SL-Erkennung für Matrizengruppen der Dimension 2

Sabina Groth

Software Praktikum
Sommersemester 2007

Inhaltsverzeichnis

1 Vorwort **2**

2 GAP-System **2**

3 Paket „recogbase“ **2**
3.1 Idee . . . 2
3.2 Konzept . . . 2

4 Motivation des Projekts **4**

5 Algorithmus **4**
5.1 Satz . . . 4
5.2 Verfahren . . . 5

6 Methoden **6**
6.1 Neu definierte Methoden . . . 7
6.1.1 RECOG.TestRandomElementCase2 (*recognise*, *grp*) . . . 7
6.1.2 RECOG.IsAbelian (*recognise*, *grp*) . . . 7
6.1.3 RECOG.HasExp2 (*recognise*, *grp*) . . . 8
6.1.4 RECOG.IsSubgroupOfGammaL (*recognise*, *grp*) . . . 9
6.1.5 RECOG.IsImprimitive (*recognise*, *grp*) . . . 10
6.1.6 RECOG.IsAlt5Alt4Sym4 (*recognise*, *grp*) . . . 11
6.1.7 RECOG.IsSL2Contained (*recognise*, *grp*) . . . 12
6.2 Vorhandene Methoden . . . 13
6.2.1 RECOG.IsReducible (*recognise*, *grp*) . . . 14
6.2.2 RECOG.MeatAxe (*recognise*, *grp*) . . . 14

7 Beispiele **15**
7.1 $\mathrm{SL}(2,\mathrm{q}) \nleq \mathrm{G}$ . . . 15
7.2 $\mathrm{SL}(2,\mathrm{q}) \leq \mathrm{G}$ . . . 17

8 Bibliografie **18**

1 Vorwort

Der vorliegende Text dokumentiert im Rahmen des Softwarepraktikums II an der Universität Bayreuth die Implementierung eines SL-Erkunnungsalgorithmus für Matrizengruppen von Dimension 2, der von Peter Neumann und Cheryl Praeger in [1] entwickelt worden ist. Kapitel 2 stellt das Softwarepaket GAP [3] kurz vor. Kapitel 3 bietet eine knappe Einführung in das Paket „recogbase“, welches von unserem Programm benutzt wird. Sodann werden die Motivation des Projekts (siehe Kapitel 4) und der zu Grunde liegender Algorithmus (siehe Kapitel 5) erklärt. Kapitel 6 beschreibt die neu definierten sowie bereits vorhandenen Methoden des Programms. Anwendungsbeispiele findet man in Kapitel 7.

2 GAP-System

GAP [3], das im Englischen für „**G**roups, **A**lgorithms, and **P**rogramming“ steht, ist ein Softwarepaket für Berechnungen im Gebiet der diskreten abstrakten Algebra. Das System ist „erweiterbar“, das heißt es ist möglich eigene Programme in der GAP-Sprache zu verfassen und diese genauso zu verwenden wie Programme, die Teil des Systems sind. Die Entwicklung von GAP begann im Jahre 1985 am Lehrstuhl D für Mathematik, RWTH-Aachen, ist jedoch mittlerweile ein internationales Projekt mit Hauptsitz in St Andrews.

Das System besteht aus einem Kernsystem und einer Anzahl von Paketen, die eine Erweiterung des Kerns darstellen. Ein Paket kann mittels des Aufrufs `LoadPackage` in das System geladen werden, woraufhin die Funktionen des Paketes sowie seine Dokumentation verfügbar werden. Das von uns genutzte Paket „recogbase“ ist das Grundgerüst unseres Programms und wird im nächsten Kapitel näher beschrieben.

3 Paket „recogbase“

Um das Paket zu installieren, ruft man `LoadPackage(‘‘recogbase’’);` auf. Der Befehl `LoadPackage(‘‘recog’’);` lädt neben dem Paket „recogbase“ ebenfalls die bereits implementierten Gruppenerkennungs-Methoden.

