Universiteit Gent
Faculteit Wetenschappen
Vakgroep Zuivere Wiskunde en
Computeralgebra

Verdere ontwikkelingen en toepassingen van het gebruik van GAP in eindige meetkunde

Anja Hallez

Promotor: Dr. J. De Beule

Scriptie ingediend tot het behalen van de academische graad
van licentiaat wiskunde, optie zuivere wiskunde

Academiejaar 2005–2006

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 GAP

GAP (**G**roups, **A**lgorithms and **P**rogramming) is een systeem voor computationele discrete algebra met de nadruk op computationele groepentheorie. Het bestaat uit een programmeertaal, een bibliotheek met functies en een grote databibliotheek met objecten. **GAP** wordt gebruikt als ondersteuning bij onderzoek en onderwijs. Het systeem, inclusief de code, wordt gratis gedistribueerd, zodat het gemakkelijk bestudeerd, gewijzigd en uitgebreid kan worden, naargelang de noden van je gebruik. Dit heeft geleid tot een aantal shared packages, ontwikkeld door gebruikers, die samen met **GAP** verdeeld worden. We beschikken over zo'n **GAP**-pakket **PG** (**P**rojective **G**eometries) ontwikkeld door Patrick Govaerts en Jan De Beule.

Op 26 september 2005 verscheen **GAP4r6**, de zesde release van **GAP4**. In maart 2006 verscheen een volgende release. Het bestaande pakket werd uiteraard ontwikkeld onder een vorige versie, maar er deden zich geen problemen voor bij de overstap. We hebben alle verdere ontwikkelingen aan het pakket gedaan met **GAP4r6**. Het is mogelijk dat bepaalde uitbreidingen niet werken onder een vorige release van **GAP**. Iedereen kan echter gratis de nieuwste versie van **GAP** downloaden van de website:

www.gap-system.org

Het installeren van het pakket is eenvoudig en de volledige uitleg over de installatie is ook te vinden op de website. Eens **GAP** geïnstalleerd en opgestart is, moeten de pakketten eerst geladen worden vooraleer je ze kunt gebruiken. Dit kan je doen met het volgende commando:

`LoadPackage("` *package*`")`

Een diepgaande uitleg over de technische details van **GAP** zou ons te ver leiden. Hiervoor verwijzen we naar de reference manual via de website of naar de thesis van Patrick Govaerts [4]. In de volgende paragrafen lichten we toch de belangrijkste aspecten toe die nodig zijn voor een goed begrip van het vervolg van de scriptie.

1.2 Notaties

We volgen de notatieconventies aangenomen in [4].

text
boldface: dient om woorden te benadrukken, zoals nieuwe begrippen.

text
italics: wordt gebruikt voor argumenten in de beschrijving van functies. Deze tekst moet door de gebuiker niet ingegeven worden in **GAP**, maar dient vervangen te worden door een geschikte tekst, afhankelijk van wat men wil doen.

`text`
verbatim: wordt gebruikt voor namen van variabelen, functies en andere tekst die zo letterlijk in de computer ingegeven moeten worden en ook zo op het scherm zal verschijnen. De tekst kan uitdrukkingen in italics bevatten.

`Oper(`*arg1, arg2[, opt]*`) F`

Gaat over het commando `Oper` dat twee argumenten *arg1* en *arg2* en een optioneel derde argument *opt* heeft. Hier moet je de strings "Oper(" en ")" letterlijk zo intikken, terwijl de argumenten moeten vervangen worden door een gepaste uitdrukking of waarde. De letter `F` op het einde van de lijn geeft aan dat het om een functie gaat. Andere mogelijkheden zijn `A, P, O, C, R`. Zij betekenen respectievelijk "Attribuut", "Eigenschap", "Operatie", "Categorie" en "Representatie".

1.3 Types van objecten

Een **object** in **GAP** is alles wat kan toegekend worden aan een variabele en omgekeerd is alles wat kan toegekend worden aan een variabele een object. Bijna alles is dus een object in **GAP**. Bijvoorbeeld: een getal, een matrix, een projectieve ruimte, ... Commentaar en syntactische constructies, zoals lussen, zijn voorbeelden van dingen die geen objecten zijn.

Elk **GAP**-object heeft een **type**. Types worden gebruikt om te beslissen of een operatie toegelaten of mogelijk is, en indien dit zo is, hoe die operatie uit te voeren. Het type van een object bestaat uit verschillende onderdelen (familie, categorie, representatie, attributen en eigenschappen), die verschillende kenmerken van het object beschrijven.

`TypeObj(` *obj* `) F`

geeft het type van het object *obj* weer. Twee types zijn gelijk als en slechts als alle onderdelen van de twee types gelijk zijn.

1.3.1 Families

De **familie** (*family*) bepaalt de relatie van het object tot andere objecten.
Bijvoorbeeld alle deelruimten van eenzelfde projectieve ruimte liggen in dezelfde familie. De families vormen een partitie van alle **GAP**-objecten, zodanig dat de volgende voorwaarden voldaan zijn: objecten die gelijk zijn met betrekking tot `"="` liggen in dezelfde familie en de familie van het resultaat van een operatie is enkel afhankelijk van de families van de argumenten.

`FamilyObj(` *obj* `) F`

geeft de familie van het object *obj* weer.

1.3.2 Categorieën

De **categorieën** (*categories*) bepalen welke operaties een object toelaat. Bijvoorbeeld alle kwadrieken vormen een categorie, evenals alle projectieve punten. Een object dat tot een welbepaalde categorie behoort, moet methoden hebben voor zekere operaties. Zo moet bijvoorbeeld een object gelegen in de categorie `IsList`, methoden hebben die toelaten de lengte te berekenen.

Een object kan in meerdere categorieën liggen. Elk object ligt in de categorie `IsObject`. Een ingebedde deelruimte behoort tot de categorie `IsEmbeddedPG`, dat een deelcategorie is van `IsPG`, dat op zijn beurt deel uitmaakt van de categorie `IsObject`.
De verzameling van categorieën heeft een boomstructuur waarbij `IsObject` de top van de boom is en elk kind is een deelcategorie van zijn ouder.

De categorie dient om te bepalen welke operaties toegelaten zijn. Elke operatie eist dan ook dat elk van zijn argumenten tot een bepaalde categorie behoren. Dit is echter geen garantie dat voor elke verzameling argumenten uit deze categorieën een resultaat bekomen wordt. De operatie die twee matrices vermenigvuldigt eist dat de twee argumenten in de categorie `IsMatrix` liggen. Het is echter onmiddellijk duidelijk dat dit niet garandeert dat deze twee matrices ook vermenigvuldigd kunnen worden.

1.3.3 Representaties

De **representatie** (*representation*) bepaalt hoe een object inwendig wordt weergegeven. Een projectieve deelruimte kan voorgesteld worden als span van een stel projectieve punten of als doorsnede van een aantal hypervlakken. Door gebruik te maken van representaties kunnen we een onderscheid maken tussen die twee.

GAP onderscheidt vier essentieel verschillende manieren om objecten te representeren. Ten eerste hebben we de representatie `IsInternalRep` voor interne objecten zoals gehele getallen en permutaties. `IsDataObjectRep` dient om andere objecten waarvan de data enkel toegankelijk

is voor kernfuncties te representeren. Alle andere objecten behoren ofwel tot de representatie `IsComponentObjectRep` ofwel tot de representatie `IsPositionalObjectRep`.
Een object uit de representatie `IsComponentObjectRep` of een deelrepresentatie daarvan wordt een **componentobject** genoemd. Zulke objecten *obj* worden opgebouwd uit subobjecten die opgevraagd kunnen worden via *obj!.name*, zoals de componenten van een record. Er kunnen ook waarden toegekend worden aan de componenten via *obj!.name:=val*. Een **positioneel object** is een object uit de representatie `IsPositionalObjectRep` of een subrepresentatie. De subobjecten van zulk object kunnen opgevraagd worden door *obj![pos]*, zoals de posities van een lijst. Ook hier kunnen waarden toegekend worden aan de posities via *obj![pos]*.

De subrepresentaties van deze laatste twee vormen bomen, waardoor twee representaties dus ofwel disjunct zijn, ofwel is de één volledig bevat in de andere. Een belangrijke subrepresentatie van `IsComponentObjectRep` is `IsAttributeStoringRep`. Een object in deze representatie slaat alle attributenwaarden automatisch op eens ze berekend worden.

Objecten kunnen van representatie veranderen. Zo kan een projectieve ruimte die als span van punten gerepresenteerd wordt, omgezet worden in een representatie die de doorsnede van hypervlakken voorstelt.

1.3.4 Attributen

De **attributen** (*attributes*) beschrijven kennis over het object. Zoals bijvoorbeeld de generatoren van een kwadriek.

Een attribuut is een unaire bewerking.

Indien een object tot de representatie `IsAttributeStoringRep` behoort dan wordt bij elke eerste oproep van een attribuut van dat object, de waarde berekend en bewaard. Bij volgende oproepen van datzelfde attribuut wordt die bijgehouden waarde zonder verdere berekeningen weergegeven.

`KnownAttributesOfObject(` *obj* `) O`

geeft een lijst weer met de namen van alle attributen waarvan de waarde gekend is.

Voor elk attribuut zijn er twee operaties gedefinieerd, de **attribuutplaatser** en de **attribuuttester**. Stel dat voor een object *obj* een attribuut *attr* gedeclareerd is. Als de waarde van *attr* ergens als tussenresultaat van een berekening verkregen wordt, dan hebben we via de plaatser de mogelijkheid om deze waarde op te slaan.

`Setter(` *attr* `) O`

Als de attribuutwaarde voor het eerst berekend wordt dan wordt deze plaatser automatisch opgeroepen. Met de oproep `Setter(` *attr* `)(`*obj, val* `)` kan men de waarde *val* expliciet toekennen aan het object *obj*.

De situatie kan op een bepaald moment zo zijn dat de berekening van een attribuut tijdrovend is, maar waarbij de kennis van dat attribuut kan leiden tot een sterke vereenvoudiging van verder rekenwerk. Als we via een tester weten dat dit attribuut al gekend is, kunnen we het zonder berekeningen oproepen en gebruiken. Indien we merken dat dit attribuut nog niet gekend is zouden we kunnen besluiten dat het beter is om de berekening van dit attribuut te omzeilen.

`Tester(` *attr* `) O`

gaat na of de waarde van het attribuut *attr* voor een object gekend is. Elk attribuut dat gedeclareerd werd met `DeclareAttribute` wordt automatisch toegankelijk gemaakt onder de namen `Has`*attr* en `Set`*attr*.

1.3.5 Eigenschappen

De **eigenschappen** (*properties*) zijn die attributen die als mogelijke waarden `true` of `false` hebben.

Een eigenschap definieert twee verzamelingen van objecten: enerzijds de verzameling van objecten waarvoor de waarde van die eigenschap gekend is en anderzijds de verzameling van alle objecten waarvan geweten is dat de waarde `true` is voor die eigenschap. Het bevat zijn in de eerste verzameling kunnen we testen via `Tester(`*prop*`)(`*obj*`)`. Om zeker te zijn dat een object tot de tweede verzameling behoort, moeten we bijkomend nog nagaan dat *prop(obj)* ook `true` weergeeft.

1.3.6 Filters

Elke categorie, representatie of attribuuttester kan geïdentificeerd worden met de verzameling van alle objecten die in de respectievelijke categorie, representatie of attribuut liggen. Voor eigenschappen is niet alleen de tester, maar ook de eigenschap zelf zo'n verzameling, namelijk de verzameling van alle objecten waarvan geweten is dat de eigenschap de waarde `true` heeft. De verschillende onderdelen van het type van een object bepalen een veelheid aan verzamelingen die we allen kunnen bundelen onder de noemer **filters**. Filters zijn speciale **GAP**-functies die `true` of `false` geven, al naargelang het argument wel of niet in de verzameling ligt, die gedefinieerd is door de filter.

De intersectie van twee filters is opnieuw een filter, die bepaald wordt door *filt1* `and` *filt2*.
Een voorbeeld: `IsSubspaceOfPG and IsSubspaceIntersRep` is een filter. Een filter die geen intersectie vormt van een aantal filters, noemt men een **simpele filter**. Elke filter heeft een rangorde die gebruikt wordt om de volgorde te bepalen waarin de methodes van een operatie zullen toegepast worden.

`RankFilter(`*filt* `) F`

geeft de rang weer van de filter *filt*.
Zonder in detail te treden kunnen we stellen dat de rang van een simpele filter bepaald wordt bij de constructie van die filter, terwijl dit bij andere filters gebeurt door de som van de rangordes van de betrokken simpele filters.

Aan de hand van filters kan men bepalen welke methodes toepasbaar zijn op een object.

1.4 Nieuwe objecten creëren

`Objectify(` *type, data* `)`

maakt een nieuw object aan, waarbij *data* een lijst of een record is en *type* het type van het object. Als *data* een lijst is, maakt `Objectify` er een positioneel object van, anders wordt er een componentobject van gemaakt. Het type aanmaken gaat als volgt:

`NewType(` *fam, filt, [data]* `)`

waarbij de familie en de filter nodig zijn. Om de verschillende onderdelen van het type aan te maken, kunnen we gebruik maken van twee werkwijzen, nl. `New`*Iets* of `Declare`*Iets*. Meestal zullen we de laatste werkwijze gebruiken, waarbij de variabelen read-only aangemaakt worden. Voor meer informatie hieromtrent verwijzen we naar [4] of naar de website van **GAP**.

1.5 Operaties en methodes

Nu we voor een stuk de werking van **GAP** verklaard hebben, zullen we enkele aspecten bespreken om **GAP** met nieuwe functies uit te breiden.

Operaties zijn **GAP**-functies, zodat ze kunnen toegepast worden op argumenten en een resultaat weergeven of een neveneffect veroorzaken. Maar operaties zijn meer dan dat. Een operatie komt overeen met een verzameling **GAP**-functies die we haar methodes noemen. Alle methodes van een bepaalde operatie berekenen hetzelfde resultaat, maar elke methode is geschreven voor een specifiek type van argumenten. Afhankelijk van het type van de argumenten wordt bij het oproepen van een operatie één van haar toepasbare methodes gekozen en uitgevoerd.

Attributen en eigenschappen zijn ook operaties. De eerste stap in het aanmaken van een nieuwe operatie is het declareren.

`DeclareOperation(` *name, args-filts* `)`
`DeclareAttribute(` *name, filt* `)`
`DeclareProperty(` *name, filt* `)`

declareren respectievelijk een operatie, attribuut of eigenschap *oper* met naam *name*, waarbij de laatste twee unaire operaties zijn en dus maar één filter hebben. We zullen de verdere uitleg over de selectie van een methode voor een operatie beperken tot het algemene geval van een gewone operatie.

`InstallMethod(` *oper, info, famp, args-filts, val, method* `)`

installeert een functie *method* als methode voor de operatie *oper*. *Info* is hierbij een optioneel argument dat informatie weergeeft over de situatie waarin de methode toegepast wordt. *famp* moet een functie bevatten die toegepast wordt op de families van de argumenten, *args-filts* is een lijst met voorwaarden op de argumenten, waarbij elk element van de lijst een filter is. *val* is een geheel getal dat de prioriteit van de methode aangeeft. Dit wordt gevolgd door de eigenlijke methode.

Opdat een methode toepasbaar zou zijn moet het argumentental aan de volgende voorwaarden voldoen: het aantal argumenten is gelijk aan de lengte van de lijst *args-filts*, het i-de argument ligt in de filter *args-filts[i]* en *famp* geeft `true` weer wanneer het toegepast wordt op de families van de argumenten. Daar de familie van een object de relatie tot andere objecten bepaalt, beschrijft *famp* dus relaties tussen de argumenten.

Voor unaire operaties is het opstellen van dergelijke relaties zinloos en dan nemen we voor *famp* de operatie `ReturnTrue` of kortweg gewoon `true`. Voor binaire operaties wordt meestal `IsIdenticalObj` gebruikt, wat betekent dat beide argumenten in dezelfde familie moeten liggen.

Als er geen toepasbare methode is, dan wordt de foutmelding `no method found` weergegeven. Indien dit wel het geval is, wordt de methode met de hoogste rangorde geselecteerd en opgeroepen. De rangorde wordt bekomen door de som van de rangordes van de filters in de lijst *args-filts* te vermeerderen met het getal *val*. Het argument *val* kan dus gebruikt worden om de prioriteit van de methode ten opzichte van andere methodes te verhogen. Dit heeft tot doel dat we de efficiënste methode, dit is de methode die het minste tijd en geheugencapaciteit nodig heeft, de hoogste prioriteit geven.

Het kan voorkomen dat na het oproepen van een methode blijkt dat deze methode het gevraagde resultaat niet kan berekenen en opgeeft door `TryNextMethod()` op te roepen. Dit is eigenlijk een stopzetting van de methode en de methodeslectie gaat op zoek naar de volgende methode die toepasbaar is op de oorspronkelijke argumenten en dit volgens dalende rangordes.

Stel dat we een operatie *oper* hebben met als vereiste argumenten *args-filts*. We kunnen de restricties op de argumenten omzeilen door een methode te installeren via `InstallOtherMethod`.

We zullen dit alles aan de hand van een voorbeeld verduidelijken. We hebben projectieve deelruimten en daarop hebben we de operatie `Dimension` als volgt gedeclareerd

```
DeclareAttribute(" Dimesion", IsSubspaceOfPG );
```

We hebben in de filter van dit attribuut niets gezegd over de representatie van de projectieve deelruimten. We kunnen nu naar believen objecten *obj* definieren in de categorie `IsSubspaceOfPG` met verschillende representaties. Bij het schrijven van de nieuwe methodes geven we dan aan voor welke representatie deze methode geschikt is. De oproep

`Dimension(`*obj* `)`

zal dan automatisch die nieuwe methode oproepen door het gepast gebruik van de filters. Willen we nu ook de dimensie van projectieve punten vastleggen, dan moeten we gebruik maken van `InstallOtherMethod`, omdat projectieve punten niet tot de filter `IsSubspaceOfPG` behoren.

1.6 Uitbreidingen van GAP en functies

Het doel van deze scriptie is het pakket **PG** uit te breiden. We zullen beginnen met het bespreken van de algemene werkwijze van dergelijke uitbreidingen. Een uitbreiding begint vaak, maar zeker niet noodzakelijk, bij de constructie van een nieuw object *obj*. We willen er nu ook voor zorgen dat er operaties toepasbaar zijn op *obj*. Als we nieuwe operaties declareren zullen we de filters zo algemeen mogelijk houden. Dit kan onder andere door geen restricties te leggen op de representatie van de objecten. Bij het installeren van de methodes voor die operatie zullen de filters veel strikter zijn. Op die manier is het steeds mogelijk om nieuwe objecten te creëren met een andere representatie of in een deelcategorie waarop de operaties nog steeds toepasbaar zijn.

Bijvoorbeeld: We hebben het attribuut `Generators` gedeclareerd voor objecten in de categorie `IsVariety`. Hierdoor kunnen we dit gebruiken voor de kwadrieken en hermitische variëteiten onafhankelijk van hun representatie. Als we in de toekomst andere variëteiten toevoegen aan het pakket zal de operatie `Generators` automatisch ook toepasbaar zijn op de nieuwe variëteiten.

We hebben al meerdere voorbeelden gegeven van de voordelen van het gebruik van operaties. We hadden in elk van die gevallen kunnen opteren om een gewone functie te schrijven. Maar hadden we een functie `Generators` geschreven die alle generatoren van een kwadriek bepaalt, dan konden we geen nieuwe methode schrijven voor die functie, maar moesten we voor elke nieuwe variëteit een andere functie, met een andere naam declareren. Dit zou tot gevolg hebben dat de oproep `Generators(`*obj* `)` enkel werkt indien *obj* een kwadriek is. Dit beperkt niet alleen de uitbreidingsmogelijkheden, maar maakt het pakket ook een stuk gebruiksonvriendelijker. Bij het gebruik van operaties dient de gebruiker maar één naam te onthouden en kan hij die toepassen op alle objecten waarvoor de operatie nuttig is.

Gewone functies bestaan echter niet voor niets. Het kan zijn dat er in verschillende methodes steeds een aantal handelingen dienen herhaald te worden. Stel dat bijvoorbeeld telkens dezelfde coördinatentransformatie moet doorgevoerd worden en dat in meerdere methodes. In dat geval is het nuttig om deze berekeningen in een aparte functie te zetten, zodat we die maar eenmaal hoeven te installeren. Doordat deze hulpfunctie telkens vanuit een methode opgeroepen wordt, zijn we zeker dat aan de voorwaarden voor de parameters voldaan is. Dit maakt het onnodig om

hiervoor een operatie te schrijven die nog eens een controle doet op de argumenten. Dergelijke hulpfuncties zijn over het algemeen niet bedoeld voor de gebruiker.
We kunnen ook globale functies schrijven die wel voor de gebruiker bedoeld zijn. Zo hebben we in het pakket de functie `ProjectiveGeometry`, die als argumenten n en q heeft en $\mathrm{PG}(n, q)$ teruggeeft. Doordat we aan de hand van n en q niet kunnen uitmaken in welke representatie de projectieve ruimte moet liggen, is het duidelijk dat we geen andere methodes nodig hebben die nieuwe representaties introduceren. Als we eventueel een andere representatie willen, dan zouden we hoe dan ook een nieuwe functie moeten schrijven.

`DeclareGlobalFunction(` *name* `) F`

Declareert een globale functie *name*.

`InstallGlobalFunction(` *name, func* `) F`

kent de functie *func* toe aan de globale functie gedeclareerd als *name*.

1.7 Veranderlijke versus onveranderlijke objecten

Twee objecten die niet enkel gelijk zijn als objecten, maar die daadwerkelijk naar dezelfde geheugenplaats verwijzen worden **identiek** genoemd. Normaalgesproken maken we geen onderscheid tussen beide soorten gelijkheid, maar volgend voorbeeld maant ons toch aan tot voorzichtigheid.

```
gap> l1:=[1,2,3];l2:=l1;
[ 1, 2, 3 ]
[ 1, 2, 3 ]
gap> l1[3]:=30;
30
gap> l1;l2;
[ 1, 2, 30 ]
[ 1, 2, 30 ]

gap> l1:=[1,2,3];l2:=[1,2,3];
[ 1, 2, 3 ]
[ 1, 2, 3 ]
gap> l1[3]:=30;
30
gap> l1;l2;
[ 1, 2, 30 ]
[ 1, 2, 3 ]
```

In het eerste deel zijn beide lijsten identiek, zodat elke verandering in het ene object zich ook uit in het andere object, want de verandering is gebeurd op de geheugenplaats waarnaar `l1`

verwijst, maar `l2` komt met dezelfde plaats overeen en verandert dus mee. In het tweede deel zijn de lijsten wel gelijk, maar niet identiek, waardoor de verandering niet in beide lijsten te zien is.

In hetvoorgaande hebben we de waarden in de lijsten gewijzigd wat erop wijst dat deze objecten **veranderlijke objecten** zijn. Dit is op vele plaatsen nuttig, maar er zijn ook situaties waarin we niet willen dat bepaalde objecten veranderen. Een eenvoudig voorbeeld zijn de eenheidselementen van groepen, zoals van matrices. We willen er zeker van zijn dat deze matrix na verloop van tijd nog altijd de eenheidsmatrix voorstelt.

`Immutable(` *obj* `) O`

geeft een onveranderlijke structurele kopie van *obj* weer waarin de subobjecten onveranderlijke kopieën zijn van de subobjecten van *obj*. Wanneer we toch proberen een onveranderlijk object te veranderen, krijgen we een foutmelding.

Het voorgaande voorbeeld kunnen we als volgt aanpassen.

```
gap> l1:=[1,2,3];l2:=l1;
[ 1, 2, 3 ]
[ 1, 2, 3 ]
gap> l2:=Immutable(l2);
[ 1, 2, 3 ]
gap> IsMutable(l1);IsMutable(l2);
true
false
gap> IsIdenticalObj(l1,l2);
false
gap> l2[2]:=3;
Lists Assignment: <list> must be a mutable list
not in any function
Entering break read-eval-print loop ...
you can 'quit;' to quit to outer loop, or
you can 'return;' and ignore the assignment to continue
```

We zien dat we inderdaad een foutmelding krijgen als we de nu onveranderlijke lijst $l2$ proberen te wijzigen. Door een van beide lijsten onveranderlijk te maken, zijn beide lijsten ook niet meer identiek.

Indien zowel een veranderlijke als een onveranderlijke vorm van een object bestaat, dan wordt het object **kopieerbaar** genoemd. Voorbeelden zijn lijsten en records.

`ShallowCopy( obj ) O`

geeft een veranderlijke kopie weer van *obj*, waarbij de subobjecten dezelfde zijn als van *obj*. Is *obj* een onveranderlijke matrix dan zal het resultaat een veranderlijke matrix zijn, waarvan de rijen nog steeds onveranderlijk zullen zijn. Dit kan met de volgende oproep verholpen worden: `List( obj, ShallowCopy )`

Het resultaat van sommige operatoren is onveranderlijk. Zo hebben we vanzelfsprekend `One`, `Zero`, `Inverse` en ook de infix operaties `+,-,*,/,^,mod`. Een ander voorbeeld is `TransposedMat`. In algoritmen kan het echter wenselijk zijn om de ingangen van een matrix te manipuleren. Hiervoor moet de matrix veranderlijk zijn, maar doordat het resultaat van vele operaties die toepasbaar zijn op matrices onveranderlijk is, is dit dus steeds een probleem. Voor zulke situaties bestaan een aantal methoden waarvan het resultaat wel veranderlijk is. Zo hebben we `MutableTransposedMat`.

Dit alles vereist ook aandacht bij het gebruik van functies. We stelden reeds dat objecten alles zijn wat toegekend kan worden aan een variabele. Een variabele heeft een naam, die binnen een bepaald geldigheidsgebied naar een unieke variabele verwijst. We onderscheiden een **globaal** geldigheidsgebied, dat de ganse programmatekst beslaat en een **lokaal** dat begint vanaf het sleutelwoord `function` tot aan het corresponderende sleutelwoord `end`. Een functie introduceert nieuwe variabelen via de argumentenlijst en de `local` declaratie. Deze variabelen zijn allen lokaal.
Bij een oproep van een functie worden de meegegeven argumenten vergeleken met de argumenten in de declaratie van de functie, waarbij het aantal gelijk moet zijn. Bij uitvoering wordt een nieuwe variabele voorzien voor zowel de argumenten als de namen gedeclareerd in de `local` van de functie. Daarna worden de argumenten geëvalueerd en wordt de waarde toegekend aan de nieuwe variabele, die hierbij dus lokaal is.

```
gap> getal:=function(get)
>       local p;
>       p:=5;
>       Print(get,"\n");
>       get:=get+p;
>       return get;
> end;
function( get ) ... end
gap>
gap> a:=4;
4
gap> getal(a);
4
9
gap> a;
4
```

We zien dat de waarde van `a` niet veranderd is, maar het volgende voorbeeld toont aan dat dit genuanceerd moet worden.

```
gap> lijst:=function(list)
>       local p;
>       p:=5;
>       list[Length(list)]:=p;
>       list:=[];
>       return list;
> end;
function( list ) ... end
gap> l:=[1,2,3];
[ 1, 2, 3 ]
gap> lijst(l);
[  ]
gap> l;
[ 1, 2, 5 ]
gap>
```

Bij het uitvoeren van de functie wordt voor elk argument een variabele voorzien, waarbij deze variabele niet alleen de waarde krijgt van het argument, maar ook de verwijzing naar de deelobjecten. Zo hebben zowel de lokale als de globale variabelen identieke deelobjecten. Wanneer die veranderen, gebeurt dit in beide objecten.

We zien dat we moeten opletten als we in voorgaande situatie komen. Dit is vooral het geval als we werken met lijsten en matrices.

1.8 Eindige velden

We kunnen in **GAP** met eindige velden op dezelfde manier werken als met andere velden, maar er bestaan nog een aantal extra functies speciaal voor eindige velden. Alle volgende functies geven als resultaat een eindig veld weer.

`GaloisField(` p^d `) F`
`GaloisField(` *p, d* `) F`
`GaloisField(` *S, d* `) F`
`GaloisField(` *p, pol* `) F`
`GaloisField(` *S, pol* `) F`

Waarbij al deze commando's ook werken, door gebruik te maken van de afkorting `GF`. Het eerste argument bepaalt het deelveld S of $\mathrm{GF}(p)$ waarover we het nieuwe veld willen construeren. Het tweede argument specifieert de uitbreiding. Dit is een geheel getal d, waarbij het nieuw veld

geconstrueerd wordt als de polynomiale uitbreiding met behulp van het Conway polynoom van graad d over het deelveld, of een irreducibel polynoom *pol*, waarbij de polynomiale extensie geconstrueerd wordt met behulp van dit polynoom.

