i) = \frac{1}{n}(nA_i) = A_i.$$

Dus $M(Z) = \sum_{i=1}^{n} c(\alpha^i) Z^{n-i} = \sum_{i=1}^{n} A_i Z^{n-i} = P(Z)$ is het Mattson-Solomon polynoom van woord $c(X)$.

(b) We weten $\forall j \in \mathbf{Z}/n\mathbf{Z} : \; A_{jq} = A_j^q$. Nu:

$$\begin{array}{rcccccccc} A_j & = & c_0 & + & c_1(\alpha^j) & + & \ldots & + & c_{n-1}(\alpha^j)^{n-1}. \\ & & & & & & & & \\ A_j^q & = & c_0^q & + & c_1^q(\alpha^{jq}) & + & \ldots & + & c_{n-1}^q(\alpha^{jq})^{n-1} + q\{\ldots\ldots\} \\ & = & c_0^q & + & c_1^q(\alpha^{jq}) & + & \ldots & + & c_{n-1}^q(\alpha^{jq})^{n-1}. \\ & & & & & & & & \\ A_{jq} & = & c_0 & + & c_1(\alpha^{jq}) & + & \ldots & + & c_{n-1}(\alpha^{jq})^{n-1}. \end{array}$$

Nu gaan we A_j^q sommeren over $j \in \mathbf{Z}/n\mathbf{Z}$.

$$\begin{array}{rcccccccc} \sum_j A_j^q & = & c_0^q & + & c_1^q(\alpha^q) & + & \dots & + & c_{n-1}^q(\alpha^q)^{n-1}+ \\ & & c_0^q & + & c_1^q(\alpha^{2q}) & + & \dots & + & c_{n-1}^q(\alpha^{2q})^{n-1}+ \\ & & \vdots & & \vdots & & \vdots & & \vdots \\ & & c_0^q & + & c_1^q(\alpha^{nq}) & + & \dots & + & c_{n-1}^q(\alpha^{nq})^{n-1}. \end{array}$$

Aangezien α^q een nulpunt is van $X^n - 1$ kunnen we weer Lemma 10 (op pagina 17) toepassen.

$$\sum_{j \in \mathbf{Z}/n\mathbf{Z}} A_j^q = nc_0^q.$$

Op analoge wijze vind ik:

$$\sum_{j \in \mathbf{Z}/n\mathbf{Z}} A_{jq} = nc_0.$$

Omdat $A_{jq} = A_j^q$ voor alle $j \in \mathbf{Z}/n\mathbf{Z}$ volgt hieruit dat

$$c_0^q = c_0.$$

Als we nu voor het sommeren zowel A_{jq} als A_j^q met α^i vermenigvuldigen voor zekere $i \in \mathbf{Z}/n\mathbf{Z}$, dan vinden we $c_i^q = c_i$. Dus:

$$c_0^q = c_0,\ c_1^q = c_1, \dots,\ c_{n-1}^q = c_{n-1}.$$

We weten dat elementen van $\mathbf{F}_q$ die elementen in $\mathbf{F}_{q^e}$ zijn die vastblijven onder het Frobenius automorfisme: $x \to x^q$. Hieruit volgt dat $c_0, \dots, c_{n-1} \in \mathbf{F}_q$ en dat $c(X)$ dus inderdaad een woord in $\mathbf{F}_q[X]/(X^n - 1)$ is.

Tenslotte weten we nog $\forall\ j \in J(C) :\ A_j = c(\alpha^j) = 0$, dus is $c(X)$ een woord in de cyclische code C met volledige definiërende verzameling $J(C)$. Dus A is inderdaad het Mattson-Solomon polynoom van een woord van de cyclische code C.

QED

Zij $c(X)$ een woord van gewicht w. De *locators* $x_1, x_2, \dots, x_w$ van $c(X)$ definiëren we als

$$\{x_1, x_2, \dots, x_w\} = \{\alpha^l \mid c_l \neq 0\}.$$

Met $y_j = c_l$ voor $x_j = \alpha^l$ geeft dat:

$$A_i = c(\alpha^i) = \sum_{j=1}^{w} y_j {x_j}^i.$$

Het polynoom σ heeft per definitie als nulpunten de inversen van de locators van $c(X)$:

$$\sigma(Z) = \prod_{j=1}^{w} (1 - x_j Z) \in \mathbf{F}_{q^e}[Z].$$

We kunnen dit polynoom herschrijven tot:

$$\sigma(Z) = \sum_{i=0}^{w} \sigma_i Z^i,$$

met

$$\sigma_t = (-1)^t \sum_{1 \le j_1 < j_2 < \ldots < j_t \le w} x_{j_1} x_{j_2} \cdots x_{j_t}.$$

Stelling 12 *Voor alle i geldt:*

$$A_{i+w} + \sigma_1 A_{i+w-1} + \ldots + \sigma_w A_i = 0, \text{ met indices } \bmod n.$$

Bewijs: We bekijken de gelijkheid:

$$1 + \sigma_1 Z + \ldots + \sigma_w Z^w = \prod_{j=1}^{w} (1 - x_j Z).$$

Als we voor Z nu $\frac{1}{x_j}$ invullen krijgen we

$$1 + \sigma_1 (\frac{1}{x_j}) + \ldots + \sigma_w (\frac{1}{x_j})^w = 0.$$

Vervolgens vermenigvuldigen we met $y_j {x_j}^{i+w}$:

$$y_j {x_j}^{i+w} + \sigma_1 y_j {x_j}^{i+w-1} + \ldots + \sigma_w y_j {x_j}^i = 0.$$

Tenslotte sommeren we over $j = 1, \ldots, w$:

$$\begin{aligned} 0 &= \textstyle\sum_{j=1}^{w} (y_j {x_j}^{i+w} + \sigma_1 y_j {x_j}^{i+w-1} + \ldots + \sigma_w y_j {x_j}^i) \\ &= \textstyle\sum_{j=1}^{w} y_j {x_j}^{i+w} + \sigma_1 \sum_{j=1}^{w} y_j {x_j}^{i+w-1} + \ldots + \sigma_w \sum_{j=1}^{w} y_j {x_j}^i \\ &= A_{i+w} + \sigma_1 A_{i+w-1} + \ldots + \sigma_w A_i. \end{aligned}$$

QED

We zijn nu klaar om een stelsel van vergelijkingen te definiëren.
Zij C een cyclische code van lengte n over $\mathbf{F}_q$ met volledige definiërende verzameling $J(C)$. Zij $\mathbf{F}_{q^e}$ het kleinste uitbreidingslichaam van $\mathbf{F}_q$ dat een primitieve n-de eenheidswortel α bevat. We gaan het stelsel van vergelijkingen $S_C(w)$ over de polynoomring $\mathbf{F}_q[a_1, \ldots, a_n, \tau_1, \ldots, \tau_w]$ bekijken. Dit stelsel bestaat uit $2n + \#J(C)$ vergelijkingen in de $n + w$ onbekenden $a_1, \ldots, a_n$ en $\tau_1, \ldots, \tau_w$. Hierbij zijn de indices i van a_i elementen van $\mathbf{Z}/n\mathbf{Z}$ en is $w \in \{1, \ldots, n\}$ de variabele die ingevoerd wordt in het stelsel.

$$S_C(w) = \left\{ \begin{array}{lllllll} a_{w+1} & + & \tau_1 a_w & + & \ldots & + \ \tau_w a_1 & = 0 \\ a_{w+2} & + & \tau_1 a_{w+1} & + & \ldots & + \ \tau_w a_2 & = 0 \\ \vdots & & \vdots & & \vdots & \vdots & \vdots \\ a_{w+n} & + & \tau_1 a_{w+n-1} & + & \ldots & + \ \tau_w a_n & = 0 \\ & & & & \forall j \in J(C): & a_j & = 0 \\ & & & & \forall j \in \mathbf{Z}/n\mathbf{Z}: & a_{qj} & = {a_j}^q. \end{array} \right.$$

We gaan dit stelsel nu voor een zekere $w \in \{1, \ldots, n\}$ oplossen over het lichaam $\mathbf{F}_{q^e}$.

Stelling 13 *Stel we vinden oplossingen* $(A_1, \ldots, A_n, \sigma_1, \ldots, \sigma_w)$ *voor* $(a_1, \ldots, a_n, \tau_1, \ldots, \tau_w)$ *van* $S_C(w)$ *over het lichaam* $\mathbf{F}_{q^e}$. *Dan bepaalt ieder rijtje* $(A_1, \ldots, A_n)$ *het Mattson-Solomon polynoom van een codewoord van gewicht maximaal* w. *Het rijtje* $(\sigma_1, \ldots, \sigma_w)$ *bepaalt het bijbehorende polynoom* σ, *waarvan de nulpunten de inversen van de locators van het codewoord zijn.*

Bewijs: Dat ieder rijtje $(A_1, \ldots, A_n)$ het Mattson-Solomon polynoom van een codewoord van gewicht maximaal w bepaalt is een direct gevolg van Stelling 11 (op pagina 17) en Stelling 12 (op pagina 21). Voor het bewijs dat het rijtje $(\sigma_1, \ldots, \sigma_w)$ het bijbehorende polynoom σ bepaalt, verwijs ik u naar pagina 245 van [3]. QED

Stelling 14 *De minimumafstand* d *van code* C *is gelijk aan de kleinste waarde* w *waarvoor het stelsel* $S_C(w)$ *een oplossing over* $\mathbf{F}_{q^e}$ *heeft die ongelijk aan* 0 *is.*

Bewijs: Dit is wederom een direct gevolg van Stelling 11 en Stelling 12. QED

De codewoorden van minimumgewicht van een cyclische code kunnen dus worden vastgesteld door het oplossen van een stelsel vergelijkingen over de polynoomring $\mathbf{F}_q[a_1, \ldots, a_n, \tau_1, \ldots, \tau_w]$. Er zijn meerdere manieren om het stelsel op te lossen. Ik zal het stelsel oplossen door een Gröbner basis van het ideaal dat door $S_C(w)$ wordt voortgebracht te berekenen.

4.2 Het codewoordenalgoritme

Het algoritme dat ik heb geschreven maakt gebruik van de theorie in de vorige paragraaf. In Bijlage 2 op pagina 47 is het codewoordenalgoritme afgedrukt. Ik zal het algoritme hieronder in stappen beschrijven. Hiervoor wil ik nog enkele dingen opmerken:

1. Zoals eerder gezien geldt voor de BCH grens van een code dat de minimumafstand groter of gelijk aan deze grens is. Daarom heeft het algoritme een startwaarde w die gelijk is aan de BCH grens.

2. Om het stelsel $S_C(w)$ op te lossen maak ik eerst het ideaal $I \subset \mathbf{F}_q[a_1, \ldots, a_n, \tau_1, \ldots, \tau_w]$ voortgebracht door stelsel $S_C(w)$. Vervolgens bereken ik de Gröbner basis G van het ideaal I. Een Gröbner basis G heet *goed* als G oplossingen over $\mathbf{F}_{q^e}$ heeft ongelijk aan 0. Dan heeft het stelsel namelijk ook oplossingen over $\mathbf{F}_{q^e}$ ongelijk aan 0. Om te kijken of G oplossingen over $\mathbf{F}_{q^e}$ heeft die ongelijk aan 0 zijn licht ik het laatste element van G eruit. Ik check of dit element een polynoom is in één variabele en kijk vervolgens of het polynoom nulpunten ongelijk aan 0 heeft. Als dat zo is heeft G en dus ook het stelsel oplossingen ongelijk aan 0.

3. De lexicografische ordening, $\prec_L$, die ik gebruik om de Gröbner basis te berekenen, heeft de volgende ordening der elementen:

$$a_1 \prec_L \cdots \prec_L a_n \prec_L \tau_1 \prec_L \cdots \prec_L \tau_w.$$

Het codewoordenalgoritme:

INPUT

$n :=$; $q :=$; $J := \{\ \}$; load "gegevenscode"; $w :=$ BCH;

Stap 1: Maak stelsel $S_C(w)$.
OUTPUT $S_C(w)$

Stap 2: Maak het ideaal dat wordt voortgebracht door dit stelsel.

Stap 3: Bereken de Gröbnerbasis G van dat ideaal.
OUTPUT G

Stap 4: Als G geen goede Gröbner basis is doe $w := w + 1$ en herhaal Stap 1 tot en met Stap 4.

Stap 5: Bereken het aantal oplossingen, p, dat G heeft.
OUTPUT De minimumafstand w en het aantal woorden, p, van overeenkomend minimumgewicht.

Stap 6: Bereken uit G oplossingsrijtjes $(\sigma_w, \ldots, \sigma_1, A_n, \ldots, A_1)$.

Stap 7: Bereken met de oplossingsrijtjes polynoom $\sigma(Z)$, vervolgens de locators en hieruit de niet 0 posities van alle woorden.
OUTPUT De verzameling met rijtjes van niet 0 posities van de codewoorden van minimumgewicht.

Stap 8: Bereken met de oplossingsrijtjes het Mattson-Solomon polynoom en vervolgens alle codewoorden.
OUTPUT Alle codewoorden van minimumgewicht.

Nadat een goede Gröbner basis G en bijbehorende oplossingsrijtjes zijn gevonden doet mijn algoritme twee verschillende dingen.
In Stap 7 wordt per oplossingsrijtje het polynoom $\sigma(Z)$ berekend. Met deze polynomen wordt vervolgens een verzameling van rijtjes berekend met in elk rijtje de plaatsen die ongelijk aan 0 zijn van een codewoord. Aangezien de codes die ik bekijk cyclisch zijn kan ik deze verzameling vereenvoudigen door alle cyclische verschuivingen eruit te gooien. Ik houd dan een kleine overzichtelijke verzameling van positierijtjes over, de basisposities. In $\mathbf{F}_2$ geeft deze verzameling genoeg informatie om alle codewoorden van minimumgewicht te vinden.
In Stap 8 worden ook voor grotere lichamen alle codewoorden van minimumgewicht bepaald. Per oplossingsrijtje wordt het Mattson-Solomon polynoom berekend. Zoals we in de theorie al zagen kun je uit het Mattson-Solomon polynoom gemakkelijk het bijbehorende codewoord terugvinden.

4.3 Resultaten en afleidingen

In Bijlage 3 op pagina 52 zijn de input en output van het algoritme voor de codes $\mathbf{G}_{11}$ en $\mathbf{G}_{23}$ te vinden. Hieronder zal ik de resultaten samenvatten. Ook zal ik uit de gevonden resultaten enkele dingen afleiden.