3.1 Idee

Das „recogbase“-Paket stellt einen Rahmen für Situationen dar, in denen für eine bestimmte Aufgabe zwar viele Methoden (eine Definition des Begriffs findet sich in Kapitel 3.2) zur Verfügung stehen, allerdings nicht von vornherein klar ist, welche dieser Methoden die „beste“ ist und deswegen verwendet werden sollte. Wir entkommen der Qual der Wahl, indem wir eine Reihenfolge festgelegen, in der die verschiedenen Methoden ausprobiert werden sollen. Die Methoden wiederum geben an, ob ihre Ausführung erfolgreich war, und falls dies nicht der Fall war, ob es Sinn macht die jeweilige Methode zu einem späteren Zeitpunkt nochmal aufzurufen.

3.2 Konzept

Zuerst erklären wir, was unter einer „Methode“ zu verstehen ist. Eine Methode ist eine GAP-Funktion, die einen der folgenden Werte zurückgeben muss:

true bedeutet, dass die Methode erfolgreich war und keine weiteren Methoden verwendet werden müssen,

false bedeutet, dass die Methode nicht erfolgreich war und in dieser Situation auch nicht erneut aufgerufen werden soll,

fail bedeutet, dass die Methode abgebrochen wird, es aber durchaus Sinn macht sie zu einem späterem Zeitpunkt nochmal aufzurufen,

NotApllicable bedeutet, dass die Methode zum Zeitpunkt des Aufrufs nicht anwendbar ist, jedoch später erneut aufgerufen werden sollte.

Die Ausgebewerte *fail* und *NotApplicable* mögen anfangs gleich erscheinen. Der Unterschied beider Werte wird zum Ende dieses Kapitels klar.

Eine Methode wird in der Methodenauswahl mittels eines Rekords[1] beschrieben. Dieser beinhaltet folgende Komponenten:

Methode: Name der Funktion;

Rang: Eine natürliche Zahl, die zur Sortierung unterschiedlicher Methoden dient. Je höher die Zahl, umso früher wird die jeweilige Methode aufgerufen;

Stempel: Ein String, der die Methode eindeutig beschreibt;

Kommentar: Ein String, der als Kommentar dient. Dieses Feld ist optional und kann auch weggelassen werden.

Methoden für eine bestimmte Aufgabe finden wir in so genannten Methoden-Datenbanken. Eine Methoden-Datenbank ist eine Liste von Rekords, wobei – wie bereits geschildert – jeder Rekord eine Methode beschreibt. Um Methoden zur Methoden-Datenbank hinzuzufügen, bedient man sich der Funktion `AddMethod( db, method, rank, stamp[, comment] )`. Dabei ist *db* eine Methoden-Datenbank (also eine Liste von Rekords), *method* eine Methode, *rank* ist der Rang der Methode und *stamp* ein String. Die optionale Position *commment* ist gegebenenfalls auch ein String.

Immer wenn die Methodenauswahl benutzt werden soll, wird die Funktion `CallMethods( db, limit[, furtherargs] )` aufgerufen. Die Funktion gibt sodann einen Rekord zurück, der das Verfahren der Methodenauswahl beschreibt. Das Argument *db* muss eine Methoden-Datenbank sein, *limit* eine natürliche Zahl, und *furtherargs* steht für eine beliebige Anzahl von weiteren Argumenten, die an die aufgerufenen Methoden weitergegeben werden.

Der von `CallMethods` verwendete Algorithmus, funktioniert wie folgt. Ein interner Toleranzzähler wird auf Null gesetzt. Die Hauptschleife beginnt am Anfang der Methoden-Datenbank und durchläuft nach der Reihe alle Methoden. Eine Methode wird allerdings nur dann aufgerufen, wenn sie zuvor nicht bereits *false* oder in einem Durchlauf mit derselben Toleranz *fail* zurückgegeben hat. Abhängig von dem Rückgabewert der aufgerufenen Methode passiert folgendes:

false: Die Hauptschleife beginnt wieder am Anfang der Methoden-Datenbank;

[1] engl. „record“. Ein Rekord ist eine Datenstruktur, die ähnlich eine Liste Objekte beinhaltet. Im Gegensatz zu einer Liste sind diese jedoch nicht mit Zahlen, sondern Namen indiziert.

fail: Die Hauptschleife beginnt wieder am Anfang der Methoden-Datenbank;

NotApplicable: Die Hauptschleife geht zur nächsten Methoden in der Methoden-Datenbank über;

true: Das Verfahren wird beendet.