Elementen van eindige velden zijn objecten die in de categorie `IsFFE` zitten. Om elementen van een eindig veld te creëren kunnen we gebruik maken van de functie

`Z(` p^d `) F`

Deze oproep geeft de generator van de multiplicatieve groep van GF(p^d) weer. Deze generator is een wortel van het unieke Conway polynoom van GF(p^d), waarbij de functie steeds dezelfde wortel weergeeft.
Het eenheidselement voor de optelling en de vermenigvuldiging wordt gegeven door respectievelijk `0*Z(p)` en `Z(p)^0`. Deze objecten zijn verschillend van de gehele getallen 0 en 1, want `IsInt( Z(p)^0 )` is vals, terwijl dit voor 1 waar is. Ondanks het feit dat deze objecten in verschillende categorieën liggen, is het wel mogelijk een element van een eindig veld te vermenigvuldigen met of op te tellen bij een natuurlijk getal. Met volgend voorbeeld illustreren we beknopt wat de mogelijkheden daarvan inhouden. Stel dat we een matrix van een lineaire afbeelding hebben en we een basisovergang uitvoeren waarbij de twee basisvectoren verwisseld worden. De matrix van de afbeelding `A` moet gegeven worden met elementen uit het eindig veld, anders wordt die niet gezien als een afbeelding tussen eindige velden. De matrix `B` van de basisovergang kunnen we gewoon in gehele getallen laten. We zien dat het resultaat `A*B` onafhankelijk is van het feit of `B` opgebouwd is uit elementen van een eindig veld of niet.

```
gap> A:=[[Z(4)^0,Z(4)*0],[0*Z(4),Z(4)^0]];
[ [ Z(2)^0, 0*Z(2) ], [ 0*Z(2), Z(2)^0 ] ]
gap> B:=[[0*Z(4),Z(4)^0],[Z(4)^0,Z(4)*0]];
[ [ 0*Z(2), Z(2)^0 ], [ Z(2)^0, 0*Z(2) ] ]
gap> A*B;
[ [ 0*Z(2), Z(2)^0 ], [ Z(2)^0, 0*Z(2) ] ]
gap> B:=[[0,1],[1,0]];
[ [ 0, 1 ], [ 1, 0 ] ]
gap> A*B;
[ [ 0*Z(2), Z(2)^0 ], [ Z(2)^0, 0*Z(2) ] ]
```

Bij het construeren van een projectief punt of een hypervlak moeten we er steeds op letten dat de coördinaten elementen zijn van een eindig veld, zodat het object weldegelijk tot de projectieve ruimte zal behoren.

1.9 GRAPE

We zullen gebruik maken van het pakket **GRAPE** (GRaph Algorithms using PErmutation groups), dat ontwikkeld werd voor de constructie en analyse van grafen geassociëerd aan groepen,

designs en meetkundes. Speciale nadruk ligt op het bepalen van regulariteitsvoorwaarden en structuren van deelgrafen.
De filosofie achter **GRAPE** is dat aan een graaf altijd een deelgroep G van de automorfismegroep van de graaf gerelateerd is. G wordt gebruikt om de opslagruimte en de CPU-tijd van de berekeningen te reduceren. Deze groep kan eventueel ook de triviale groep zijn.

Het laden van het pakket gebeurt zoals bij het pakket **pg**. Bij het laden krijgen we volgende boodschap:

```
gap> LoadPackage("grape");
#I  Package 'GRAPE': non-Unix architecture or binaries not compiled
#I  Package 'GRAPE': functions depending on nauty will not work

Loading  GRAPE 4.2  (GRaph Algorithms using PErmutation groups),
by L.H.Soicher@qmul.ac.uk.
```

De constructie van de graaf Γ gebeurt met de functie:

`Graph(`*G, L, act, rel*`)`
`Graph(`*G, L, act, rel, invt*`)`

Als de optionele parameter *invt* de waarde `false` heeft of ongebonden is, dan moet de lijst L elementen bevatten van de verzameling S waarop de groep G werkt, waarbij de werking beschreven wordt door de functie *act*. De parameter *rel* moet een booleaanse functie zijn die een G-invariante relatie op S definieert, zodat geldt

$$\forall g \in G, \forall x, y \in S \quad rel(x, y) \Leftrightarrow rel(act(x, g), act(y, g)).$$

De functie `Graph` geeft dan een graaf Γ weer, waarbij de toppen als namen hebben

$$\texttt{Concatenation(Orbits(}\ G, L, act\ \texttt{)}).$$

Toppen v en w van Γ zijn adjacent als en slechts als

$$rel(\texttt{Vertexname(}\ \Gamma, v\ \texttt{)}, \texttt{Vertexname(}\ \Gamma, v\ \texttt{)})$$

Indien de parameter *invt* bestaat en de waarde `true` heeft, dan wordt aangenomen dat L invariant is onder G ten opzichte van de actie *act*. De functie `Graph` gedraagt zich zoals hierboven, behalve dat de toppen een onveranderlijke kopie van L zijn.

Eens de graaf geconstrueerd, zullen we telkens een aantal standaardeigenschappen nagaan. We geven hieronder al een overzicht. Merk hierbij op dat de naam van de operatie al voor zich spreekt.

`OrderGraph (`Γ `)`

Deze functie geeft het aantal toppen van de graaf weer.

`IsLoopy (Γ )`

Dit is een booleaanse functie die `true` geeft als de graaf een lus bevat. Een lus is een boog die een top met zichzelf verbindt.

Een graaf is simpel als er geen lussen zijn en als voor elke boog $[x, y]$ ook $[y, x]$ een boog is. We kunnen dit op de volgende manier nagaan:

`IsSimpleGraph (Γ )`

`Diameter (Γ )`

Als een graaf niet samenhangend is wordt als diameter -1 weergegeven, anders is de diameter de maximale afstand in de graaf tussen twee willekeurige toppen.

`Girth (Γ )`

Deze functie geeft de lengte van de kortste cykel weer. Als er geen cykels zijn, is het resultaat van deze functie -1.

`IsRegularGraph (Γ )`

Indien elke top verbonden is met evenveel andere toppen, dit is als alle toppen dezelfde valentie hebben, dan noemen we de graaf regulier.

`GlobalParameters (Γ )`

Stel Γ is een simpele graaf, waarbij $0 \leqslant i \leqslant$ `Diameter( Γ )`. Deze functie bepaalt alle globale parameters c_i, a_i en b_i die Γ zou kunnen hebben. We zeggen dat Γ de globale parameter c_i (respectievelijk a_i, b_i) heeft, indien het aantal toppen op afstand $i-1$ (respectievelijk i, $i+1$) van een top v die adjacent is met een top w, op afstand i van v, de constante c_i (respectievelijk a_i, b_i) is, enkel afhankelijk is van i en niet van v en w. De functie `GlobalParameters` geeft een lijst terug met lengte diameter(Γ)+1, waarvan het i^{de} element de lijst $[c_{i-1}, a_{i-1}, b_{i-1}]$ is, tenzij een bepaalde globale parameter niet bestaat. In dit laatste geval verschijnt een -1 op de corresponderende plaats.

Hoofdstuk 2

Ingebedde projectieve ruimten

In dit hoofstuk beschrijven we operaties en implementaties om projectieve meetkundes in te bedden in andere projectieve meetkundes. We zullen twee types inbeddingen in beschouwing nemen. Enerzijds zullen we een methode implementeren die $\mathrm{PG}(n,q)$ inbedt in $\mathrm{PG}(m,q)$, $n<m$, als een deelruimte. Anderzijds zullen we een methode implementeren die $\mathrm{PG}(n,q)$ inbedt in PG(n,q'), als Baer-deelmeetkunde. Als toepassing bestuderen we de veralgemeende vierhoek $T_2(\mathcal{O})$.

2.1 Declaratie en constructie van ingebedde deelruimten

Zoals aangegeven moet de inbedding van projectieve meetkundes ingedeeld worden in twee delen. Eerst zullen we de constructie beschouwen van $\mathrm{PG}(n,q)$ in $\mathrm{PG}(m,q)$ met $n<m$, waarna het ander deel analoog zal verlopen.

We declareren ingebedde deelruimten als een deelcategorie van de projectieve ruimten, waardoor bestaande operaties zonder meer toepasbaar zijn op de nieuwe objecten, op voorwaarde dat we de nodige methoden installeren.
Aangezien er gekozen werd om voor de projectieve ruimten de representatie `IsAttributeStoringRep` te gebruiken, is het logisch dit ook voor de nieuwe deelcategorie te doen. Door deze representatie worden de ingebedde deelruimten componentobjecten. We zullen gebruik maken van drie componenten, nl. twee projectieve ruimten waarvan de eerste ingebed zal worden in de tweede en een deelruimte die aangeeft hoe de ingebedde deelruimte er uitziet. Als we bijvoorbeeld $\mathrm{PG}(2,3)$ inbedden in $\mathrm{PG}(3,3)$ dan vertelt de informatie opgeslaan in de deelruimte ons dat $\mathrm{PG}(2,3)$ het hypervlak $X_3=0$ is.
De declaratie van de categorie en de representatie gebeurt dan zoals aangegeven in de inleiding.

```
DeclareCategory( "IsEmbeddedPG", IsPG );
```

```
DeclareRepresentation( "IsEmbeddedPGRep", IsAttributeStoringRep,
["pg","motherpg","subspace"]);
```

Voor deelruimten werden twee verschillende representaties gedeclareerd, nl. enerzijds de deelruimten als span van punten en anderzijds als doorsnijding van een aantal hypervlakken [1]. Het gebruik van deelruimten impliceert dat we onderscheid moeten maken in die twee verschillende representaties, waardoor we twee methodes moeten implementeren om ingebedde deelruimten te construeren. Met het oog op later gebruik kiezen we ervoor het component `subspace` op te slaan als doorsnede van hypervlakken, omdat dit handig zal blijken te zijn voor verdere methodes.

We declareren een operatie die een ingebedde deelruimte construeert. Door het gebruik van een operatie in plaats van een functie is dit ook al geschikt om door de gebruiker toegepast te worden. Bij de constructie moeten ook de nodige families gecreëerd worden. Indien een familie nog niet bestaat, wordt deze aangemaakt en opgeslaan als een nieuw component (`embeddedfam`) van de projectieve ruimte die ingebed wordt. Op deze manier kunnen we die familie simpelweg opvragen en moeten we ze niet telkens opnieuw construeren.
Doordat de gebruiker deze operaties kan gebruiken, moeten we erop toezien dat de juiste gegevens meegegeven worden. Daarom controleren we eerst of het eerste argument een projectieve ruimte is van lagere dimensie dan de tweede en of de deelruimte dezelfde dimensie heeft als de ruimte die we willen inbedden. Dit alles leidt tot de volgende methode:

```
DeclareOperation("Embed",[IsPG, IsPG, IsSubspaceOfPG ]);

InstallMethod(Embed,
    "for pg and supspace of PG",
    true,
    [IsPG and IsPGRep, IsPG and IsPGRep,
    IsSubspaceOfPG and IsSubspaceIntersRep],
    0,
    function(P1 ,P2 ,S )
      local F, E, data, fam;
    F := FamilyObj(P1!.projpt);
    E := FamilyObj(P2!.projpt);
    if not ((F!.n < E!.n) and (F!.q = E!.q)) then
      Error( "P1 must be a subspace of P2");
    fi;
    if not ((PGOfSubspace(S)=P2) and (Dimension(S)=F!.n)) then
      Error("S must be a subspace of P2 with the same dimension as P1");
    fi;
    if IsBound ( P1!.embeddedfam )
      then fam := P1!.embeddedfam;
    else
      fam := NewFamily(Concatenation( "EmbeddedPG( ", String(F!.n),
```

```
      " , " , String(F!.q)  , " )"), IsEmbeddedPG ) ;
      P1!.embeddedfam:= fam;
    fi;
    data:= rec( pg:=P1, motherpg:= P2, subspace:=S );
    return Objectify(NewType(fam, IsEmbeddedPG and IsEmbeddedPGRep), data);
end);
```

Opmerking
In de test op de dimensie van de deelruimte hebben we het attribuut `Dimension` opgeroepen. Door onze kennis van de interne representatie van die oproep weten we dat daardoor de hypervlakken die de deelruimte bepalen lineair onafhankelijk zijn (zie [1]).

Indien de deelruimte opgespannen wordt door een verzameling punten, zal de voorgaande methode niet werken. Door de representatie van die deelruimte te wijzigen via `AsIntersectingHyperplanes`, kunnen we die methode wel oproepen. Dan krijgen we het volgende:

```
InstallMethod(Embed,
      "for pg and supspace of PG",
      true,
    [IsPG and IsPGRep, IsPG and IsPGRep,
    IsSubspaceOfPG and IsSubspaceSpanRep],
    0,
    function(P1 ,P2 ,S )
    return Embed(P1, P2, AsIntersectingHyperplanes(S));
end);
```

In het geval van Baer-deelmeetkundes willen we $\mathrm{PG}(n, q^n)$ inbedden in $\mathrm{PG}(n, q^m)$, met $n \mid m$. Veel blijft zoals in het vorige geval. We hebben nu echter geen keuze over hoe de ingebedde ruimte er zal uitzien. Een punt van $\mathrm{PG}(n, q^m)$ behoort tot $\mathrm{PG}(n, q^n)$ als zijn coördinaatvector enkel elementen uit het deelveld $\mathrm{GF}(q^n)$ van $\mathrm{GF}(q^m)$ bevat. De component `subspace` wordt hierdoor nutteloos en dus zal de methode nu gebruik maken van maar 2 argumenten. Doordat de declaratie van de operatie `Embed` drie argumenten eist, zullen we `InstallOtherMethod` nodig hebben. Ook de controle op de gegevens is anders. We moeten bij Baer-deelmeetkunden controleren of het veld $\mathrm{GF}(q^n)$ van $\mathrm{PG}(n, q^n)$ wel een deelveld is van $\mathrm{GF}(q^m)$ van $\mathrm{PG}(n, q^m)$, dus of $n \mid m$.

```
InstallOtherMethod(Embed,
    "for a PG and a Baer-subspace of PG",
    true,
    [IsPG and IsPGRep,IsPG and IsPGRep],
    0,
    function(P1,P2)
```

```
    local F,G,q,p,fam,data;
    F:=FamilyObj(P1!.projpt);
    G:=FamilyObj(P2!.projpt);
    q:=Factors(F!.q);
    p:=Factors(G!.q);
    if not ((q[1]=p[1]) and (IsInt(Length(q)/Length(p)) then
      Error( "P1 must be a Baer-subspace of P2");
    fi;
    if IsBound ( P1!.embeddedfam )
      then fam := P1!.embeddedfam;
    else
      fam := NewFamily(Concatenation( "EmbeddedPG( ", String(F!.n),
      " , " , String(F!.q), " )"), IsEmbeddedPG ) ;
      P1!.embeddedfam:= fam;
    fi;
    data:= rec( pg:=P1, motherpg:= P2);
    return Objectify(NewType(fam, IsEmbeddedPG and IsEmbeddedPGRep), data);;
end);
```

We installeren al de methodes voor de operaties `ViewObj` en `PrintObj`. `ViewObj` wordt automatisch opgeroepen telkens een object op het scherm getoond wordt. Dit geeft informatie over het object op een beknopte manier. `PrintObj` wordt opgeroepen door het commando `Print` in te geven. Meestal zal dit meer gedetailleerd zijn.

```
InstallMethod( ViewObj,
    "for a PG embedded",
    true,
    [IsEmbeddedPG and IsEmbeddedPGRep],
    0,
    function(x)
    Print ( x!.pg," embedded in ",  x!.motherpg );
end);
```

```
InstallMethod(PrintObj,
      "for EmbeddedPGs",
      true,
      [IsEmbeddedPG and IsEmbeddedPGRep],
      0,
      function(x)
      Print ( " PG embedded by ", x!.subspace  );
end);
```

Voorbeelden 1.

```
gap> F:=ProjectiveGeometry(1,3);
PG( 1, 3 )
gap> G:=ProjectiveGeometry(3,3);
PG( 3, 3 )
gap> a:=Hyperplane(G,[1,1,1,1]*Z(3)^0)
[ Z(3)^0, Z(3)^0, Z(3)^0, Z(3)^0 ]
gap> b:=Hyperplane(G,[0,1,2,0]*Z(3)^0)
[ 0*Z(3), Z(3)^0, Z(3), 0*Z(3) ]
gap> S:=SubspaceOfPG([a,b]);
Subspace of PG( 3, 3 )
gap> P:=Embed(F,G,S);
PG( 1, 3 ) embedded in PG( 3, 3 )
gap> Print(P);
 PG embedded by Subspace of PG( 3, 3 ) by intersection of:
[ Hyperplane( PG( 3, 3 ), [ Z(3)^0, Z(3)^0, Z(3)^0, Z(3)^0 ] ),
  Hyperplane( PG( 3, 3 ), [ 0*Z(3), Z(3)^0, Z(3), 0*Z(3) ] ) ]
gap> F:=ProjectiveGeometry(2,2);
PG( 2, 2 )
gap> G:=ProjectiveGeometry(2,8);
PG( 2, 8 )
gap> P:=Embed(F,G);
PG( 2, 2 ) embedded in PG( 2, 8 )
```

2.2 Projectieve punten in ingebedde deelruimten

We hebben nu $\mathrm{PG}(n,q)$ ingebed in $\mathrm{PG}(m,q)$ met $n < m$. De volgende stap is het omzetten van een punt $p(x_0,\ldots,x_n)$ van de ingebedde deelruimte $\mathrm{PG}(n,q)$ naar een punt $p'(x_0,\ldots,x_m)$ van de ruimte $\mathrm{PG}(m,q)$.

We schrijven eerst een hulpfunctie `REMOVE` die gegeven een $n \times m-$matrix, met $n < m$, $m-n$ kolommen bepaalt, zodat de vierkante matrix die overblijft na het schrappen van die $m-n$ kolommen een reguliere matrix is. Dit zal voor een willekeurige matrix niet altijd mogelijk zijn, maar wel in het specifieke geval waarin we die hulpfunctie zullen oproepen. Doordat de functie slechts in zeer specifieke gevallen zinvol is, is het niet nodig deze beschikbaar te stellen voor de gebruikers. De functie begint met het volgende:

```
N:=Immutable(N);
M:=List(N,ShallowCopy);
```

De functie heeft als bedoeling om een lijst met nummers van kolommen weer te geven, zonder evenwel de matrix zelf te veranderen. We moeten aan **GAP** expliciet vermelden dat de matrix N niet mag veranderd worden, wat gebeurt op de eerste regel. Vele bewerkingen kunnen alleen

toegepast worden op objecten die `mutable` zijn. Daarom is het nodig om een kopie te nemen van de matrix N, maar dan zodanig dat die kopie M wel veranderlijk is. De functie zal nu de verdere berekeningen uitvoeren met de veranderlijke M, terwijl de matrix N die meegegeven werd bij de oproep van de functie niet gewijzigd wordt.

We gaan nu als volgt te werk. We bekijken de rijen van de matrix in volgorde, volgens het aantal niet-nul elementen die de rij bevat, beginnende met de rij met het minste niet-nul elementen. Stel dat de 1^{ste} rij die we bekijken n_1 elementen verschillend van nul bevat op kolommen $k_1^1, \ldots, k_{n_1}^1$. Dan houden we de eerste $(n_1 - 1)$ kolommen daarvan bij in de lijst `result` en verwijderen alle elementen $k_1^1, \ldots, k_{n_1}^1$ uit de lijst `lijst`$= [1, 2, .., m]$. Deze lijst bevat dan de nummers van alle kolommen die nog verwijderd kunnen worden in de volgende stappen. In de i^{de} stap bekijken we rij j die n_i niet-nul elementen bevat op plaats $k_1^i, \ldots, k_{n_i}^i$. Nu bepalen we eerst de doorsnee van die plaatsen met `lijst` van nog mogelijke kolommen en als die doorsnee n elementen bevat, zullen we opnieuw $n - 1$ ervan toevoegen aan de lijst `result` om daarna alle n de kolommen te verwijderen uit `lijst`. Op die manier zorgen we ervoor dat op elke rij minimum 1 niet-nul element overblijft na het schrappen van de kolommen. Dit doen we tot we ten minste $m - n$ kolommen gevonden hebben die we kunnen verwijderen.

```
InstallGlobalFunction(REMOVE,
    function(N,q)
    local dim,n,result,lijst,perm,nr,Snr,i,j,k,m,kol,pos,M;
    N:=Immutable(N);
    M:=List(N,ShallowCopy);
    dim:=DimensionsMat(M);
    n:=dim[2]-dim[1]; # n staat hier voor wat we anders n+1 noemen
    result:=[];
    lijst:=[1..dim[2]];
    perm:=[];
    nr:=[];
    for i in [1..dim[1]] do
      perm[i]:=Inverse(Sortex(M[i]));
      M[i]:=Reversed(M[i]);
      if (M[i][dim[2]]=0*Z(q)) then
        nr[i]:=First([1..dim[2]],j-> M[i][j]=0*Z(q))-1;
      else
        nr[i]:=dim[2];
      fi;
    od;
```

Tot hiertoe hebben we elke rij van de matrix gesorteerd en omgedraaid, zodat de nullen achteraan staan. Hierdoor is het aantal niet-nullen juist de positie van de eerste nul met 1 verminderd. Het sorteren van de rij deden we door gebruik te maken van de volgende **GAP**-operatie:

`Sortex(`*List*`) O`

Deze operatie sorteert de lijst *List* en geeft de permutatie weer die het sorteren bewerkstelligt. De inverse van deze permutatie kunnen we later gebruiken om de positie van de niet-nullen na te gaan in de oorspronkelijke matrix.

```
    Snr:=SortedList(nr);
    j:=1;
    while (Length(result)<n) do
      pos:=Position(nr,Snr[j]);
      kol:=[];
      for m in [dim[2]-(Snr[j]-1)..dim[2]] do
        Add(kol,m^perm[pos]);
      od;
      kol:=Intersection(kol,lijst);
      for k in [1..Length(kol)-1] do
        Add(result,kol[k]);
      od;
      lijst:=Difference(lijst,kol);
      j:=j+1;
      nr[pos]:=0;
    od;
    result:=Reversed(SSortedList(result));
    N:=List(N,ShallowCopy);
    return result;
end);
```

We zullen nu een methode schrijven die gegeven een punt $p(x_0, x_1, ..., x_n) \in \mathrm{PG}(n, q)$ het corresponderende punt $p'(x'_0, x'_1, ..., x'_m)$ berekent zodat $p' \in \mathrm{PG}(m, q)$, waarbij we voorgaande functie gebruiken. De inbedding is zodanig dat $\mathrm{PG}(n, q)$ voorgesteld wordt als de doorsnede van een aantal hypervlakken van $\mathrm{PG}(m, q)$, met $n < m$. Zoals reeds vermeld zorgt de oproep `Dimension` ervoor dat de hypervlakken in het data gedeelte van de deelruimte gereduceerd zijn tot een lijst van lineair onafhankelijke hypervlakken. Aangezien deze oproep op dit punt reeds uitgevoerd werd, weten we dus dat we over juist $m-n$ lineair onafhankelijke hypervlakken beschikken. Het punt p' moet in elk hypervlak liggen, anders gezegd p' moet een oplossing zijn van het volgende stelsel:

$$\begin{cases} a_0^1 X_0 + a_1^1 X_1 + .. + a_m^1 X_m = 0 \\ \vdots \\ a_0^{m-n} X_0 + a_1^{m-n} X_1 + .. + a_m^{m-n} X_m = 0 \end{cases}$$

De lijst met hypervlakken is een component van de deelruimte die meegegeven werd bij de constructie van de inbedding. Een lijst van hypervlakken wordt voorgesteld door een lijst van lijsten, waarbij elke lijst de coëfficiënten van een hypervlak bevat. Dit kan ook gezien worden

als een matrix waarbij op rij i de coëfficiënten van het i^{de} hypervlak staan, nl:

$$M = \begin{bmatrix} a_0^1 & a_1^1 & \dots & a_m^1 \\ \vdots & & & \\ a_0^{m-n} & a_1^{m-n} & \dots & a_m^{m-n} \end{bmatrix}$$

M is dus een $(m-n) \times (m+1)-$matrix.

De matrix M en het punt $p(x_0, x_1, ..., x_n)$ van PG(n, q) zijn gekend. Hiermee moeten we de coördinaten van het punt $p'(x'_0, x'_1, ..., x'_m)$ berekenen. Is er een hypervlak dat een vergelijking heeft van de vorm $X_i = 0$, dan is het onmiddellijk duidelijk dat moet gelden $x'_i = 0$. Zo'n hypervlak correspondeert in M met een rij j met enkel op de i^{de} positie een 1 en voor de rest allemaal nullen. We zullen de index i bijhouden in een lijst en in M de i^{de} kolom en de j^{de} rij verwijderen uit de matrix. Dit doen we nu voor alle hypervlakken van die welbepaalde vorm.

Stel dat er x vergelijkingen waren van de eerste soort. Als $x < m - n$, dan zijn er nog andere hypervlakken en is de matrix nog een $(m-(n+x)) \times (m+1-x)-$matrix. Het is nu de bedoeling om de overblijvende coördinaten van p' te berekenen door aan $n+1$ onbekenden de waarden te geven van de coördinaten van p . We willen die coördinaten zodanig toekennen dat de overige onbekenden een unieke oplossing hebben. We mogen dus niet zomaar de eerste $n+1$ onbekenden nemen, want in volgend geval zou dit niet werken.

$$\begin{cases} X_0 + X_1 = 0 \\ X_2 + X_3 = 0 \\ X_4 + X_5 = 0 \end{cases}$$

Hierin zouden we een waarde toekennen aan de eerste drie onbekenden, wat geen uitsluitsel geeft over de oplossing voor de laatste twee onbekenden. We moeten er dus voor zorgen dat er in elke rij nog een onbekende overblijft. We merken op dat

$$(m + 1 - x) - (m - (n + x)) = n + 1.$$

We kunnen de hulpfunctie `REMOVE` gebruiken. Deze zal $n+1$ kolommen teruggeven waaraan we dan de waarde van de coördinaten van het punt $p(x_0, x_1, ..., x_n)$ kunnen toekennen. Dit doen we door die kolommen uit M te verwijderen en er een nieuwe matrix N mee te maken. N is dan een $y \times n + 1-$matrix waarbij $y = (m - (n + x))$. We berekenen

$$A = N * [x_0, x_1, ..., x_n]^{\perp}$$

waarbij A dan een vector is van lengte y. Doordat we uit M $n+1$ kolommen verwijderd hebben is M nog een $(m-(n+x)) \times ((m+1-x)-n+1) = (y \times y)-$matrix. Om nu de waarde van de y overblijvende onbekenden te kennen moeten we de oplossing bepalen van het volgend stelsel:

$$M * X^{\perp} = -A \text{ met } X = [X_1, X_2, .., X_y]$$

Dit is een stelsel van y vergelijkingen met y onbekenden, waarbij M een reguliere matrix is. Dit stelsel heeft dus een unieke oplossing. Rest ons nu nog de nullen bij te voegen die corresponderen

met de vergelijkingen $X_i = 0$ op de plaatsen die we reeds bepaald hebben. Hiermee hebben we een punt $p'(x'_0, x'_1, ..., x'_m)$ berekend dat voldoet aan alle voorwaarden.

We verduidelijken voorgaande eerst nog met een eenvoudig voorbeeld. Beschouw de inbedding van $\mathrm{PG}(2, q)$ in $\mathrm{PG}(4, q)$ als de doorsnijding van twee hypervlakken:

$$X_2 = 0 \text{ en } X_0 + X_1 + X_2 + X_3 + X_4 = 0.$$

Stel $p(x_0, x_1, x_2)$ een punt van $\mathrm{PG}(2, q)$. Het corresponderende punt $p' \in \mathrm{PG}(4, q)$ moet voldoen aan volgende stelsel:

$$\begin{cases} X_2 = 0 \\ X_0 + X_1 + X_2 + X_3 + X_4 = 0 \end{cases}$$

We zien dat er een hypervlak is van de vorm $X_i = 0$. De eerste stap is het bijhouden van het getal 2 in een lijst en het reduceren van het stelsel tot de volgende vorm:

$$\begin{cases} X_0 + X_1 + X_3 + X_4 = 0 \end{cases}$$

Uit de overblijvende onbekenden selecteren we dan 3 onbekenden waaraan we de coördinaten van p toekennen. Stel dat dit X_0, X_1 en X_3 zijn, dan wordt

$$X_4 = -\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = -(x_0 + x_1 + x_2)$$

Nu kennen we de waarde van de coördinaten x'_0, x'_1, x'_3, x'_4 van p'. Als laatste stap plaatsen we op de positie van het bijgehouden getal, hier 2, een nul. p' heeft dan als coördinaten $(x_0, x_1, 0, x_2, -(x_0 + x_1 + x_2))$.