4.3.1 $\mathbf{G}_{11}$

Het codewoordenalgoritme gebruikt als startwaarde de BCH grens. Voor $\mathbf{G}_{11}$ is die grens gelijk aan 4 zoals we al eerder zagen. In de output zien we daarom als eerste het stelsel $S_{\mathbf{G}_{11}}(4)$. Het ideaal voortgebracht door dit stelsel heeft geen goede Gröbner basis. Vervolgens rekent het algoritme de Gröbner basis van het ideaal voortgebracht door stelsel $S_{\mathbf{G}_{11}}(5)$ uit. Deze Gröbner basis blijkt wel goed te zijn. Dus heeft dit stelsel oplossingen ongelijk aan 0. Volgens Stelling 11 (op pagina 17) is de minimumafstand van $\mathbf{G}_{11}$ dus 5. Het kleinste polynoom in ordening van de goede Gröbner basis ga ik nader bekijken. Dit polynoom is:

$$f = X^{133} + 2X^{111} + 2X^{89} + 2X^{67} + X^{45} + X.$$

Polynoom f heeft 133 nulpunten in $\mathbf{F}_{3^5}$, waarvan 0 er een is. Uit de driehoekige vorm van de Gröbner basis volgt dat er dus 132 oplossingen ongelijk aan 0 zijn van het stelsel. Er zijn dus 132 codewoorden van minimumgewicht 5. Vervolgens is in de output een rij van positierijtjes te zien. Deze rij heeft het algoritme berekend met behulp van het polynoom $\sigma(Z)$. Tenslotte worden alle codewoorden van minimumgewicht gegeven. Deze heeft mijn algoritme met behulp van het Mattson-Solomon polynoom berekend. Onderaan de output staat de tijd die Magma bezig is geweest met rekenen. Voor de verzameling positierijtjes met daarin alle rijtjes met in elk rijtje de plaatsen die ongelijk aan 0 zijn van een codewoord geldt de volgende eigenschap:

Stelling 15 *Zij B de verzameling positierijtjes. Dan is B de verzameling van blokken van een 4-design, het Steiner systeem $S(4,5,11)$.*

Bewijs: B is een verzameling met 66 rijtjes van 5 elementen. Alle elementen komen uit de verzameling $\{1,\ldots,11\}$. Dus we kunnen B als een verzameling van deelverzamelingen van grootte 5 uit $\{1,\ldots,11\}$ zien.
Stel we bekijken een deelverzameling $\{x_1,x_2,x_3,x_4,x_5\}$ en hiervan het viertal $x_1,\ldots,x_4$. Kan dit viertal dan nog in een andere deelverzameling van B voorkomen? Nee, want dan zou de afstand tussen de twee bijbehorende codewoorden 2 zijn en dat is minder dan de minimumafstand, dus onmogelijk. Dus we kunnen B als een verzameling van deelverzamelingen van grootte 5 uit $\{1,\ldots,11\}$ zien, waarvoor geldt dat elk viertal elementen in hoogstens één deelverzameling voorkomt.
Nu hoef ik alleen nog maar te controleren of elk viertal uit $\{1,\ldots,11\}$ in een deelverzameling voorkomt. Het totale aantal mogelijke viertallen is:

$$\binom{11}{4} = 330.$$

Uit elke deelverzameling kun je op meerdere manieren een viertal halen:

$$\binom{5}{4} = 5.$$

Dus zijn er $\frac{330}{5} = 66$ deelverzamelingen van grootte 5 nodig om alle viertallen te bevatten. En dat is precies het aantal deelverzamelingen dat B bevat. Dus we kunnen B als een verzameling van deelverzamelingen van grootte 5 uit $\{1, \ldots, 11\}$ zien, waarvoor geldt dat elk viertal elementen in precies één deelverzameling voorkomt. Dit is de definitie van het Steiner systeem $S(4, 5, 11)$.

QED

4.3.2 $\mathbf{G}_{23}$

Aangezien de BCH grens voor $\mathbf{G}_{23}$ gelijk aan 5 is begint het algoritme vanaf 5 te lopen. Het ideaal voortgebracht door het stelsel $S_{\mathbf{G}_{23}}(5)$ heeft geen goede Gröbnerbasis, dus gaat mijn algoritme verder voor $w = 6$. Ook het ideaal voortgebracht door stelsel $S_{\mathbf{G}_{23}}(6)$ heeft een Gröbner basis die geen oplossingen ongelijk aan 0 heeft. Tenslotte wordt het stelsel $S_{\mathbf{G}_{23}}(7)$ door het algoritme ingevoerd. Helaas is het niet gelukt om voor het ideaal I voortgebracht door dit stelsel een Gröbner basis te berekenen. In het hoofdstuk, waarin ik de theorie van de Gröbner bases introduceer, heb ik een probleem aangegeven. Dit probleem zat in het Buchberger algoritme, waarmee een Gröbner basis berekend kan worden. Het komt erop neer dat tijdens het algoritme het aantal gegevens explosief kan groeien. Bij de berekening van de Gröbner basis van ideaal I gebeurt dit. Dan nog zou er op een gegeven moment een Gröbner basis gevonden moeten worden, als je Magma maar lang genoeg laat lopen. De reden dat dit niet werkt is het beperkte geheugen van Magma. Tijdens een berekening slaat Magma alle gegevens en tussenresultaten op in een zogenaamd *online geheugen*. Dit geheugen heeft slechts een beperkte grootte. Tijdens de berekening van de Gröbner basis van I werd de hoeveelheid tussenresultaten te groot voor het online geheugen. Verschillende pogingen om het probleem op andere machines met meer online geheugen te laten lopen hebben helaas nog geen succes gehad.

In Hoofdstuk 3 heb ik reeds laten vallen wat de minimumafstand van $\mathbf{G}_{23}$ is; namelijk 7. Stel mijn codewoordenalgoritme had wel een Gröbner basis voor I gevonden, dan waren we dus ook klaar geweest. Het algoritme lost dan net zoals bij $\mathbf{G}_{11}$ het stelsel op met behulp van de gevonden basis. Er is een klein verschil met de methode bij $\mathbf{G}_{11}$. Daar hadden we het Mattson-Solomon polynoom nodig om de codewoorden te vinden. Bij $\mathbf{G}_{23}$ is dat echter niet nodig. Aangezien de woorden van $\mathbf{G}_{23}$ over het lichaam $\mathbf{F}_2$ zijn gedefinëerd hebben we genoeg aan de rij positierijtjes. Elk positierijtje geeft dan aan op welke posities een codewoord van minimumgewicht het element 1 heeft staan. Op die manier zou mijn algoritme dan alle codewoorden van gewicht 7 vinden.

Ook al heeft het algoritme voor $\mathbf{G}_{23}$ nog geen succes gehad neem ik vanaf nu aan dat de minimumafstand van Golay code $\mathbf{G}_{23}$ bekend is en gelijk aan 7 is.

5 Decoderen

Het doel van een boodschap coderen is dat eventuele fouten bij het verzenden van een gecodeerde boodschap makkelijk hersteld kunnen worden (tot een bepaald aantal fouten althans). In dit deel zal ik twee methodes bespreken om ontvangen woorden van cyclische codes te decoderen.

Het algoritme dat ik volgens de eerste methode heb geïmplementeerd kan gebruikt worden voor het decoderen van alle cyclische codes. Volgens de tweede methode heb ik twee algoritmes geïmplementeerd. Het ene algoritme decodeert ontvangen woorden van $\mathbf{G}_{11}$ en het andere algoritme decodeert ontvangen woorden van $\mathbf{G}_{23}$. Met enkele aanpassingen kan ik beide algoritmes herschrijven zodat het voor een andere code gebruikt kan worden. Er moet alleen per specifieke code wat voorbereidingswerk verricht worden voordat het algoritme geschreven en gebruikt kan worden.

Ik zal vanaf nu gebruik maken van de gegevens die ik gevonden heb in het vorige hoofdstuk. Ik weet dus de minimumafstand van beide codes en hieruit volgt (zie Stelling 8 (op pagina 12) en 9 (op pagina 12)) dat beide codes perfect zijn.

Tenslotte ga ik de twee methodes voor het decoderen die ik behandel met elkaar vergelijken. Dit doe ik door de snelheden van de algoritmes te vergelijken door ze voor gelijke input te laten lopen. Ook zal ik naar de tijd kijken die nodig was ter voorbereiding aan het implementeren van de codes; de zogeheten *preprocesseertijd*.

5.1 Algoritme met gegeven syndromen

5.1.1 De theorie

Zij C een $[n,k,d]_q$ cyclische code met generatorpolynoom $g(X)$ en definiërende verzameling $J=\{j_1,\ldots,j_r\}$. Zij $\mathbf{F}_{q^e}$ een uitbreiding van $\mathbf{F}_q$ die alle nulpunten van $g(X)$ bevat en $\alpha \in \mathbf{F}_{q^e}$ een primitieve n-de eenheidswortel. Dan is H een parity check matrix van C:

$$H = \begin{pmatrix} 1 & \alpha^{j_1} & \alpha^{2j_1} & \ldots & \alpha^{(n-1)j_1} \\ 1 & \alpha^{j_2} & \alpha^{2j_2} & \ldots & \alpha^{(n-1)j_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{j_r} & \alpha^{2j_r} & \ldots & \alpha^{(n-1)j_r} \end{pmatrix}.$$

We gaan H uitbreiden tot $\hat{H}$:

$$\hat{H} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \alpha^{2\cdot 2} & \ldots & \alpha^{(n-1)2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^n & \alpha^{2n} & \ldots & \alpha^{(n-1)n} \end{pmatrix}.$$

Zij $e(X)$ een foutvector met foutposities (niet 0 posities) $i_1, \ldots, i_t$ en foutwaarden $e_{i_1}, \ldots, e_{i_t}$. Dan kennen we deze syndromen:

$$s_j = \sum_{m=1}^{t} e_{i_m} (\alpha^{i_m})^j, \quad met \;\; j \in J.$$

We gaan nu een zeker stelsel van vergelijkingen $S(s, v)$ over de polynoomring $\mathbf{F}_{q^e}[X_1, \ldots, X_v, Y_1, \ldots, Y_v]$ bekijken. Dit stelsel bestaat uit $2v + \#J$ vergelijkingen in de $2v$ onbekenden $X_1, \ldots, X_v$ en $Y_1, \ldots, Y_v$. Hierbij is $v \in \{1, \ldots, n\}$ een variabele die ingevoerd wordt in het stelsel en s is de syndroomvector van de foutvector.

$$S(s, v) = \begin{cases} \sum_{m=1}^{v} Y_m X_m^j & = & s_j & j \in J \\ Y_m^q & = & Y_m & m = 1, \ldots, v \\ X_m^n & = & 1 & m = 1, \ldots, v. \end{cases}$$

We gaan dit stelsel nu voor een zekere $v \in \{1, \ldots, n\}$ en een gegeven syndroomvector s oplossen over het lichaam $\mathbf{F}_{q^e}$.

Laten we het stelsel eerst eens wat beter bekijken.
We gaan naar eventuele oplossingen over $\mathbf{F}_{q^e}$ zoeken. Dus eventuele oplossingen voor de Y_m zitten in $\mathbf{F}_{q^e}$. Er geldt $Y_m^q = Y_m$ *met* $m = 1, \ldots, v$, dus eventuele oplossingen voor de Y_m zitten in $\mathbf{F}_q$ voor alle m. Zo ook zitten oplossingen voor de X_m in $\mathbf{F}_{q^e}$ en dit is het lichaam met α als primitieve n-de eenheidswortel. Er geldt $X_m^n = 1$ *met* $m = 1, \ldots, v$, dus eventuele oplossingen voor de X_m zitten in $\{\alpha^0, \alpha^1, \ldots, \alpha^{n-1}\}$ voor alle m. Dus $S(s, v)$ ziet er vereenvoudigd als volgt uit:

$$S(s, v) = \begin{cases} \sum_{m=1}^{v} Y_m X_m^j = s_j \\ j \in J, \; Y_m \in \mathbf{F}_q \text{ en } X_m = \alpha^i \text{ (zekere } i \; \in \{0, \ldots, n-1\}), \\ \text{voor } m = 1, \ldots, v. \end{cases}$$

Stelling 16 *Stel dat tijdens het verzenden van een codewoord t fouten zijn opgetreden met $t \leq \frac{(d-1)}{2}$. Dan geldt:*

1. *Als $v < t$ dan heeft het stelsel $S(s, v)$ geen oplossingen.*
2. *Als $v = t$ dan heeft het stelsel $S(s, v)$ een unieke oplossing op permutatie na. De oplossing correspondeert met de foutvector van het kleinste gewicht die voldoet aan de syndroomvergelijkingen. De X_i van de oplossing zijn de foutlocators en de Y_i zijn de bijbehorende foutwaarden.*
3. *Als $v > t$ dan heeft het stelsel voor iedere j een oplossing met $X_1 = \alpha^j$.*

Bewijs: We vervangen in stelsel $S(s, v)$ de variabelen Y_m door $\lambda_m \in \mathbf{F}_q$ en de variabelen X_m door α^{μ_m} met $\mu_m \in \{0, \ldots, n-1\}$.

1. Zij $v < t$, dan wordt het stelsel:

$$S(s,v) = \begin{cases} \lambda_1(\alpha^{\mu_1})^{j_1} + \ldots + \lambda_v(\alpha^{\mu_v})^{j_1} & = & e_{i_1}(\alpha^{i_1})^{j_1} + \ldots + e_{i_t}(\alpha^{i_t})^{j_1} \\ \lambda_1(\alpha^{\mu_1})^{j_2} + \ldots + \lambda_v(\alpha^{\mu_v})^{j_2} & = & e_{i_1}(\alpha^{i_1})^{j_2} + \ldots + e_{i_t}(\alpha^{i_t})^{j_2} \\ \vdots & & \vdots \\ \lambda_1(\alpha^{\mu_1})^{j_r} + \ldots + \lambda_v(\alpha^{\mu_v})^{j_r} & = & e_{i_1}(\alpha^{i_1})^{j_r} + \ldots + e_{i_t}(\alpha^{i_t})^{j_r}, \end{cases}$$

met $\lambda_1, \ldots, \lambda_v \in \mathbf{F}_q$ en $\mu_1, \ldots, \mu_v \in \{0, \ldots, n-1\}$.
Dit kunnen we vertalen naar

$$\lambda_1 \begin{pmatrix} (\alpha^{\mu_1})^{j_1} \\ (\alpha^{\mu_1})^{j_2} \\ \vdots \\ (\alpha^{\mu_1})^{j_r} \end{pmatrix} + \ldots + \lambda_v \begin{pmatrix} (\alpha^{\mu_v})^{j_1} \\ (\alpha^{\mu_v})^{j_2} \\ \vdots \\ (\alpha^{\mu_v})^{j_r} \end{pmatrix} = e_{i_1} \begin{pmatrix} (\alpha^{i_1})^{j_1} \\ (\alpha^{i_1})^{j_2} \\ \vdots \\ (\alpha^{i_1})^{j_r} \end{pmatrix} + \ldots + e_{i_t} \begin{pmatrix} (\alpha^{i_t})^{j_1} \\ (\alpha^{i_t})^{j_2} \\ \vdots \\ (\alpha^{i_t})^{j_r} \end{pmatrix}.$$

Nu halen we alle kolommen naar links. Dan krijgen we een lineaire combinatie van k kolommen die gelijk is aan 0, waarbij $t - v \leq k \leq t + v$ en alle kolommen van de vorm $((\alpha^l)^{j_1}, (\alpha^l)^{j_2}, \ldots, (\alpha^l)^{j_r})^\top$ zijn. Omdat $v < t \leq \frac{(d-1)}{2}$ geldt dus dat:

$$1 \leq k \leq t + v < d - 1.$$

De minimumafstand is d, dus kan er geen lineaire combinatie van $< d$ dergelijke kolommen zijn die 0 oplevert. Dit weten we nog van het bewijs van Stelling 3 (op pagina 4). Dus er is geen oplossing van het stelsel als $v < t$.