Sobald die Hauptschleife das Ende der Methoden-Datenbank erreicht ohne dabei eine Methode aufgerufen zu haben, wird der Toleranzzähler um eins erhöht und alles beginnt von Neuem. Dieses Verfahren wird solange wiederholt bis die Toleranz größer ist als die im zweiten Argument von `CallMethods` gesetzte Grenze. Hier wird auch der Unterschied zwischen dem Rückgabewert *fail* und *NotApplicable* deutlich. Während im Fall *NotApplicable* die Hauptschleife zur nächsten Methode in der Datenbank übergeht, geht sie im Fall *fail* an den Anfang der Datenbank zurück. Man bemerke jedoch, dass dies allerding zu keiner Endlosschleife führt, da die jeweilige Methode solange übersprungen wird bis der Toleranzzähler erhöht wird. Erst nachdem der Zähler um eins nach oben gesetzt wird, werden auch Methoden, die zuvor *fail* zurückgegeben haben erneut aufgerufen.

4 Motivation des Projekts

Unser Augenmerk gilt Matrizengruppen, das heißt Untergruppen der allgemeinen linearen Gruppe $\mathrm{GL(n,q)}$, die aus allen invertierbaren $n \times n$ Matrizen über dem endlichen Körper von Ordnung q besteht. Wollen wir Matrizengruppen auf bestimmte Eigenschaften untersuchen, so stoßen wir oftmals auf Probleme, da die Ordnung von $\mathrm{GL(n,q)}$ exponentiell in n wächst und damit „gewöhnliche“ Methoden nicht selten scheitern lässt. Eine Abhilfe bilden hierbei randomisierte Algorithmen, die auf der zufälligen Wahl von Elementen aus der Matrizengruppe G beruhen. Beide im nächsten Absatz angesprochenen Algorithmen sind randomisiert.

Peter Neumann and Cheryl Praeger haben 1992 als erste einen Algorithmus [1] zur Erkennung der speziellen linearen Gruppe $\mathrm{SL(n,q)}$ entwickelt. Die $\mathrm{SL(n,q)}$ ist eine Untergruppe der $\mathrm{GL(n,q)}$, die aus Matrizen mit Determinante 1 besteht. Sechs Jahre später ist dieser Algorithmus in [2] von Alice Niemeyer und Cheryl Praeger verbessert, erweitert und in GAP implementiert worden. Allerdings ist die verallgemeinerte Version lediglich auf Matrizengruppen von Dimension $d \geq 3$ anwendbar. Demzufolge bestand meine Aufgabe nun darin, den von Alice Niemeyer angefertigten Algorithmus für den Fall $d = 2$ zu erweitern. Dabei orientierte ich mich an einem in [1] vorgeschlagenen Algorithmus.

5 Algorithmus

5.1 Satz

Der Algorithmus basiert auf einem von Peter Neumann and Cheryl Praeger veröffentlichte Paper „A recognition algorithm for special linear groups“ [1] und entscheidet, ob eine vorgegebene Matrizengruppe G die spezielle lineare Gruppe $\mathrm{SL(2,q)}$ enthält.

Sei Z das Zentrum der Gruppe G und somit die Untergruppe der Skalarmatrizen, die in G enthalten sind. Wir betrachten den folgenden Satz.

Satz *Sei $G \leq \mathrm{GL(2,q)}$ und $\mathrm{SL(2,q)} \nleq \mathrm{G}$. Dann gilt eine oder mehrere der folgenden Aussagen:*

1. *G ist reduzibel, das heißt G ist konjugiert zu einer Untergruppe von unteren 2×2 Dreiecksmatrizen;*

2. *G ist imprimitiv, das heißt G ist konjugiert zu einer Untergruppe von* $\mathrm{GL}(1, \mathrm{q}) \,\mathrm{wr}\, \mathrm{Sym}(2)$*;*

3. *G ist konjugiert zu einer Untergruppe von* $\Gamma\mathrm{L}(1, \mathrm{q}^2)$*;*

4. *G ist konjugiert zu einer Untergruppe von* $\mathrm{GL}(2, \mathrm{r}).\mathrm{Z}$*, wobei $\mathbb{F}_r$ einen echten Unterkörper von $\mathbb{F}_q$ bezeichne;*