Voor latere toepassingen beschrijven we eerst een methode die niet een punt als argument heeft, maar een vector.

```
InstallOtherMethod(ProjectivePoint,
      "for a PG embedded and a vector",
      true,
      [IsEmbeddedPG and IsEmbeddedPGRep, IsList and IsFFECollection],
     0,
      function(P,vector)
      local M,P1,P2,F,S,n,m,q,nr,x,row,col,N,A,i,s,list,L;
      S:=P!.subspace;
      M :=ShallowCopy(HyperplaneToVector(S!.list));
     P1:=P!.pg;
      P2:=P!.motherpg;
     F:=FamilyObj(P1!.projpt);
     n:=F!.n;
     q:=F!.q;
```

```
    m:=FamilyObj(P2!.projpt)!.n;
    if not (Length(vector)=n+1) then
       Error( "The length of the vector is not correct");
    fi;
     for i in [1..Length(vector)]do
      if not vector[i] in F!.field then
        Error("The components of the vector are not correct");
      fi;
    od;
```

Dit waren de controles op de door de gebruiker meegegeven argumenten. De eerste controle gaat na of de vector wel de juiste lengte heeft, zodat die vector effectief een punt zou kunnen representeren. De tweede test gaat na of de elementen van de vector tot het juiste veld behoren.

```
 x:=0;
  row:=[];
col:=[];
for i in [1..m-n] do
  nr:=Number([1..m+1], j->M[i][j]=0*Z(q));
  if nr=m then
    x:=x+1;
    Add(row,i);
    Add(col,First([1..m+1], j->M[i][j]=Z(q)^0));
  fi;
od;
col:=SortedList(col);
if not x=0 then
  for i in [1..x] do
    Remove(M,row[x-(i-1)]);
  od;
  M:=MutableTransposedMat(M);
  for i in [1..x] do
    if not M=[] then
      Remove(M,col[x-(i-1)]);
    fi;
  od;
  M:=MutableTransposedMat(M);
 fi;
```

`Number(`*list, func*`) F`

geeft het aantal elementen van de lijst *list* terug dat voldoet aan de unaire functie *func*. Hiermee gaan we na hoeveel elementen gelijk zijn aan nul in elke rij van de matrix. Indien er maar 1 element verschillend van nul is maken we gebruik van de operatie

`Remove(`*list, pos*`) O`

die in *list* het element op positie *pos* verwijdert. In een matrix komt een positie verwijderen overeen met het weg doen van een rij. Om dan kolommen weg te doen, transponeren we de matrix eerst, zodat dit opnieuw neerkomt op het verwijderen van een rij.
We mogen niet zomaar een aantal rijen na elkaar uit een matrix verwijderen. We moeten er rekening mee houden dat door het verwijderen van een rij de volgende rijen een rij naar boven opschuiven. Stel dat we de rijen $(i_1, i_2, \ldots, i_n)$ willen weg doen. Na de i_1^{ste} weggelaten te hebben, staat de rij i_2 nu op de rij $i_2 - 1$. Dit kunnen we oplossen door van achter naar voor te werken. Als we eerst de laatste rij verwijderen, verandert er niets aan de positie van voorgaande rijen. Is de matrix niet-ledig, dan gaan we verder met bovenstaande redenering.

```
if not M=[] then
  list:=REMOVE(M,q);
  M:=MutableTransposedMat(M);
  N:=[];
  if not (m-(n+x))=0 then
    for i in [1..(n+1)] do
      Add(N,Remove(M,list[i]));
    od;
  fi;
  N:=Reversed(N);
  A:=AdditiveInverse(vector*N);
  s:=SolutionMat(M,A);
  L:=Difference([1..(m+1-x)],list);
  for i in [1..Length(s)]do
    Add(vector,s[i],L[i]);
  od;
  fi;
  if not x=0 then
    for i in [1..x] do
       Add(vector,0*Z(q),col[i]);
    od;
  fi;
return ProjectivePoint(P2,vector);
end);
```

De methode voor een punt maakt gebruik van `PointToVector` om dan voorgaande methode op te roepen. We zetten er meteen ook een deel in dat werkt in het geval van een Baer-deelmeetkunde. Zo moet er niets veranderd worden aan de coördinaten van het punt en moet het alleen maar geherdefinieerd worden worden als punt van de grote ruimte.

```
InstallOtherMethod( ProjectivePoint,
      "for EmbeddedPG's and a projective point",
      true,
[IsEmbeddedPG and IsEmbeddedPGRep,
 IsProjectivePoint and IsProjectivePointRep],
0,
function(P,p)
local vector;
if not IsBound(P!.subspace) then
  vector:=PointToVector(p);
  return ProjectivePoint(P!.motherpg,vector);
else
  vector:=ShallowCopy(PointToVector(p));
   return ProjectivePoint(P,vector);
fi;
end);
```

Om nu alle punten van de ingebedde ruimte te berekenen, volstaat het in het eerste geval de methode `ProjectivePoints` toe te passen op de component `subspace`. Daar dit een deelruimte is van de grootste projectieve ruimte, zullen de punten berekend door `ProjectivePoints` ook meteen in die ruimte liggen. Voor Baer-deelmeetkunden moeten we effectief elk punt omzetten, omdat we hier geen deelruimte hebben. Beide gevallen onderscheiden zich van elkaar doordat in het eerste geval de component `subspace` gebonden is en in het andere niet. We installeren een methode met hogere prioriteit voor het eerste geval. Die methode zal overgaan naar een volgende methode van lagere prioriteit met behulp van `TryNextMethod()`. Dit geeft ons volgende twee methoden:

```
InstallMethod(ProjectivePoints,
    "for a PG embedded in an other PG",
      true,
      [IsEmbeddedPG and IsEmbeddedPGRep],
      1,
      function(P)
      if not IsBound(P!.subspace) then
        TryNextMethod();
      else
       return(ProjectivePoints(P!.subspace));
      fi;
end);
```

```
InstallMethod( ProjectivePoints,
"for a PG embedded in an other PG",
```

```
true,
[IsEmbeddedPG and IsEmbeddedPGRep],
0,
function(P)
local i,pts,list,x;
pts:=ProjectivePoints(P!.pg);
list:=[];
for x in pts do
   Add(list,ProjectivePoint(P,x));
od;
return list;
end);
```

Voorbeelden 2.

```
gap> F:=ProjectiveGeometry(2,8);
PG( 2, 8 )
gap> G:=ProjectiveGeometry(5,8);
PG( 5, 8 )
gap> a:=Hyperplane(G,[1,1,1,1,1,1]*Z(8)^0);
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ]
gap> b:=Hyperplane(G,[0,0,1,0,0,0]*Z(8)^0);
[ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ]
gap> c:=Hyperplane(G,[0,1,0,0,1,0]*Z(8)^0);
[ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ]
gap> S:=SubspaceOfPG([a,b,c]);
Subspace of PG( 5, 8 )
gap> P:=Embed(F,G,S);
PG( 2, 8 ) embedded in PG( 5, 8 )
gap> p:=ProjectivePoint(F,[1,0,1]*Z(8));
[ Z(2)^0, 0*Z(2), Z(2)^0 ]
gap> ProjectivePoint(P,p);
[ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ]
```

2.3 Andere methoden

Toen de projectieve ruimten gecreëerd werden in [4], werd daarbij ook een functie ingebouwd die de punten in een nummer omzet. We kunnen die functie ook uitwerken voor inbeddingen, zodat we niet alleen de punten van $\mathrm{PG}(n,q)$ hebben als punten van $\mathrm{PG}(m,q)$, maar ook hun nummer ten opzichte van $\mathrm{PG}(m,q)$. Nu heeft de methode `PointToNumber` enkel een punt als argument, maar hier zal dat niet volstaan, omdat we aan de hand van dat punt alleen niet kunnen weten in welke ruimte $\mathrm{PG}(n,q)$ ingebed is.
We moeten dus een nieuwe methode schrijven die ook de ingebedde ruimte als argument heeft.

```
InstallOtherMethod(PointToNumber,
"for a projective point and an embedded PG",
true,
[IsEmbeddedPG and IsEmbeddedPGRep,
 IsProjectivePoint and IsProjectivePointRep],
0,
function(P,p)
local F;
p:=ProjectivePoint(P,p);
return PointToNumber(p);
end);
```

Net als bij het originele `PointToNumber` voegen we een methode toe die werkt op een lijst van projectieve punten.

```
InstallOtherMethod(PointToNumber,
   "for a projective point and an embedded PG",
       true,
       [IsEmbeddedPG and IsEmbeddedPGRep,
       IsProjectivePointCollection and IsList],
       0,
       function(P,list)
       local F,result;
       result:=[];
       for p in list do
         p:=ProjectivePoint(P,p);
         Add(result,PointToNumber(p));
       od;
       return result;
end);
```

Het kan in toepassingen handig zijn om, nadat een projectieve ruimte P_1 ingebed is in een grotere ruimte P_2, niet alleen punten van P_1 om te zetten naar punten van P_2, maar ook om deelruimten van P_1 in te bedden in P_2. We kunnen dit eenvoudig realiseren door `ProjectivePoint` toe te passen op de punten die de deelruimte opspannen.

```
InstallOtherMethod(SubspaceOfPG,
         "for a PG embedded and a subspace",
            true,
         [IsEmbeddedPG and IsEmbeddedPGRep, IsSubspaceOfPG],
         0,
         function(P,S)
```

```
        local list,i;
        if IsSubspaceIntersRep(S) then
          S:=AsSpanningPoints(S);
        fi;
        list:=S!.list;
        for i in [1..Length(list)] do
          list[i]:=ProjectivePoint(P,list[i]);
        od;
        return SubspaceOfPG(list);
end);
```

We hebben nu een projectieve deelruimte P_1 ingebed in een andere projectieve deelruimte P_2, punten van de ene ruimte omgezet in punten van de andere ruimte en hun nummer berekend. De mogelijkheden van **GAP** laten echter nog meer toe. Stel P is de inbedding $P_1 \subset P_2$, dan is het mogelijk om P in te bedden in een andere projectieve ruimte P_3. Zo bekomen we de inbedding $P_1 \subset P_2 \subset P_3$. Het is mogelijk dit rijtje uit te breiden naar $P_1 \subset P_2 \subset P_3 \subset P_4 \ldots$. Op die manier kunnen we een oneindig lange ketting van ingebedde ruimten. Dit is een voorbeeld van de mogelijkheid om in **GAP** ook met meer abstracte structuren te werken.

Stel dat we P_1 als de deelruimte S ingebed hebben in P_2. Om nu deze inbedding nog eens in te bedden als deelruimte S' in P_3, kunnen we niet zomaar hetzelfde doen als voorheen. Dit omdat P_1 dan nog steeds als deelruimte van P_2 zou gezien worden en dus niet echt mee ingebed wordt. We moeten de deelruimte S in die inbedding omzetten naar een deelruimte van P_3. P_1 zal dan voorgesteld worden als de doorsnede van beide deelruimten $S \cap S'$.
We passen nog het commando `Reduce` toe op die deelruimte, zodat deze gerepresenteerd wordt als doorsnede van linaeir onafhankelijke hypervlakken.

In het geval P_1 een Baer-deelmeetkunde van P_2 is, hebben we geen deelruimte die nog moet omgezet worden. Deze bemerking is enkel van toepassing als het component `Subspace` van de inbedding $P_1 \subset P_2$ bestaat of gebonden is.

```
InstallOtherMethod(Embed,
"for an embedded pg P to be embedded in P2 as subspace S",#jdb
true,
[IsEmbeddedPG and IsEmbeddedPGRep, IsPG and IsPGRep,
IsSubspaceOfPG and IsSubspaceIntersRep],
0,
function(P ,P2 ,S )
local F, E, P1, data, fam;
P1:=P!.motherpg;
F := FamilyObj(P1!.projpt);
E := FamilyObj(P2!.projpt);
if not ((F!.n < E!.n) and (F!.q = E!.q)) then
```

```
  Error ( " P1 must be a subspace of P2 " );
fi;
if not ((PGOfSubspace(S)=P2) and (Dimension(S)=F!.n)) then
  Error ( " S must be a subspace of P2 with the same dimension as P1 " );
fi;
if not (IsSubspaceIntersRep(S)) then
S:=AsIntersectingRep(S);
fi;
if IsBound(P!.subspace)then
P4:=Embed(P1,P2,S);
s:=SubspaceOfPG(P4,P!.subspace);
S:=Intersection(s,S);
S:=Reduce(S);
fi;
if IsBound ( P1!.embeddedfam ) then
  fam := P1!.embeddedfam;
else
fam := NewFamily ( Concatenation ( " EmbeddedPG( ", String(F!.n),
 " , ", String(F!.q) , " )"), IsEmbeddedPG ) ;
P1!.embeddedfam:= fam;
fi;
data:= rec( pg:=P!.pg, motherpg:= P2, subspace:=S, embeddedpg:=P );
return Objectify( NewType( fam , IsEmbeddedPG and IsEmbeddedPGRep ) , data);
end);
```

Willen we nu $\mathrm{PG}(2,2) \subset \mathrm{PG}(2,4) \subset \mathrm{PG}(2,8)$, dan maken we helemaal geen gebruik van deelruimten, zodat het aantal argumenten niet overeenstemt met dat van voorgaande methode. Hiervoor maken we gebruik van de methode voor de inbedding van Baer-deelmeetkundes, maar met andere argumenten.

```
InstallOtherMethod(Embed,
"for an embedded pg P to be embedded in P2 as subspace S",
true,
[IsEmbeddedPG and IsEmbeddedPGRep, IsPG and IsPGRep],
0,
function(P ,P2 ,S )
local P1,F,G,q,p,fam,data;
P1:=P!.motherpg;
F:=FamilyObj(P1!.projpt);
G:=FamilyObj(P2!.projpt);
q:=Factors(F!.q);
p:=Factors(G!.q);
if not ((q[1]=p[1]) and IsInt(Length(q)/Length(p))) then
```

```
   Error("P1 must be a Baer-subspace of P2");
fi;
if IsBound ( P1!.embeddedfam )
  then fam := P1!.embeddedfam;
else
  fam := NewFamily ( Concatenation ( " EmbeddedPG( ", String(F!.n),
   " , ", String(F!.q) , " )"), IsEmbeddedPG ) ;
  P1!.embeddedfam:= fam;
fi;
data:= rec( pg:=P!.pg, motherpg:= P2);
return Objectify( NewType( fam , IsEmbeddedPG and IsEmbeddedPGRep ) , data);;
end);
```

We geven een voorbeeldje waarin P_1 als Baer-deelmeetkunde ingebed wordt in P_2 dat op zijn beurt als deelruimte ingebed wordt in P_3, waarna we een punt van P_1 omzetten in een punt van P_3.

Voorbeelden 3.

```
gap> P1 := ProjectiveGeometry(2,2);
PG( 2, 2 )
gap> P2 := ProjectiveGeometry(2,4);
PG( 2, 4 )
gap> P3 := ProjectiveGeometry(3,4);
PG( 3, 4 )
gap> h := Hyperplane(P3,[1,1,1,1]*Z(2)^0);
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ]
gap> s := SubspaceOfPG(h);
Subspace of PG( 3, 4 )
gap> P4 := Embed(P1,P2);
PG( 2, 2 ) embedded in PG( 2, 4 )
gap> P5 := Embed(P4,P3,s);
PG( 2, 2 ) embedded in PG( 3, 4 )
gap> pts := ProjectivePoints(P1);
[ [ Z(2)^0, 0*Z(2), 0*Z(2) ], [ 0*Z(2), Z(2)^0, 0*Z(2) ],
  [ 0*Z(2), 0*Z(2), Z(2)^0 ], [ Z(2)^0, Z(2)^0, 0*Z(2) ],
  [ 0*Z(2), Z(2)^0, Z(2)^0 ], [ Z(2)^0, Z(2)^0, Z(2)^0 ],
  [ Z(2)^0, 0*Z(2), Z(2)^0 ] ]
gap> p1 := pts[1];
[ Z(2)^0, 0*Z(2), 0*Z(2) ]
gap> ProjectivePoint(P5,p1);
[ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ]
```

2.4 Toepassing

We zullen de voorgaande operaties nu eens gebruiken in een toepassing. We beschouwen $T_2(\mathcal{O})$, een voorbeeld van een veralgemeende vierhoek.

Definitie 2.4.1.
Een veralgemeende vierhoek is een incidentiestructuur $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ waarin $\mathcal{P}$ en $\mathcal{B}$ disjuncte niet-lege verzamelingen zijn die we respectievelijk punten en rechten noemen en waarbij $\mathcal{I}$ een incidentierelatie is waarvoor voldaan is aan volgende axioma's.

(a) Elk punt is incident met $t+1$ rechten en twee verschillende punten liggen op ten hoogste 1 rechte.

(b) Elke rechte is incident met $s+1$ punten en twee verschillende rechten zijn incident met ten hoogste 1 punt.

(c) Is x een punt en L een rechte niet incident met x, dan bestaat er juist 1 koppel $(y, M) \in \mathcal{P} \times \mathcal{B}$ zodat x I M I y I L.

De gehele getallen s en t worden de *parameters* van de veralgemeende vierhoek genoemd en men zegt dat $\mathcal{S}$ de orde (s, t) heeft. De veralgemeende vierhoek die voldoet aan de volgende definitie wordt dan als $T_2(\mathcal{O})$ genoteerd.

Definitie 2.4.2.
Stel $\mathcal{O}$ is een ovaal van π_0=PG$(2, q)$ en bed π_0 in als hypervlak in PG$(3, q)$. Dan definïeren we de punten $\mathcal{P}$ als:

(i) de punten van PG$(3, q)\backslash\pi_0$,

(ii) de vlakken π' zodat $\pi' \cap \pi_0$ een raaklijn is aan $\mathcal{O}$ in PG$(2, q)$,

(iii) $\infty \equiv \pi_0$

en de rechten $\mathcal{B}$ als:

(i) de rechten van PG$(3, q)\backslash$ PG$(2, q)$ die π_0 snijden in een punt van $\mathcal{O}$,

(ii) de punten van $\mathcal{O}$.

Incidentie $\mathcal{I}$ is gedefinieerd als de natuurlijke incidentie waarbij ∞ incident is met alle rechten van type (b) en geen enkele rechte van type (a).

Door deze definitie is de incidentiestructuur $(\mathcal{P}, \mathcal{B}, \mathcal{I})$ een veralgemeende vierhoek met parameters $s = t = q$.

We veronderstellen dat q even is, en nemen als ovaal de standaard niet-singuliere parabolische kwadriek. π_0 zal ingebed worden als het hypervlak dat voldoet aan de vergelijking $X_3 = 0$. We zullen de incidentierelaties nagaan voor deze veralgemeende vierhoek, d.w.z. we zullen aantonen dat $T_2(\mathcal{O})$ voldoet aan de drie axioma's.

We zullen eerst $T_2(\mathcal{O})$ construeren, d.w.z. dat we twee lijsten aanmaken die de punten en de rechten zullen bevatten van $T_2(\mathcal{O})$. We willen echter dat alle punten en rechten zodanig zijn dat ze gezien worden als punten en rechten van $\mathrm{PG}(3, q)$, waardoor we de methoden die we hierboven beschreven meermaals zullen moeten toepassen.
We beschouwen de inbedding van $\mathrm{PG}(2, q)$ in $\mathrm{PG}(3, q)$. We noemen de ingebedde ruimte P. Om de punten die aan (i) voldoen te berekenen volstaat het de lijst met punten van $\mathrm{PG}(3, q)$ te bepalen en daaruit de punten van $\mathrm{PG}(2, q)$ te verwijderen, dit zijn juist de punten bekomen door `ProjectivePoints` toe te passen op P. Voor de rechten die aan (ii) voldoen volstaat het niet om de punten van de kwadriek te berekenen, want de kwadriek is bepaald in $\mathrm{PG}(2, q)$ en doordat we die punten willen als punten van $\mathrm{PG}(3, q)$ moeten we er `ProjectivePoint` op toepassen. Voor de punten (ii), met name de vlakken, beschouwen we eerst een punt van de kwadriek, maar dit zijn juist de rechten (ii) die we reeds geconstrueerd hebben. Vertrekkende van zo'n punt bepalen we twee punten die de raaklijn in dat punt aan de kwadriek opspannen. Deze twee punten, samen met een punt van $\mathrm{PG}(3, q)\backslash\pi_0$ (=punten(i)) spannen een vlak op van het type punten (ii). Om er voor te zorgen dat niet meermaals hetzelfde vlak geconstrueerd wordt, houden we per punt van de kwadriek een lijst bij met punten van $PG(3, q)\backslash\pi_0$ die nog niet in een vlak liggen. Volledig analoog verloopt de constructie van rechten(i), maar dan moeten we geen gebruik maken van raaklijnen van de kwadriek, enkel van de punten.

```
constructie:=function(q)
local F,G,a,S,P,O,punten1,punten2,punten3,punten,rechten1,rechten2,rechten;
F:=ProjectiveGeometry(2,q);
G:=ProjectiveGeometry(3,q);
a:=Hyperplane(G,[0,0,0,1]*Z(q));
S:=SubspaceOfPG([a]);
P:=Embed(F,G,S);
O:=Quadric(F,[2,1]);
punten1:=Difference(ProjectivePoints(G),ProjectivePoints(P));
punten2:=[];
punten3:=[S];
rechten1:=[];
pts:=ProjectivePoints(O);
rechten2:=[];
for x in pts do
   y:=ProjectivePoint(P,x);
   Add(rechten2,y);
```

```
   H:=SubspaceOfPG(TangentHyperplane(O,x));
   H:=AsSpanningPoints(H);
   list:=H!.list;
   list[1]:=ProjectivePoint(P,list[1]);
   list[2]:=ProjectivePoint(P,list[2]);
   dummy:=punten1;
   repeat
     p:=dummy[1];
     T:=SubspaceOfPG([list[1],list[2],p]);
     Add(punten2,T);
     dummy:=Difference(dummy,ProjectivePoints(T));
   until Length(dummy)=0;
   dummy:=punten1;
   repeat
      p:=dummy[1];
    U:=SubspaceOfPG([p,y]);
    Add(rechten1,U);
    dummy:=Difference(dummy,ProjectivePoints(U));
   until Length(dummy)=0;
od;
punten:=Concatenation(punten1,punten2,punten3);;
rechten:=Concatenation(rechten1,rechten2);;
return (punten,rechten);
end;
```

Verder maken we nog een functie die de gewone incidentie controleert, gebruik makend van bestaande methoden voor `in`. Deze functie gaat eerst na of een van de twee argumenten een punt is en of dat punt dan in het andere argument ligt. Is geen van beide een punt, dan gaat de methode na of het argument met kleinste dimensie in dat met grootste dimensie ligt. Aangezien we hier enkel punten, rechten en vlakken hebben, zal dit geval zich dus voordoen bij een rechte en een vlak en gaat men dus na of de rechte in het vlak ligt. Twee punten zijn vanzelfsprekend niet incident.

```
In:=function(p,q)
if (IsProjectivePoint(p) and IsProjectivePoint(q)) then
   return false;
elif IsProjectivePoint(p) then
   return p in q;
elif IsProjectivePoint(q) then
   return q in p;
elif Dimension(p)<Dimension(q) then
   pts:=ProjectivePoints(p);
   return ((pts[1] in q) and (pts[2] in q));
```

```
else
   pts:=ProjectivePoints(q);
   return ((pts[1] in p) and (pts[2] in p));
fi;
end;
```

Om nu daadwerkelijk de drie incidenties na te gaan, maken we nog gebruik van drie functies waarvan we hier enkel de derde zullen uitwerken. de andere twee zijn analoog, maar eenvoudiger. De functie `incidentie3` moet dus voor elk koppel (x, L), waarvoor $x \not\mathrm{I}\ L$ geldt, een koppel (y, M) vinden zodat $x\ \mathrm{I}\ M\ \mathrm{I}\ y\ \mathrm{I}\ L$ geldt. De functie overloopt alle punten (x) en gaat voor elk punt op zoek naar de rechten waarmee het niet incident is (L). Indien zo'n rechte gevonden wordt, overloopt de functie weer alle punten uitgezonderd het punt waar we mee bezig (x) zijn en zoekt een punt y dat incident is met de rechte L. Daarvoor zoekt het dan nog een rechte zodat aan de andere twee voorwaarden voor de derde incidentierelatie voldaan is. De bedoeling is het aantal koppels (y, M) te tellen dat aan de voorwaarden voldoet en zolang er slechts 1 koppel gevonden wordt voor elke (x, L), waarvoor $x \not\mathrm{I}\ L$, houden we een variabele `ok` op true. Is na het doorlopen van alle mogelijke combinaties van punten en rechten `ok` nog altijd true dan voldoet het veralgemeende vierkant aan de incidentierelatie.

```
incidentie3:=function(q)
local dpunten,drechten,i,j,y,M,ok,aantal,x,L;
ok:=true;
for i in [1..Length(punten)] do
   x:=punten[i];
   for j in [1..Length(rechten)] do
      L:=rechten[j];
       aantal:=0;
       if not In(x,L) then
        dpunten:=ShallowCopy(punten);
        Remove(dpunten,i);
        drechten:=ShallowCopy(rechten);
        Remove(drechten,j);
        for y in dpunten do
           if In(y,L) then
             for M in drechten do
                if (In(x,M) and In(y,M)) then
                  aantal:=aantal+1;
                fi;
              od;
           fi;
        od;
        if not aantal=1 then
          ok:=false;
```

```
        fi;
      fi;
   od;
od;
return ok;
end;
```

We passen dit alles nu eens toe op $q = 2$ en $q = 4$. Het geval $q = 2$ werken we volledig uit.

```
gap> Read("VeralgVierhoek2.txt");
PG( 2, 2 )
PG( 3, 2 )
Hyperplane( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] )
Subspace of PG( 3, 2 ) by intersection of:
[ Hyperplane( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ) ]
 PG embedded by Subspace of PG( 3, 2 ) by intersection of:
[ Hyperplane( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ) ]
Quadric of PG( 2, 2 )
x_0^2+x_1*x_2
punten1 :=[ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ] ),
  ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ] ) ]
punten2 :=[  ]
punten3 :=[ Subspace of PG( 3, 2 ) by intersection of:
    [ Hyperplane( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ) ] ]
rechten1 :=[ Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
```

```
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ] ) ],
  Subspace of PG( 3, 2 ) spanned by:
    [ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ] ),
      ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ] ) ] ]
rechten2 :=[ ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ] )
    , ProjectivePoint( PG( 3, 2 ), [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ] ),
  ProjectivePoint( PG( 3, 2 ), [ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ] ) ]
gap> incidentie1(2);
true
gap> incidentie2(2);
true
gap> incidentie3(2);
true
```

De uitwerking van $q = 4$ zou te omslachtig zijn, daarom geven we enkel de controles om de incidenties na te gaan.

```
gap> q;
4
gap> incidentie1(4);incidentie2(4);incidentie3(4);
true
true
true
```

Hoofdstuk 3

Generatoren van hermitische variëteiten en kwadrieken

In dit hoofdstuk beschrijven we de implementatie van methodes om de generatoren van hermitische variëteiten en kwadrieken te berekenen. Als toepassing van deze nieuwe methodes en de nieuwe methodes uit het vorige hoofdstuk, construeren we een maximale partiële spread van de hermitische variëteit H$(5, 9)$, die we verkrijgen door de ruimte W$(5, 3)$ in te bedden in H$(5, 9)$. Nadien construeren we hieruit een nieuwe maximale partiële spread van H$(5, 9)$. We herhalen de eerste constructie in 7 dimensies: we bedden de ruimte W$(7, 2)$ in in de ruimte H$(7, 4)$, en we tonen aan dat de verkregen partiële spread van H$(7, 4)$ maximaal is.

3.1 Generatoren

3.1.1 Hermitische variëteiten

Definitie 3.1.1.
Is $F(X_0, X_1, \ldots, X_n)$ een homogeen polynoom van graad m, $m \geqslant 1$, in $K[X_0, X_1, \ldots, X_n]$, K een veld, dan is V(F) de verzameling van alle punten $(x_0, x_1, \ldots, x_n)$ van PG(n, K) waarvoor $F(x_0, x_1, \ldots, x_n) = 0$.
Een **hermitische variëteit** H van PG(n, K), $n \geqslant 1$, is elke verzameling punten van PG(n, K) van de gedaante V(F) met

$$F = \sum_{i,j=0}^{n} a_{ij} X_i Y_j^{\theta},$$

waarbij $\theta \in$Aut K met $\theta \neq 1$ en $\theta^2 = 1$, en $a_{ij} \in K$ niet alle nul met $a_{ij}^{\theta} = a_{ji}$ en K een veld.

We werken hier enkel over eindige velden, dus $K = GF(q)$. Het Galoisveld $GF(q)$, bezit slechts een involutorisch automorfisme $\theta \neq 1$, als en slechts als q een kwadraat is. We noteren $K =$

$GF(q^2)$, dat dan een uniek involutorisch veldautomorfisme bezit, namelijk:

$$\theta : GF(q^2) \to GF(q^2) : x \mapsto x^q$$

Een willekeurige hermitische variëteit noteren we als $H(n, q^2)$ of als H_n.

Definitie 3.1.2.
Een **generator** van een hermitische variëteit H is een deelruimte van maximale dimensie bevat in H. De dimensie van een generator wordt genoteerd met $g = g(H)$.

De dimensie van een generator is gekend dankzij volgende stelling

Stelling 3.1.3. *Is H_n een niet-singuliere hermitische variëteit in $PG(n, q^2)$ en is H_{n-2} een niet-singuliere hermitische variëteit in $PG(n-2, q^2)$, $n \geqslant 3$, dan is $g(H_n) = g(H_{n-2}) + 1$. Bovendien is*

$$g(H_n) = [\frac{n-1}{2}].$$

Bewijs. voor een bewijs zie cursus meetkunde [7] □

Een singuliere variëteit $H(n, q^2)$ is een kegel $\pi_{n-r-1}H'(r, q^2)$, met als top de $(n-r-1)$-dimensionale deelruimte der singuliere punten en als basis een niet-singuliere hermitische variëteit in $\mathrm{PG}(r, q^2)$. Voor een singuliere hermitische variëteit kunnen we bovenstaande stelling gebruiken in combinatie met de volgende.