2. Zij $v = t$. Op een analoge wijze als in het eerste deel van het bewijs zien we dat we k kolommen krijgen die in een lineaire combinatie 0 opleveren. Voor deze k geldt:

$$t - v \leq k \leq t + v,$$

dus:

$$0 \leq k \leq d - 1.$$

Hieraan is direct te zien dat er precies één oplossing is; namelijk voor $k = 0$. Dat wil zeggen dat in het stelsel $S(s,t)$ links en rechts van het gelijkteken exact dezelfde lineaire combinatie staat; op eventuele permutatie na. De X_i zijn de foutlocators en de Y_i zijn de foutwaarden. Nu weten we met behulp van Stelling 4 (op pagina 8) dat deze oplossing uniek is; en wel de foutvector. Dit is dus de vector met het kleinste gewicht die aan de syndroomvergelijkingen voldoet.

3. Zij $v > t$, dan bekijken we weer:

$$\lambda_1 \begin{pmatrix} (\alpha^{\mu_1})^{j_1} \\ (\alpha^{\mu_1})^{j_2} \\ \vdots \\ (\alpha^{\mu_1})^{j_r} \end{pmatrix} + \ldots + \lambda_v \begin{pmatrix} (\alpha^{\mu_v})^{j_1} \\ (\alpha^{\mu_v})^{j_2} \\ \vdots \\ (\alpha^{\mu_v})^{j_r} \end{pmatrix} = e_{i_1} \begin{pmatrix} (\alpha^{i_1})^{j_1} \\ (\alpha^{i_1})^{j_2} \\ \vdots \\ (\alpha^{i_1})^{j_r} \end{pmatrix} + \ldots + e_{i_t} \begin{pmatrix} (\alpha^{i_t})^{j_1} \\ (\alpha^{i_t})^{j_2} \\ \vdots \\ (\alpha^{i_t})^{j_r} \end{pmatrix},$$

met $\lambda_1, \ldots, \lambda_v \in \mathbf{F}_q$ en $\mu_1, \ldots, \mu_v \in \{0, \ldots, n-1\}$.
Omdat nu $v > t$ geldt zie je meteen dat je voor μ_1 alles kan kiezen wat je wil. Kies namelijk maar een zekere μ_1, dan:

(a) Stel $\mu_1 = j \in \{i_1, \ldots, i_t\}$, dan kies je $\lambda_1 = e_j$. Vervolgens houd je een lineaire combinatie van $v-1$ kolommen over die je gelijkstelt aan een lineaire combinatie van $t-1$ kolommen; met $v > t$. Daar is natuurlijk een oplossing van.

(b) Stel $\mu_1 \notin \{i_1, \ldots, i_t\}$, dan kies je $\lambda_1 = 0$. Vervolgens houd je een lineaire combinatie van $v-1$ kolommen over die je gelijkstelt aan een lineaire combinatie van t kolommen; met $v > t$. Ook hier is natuurlijk een oplossing van.

Dus is er voor iedere j een oplossing met $X_1 = \alpha^j$.

QED

Lemma 17 *Zij I een ideaal in $\mathbf{F}[Z_1, Z_2, \ldots, Z_w]$ met eindig veel nulpunten over de algebraïsche afsluiting $\bar{\mathbf{F}}$ van $\mathbf{F}$, die allemaal in $\mathbf{F}$ zitten. Zij V de verzameling nulpunten in $\mathbf{F}^w$ van het ideaal I. Dan is voor $1 \leq i \leq w$ de verzameling nulpunten van $I \cap \mathbf{F}[Z_1, Z_2, \ldots, Z_i]$ gelijk aan de projectie van V op de eerste i coördinaten.*

Bewijs: Het is triviaal dat de projectie van V op de eerste i coördinaten bevat is in de verzameling nulpunten van $I \cap \mathbf{F}[Z_1, Z_2, \ldots, Z_i]$. Om te bewijzen dat de verzameling nulpunten van $I \cap \mathbf{F}[Z_1, Z_2, \ldots, Z_i]$ bevat is in de projectie van V op de eerste i coördinaten moet er wat werk verricht worden. Ik zal het bewijzen voor $i = w-1$. Met inductie volgt dan het bewijs voor $1 \leq i \leq w$. Zij $(b_1, \ldots, b_{w-1})$ een nulpunt in $I \cap \mathbf{F}[Z_1, Z_2, \ldots, Z_{w-1}]$. Omdat I eindig veel nulpunten heeft over de algebraïsche afsluiting $\bar{\mathbf{F}}$ van $\mathbf{F}$, die allemaal in $\mathbf{F}$ zitten, geldt er: $A = \mathbf{F}[Z_1, Z_2, \ldots, Z_w]/I$ is eindig dimensionaal. Zij r de dimensie van A. Zij nu $\bar{Z}_w = Z_w + I$, dan zijn $1, \bar{Z}_w, \ldots, \bar{Z}_w^{r+1} \in A$ afhankelijk. Dus er zijn $\lambda_0, \ldots, \lambda_r$ met:

$$\lambda_0 + \lambda_1 \bar{Z}_w + \ldots + \bar{Z}_w^{r+1} = 0,$$

dus

$$p = \lambda_0 + \lambda_1 Z_w + \ldots + Z_w^{r+1} \in I.$$

Er is nu een β nulpunt van p, waarvoor $(b_1, \ldots, b_{w-1}, \beta)$ een nulpunt is van I. Dus is $(b_1, \ldots, b_{w-1})$ de projectie van een nulpunt van V op de eerste $w-1$ coördinaten. QED

Stelling 18 *Stel dat tijdens het verzenden van een codewoord t fouten zijn opgetreden met $t \leq \frac{(d-1)}{2}$. Zij $g(X_1)$ de monische generator van het ideaal voortgebracht door $S(s,t) \cap \mathbf{F}_{q^e}[X_1]$. Dan zijn de nulpunten van g de foutlocators.*

Bewijs: Dit is een direct gevolg van Lemma 17 (op pagina 29) en Stelling 6 (op pagina 11). In het kort: Zij $(x_1, \ldots, x_v, y_1, \ldots, y_v)$ een oplossing van het stelsel $S(s,v)$. Dan is ook $(x_2, \ldots, x_v, x_1, y_2, \ldots, y_v, y_1)$ een oplossing. Zo zijn ook alle analoge permutaties oplossingen. Dus komt elke foutlocator een keer op de eerste coördinaat van een oplossing van $S(s,v)$ te staan. Deze opmerking levert samen met Lemma 17 en Stelling 6 het bewijs. QED

Stelling 19 *Stel dat tijdens het verzenden van een codewoord t fouten zijn opgetreden met* $t \leq \frac{(d-1)}{2}$. *Zij* $l(X_1)$ *een monisch polynoom, zodat:* $l(x) = 0 \Longleftrightarrow x$ *is een foutlocator. We noemen l het foutlocator polynoom. Zij* $g(X_1)$ *de monische generator van het ideaal voortgebracht door* $S(s,v) \cap \mathbf{F}_{q^e}[X_1]$. *Dan:*

$$g(X_1) = \begin{cases} 1 & \textit{voor } v < t \\ l(X_1) & \textit{voor } v = t \\ X_1^n - 1 & \textit{voor } v > t. \end{cases}$$

Bewijs: Het bewijs van deze stelling is ook triviaal. Eigenlijk staat er hetzelfde als in Stelling 16 (op pagina 27), alleen nu toegepast op $S(s,v) \cap \mathbf{F}_{q^e}[X_1]$. Samen met Stelling 18 (op pagina 29) ben je dan klaar. QED

5.1.2 Het decodeeralgoritme

Het algoritme dat ik heb geschreven maakt gebruik van de theorie in de vorige subparagraaf. In Bijlage 4 op pagina 58 is het decodeeralgoritme afgedrukt. Ik zal het algoritme hieronder in stappen beschrijven. Hiervoor nog enkele opmerkingen.

1. Mijn decodeeralgoritme zal voor oplopende $v = 1, 2, \ldots$ het stelsel $S(s,v)$ gaan oplossen. Hierbij geeft Stelling 19 (op pagina 30) aan wanneer het algoritme moet stoppen. De gevonden Gröbner basis G van het ideaal voortgebracht door $S(s,v)$ zal volgens deze stelling namelijk telkens 1 bevatten als er geen oplossing is. Dus zodra G geen 1 meer bevat heeft het stelsel een oplossing en wel de oplossing met het minste gewicht, aangezien je met oplopende v werkt.
2. De lexicografische ordening, $\prec_L$, die ik gebruik om de Gröbner basis G te berekenen maakt de volgende ordening:

$$X_1 \prec_L \ldots \prec_L X_v \prec_L Y_1 \prec_L \ldots \prec_L Y_v.$$

Het decodeeralgoritme:

INPUT

$n :=$; $q :=$; $J := \{\ \}$; load "gegevenscode"; $y :=$;

Stap 1: $s := yH^\top$.

Stap 2: Als $s = 0$, stop het algoritme.
OUTPUT y.

Stap 3: v:=1.

Stap 4: Maak stelsel $S(s,v)$.
OUTPUT v en $S(s,v)$

Stap 5: Maak het ideaal dat wordt voortgebracht door dit stelsel.

Stap 6: Bereken de Gröbnerbasis G van dat ideaal.
OUTPUT G

Stap 7: Als $1 \in G$ doe $v := v + 1$ en herhaal Stap 4 tot en met Stap 7.

Stap 8: Bereken uit G het foutlocator polynoom $g(X_1)$.
OUTPUT Het foutlocator polynoom $g(X)$

Stap 9: Bereken met behulp van g de foutvector en de correcte boodschap.
OUTPUT De foutvector e en het gedecodeerde woord y'.

In Stap 9 maakt mijn algoritme onderscheid tussen twee gevallen:

1. Voor $q = 2$ hebben we genoeg aan het foutlocator polynoom. De nulpunten van dit polynoom geven de plekken aan waar in de foutvector een 1 moet staan.

2. Voor $q \neq 2$ heeft het algoritme nog wat meer werk. Met behulp van de nulpunten van het foutlocator polynoom en de Gröbnerbasis kan bij elke positie bijbehorende foutwaarde berekend worden.

5.1.3 Resultaten en afleidingen

In Bijlage 5 zijn de input (op pagina 61) en output (op pagina 64) van het decodeeralgoritme voor de codes $\mathbf{G}_{11}$ en $\mathbf{G}_{23}$ te vinden. Voor beide codes voer ik meerdere ontvangen woorden in:

1. Bij allebei de codes voer ik als eerste een codewoord c in. Het decodeeralgoritme geeft het codewoord c weer als output en de tijd die gebruikt is bij de berekening.

2. Vervolgens plaats ik per keer dat ik het algoritme laat lopen telkens één fout meer in c. Zolang het aantal fouten onder de grens van $\frac{(d-1)}{2}$ blijft geeft mijn algoritme de juiste foutvector en het codewoord c weer. Ook wordt de rekentijd weer aangegeven. In de uitvoer worden als tussenstappen telkens de stelsels $S(s, v)$ en bijbehorende Gröbner bases voor oplopende v weergegeven.

3. Als laatste voer ik voor beide codes het codewoord c in met meer dan $\frac{(d-1)}{2}$ fouten; namelijk $\frac{(d-1)}{2} + 1$ fouten. Voor oplopende v wordt het stelsel $S(s, v)$ afgedrukt tot een Gröbner basis, G, is gevonden met $1 \notin G$. Mijn algoritme vindt voor $v = \frac{d-1}{2}$ zo'n basis. Het algoritme geeft dus een foutvector met v posities ongelijk aan 0 en een codewoord c' dat ongelijk is aan het aanvankelijke codewoord c. Hierna volgt de benodigde rekentijd.

Over het decodeeralgoritme wil ik twee dingen opmerken:

Ten eerste werkt het algoritme niet precies hetzelfde voor beide codes. Voor $\mathbf{G}_{23}$ ben je klaar als je de nulpunten van het foutlocator polynoom hebt berekend. Voor $\mathbf{G}_{11}$ gebruik je één nulpunt van het foutlocator polynoom om vervolgens met behulp van de Gröbner basis alle foutposities en bijbehorende foutwaarden te berekenen. Dit verschil zit 'm zoals al eerder gezegd in rekenen over een lichaam met twee elementen of over een lichaam met meer dan twee elementen.

Ten tweede wou ik het over de uitvoer hebben, wanneer ik meer dan $\frac{(d-1)}{2}$ fouten in het codewoord plaats. Het algoritme vindt dan een ander codewoord. Dit heeft te maken met het feit dat beide codes perfecte codes zijn. Voor elk willekeurige ontvangen woord geldt namelijk dat er één uniek codewoord is met afstand $\leq \frac{(d-1)}{2}$ tot dit ontvangen woord. Mijn algoritme vindt altijd het codewoord met de kortste afstand tot het ontvangen woord en dat is dus het unieke codewoord met afstand $\leq \frac{(d-1)}{2}$ tot het ontvangen woord.

5.2 Eénstapsmethode

De methode om cyclische codes te decoderen die ik in deze paragraaf ga bekijken lijkt erg op de decodeermethode uit de vorige paragraaf.