Stelling 3.1.4. *Is $H = \pi_k H'$, met H geen hypervlak, een singuliere hermitische variëteit met top π_k en basis H', dan is elke generator van H van de gedaante $\pi_k\pi'$, met π' een generator van H' en omgekeerd.*

Bewijs. voor een bewijs zie cursus meetkunde [7] □

De generatoren van een singuliere hermitische variëteit $H(n, q^2) = \pi_{n-r-1}H(r, q^2)$ bepalen komt dus neer op het bepalen van de generatoren van de niet-singuliere basis $H(r, q^2)$. We zullen ons daarom ook concentreren op het construeren van alle generatoren van een niet-singuliere hermitische variëteit $H(r, q^2)$.
Is de dimensie van de generatoren gelijk aan 0, dan zijn de generatoren van $H_r = H(r, q^2)$ juist de punten van H_r. Indien $g(H_r) > 0$ zullen we eerst één generator van de niet-singuliere hermitische variëteit H_r bepalen. Daartoe kiezen we een willekeurig punt p van H_r. Het raakhypervlak door p snijdt H_r in een kegel $pH(r-2, q^2)$, of dus $T_p(H_r) \cap H_r = pH_{r-2}$. Elke generator door p wordt opgespannen door het punt p en een generator van H_{r-2}. We kiezen een willekeurig punt p' in pH_{r-2}. Doordat dit punt gelegen is in het raakhypervlak door p weten we dat de rechte $< p, p' >$ volledig bevat is in H_r. Is de dimensie van de generatoren 1, dan hebben we nu al een generator van H_r gevonden, anders nemen we een punt $p'' \in T_{p'}(H_r) \cap pH_{r-2} \setminus < p, p' >$ en beschouwen de deelruimte $< p, p', p'' >$. We herhalen dit totdat we een deelruimte gevonden

hebben van dimensie $g(H_r)$.
Stel dat we nu een generator π_g kennen. We kunnen de stabilisatorgroep van $\mathrm{H}(r, q^2)$ gemakkelijk berekenen dankzij de methodes die daarvoor geschreven zijn in [1]. Het is geweten dat de stabilisatorgroep van een hermitische variëteit transitief werkt op de generatoren. Als we nu de werking van die groep op de generatoren hebben, dan kunnen we de baan van π_g onder de groep berekenen. Doordat de groep transitief is bevat de baan van π_g juist alle generatoren. De actie van de stabilisatorgroep is beschikbaar via `AsPermutationGroup`. Deze permutatiegroep stelt ons in staat om in GAP de baan van één generator te bepalen. We moeten enkel de berekende generator omzetten in een verzameling projectieve punten, waarna we deze verzameling omzetten in een verzameling nummers. Deze werkwijze zorgt ervoor dat we de generatoren als verzameling van nummers van projectieve punten hebben. Met het oog op latere toepassingen, waarin het handig kan zijn de generatoren juist zo gerepresenteerd te hebben, voorzien we twee attributen: `GeneratorsAsSetsOfNumbers` en `Generators`.
Indien we vertrokken waren van een niet-singuliere hermitische variëteit zullen we nu alle generatoren geconstrueerd hebben. De declaratie van de attributen:

```
DeclareAttribute("GeneratorsAsSetsOfNumbers", IsVariety);

DeclareAttribute("Generators", IsVariety);
```

De methodes gaan dan als volgt:

```
InstallMethod(GeneratorsAsSetsOfNumbers,
"for Hermitian varieties",
true,
[IsHermitianVariety and IsHermitianVarietyRep],
0,
function(H)
local gens,gensn,r,g1,g1n,g,pts,p,bpts,i,cone,remove,Q,H1,n,S,sing,gen;
n:=FamilyObj(H)!.n;
r:=Nature(H);
g:=QuoInt((r-1),2);
```

`QuoInt(`m`,`n`)` `F`

bepaalt het gehele deel van de deling $\frac{m}{n}$, wat overeenkomt met wat wij als $[m/n]$ genoteerd hebben.

```
if g=0 then
  gens := ShallowCopy(PointToNumber(BasePoints(H)));
  for i in[1..Length(gens)] do
    gens[i] := [gens[i]];
  od;
```

Zoals reeds opgemerkt zijn de generatoren juist de punten van de variëteit indien $g = 0$. Als representatie van de generatoren willen we een verzameling van punten hebben, wat hier resulteert dat we elk punt apart beschouwen als een verzameling van punten. Aangezien we alleen de generatoren van de niet-singuliere variëteit H_r beschouwen, maken we gebruik van de functie `BasePoints` die alle punten van de niet-singuliere basis van H_n berekent.

```
else
  pts := [];
  bpts := BasePoints(H);
  for i in [1..g+1] do
    p := bpts[1];
    Add(pts,p);
    cone := Intersection(ProjectivePoints(TangentHyperplane(H,p)),bpts);
    remove := ProjectivePoints(SubspaceOfPG(pts));
    bpts := Difference(cone,remove);
  od;
  g1 := SubspaceOfPG(pts);
  g1n := Set(PointToNumber(ProjectivePoints(g1)));
  Q := AsPermutationGroup(stabilizerGroup(H));
  H1 := stabilizer(Q,Set(PointToNumber(BasePoints(H))),OnSets);
  gens := Orbit(H1,g1n,OnSets);
fi;
return gens;
end);
```

`Q` bevat de permutatievoorstelling van de stabilisatorgroep van $H_n = \pi_{n-r-1}H_r$. We willen echter de deelgroep hiervan hebben die H_r stabiliseert als verzameling punten. Daarom maken we gebruik van het volgende commando:

`stabilizer(`*G,Set,OnSets*`) F`

dat de deelgroep van G bepaalt die *Set* als verzameling invariant laat.

Om de generatoren als deelruimte te hebben, roepen we eerst voorgaande methode op, waarna we de verzameling nummers omzetten in een verzameling punten. Deze punten zijn net alle punten bevat in een generator, en we gebruiken deze verzameling als argument voor `SubspaceOfPG`. Dit gaat als volgt:

```
InstallMethod(Generators,
"for Hermitian varieties",
true,
[IsHermitianVariety and IsHermitianVarietyRep],
0,
```

```
function(H)
local gensn,gens,sub,pts,g,P,n,q;
gensn := GeneratorsAsSetsOfNumbers(H);
gens := [];
n:=FamilyObj(H)!.n;
q:=FamilyObj(H)!.q;
P := ProjectiveGeometry(n,q);
for g in gensn do
  pts := NumberToPoint(P,g);
  sub := SubspaceOfPG(pts);
  Add(gens,sub);
od;
return gens;
end);
```

3.1.2 Kwadrieken

Definitie 3.1.5.
Een **kwadriek** Q in PG(n, q), $n \geqslant 1$, is elke verzameling punten in PG(n, q) van de gedaante V(F) met

$$F = \sum_{i,j=0, i \leqslant j}^{n} a_{ij} X_i Y_j,$$

De redenering om de generatoren van een kwadriek te berekenen verloopt analoog als voor de hermitische variëteiten, enkel de dimensie van de generatoren vereist nog wat aandacht.

De stelling over generatoren van een singuliere hermitische variëteit (stelling 3.1.4) geldt ook voor kwadrieken. We zullen ons beperken tot de niet-singuliere kwadrieken. Een kwadriek kan door een coördinatentransformatie in een kwadriek in standaardvorm omgezet worden. Deze heeft dan één van de volgende drie gedaantes:

$$X_0X_1 + X_2X_3 + \ldots + X_{n-1}X_n$$
$$X_0^2 + X_1X_2 + \ldots + X_{n-1}X_n$$
$$g(X_0, X_1) + X_2X_3 + \ldots + X_{n-1}X_n$$

met $g(X_0, X_1)$ irreducibel over GF(q). Een kwadriek waarvan de vergelijking reduceert tot voorgaande vergelijkingen noemen we respectievelijk hyperbolisch, parabolisch of elliptisch. Voor een niet-singuliere parabolische kwadriek is n noodzakelijk even, bij de andere oneven. We spreken ook over het **karakter** van een kwadriek en noteren dit met $w = 0, 1, 2$ als resp. Q elliptisch, parabolisch of hyperbolisch is. We hebben dan de volgende stelling over de dimensie van een kwadriek.

Stelling 3.1.6. *Is Q_n een niet-singuliere kwadriek in PG(n, q) en is Q_{n-2} een niet-singuliere kwadriek in PG(n − 2, q) met hetzelfde karakter w als Q_n, $n \geq 3$, dan is $g(Q_n) = g(Q_{n-2}) + 1$. Bovendien is*

$$g(Q_n) = \frac{n - 3 + w}{2}$$

Bewijs. Voor een bewijs zie cursus meetkunde[7] □

De methode zal enkel verschillen van deze voor hermitische variëteiten, daar waar de dimensie van de generatoren bepaald wordt.
De attributen `GeneratorsAsSetsOfNumbers` en `Generators` zijn gedeclareerd voor objecten uit de categorie `IsVariety` waartoe ook de kwadrieken behoren, zodat we gewoon een nieuwe methode moeten installeren.

```
InstallMethod(GeneratorsAsSetsOfNumbers,
"for quadrics",
true,
[IsQuadric and IsQuadricRep],
0,
function(Q)
local gens,gensn,nat,g1,g1n,g,pts,p,bpts,i,cone,remove,H,H1,n,S,sing,gen;
nat := Nature(Q);
n:=FamilyObj(Q)!.n;
g:=(nat[1]+nat[2]-3)/2;
if g=0 then
  gens := ShallowCopy(PointToNumber(BasePoints(Q)));
  for i in[1..Length(gens)] do
    gens[i] := [gens[i]];
  od;
else
  pts := [];
  bpts := BasePoints(Q);
  for i in [1..g+1] do
    p := bpts[1];
    Add(pts,p);
    cone := Intersection(ProjectivePoints(TangentHyperplane(Q,p)),bpts);
    remove := ProjectivePoints(SubspaceOfPG(pts));
    bpts := Difference(cone,remove);
  od;
  g1 := SubspaceOfPG(pts);
  g1n := Set(PointToNumber(ProjectivePoints(g1)));
  H := AsPermutationGroup(stabilizerGroup(Q));
  H1 := stabilizer(H,Set(PointToNumber(BasePoints(Q))),OnSets);
```

```
  gens := Orbit(H1,g1n,OnSets);
fi;
return gens;
end);
```

We installeren een methode die de generatoren van een kwadriek als deelruimten berekent.

```
InstallMethod(Generators,
"for quadrics",
true,
[IsQuadric and IsQuadricRep],
0,
function(Q)
local gensn,gens,sub,pts,g,P,n,q;
gensn := GeneratorsAsSetsOfNumbers(Q);
gens := [];
n:=FamilyObj(Q)!.n;
q:=FamilyObj(Q)!.q;
P := ProjectiveGeometry(n,q);
for g in gensn do
  pts := NumberToPoint(P,g);
  sub := SubspaceOfPG(pts);
  Add(gens,sub);
od;
return gens;
end);
```

Voorbeelden 4.

```
gap> P:=ProjectiveGeometry(3,4);
PG( 3, 4 )
gap> H:=HermitianVariety(P,3);
Hermitian Variety of PG( 3, 4 )
gap> Generators(H);
[ Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ),
  Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ), Subspace of PG( 3, 4 ) ]
gap> Q:=Quadric(P,[2,1]);
```

```
Quadric of PG( 3, 4 )
gap> GeneratorsAsSetsOfNumbers(Q);
[ [ 3 ], [ 2 ], [ 13 ], [ 29 ], [ 67 ] ]
```

3.1.3 Toepassing

aan de hand van de nieuwe functies gaan we de volgende stelling na.

Stelling 3.1.7. *Stel dat π_1, π_2 en π drie verschillende generatoren zijn van $H(2n+1,q^2)$ die 2 aan 2 disjunct zijn. De punten van π gelegen op een rechte van $H(2n+1,q^2)$ die π_1 en π_2 snijdt, vormen een hermitische variëteit $H(n,q^2)$ in π.*

Beschouw een willekeurig punt p van de generator π. Dan moeten we nagaan of er een rechte door p gaat die π_1 en π_2 snijdt en tevens op de hermitische variëteit ligt. Om er voor te zorgen dat alle rechten die we bekijken op die variëteit liggen, nemen we het raakhypervlak $T_p(H)$ in p aan $H(2n+1,q^2)$ en beschouwen de doorsnede van dat raakhypervlak met π_1. Alle rechten door p en een punt van die doorsnede liggen op $H(2n+1,q^2)$. Voor elk punt p' van die doorsnede gaan we na of de rechte $< p,p' > \pi_2$ snijdt. Is dit het geval dan houden we p bij in een lijst.
Na het overlopen van alle punten van π houden we een lijst met punten over waarvan we moeten nagaan of die een hermitische variëteit in π vormen.

Eerst gaan we na of de lengte van de lijst gelijk is aan het aantal punten van een hermitische variëteit $H(n,q^2)$. Gebruiken we de notatie ν_n voor het aantal punten van een hermitische variëteit $H(n,q^2)$, dan geldt:

$$\begin{aligned} \nu_{2m} &= (1+q^3)(1+q^5)\dots(1+q^{2m+1}) \\ \nu_{2m+1} &= (1+q)(1+q^3)\dots(1+q^{2m+1}) \end{aligned}$$

Daarmee weten we natuurlijk nog niet of deze punten ook aan de eigenschappen van een hermitische variëteit voldoen. We zullen dit op 2 manieren nagaan voor $n=2$.

We weten dat elke rechte van $\mathrm{PG}(n,q^2)$ een hermitische variëteit $H(n,q^2)$ snijdt in 1 of $q+1$ punten. Dit is een nodige voorwaarde opdat we met een hermitische variëteit zouden te doen hebben, maar geen voldoende voorwaarde. Unitalen zijn meetkundige objecten waartoe ook de hermitische variëteiten behoren die voldoen aan deze eigenschap. Volgende stelling maakt duidelijk wanneer een unitaal een hermitische variëteit is.

Stelling 3.1.8. *(Thas[1992]) In $PG(2,q^2)$ is een unitaal waarbij de raaklijnen in collineare punten concurrent zijn, een hermitische variëteit.*

Om de voorwaarde in deze stelling te controleren, construeren we voor elk punt in het vlak π de rechten door dit punt en gaan na of die 1 of $q+1$ snijpunten heeft met de verzameling. We houden een lijst bij met de snijpunten van de rechten die maar in 1 punt snijden.

Behoort p tot de verzameling punten waarvan we willen aantonen dat ze een hermitische variëteit vormt, dan gaan we na dat er juist 1 rechte is die die verzameling in slechts 1 punt snijdt, dit is de raakrechte in dat punt.
Als p niet tot die verzameling behoort dan moet die lijst met snijpunten juist $q+1$ punten bevatten. Om na te gaan of deze $q+1$ punten op eenzelfde rechte liggen, moet de ruimte die ze opspannen dimensie 1 hebben.

Deze eerste voorwaarde wordt nagegaan in een functie die we `test1` genoemd hebben. We werken met nummers i.p.v. punten omdat we de lengte van de doorsnede van die lijst met rechten willen weten. De intersectie van twee deelruimten zou ons een nieuwe deelruimte geven, waarvan we dan de dimensie moeten nagaan. Door met nummers te werken, is de intersectie een lijst met nummers, waarvan we gemakkelijk de lengte kunnen bepalen. Daarna roepen we de twee voorgaande testfuncties op en laten het resultaat ervan afdrukken.

```
test1:=function(U,pi)
local p,dummy,nrs,inter,list,lines,S,pts,l,q;
dummy:=PointToNumber(U);
q:=Sqrt(FamilyObj(pi)!.q);
pts:=ProjectivePoints(pi);
for p in pts do
  lines:=SubspacesThroughSubspace(pi,p,1);
  list:=[];
  for l in lines do
    nrs:=PointToNumber(ProjectivePoints(l));
    inter:=Length(Intersection(dummy,nrs));
    if not (inter=1 or inter=q+1) then
      return false;
    elif inter=1 then
      Add(list,Intersection(dummy,nrs));
    fi;
  od;
  if p in U then
    if not Length(list)=1 then
    return false;
    fi;
  else
    if not Length(list)=q+1 then
     return false;
    else
       pts:=NumberToPoint(PGOfSubspace(pi),Concatenation(list));
     S:=SubspaceOfPG(pts);
     if not Dimension(S)=1 then
        return false;
       fi;
```

```
    fi;
  fi;
od;
return true;
end;
```

In de functie `test2` benaderen we voorgaande vanuit een andere hoek.

Een hermitische variëteit is ook bepaald door een hermitische matrix A, waarbij de punten van $\mathrm{PG}(2, q^2)$ op de hermitische variëteit liggen als en slechts als voldaan is aan volgende uitdrukking:

$$p(x_0, x_1, x_2) \in H(2, q^2) \Leftrightarrow (x_0, x_1, x_2) A \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}^q = 0$$

We kunnen bepalen door hoeveel punten de matrix A bepaald wordt. Hierna moeten we controleren of de andere punten voldoen aan bovenstaande vergelijking. De $3 \times 3-$matrix A bevat 9 elementen a_{ij}. Voorgaande vergelijking komt overeen met

$$X * M = (x_0 x_0^q, x_0 x_1^q, x_0 x_2^q, x_1 x_0^q, x_1 x_1^q, x_1 x_2^q, x_2 x_0^q, x_2 x_1^q, x_2 x_2^q) \begin{pmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{33} \end{pmatrix} = 0$$

waarbij uit het hermitische zijn van de matrix A volgt dat moet gelden:

$$\begin{array}{lll} a_{12}^q = a_{21} & a_{13}^q = a_{31} & a_{23}^q = a_{32} \\ a_{11}^q = a_{11} & a_{22}^q = a_{22} & a_{33}^q = a_{33} \end{array}$$

Hierdoor blijven er nog 6 onbekenden over, zodat de oplossing bepaald kan worden door 6 vergelijkingen, d.w.z. dat we 6 punten nodig hebben om de matrix vast te leggen. door het voorkomen van de termen a_{ij}^q, is dit echter een niet lineair stelsel. We zullen gebruik maken van het commando

`NullspaceMat(`*mat*`) A`

dat een lijst van rijmatrices weergeeft die een basis vormen voor de oplossing van de vergelijking $vec * mat = 0$.

Dit commando kan dus een oplossing bepalen van een lineair stelsel. Daarom zullen we de termen a_{ij}^q als onbekenden beschouwen, 9 vergelijkingen opstellen, en de oplossing bepalen indien mogelijk. Als we een oplossing vinden, kunnen we de matrix A bepalen en controleren of dit een hermitische matrix is. Zo ja, dan weten we meteen dat de negen punten een hermitische variëteit vormen.

```
test2:=function(U,pi)
local q,F,G,P,p,i,pts,dummy,M,X,S,vector,dummy2,lijst,s;
q:=FamilyObj(pi)!.q;
F:=ProjectiveGeometry(Dimension(pi),q);
G:=PGOfSubspace(pi);
P:=Embed(F,G,pi);
pts:=ProjectivePoints(F);
dummy:=[];
for p in pts do
  if ProjectivePoint(P,p) in U then
Add(dummy,p);
  fi;
od;
lijst:=[dummy[1],dummy[2],dummy[3],dummy[4],dummy[5],dummy[6]];
x:=[];
q:=Sqrt(q);
for p in dummy do
vector:=p![1];
  Add(x,[vector[1]*vector[1]^q,vector[1]*vector[2]^q,vector[1]*vector[3]^q,
         vector[2]*vector[1]^q,vector[2]*vector[2]^q,vector[2]*vector[3]^q,
         vector[3]*vector[1]^q,vector[3]*vector[2]^q,vector[3]*vector[3]^q]);
od;
x:=MutableTransposedMat(x);
S:=NullspaceMat(x)[1];
if not (S[2]^q=S[4] and S[3]^q=S[7] and S[6]^q=S[8]) then
if not (S[1]^q=S[1] and S[5]^q=S[5] and S[9]^q=S[9]) then
  return false;
fi;
fi;
return true;
end;
```

De functie die de eigenschap uit stelling 3.1.7 nagaat, vertrekt van een hermitische variëteit H en drie natuurlijke getallen i, j en k. De generator π is de i^{de} generator. Voor de generatoren π_1 en π_2 wordt gezocht naar de eerste generator vanaf de j^{de} resp. de k^{de} generator die scheef is aan π, resp. π en π_1.
We doen eerst een test op de dimensie van de ruimte waarin de hermitische variëteit ligt, want de stelling spreekt enkel over oneven dimensie. Hierna roepen we de voorgaande tests op en drukken hun resultaat af.

```
result:=function(i,j,k,H)
local n,m,q,x,y,p,pi,pi1,pi2,h,dummy,F,G,P,gens,
  ptsV,ptspi,ptspi1,pts,V,lines,test,inter,nrs;
```

```
m:=FamilyObj(H)!.n;
q:=FamilyObj(H)!.q;
n:=(m-1)/2;
F:=ProjectiveGeometry(n,q);
G:=ProjectiveGeometry(m,q);
if m mod 2 =0 then
  Error("m must be odd");
fi;
gens:=Generators(H);;
pi:=gens[i];
repeat
  j:=j+1;
  pi1:=gens[j];
until IsEmpty(Intersection(pi,pi1));
repeat
  k:=k+1;
  pi2:=gens[k];
until (IsEmpty(Intersection(pi,pi2)) and IsEmpty(Intersection(pi1,pi2)));
ptspi:=ProjectivePoints(pi);
dummy:=[];
ptspi1:=ProjectivePoints(pi1);
for x in ptspi do
  h:=SubspaceOfPG(TangentHyperplane(H,x));
  pts:=ProjectivePoints(Intersection(pi1,h));
  for y in pts do
    line:=SubspaceOfPG([x,y]);
    if not IsEmpty(Intersection(line,pi2)) then
      Add(dummy,x);
    fi;
  od;
od;
U:=PointToNumber(dummy);
q:=Sqrt(q);
nu:=1;
l:=3;
while (l <= n+1) do
  nu:=nu*(q^l+1);
  l:=l+2;
od;
if not (0 = n mod 2) then
  nu:=nu*(q+1);
fi;
if not (Length(U)= nu) then
  return false;
```

```
fi;
Print("De eerste test heeft als resultaat: " ,test1(dummy,pi),"\n");
Print("De tweede test heeft als resultaat: " ,test2(dummy,pi),"\n");
end;
```

Bij het testen merken we op dat de eerste test veel tijdrovender is dan de tweede test.

Voorbeelden 5.

```
gap> P:=ProjectiveGeometry(5,4);
PG( 5, 4 )
gap> H:=HermitianVariety(P,5);
Hermitian Variety of PG( 5, 4 )
gap> result(1,2,3,H);
De eerste test heeft als resultaat: true
De tweede test heeft als resultaat: true
gap> result(10,20,30,H);
De eerste test heeft als resultaat: true
De tweede test heeft als resultaat: true
gap> result(100,200,300,H);
De eerste test heeft als resultaat: true
De tweede test heeft als resultaat: true
```

3.2 Een maximale partiële spread van $H(5,9)$

Definitie 3.2.1.
Beschouw een n-dimensionale projectieve ruimte PG(n,q) over het eindig veld GF(q). Een **polariteit** β is een permutatie van de verzameling van de deelruimten van PG(n,q) die de relatie bevat zijn in omdraait, d.i.

$$\pi_i \subseteq \pi_j \Leftrightarrow \pi_j^\beta \subseteq \pi_i^\beta \qquad \forall \pi_i, \pi_j \subset PG(n,q)$$

en waarvoor geldt $\beta^2 = id$

De werking van een polariteit β op de puntenverzameling kan volledig beschreven worden door een $(n+1)\times(n+1)$-matrix A over GF(q) en $\theta \in$ Aut GF(q):

$$\beta : PG(n,q) \to PG(n,q)$$

$$p(x_0, x_1, \ldots, x_n)^\beta = A \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix}^\theta$$

We gebruiken de volgende terminologie:

- We noemen de polariteit **symplectisch** als n oneven is, $\theta = \mathbf{1}$ en $A^T = -A$.
- We noemen de polariteit **hermitisch** als $(A^T)^q = A$ en $\theta^2 = \mathbf{1}$, met $\theta \neq \mathbf{1}$.

Definitie 3.2.2.
Een **polaire ruimte** van (eindige) rang n, $n \geqslant 3$, is een verzameling P van elementen, *punten* genoemd, samen met een verzameling deelverzamelingen van P, *deelruimten* genoemd, met volgende eigenschappen:

(i) Een deelruimte samen met de deelruimten die ertoe behoren is isomorf met een PG(d,q), $-1 \leqslant d \leqslant n-1$; men zegt dat zulke deelruimte dimensie d heeft.

(ii) De doorsnijding van elke twee deelruimten is een deelruimte.

(iii) Gegeven zijnde een deelruimte π van dimensie $n-1$ en een punt $p \in P - \pi$, dan bestaat er een unieke deelruimte π' die p bevat zodanig dat de dimensie van $\pi \cap \pi'$ $n-2$ is. De deelruimte $\pi \cap \pi'$ is de verzameling van alle punten van π die samen met p in een deelruimte van dimensie 1 gelegen zijn.

(iv) Er bestaan disjuncte deelruimten van dimensie $n-1$.

Voorbeelden 6.
(a) Een niet-singuliere hermitische variëteit H in PG(n,q^2), $n \geqslant 3$, heeft wegens stelling 3.1.3 deelruimten van dimensie $[\frac{n-1}{2}]$. H vormt dan samen met de puntenverzamelingen van de projectieve deelruimten die op H gelegen zijn een polaire ruimte van rang $[\frac{n+1}{2}]$.
(b) Beschouw een symplectische polariteit β van PG(n,q), n oneven en $n \geqslant 3$. Dan vormt de puntenverzameling van PG(n,q) samen met de puntenverzamelingen van de absolute deelruimten t.o.v. β in PG(n,q) (dit zijn de deelruimten π waarvoor geldt $\pi \subseteq \pi^\beta$) een polaire ruimte met rang $(n+1)/2$.

Een **spread** van een polaire ruimte is een verzameling onderling scheve generatoren die een partitie vormen van de puntenverzameling. Door een resultaat van J.A. Thas [2] is geweten dat een hermitische variëteit H$(2n+1, q^2)$ over het veld F_{q^2} geen spreads bezit. De volgende stap is dan het zoeken naar maximale partiële spreads. dit is een verzameling generatoren die nog altijd onderling disjunct zijn, maar die geen partitie van de puntenverzameling van de ruimte meer vormen en waaraan geen generator meer kan toegevoegd worden. Om een maximale partiële spread te construeren maken we de volgende beschouwing.

We vertrekken van de inbedding $PG(5,q) \subset PG(5,q^2)$ en beschouwen de matrix A

$$A = \begin{bmatrix} 0 & \epsilon & 0 & 0 & 0 & 0 \\ -\epsilon & 0 & 0 & 0 & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & \dots & \dots & 0 & \epsilon \\ 0 & 0 & \dots & \dots & -\epsilon & 0 \end{bmatrix},$$

waarbij $\epsilon \in GF(q^2)\backslash GF(q)$ waarvoor geldt $\epsilon^q = -\epsilon$. Door de keuze van onze ϵ is het onmiddellijk duidelijk dat A aan de volgende eigenschappen voldoet:

$$A^T = -A \quad \text{en} \quad (A^T)^q = A.$$

Hierdoor kan en beschouwd worden als de matrix van zowel een symplectische als een hermitische polariteit.

- Stel de hermitische polariteit $\phi : PG(5,q^2) \to PG(5,q^2)$:

$$\begin{cases} (A^T)^q = A \\ \theta : x \mapsto x^q \end{cases}$$

  Deze polariteit bepaalt een hermitische variëteit $H = V(F)$ in PG$(5,q^2)$, met $F = X_0X_1^q - X_0^qX_1 + \ldots + X_4X_5^q - X_4^qX_5$.

- Het is duidelijk dat A een symplectische polariteit φ van PG$(5,q^2)$ bepaalt. Indien we de restrictie van φ beschouwen tot de Baer-deelmeetkunde PG$(5,q^2)$ die we verkrijgen door enkel coördinaten over GF(q) toe te laten, dan is het duidelijk dat φ een symplectische polariteit van PG$(5,q)$ is met matrix $\frac{1}{\epsilon}A$. Deze matrix zal ook een symplectische polariteit bepalen $\varphi : PG(5,q) \to PG(5,q)$:

$$\begin{cases} (A^T) = -A \\ \theta = id \end{cases}$$

  De polaire ruimte bepaald door deze polariteit noteren we als W$(5,q)$. W$(5,q)$ is dus opgebouwd uit alle deelruimten van PG$(5,q)$ die absoluut zijn t.o.v. φ.

Dat de punten van W$(5,q)$juist alle punten van PG$(5,q)$ zijn, zien we als volgt. Stel $p(x_0,\ldots,x_5) \in PG(5,q)$ dan is $x_i \in GF(q), \forall i = 0,1,\ldots,5$.

$$p \in p^\varphi \Leftrightarrow (x_0,\ldots,x_5)A\begin{pmatrix} x_0 \\ \vdots \\ x_5 \end{pmatrix} = 0$$

$\Leftrightarrow$

$$-x_1x_0 + x_0x_1 - .. - x_5x_4 + x_4x_5 = 0$$

Stelling 3.2.3. *W(5,q) $\subset$ H(5,q^2) .*

Bewijs. Stel $p(x_0,\ldots,x_5) \in PG(5,q)$ dan is $x_i \in GF(q), \quad \forall i = 0,1,\ldots,5$. Opdat p op $H(5,q^2)$ zou liggen moet gelden:

$$p \in p^\phi \Leftrightarrow (x_0,\ldots,x_5)A\begin{pmatrix} x_0 \\ \vdots \\ x_5 \end{pmatrix}^q = 0$$

Maar $x_i \in GF(q)$, dus we hebben $x_i^q = x_i$ en dan wordt het voorgaande:

$$(x_0, \ldots, x_5)A \begin{pmatrix} x_0 \\ \vdots \\ x_5 \end{pmatrix} = 0$$

waarvan we al weten dat dit geldt. We beschouwen nu een rechte L van $\mathrm{W}(5,q)$. Stel $L = pr$ met $p(p_0, \ldots, p_5)$ en $r(r_0, \ldots, r_5)$. We moeten aantonen dat $L = pr \subset L^\phi = pr^\phi = p^\phi \cap r^\phi$. Als $p \in p^\phi \cap r^\phi$ en $r \in p^\phi \cap r^\phi$, dan ligt de volledige rechte L in zijn beeld. We weten reeds dat $p \in p^\phi$ en we tonen aan dat ook $r \in p^\phi$ geldt.

$$(r_0, \ldots, r_5)A \begin{pmatrix} p_0 \\ \vdots \\ p_5 \end{pmatrix}^q = 0$$

$\Leftrightarrow$

$$(r_0, \ldots, r_5)A \begin{pmatrix} p_0 \\ \vdots \\ p_5 \end{pmatrix} = 0$$

$\Leftrightarrow$

$$-r_1p_0 + p_0r_1 - \ldots - r_5p_4 + p_4r_5 = 0$$

Hieruit volgt dat $pr \subset p^\varphi$. Analoog tonen we $pr \subset r^\varphi$ aan en daarmee is het gestelde bewezen. □

Het is geweten dat de symplectische polaire ruimte een spread bezit. De generatoren van $\mathrm{W}(5,q)$ zijn vlakken. We weten al dat de punten van $\mathrm{W}(5,q)$ alle punten van $\mathrm{PG}(5,q)$ zijn en dus bezit $\mathrm{W}(5,q)$ juist $\frac{q^6-1}{q-1} = (q^3+1)(q^2+q+1)$ punten. Daar elk vlak (q^2+q+1) punten bevat, moet de spread dus uit q^3+1 vlakken bestaan. We zullen nagaan dat deze q^3+1 vlakken een maximale partiële spread van $\mathrm{H}(5,q^2)$ vormen (zie [8]).

Zoals reeds aangegeven willen we enkele nieuwe resultaten aantonen over maximale partiële spreads van $\mathrm{H}(5,q^2)$ met $q = 3$. De inbedding van $\mathrm{PG}(5,3)$ in $\mathrm{PG}(5,9)$ vertaalt zich in de volgende **GAP**-code.

```
LoadPackage("pg");
n := 5;q := 3;
P1 := ProjectiveGeometry(n,q);
P2 := ProjectiveGeometry(n,q^2);
P3 := Embed(P1,P2);
subpts := ProjectivePoints(P3);;
subptsn := PointToNumber(subpts);
```

We beschouwen de punten van $\mathrm{W}(5,q)$ als punten van $\mathrm{PG}(5,q^2)$. Om er gemakkelijk mee te kunnen werken, werken we met de nummers van die punten die we opslaan in `subptsn`. Hierna

zoeken we het eerste element $\epsilon \in GF(q^2)\backslash GF(q)$ waarvoor $\epsilon^q = -\epsilon$ en construeren de matrix A en de corresponderende hermitische variëteit. We willen de deelgroep van de stabilisatorgroep van $\mathrm{H}(5,q^2)$ bepalen die de punten van $\mathrm{W}(5,q)$ stabiliseert als verzameling. Daartoe moeten we de stabilisatorgroep eerst omzetten in een permutatiegroep, zodat we dan de deelgroep kunnen bepalen die de verzameling nummers van de punten van $\mathrm{W}(5,q)$ stabiliseert als verzameling.

```
list := Difference(List(GF(q^2)),List(GF(q)));
e :=  First(list,x->x^q=-x);
mat := [[0,1,0,0,0,0],[-1,0,0,0,0,0],[0,0,0,1,0,0],[0,0,-1,0,0,0],
[0,0,0,0,0,1],[0,0,0,0,-1,0]]*e;
h := HermitianVariety(P2,mat);
Set(List(subpts,x->x in h));
G := stabilizerGroup(h);
H := AsPermutationGroup(G);
subptsn := Set(subptsn);
S := stabilizer(H,subptsn,OnSets);
mat2 := [[0,1,0,0,0,0],[-1,0,0,0,0,0],[0,0,0,1,0,0],[0,0,-1,0,0,0],
[0,0,0,0,0,1],[0,0,0,0,-1,0]]*Z(q)^0;
phi := Polarity(P1,0,mat2);
```

`Polarity(`P`,`i`,`mat`)`[1]

Dit commando bepaalt de polariteit van $P \to P$ met matrix mat en automorfisme $\theta : x \mapsto x^{p^i}$. We willen de polariteit die $\mathrm{W}(5,q)$ absoluut laat bepalen dus zal P de projectieve ruimte $\mathrm{PG}(5,3)$ zijn. Daar in ons geval $q = 3 = p$ is en we voor θ de identieke afbeelding gebruiken, is $i = 0$. Zoals reeds opgemerkt gebruiken we de matrix $\frac{1}{\epsilon}A$ om de polariteit te construeren.
Om nu een spread van $\mathrm{W}(5,q)$ te construeren, vertrekken we van 1 generator. Daartoe nemen we een willekeurig punt p_1 van $\mathrm{W}(5,q)$ en bepalen zijn beeld p_1^φ, waaruit we een tweede punt $p_2 \neq p_1$ nemen. Dan weten we al dat de rechte $< p_1, p_2 >$ in $\mathrm{W}(5,q)$ ligt, want wegens constructie geldt

$$p_2 \in p_1^\varphi \Rightarrow p_1^{\varphi^2} = p_1 \in p_2^\varphi$$

en dus

$$p_1 p_2 \in p_1^\varphi \cap p_2^\varphi.$$

Een derde punt p_3 zoeken we in de doorsnede van de hypervlakken p_1^φ en p_2^φ, maar niet op de rechte $< p_1, p_2 >$. We zien gemakkelijk in dat deze drie punten een vlak opspannen dat in $\mathrm{W}(5,q)$ gelegen is.

```
pts := ProjectivePoints(P1);;
p1 := pts[1];
h1 := p1^phi;
choices := Difference(ProjectivePoints(h1),[p1]);;
```

[1]Deze operatie is het resultaat van een vorig thesisproject. Dit werd echter tot op heden nog niet afgewerkt.

```
p2 := choices[1];
h2 := p2^phi;
choices := Intersection(ProjectivePoints(h1),ProjectivePoints(h2));;
choices := Difference(choices,ProjectivePoints(SubspaceOfPG([p1,p2])));;
p3 := choices[1];
plane := SubspaceOfPG([p1,p2,p3]);
extendedplane := SubspaceOfPG(P3,plane);
```

We doen eerst twee tests zodat we zeker zijn dat het vlak dat we geconstrueerd hebben wel degelijk absoluut is t.o.v. φ. Eenmaal door te kijken of $\pi^\varphi = \pi$ en andermaal door te controleren of alle punten van π in $\mathrm{H}(5,q^2)$ liggen, want als het absoluut is, ligt het in $\mathrm{W}(5,q)$ en dus ook in $\mathrm{H}(5,q^2)$.

```
gap> Set(List(ProjectivePoints(extendedplane),x->x in h));
[ true ]
gap> plane^phi=plane;
true
```

We hebben nu een generator van $\mathrm{W}(5,q)$ en kunnen alle generatoren berekenen op dezelfde wijze als bij de variëteiten, nl. door de baan te berekenen van de gevonden generator onder de stabilisatorgroep van $W(5,q)$. De spread die we zullen construeren zal een zogenaamde reguliere spread zijn. Met een generator en een cyclische groep, deelgroep van de symplectische groep, van orde $q^3+1 = 28$, bepalen we het beeld van die generator onder die cyclische groep en dit zal ons een spread opleveren.

`Action(`*G, Omega, act*`) O`

bepaalt een permutatiegroep op | *Omega* | elementen die de actie van G op *Omega* weergeeft. Door aan elk element in *Omega* een nummer toe te kennen, kunnen we de actie van G onmiddellijk zien als een permutatie van gehele getallen. We bepalen de actie van de stabilisatorgroep `S` van $\mathrm{W}(5,q)$ op de generatoren, waarbij dus aan elke generator een nummer toegekend wordt. In deze permutatiegroep zoeken we een element van orde 28 en bepalen de banen van dit element. Er moet minstens één baan van lengte 28 zijn. We controleren nadien of er één is die een spread vormt. We zullen zien dat alle banen lengte 28 hebben en we dus de eerste baan moeten zoeken die de puntenverzameling partitioneert.

```
gap> planeptsn := Set(PointToNumber(ProjectivePoints(extendedplane)));
[ 1, 4, 6, 1198, 1912, 2085, 3646, 3661, 5940, 7344, 7441, 8650, 9230, 11983,
  12064, 12353, 12518, 13352, 14581, 16780, 17422, 17482, 18641, 18758,
  19258, 19383, 19863, 19904, 20423, 21233, 21624, 23228, 23239, 23737,
  24023, 25167, 25675, 25753, 25895, 27712, 28956, 29402, 30050, 30886,
  31227, 33365, 33926, 36534, 37378, 38648, 41242, 44309, 45845, 45878,
  45901, 46185, 46929, 47630, 48230, 48273, 48636, 48722, 50526, 50938,
```

```
  51119, 51461, 52045, 52165, 52235, 52890, 53157, 53208, 54151, 54421,
  57300, 58533, 58949, 59313, 59335, 60083, 60132, 60226, 60547, 61417,
  61885, 61986, 62285, 63569, 64184, 64908, 66058 ]
gap> wgens:=Orbit(S,planeptsn,OnSets);;
gap> Sact := Action(S,wgens,OnSets);
<permutation group with 13 generators>
gap> o := 0;
0
gap> while o<>28 do
>   g := Random(Sact);
>   o := Order(g);
> od;
gap> groep := Group([g]);
<permutation group with 1 generators>
gap> orbits := Orbits(groep);;
gap> List(orbits,x->Length(x));
[ 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
  28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
  28, 28 ]
```

We hebben telkens $q^3 + 1 = 28$ vlakken. De enige voorwaarde opdat die vlakken een spread zouden vormen is dat ze onderling disjunct zijn. Als we weten dat de unie van alle punten van die vlakken juist alle punten van $\mathrm{W}(5,q)$ zijn, dan weten we dus dat deze vlakken een spread vormen. Geen enkel vlak kan dan een ander snijden, want dan zouden deze vlakken niet alle punten van $\mathrm{W}(5,q)$ kunnen bevatten.
We merken op dat we de vlakken als vlakken van $\mathrm{H}(5,q^2)$ beschouwen. Daardoor moeten we de unie van de punten beschouwen die in de doornede van die vlakken met $\mathrm{PG}(5,3)$ liggen. Bevat de unie alle punten van $\mathrm{PG}(5,3)$ dan hebben we een spread.

```
gap> is_spread := function(tspread)
> local pts,p;
> pts := [];
>   for p in tspread do
>   pts := Union(pts,Intersection(wgens[p],subptsn));
> od;
> return Set(pts)=subptsn;
> end;
function( tspread ) ... end
gap> spread := Set(orbits[1]);
[ 1, 26, 30, 100, 104, 140, 219, 250, 254, 289, 374, 403, 444, 448, 501, 514,
  607, 693, 700, 702, 801, 887, 889, 953, 986, 1022, 1032, 1065 ]
gap> is_spread(spread);
false
```

```
gap> spread := Set(orbits[10]);
[ 11, 47, 53, 58, 267, 283, 313, 324, 378, 393, 401, 437, 474, 485, 502, 509,
  614, 675, 677, 708, 722, 909, 979, 994, 995, 1014, 1027, 1060 ]
gap> is_spread(spread);
true
```

Baan 10 levert de gezochte spread op.

Veronderstel nu dat S een maximale partiële spread van $H(5,q^2)$ is, die we verkregen hebben door een spread van $W(5,q)$ in te bedden. Eerst voeren we enkele nieuwe benamingen in. De punten die op een vlak van $\mathcal{S}$ liggen noemen we **gebonden** punten, de andere punten zijn dan vanzelfsprekend **ongebonden**. Elk hermitisch vlak, d.i. een vlak van $\mathrm{H}(5,q^2)$, bevat een gebonden punt, anders zou de spread niet maximaal zijn. Anderzijds kunnen we ook opmerken dat een hermitisch vlak dat niet tot $\mathcal{S}$ behoort ten hoogste een rechte met een vlak van $\mathcal{S}$ gemeen kan hebben. De hermitische vlakken niet in $\mathcal{S}$ die geen rechte gemeen hebben met een vlak van $\mathcal{S}$ zullen we voortaan **vrije** vlakken noemen. In [2] wordt aangetoond dat elk vrij vlak q^2-q+1 gebonden punten bevat.
We werken in het geval van $q=3$ en dus zal $\mathcal{S}$ juist $q^3+1=28$ vlakken bevatten. Stel dat T een verzameling van $q^2-q+1=7$ vlakken van $\mathcal{S}$ bevat dan duiden we met $F(T)$ de verzameling vrije vlakken van $\mathrm{H}(5,q^2)$ aan die elk vlak van T snijdt. Het vermoeden is dat deze verzameling 7 vlakken bevat die onderling scheef zijn en met uitzondering van de vlakken van T geen enkel ander vlak van $\mathcal{S}$ sijnden. We kunnen nu de 7 vlakken van $\mathcal{S}$ vervangen door die 7 vlakken uit $F(T)$ en zo bekomen we een nieuwe spread van grootte 28, die we een **afgeleide spread** zullen noemen. We zeggen dan dat $F(T)$ aan eigenschap D voldoet.

Om dit vermoeden in **GAP** te controleren gaan we op zoek naar een generator f van $\mathrm{H}(5,q^2)$ die juist 7 vlakken van $\mathcal{S}$ snijdt in een punt. We maken daarvoor gebruik van de operatie:

`Random(` *list* `)` `O`

Random geeft een (pseudo-)random element weer van de lijst *list*. Er wordt ad random een generator van $\mathrm{H}(5,q^2)$ gekozen en nagegaan hoe die generator de spread snijdt.

```
gap> hgens:=GeneratorsAsSetsOfNumbers(h);;
gap> f := Random(hgens);
[ 1100, 1455, 1474, 1714, 2721, 3049, 3340, 3851, 4292, 4306, 5198, 6502,
  7991, 8180, 8358, 9073, 9610, 9827, 10610, 11802, 12034, 12641, 14592,
  15167, 15527, 16643, 19178, 19596, 20619, 22548, 23389, 24634, 26693,
  28299, 28329, 28409, 28951, 28974, 29317, 29362, 31027, 31346, 31685,
  32408, 33079, 34585, 34969, 36357, 37840, 38279, 38528, 38864, 39072,
  39291, 39569, 39710, 39902, 40451, 40891, 42286, 43459, 44687, 44990,
  47276, 47978, 50221, 50400, 50701, 50888, 52022, 52071, 52228, 53164,
  54088, 54138, 54519, 56513, 56847, 59810, 60075, 60103, 61486, 61723,
  61884, 62065, 62122, 62855, 63681, 64606, 64950, 65226 ]
```

```
gap> l := [];
[  ]
gap> for p in spread do
>   Add(l,Length(Intersection(wgens[p],f)));
> od;
gap> l;
[ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
  1, 0, 0 ]
```

De eerste keuze is meteen al een goede. Nu wordt bepaald welke vlakken van $\mathcal{S}$ f snijden in een punt. Dit is de verzameling T. Deze verzameling T zal een lijst met nummers bevatten die de positie aanduidt van de vlakken van T in de lijst met generatoren van $\mathrm{W}(5,q)$. Het is de bedoeling dat we de positie kennen in de lijst van generatoren van $\mathrm{H}(5,q^2)$. Daartoe zullen we de nummers van T omzetten in een lijst Tn en de positie bepalen van het vlak f.

```
gap> T := [];
[  ]
gap> for p in spread do
>   if Length(Intersection(wgens[p],f))=1 then
>     Add(T,p);
>   fi;
> od;
gap> T;
[ 58, 401, 437, 485, 675, 722, 1014 ]
gap> Tn := [];
[  ]
gap> for p in T do
>   Add(Tn,Position(hgens,wgens[p]));
> od;
gap> Tn;
[ 11406, 6959, 8554, 6003, 20761, 27291, 22266 ]
gap> fn := Position(hgens,f);
7871
```

De stabilisatorgroep van $\mathrm{H}(5,q^2)$ bepaalt een actie op de generatoren van $\mathrm{H}(5,q^2)$. We zoeken naar de deelgroep van deze permutatiegroep die de verzameling T als verzameling stabiliseert. De incidentie blijft bewaard onder die permutatie en dus zal het beeld van f een generator zijn die de vlakken van T ook snijdt in een punt. We moeten 7 elementen zoeken in de baan van f die onderling disjunct zijn. We vertrekken van het eerste element en zien dat er nog juist 6 elementen zijn die scheef hieraan zijn. Deze zullen ook onderling scheef zijn.

```
gap> Sacth := Action(H,hgens,OnSets);
<permutation group with 9 generators>
gap> Sacth1 := stabilizer(Sacth,Set(Tn),OnSets);
<permutation group of size 588 with 5 generators>
gap> fsn := Orbit(Sacth1,fn);
[ 7871, 8201, 8753, 839, 5549, 8158, 21827, 2132, 7223, 23243, 6475, 20120,
  12109, 4721 ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[2]]);
[ 54088 ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[3]]);
[ 3340 ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[5]]);
[  ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[6]]);
[  ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[8]]);
[  ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[9]]);
[  ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[11]]);
[  ]
gap> Intersection(hgens[fsn[1]],hgens[fsn[12]]);
[  ]
gap> nfsn:=[1,5,6,8,9,11,12];
[ 1, 5, 6, 8, 9, 11, 12 ]
gap> ofsn := fsn{nfsn};
[ 7871, 5549, 8158, 2132, 7223, 6475, 20120 ]
```

Rest ons nu nog de vlakken van T te vervangen door deze 7 vlakken en na te gaan of we een maximale partiële spread hebben. Nu kunnen we dit niet doen door na te gaan of de unie van de punten gelegen in een vlak van de nieuwe spread alle punten van $\mathrm{H}(5,q^2)$ zijn, want we werken met een partiële spread. We werken in $\mathrm{PG}(5,q^2)$, met $q=3$, en dus zal elk vlak $q^4+q^2+1=91$ punten bevatten. Opdat we een maximale partiële spread zouden hebben, moeten de vlakken dus $(q^3+1)(q^4+q^2+1)=28.91=2548$ punten bevatten.

```
true
gap> original_spread := [];
[  ]
gap> for p in spread do
>   Add(original_spread,Position(hgens,wgens[p]));
> od;
gap> new_spread := Union(Difference(original_spread,Tn),ofsn);
[ 238, 1572, 2132, 3737, 5549, 6475, 7223, 7342, 7871, 8158, 8627, 8649,
```

```
  10020, 10144, 11089, 12168, 15476, 16654, 19166, 20120, 20926, 21950,
  22786, 22831, 23537, 24979, 25844, 27210 ]
gap> is_spread2 := function(tspread)
> local pts,p;
> pts := [];
>   for p in tspread do
>   pts := Union(pts,hgens[p]);
> od;
> pts:=Set(pts);
> return Length(pts)=(q^3+1)*(q^4+q^2+1);
> end;
function( tspread ) ... end
gap>
gap> is_spread2(new_spread);
true
```

We hebben een nieuwe maximale partiële spread van H$(5, q^2)$ gevonden!

3.3 Een maximale partiële spread van $H(7,4)$

In voorgaande sectie hebben we een spread van W$(5, q)$ geconstrueerd en we zijn dan nagegaan of dit een maximale partiële spread van $H(5, q^2)$ is. Dit is een resultaat dat geldt als de dimensie $4n + 1$ is. Over dimensie $4n + 3$ is enkel geweten dat een spread van $W(4n + 3, q)$ een partiële spread is van $H(4n + 3, q^2)$. We onderzoeken het kleinste geval, waarbij de dimensie gelijk is aan 7 en $q = 2$. De constructie van de spread in W$(7, 2)$ gebeurt zoals we deden in dimensie 5. We beschouwen de inbedding van W$(7, 2)$ in $H(7, 4)$ en construeren een reguliere spread van W$(7, 2)$. Daarna bedden we deze spread in in $H(7, 4)$ en controleren we of deze maximaal is.

We vertrekken van de inbedding van P_1=PG$(7, 2)$ in P_2=PG$(7, 4)$, waarna we ook nu weer de punten van P_1 als nummers t.o.v. P_2 beschouwen. We controleren of al deze punten op de hermitische variëteit $H(7, 4)$ liggen. Daarna volgt de constructie van 1 generator van W$(7, 2)$ zoals voorheen. De generatoren zijn nu 3-dimensionele ruimtes.

```
n := 7;q := 2;
P1 := ProjectiveGeometry(n,q);
P2 := ProjectiveGeometry(n,q^2);
P3 := Embed(P1,P2);
subpts := ProjectivePoints(P3);;
subptsn := Set(PointToNumber(subpts));;
mat := [[0,1,0,0,0,0,0,0],[-1,0,0,0,0,0,0,0],[0,0,0,1,0,0,0,0],
[0,0,-1,0,0,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,-1,0,0,0],[0,0,0,0,0,0,0,1],
```

```
[0,0,0,0,0,0,-1,0]]*Z(q)^0;
h := HermitianVariety(P2,mat);
hpts := ProjectivePoints(h);;
hptsn := PointToNumber(hpts);;
Set(List(subpts,x->x in h));
G := stabilizerGroup(h);
H := AsPermutationGroup(G);
phi := Polarity(P1,0,mat);
pts := ProjectivePoints(P1);;
p1 := pts[1];
h1 := p1^phi;
choices := Difference(ProjectivePoints(h1),[p1]);;
p2 := choices[1];
h2 := p2^phi;
choices := Intersection(ProjectivePoints(h1),ProjectivePoints(h2));;
choices := Difference(choices,ProjectivePoints(SubspaceOfPG([p1,p2])));;
p3 := choices[1];
h3 := p3^phi;
choices := Intersection(ProjectivePoints(h1),ProjectivePoints(h2),
ProjectivePoints(h3));;
choices := Difference(choices,ProjectivePoints(SubspaceOfPG([p1,p2,p3])));;
p4 := choices[1];
gen := SubspaceOfPG([p1,p2,p3,p4]);
egen := SubspaceOfPG(P3,gen);
gap> egenpts := ProjectivePoints(egen);;
gap> egenptsn := Set(PointToNumber(egenpts));;
```

`gen` is nu een generator van $\mathrm{W}(7,2)$. We kunnen nu weer controleren of deze generator absoluut is t.o.v. de polariteit die $\mathrm{W}(7,2)$ bepaalt.

```
gap>Set( List(egenptsn,x->x in hptsn));
[ true ]
gap> gen^phi=gen;
true
```

Doordat de generatoren nu 3-dimensionale ruimtes zijn bevatten ze $\frac{q^4-1}{q-1}$ punten, terwijl de punten van $\mathrm{W}(7,2)$ alle punten van $\mathrm{PG}(7,2)$ zijn. Dit zijn juist $\frac{q^8-1}{q-1}$ punten. De spread moet dan uit q^4+1 generatoren bestaan, dus voor $q=2$ zijn dit 17 generatoren.

Om een reguliere spread te construeren bepalen we de actie van de deelgroep van de stabilisatorgroep van $H(7,4)$, die de punten van $\mathrm{W}(7,2)$ stabiliseert op de generatoren van $\mathrm{W}(7,2)$. Omdat de spread grootte 17 moet hebben zoeken we in deze permutatiegroep een element van orde 17 en bepalen de banen ervan. Deze banen zijn weer kandidaten voor een spread. Hierin

zoeken we naar een baan van lengte 17, waarin alle generatoren onderling disjunct zijn. Deze baan zal dan een spread vormen van $\mathrm{W}(7,2)$.

```
gap> S := stabilizer(H,subptsn,OnSets);;
gap> wgens := Orbit(S,egenptsn,OnSets);;
gap> Sact := Action(S,wgens,OnSets);
<permutation group with 15 generators>
gap> o := 0;
0
gap> while o<>17 do
>   g := Random(Sact);
>   o := Order(g);
> od;
gap> groep:=Group([g]);
<permutation group with 1 generators>
gap> orbits:=Orbits(groep);;
gap> Set(List(orbits,x->Length(x)));
[ 17 ]
```

Net zoals in het vorige voorbeeld hebben alle banen de vereiste lengte. De enige bijkomende voorwaarde opdat een baan een spread zou zijn, is dat de unie van alle elementen van die baan juist alle punten van $\mathrm{W}(7,2)$ bevat.

```
gap> is_spread := function(tspread)
> local pts,p;
> pts := [];
>   for p in tspread do
>   pts := Union(pts,Intersection(wgens[p],subptsn));
> od;
> return Set(pts)=subptsn;
> end;
function( tspread ) ... end
gap> spread:=orbits[1];
[ 1, 1625, 57, 1724, 859, 506, 748, 432, 970, 719, 124, 2056, 1337, 1509,
  2005, 1282, 903 ]
gap> is_spread(spread);
false
gap> spread:=orbits[17];
[ 17, 1906, 1221, 720, 1559, 1832, 575, 2133, 1407, 223, 1375, 557, 811,
```

Baan nummer 17 is een spread.

In dimensie 5 wisten we dat deze spread een maximale partiële spread van de hermitische variëteit was. Nu moeten we enkel nog deze eigenschap nagaan. Een partiële spread is niet maximaal als er een generator kan gevonden worden die disjunct is met alle generatoren van die spread. We overlopen de generatoren uit de spread en elke generator van de hermitische variëteit die zo'n generator snijdt, wordt verwijderd uit de lijst met generatoren. Na elk element van de spread wordt afgeprint hoeveel generatoren er nog disjunct zijn met de al overlopen elementen van de spread. Als dit 0 is na de ganse spread te hebben doorlopen, weten we dat die spread maximaal is.

```
gap> hgens := GeneratorsAsSetsOfNumbers(h);;
gap> skew := ShallowCopy(hgens);;
gap> for i in spread do
>   w := wgens[i];
>   for s in [1..Length(skew)] do
>     if Length(Intersection(skew[s],w)) > 0  then
>       Unbind(skew[s]);
>     fi;
>   od;
>   skew := Set(skew);
>   Print(Length(skew),"\n");
> od;
65536
38080
22464
13408
8032
4896
3100
1979
1258
762
406
265
164
103
62
31
0
```

We hebben dus inderdaad een maximale partiële spread van $H(7,4)$ gevonden.

Hoofdstuk 4

Grassmann coördinaten en toepassingen

We bespreken operaties en methoden om Grassmann coördinaten van deelruimten van een projectieve ruimte te berekenen. Als toepassing bepalen we een graaf geassocieerd aan de Split-Cayley hexagon $H(q)$, om daarin te zoeken naar afstands 2-spreads van $H(q)$.