5.2.1 De theorie

Bij het decodeeralgoritme uit de vorige paragraaf maakten we gebruik van het stelsel vergelijkingen $S(s, v)$. In dit stelsel waren de s_j bekende constanten. Voor de éénstapsmethode ga ik hetzelfde stelsel bekijken, met als enige verschil dat ik de syndromen nu als variabelen zal beschouwen. Zo krijgen we het volgende stelsel:

$$S(v) = \begin{cases} \sum_{m=1}^{v} Y_m X_m^j &= S_j \quad j \in J \\ Y_m^q &= Y_m \quad m = 1, \dots, v \\ X_m^n &= 1 \quad m = 1, \dots, v, \end{cases}$$

Dit stelsel brengt een ideaal I in de polynoomring $\mathbf{F}_q[X_1, \dots, X_v, Y_1, \dots, Y_v, S_{j_1}, \dots, S_{j_r}]$ voort. We willen dit stelsel nu voor een zekere $v \in \{1, \dots, n\}$ over het lichaam $\mathbf{F}_q$ oplossen. Hiertoe berekenen we eerst de Gröbner basis G van het ideaal I met betrekking tot lexicografische ordening $\prec_L$ met:

$$X_1 \prec_L S_{j_1} \prec_L \dots \prec_L S_{j_r} \prec_L X_2 \prec_L \dots \prec_L X_v \prec_L Y_1 \prec_L \dots \prec_L Y_v.$$

We willen namelijk net zoals bij het decodeeralgoritme de variabelen $X_2, \dots, X_v$ en $Y_1, \dots, Y_v$ elimineren. Daarom bekijken we vervolgens de polynoomverzameling $G_1 = G \cap \mathbf{F}_q[X_1, S_{j_1}, \dots, S_{j_r}]$.

Het doel van de éénstapsmethode is om letterlijk in één stap klaar te zijn. Dus gaan we op zoek naar het zogenaamde *éénstapspolynoom* f in $\mathbf{F}_q(S_{j_1}, \dots, S_{j_r})[X_1]$. Dit polynoom f is per definitie voor iedere syndroomvector s gelijk aan bijbehorende foutlocator polynoom wanneer we de variabelen S_j door de constanten s_j vervangen voor $j \in J$. Er moet veel werk verricht worden om dit éénstapspolynoom te vinden..

Voor alle mogelijke syndroomvectoren s vervangen we de variabelen S_j in de polynomen van G_1 door s_j voor $j \in J$. Vervolgens berekenen we telkens de grootste gemene deler f_s van deze polynomen. Hiermee kunnen we een polynoom f in $\mathbf{F}_q(S_{j_1}, \dots, S_{j_r})[X_1]$ berekenen dat gelijk is aan f_s als je de variabelen S_j door de constanten s_j voor $j \in J$ vervangt. Dit polynoom f is het éénstapspolynoom.

Nu rest mij nog te bewijzen dat de éénstapsmethode werkt. Het is zo niet duidelijk dat het vervangen van de variabelen S_j door de syndromen na eliminatie hetzelfde antwoord geeft als de eerdere methode met bekende syndromen. Ik zal aantonen dat het wel opgaat.
Zij G een verzameling van elementen uit de polynoomring $\mathbf{F}_q[X_1, \dots, X_v, Y_1, \dots, Y_v, S_j, j \in J]$. Zij $G_1 = G \cap \mathbf{F}_q[X_1, S_{j_1}, \dots, S_{j_r}]$ en

$s = (s_{j_1}, \ldots, s_{j_r})$ een vector met coördinaten in $\mathbf{F}_q$. Dan is $G_1(s)$ de verzameling die je krijgt door in G_1 alle S_j te vervangen door de waarden s_j.

Stelling 20 *Zij G een Gröbner basis van $S(t)$ met betrekking tot $\prec_L$. Zij y een ontvangen woord met t fouten. Zij s het bijbehorende syndroom. Ga er verder ook vanuit dat het dichtstbijzijnde codewoord van y uniek is. Dan is G_1 de Gröbner basis van $S(t) \cap \mathbf{F}_q[X_1, S_j, j \in J]$. Ook is $G_1(s)$ een Gröbner basis en het foutlocator polynoom is een element van $G_1(s)$.*

Bewijs: Dat G_1 de Gröbner basis van $S(t) \cap \mathbf{F}_q[X_1, S_j, j \in J]$ is volgt uit Stelling 6 (op pagina 11). Voor de rest van het bewijs zie [1], pagina 267.

Deze methode heeft enkele voordelen. Ten eerste rekenen we over het lichaam $\mathbf{F}_q$ dat vaak veel kleiner is dan het lichaam $\mathbf{F}_{q^e}$. Ten tweede hoeven we de vergelijkingen maar één keer op te lossen. Dit preprocesseren kost weliswaar veel tijd, maar tijdens het lopen van het éénstapsalgoritme zal tijd bespaard worden vergeleken met het decodeeralgoritme. We hebben namelijk door het preprocesseren het éénstapspolynoom f gevonden. Om nu het foutlocator polynoom te vinden bij een zekere ontvangen vector hoeven we alleen nog maar de variabelen S_j in f te vervangen door de constanten s_j van de syndroomvector.

5.2.2 Eénstapsmethode voor $\mathbf{G}_{11}$

Ik ga nu de theorie gebruiken om een algoritme voor $\mathbf{G}_{11}$ te maken dat ontvangen woorden decodeert. Het preprocesseren van het algoritme heb ik in Bijlage 6.6.1 op pagina 68 afgedrukt. Ik zal eerst beschrijven wat daar gebeurt. Vervolgens zal ik het éénstapsalgoritme beschrijven en afdrukken. Tenslotte laat ik het algoritme voor verschillende ontvangen vectoren lopen.

Preprocesseren We weten dat de minimumafstand van $\mathbf{G}_{11}$ gelijk is aan 5 en dat de code perfect is. Hierdoor zagen we al eerder dat een ontvangen vector y altijd hoogstens 2 fouten verwijderd is van een codewoord c'. Dit codewoord hoeft niet gelijk te zijn aan het oorspronkelijke verzonden codewoord c. Maar aangezien we er bij minimumafstands decoderen vanuitgaan dat de kans groot is dat er weinig fouten in y zitten, geven we vector c' als output. Voor het éénstapsalgoritme gaan we dus uit van een maximaal aantal van 2 fouten in een ontvangen vector. Het stelsel vergelijkingen wordt dus:

$$S(2) = \begin{cases} Y_1X_1 + Y_2X_2 - S &= 0 \\ Y_1^3 - Y_1 &= 0 \\ Y_2^3 - Y_1 &= 0 \\ X_1^{11} - 1 &= 0 \\ X_2^{11} - 1 &= 0. \end{cases}$$

Van het ideaal voortgebracht door dit stelsel wordt de Gröbner basis berekend met betrekking tot de lexicografische ordening $\prec_L$:

$$X_1 \prec_L S \prec_L X_2 \prec_L Y_1 \prec_L Y_2.$$

Het preprocesseeralgoritme vindt de volgende Gröbner basis bestaande uit 8 polynomen in $\mathbf{F}_3[X_1, X_2, Y_1, Y_2, S]$ (gedeeltelijk afgedrukt):

```
[
 Y2 + Y1 + 2*S^33 + 2*S^31*X1^2 + 2*S^29*X1^4 +
  2*S^27*X1^6 + S^9*X1^2 +  S^7*X1^4 + S^5*X1^6,
 Y1^3 + 2*Y1,
 ..........
 ..........,
 S^63 + 2*S^57*X1^6 + S^45*X1^7 + 2*S^37*X1^4 +
  2*S^33*X1^8 + S^31*X1^10 +  S^27*X1^3 + 2*S^19 +
  S^15*X1^4 + S^13*X1^6 + S^11*X1^8 + 2*S^9*X1^10 +
  2*S^5*X1^3 + 2*S*X1^7,
 X1^11 + 2
]
```

Zij f het laatste basiselement en g het op één na laatste basiselement. Deze laatste twee basiselementen zijn de enige polynomen in uitsluitend de variabelen X_1 en S. Uit Stelling 6 (op pagina 11) volgt dat $\{f, g\}$ de Gröbner basis is van $S(2) \cap \mathbf{F}_q[X_1, S]$. Zij s een syndroom. Dan volgt uit Stelling 20 (op pagina 34) dat $\{f(s), g(s)\}$ een Gröbner basis is en dat het foutlocator polynoom een element is van deze basis. In het preprocesseeralgoritme ga ik nu achtereenvolgens alle mogelijke syndromen invullen. Per syndroom bereken ik de grootste gemene deler van $f(s)$ en $g(s)$. Die deler is telkens een polynoom van graad maximaal 2 en is gelijk aan het foutlocator polynoom behorend bij het syndroom.

Ik wil daarbij opmerken dat ik voor syndromen afkomstig van één fout iets anders te werk ga. Het foutlocator polynoom is voor die syndromen triviaal. Voor een syndroom $s = \lambda\alpha^i$ is $X_1 - \alpha^i$ het foutlocator polynoom.

Zo vind ik uiteindelijk een rij met 242 polynomen. Hierna bereken ik het éénstapspolynoom $U(S)X^2 + V(S)X + W(S)$. Als je voor S een syndroom invult is het éénstapspolynoom gelijk aan het foutlocator polynoom uit het rijtje. Ik heb nu dus een algemeen foutlocator polynoom gevonden voor alle mogelijke syndromen. De tijd die Magma nodig had om dit polynoom te vinden was:

```
De rekentijd van het preprocesseren is:
3.490
```

In Bijlage 6.6.2 op pagina 70 heb ik het polynoom afgedrukt.

Het éénstapsalgoritme voor $\mathbf{G}_{11}$ Het daadwerkelijke éénstapsalgoritme stelt nu niet veel meer voor. Al het werk is al tijdens het preprocesseren gebeurd. Na enkele korte opmerkingen over de stappen van het algoritme druk ik het hieronder af. Met behulp van die opmerkingen is makkelijk te begrijpen hoe het algoritme werkt.

Het éénstapsalgoritme voor $\mathbf{G}_{11}$:

INPUT
$n := 11;\ q := 3;\ J := \{1\};$ load "gegevenscode";
load "polynoomG11"; $y :=$;

Stap 1: $s := yH^\top$.

Stap 2: Als $s = 0$, stop het algoritme.
OUTPUT y.

Stap 3: Vervang variabele S in het éénstapspolynoom door syndroom s.

Stap 4: Bepaal met het foutlocator polynoom dat in Stap 3 is gevonden posities waar niet 0 staat in de foutvector.

Stap 5: Bepaal de foutwaarden op ieder van die posities.
OUTPUT De foutvector e en het gedecodeerde woord y'.

Bij Stap 5 ga ik iets anders te werk dan gebruikelijk is. Normaal gesproken bereken je de foutwaarden met behulp van de eerder gevonden Gröbner basis. In mijn algoritme ga ik echter gewoon even de mogelijkheden af. Dit zijn er namelijk zo weinig dat het op deze manier makkelijker en sneller is.

Het éénstapsalgoritme voor $\mathbf{G}_{11}$ in Magma:

```
tijd:=Cputime(0.00);

//Wil van J een Sequence maken, maar soms is t dat al,
//dus met een omweg:
JJ:=J;J:=[];
for j in JJ do
        J:=J cat [j];
end for;

H:=Matrix(Fqe,#(J),n,[[a^((i-1)*j) : i in [1..n]] : j in J]);
s:=[];
for i in [1..#(J)] do
s[i]:=InnerProduct(y,(Vector(RowSubmatrix(H,i,1))));
end for;
nul:=[0 : i in [1..#(J)]];
if s eq nul then
        "Tijdens het verzenden zijn geen fouten opgetreden.";
        "Het verzonden codewoord is dus:";
        y;
else
        "Tijdens het verzenden zijn fouten opgetreden.";
        Pol<Y>:=PolynomialRing(GF(3,5));
        syn:=s[1];
```

```
        u:=Evaluate(U,syn);
        v:=Evaluate(V,syn);w:=Evaluate(W,syn);
        f:=Pol ! [w,v,u];
        nulp:=Roots(f);
        e:=Vector(GF(q),[0 : i in [1..n]]);
        locators:=[];
        for j in [1..#(nulp)] do
                locators:=Append(locators,Log(a,nulp[j][1]));
        end for;
        waarde:=[];
        if #locators eq 1 then
                for k in [1,2] do
                        effe:=k*(a^(locators[1]));
                        if effe eq syn then
                                waarde[1]:=k;
                        end if;
                end for;
        else
                for k in [[1,1],[1,2],[2,1],[2,2]] do
                        effe:=k[1]*a^(locators[1])+
                              k[2]*a^(locators[2]);
                        if effe eq syn then
                                waarde[1]:=k[1];waarde[2]:=k[2];
                        end if;
                end for;
        end if;
        for j in [1..#locators] do
                e[locators[j]+1]:=waarde[j];
        end for;
        "De errorvector is:";
        e;
        "De oorspronkelijke boodschap was dus:";
        c:=y-e;
        c;
end if;

"De totale rekentijd van het programma is:";
print Cputime(tijd);
```

Algoritme in werking Om het algoritme uit te proberen heb ik dezelfde ontvangen vectoren als bij het decodeeralgoritme ingevoerd (zie Bijlage 6.5.1 op pagina 61):

INPUT

$n := 11;\ q := 3;\ J := \{1\};$ load "gegevenscode";
load "polynoomG11";

1. INPUT

$$y := \text{Vector}(GF(q), [0, 0, 0, 2, 2, 2, 0, 2, 0, 0, 2]);$$

OUTPUT

```
Tijdens het verzenden zijn geen fouten opgetreden.
Het verzonden codewoord is dus:
(0 0 0 2 2 2 0 2 0 0 2)
De totale rekentijd van het programma is:
0.030
```

2. INPUT

$$y := \text{Vector}(GF(q), [1, 0, 0, 2, 2, 2, 0, 2, 0, 0, 2]);$$

OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
De errorvector is:
(1 0 0 0 0 0 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(0 0 0 2 2 2 0 2 0 0 2)
De totale rekentijd van het programma is:
0.040
```

3. INPUT

$$y := \text{Vector}(GF(q), [0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2]);$$

OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
De errorvector is:
(0 2 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(0 0 0 2 2 2 0 2 0 0 2)
De totale rekentijd van het programma is:
0.030
```

4. INPUT

$$y := \text{Vector}(GF(q), [1, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2]);$$

OUTPUT

```
De errorvector is:
(0 0 0 1 0 0 0 1 0 0 0)
De oorspronkelijke boodschap was dus:
(1 2 0 1 2 0 0 1 0 0 2)
De totale rekentijd van het programma is:
0.030
```

De uitvoer spreekt voor zich, dus voeg ik verder geen commentaar toe.