4.1 Grassmann coördinaten

Onderstel dat π_r een r-dimensionale deelruimte is van $\mathrm{PG}(n,q)$, $n \geq 3, 1 \leq r \leq n-2$, en onderstel dat $p_0(\bar{x}_0), p_1(\bar{x}_1), \ldots, p_r(\bar{x}_r)$, met $\bar{x}_i = (x_0^i, x_1^i, \ldots, x_n^i)$, $r+1$ lineair onafhankelijke punten van π_r zijn. Stel

$$T_x = \begin{pmatrix} x_0^0 & x_1^0 & \ldots & x_n^0 \\ x_0^1 & x_1^1 & \ldots & x_n^1 \\ \vdots & \vdots & & \vdots \\ x_0^r & x_1^r & \ldots & x_n^r \end{pmatrix}.$$

Verder schrijven we $(i_0 i_1 \ldots i_r)$ voor de determinant van de orde $r+1$ waarvan de kolommen samenvallen met de (i_0+1)-de, (i_1+1)-de, …, (i_r+1)-de kolom van de matrix T_x, met $i_0, i_1, \ldots, i_r \in \{0, 1, \ldots, n\}$. Indien twee van de i_j's verwisseld worden, wordt $(i_0 i_1 \ldots i_r)$ met -1 vermenigvuldigd, want dit komt overeen met het verwisselen van twee kolommen in de determinant. Zijn twee van de i_j's gelijk, dan geldt $(i_0 i_1 \ldots i_r) = 0$. Ten minste één van de determinanten $(i_0 i_1 \ldots i_r)$ is verschillend van nul, anders zou dit in strijd zijn met het lineair onafhankelijk zijn van de punten $p_0, p_1, \ldots, p_r$.
Kies $\binom{n+1}{r+1}$ geordende $(r+1)$-tallen $(i_0 i_1 \ldots i_r)$ zodanig dat $i_0, i_1, \ldots, i_r$ verschillende elementen van $\{0, 1, \ldots, n\}$ zijn en zodanig dat de verzamelingen $\{i_0 i_1 \ldots i_r\}$ juist alle deelverzamelingen met $r+1$ elementen van $\{0, 1, \ldots, n\}$ zijn. Orden de verzameling $\mathcal{V}$ van deze $(r+1)$-tallen. Een

coördinaatvector van π_r is bij definitie de vector

$$\bar{l} = (l_0, l_1, l_2, \ldots, l_{c(n+1,r+1)-1})$$

waarbij $\binom{n+1}{r+1} = c(n+1, r+1)$ en de l_j's de elementen $(i_0 i_1 \ldots i_r) \in \mathcal{V}$ in de gegeven volgorde zijn.
Het geordende stel $(l_0, l_1, l_2, \ldots, l_{c(n+1,r+1)-1})$ noemt men een stel **coördinaten** van de r-dimensionale deelruimte π_r van PG(n, q). De coördinaat vector $\bar{l}$ is door de ruimte π_r op een factor $\lambda \in GF(q) - \{0\}$ na bepaald. Voor $n = 3$ en $r = 1$ noemt men deze coördinaten de **Plücker coördinaten** van de rechte π_1, in alle andere gevallen worden deze coördinaten de **Grassmann coördinaten** van π_r genoemd.

Deze definitie is nu gemakkelijk om te zetten in een methode om de **Grassmann coördinaten** van een deelruimte π_r in **GAP** te berekenen. Zoals gezien moeten we vertrekken van $r+1$ punten die de deelruimte opspannen. Als de deelruimte voorgesteld wordt als span van punten, dan zijn die punten bevat in een component van die deelruimte en kunnen ze eenvoudig opgevraagd worden. Een deelruimte die niet de span is van punten moet via `AsSpanningPoints` aangepaste worden om ervoor te zorgen dat dit wel zo is.
In hoofdstuk 1 hebben we reeds opgemerkt dat de lijst van punten die de deelruimte opspannen kan gezien worden als matrix, waarbij het i^{de} element in de lijst overeenkomt met de i^{de} rij van de matrix of dus de coördinaten van het i^{de} punt. Er moeten nu van verschillende combinaties van $r + 1$ kolommen een determinant berekend worden. Als we de matrix transponeren komen die kolommen overeen met rijen en dus is de i^{de} kolom dan het i^{de} element in de lijst. Hierdoor moeten we voor de combinatie $(i_0 i_1 \ldots i_r)$, slechts de $(i_0 + 1)$-de, $(i_1 + 1)$-de, ..., $(i_r + 1)$-de elementen uit de lijst halen om daar dan de determinant van te berekenen.
Het enige wat ons dan nog rest is het berekenen van de $\binom{n+1}{r+1}$ $(r + 1)$-tallen $(i_0 i_1 \ldots i_r)$ zodanig dat $i_0, i_1, \ldots, i_r$ verschillende elementen van $\{0, 1, \ldots, n\}$ zijn. We weten dat $(i_0 i_1 \ldots i_i i_j \ldots i_r) = -(i_0 i_1 \ldots i_j i_i \ldots i_r)$, zodat de volgorde van de elementen $i_0, i_1, \ldots, i_r$ bepalend is voor het teken van de coördinaten. Een logische keuze is om met geordende $(r+1)$-tallen te werken. De ordening van de verzameling $\mathcal{V}$ zegt iets over de volgorde van de coördinaten. Hier is het dus ook zinvol om de ordening vast te leggen. Dit doen we als volgt: $(i_0 i_1 \ldots i_r)$ komt voor $(j_0 j_1 \ldots j_r)$ als en slechts als

$$(i_0 < j_0) \vee (i_0 = j_0 \wedge i_1 < j_1) \vee \ldots \vee (i_0 = j_0 \wedge \ldots \wedge i_{r-1} = j_{r-1} \wedge i_r < j_r).$$

Hiervoor gebruiken we:

`Combinations(`*mset [,k]*`) F`

geeft alle geordende combinaties van k elementen uit de lijst *mset*. Als k niet gegeven is worden alle combinaties van *mset* weergegeven. Deze functie hanteert dezelfde volgorde als hierboven vastgelegd, zodat we die kunnen verduidelijken met een voorbeeldje.

```
gap> Combinations([1..5],3);
[ [ 1, 2, 3 ], [ 1, 2, 4 ], [ 1, 2, 5 ], [ 1, 3, 4 ], [ 1, 3, 5 ],
  [ 1, 4, 5 ], [ 2, 3, 4 ], [ 2, 3, 5 ], [ 2, 4, 5 ], [ 3, 4, 5 ] ]
```

Dit alles geeft ons de volgende declaratie en methode.

```
DeclareAttribute("Grassmann", IsSubspaceOfPG);

InstallMethod(Grassmann,
    "for a subspace of a PG",
    true,
    [IsSubspaceOfPG],
    0,
    function(S)
    local F,P,n,nr,r,M,N,comb,combn,i,j,l,det;
    F:=FamilyObj(S);
    P:=F!.PG;
    n:=F!.n;
    if IsSubspaceIntersRep(S) then
     S:=AsSpanningPoints(S);
    fi;
    r:=Dimension(S);
    M:=ShallowCopy(PointToVector(S!.list));
    M:=MutableTransposedMat(M);
    combn:=Combinations([1..n+1],r+1);
    nr:=NrCombinations([1..n+1],r+1);
    l:=[];
    for i in [1..nr] do
     comb:=combn[i];
     N:=[];
     for j in [1..r+1] do
        Add(N,M[comb[j]]);
     od;
     det:=Determinant(N);
     Add(l,det);
    od;
return l;
end);
```

Het resultaat van voorgaande methode is een vector. Dit kan ook gezien worden als de coördinaatvector van een punt in de projectieve ruimte $\mathrm{PG}(N, q)$, met $N = \binom{n+1}{r+1} - 1$. In sommige omstandigheden kan het nuttig zijn de **Grassmann coördinaten** als punten te beschouwen en daartoe schrijven we nog de volgende methode.

```
DeclareAttribute("GrassmannAsPoint", IsSubspaceOfPG);

InstallMethod(GrassmannAsPoint,
```

```
    "for a subspace of a PG",
    true,
    [IsSubspaceOfPG],
    0,
    function(S)
    local P,F,q,n,r,d;
    F:=FamilyObj(S);
    q:=F!.q;
    n:=F!.n;
    r:=Dimension(S);
    d:=NrCombinations([1..n+1],r+1);
    P:=ProjectiveGeometry(d-1,q);
return ProjectivePoint(P,Grassmann(S));
end);
```

Stel dat L een rechte is in $\mathrm{PG}(3,q)$ en stel dat $y(y_0,y_1,y_2,y_3)$ en $z(z_0,z_1,z_2,z_3)$ verschillende punten van L zijn. We voeren een nieuwe notatie in, waarbij p_{ij} staat voor wat we hierboven steeds genoteerd hebben als (ij)

$$p_{ij} = y_i z_j - y_j z_i = \begin{vmatrix} y_i & y_j \\ z_i & z_j \end{vmatrix}.$$

Het geordende zestal $(p_{01},p_{02},p_{03},p_{23},p_{31},p_{12})$ noemt men een stel Plücker coördinaten van L. Noem α de afbeelding van de rechtenverzameling van $\mathrm{PG}(3,q)$ op de puntenverzameling van $\mathrm{PG}(5,q)$ gedefinieerd door

$$\alpha : L \mapsto (p_{01},p_{02},p_{03},p_{23},p_{31},p_{12}).$$

Stel

$$\Phi = \Phi(P_{01},P_{02},P_{03},P_{23},P_{31},P_{12}) = P_{01}P_{23} + P_{02}P_{31} + P_{03}P_{12},$$

dan is $\mathcal{K} = V(\Phi)$ een niet-singuliere kwadriek van $\mathrm{PG}(5,q)$, die de **kwadriek van Klein** genoemd wordt. We hebben dan volgende stelling.

Stelling 4.1.1. *De afbeelding α is een bijectie van de rechtenverzameling van PG$(3,q)$ op de kwadriek van Klein.*

Bewijs. Voor een bewijs zie cursus meetkunde [7] □

We zullen enkel nagaan of de Plücker coördinaten van de rechten van $\mathrm{PG}(3,q)$ op de kwadriek van Klein liggen. In het volgende hoofdstuk zullen we methodes bespreken om alle rechten van een deelruimte te construeren. We vermelden hier enkel dat deze operatie `Subspaces` twee argumenten heeft, als eerste de projectieve ruimte en als tweede de dimensie van de gewenste deelruimten. Het resultaat van de methode `Grassmann` zoals we die hebben geïmplementeerd is

$(p_{01}, p_{02}, p_{03}, p_{12}, p_{13}, p_{23})$, wat niet gelijk is aan de definitie van de Plücker coördinaten, zodat we een kleine aanpassing moeten doen aan de vergelijking van de kwadriek van Klein een weinig moeten aanpassen. Deze wordt dan:

$$X_0X_5 - X_1X_4 + X_2X_3 = 0$$

De volgende functie `Klein` gaat met gegeven q na dat de rechten van $\mathrm{PG}(3, q)$ als beeld de punten van $\mathcal{K}$ zijn.

```
Klein:=function(q)
local P,rechten,x,g;
P:=ProjectiveGeometry(3,q);
rechten:=Subspaces(P,1);
for x in rechten do
   g:=Grassmann(x);
   if not g[1]*g[6]-g[2]*g[5]+g[3]*g[4]=0*Z(q) then
return false;
   fi;
od;
return true;
end;
```

We geven nu enkele voorbeelden

Voorbeelden 7.

```
gap> P:=ProjectiveGeometry(5,5);
PG( 5, 5 )
gap> pts:=ProjectivePoints(P);;
gap> S:=SubspaceOfPG([pts[1],pts[3],pts[56]]);
Subspace of PG( 5, 5 )
gap> Grassmann(S);
[ Z(5)^2, 0*Z(5), 0*Z(5), 0*Z(5), Z(5), Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5),
  0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5),
  0*Z(5) ]
gap> GrassmannAsPoint(S);
[ Z(5)^0, 0*Z(5), 0*Z(5), 0*Z(5), Z(5)^3, Z(5)^3, 0*Z(5), 0*Z(5), 0*Z(5),
  0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5),
  0*Z(5), 0*Z(5) ]
gap> Klein(2);
true
gap> Klein(9);
true
```

4.2 Spreads in H(q)

We beschouwen de volgende contructie van de veralgemeende zeshoek $H(q)$. In het volgende hoofdstuk zullen we expliciet nagaan of deze constructie een veralgemeende zeshoek is.

We beschouwen in PG$(6, q)$ de parabolische kwadriek met vergelijking

$$X_0X_4 + X_1X_5 + X_2X_6 = X_3^2.$$

De punten van H(q) zijn alle punten van $Q(6, q)$. De rechten van de hexagon zijn juist de rechten gelegen op $Q(6, q)$ waarvan de Grassmann coördinaten voldoen aan de volgende zes lineaire vergelijkingen

$$\begin{array}{lll} p_{01} = p_{36} & p_{12} = p_{34} & p_{20} = p_{56} \\ p_{03} = p_{56} & p_{13} = p_{64} & p_{23} = p_{45} \end{array}$$

De incidentie is de natuurlijke incidentie van PG$(6, q)$.
Een stel Grassmann coördinaten van een rechte in PG$(6, q)$ is een geordend 21-tal ($\binom{7}{2} = 21$), nl.

$$(p_{01}, p_{02}, p_{03}, p_{04}, p_{05}, p_{06}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{23}, p_{24}, p_{25}, p_{26}, p_{34}, p_{35}, p_{36}, p_{45}, p_{46}, p_{56})$$

en kan dus beschouwd worden als een punt van PG$(20, q)$.

Alle rechten berekenen gebeurt analoog aan de berekening van de generatoren van een kwadriek of hermitische variëteit. We merken op dat de rechten van $Q(6, q)$ geen generatoren zijn, want de generatoren hebben dimensie $\frac{6+1-3}{2} = 2$ en zijn dus vlakken. Eerst construeren we een rechte op $Q(6, q)$ door de deelruimte te beschouwen door een punt p en een punt p' in de doorsnede $T_p(Q) \cap Q \backslash p$. Omdat de stabilisatorgroep van een variëteit transitief werkt op de deelruimten van die variëteit, bevat de baan van een rechte onder die stabilisatorgroep alle rechten van $Q(6, q)$. We bepalen opnieuw de actie van de stabilisatorgroep van $Q(6, q)$ op de punten en we stellen de rechten voor als een verzameling nummers.

```
Rechten:=function(Q)
local p,pts,dummy,h,q,L,l,lines,Lines,G,P;
pts:=ProjectivePoints(Q);
p:=pts[1];
h:=TangentHyperplane(Q,p);
dummy:=Difference(Intersection(ProjectivePoints(h),pts),[p]);
q:=dummy[1];
L:=SubspaceOfPG([p,q]);
L:=Set(PointToNumber(ProjectivePoints(L)));
G:=AsPermutationGroup(StabilizerGroup(Q));
lines:=Orbit(G,L,OnSets);
return lines;
end;
```

De rechten worden nu weergegeven als een lijst met nummers. Bij verdere bewerkingen zullen we zowel operaties tegenkomen die alleen op deelruimten toepasbaar zijn, als operaties die veel sneller verlopen indien ze toegepast worden op nummers. We moeten er nu voor zorgen dat de tijdswinst, die we gewonnen hebben door de efficiëntste methode te gebruiken niet verloren gaat door nodeloze omzettingen van deelruimten naar nummers en vice versa.

We beschouwen de Grassmann coördinaten van de rechten als punten van PG$(20, q)$ en beschouwen die rechten L waarvan het beeld L^α in de deelruimte S van PG$(20, q)$ gelegen is, waarbij S gegeven wordt door volgend stelsel vergelijkingen die rechtstreeks volgen uit de vergelijkingen voor de Grassmann coördinaten in de definitie H(q).

$$\begin{cases} X_0 - X_{17} = 0 \\ X_6 - X_{15} = 0 \\ X_1 + X_{16} = 0 \\ X_2 - X_{20} = 0 \\ X_7 + X_{19} = 0 \\ X_{11} - X_{18} = 0 \end{cases}$$

Bovenstaande laat ons nu toe H(q) te construeren.

```
P:=ProjectiveGeometry(6,q);
x:=Indets(P);
F:=x[1]*x[5]+x[2]*x[6]+x[3]*x[7]-x[4]^2;
Q:=Quadric(P,F);
rechten:=Rechten(Q);;
vec:=[];
for i in [1..21] do
  vec[i]:=0*Z(q);
od;
h:=[];
for i in [1..6] do
 h[i]:=ShallowCopy(vec);
od;
h[1][1]:=Z(q);
h[1][18]:=-Z(q);
h[2][7]:=Z(q);
h[2][16]:=-Z(q);
h[3][2]:=Z(q);
h[3][17]:=Z(q);
h[4][21]:=Z(q);
h[4][3]:=-Z(q);
h[5][8]:=Z(q);
h[5][20]:=Z(q);
```

```
h[6][12]:=Z(q);
h[6][19]:=-Z(q);
R:=ProjectiveGeometry(20,q);
h1:=Hyperplane(R,h[1]);
h2:=Hyperplane(R,h[2]);
h3:=Hyperplane(R,h[3]);
h4:=Hyperplane(R,h[4]);
h5:=Hyperplane(R,h[5]);
h6:=Hyperplane(R,h[6]);
S:=SubspaceOfPG([h1,h2,h3,h4,h5,h6]);
lijst:=[];
lijstn:=[];
for i in [1..Length(rechten)] do
  x:=rechten[i];
  l:=SubspaceOfPG(NumberToPoint(P,x));
  if ProjectivePoint(R,Grassmann(l)) in S then
    Add(lijstn,i);
    Add(lijst,l);
  fi;
od;
```

We hebben twee lijsten aangelegd met rechten van H(q), een lijst (`lijst`) bevat de rechten als deelruimten en de andere (`lijstn`) de positie van de rechten in de lijst met rechten van $Q(6,q)$. Op deze manier correspondeert elke rechte met een nummer.

We beschouwen een graaf Γ geassociëerd aan H(q). De toppen van Γ zijn de rechten van H(q). Door gebruik te maken van de 2^{de} lijst hebben de toppen als naam hun positie in de lijst met rechten van $Q(6,q)$. Twee toppen zijn adjacent als hun overeenkomstige rechten snijden. De automorfismegroep van de graaf is de deelgroep van de stabilisatorgroep van $Q(6,q)$ die de verzameling rechten van H(q) stabiliseert. Doordat we een lijst hebben waarin elke rechte correspondeert met een getal en door de werking van `Action` kunnen we die deelgroep berekenen. Dit laat ons toe de functie `rel` op te stellen die de adjacenties definieert. Γ wordt dan als volgt geconstrueerd.

```
rel:=function(i,j)
if i=j then
  return false;
else
  return not IsEmpty(Intersection(rechten[i],rechten[j]));
fi;
end;

toppen:=lijstn;
```

```
H:=AsPermutationGroup(StabilizerGroup(Q));
act:=Action(H,rechten,OnSets);
G:=Stabilizer(act,Set(lijstn),OnSets);

Gamma:=Graph(G,toppen,OnPoints,rel,true);
```

Definitie 4.2.1.
Een **rechtenspread** van $Q(6,q)$ is een verzameling rechten die de puntenverzameling partitioneert.

Definitie 4.2.2.
Een **afstands-2-spread** van een hexagon is een verzameling van rechten, zodat elk punt op exact 1 rechte van die verzameling ligt.

Ons doel is om aan de hand van de graaf Γ een afstands-2-spread van H(q) te construeren. Merk op dat een afstands-2-spread van H(q) ook een rechtenspread van $Q(6,q)$ is. Het omgekeerde is niet waar, daar het niet noodzakelijk is dat alle rechten van die rechtenspread tot H(q) behoren. Een afstands-2-spread is een verzameling rechten die onderling scheef zijn. Scheve rechten corresponderen in Γ met 2 toppen die niet adjacent zijn. Het vinden van een afstands-2-spread komt dan overeen met het zoeken naar een maximale verzameling van toppen waarvan er geen 2 adjacent zijn. In **Graph** hebben we daarvoor het volgende commando:

`IndependentSet(`*Gamma*`)`

We kunnen voorgaande ook op een andere manier benaderen. Stel Γ^c het complement van Γ, d.i. 2 toppen zijn adjacent als ze in Γ niet adjacent zijn. Een afstands-2-spread van H(q) komt hierin overeen met een maximale verzameling toppen die alle adjacent zijn met elkaar. We moeten dus zoeken naar de maximale complete deelgraaf (=*kliek*).

`CompleteSubgraphs(`*Gamma, k, all*`)`

Deze functie geeft een verzameling complete deelgrafen van Gamma weer, waarbij een complete deelgraaf weergegeven wordt door zijn toppenverzameling. Als k niet negatief is dan hebben alle deelgrafen grootte k. $all = 0$ zorgt dat de functie maar 1 complete deelgraaf van maximale grootte (als k negatief is) of grootte k (als k niet negatief is) weergeeft.

Indien een afstands-2-spread van H(q) bestaat moet die q^4+q^2+1 rechten bevatten, want H(q) bevat $\frac{q^6-1}{q-1}$ punten en elke rechte bevat $q+1$ punten en $q^4+q^2+1 = \frac{q^6-1}{(q-1)(q+1)}$.

Stel $q = 2$. Indien een afstands-2-spread van H(2) bestaat, moet die $16+4+1 = 21$ rechten bevatten. Er bestaat echter geen afstands-2-spread van H(2). Zillen we ons in dit geval moeten beperken tot het zoeken naar een maximale partiële afstands-2-spread.
We construeren de graaf Γ en gaan enkele eigenschappen zoals de taille, de diameter, ... na.

```
gap> Gamma:=Graph(G,toppen,OnPoints,rel,true);
```

```
rec( isGraph := true, order := 63, group := <permutation group with
    8 generators>,
  schreierVector := [ -1, 5, 5, 5, 2, 2, 2, 4, 1, 2, 4, 4, 5, 1, 5, 4, 1, 7,
      2, 7, 1, 1, 1, 2, 6, 3, 8, 7, 1, 1, 7, 1, 6, 6, 8, 8, 8, 7, 1, 1, 1, 5,
      5, 3, 8, 4, 1, 1, 1, 4, 8, 4, 2, 1, 2, 7, 7, 8, 1, 1, 2, 3, 3 ],
  adjacencies := [ [ 2, 7, 8, 12, 14, 63 ] ], representatives := [ 1 ],
  names := [ 4, 9, 12, 13, 23, 24, 38, 43, 73, 79, 83, 92, 93, 97, 98, 113,
      115, 116, 118, 119, 122, 128, 129, 144, 148, 169, 189, 190, 193, 194,
      195, 204, 206, 207, 208, 213, 214, 215, 218, 229, 230, 236, 243, 246,
      251, 252, 255, 256, 257, 265, 266, 272, 273, 278, 281, 282, 289, 291,
      297, 298, 301, 302, 303 ] )
gap> OrderGraph(Gamma);
63
gap> Diameter(Gamma);
3
gap> Girth(Gamma);
3
gap> IsRegularGraph(Gamma);
true
gap> IsSimpleGraph(Gamma);
true
gap> GlobalParameters(Gamma);
[ [ 0, 0, 6 ], [ 1, 1, 4 ], [ 1, 1, 4 ], [ 3, 3, 0 ] ]
gap> indepen:=IndependentSet(Gamma);
[ 1, 3, 6, 9, 10, 11, 13, 16, 22, 25, 26, 28, 30, 34, 45, 50, 58, 59, 62 ]
gap> time;
0
gap> Length(indepen);
19
```

We zien dat **GAP** zeer vlug een onafhankelijke verzameling van Γ van grootte 19 vindt. Dit doet vermoeden dat, als men 20 scheve rechten vindt, men er automatisch ook 21 kan vinden. We gaan nog na wat de resultaten in het complement van Γ zijn.

```
gap> cgamma:=ComplementGraph(Gamma);
rec( isGraph := true, order := 63, group := <permutation group with
    8 generators>,
  schreierVector := [ -1, 5, 5, 5, 2, 2, 2, 4, 1, 2, 4, 4, 5, 1, 5, 4, 1, 7,
      2, 7, 1, 1, 1, 2, 6, 3, 8, 7, 1, 1, 7, 1, 6, 6, 8, 8, 8, 7, 1, 1, 1, 5,
      5, 3, 8, 4, 1, 1, 1, 4, 8, 4, 2, 1, 2, 7, 7, 8, 1, 1, 2, 3, 3 ],
  adjacencies :=
    [ [ 3, 4, 5, 6, 9, 10, 11, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
          25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
```

```
          42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
          59, 60, 61, 62 ] ], representatives := [ 1 ],
  names := [ 4, 9, 12, 13, 23, 24, 38, 43, 73, 79, 83, 92, 93, 97, 98, 113,
      115, 116, 118, 119, 122, 128, 129, 144, 148, 169, 189, 190, 193, 194,
      195, 204, 206, 207, 208, 213, 214, 215, 218, 229, 230, 236, 243, 246,
      251, 252, 255, 256, 257, 265, 266, 272, 273, 278, 281, 282, 289, 291,
      297, 298, 301, 302, 303 ], isSimple := true )
gap> CompleteSubgraphs(cgamma,19,0);
[ [ 1, 4, 5, 9, 15, 18, 20, 22, 24, 26, 37, 39, 43, 46, 50, 54, 56, 57, 61 ] ]
gap> time;
31
gap> CompleteSubgraphsOfGivenSize(cgamma,19,0);
[ [ 1, 4, 5, 9, 15, 18, 20, 22, 24, 26, 37, 39, 43, 46, 50, 54, 56, 57, 61 ] ]
gap> time;
32
gap> CompleteSubgraphs(cgamma,20,0);
[  ]
```

Ook hier vinden we slechts 19 onafhankelijke toppen, wat ons vermoeden bevestigt. Het valt op dat de berekeningen in Γ^c veel langer duren dan in Γ, 31 milliseconden ten opzichte van 0 milliseconden.

Stel $q = 3$. Hier zou een afstands-2-spread lengte $81 + 9 + 1 = 91$ moeten hebben.

```
gap> Gamma:=Graph(G,toppen,OnPoints,rel,true);;
gap> OrderGraph(Gamma);
364
gap> Diameter(Gamma);
3
gap> Girth(Gamma);
3
gap> IsRegularGraph(Gamma);
true
gap> IsSimpleGraph(Gamma);
true
gap> GlobalParameters(Gamma);
[ [ 0, 0, 12 ], [ 1, 2, 9 ], [ 1, 2, 9 ], [ 4, 8, 0 ] ]
gap> indepen:=IndependentSet(Gamma);
[ 1, 5, 7, 8, 9, 10, 12, 13, 15, 17, 26, 29, 34, 40, 41, 45, 51, 59, 67, 72,
  80, 94, 102, 107, 108, 110, 113, 118, 119, 121, 123, 125, 128, 133, 134,
  135, 136, 140, 141, 144, 156, 157, 162, 176, 181, 183, 185, 188, 191, 200,
  201, 207, 215, 219, 220, 224, 226, 230, 232, 240, 241, 244, 256, 273, 281,
  287, 288, 289, 293, 295, 308, 314, 342, 345, 346, 353, 356, 360 ]
```

```
gap> time;
16
gap> Length(indepen);
78
```

In tegenstelling tot bij $q = 2$ zouden we hier wel een afstands-2-spread moeten vinden. De handleiding van **GRAPE** geeft al aan dat `IndependentSet` een "hopelijk" grote verzameling weergeeft. Hier is dit duidelijk niet het geval. Werken met het complement van $\Gamma_¡$ is ook niet aan te raden. Na een week rekenwerk waren er nog altijd geen resultaten. Hier zien we wat bij $q = 2$ al naar voor kwam. De bewerkingen in het complementair graaf zijn zeer tijdrovend. De rekentijd is te lang om er iets mee te kunnen doen.