5.2.3 Eénstapsmethode voor $\mathbf{G}_{23}$

Dit keer ga ik de theorie gebruiken om een algoritme voor $\mathbf{G}_{23}$ te maken dat ontvangen woorden decodeert. Het preprocesseren van het algoritme is wederom in de bijlagen afgedrukt; en wel in Bijlage 6.6.3 op pagina 71. Ik zal weer eerst beschrijven wat daar gebeurt en daarna wat in het éénstapsalgoritme gebeurt. Tenslotte zal ik het algoritme afdrukken en uitproberen voor verschillende ontvangen vectoren.

Preprocesseren Van de perfecte Golay code $\mathbf{G}_{23}$ is de minimumafstand 7. Om dezelfde reden als bij Golay code $\mathbf{G}_{11}$ gaan we uit van een maximaal aantal van 3 fouten in een ontvangen vector. Dus gebruiken we het volgende stelsel vergelijkingen:

$$S(3) = \begin{cases} X_1 + X_2 + X_3 + S & = & 0 \\ X_1^{24} + X_1 & = & 0 \\ X_2^{24} + X_2 & = & 0 \\ X_3^{24} + X_3 & = & 0. \end{cases}$$

Ik voer in het stelsel $X_j^{24} + X_j = 0$ in plaats van $X_j^{23} + 1 = 0$ in. Zo krijgen we namelijk een minder gecompliceerde Gröbner basis. Van het ideaal voortgebracht door dit stelsel wordt de Gröbner basis van berekend met betrekking tot lexicografische ordening $\prec_L$:

$$X_1 \prec_L S \prec_L X_2 \prec_L X_3.$$

Het preprocesseeralgoritme vindt Gröbner basis:

```
[
 X3 + X2 + S + X1,
 X2^24 + X2,
 X2^2*S + X2^2*X1 + X2*S^2 + X2*X1^2 + S^256 + S^3 + S^2*X1 +
        S*X1^2,
 S^277 + S^276*X1 + S^273*X1^4 + S^272*X1^5 + S^261*X1^16 +
  S^260*X1^17 + S^257*X1^20 + S^256*X1^21 + S^70 + S^68*X1^2 +
  S^66*X1^4 + S^64*X1^6 + S^47 + S^46*X1 + S^45*X1^2 + S^44*X1^3+
  S^43*X1^4 + S^42*X1^5 + S^41*X1^6 + S^40*X1^7 + S^39*X1^8 +
  S^38*X1^9 + S^37*X1^10 + S^36*X1^11 + S^35*X1^12 + S^34*X1^13 +
  S^33*X1^14 + S^32*X1^15 + S^21*X1^3 + S^20*X1^4 + S^17*X1^7 +
  S^16*X1^8 + S^15*X1^9 + S^14*X1^10 + S^13*X1^11 + S^12*X1^12 +
  S^11*X1^13 + S^10*X1^14 + S^9*X1^15 + S^8*X1^16 + S^7*X1^17 +
  S^4*X1^20 + S^3*X1^21 + S,
 X1^24 + X1
]
```

Zij f het laatste basiselement en g het op één na laatste basiselement. Ik ga nu hetzelfde te werk als bij code $\mathbf{G}_{11}$. Zo zal ik per syndroom de grootste gemene deler van $f(s)$ en $g(s)$ berekenen en opslaan in een rijtje. Die deler is telkens een polynoom van maximaal graad 3 en is gelijk aan het foutlocator polynoom horend bij het syndroom.

Wederom ga ik voor syndromen afkomstig van één fout iets anders te werk. Het foutlocator polynoom is voor die syndromen triviaal. Voor een syndroom $s = \alpha^i$ is $X_1 - \alpha^i$ het foutlocator polynoom.

Zo vind ik uiteindelijk een rij met 2047 polynomen. Hierna bereken ik het éénstapspolynoom $T(S)X^3 + U(S)X^2 + V(S)X + W(S)$. Als je voor S een syndroom invult is het éénstapspolynoom gelijk aan het foutlocator polynoom uit het rijtje. Ik heb dus weer een algemeen foutlocator polynoom gevonden voor alle mogelijke syndromen. De tijd die Magma nodig had om dit polynoom te vinden was:

```
De rekentijd van het preprocesseren is:
1506.160
```

In Bijlage 6.6.4 op pagina 73 heb ik het polynoom (gedeeltelijk) afgedrukt.

Het éénstapsalgoritme voor $\mathbf{G}_{23}$ Het éénstapsalgoritme voor $\mathbf{G}_{23}$ is nog makkelijker dan die voor $\mathbf{G}_{11}$. Het algoritme volgt dezelfde stappen 1, 2, 3 en 4 als het éénstapsalgoritme voor $\mathbf{G}_{11}$. Dan zijn de niet 0 posities van de foutvector gevonden en zijn we dus klaar.

Het éénstapsalgoritme voor $\mathbf{G}_{23}$:

INPUT

$n := 23;\ q := 2;\ J := \{1\};$ load ”gegevenscode”;
load ”polynoomG23”; $y :=$;

Stap 1: $s := yH^\top$.

Stap 2: Als $s = 0$, stop het algoritme.
OUTPUT y.

Stap 3: Vervang variabele S in het éénstapspolynoom door syndroom s.

Stap 4: Bepaal met het foutlocator polynoom dat in Stap 3 is gevonden posities waar niet 0 staat in de foutvector.
OUTPUT De foutvector e en het gedecodeerde woord y'.

Het éénstapsalgoritme voor $\mathbf{G}_{23}$ in Magma:

```
tijd:=Cputime(0.00);

//Wil van J een Sequence maken, maar soms is t dat al,
//dus met een omweg:
```

```
JJ:=J;J:=[];
for j in JJ do
        J:=J cat [j];
end for;

H:=Matrix(Fqe,#(J),n,[[a^((i-1)*j) : i in [1..n]] : j in J]);
s:=[];
for i in [1..#(J)] do
        s[i]:=InnerProduct(y,(Vector(RowSubmatrix(H,i,1))));
end for;
nul:=[0 : i in [1..#(J)]];
if s eq nul then
        "Tijdens het verzenden zijn geen fouten opgetreden.";
        "Het verzonden codewoord is dus:";
        y;
else
        "Tijdens het verzenden zijn fouten opgetreden.";
        Pol<Y>:=PolynomialRing(GF(2,11));
        syn:=s[1];
        t:=Evaluate(T,syn);u:=Evaluate(U,syn);
        v:=Evaluate(V,syn);w:=Evaluate(W,syn);
        f:=Pol ! [w,v,u,t];
        nulp:=Roots(f);
        e:=Vector(GF(q),[0 : i in [1..n]]);
        locators:=[];
        for j in [1..#(nulp)] do
                locators:=Append(locators,Log(a,nulp[j][1]));
        end for;
        for j in locators do
                e[j+1]:=1;
        end for;
        "De errorvector is:";
        e;
        "De oorspronkelijke boodschap was dus:";
        c:=y-e;
        c;
end if;

"De totale rekentijd van het programma is:";
print Cputime(tijd);
```

Algoritme in werking Om het algoritme uit te proberen heb ik dezelfde ontvangen vectoren als bij het decodeeralgoritme ingevoerd (zie Bijlage 6.5.2 op pagina 64):

INPUT

$n := 23;\ q := 2;\ J := \{1\};$ load "gegevenscode";
load "polynoomG23";

1. INPUT

   $y := \text{Vector}(GF(q), [1,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]);$

   OUTPUT

```
Tijdens het verzenden zijn geen fouten opgetreden.
Het verzonden codewoord is dus:
(1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
De totale rekentijd van het programma is:
0.010
```

2. INPUT

   $y := \text{Vector}(GF(q), [1,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0]);$

   OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
De errorvector is:
(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
De totale rekentijd van het programma is:
0.020
```

3. INPUT

   $y := \text{Vector}(GF(q), [0,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0]);$

   OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
De errorvector is:
(1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
De totale rekentijd van het programma is:
0.020
```

4. INPUT

   $y := \text{Vector}(GF(q), [0,1,0,1,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0]);$

   OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
De errorvector is:
(1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
De totale rekentijd van het programma is:
0.020
```

5. INPUT

$y :=$ Vector$(GF(q), [0,1,0,1,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,1]);$

OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
De errorvector is:
(0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1)
De totale rekentijd van het programma is:
0.020
```

De uitvoer spreekt voor zich, dus voeg ik verder geen commentaar toe.

5.3 De twee methodes vergeleken

Hieronder heb ik een tabel afgedrukt. Hierin staan de tijden die de algoritmes nodig hadden om verschillende ontvangen woorden te decoderen. Ik heb voor elke mogelijke hoeveelheid fouten een aantal vectoren ingevoerd. Per vector heb ik de rekentijden (in seconden) van beide algoritmes naast elkaar geplaatst. Ook de preprocesseertijd (ook in seconden) staat in de tabel. Op de volgende bladzijde trek ik mijn conclusie uit al deze gegevens.

Code	Aantal fouten	Decodeeralgoritme	Eénstapsalgoritme
$\mathbf{G}_{11}$	0	0.040	0.030
	0	0.080	0.050
	0	0.030	0.020
	0	0.020	0.020
	1	0.060	0.040
	1	0.070	0.020
	1	0.030	0.020
	1	0.020	0.020
	2	0.090	0.030
	2	0.080	0.030
	2	0.080	0.010
	2	0.080	0.030
Preprocesseertijd		0.000	3.490
$\mathbf{G}_{23}$	0	0.020	0.010
	0	0.020	0.010
	0	0.030	0.010
	0	0.030	0.010
	1	0.030	0.020
	1	0.030	0.030
	1	0.030	0.020
	1	0.020	0.020
	2	0.040	0.020
	2	0.040	0.020
	2	0.040	0.020
	2	0.030	0.020
	3	0.190	0.020
	3	0.180	0.020
	3	0.190	0.020
	3	0.230	0.020
Preprocesseertijd		0.000	1506.160

Conclusie:

De tabel spreekt boekdelen. De preprocesseertijd van het éénstapsalgoritme is (veel) groter dan van het decodeeralgoritme (vooral voor $\mathbf{G}_{23}$). Het gebruik van het éénstapsalgoritme is echter juist altijd (veel) zuiniger dan van het decodeeralgoritme. Hoe meer fouten er in een ontvangen woord zitten des te groter wordt het verschil in rekentijd. Dit is vrij logisch. Bij het decodeeralgoritme wordt namelijk per fout een stelsel gemaakt en opgelost. Dus bij veel fouten moet er flink wat werk verzet worden. Voor het éénstapsalgoritme maakt het niet veel uit hoe veel fouten er in de vector zaten. Er moet altijd praktisch hetzelfde berekend worden (vandaar ook de naam *één*stapsmethode). Nu is de kans op weinig fouten in een ontvangen woord natuurlijk groter dan op veel fouten. Maar ook voor weinig fouten is het éénstapsalgoritme sneller. In mijn ogen is het éénstapsalgoritme een beter algoritme. Het kost wel wat tijd om het voor te bereiden, maar als het eenmaal af is functioneert het goed en snel. Het decodeeralgoritme functioneert goed, maar zeker niet snel. Elke keer opnieuw moet het weer praktisch dezelfde stelsels maken en oplossen. Dat is natuurlijk niet handig en ik zou het decodeeralgoritme dus nooit verkiezen boven het éénstapsalgoritme.

6 Bijlagen

6.1 Bijlage 1: De Golay codes

6.1.1 De generatormatrix van $\mathbf{G}_{11}$

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 2 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 2 & 2 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 2 & 2 \\
0 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 2 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 2 & 2
\end{pmatrix}$$