Door geen beperkingen op te leggen aan de rechten van $Q(6, q)$ kunnen we analoog als hierboven een rechtenspread van $Q(6, 2)$ zoeken. Nu zal de graaf meer toppen bevatten, zodat het mogelijks ook hier lang zal duren om tot een resultaat te komen.

```
gap> P:=ProjectiveGeometry(6,q);
PG( 6, 2 )
gap> x:=Indets(P);
[ x_0, x_1, x_2, x_3, x_4, x_5, x_6 ]
gap> F:=x[1]*x[5]+x[2]*x[6]+x[3]*x[7]-x[4]^2;
x_0*x_4+x_1*x_5+x_2*x_6+x_3^2
gap> Q:=Quadric(P,F);
Quadric of PG( 6, 2 )
gap> rechten:=Rechten(Q);;
gap> toppen:=[1..Length(rechten)];
[ 1 .. 315 ]
gap> H:=AsPermutationGroup(StabilizerGroup(Q));
<permutation group of size 1451520 with 9 generators>
gap> act:=Action(H,rechten,OnSets);
<permutation group with 9 generators>
gap> G:=Stabilizer(act,toppen,OnSets);
<permutation group with 9 generators>
gap> Gamma:=Graph(G,toppen,OnPoints,rel,true);
rec( isGraph := true, order := 315, group := <permutation group with
    9 generators>,
  schreierVector := [ -1, 1, 5, 8, 5, 8, 7, 8, 2, 5, 7, 8, 9, 7, 8, 7, 8, 9,
      6, 3, 4, 7, 8, 9, 7, 8, 9, 7, 8, 9, 4, 5, 6, 8, 9, 2, 3, 4, 5, 7, 2, 3,
      4, 5, 7, 8, 9, 6, 7, 8, 9, 4, 5, 6, 8, 9, 2, 3, 7, 9, 2, 3, 4, 5, 7, 8,
      9, 7, 9, 7, 9, 4, 5, 6, 8, 9, 2, 3, 4, 5, 2, 3, 4, 5, 8, 9, 4, 5, 6, 8,
      9, 2, 3, 4, 5, 9, 2, 3, 4, 5, 6, 8, 9, 2, 3, 9, 2, 3, 9, 5, 6, 7, 7, 7,
      8, 9, 7, 9, 7, 8, 9, 7, 8, 7, 8, 9, 7, 8, 9, 8, 9, 7, 8, 9, 7, 7, 4, 5,
      6, 8, 9, 2, 4, 5, 1, 2, 4, 5, 8, 9, 4, 5, 6, 8, 9, 2, 3, 4, 5, 9, 2, 3,
```

```
      4, 5, 8, 9, 5, 6, 7, 7, 7, 8, 9, 7, 9, 7, 8, 7, 8, 9, 2, 7, 8, 8, 9, 8,
      9, 4, 2, 3, 4, 5, 6, 1, 3, 4, 8, 9, 4, 5, 9, 3, 4, 5, 9, 2, 3, 5, 8, 9,
      7, 9, 9, 3, 5, 8, 4, 5, 5, 8, 9, 4, 5, 8, 9, 7, 4, 4, 8, 9, 6, 7, 8, 9,
      9, 8, 8, 9, 7, 1, 5, 7, 8, 8, 6, 7, 8, 9, 9, 8, 8, 9, 5, 8, 8, 9, 2, 4,
      5, 9, 5, 7, 9, 5, 8, 9, 8, 9, 2, 5, 2, 6, 4, 4, 4, 5, 6, 1, 5, 9, 8, 9,
      7, 9, 4, 5, 4, 4, 5, 4, 5, 7, 9, 8, 9, 9, 8, 9, 8, 9, 8, 9, 8, 7, 9, 7,
      9, 7, 7, 8, 9, 8, 8, 7, 4 ],
  adjacencies := [ [ 2, 3, 5, 7, 14, 19, 23, 32, 33, 48, 77, 78, 79, 88, 89,
          110, 111, 113, 115, 138, 139, 141, 149, 179, 184, 187, 197, 209,
          212, 214, 217, 229, 231, 241, 253, 280, 284, 296, 299, 301, 302,
          303 ] ], representatives := [ 1 ], names := [ 1 .. 315 ] )
gap> indepen:=IndependentSet(Gamma);
[ 1, 4, 18, 31, 49, 61, 90, 103, 118, 126, 128, 156, 162, 163, 172, 181, 198,
  221, 266 ]
gap> time;
15
gap> Length(indepen);
19
```

Ook nu krijgen we slechts een spread met 19 rechten als resultaat. Daar $Q(6,2)$ in tegenstelling tot H(2) wel een rechtenspread heeft, zouden we die ook graag vinden. Hiervoor zoeken we nog eens in Γ^c.

```
gap> cgamma:=ComplementGraph(Gamma);
rec( isGraph := true, order := 315, group := <permutation group with
    9 generators>,
  schreierVector := [ -1, 1, 5, 8, 5, 8, 7, 8, 2, 5, 7, 8, 9, 7, 8, 7, 8, 9,
      6, 3, 4, 7, 8, 9, 7, 8, 9, 7, 8, 9, 4, 5, 6, 8, 9, 2, 3, 4, 5, 7, 2, 3,
      4, 5, 7, 8, 9, 6, 7, 8, 9, 4, 5, 6, 8, 9, 2, 3, 7, 9, 2, 3, 4, 5, 7, 8,
      9, 7, 9, 7, 9, 4, 5, 6, 8, 9, 2, 3, 4, 5, 2, 3, 4, 5, 8, 9, 4, 5, 6, 8,
      9, 2, 3, 4, 5, 9, 2, 3, 4, 5, 6, 8, 9, 2, 3, 9, 2, 3, 9, 5, 6, 7, 7, 7,
      8, 9, 7, 9, 7, 8, 9, 7, 8, 7, 8, 9, 7, 8, 9, 8, 9, 7, 8, 9, 7, 7, 4, 5,
      6, 8, 9, 2, 4, 5, 1, 2, 4, 5, 8, 9, 4, 5, 6, 8, 9, 2, 3, 4, 5, 9, 2, 3,
      4, 5, 8, 9, 5, 6, 7, 7, 7, 8, 9, 7, 9, 7, 8, 7, 8, 9, 2, 7, 8, 8, 9, 8,
      9, 4, 2, 3, 4, 5, 6, 1, 3, 4, 8, 9, 4, 5, 9, 3, 4, 5, 9, 2, 3, 5, 8, 9,
      7, 9, 9, 3, 5, 8, 4, 5, 5, 8, 9, 4, 5, 8, 9, 7, 4, 4, 8, 9, 6, 7, 8, 9,
      9, 8, 8, 9, 7, 1, 5, 7, 8, 8, 6, 7, 8, 9, 9, 8, 8, 9, 5, 8, 8, 9, 2, 4,
      5, 9, 5, 7, 9, 5, 8, 9, 8, 9, 2, 5, 2, 6, 4, 4, 4, 5, 6, 1, 5, 9, 8, 9,
      7, 9, 4, 5, 4, 4, 5, 4, 5, 7, 9, 8, 9, 9, 8, 9, 8, 9, 8, 9, 8, 7, 9, 7,
      9, 7, 7, 8, 9, 8, 8, 7, 4 ],
  adjacencies := [ [ 4, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 20, 21, 22,
          24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42,
          43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
```

```
          61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 80,
          81, 82, 83, 84, 85, 86, 87, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
          100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 112, 114, 116,
          117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
          130, 131, 132, 133, 134, 135, 136, 137, 140, 142, 143, 144, 145,
          146, 147, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
          160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
          173, 174, 175, 176, 177, 178, 180, 181, 182, 183, 185, 186, 188,
          189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200, 201, 202,
          203, 204, 205, 206, 207, 208, 210, 211, 213, 215, 216, 218, 219,
          220, 221, 222, 223, 224, 225, 226, 227, 228, 230, 232, 233, 234,
          235, 236, 237, 238, 239, 240, 242, 243, 244, 245, 246, 247, 248,
          249, 250, 251, 252, 254, 255, 256, 257, 258, 259, 260, 261, 262,
          263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275,
          276, 277, 278, 279, 281, 282, 283, 285, 286, 287, 288, 289, 290,
          291, 292, 293, 294, 295, 297, 298, 300, 304, 305, 306, 307, 308,
          309, 310, 311, 312, 313, 314, 315 ] ], representatives := [ 1 ],
  names := [ 1 .. 315 ], isSimple := true )
gap> CompleteSubgraphs(cgamma,21,0);
[ [ 1, 8, 11, 29, 46, 57, 63, 69, 80, 117, 130, 153, 166, 170, 193, 207, 256,
      262, 269, 288, 298 ] ]
gap> time;
1045078
```

Het graaf Γ^c geeft na enige tijd wel de gevraagde spread weer. Door de lange rekentijden besluiten we verdere berekingen voor grotere q te staken.

Opmerking

Zoeken naar spreads in H(q) via de graaf is niet optimaal. Enerzijds hebben we de operatie `IndependentSet` waarvan de resultaten ver beneden de verwachtingen liggen. Zelfs in het laatste geval waarbij de spread via Γ^c effectief gevonden wordt, gaf deze operatie maar een partiële spread als resultaat weer. Het werken met Γ^c heeft dan wel een veel accurater resultaat. Het nadeel hier is dat de rekentijden zeer hoog oplopen, waardoor dit ook niet erg bruikbaar is voor de gebruiker.

Hoofdstuk 5

Quotiëntruimten

5.1 Deelruimten van dimensie k

In voorgaande hoofdstukken hebben we reeds gebruik gemaakt van de operatie die alle rechten van een bepaalde projectieve ruimte construeert. We zullen de implementatie van die operatie hier in zijn algemeenheid bespreken. Het is de bedoeling alle deelruimten van dimensie k van een projectieve ruimte $\mathrm{PG}(n, q)$ te construeren.
Is $k = 0$, dan zijn die deelruimten alle punten van $\mathrm{PG}(n, q)$, is $k = n - 1$ dan zijn die deelruimten de hypervlakken van de projectieve ruimte. In beide gevallen hebben we reeds een operatie `hyperplanes` die deze deelruimten construeert.
Het algemeen geval verloopt opniew zoals de generatoren van een kwadriek of hermitische variëteit. We construeren een deelruimte van dimensie k als volgt. We nemen een lijst waarin initiëel de eerste twee punten van $\mathrm{PG}(n, q)$ zitten en beschouwen de deelruimte S, opgespannen door die lijst. Hiervoor voegen we telkens een punt niet in S gelegen toe en beschouwen opniew de deelruimte opgespannen door die lijst totdat de deelruimte de gewenste dimensie heeft. De baan van die deelruimte onder $\mathrm{P\Gamma L}(n + 1, q)$ bevat dan alle deelruimten van dimensie k.

```
DeclareOperation("Subspaces", [IsPG, IsInt]);

InstallMethod(Subspaces,
"for a PG and a dimension",
true,
[IsPG,IsInt],
0,
function(P,k)
local pts,S,G,F,lijst,i,subspace,orbit;
F:=FamilyObj(P!.projpt);
pts:=ShallowCopy(ProjectivePoints(P));
if (k<0 or F!.n-1<k) then
```

```
  Error("The dimension of the subspace must be between 0 and n");
fi;
```

We doen nog een controle op de dimensie k, want het is slechts zinvol een deelruimte te construeren als $0 \leq k \leq n-1$.

```
if k=0 then
   return List(ProjectivePoints(P),x->SubspaceOfPG(x));
elif k=F!.n-1 then
   return List(Hyperplanes(P),x->SubspaceOfPG(x));
fi;
lijst:=[pts[1]];
i:=2;
repeat
   Add(lijst,pts[i]);
   S:=SubspaceOfPG(lijst);
   pts:=Difference(pts,ProjectivePoints(S));
until Dimension(S)=k;
S:=Set(PointToNumber(ProjectivePoints(S)));
G:=AsPermutationGroup(PGammaL(F!.n+1,F!.q));
orbit:=Orbit(G,S,OnSets);
subspace:=[];
for i in orbit do
   Add(subspace,SubspaceOfPG(NumberToPoint(P,i)));
od;
return subspace;
end);
```

Om nu de deelruimten van dimensie k te hebben van een deelruimte S van $\mathrm{PG}(n,q)$ moeten we het voorgaande een weinig aanpassen. We hebben `hyperplanes` gebruikt, maar deze operatie is enkel toepasbaar op projectieve ruimten en niet op zijn deelruimten. Het geval $k = n-1$ moeten we dus behandelen zoals het algemene geval.
We kunnen niet zomaar de baan van de deelruimten berekenen onder $\mathrm{P\Gamma L}(n+1,q)$, want deze bevat ook deelruimten die niet meer in S vervat zijn. We bepalen daartoe de deelgroep van $\mathrm{P\Gamma L}(n+1,q)$ die S invariant laat.

```
InstallOtherMethod(Subspaces,
"for a PG and a dimension",
true,
[IsSubspaceOfPG,IsInt],
0,
function(S,k)
```

```
local pts,s,n,G,r,H,P,F,lijst,i,subspace,orbit,dummy;
F:=FamilyObj(S);
P:=F!.PG;
r:=Dimension(S);
pts:=ShallowCopy(ProjectivePoints(S));
if (k<0 or r-1<k) then
  Error("The dimension of the subspace must be between 0 and n");
fi;
if k=0 then
   return List(pts,x->SubspaceOfPG(x));
fi;
dummy:=ShallowCopy(pts);
lijst:=[pts[1]];
i:=2;
repeat
   Add(lijst,pts[i]);
   s:=SubspaceOfPG(lijst);
       dummy:=Difference(dummy,ProjectivePoints(s));
until Dimension(s)=k;
s:=Set(PointToNumber(ProjectivePoints(s)));
G:=AsPermutationGroup(PGammaL(FamilyObj(P!.projpt)!.n+1,F!.q));
H:=Stabilizer(G,Set(PointToNumber(pts)),OnSets);
orbit:=Orbit(H,s,OnSets);
subspace:=[];
for i in orbit do
   Add(subspace,SubspaceOfPG(NumberToPoint(P,i)));
od;
return subspace;
end);
```

5.2 Deelruimten van dimensie k die een bepaalde deelruimte bevatten

Beschouw π_r een deelruimte van PG(n, q). We gaan op zoek naar alle deelruimten van dimensie k van PG(n, q) die π_r bevatten, met $k > r$. Dit kan het best gedaan worden via quotiëntruimten. Beschouw een deelruimte π_{n-r-1} die π_r niet bevat. Alle $k-$dimensionale deelruimten worden opgespannen door π_r en een $k - r - 1$-dimensionale deelruimte van π_{n-r-1}.

Dankzij `Subspaces` beschikken we over alle π_{n-r-1} van PG(n, q). Via van `Random` zoeken we dan naar een geschikte π_{n-r-1}, zodat $\pi_r \cap \pi_{n-r-1} = \emptyset$

```
InstallMethod(SubspacesThroughSubspace,
```

```
"for a subspace of a PG and a dimension",
true,
[IsPG,IsSubspaceOfPG,IsInt],
0,
function(P,S,k)
local F,n,r,i,list,S2,list2,subspace;
F:=FamilyObj(P!.projpt);
n:=F!.n;
r:=Dimension(S);
list:=Subspaces(P,n-r-1);
S2:=Random(list);
while not IsEmpty(Intersection(S2,S)) do
     S2:=Random(list);
od;
list2:=Subspaces(S2,k-r-1);
subspace:=[];
for i in list2 do
   Add(subspace,SubspaceOfPG(Concatenation(i!.list,S!.list)));
od;
return subspace;
end);
```

Dezelfde redenering is ook bruikbaar om deelruimten te bepalen die een gegeven deelruimte bevatten te bepalen, die in een andere deelruimte liggen.

```
InstallOtherMethod(SubspacesThroughSubspace,
"for a subspace of a PG and a dimension",
true,
[IsSubspaceOfPG,IsSubspaceOfPG,IsInt],
0,
function(P,S,k)
local F,n,r,i,list,S2,list2,subspace;
n:=Dimension(P);
r:=Dimension(S);
list:=Subspaces(P,n-r-1);
S2:=Random(list);
while not IsEmpty(Intersection(S2,S)) do
     S2:=Random(list);
od;
list2:=Subspaces(S2,k-r-1);
subspace:=[];
for i in list2 do
   Add(subspace,SubspaceOfPG(Concatenation(i!.list,S!.list)));
```

```
od;
return subspace;
end);
```

Stel dat we alle vlakken die een gegeven punt p bevatten willen bepalen. Een projectief punt behoort niet tot de categorie van de deelruimten en dus zal de voorgaande methode een punt niet aanvaarden als argument. Via `SubspaceOfPG`([p]) is het punt nu wel een deelruimte van dimensie 0, waarop we voorgaande methode kunnen toepassen.

```
InstallOtherMethod(SubspacesThroughSubspace,
"for a subspace of a PG and a dimension",
true,
[IsSubspaceOfPG,IsProjectivePoint,IsInt],
0,
function(P,p,k)
local S;
S:=SubspaceOfPG([p]);
return SubspacesThroughSubspace(P,S,k);
end);
```

Voorbeelden 8.

```
gap> P:=ProjectiveGeometry(5,3);
PG( 5, 3 )
gap> pts:=ProjectivePoints(P);;
gap> S:=SubspaceOfPG([pts[1],pts[2]]);
Subspace of PG( 5, 3 )
gap> SubspacesThroughSubspace(P,S,2);
[ Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ) ]
gap> time;
984
```

```
gap> P:=ProjectiveGeometry(3,2);
PG( 3, 2 )
gap> Subspaces(P,1);
[ Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ),
  Subspace of PG( 3, 2 ), Subspace of PG( 3, 2 ) ]
```

5.3 Grassmannvariëteit

Onderstel dat $\mathrm{PG}^{(r)}(n,q)$ de verzameling is van alle r-dimensionale deelruimten van $\mathrm{PG}(n,q)$, $1 \leqslant r \leqslant n-2$ en $n \geqslant 3$. Is

$$\bar{l} = (l_0, l_1, l_2, \ldots, l_{c(n+1,r+1)-1})$$

een coördinaatvector van $\pi_r \in \mathrm{PG}^{(r)}(n,q)$, dan is $p(\bar{l})$ een punt van de projectieve ruimte PG(N,q), met $N = \binom{n+1}{r+1} - 1$.
De afbeelding die π_r afbeeldt op $p(\bar{l})$ stellen wij voor door α. De puntenverzameling $\mathrm{PG}^{(r)}(n,q)^{\alpha}$ van $\mathrm{PG}(N,q)$ wordt de *Grassmanniaan* of de **Grassmann variëteit** van de r-dimensionale deelruimten van $\mathrm{PG}(n,q)$ genoemd. We noteren deze variëteit met $\mathcal{G}_{r,n,q}$ of $\Gamma_{n,r,q}$, waarbij dit laatste meer slaat op de punt-rechte meetkunde dan op de variëteit.

Definitie 5.3.1.
Gegeven een $(r+1)-$deelruimte π_{r+1} en een $(r-1)-$deelruimte π_{r-1}, waarbij $\pi_{r-1} \subset \pi_{r+1}$. Een **pencil** is de verzameling van alle r-deelruimten van $\mathrm{PG}(n,q)$ gelegen in π_{r+1} die π_{r-1} bevatten.

Stelling 5.3.2. *Het beeld van een pencil van $r-$deelruimten onder α is een rechte van $\Gamma_{n,r,q}$.*

Bewijs. Stel π_{r-1} een gegeven $(r-1)$-deelruimte van $\mathrm{PG}(n,q)$ en $\pi_{r+1} \supset \pi_{r-1}$ een gegeven $(r+1)-$deelruimte van $\mathrm{PG}(n,q)$. Stel l een rechte in π_{r+1} die met π_{r-1} geen enkel punt gemeen heeft. We noemen $p(\bar{x}_0), p(\bar{x}_1), \ldots, p(\bar{x}_{r-1})$ r lineair onafhankelijke punten van π_{r-1}.
Beschouw 2 verschillende elementen π_r^1, π_r^2 van de pencil. De doorsneden van π_r^1, π_r^2 met l zijn de respectievelijke punten $p(\bar{x_r}^1)$ en $p(\bar{x}_r^2)$. De Grassmann coördinaten van $\pi_r^i, i = 1, 2$ zijn volledig

bepaald door de matrix

$$T_x^i = \begin{pmatrix} \bar{x}_0 \\ \bar{x}_1 \\ \vdots \\ \bar{x}_{r-1} \\ \bar{x}_r^i \end{pmatrix}.$$

Voor elk ander element π_r van de pencil geldt dat de coördinaatvector van het punt $\pi_r \cap l$ van de volgende vorm is

$$t_1\bar{x}_r^1 + t_2\bar{x}_r^2, \quad t_1, t_2 \in GF(q).$$

Omgekeerd definieert elke vector van dit type een element π_r van de pencil. De coördinaten van π_r zijn bepaald door de matrix

$$T_x = \begin{pmatrix} \bar{x}_0 \\ \bar{x}_1 \\ \vdots \\ \bar{x}_{r-1} \\ t_1\bar{x}_r^1 + t_2\bar{x}_r^2 \end{pmatrix}.$$

Als

$$\bar{l}_i = (l_0^i, l_1^i, \ldots), \quad i = 1, 2,$$

de coördinaatvector is, gedefinieerd door T_x^i, dan is $t_1\bar{l}_1 + t_2\bar{l}_2$ de coördinaatvector gedefinieerd door de matrix T_x. Dus alle elementen van de pencil worden afgebeeld op de rechte $< \pi_r^{1^\alpha}, \pi_r^{2^\alpha} >$. □

We beschouwen PG(4, 2) en zijn rechtenverzameling. De rechten worden door α afgebeeld op punten van PG(9, 2). De Grassmann variëteit hiervan is dan $\Gamma_{4,1,2}$ in PG(9, 2). De pencils zijn hier juist de rechten gelegen in een vlak door een gegeven punt in dat vlak. Dit worden ook stralenwaaiers genoemd. Om de rechten van $\Gamma_{4,1,2}$ te kennen moeten we eerst alle stralenwaaiers van PG(4, 2) construeren. Hun beelden onder α zijn dan alle rechten van de Grassmannvariëteit en dan hebben we de punt-rechte meetkunde $\Gamma_{4,1,2} = (\mathcal{P}, \mathcal{L}, I)$.

Definitie 5.3.3.
Een verzameling van punten B is een *Blocking set* van $\mathcal{P}$, als elke rechte een punt van B bevat. Men zegt dan dat elke rechte geblokkeerd wordt.

$$B \subset \mathcal{P} \Leftrightarrow (\forall l \in \mathcal{L})(l \cap B \neq \emptyset)$$

B is een *minimale blocking set* als geldt:

$$\forall p \in B : B \backslash p \quad \text{is geen blocking set.}$$

We zoeken een minimale blocking set van $\Gamma_{4,1,2}$. Stel π een hypervlak van PG(9, 2), dan ligt elke rechte van PG(9, 2) ofwel in dat hypervlak, ofwel heeft het er een punt mee gemeen. $\mathcal{L}$ is een verzameling rechten van PG(9, 2). De doorsnede van $\Gamma_{4,1,2}$ met het hypervlak π zal van elke

rechte minimum 1 punt bevatten en dus altijd een blocking set opleveren. Het is een open vraag of deze blocking set de kleinste minimale blocking set is.

De constructie van $\Gamma_{4,1,2}$ gebeurt met voorgaande operatie die alle deelruimten van een gegeven dimensie van $\mathrm{PG}(4,2)$ construeert. Door in elk vlak alle rechten door een gegeven punt te construeren, kunnen we dankzij de operatie `GrassmannAsPoint` de rechten van $\Gamma_{4,1,2}$ berekenen. Daarna berekenen we voor elk hypervlak van $\mathrm{PG}(9,2)$ de lengte van de doorsnede met $\Gamma_{4,1,2}$.

```
gap> P:=ProjectiveGeometry(4,2);
PG( 4, 2 )
gap> rechten:=Subspaces(P,1);;
gap> vlakken:=Subspaces(P,2);;
gap> ptsG:=[];
[  ]
gap> rechtenG:=[];
[  ]
gap> for L in rechten do
>       Add(ptsG,GrassmannAsPoint(L));
> od;
gap> for r in vlakken do
>       pts:=Subspaces(r,0);
>       for p in pts do
>               lines:=SubspacesThroughSubspace(r,p,1);
> $int(lines[1])]);
>               Add(rechtenG,S);
>       od;
> od;
gap> F:=ProjectiveGeometry(9,2);
PG( 9, 2 )
gap> h:=Hyperplanes(F);;
gap> lijst:=[];
[  ]
gap> for i in h do
> pts:=Intersection(ProjectivePoints(h[3]),ptsG);;
> Add(lijst,Length(pts))
> ;
> od;
gap> Set(lijst);
[ 91 ]
```

We zien dat de doorsnede met elk hypervlak een blocking set van 91 punten oplevert, waarbij we weten dat $\Gamma_{4,1,2}$ maar 155 punten bevat. $\Gamma_{4,1,2}$ bevat 1085 rechten. Toen we met behulp van exhaustieve methoden probeerden aan te tonen dat de gevonden blocking sets de kleinste zijn,

bereikten we al snel de grenzen van de capaciteit van de gebruikte computers. Toch zou het interessant kunnen zijn om de meetkunde $\Gamma_{4,1,2}$ in **GAP** te construeren, en dan deze meetkunde eventueel met het pakket **GRAPE** voor te stellen als een graaf, zodat we dan met behulp van typische algoritmen voor zoektochten in grafen dit probleem kunnen aanpakken.

Hoofdstuk 6

Ordening op deelruimten

6.1 Probleemstelling

We hernemen de toepassing beschreven in hoofstuk 7, sectie 3 uit de scriptie [1]. Het is een toepassing op de Split-Cayley hexagon H(q), waarvan het begin analoog is aan de toepassing die we reeds besproken hebben. Nu zullen we een andere graaf associëren met H(q). De graaf Γ geassociëerd aan H(q) heeft nu punten en rechten als toppen. Twee toppen zullen adjacent zijn als en slechts als een top een punt is dat gelegen is op de rechte die correspondeert met de andere top.

In het besluit van de scriptie [1] werd opgemerkt dat we ondanks de mogelijkheden omtrent collineatiegroepen verplicht zijn de graaf Γ te construeren door de triviale groep te gebruiken. Dit omdat de toppen nummers zijn die zowel op punten als op rechten kunnen slaan en we enkel een permutatievoorstelling hebben van de collineatiegroep die werkt op de punten. Om er voor te zorgen dat die permutatievoorstelling ook betrekking heeft op deelruimten, werd geschreven dat het volstaat om een ordening te hebben op die deelruimten. Het idee is om dan als toppenverzameling een lijst met projectieve punten en deelruimten te gebruiken en als automorfismegroep een collineatiegroep die reeds gekend is. De ordening moet er dan voor zorgen dat de constructie van de graaf gebeurt door middel van de collineatiegroep, zodat de constructie in verdere bewerkingen met die graaf op een vlottere wijze kan gebeuren.

In een eerste paragraaf zullen we die ordening implementeren, waarna we in een volgende paragraaf de nieuwe mogelijkheden zullen testen. We gaan na in hoeverre die werkwijze een elegante oplossing biedt en of de opmerking terecht gesteld werd. Bijkomend onderzoeken we wat de implicaties zijn op de duur van de berekeningen.

6.2 Ordening

De operatie `Set` werkt in op een lijst en zal deze omzetten in een geordende lijst, waaruit de duplicaten verwijderd zijn. Opdat deze operatie zou kunnen inwerken op een lijst met objecten moet er een ordening bestaan op deze objecten, d.i. er moet een methode geïnstalleerd zijn voor de operatie `LT` die inwerkt op de objecten van de lijst in het argument. Indien de lijst enkel uit projectieve punten en hypervlakken bestaat is zo'n ordening al gegeven. Het kan nuttig zijn om een lijst met deelruimten van willekeurige dimensie te ordenen.
Neem een punt p en een hypervlak π dan stellen we steeds $p < \pi$. Deze redenering kunnen we nu doortrekken. Beschouw twee deelruimten π en π' van $\mathrm{PG}(n,q)$ met respectievelijke dimensie l en m, dan geldt $\pi < \pi'$ als en slechts als $l < m$. Stel nu dat π en π' twee deelruimten zijn van dimensie k, $0 \leqslant k \leqslant n-1$. Indien $k = 0$ of $k = n-1$, dan wordt de deelruimte omgezet naar een punt respectievelijk een hypervlak en worden de bestaande methodes toegepast. We hebben een ordening op punten van eender welke projectieve ruimte en via de Grassmann coördinaat wordt elke deelruimte van $\mathrm{PG}(n,q)$ omgezet in een punt van $\mathrm{PG}(N,q)$. Aldus definiëren we de ordening

$$\pi < \pi' \Leftrightarrow \pi^{\alpha} < \pi'^{\alpha}$$

Voor de implementatie van de ordening van deelruimten moeten we telkens twee methodes installeren, omdat hier de volgorde waarin de argumenten meegegeven worden van belang is.
Voor een punt en een deelruimte:

```
InstallOtherMethod(LT,
"for a projective point and a subspace from the same PG",
true,
[IsProjectivePoint and IsProjectivePointRep,
 IsSubspaceOfPG],
0,
function(x,y)
if Dimension(y)=0 then
return LT(x,ProjectivePoints(y)[1]);
else
return true;
fi;
end);


InstallOtherMethod(LT,
"for a subspace and a projective point from the same PG",
true,
[IsSubspaceOfPG,
 IsProjectivePoint and IsProjectivePointRep],
0,
```

```
function(x,y)
if Dimension(x)=0 then
return LT(ProjectivePoints(x)[1],y);
else
return false;
fi;
end);
```

Voor een hypervlak en een deelruimte:

```
InstallOtherMethod(LT,
"for a subspace and a hyperplane from the same PG",
true,
[IsSubspaceOfPG,IsHyperplane],
0,
function(x,y)
if Dimension(x)=FamilyObj(x)!.n-1 then
return LT(x,Hyperplanes(y)[1]);
else
return true;
fi;
end);
```

```
InstallOtherMethod(LT,
"for a hyperplane and a subspace from the same PG",
true,
[IsHyperplane,IsSubspaceOfPG],
0,
function(x,y)
if Dimension(y)=FamilyObj(y)!.n-1 then
return LT(Hyperplanes(x)[1],y);
else
return false;
fi;
end);
```

Voor twee deelruimten:

```
InstallOtherMethod(LT,
"for two subspaces from the same PG",
true,
[IsSubspaceOfPG,IsSubspaceOfPG],
```

```
0,
function(x,y)
if Dimension(x)=Dimension(y) then
if Dimension(x)=0 then
return LT(ProjectivePoints(x)[1],ProjectivePoints(y)[1]);
elif Dimension(x)=FamilyObj(x)!.n-1 then
return LT(Hyperplanes(x)[1],Hyperplanes(y)[1]);
else
return LT(GrassmannAsPoint(x),GrassmannAsPoint(y));
fi;
else
return Dimension(x)< Dimension(y);
fi;
end);
```

Voorbeelden 9.

```
gap> P:=ProjectiveGeometry(4,3);;
gap> P:=ProjectiveGeometry(5,3);;
gap> P:=ProjectiveGeometry(5,3);
PG( 5, 3 )
gap> pts:=ProjectivePoints(P);;
gap> h:=Hyperplanes(P);;
gap> S:=SubspaceOfPG([pts[1],pts[2],pts[5]]);
Subspace of PG( 5, 3 )
gap> S1:=SubspaceOfPG([pts[13],pts[2],pts[5]]);
Subspace of PG( 5, 3 )
gap> S2:=SubspaceOfPG([pts[13],pts[5]]);
Subspace of PG( 5, 3 )
gap> lijst:=[S,S1,S2,pts[10],h[10]];
[ Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  [ 0*Z(3), Z(3)^0, Z(3)^0, 0*Z(3), Z(3)^0, 0*Z(3) ],
  [ Z(3)^0, Z(3)^0, 0*Z(3), 0*Z(3), Z(3), 0*Z(3) ] ]
gap> Set(lijst);
[ [ 0*Z(3), Z(3)^0, Z(3)^0, 0*Z(3), Z(3)^0, 0*Z(3) ], Subspace of PG( 5, 3 ),
  Subspace of PG( 5, 3 ), Subspace of PG( 5, 3 ),
  [ Z(3)^0, Z(3)^0, 0*Z(3), 0*Z(3), Z(3), 0*Z(3) ] ]
```

6.3 Vergelijkende test

We bespreken eerst gedetailleerd de constructie van de graaf aan de hand van de collineatiegroep, waarna we de oude constructie nog eens kort toelichten om ten slotte beide te vergelijken.