6.1.2 De generatormatrix van $\mathbf{G}_{23}$

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

6.2 Bijlage 2: Het codewoordenalgoritme

```
t:=Cputime(0.00);

//Aantal variabelen zoeken:
K:={1..n};
K:=K diff JC;L:={};M:=[];
while not IsEmpty(K) do
        i:=Min(K);K:=K diff {i};L:=Include(L,i);
        for j in {1..(e)} do
                l:=(i*q^j) mod n;
                if l in K then
                        M:=M cat [[l,i,q^j]];
                        K:=K diff {l};
                end if;
        end for;
end while;
K:=L;
if n in K then
        onthoud:=true;
end if;

//Variabelen benoemen:
//Let op, variabelen zijn aftellend benoemd, zodat xk >..> x1
k:=#(K);
FF:=RationalFunctionField(GF(q),k);
AssignNames(~FF,["x" cat IntegerToString(k+1-i) : i in [1..k]]);
A:=[FF|-1 : i in {1..n}];
for i in JC do
        A[i]:=0;
end for;
j:=1;
for i in [1..n] do
        if A[i] eq -1 and i in K then
                A[i]:=FF.(k+1-j);
                j:=j+1;
        elif A[i] eq -1 then
                for s in M do
                        if s[1] eq i then
                                A[i]:=A[s[2]]^s[3];
                        end if;
                end for;
        end if;
end for;

//Stelsel invoeren en oplossen:
flag1:=false;
w:=BCH;
```

```
while not flag1 do
        //Let op, variabelen zijn aftellend benoemd;
        //S(w+1) >..> S1 > Xk >..>X1
        H:=PolynomialRing(FF,w+1);
        S:=Matrix(H,n,w+1,[<i,j,A[n-((n-(w+1-j+i)) mod n)]> :
        i in [1..n], j in [1..w+1]]);
        AssignNames(~H,["s" cat IntegerToString(w+2-i) :
        i in [1..(w+1)]]);
        V:=Vector(w+1,[1] cat [H.(w+2-i) : i in [1..(w)]]);
        Stelsel:={H|};
        for i in [1..n] do
                Stelsel:=Include(Stelsel,InnerProduct(S[i],V));
        end for;
        if onthoud then
                Stelsel:=Include(Stelsel,FF.1^3-FF.1);
        end if;
        Y := PolynomialRing(Fqe, k+w+1);
        AssignNames(~Y,
        [ "S" cat IntegerToString(w+2-i) : i in [1..(w+1)]] cat
        [ "X" cat  IntegerToString(k+1-i) : i in [1..k]]);
        g := hom< FF -> Y | [ Y.j : j in [w+2..w+k+1]] >;
        h := hom< H -> Y | g, [ Y.j : j in [1..w+1]]>;
        "Voor w is:";
                w;
        "krijgen we het volgende stelsel:";
                Stelsel;
        G := GroebnerBasis(h(Stelsel));
        flag2,f:=IsUnivariate(G[#(G)],Y.(w+1+k));
        if flag2 then
                flag3:=false;l:=1;D:=Roots(f);
                while not flag3 and l le #(D) do
                        if D[l][1] ne 0 then
                                flag1:=true;flag3:=true;
                        else
                                l:=l+1;
                        end if;
                end while;
        end if;
        if not flag1 then
                "Het stelsel heeft deze Groebnerbasis:";
                G;
                "en die heeft geen niet-nul-oplossingen.";"\n";
                w:=w+1;
        end if;
end while;

"De bijbehorende Groebnerbasis is:";
G;
"Het volgende polynoom geeft dus alle oplossingen:";
R<X>:=PolynomialRing(Fqe);
```

```
f;
"En dat zijn er:";
#(D)-1;
"De minimumafstand is dus:";
w;
"en van deze afstand zijn het volgende aantal codewoorden:";
#(D)-1;

//Niet 0 posities codewoorden bepalen:

//Rijtjes van mogelijke oplossingen
//[Sw+1 .. S1 , Xk .. X1] maken:
oplossingen:=[];
for i in [1..#(D)] do
        nulpunt:=[Y|];
        if D[i][1] ne 0 then
                nulpunt[w+1+k]:=[D[i][1]];
                for j in [1..(#(G)-1)] do
                        evaluatierijtje:=
                        [Y.i : i in [1..(w+1+k-j)]] cat
                        [nulpunt[l] :
                        l in [(w+1+k-(j-1))..(w+1+k)]];
                        bullshitflag,f:=
                        IsUnivariate
                        (Evaluate(G[#(G)-j],evaluatierijtje));
                        nulpunt[w+1+k-j]:=Roots(f)[1][1];
                end for;
                oplossingen:=oplossingen cat [nulpunt];
        end if;
end for;

//Per rijtje het error-locator-polynoom maken en oplossen:
codeposities:=[];
for p in [1..#(oplossingen)] do
        f:=1;
        for i in [2..(w+1)] do
                f:=f+(Fqe ! oplossingen[p][i])*(X^(w+2-i));
        end for;
        nul:=Roots(f);
        woordposities:=[];
        for i in [1..#(nul)] do
                woordposities:=
                woordposities cat [11-Log(a,nul[i][1])];
        end for;
        codeposities:=Append(codeposities,woordposities);
end for;
codeposities:=SequenceToSet(codeposities);
"De volgende verzameling geeft alle niet 0 posities
van de codewoorden aan:";
codeposities;
```

```
"Deze verzameling geeft dus het volgende aantal woorden:";
#codeposities;
if #codeposities eq (#D-1) then
        "Dit zijn ook alle woorden,
         want er zijn geen lineaire veelvouden.";
else
        "Het totaal aantal woorden is groter,
         want er zijn ook nog lineaire veelvouden.";
end if;

"Alle basisposities zonder hun (eventuele) lineaire veelvouden
 en hun cyclische shifts:";
basisposities:={};
weggooiposities:=codeposities;
while #(basisposities) ne (((#(oplossingen))/n)/(q-1)) do
        positie:=Random(weggooiposities);
        weggooiposities:=weggooiposities diff {positie};
        basisposities:=Include(basisposities,positie);
        for j in [1..(n-1)] do
                cycle:=[];
                for i in [1..#(positie)] do
                        cycle:= cycle cat [positie[i]-1];
                end for;
                if cycle[#(cycle)] eq 0 then
                        cycle:=[n] cat cycle[1..(#(cycle)-1)];
                end if;
                if cycle in weggooiposities then
                        weggooiposities:=
                        weggooiposities diff {cycle};
                end if;
                positie:=cycle;
        end for;
end while;
basisposities;


//De daadwerkelijke codewoorden bepalen:
codewoorden:=[];
if (q-1) eq 1 then
        for pos in codeposities do
                c:=[0 : i in [1..n]];
                for j in pos do
                        c[j]:=1;
                end for;
                codewoorden:=Include(codewoorden,c);
        end for;
        "Mbv de niet 0 posities vinden we nu alle codewoorden:";
else
        Xrijtje:=g(A);
        for i in [1..#(oplossingen)] do
```

```
                L<Z>:=PolynomialRing(Fqe);
                AA:=[L|];
                evaluatierijtje:=
                [0] cat oplossingen[i][2..(w+1+k)];
                for j in [1..n] do
                        AA[j]:=
                        Evaluate(Xrijtje[j],evaluatierijtje);
                end for;
                pol:=0;
                for l in [1..n] do
                        pol:=pol + AA[l]*Z^(n-l);
                end for;
                c:=[];
                for m in [1..n] do
                        c[m]:=(Evaluate(pol,a^(m-1)))/n;
                end for;
                codewoorden:=Include(codewoorden,c);
        end for;
        "Mbv het Mattson-Solomon polynoom
        vinden we nu alle codewoorden:";
end if;
codewoorden;

"De totale rekentijd van het programma is:";
print Cputime(t);
```

6.3 Bijlage 3: Het codewoordenalgoritme in werking

Ik heb bij beide codes niet de volledige output van het algoritme afgedrukt. Op plaatsen met veel punten op een rij heb ik wat weggelaten. Dit heb ik gedaan, omdat het weggelatene geen extra waarde toevoegde en alleen ruimte innam. Verder is het ook wel duidelijk wat er ongeveer hoort te staan.

6.3.1 Voor $\mathbf{G}_{11}$

INPUT

$$n := 11;\ q := 3;\ J := \{1\};\ \text{load "gegevenscode"};$$

OUTPUT

```
Voor w is:
4
krijgen we het volgende stelsel:
{
 x2^3 + 2*x2,
 x1^27*s4 + x2*s3 + x1*s1,
 x1^27*s3 + x2*s2 + x1,
 x1^81*s4 + x1^27*s2 + x2*s1,
 x1^9*s4 + x1^81*s3 + x1^27*s1 + x2,
 x1^3*s3 + x1^9*s2 + x1^81*s1,
 x1^3*s1 + x1^9,
 x1*s4 + x1^3,
 x1*s3,
 x1^3*s2 + x1^9*s1 + x1^81,
 x1^3*s4 + x1^9*s3 + x1^81*s2 + x1^27,
 x2*s4 + x1*s2
}
Het stelsel heeft deze Groebnerbasis:
[
 X2,
 X1
]
en die heeft geen niet-nul-oplossingen.

Voor w is:
5
krijgen we het volgende stelsel:
{
 x1*s4 + x1^3,
 x1^3*s4 + x1^9*s3 + x1^81*s2 + x1^27,
 x1^81*s5 + x1^27*s3 + x2*s2 + x1,
 x1^9*s5 + x1^81*s4 + x1^27*s2 + x2*s1,
 x2^3 + 2*x2,
```

```
 x1^27*s5 + x2*s4 + x1*s2,
 x1^27*s4 + x2*s3 + x1*s1,
 x1^3*s5 + x1^9*s4 + x1^81*s3 + x1^27*s1 + x2,
 x1*s5 + x1^3*s1 + x1^9,
 x1^3*s3 + x1^9*s2 + x1^81*s1,
 x1^3*s2 + x1^9*s1 + x1^81,
 x2*s5 + x1*s3
}
De bijbehorende Groebnerbasis is:
[
 S5*X1 + 2*X1^31 + 2*X1^9,
 S4*X1 + X1^3,
 S3*X1 + 2*X1^107 + X1^41 + 2*X1^19,
 S2*X1 + X1^79 + 2*X1^35 + X1^13,
 S1*X1 + X1^29 + 2*X1^7,
 X2 + X1^77 + 2*X1^55 + X1^33 + X1^11,
 X1^133 + 2*X1^111 + 2*X1^89 + 2*X1^67 + X1^45 + X1
]

Het volgende polynoom geeft dus alle oplossingen:
X^133 + 2*X^111 + 2*X^89 + 2*X^67 + X^45 + X
En dat zijn er:
132
De minimumafstand is dus:
5
en van deze afstand zijn het volgende aantal codewoorden:
132

De volgende verzameling geeft alle niet 0 posities
van de codewoorden aan:
{
 [ 11, 7, 5, 3, 1 ],
 [ 11, 8, 7, 5, 2 ],
 [ 11, 10, 6, 3, 1 ],
 [ 8, 6, 4, 2, 1 ],
 [ 11, 10, 8, 7, 4 ],
 [ 11, 8, 7, 6, 1 ],
 ................
 ................,
 [ 9, 8, 7, 6, 4 ],
 [ 11, 10, 8, 5, 3 ],
 [ 10, 9, 6, 5, 4 ],
 [ 9, 8, 5, 4, 3 ],
 [ 10, 7, 5, 4, 3 ],
 [ 10, 8, 7, 3, 1 ]
}

Deze verzameling geeft dus het volgende aantal woorden:
66
Het totaal aantal woorden is groter,
```

```
want er zijn ook nog lineaire veelvouden.

Alle basisposities zonder hun (eventuele) lineaire veelvouden
en hun cyclische shifts:
{
 [ 11, 9, 6, 4, 1 ],
 [ 11, 8, 4, 3, 1 ],
 [ 11, 10, 9, 4, 3 ],
 [ 8, 7, 6, 5, 3 ],
 [ 11, 7, 4, 2, 1 ],
 [ 10, 9, 5, 3, 1 ]
}

Mbv het Mattson-Solomon polynoom vinden we nu alle codewoorden:
[
 [ 0, 0, 2, 0, 0, 0, 2, 2, 2, 0, 2 ],
 [ 1, 0, 0, 2, 1, 0, 0, 0, 0, 2, 2 ],
 [ 1, 1, 0, 0, 0, 2, 0, 0, 2, 2, 0 ],
 [ 2, 1, 1, 0, 0, 0, 0, 0, 2, 0, 1 ],
 [ 0, 0, 1, 2, 0, 2, 0, 1, 0, 2, 0 ],
 [ 2, 0, 0, 2, 0, 2, 0, 0, 1, 0, 1 ],
 ...................................
 ...................................,
 [ 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1 ],
 [ 1, 2, 0, 0, 0, 0, 1, 1, 2, 0, 0 ],
 [ 0, 0, 1, 0, 0, 1, 1, 0, 2, 2, 0 ],
 [ 0, 0, 0, 0, 0, 1, 0, 2, 1, 2, 2 ],
 [ 1, 0, 1, 0, 2, 0, 1, 0, 0, 0, 2 ],
 [ 1, 0, 1, 0, 0, 2, 0, 2, 1, 0, 0 ]
]

De totale rekentijd van het programma is:
3.819
```

6.3.2 Voor $\mathbf{G}_{23}$

INPUT

$n := 23;\ q := 2;\ J := \{1\};$ load "gegevenscode";

OUTPUT

```
Voor w is:
5
krijgen we het volgende stelsel:
{
 x1^2*s5 + x1^16*s4 + x1^1024*s1 + x1^256,
 x1^512*s4 + x1^2*s1 + x1^16,
 x2^3 + x2,
 x1^128*s5 + x1^4*s4 + x1^64*s3 + x1^32*s2 + x2*s1,
 x1^8*s5 + x1^128*s3 + x1^4*s2 + x1^64*s1 + x1^32,
 x1^2*s4 + x1^16*s3 + x1^1024,
 x1*s2 + x1^512,
 x1*s3 + x1^512*s1,
 x1^2*s3 + x1^16*s2,
 x1*s1,
 x1^8*s4 + x1^128*s2 + x1^4*s1 + x1^64,
 x1*s5 + x1^512*s3 + x1^2,
 x1^128*s4 + x1^4*s3 + x1^64*s2 + x1^32*s1 + x2,
 x1^512*s5 + x1^2*s2 + x1^16*s1,
 x1^4*s5 + x1^64*s4 + x1^32*s3 + x2*s2,
 x1^1024*s4 + x1^256*s3 + x1^8*s1,
 x1*s4 + x1^512*s2,
 x1^256*s5 + x1^8*s3 + x1^128*s1 + x1^4,
 x2*s5 + x1,
 x1^64*s5 + x1^32*s4 + x2*s3,
 x1^1024*s5 + x1^256*s4 + x1^8*s2 + x1^128,
 x1^32*s5 + x2*s4,
 x1^1024*s3 + x1^256*s2 + x1^8,
 x1^16*s5 + x1^1024*s2 + x1^256*s1
}
Het stelsel heeft deze Groebnerbasis:
[
 X2,
 X1
]
en die heeft geen niet-nul-oplossingen.

Voor w is:
6
krijgen we het volgende stelsel:
{
 x1^2*s5 + x1^16*s4 + x1^1024*s1 + x1^256,
 x2^3 + x2,
```

```
 x1^128*s5 + x1^4*s4 + x1^64*s3 + x1^32*s2 + x2*s1,
 x1^8*s5 + x1^128*s3 + x1^4*s2 + x1^64*s1 + x1^32,
 x2*s6 + x1*s1,
 x1^16*s6 + x1^1024*s3 + x1^256*s2 + x1^8,
 x1^2*s4 + x1^16*s3 + x1^1024,
 x1*s2 + x1^512,
 x1*s3 + x1^512*s1,
 x1^1024*s6 + x1^256*s5 + x1^8*s3 + x1^128*s1 + x1^4,
 x1^4*s6 + x1^64*s5 + x1^32*s4 + x2*s3,
 x1^32*s6 + x2*s5 + x1,
 x1*s5 + x1^512*s3 + x1^2,
 x1^512*s5 + x1^2*s2 + x1^16*s1,
 x1^8*s6 + x1^128*s4 + x1^4*s3 + x1^64*s2 + x1^32*s1 + x2,
 x1^1024*s4 + x1^256*s3 + x1^8*s1,
 x1*s4 + x1^512*s2,
 x1^2*s6 + x1^16*s5 + x1^1024*s2 + x1^256*s1,
 x1^64*s6 + x1^32*s5 + x2*s4,
 x1^1024*s5 + x1^256*s4 + x1^8*s2 + x1^128,
 x1^128*s6 + x1^4*s5 + x1^64*s4 + x1^32*s3 + x2*s2,
 x1^256*s6 + x1^8*s4 + x1^128*s2 + x1^4*s1 + x1^64,
 x1^512*s6 + x1^2*s3 + x1^16*s2,
 x1*s6 + x1^512*s4 + x1^2*s1 + x1^16
}
Het stelsel heeft deze Groebnerbasis:
[
 X2,
 X1
]
en die heeft geen niet-nul-oplossingen.

Voor w is:
7
krijgen we het volgende stelsel:
{
 x1^2*s5 + x1^16*s4 + x1^1024*s1 + x1^256,
 x2*s7 + x1*s2 + x1^512,
 x2^3 + x2,
 x1^16*s7 + x1^1024*s4 + x1^256*s3 + x1^8*s1,
 x1^64*s7 + x1^32*s6 + x2*s5 + x1,
 x1^256*s7 + x1^8*s5 + x1^128*s3 + x1^4*s2 + x1^64*s1 + x1^32,
 x1^8*s7 + x1^128*s5 + x1^4*s4 + x1^64*s3 + x1^32*s2 + x2*s1,
 x1*s3 + x1^512*s1,
 x1^1024*s6 + x1^256*s5 + x1^8*s3 + x1^128*s1 + x1^4,
 x1^2*s7 + x1^16*s6 + x1^1024*s3 + x1^256*s2 + x1^8,
 x1^32*s7 + x2*s6 + x1*s1,
 x1*s5 + x1^512*s3 + x1^2,
 x1^512*s7 + x1^2*s4 + x1^16*s3 + x1^1024,
 x1^8*s6 + x1^128*s4 + x1^4*s3 + x1^64*s2 + x1^32*s1 + x2,
 x1^128*s7 + x1^4*s6 + x1^64*s5 + x1^32*s4 + x2*s3,
 x1*s4 + x1^512*s2,
```

```
 x1^2*s6 + x1^16*s5 + x1^1024*s2 + x1^256*s1,
 x1^1024*s7 + x1^256*s6 + x1^8*s4 + x1^128*s2 + x1^4*s1 + x1^64,
 x1^4*s7 + x1^64*s6 + x1^32*s5 + x2*s4,
 x1*s7 + x1^512*s5 + x1^2*s2 + x1^16*s1,
 x1^1024*s5 + x1^256*s4 + x1^8*s2 + x1^128,
 x1^128*s6 + x1^4*s5 + x1^64*s4 + x1^32*s3 + x2*s2,
 x1^512*s6 + x1^2*s3 + x1^16*s2,
 x1*s6 + x1^512*s4 + x1^2*s1 + x1^16
}
```

6.4 Bijlage 4: Het decodeeralgoritme

```
t:=Cputime(0.00);

//Wil van J een Sequence maken, maar soms is t dat al,
dus met een omweg:
JJ:=J;J:=[];
for j in JJ do
        J:=J cat [j];
end for;

H:=Matrix(Fqe,#(J),n,[[a^((i-1)*j) : i in [1..n]] : j in J]);
s:=[];
for i in [1..#(J)] do
        s[i]:=InnerProduct(y,(Vector(RowSubmatrix(H,i,1))));
end for;
nul:=[0 : i in [1..#(J)]];
if s eq nul then
        "Tijdens het verzenden zijn geen fouten opgetreden.";
        "Het verzonden codewoord is dus:";
        y;
else
        "Tijdens het verzenden zijn fouten opgetreden.";
        v:=0;G:={1};
        while 1 in G do
                Stelsel:={};
                v:=v+1;
                if (q-1) eq 1 then
                        FF:=PolynomialRing(Fqe,v);
                        AssignNames(~FF,["X" cat
                        IntegerToString(v+1-i) : i in [1..v]]);
                        for i in [1..v] do
                                Stelsel:=
                                Include(Stelsel,(FF.(i))^n-1);
                        end for;
                        for j in [1..#(J)] do
                                som:=0;
                                for i in [1..v] do
                                        som:=som+(FF.(i))^(J[j]);
                                end for;
                                som:=som-s[j];
                                Stelsel:=Include(Stelsel,som);
                        end for;
                else
                        FF:=PolynomialRing(Fqe,2*v);
                        AssignNames(~FF,
                        ["Y" cat IntegerToString(v+1-i) :
                        i in [1..v]]
```

```
                cat
                ["X" cat IntegerToString(v+1-i) :
                i in [1..v]]);
                for i in [1..v] do
                        Stelsel:=
                        Include(Stelsel,(FF.i)^q-FF.i);
                        Stelsel:=
                        Include(Stelsel,(FF.(v+i))^n-1);
                end for;
                for j in [1..#(J)] do
                        som:=0;
                        for i in [1..v] do
                            som:=som+
                            ((FF.i)*(FF.(i+v))^(J[j]));
                        end for;
                        som:=som-s[j];
                        Stelsel:=Include(Stelsel,som);
                end for;
        end if;
        SetPowerPrinting(Fqe,false);
        "We krijgen het volgende stelsel voor v is:";
        v;Stelsel;
        "Dit stelsel heeft Groebnerbasis:";
        G:=GroebnerBasis(Stelsel);G;
end while;

//Mbv de Groebnerbasis de errorvector
//en de boodschap zoeken:
if (q-1) eq 1 then
        flag,f:=IsUnivariate(G[#(G)],FF.(v));
else
        flag,f:=IsUnivariate(G[#(G)],FF.(2*v));
end if;
R<X>:=PolynomialRing(Fqe);
"Dus is dit het error-locator polynoom:";
f;
if (not flag) or (Degree(f) gt v) then
        //Dit kan bij de Golay codes nooit voorkomen.
        //Het zijn namelijk perfecte codes.
        "Tijdens het verzenden van de boodschap zijn
         te veel fouten gemaakt om het oorspronkelijke
         woord nog te hervinden.";
else
        nulpunten:=Roots(f);
end if;
e:=Vector(GF(q),[0 : i in [1..n]]);
if (q-1) eq 1 then
        locators:=[];
        for j in [1..#(nulpunten)] do
                locators:=
```

```
                        Append(locators,Log(a,nulpunten[j][1]));
                end for;
                for j in locators do
                        e[j+1]:=1;
                end for;
        else
                evaluatierijtje:=[];
                evaluatierijtje[2*v]:=nulpunten[1][1];
                for j in [1..(2*v-1)] do
                        rijtje:=[FF.i : i in [1..(2*v-j)]] cat
                        [evaluatierijtje[l] :
                        l in [(2*v-(j-1))..(2*v)]];
                        bullshitflag,f:=
                        IsUnivariate(Evaluate(G[#(G)-j],rijtje));
                        evaluatierijtje[2*v-j]:=Roots(f)[1][1];
                end for;
                for m in [1..v] do
                        e[Log(a,evaluatierijtje[2*v+1-m])+1]:=
                        evaluatierijtje[v+1-m];
                end for;
        end if;
        "De errorvector is:";
        e;
        "De oorspronkelijke boodschap was dus:";
        c:=y-e;
        c;
end if;

"De totale rekentijd van het programma is:";
print Cputime(t);
```

6.5 Bijlage 5: Het decodeeralgoritme in werking

Ik laat het algoritme bij beide codes voor meerdere ontvangen woorden (foute en goede codewoorden) lopen.

6.5.1 Voor $\mathbf{G}_{11}$

INPUT

$$n := 11;\ q := 3;\ J := \{1\};\ \text{load "gegevenscode"};$$

1. Eerst voeren we een boodschap in zonder fouten bij het verzenden:
   INPUT
   $$y := \text{Vector}(GF(q), [0, 0, 0, 2, 2, 2, 0, 2, 0, 0, 2]);$$
   OUTPUT
   ```
   Tijdens het verzenden zijn geen fouten opgetreden.
   Het verzonden codewoord is dus:
   (0 0 0 2 2 2 0 2 0 0 2)
   De totale rekentijd van het programma is:
   0.040
   ```

2. Nu voeren we diezelfde boodschap in met één fout tijdens het verzenden:
   INPUT
   $$y := \text{Vector}(GF(q), [1, 0, 0, 2, 2, 2, 0, 2, 0, 0, 2]);$$
   OUTPUT
   ```
   Tijdens het verzenden zijn fouten opgetreden.
   We krijgen het volgende stelsel voor v is:
   1
   {
    X1^11 + 2,
    Y1^3 + 2*Y1,
    Y1*X1 + 2
   }
   Dit stelsel heeft Groebnerbasis:
   [
    Y1 + 2,
    X1 + 2
   ]
   Dus is dit het error-locator polynoom:
   X + 2
   De errorvector is:
   (1 0 0 0 0 0 0 0 0 0 0)
   De oorspronkelijke boodschap was dus:
   (0 0 0 2 2 2 0 2 0 0 2)
   De totale rekentijd van het programma is:
   0.060
   ```

3. En voor twee fouten geeft het algoritme:

INPUT

$$y := \text{Vector}(GF(q), [0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2]);$$

OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
We krijgen het volgende stelsel voor v is:
1
{
 X1^11 + 2,
 Y1^3 + 2*Y1,
 Y1*X1 + b^4 + 2*b^2 + 1
}
Dit stelsel heeft Groebnerbasis:
[
 1
]
We krijgen het volgende stelsel voor v is:
2
{
 X2^11 + 2,
 X1^11 + 2,
 Y2*X2 + Y1*X1 + b^4 + 2*b^2 + 1,
 Y2^3 + 2*Y2,
 Y1^3 + 2*Y1
}
Dit stelsel heeft Groebnerbasis:
[
 Y2 + (b^4 + 2*b^2 + 2*b)*X1 + b^3 + 2*b^2 + b + 1,
 Y1 + (2*b^4 + b^2 + b)*X1 + 2*b^3 + b^2 + 2*b + 2,
 X2 + X1 + b^4 + 2*b^3 + b^2 + b + 2,
 X1^2 + (b^4 + 2*b^3 + b^2 + b + 2)*X1 + 2*b^3 + 2*b^2 + 2*b
]
Dus is dit het error-locator polynoom:
X^2 + (b^4 + 2*b^3 + b^2 + b + 2)*X + 2*b^3 + 2*b^2 + 2*b
De errorvector is:
(0 2 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(0 0 0 2 2 2 0 2 0 0 2)
De totale rekentijd van het programma is:
0.090
```

4. Als we nog een fout erin stoppen zitten we over de grens van $\frac{(d-1)}{2} = 2$ fouten heen. Het volgende gebeurt:

INPUT

$$y := \text{Vector}(GF(q), [1, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2]);$$

OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
```

```
We krijgen het volgende stelsel voor v is:
1
{
 X1^11 + 2,
 Y1^3 + 2*Y1,
 Y1*X1 + b^4 + 2*b^2
}
Dit stelsel heeft Groebnerbasis:
[
1
]

We krijgen het volgende stelsel voor v is:
2
{
 X2^11 + 2,
 X1^11 + 2,
 Y2*X2 + Y1*X1 + b^4 + 2*b^2,
 Y2^3 + 2*Y2,
 Y1^3 + 2*Y1
}
Dit stelsel heeft Groebnerbasis:
[
 Y2 + 2,
 Y1 + 2,
 X2 + X1 + b^4 + 2*b^2,
 X1^2 + (b^4 + 2*b^2)*X1 + 2*b^4 + b^3 + 2
]
Dus is dit het error-locator polynoom:
X^2 + (b^4 + 2*b^2)*X + 2*b^4 + b^3 + 2
De errorvector is:
(0 0 0 1 0 0 0 1 0 0 0)
De oorspronkelijke boodschap was dus:
(1 2 0 1 2 0 0 1 0 0 2)
De totale rekentijd van het programma is:
0.080
```

6.5.2 Voor $\mathbf{G}_{23}$

INPUT

$$n := 23;\ q := 2;\ J := \{1\};\ \text{load "gegevenscode"};$$

1. Eerst voeren we een boodschap in zonder fouten bij het verzenden:
   INPUT

   $$y := \text{Vector}(GF(q), [1,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]);$$

   OUTPUT

   ```
   Tijdens het verzenden zijn geen fouten opgetreden.
   Het verzonden codewoord is dus:
   (1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
   De totale rekentijd van het programma is:
   0.020
   ```

2. Nu voeren we diezelfde boodschap in met één fout tijdens het verzenden:
   INPUT

   $$y := \text{Vector}(GF(q), [1,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0]);$$

   OUTPUT

   ```
   Tijdens het verzenden zijn fouten opgetreden.
   We krijgen het volgende stelsel voor v is:
   1
   {
    X1 + b^10 + b^9 + b^7 + b^6,
    X1^23 + 1
   }
   Dit stelsel heeft Groebnerbasis:
   [
    X1 + b^10 + b^9 + b^7 + b^6
   ]
   Dus is dit het error-locator polynoom:
   X + b^10 + b^9 + b^7 + b^6
   De errorvector is:
   (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
   De oorspronkelijke boodschap was dus:
   (1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
   De totale rekentijd van het programma is:
   0.030
   ```

3. En voor twee fouten geeft het algoritme:
   INPUT

   $$y := \text{Vector}(GF(q), [0,1,0,0,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0]);$$

   OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
We krijgen het volgende stelsel voor v is:
1
{
 X1 + b^10 + b^9 + b^7 + b^6 + 1,
 X1^23 + 1
}
Dit stelsel heeft Groebnerbasis:
[
 1
]
We krijgen het volgende stelsel voor v is:
2
{
 X2 + X1 + b^10 + b^9 + b^7 + b^6 + 1,
 X2^23 + 1,
 X1^23 + 1
}
Dit stelsel heeft Groebnerbasis:
[
 X2 + X1 + b^10 + b^9 + b^7 + b^6 + 1,
 X1^2 + (b^10 + b^9 + b^7 + b^6 + 1)*X1 +
  b^10 + b^9 + b^7 + b^6
]
Dus is dit het error-locator polynoom:
X^2 + (b^10 + b^9 + b^7 + b^6 + 1)*X +
 b^10 + b^9 + b^7 + b^6
De errorvector is:
(1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0)
De totale rekentijd van het programma is:
0.040
```

4. En voor drie fouten geeft het algoritme:
   INPUT

   $y := \text{Vector}(GF(q), [0,1,0,1,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0]);$

   OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
We krijgen het volgende stelsel voor v is:
1
{
 X1 + b^9 + b^6 + b^3 + b^2 + 1,
 X1^23 + 1
}
Dit stelsel heeft Groebnerbasis:
[
```

```
 1
]
We krijgen het volgende stelsel voor v is:
2
{
 X2^23 + 1,
 X1^23 + 1,
 X2 + X1 + b^9 + b^6 + b^3 + b^2 + 1
}
Dit stelsel heeft Groebnerbasis:
[
 1
]
We krijgen het volgende stelsel voor v is:
3
{
 X2^23 + 1,
 X1^23 + 1,
 X3 + X2 + X1 + b^9 + b^6 + b^3 + b^2 + 1,
 X3^23 + 1
}
Dit stelsel heeft Groebnerbasis:
[
 X3 + X2 + X1 + b^9 + b^6 + b^3 + b^2 + 1,
 X2^2 + X2*X1 + (b^9 + b^6 + b^3 + b^2 + 1)*X2 + X1^2 +
  (b^9 + b^6 + b^3 + b^2 + 1)*X1 + b^6 + b^5 + b^2,
 X1^3 + (b^9 + b^6 + b^3 + b^2 + 1)*X1^2 +
  (b^6 + b^5 + b^2)*X1 + b^9 + b^5 + b^3
]
Dus is dit het error-locator polynoom:
X^3 + (b^9 + b^6 + b^3 + b^2 + 1)*X^2 +
 (b^6 + b^5 + b^2)*X + b^9 + b^5 + b^3
De errorvector is:
(1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0)
De oorspronkelijke boodschap was dus:
(1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0)
De totale rekentijd van het programma is:
0.190
```