Zoals reeds opgemerkt heeft de graaf Γ geassociëerd aan $\mathrm{H}(q)$ nu punten en rechten als toppen, waarbij twee toppen adjacent zijn als en slechts als een top een punt is dat gelegen is op de rechte die correspondeert met de andere top. De constructie van de veralgemeende zeshoek $\mathrm{H}(q)$ verloopt volledig analoog als het voorgaande voorbeeld.

We onderstellen dat $\mathrm{H}(q)$ reeds gekend is, waarbij `rechten` alle rechten van $Q(6,q)$ bevat en `lijst` en `lijstn` de rechten van $\mathrm{H}(q)$ eenmaal als deelruimte en eenmaal als nummer bevatten. De functie `rel` die de adjacenties definieert, werkt als volgt: twee toppen met zelfde dimensie zijn nooit adjacent, anders gaan we na of de top met kleinste dimensie gelegen is in de top met grootste dimensie. Het attribuut `Dimension` is niet gekend voor projectieve punten. Omdat de functie veel omslachtiger wordt zonder het gebruik van dat attribuut besluiten we dat het handiger is om een methode te installeren, zodat dit attribuut wel gekend is voor projectieve punten.

```
InstallOtherMethod(Dimension,
"for a projectivePoint",
true,
[IsProjectivePoint],
0,
function(x)
return 0;
end);
```

Het is nu logisch om er voor te zorgen dat de oproep `Dimension` ook werkt voor hypervlakken.

```
InstallOtherMethod(Dimension,
"for a hyperplane",
true,
[IsHyperplane],
0,
function(x)
return FamilyObj(x)!.n-1;
end);
```

De functie `rel` wordt hiermee:

```
rel:=function(i,j)
if Dimension(i)=Dimension(j) then
  return false;
elif Dimension(i)<Dimension(j) then
  return i in j ;
else return j in i;
```

```
fi;
end;
```

Voor de toppenverzameling hebben we:

```
pts:=ProjectivePoints(Q);;
toppen:=Union(pts,lijst);;
```

Door het gebruik van `Union` weten we dat de operatie `Set` is toegepast op de verzameling, waardoor de toppenverzameling geordend is. Nu rest nog het berekenen van de collineatiegroep. Het volstaat niet de stabilisatorgroep van $Q(6,q)$ te nemen, omdat niet alle rechten van $Q(6,q)$ behoren tot $H(q)$. Zo kan het beeld van een rechte van $\mathrm{H}(q)$ een rechte zijn die niet meer tot het hexagon behoort. We moeten dus de deelgroep bepalen die de rechten van $\mathrm{H}(q)$ stabiliseert. Hiervoor moeten we de collineatiegroep eerst omzetten in de permutatiegroep om daarna een deelgroep daarvan weer om te zetten in een collineatiegroep. Dit is al een eerste punt dat deze werkwijze minder gebruiksvriendelijk maakt. Het idee was immers om niet meer de omweg via de permutatiegroep te moeten maken. De constructie van de groep:

```
gap> G:=StabilizerGroup(Q);
<group of size 1451520 with 9 generators>
gap> H:=AsPermutationGroup(G);
<permutation group of size 1451520 with 9 generators>
gap> rechtenh:=rechten{lijstn};
[ [ 8, 12, 36 ], [ 8, 26, 61 ], [ 15, 52, 61 ], [ 9, 61, 102 ],
  [ 15, 19, 43 ], [ 9, 13, 37 ], [ 12, 57, 80 ], [ 3, 12, 93 ],
  [ 33, 50, 76 ], [ 19, 57, 77 ], [ 3, 6, 66 ], [ 36, 70, 122 ],
  [ 26, 49, 108 ], [ 36, 38, 50 ], [ 26, 47, 83 ], [ 2, 5, 65 ],
  [ 2, 19, 45 ], [ 1, 13, 126 ], [ 13, 57, 107 ], [ 65, 70, 119 ],
  [ 45, 48, 108 ], [ 96, 100, 124 ], [ 33, 73, 84 ], [ 40, 77, 86 ],
  [ 17, 40, 99 ], [ 14, 84, 105 ], [ 71, 86, 102 ], [ 73, 74, 80 ],
  [ 50, 87, 96 ], [ 39, 42, 102 ], [ 42, 80, 100 ], [ 76, 77, 83 ],
  [ 38, 99, 126 ], [ 47, 118, 126 ], [ 17, 49, 114 ], [ 1, 6, 55 ],
  [ 3, 5, 17 ], [ 47, 65, 100 ], [ 38, 39, 45 ], [ 5, 9, 33 ], [ 6, 15, 96 ],
  [ 52, 116, 119 ], [ 52, 74, 99 ], [ 14, 40, 124 ], [ 43, 114, 122 ],
  [ 37, 109, 122 ], [ 43, 84, 118 ], [ 37, 48, 124 ], [ 39, 66, 105 ],
  [ 48, 93, 116 ], [ 71, 93, 118 ], [ 66, 83, 109 ], [ 105, 107, 119 ],
  [ 49, 87, 107 ], [ 56, 74, 109 ], [ 42, 114, 127 ], [ 55, 73, 108 ],
  [ 55, 70, 86 ], [ 56, 71, 87 ], [ 76, 116, 127 ], [ 2, 7, 56 ],
  [ 1, 7, 127 ], [ 7, 8, 14 ] ]
gap> Hq:=Stabilizer(H,rechtenh,OnSetsSets);
<permutation group of size 12096 with 5 generators>
gap> gens:=GeneratorsOfGroup(Hq);;
```

```
gap> gensc:=[];
[ ]
gap> for g in gens do
>       Add(gensc,PermutationToCollineation(PGammaL(7,q),g));
> od;
gap> G:=Group(gensc);
#I  default 'IsGeneratorsOfMagmaWithInverses' method returns 'true' for
[ Collineation( PGammaL( 7, 2 ), 0,
    [ [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ] ),
  Collineation( PGammaL( 7, 2 ), 0,
    [ [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
      [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ] ),
  Collineation( PGammaL( 7, 2 ), 0,
    [ [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ] ),
  Collineation( PGammaL( 7, 2 ), 0,
    [ [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2) ],
      [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ] ] ),
  Collineation( PGammaL( 7, 2 ), 0,
    [ [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
```

```
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
      [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ] ) ]
<group with 5 generators>
```

Nu hebben we de groep die inwerkt op de punten en rechten van H(2). Na het laden van het pakket **GRAPE** kunnen we de graaf construeren.

```
gap> LoadPackage("grape");
#I  Package ‘GRAPE’: non-Unix architecture or binaries not compiled
#I  Package ‘GRAPE’: functions depending on nauty will not work

Loading  GRAPE 4.2  (GRaph Algorithms using PErmutation groups),
by L.H.Soicher@qmul.ac.uk.

true
gap> Gammaplus:=Graph(G,toppen,OnPoints,rel,true);
Error, no method found! For debugging hints type ?Recovery from NoMethodFound
Error, no 2nd choice method found for ‘ONE’ on 1 arguments called from
OneOp( elm ) called from
One( Representative( M ) ) called from
One( Source( hom ) ) called from
ImagesSource( hom ) called from
Action( G, vertexnames, act ) called from
...
Entering break read-eval-print loop ...
you can ’quit;’ to quit to outer loop, or
you can ’return;’ to continue
brk> elm;
0 ; [ [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
  [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
  [ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
  [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
  [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
  [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
  [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ]
```

We krijgen een foutmelding. We hebben een collineatie waarvan **GAP** het eenheidselement niet kan oproepen. Het is nog vereist om een methode te schrijven die, gegeven een collineatiegroep, het eenheidselement van die groep teruggeeft.

```
InstallOtherMethod(OneOp,
```

```
"for a collineation",
true,
[IsCollineation],
0,
function(x)
local F;
F:=FamilyObj(x);
return One(PGammaL(F!.n,F!.q));
end);
```

Nu kunnen we de graaf Γ construeren en nagaan hoelang dit duurt.

```
gap> Gammaplus:=Graph(G,toppen,OnPoints,rel,true);
rec( isGraph := true, order := 126, group := <permutation group with
    5 generators>,
  schreierVector := [ -1, 4, 3, 2, 4, 2, 2, 2, 2, 5, 5, 4, 4, 2, 5, 4, 4, 4,
      4, 2, 5, 4, 1, 3, 3, 5, 2, 3, 3, 5, 5, 5, 4, 2, 3, 5, 2, 4, 2, 2, 2, 5,
      5, 4, 4, 2, 5, 4, 4, 4, 5, 5, 5, 4, 4, 3, 3, 5, 2, 3, 3, 5, 5, -2, 2,
      2, 4, 3, 4, 5, 4, 2, 2, 3, 4, 4, 3, 5, 4, 4, 5, 1, 5, 3, 2, 5, 4, 4, 2,
      4, 5, 2, 5, 3, 2, 4, 2, 4, 2, 5, 2, 5, 4, 4, 4, 3, 4, 2, 4, 5, 5, 4, 4,
      3, 5, 3, 4, 5, 4, 3, 3, 3, 5, 5, 3, 4 ],
  adjacencies := [ [ 67, 79, 80 ], [ 2, 8, 9 ] ],
  representatives := [ 1, 64 ],
  names := [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ],
      [ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ],
```

```
[ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2) ],
[ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ],
[ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ],
[ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
[ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
[ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
[ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
[ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2) ],
[ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ],
[ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ],
[ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
[ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
[ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
[ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
[ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ],
[ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ],
[ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ],
[ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2) ],
[ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ],
[ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
[ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
[ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2) ],
[ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ],
[ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2) ],
[ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ],
[ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2) ],
[ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ],
[ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
[ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
[ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ],
[ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2) ],
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ],
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2) ],
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2) ],
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
```

```
      Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ),
      Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 ), Subspace of PG( 6, 2 )
     ] )
gap> time;
985
gap> OrderGraph(Gammaplus);
126
gap> Diameter(Gammaplus);
6
gap> Girth(Gammaplus);
12
gap> IsRegularGraph(Gammaplus);
true
gap> IsSimpleGraph(Gammaplus);
true
gap> GlobalParameters(Gammaplus);
[ [ 0, 0, 3 ], [ 1, 0, 2 ], [ 1, 0, 2 ], [ 1, 0, 2 ], [ 1, 0, 2 ],
  [ 1, 0, 2 ], [ 3, 0, 0 ] ]
```

Een tweede luik van dit voorbeeld was het vergelijken van de snelheid van beide manieren om tot de constructie van $\mathrm{H}(q)$ te komen. Daartoe herhalen we kort de constructie van vroeger en testen we de tijd voor $q = 2, 3$.

Bij de oude werkwijze werden de toppen genummerd. Dit heeft ook een invloed op de functie `rel`, want we kunnen nu niet de dimensie nagaan van een nummer. Deze functie vereist de kennis over het aantal toppen en het feit dat de eerste helft van de toppen de punten zijn en de rest de rechten. De constructie gebeurt met volgende commando's.

```
G:=Group(());
toppen:=[1..126];
k:=Length(toppen)/2;
rel:=function(i,j)
if (i in [1..63]) and (j in [1..63]) then
 return false;
elif (i in [64..126]) and (j in [64..126]) then
 return false;
elif (i in [1..63]) and (j in [64..126]) then
 return pts[i] in lijst[j-63];
elif (i in [64..126]) and (j in [1..63]) then
 return pts[j] in lijst[i-63];
fi;
end;
```

We krijgen de volgende graaf.

```
gap> Gamma:=Graph(G,toppen,OnPoints,rel,true);
```

```
rec( isGraph := true, order := 126, group := Group(()),
  schreierVector := [ -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13,
      -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, -24, -25, -26, -27,
      -28, -29, -30, -31, -32, -33, -34, -35, -36, -37, -38, -39, -40, -41,
      -42, -43, -44, -45, -46, -47, -48, -49, -50, -51, -52, -53, -54, -55,
      -56, -57, -58, -59, -60, -61, -62, -63, -64, -65, -66, -67, -68, -69,
      -70, -71, -72, -73, -74, -75, -76, -77, -78, -79, -80, -81, -82, -83,
      -84, -85, -86, -87, -88, -89, -90, -91, -92, -93, -94, -95, -96, -97,
      -98, -99, -100, -101, -102, -103, -104, -105, -106, -107, -108, -109,
      -110, -111, -112, -113, -114, -115, -116, -117, -118, -119, -120, -121,
      -122, -123, -124, -125, -126 ],
  adjacencies := [ [ 79, 100, 103 ], [ 81, 99, 125 ], [ 109, 115, 118 ],
      [ 71, 74, 100 ], [ 124, 125, 126 ], [ 88, 98, 100 ], [ 68, 73, 80 ],
      [ 66, 68, 104 ], [ 119, 123, 125 ], [ 93, 102, 112 ], [ 92, 117, 122 ],
      [ 68, 108, 110 ], [ 75, 83, 121 ], [ 85, 94, 101 ], [ 76, 84, 120 ],
      [ 79, 80, 124 ], [ 74, 99, 104 ], [ 79, 83, 101 ], [ 64, 70, 71 ],
      [ 64, 65, 126 ], [ 99, 120, 121 ], [ 90, 114, 122 ], [ 89, 112, 116 ],
      [ 67, 69, 103 ], [ 74, 112, 115 ], [ 118, 122, 124 ], [ 69, 81, 82 ],
      [ 72, 86, 103 ], [ 71, 113, 114 ], [ 80, 84, 102 ], [ 65, 66, 67 ],
      [ 70, 73, 82 ], [ 85, 92, 104 ], [ 89, 107, 126 ], [ 81, 96, 97 ],
      [ 84, 111, 113 ], [ 78, 95, 115 ], [ 91, 106, 118 ], [ 85, 107, 111 ],
      [ 73, 87, 95 ], [ 66, 105, 106 ], [ 78, 97, 101 ], [ 86, 91, 120 ],
      [ 83, 105, 116 ], [ 72, 77, 92 ], [ 77, 96, 102 ], [ 87, 90, 121 ],
      [ 69, 109, 111 ], [ 67, 90, 93 ], [ 82, 116, 117 ], [ 64, 75, 77 ],
      [ 65, 76, 78 ], [ 70, 91, 94 ], [ 98, 108, 119 ], [ 72, 95, 123 ],
      [ 88, 96, 106 ], [ 87, 88, 107 ], [ 86, 89, 110 ], [ 76, 98, 117 ],
      [ 105, 113, 123 ], [ 93, 94, 119 ], [ 97, 110, 114 ], [ 75, 108, 109 ],
      [ 19, 20, 51 ], [ 20, 31, 52 ], [ 8, 31, 41 ], [ 24, 31, 49 ],
      [ 7, 8, 12 ], [ 24, 27, 48 ], [ 19, 32, 53 ], [ 4, 19, 29 ],
      [ 28, 45, 55 ], [ 7, 32, 40 ], [ 4, 17, 25 ], [ 13, 51, 63 ],
      [ 15, 52, 59 ], [ 45, 46, 51 ], [ 37, 42, 52 ], [ 1, 16, 18 ],
      [ 7, 16, 30 ], [ 2, 27, 35 ], [ 27, 32, 50 ], [ 13, 18, 44 ],
      [ 15, 30, 36 ], [ 14, 33, 39 ], [ 28, 43, 58 ], [ 40, 47, 57 ],
      [ 6, 56, 57 ], [ 23, 34, 58 ], [ 22, 47, 49 ], [ 38, 43, 53 ],
      [ 11, 33, 45 ], [ 10, 49, 61 ], [ 14, 53, 61 ], [ 37, 40, 55 ],
      [ 35, 46, 56 ], [ 35, 42, 62 ], [ 6, 54, 59 ], [ 2, 17, 21 ],
      [ 1, 4, 6 ], [ 14, 18, 42 ], [ 10, 30, 46 ], [ 1, 24, 28 ],
      [ 8, 17, 33 ], [ 41, 44, 60 ], [ 38, 41, 56 ], [ 34, 39, 57 ],
      [ 12, 54, 63 ], [ 3, 48, 63 ], [ 12, 58, 62 ], [ 36, 39, 48 ],
      [ 10, 23, 25 ], [ 29, 36, 60 ], [ 22, 29, 62 ], [ 3, 25, 37 ],
      [ 23, 44, 50 ], [ 11, 50, 59 ], [ 3, 26, 38 ], [ 9, 54, 61 ],
      [ 15, 21, 43 ], [ 13, 21, 47 ], [ 11, 22, 26 ], [ 9, 55, 60 ],
      [ 5, 16, 26 ], [ 2, 5, 9 ], [ 5, 20, 34 ] ],
  representatives := [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
      17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
      35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
      53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
      71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
      89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
      105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
      119, 120, 121, 122, 123, 124, 125, 126 ],
  names := [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
      19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
      37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
```

```
      55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
      73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
      91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
      107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
      121, 122, 123, 124, 125, 126 ] )
gap> time;
2250
```

Voor $q = 3$ geven we niet meer de output van de graaf, we concentreren ons op de tijd. Wel merken we nog op dat de functie `rel` nu aangepast moet worden in de oude versie, daar er nu 728 toppen zijn en geen 126 meer. We noemen de constructie via de collineatiegroep `Gammaplus`, in de oude methode noemen we de graaf `Gamma`.

```
gap> Gammaplus:=Graph(G,toppen,OnPoints,rel,true);;
gap> time;
7437
gap> OrderGraph(Gammaplus);
728
gap> Diameter(Gammaplus);
6
gap> Girth(Gammaplus);
12
gap> IsRegularGraph(Gammaplus);
true
gap> IsSimpleGraph(Gammaplus);
true
gap> GlobalParameters(Gammaplus);
[ [ 0, 0, 4 ], [ 1, 0, 3 ], [ 1, 0, 3 ], [ 1, 0, 3 ], [ 1, 0, 3 ],
  [ 1, 0, 3 ], [ 4, 0, 0 ] ]
gap> Gamma:=Graph(G,toppen,OnPoints,rel,true);;
gap> time;
82234
```

Opmerking

We hebben voor $q = 2, 3$ enkele eigenschappen zoals de taille en de diameter van de graaf opgevraagd. De valentie is telkens $q + 1$, want elke rechte is adjacent met de $q + 1$ punten die erop liggen en analoog voor het duale. Uit de taille volgt een verklaring voor de naam hexagon. De kleinste cykel heeft lengte 12, waarbij we weten dat een cykel een opeenvolging is van punt-rechte-punt-rechte .... Een cykel van lengte 12 komt overeen met 6 rechten die een zeshoek vormen. Daar dit de kleinste cykel is, komen er geen $3-, 4-, 5-$hoeken voor in deze rang 2-meetkunde.

6.4 Bespreking

Tijdens het uitvoeren van bovenstaande vergelijkende test zijn bij beide methodes enkele voor- en nadelen naar boven gekomen. We zetten die hier eens op een rijtje.

- Zoals blijkt uit de inleiding over het pakket **GRAPE** is het doel van het meegeven van een groep als argument bij de constructie dat de CPU-tijd vermindert. Door gebruik te maken van de ordening van deelruimten zijn we niet meer verplicht de triviale groep als automorfismegroep te nemen. Door beide werkwijzen te gebruiken, hebben we een ideale basis om te testen of de rekentijd effectief vermindert en hoe groot het verschil dan wel is. We hebben de test enkel gedaan voor kleine q, maar toch is hier al duidelijk geworden dat het werken met de collineatiegroep sneller is. Voor $q = 3$ gaat het zelfs bijna 10 maal sneller, wat toch zeker niet verwaarloosbaar is.

- We hebben opgemerkt dat we de collineatiegroep niet zomaar konden berekenen, dit omdat de actie van $\mathrm{PGO}(7, q)$ op de verzameling objecten van $\mathrm{H}(q)$ niet gekend is. Dit zorgt ervoor dat de kennis van de ordening van deelruimten alleen niet volstaat om het probleem op een eenvoudige manier op te lossen. De berekening van de goede collineatiegroep is nog steeds niet vanzelfsprekend voor de gebruiker, waardoor die geneigd zal zijn om over te gaan op de trage manier.

- We hebben nog een operatie `OneOp` moeten toevoegen aan de methodes met betrekking tot collineatiegroepen. Dit toont aan dat de implementatie van collineatiegroepen en collineaties nog zeer onvolledig is. Nu dit bij het pakket zit, kunnen we de collineatiegroep gebruiken in de constructie van Γ. We kunnen niet verwachten van een doorsnee gebruiker dat die zich gaat verdiepen in methodes voor collineaties, vooraleer hij er iets mee kan doen.

- Niet alleen de nieuwe werkwijze heeft zijn nadelen. Door het gebruik van nummers als toppen is het noodzakelijk om bij elke nieuwe q een nieuwe functie `rel` te schrijven. Deze functie vereist ook de kennis over het aantal toppen. Voor deze functie moeten we ons ook meer bekommeren om interne structuren. Het is noodzakelijk te weten dat de eerste helft van de toppen punten zijn en de tweede helft rechten en we dienen te weten onder welke naam die objecten opgeslaan zijn. Dit terwijl in de functie de we geïntroduceerd hebben, we niets moeten weten over wat de toppen voorstellen en hoe ze opgeslaan zijn om na te gaan of ze al dan niet adjacent zijn.

We kunnen stellen dat de conclusie in [1] iets te voorbarig was. Er was nog enig werk vereist voor de constructie van de graaf zonder het gebruik van de triviale groep werkte. De nieuwe methodes die daarvoor nodig waren zijn aangemaakt, zodat de gebruiker nu wel eenvoudig berekeningen kan maken met het hexagon en aanverwanten. Door die extra methodes kunnen we besluiten dat het wel vlotter werken is met de collineatiegroep. Deze tijdswinst zal zich ook uiten in alle toepassingen die men doet met de graaf.

Hoofdstuk 7

Besluit

De uitbreidingen aan het pakket **PG** die we in deze scriptie beschreven hebben, kunnen we allemaal bestempelen als functionaliteit die onmiddellijk door de gebruiker aangewend kan worden om typische meetkundes en hun deelstructuren te onderzoeken. Door deze uitbreidingen van **PG** kan een theoretisch onderzoeker zonder programmeerervaring onmiddellijk aan de slag om zijn of haar favoriete meetkunde te onderzoeken. Onze voorbeelden tonen aan dat dit onmiddellijk tot resultaten kan leiden.

Het toevoegen van functionaliteit die te maken heeft met wat we in deze scriptie inbeddingen genoemd hebben, heeft doorheen de ganse scriptie een rol gespeeld. De voorbeelden die beschreven zijn tonen aan dat een theoretische onderzoeker zaken als "een projectief vlak $\mathrm{PG}(2, q)$ ingebed als *een* hypervlak in $\mathrm{PG}(3, q)$" letterlijk en met heel weinig commando's kan construeren in het pakket. Dit zou de drempel voor theoretische onderzoekers met weinig of geen praktische ervaring met computeralgebrapakketten moeten verlagen, omdat dit soort functionaliteit aansluit op hun denkwijze.

Bij het toevoegen van de verschillende mogelijkheden, werd rekening gehouden met de bestaande genericiteit van het pakket. Het was onder andere reeds mogelijk om met kwadrieken en hermitische variëteiten te werken. Ondanks het feit dat er in bijvoorbeeld $\mathrm{PG}(4, q)$, op een collineatie na, slechts één niet-singuliere kwadriek is, heeft de gebruiker naast de mogelijkheid om met een zogenaamde standaardvorm te werken, eveneens de mogelijkheid om een kwadriek te beschouwen bepaald door een zelf gekozen vergelijking. Deze filosofie hebben we gevolgd. Wanneer we een ruimte $\mathrm{PG}(m, q)$ inbedden als een deelruimte in $\mathrm{PG}(n, q)$, $m < n$, dan heeft de gebruiker de vrijheid om zelf deze deelruimte te bepalen. Dat deze genericiteit loont, wordt aangetoond in de voorbeelden uit hoofdstuk 3. De constructie van de inbedding van de symplectische ruimte in de hermitische ruimte kan immers op een zeer natuurlijke wijze gebeuren, indien de gebruiker zelf de symplectische en hermitische polariteit kan opgeven die de ruimtes bepalen. Deze voorbeelden maken meteen ook gebruik van functies die alle generatoren van een hermitische variëteit bepalen. Dankzij de onderliggende genericiteit was het daarenboven vrij eenvoudig om deze laatste functies te implementeren. Deze voorbeelden tonen nogmaals

het nut aan van resultaten die we met behulp van een computeralgebrasysteem bekomen. We hebben bijvoorbeeld aangetoond dat een spread van de symplectische ruimte $\mathrm{W}(7,2)$, ingebed in de hermitische variëteit $\mathrm{H}(7,4)$, inderdaad maximaal is in $\mathrm{H}(7,4)$. Dit resultaat is op zich niet wereldschokkend, maar het kan een aanwijzing geven aan de theoretische onderzoeker.

De zogenaamde Grassmanncoördinaten en de Grassmannvariëteiten zijn vanuit wiskundig oogpunt zeer interessant. We voegden de functies die Grassmanncoördinaten van r-dimensionale deelruimten van $\mathrm{PG}(n,q)$ bepalen toe, omdat we met behulp van Grassmanncoördinaten twee interessante punt-rechte meetkundes wilden construeren, namelijk de split-Cayley veralgemeende zeshoek $H(q)$ en de Grassmannvariëteit $\mathcal{G}_{1,4,2}$. Deze laatste noteert men ook als een punt-rechte meetkunde $\Gamma_{4,1,2}$. De constructie van de veralgemeende zeshoek $\mathrm{H}(q)$ gebruiken we om op zoek te gaan naar spreads. Hierbij stoten we dan op een gebrek van het pakket **PG**: het is in feite niet geschikt om exhaustieve zoektochten uit te voeren in grote meetkundes. Op dezelfde manier hebben we testen gedaan om te bewijzen dat de kleinste blocking set van $\Gamma_{4,1,2}$ lineair is. Ook hier vonden we het pakket **PG** ongeschikt. Ook het gebruik van het pakket **Grape** in combinatie met **PG** maakt het uitvoeren van exhaustieve zoektochten nog steeds niet mogelijk in een (grote) meetkunde zoals $\Gamma_{4,1,2}$.

We hebben een ordening op deelruimten specifiek geïmplementeerd om te gebruiken in combinatie met het pakket **Grape**. Deze uitbreiding zorgt ervoor dat de objecten uit het pakket **PG** opeens veel geschikter zijn als toppen voor bijvoorbeeld een incidentiegraaf die we dan in **grape** construeren. Tijdens het gebruik hebben we vastgesteld dat er bijkomend fundamenteel werk dient verricht te worden. Er moeten nog extra functies geïmplementeerd worden die betrekking hebben op de collineatiegroepen en de actie van collineatiegroepen op deelruimten.

Ten slotte ontbreken er nog steeds functies die bijvoorbeeld nagaan of een gegeven verzameling van meetkundige objecten aan een bepaalde eigenschap voldoet. We denken hierbij aan functies die controleren of een puntenverzameling een unitaal of ovaal is, een puntenverzameling een blocking set is, ...

Bibliografie

[1] Jan De Beule. *Implementatie van projectieve deelruimten en variëteiten in het softwarepakket GAP.* Scriptie, Universiteit Gent, 2000.

[2] Jan De Beule en Klaus Metsch. *The maximum size of a partial spread in $H(5, q^2)$ is $q^3 + 1$.* J.Combinatorial theory Ser. A, submitted.

[3] An De Wispelaere. *Ovoids and spreads of finite classical Generalize hexagons and applications.* Doctoraatsscriptie, Universiteit Gent, 2005.

[4] Patrick Govaerts. *Projectieve meetkunde en het softwarepakket GAP.* Scriptie, Universiteit Gent, 1999.

[5] The **GAP** Group. *The GAP4 Reference Manual.* GAP–Groups, Algorithms and Programming, Version 4.4; 2006. `http://www.gap-system.org`

[6] J.W.P. Hirschfeld en J.A.Thas. *General Galois Geometries.* Oxford Science publications, 1991.

[7] J.A.Thas. *Meetkunde.* Cursus licenties wiskunde, Universiteit Gent, 2001.

[8] J.A. Thas. *Two infinite classes of perfect codes in metrically regular graphs.* J. Combinatorial Theory Ser. B, 23(2-3):236–238, 1977.