5. Als we nog een fout erin stoppen zitten we over de grens van $\frac{(d-1)}{2} = 3$ fouten heen. Het volgende gebeurt:
INPUT

$$y := \text{Vector}(GF(q), [0,1,0,1,0,1,1,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,1]);$$

OUTPUT

```
Tijdens het verzenden zijn fouten opgetreden.
We krijgen het volgende stelsel voor v is:
1
```

```
{
 X1^23 + 1,
 X1 + b^10 + b^9 + b^6 + b^5 + b^4
}
Dit stelsel heeft Groebnerbasis:
[
 1
]
We krijgen het volgende stelsel voor v is:
2
{
 X2 + X1 + b^10 + b^9 + b^6 + b^5 + b^4,
 X2^23 + 1,
 X1^23 + 1
}
Dit stelsel heeft Groebnerbasis:
[
 1
]
We krijgen het volgende stelsel voor v is:
3
{
 X2^23 + 1,
 X1^23 + 1,
 X3^23 + 1,
 X3 + X2 + X1 + b^10 + b^9 + b^6 + b^5 + b^4
}
Dit stelsel heeft Groebnerbasis:
[
 X3 + X2 + X1 + b^10 + b^9 + b^6 + b^5 + b^4,
 X2^2 + X2*X1 + (b^10 + b^9 + b^6 + b^5 + b^4)*X2 + X1^2 +
  (b^10 + b^9 + b^6 + b^5 + b^4)*X1 +
   b^9 + b^8 + b^6 + b + 1,
 X1^3 + (b^10 + b^9 + b^6 + b^5 + b^4)*X1^2 +
  (b^9 + b^8 + b^6 + b + 1)*X1 + b^8 + b^6 + b
]
Dus is dit het error-locator polynoom:
X^3 + (b^10 + b^9 + b^6 + b^5 + b^4)*X^2 +
 (b^9 + b^8 + b^6 + b + 1)*X + b^8 + b^6 + b
De errorvector is:
(0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0)
De oorspronkelijke boodschap was dus:
(0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1)
De totale rekentijd van het programma is:
0.180
```

6.6 Bijlage 6: Eénstapspolynomen

6.6.1 Berekening van het polynoom voor G_{11}

```
t:=Cputime(0.00);

//Stelsel invoeren en Groebner basis berekenen:
FF<Y2,Y1,X2,S,X1>:=PolynomialRing(GF(3),5);
Stelsel:=[Y1*X1+Y2*X2-S,Y1^3-Y1,Y2^3-Y2,X1^11-1,X2^11-1];
I:=ideal<FF|Stelsel>;
Gr:=GroebnerBasis(I);

//Polynomen uit Gr over F3(S)[X] bekijken:
G<S>:=RationalFunctionField(GF(3));
F<X>:=PolynomialRing(G);
h:=hom< FF -> F | 0,0,0,S,X >;
f:=h(Gr[#Gr]);g:=h(Gr[#Gr-1]);

//Maak een rijtje met alle coefficienten van alle ggdpolynomen en
//bijbehorende syndromen bestaande uit 2 eenheidswortels
//(4 syndromen per 2 eenheidswortels
//vanwege de lineaire veelvouden):
Pol<Y>:=PolynomialRing(GF(3,5));
cf:=Coefficients(f);
cg:=Coefficients(g);
Syndromen:={{i,j} : i,j in [1..11] | #{i,j} eq 2 };
Rijtje:=[];
for j in Syndromen do
        J:=SetToSequence(j);
        for k in [[1,1],[1,2],[2,1],[2,2]] do
                syn:=k[1]*a^(J[1])+k[2]*a^(J[2]);
                df:=[Evaluate(i,syn) : i in cf];
                polf:=Pol ! df;
                dg:=[Evaluate(i,syn) : i in cg];
                polg:=Pol ! dg;
                ggd:=Gcd(polf,polg);
                Rijtje:=Rijtje cat [<syn,Coefficients(ggd)>];
        end for;
end for;

//Nu nog de syndromen bestaande uit 1 eenheidswortel en
//bijbehorende polynomen toevoegen (en het lineaire veelvoud):
for i in [1..11] do
        Rijtje:=Rijtje cat [<a^i,[-a^i,1]>];
        Rijtje:=Rijtje cat [<2*a^i,[-a^i,1]>];
end for;
```

```
//Coefficientrijtjes van lengte 3 maken:
for j in [1..#Rijtje] do
        if #(Rijtje[j][2]) ne 3 then
                for i in [1..(3-#(Rijtje[j][2]))] do
                        Rijtje[j][2]:=Rijtje[j][2] cat [0];
                end for;
        end if;
end for;

//Gaan nu bij dit rijtje polynomen U(S), V(S) en W(S) zoeken met:
//U(S)*X^2+V(S)*X+W(S) voor iedere S ingevuld is bijbehorende
//ggd die hierboven is berekend.
k:=2*2*Binomial(11,2)+2*11;
mat:=Matrix([ [Rijtje[j][1]^(k-i) : i in [1..k]]: j in [1..k]]);

//Eerst U(S):
//242 punten (S,U(S))=(Rijtje[j][1],Rijtje[j][2][3]);
vecu:=Vector([ Rijtje[j][2][3] : j in [1..k]]);
oplu:=Solution(Transpose(mat),vecu);
U:= G ! [oplu[k+1-j] : j in [1..k]];

//Nu V(S):
vecv:=Vector([ Rijtje[j][2][2] : j in [1..k]]);
oplv:=Solution(Transpose(mat),vecv);
V:= G ! [oplv[k+1-j] : j in [1..k]];

//Nu W(S):
vecw:=Vector([ Rijtje[j][2][1] : j in [1..k]]);
oplw:=Solution(Transpose(mat),vecw);
W:= G ! [oplw[k+1-j] : j in [1..k]];

"De rekentijd van het preprocesseren is:";
print Cputime(t);
```

6.6.2 Het polynoom voor $\mathbf{G}_{11}$

```
// Polynoom U(S)*X^2+V(S)*X+W(S) met:

G<S>:=RationalFunctionField(GF(3));
F<X>:=PolynomialRing(G);

U:=S^220+S^198+S^176+S^154+S^132+S^110+S^88+S^66+S^44+S^22+2;

V:=2*S^220+2*S^198+2*S^176+2*S^154+2*S^144+2*S^132+2*S^110+
   S^100+2*S^88+2*S^66+2*S^44+2*S^34+2*S^22+S^12+2;

W:=S^232+S^210+2*S^200+S^188+S^178+S^166+2*S^156+S^144+
   2*S^134+S^122+S^100+S^90+S^78+2*S^68+S^56+2*S^46+
   S^34+S^24+S^12+2*S^2;
```

6.6.3 Berekening van het polynoom voor $\mathbf{G}_{23}$

```
t:=Cputime(0.00);

//Stelsel invoeren en Groebner basis berekenen:
FF<X3,X2,S,X1>:=PolynomialRing(GF(2),4);
Stelsel:=[X3+X2+X1+S,X3^24+X3,X2^24+X2,X1^24+X1];
I:=ideal<FF|Stelsel>;
Gr:=GroebnerBasis(I);

//Polynomen uit Gr over F2(S)[X] bekijken:
G<S> := RationalFunctionField(GF(2));
F<X>:=PolynomialRing(G);
h:=hom< FF -> F | 0,0,S,X >;
f:=h(Gr[#Gr]);
g:=h(Gr[#Gr-1]);

//Maak een rijtje met alle coefficienten van alle ggdpolynomen en
//bijbehorende syndromen bestaande uit 2 of 3 eenheidswortels:
Pol<Y>:=PolynomialRing(GF(2,11));
cf:=Coefficients(f);
cg:=Coefficients(g);
Syndromen:={{i,j,k} : i,j,k in [1..23] |#{i,j,k} ge 2 };
Rijtje:=[];
for j in Syndromen do
        syn:=0;
        for i in j do
                syn:=syn+a^i;
        end for;
        df:=[Evaluate(i,syn) : i in cf];
        polf:=Pol ! df;
        dg:=[Evaluate(i,syn) : i in cg];
        polg:=Pol ! dg;
        ggd:=Gcd(polf,polg);
        Rijtje:=Rijtje cat [<syn,Coefficients(ggd)>];
end for;

//Nu nog de syndromen bestaande uit 1 eenheidswortel
//en bijbehorende polynomen toevoegen:
for i in [1..23] do
        Rijtje:=Rijtje cat [<a^i,[-a^i,1]>];
end for;

//Coefficientrijtjes van lengte 4 maken:
for j in [1..#Rijtje] do
        if #(Rijtje[j][2]) ne 4 then
                for i in [1..(4-#(Rijtje[j][2]))] do
                        Rijtje[j][2]:=Rijtje[j][2] cat [0];
```

```
                end for;
        end if;
end for;

//Gaan nu bij dit rijtje polynomen T(S), U(S), V(S) en W(S)
//zoeken met:
//T(S)*X^3+U(S)*X^2+V(S)*X+W(S) voor iedere S ingevuld is
//bijbehorende ggd die hierboven is berekend.
k:=Binomial(23,3)+Binomial(23,2)+23;
mat:=Matrix([ [Rijtje[j][1]^(k-i) : i in [1..k]]: j in [1..k]]);

//Eerst T(S):
//2047 punten (S,U(S))=(Rijtje[j][1],Rijtje[j][2][3]);
vect:=Vector([ Rijtje[j][2][4] : j in [1..k]]);
oplt:=Solution(Transpose(mat),vect);
T:= G ! [oplt[k+1-j] : j in [1..k]];

//Dan U(S):
vecu:=Vector([ Rijtje[j][2][3] : j in [1..k]]);
oplu:=Solution(Transpose(mat),vecu);
U:= G ! [oplu[k+1-j] : j in [1..k]];

//Nu V(S):
vecv:=Vector([ Rijtje[j][2][2] : j in [1..k]]);
oplv:=Solution(Transpose(mat),vecv);
V:= G ! [oplv[k+1-j] : j in [1..k]];

//Nu W(S):
vecw:=Vector([ Rijtje[j][2][1] : j in [1..k]]);
oplw:=Solution(Transpose(mat),vecw);
W:= G ! [oplw[k+1-j] : j in [1..k]];

"De rekentijd van het preprocesseren is:";
print Cputime(t);
```

6.6.4 Het polynoom voor $\mathbf{G}_{23}$

```
// Polynoom T(S)*X^3+U(S)*X^2+V(S)*X+W(S) met:

G<S> := RationalFunctionField(GF(2));
F<X>:=PolynomialRing(G);

T:=S^1840+S^1633+S^1610+S^1564+S^1426+S^1380+S^1288+S^1219+
   S^1196+S^1173+S^1104+S^1081+S^1058+S^920+S^805+S^782+S^713+
   S^690+S^644+S^598+S^552+S^529+S^460+S^391+S^345+S^322+S^299+
   S^276+S^230+S^161+S^138+S^115+S^69+1;

U:=S^2024+S^2001+S^1978+S^1955+S^1932+S^1909+S^1886+S^1863+
   S^1841+S^1817+S^1794+S^1771+S^1748+S^1725+S^1702+S^1679+
   ......................................................
   .....................................................+
   S^897+S^874+S^851+S^828+S^806+S^783+S^759+S^736+S^714+S^691+
   S^667+S^645+S^621+S^599+S^575+S^553+S^530+S^506+S^483+S^461+
   S^437+S^414+S^392+S^368+S^346+S^323+S^300+S^277+S^253+S^231+
   S^207+S^184+S^162+S^139+S^116+S^92+S^70+S^46+S^23+S+1;

V:=S^2026+S^2025+S^2024+S^2003+S^2002+S^2001+S^1980+S^1979+
   S^1978+S^1957+S^1956+S^1955+S^1934+S^1933+S^1932+S^1911+
   S^1910+S^1909+S^1887+S^1886+S^1864+S^1863+S^1840+S^1818+
   S^1817+S^1796+S^1795+S^1794+S^1773+S^1772+S^1771+S^1750+
   S^1749+S^1748+S^1727+S^1726+S^1725+S^1704+S^1703+S^1702+
   ......................................................
   .....................................................+
   S^622+S^621+S^600+S^598+S^576+S^575+S^554+S^552+S^529+
   S^508+S^507+S^506+S^484+S^483+S^460+S^439+S^438+S^437+
   S^416+S^415+S^414+S^391+S^369+S^368+S^345+S^322+S^299+
   S^276+S^255+S^254+S^253+S^230+S^209+S^208+S^207+S^186+
   S^185+S^184+S^163+S^161+S^138+S^115+S^94+S^93+S^92+S^71+
   S^69+S^48+S^47+S^46+S^24+S^23+S^2+S+1;

W:=S^2027+S^2025+S^2004+S^2002+S^1981+S^1979+S^1958+S^1956+
   S^1935+S^1933+S^1912+S^1910+S^1889+S^1888+S^1887+S^1866+
   S^1865+S^1864+S^1843+S^1842+S^1841+S^1820+S^1819+S^1818+
   S^1797+S^1795+S^1774+S^1772+S^1751+S^1749+S^1728+S^1726+
   S^1705+S^1703+S^1682+S^1681+S^1680+S^1659+S^1657+S^1636+
   ......................................................
   ....................................................+
   S^576+S^555+S^553+S^530+S^509+S^507+S^486+S^485+S^484+
   S^463+S^462+S^461+S^440+S^438+S^416+S^415+S^392+S^371+
   S^370+S^369+S^348+S^347+S^346+S^323+S^302+S^301+S^300+
   S^279+S^278+S^277+S^254+S^233+S^232+S^231+S^210+S^208+
   S^187+S^185+S^163+S^162+S^141+S^140+S^139+S^118+S^117+
   S^116+S^95+S^93+S^71+S^70+S^48+S^47+S^24+S;
```

7 Literatuurlijst

1. Cohen, Arjeh M.; Cuypers, Hans; Sterk, Hans (Eds.): *Some tapas of computer algebra*. Berlin: Springer, 1999.

2. Cameron, P.J.; Lint, J.H. van: *Designs, graphs, codes and their links.* Cambridge: Cambridge University Press, 1991.

3. MacWilliams, Jessie; Sloane, Neil: *The theory of error-correcting codes.* Amsterdam: North-Holland Publishing Co., 1977.

4. Bosma, W.; Cannon, J.J.; Playoust, C.: *The Magma Algebra System I: The user language.* Journal of Symbolic Computation, vol. 24, pagina 235-265, 1997.