5. *$G/Z \cong$* $\mathrm{Alt}(5)$*;*

6. *$G/Z \cong$* $\mathrm{Alt}(4)$ *oder* $\mathrm{Sym}(4)$*.*

5.2 Verfahren

Die obige Liste ermöglicht es nun, ein Verfahren zur Erkennung der speziellen linearen Gruppe $\mathrm{SL}(2, \mathrm{q})$ zu konstruieren. Die hierbei verwendeten Methoden können aus Kaptitel 6 entnommen werden. Bemerke, dass eine Gruppe G, die in mindestens eine der Kategorien 1 bis 4 des obigen Satzes fällt, die spezielle lineare Gruppe nicht enthält, während es für Gruppen aus Kategorie 5 und 6 einige wenige Fälle gibt, bei denen G/Z isomorph zur Gruppe $\mathrm{Alt}(5)$, $\mathrm{Alt}(4)$ oder $\mathrm{Sym}(4)$ ist, allerdings dennoch die $\mathrm{SL}(2, \mathrm{q})$ enthält. Beispielsweise ist $\mathrm{SL}(2, 5)/\mathrm{Z} \cong \mathrm{Alt}(5)$.

Sei also G durch eine Menge von invertierbaren 2×2 Matrizen $g_1, \ldots, g_k$ über dem endlichen Körper $\mathbb{F}_q$ erzeugt.

- Zunächst wird durch Methode 6.1.1 ein Pseudozufallselement $g \in G$ erzeugt.

- Im zweiten Schritt wird G auf Irreduzibilität getestet. Dabei heißt G irreduzibel, falls $\mathcal{V} = \mathbb{F}_q^2$, also der zu Grunde liegender Vektorraum, G-irreduzibel ist, das bedeutet, falls es keine echten G-invarianten Untervektorräume von $\mathcal{V}$ gibt. Dazu wird in Methode 6.2.1 untersucht wird, ob $\mathcal{V}$ bereits $\langle g \rangle$-irreduzibel ist. Trifft dies zu, dann ist auch G irreduzibel. Kann die Reduzibilität von G nach 15 Zufallselementen nicht ausgeschlossen werden, so bedient man sich in Methode 6.2.2 der so genannten MeatAxe, einem Algorithmus, der die Frage sicher beantworten kann.

  Ist G reduzibel, enthält G die $\mathrm{SL}(2, \mathrm{q}^2)$ nicht und der Algorithmus kann abgebrochen werden. Im Weiteren nehmen wir deshalb an, dass G irreduzibel auf $\mathcal{V}$ operiert.

- Sodann untersuchen wir in Methode 6.1.2 die Gruppe G auf Kommutativität.

  Falls dies der Fall ist, enthält G die spezielle lineare Gruppe nicht. Der Algorithmus wird abgebrochen. Im Weiteren können wir demnach annehmen, dass G nicht abelsch ist.

- Wir überprüfen, ob G/Z Exponent 2 hat. Wie schon bei der Frage nach Irreduzibilität betrachten wir zunächst das Zufallselement g, und überprüfen, ob sich hieraus Schlüsse für die gesamte Gruppe ziehen lassen. Falls also bereits für das Pseudozufallselement gilt, dass kein Element des endlichen Körpers $\mathbb{F}_q$ existiert, so dass $g^2 = eI$ (wobei I die 2×2 Einheitsmatrix ist), das heißt die projektive Ordnung von g ist größer 2, dann ist auch $\mathrm{Exp}(\mathrm{G}/\mathrm{Z}) > 2$ (Methode 6.1.1). Falls wir jedoch nach 15 Zufallselementen immernoch nicht ausschließen können, dass Exp(G/Z) = 2, untersuchen wir in Methode 6.1.3 die projektive Ordnung aller Generatoren.

Ist nun der Exponent der Faktorgruppe G/Z tatsächlich 2, so enthält G die spezielle lineare Gruppe sicher nicht. Wir können also abbrechen. Im Weiteren nehmen wir an, dass $\mathrm{Exp(G/Z)} > 2$ und dass wir in Methode 6.1.1 oder 6.1.3 ein Element $h \in G$ gefunden haben, so dass h^2 nicht im Zentrum von G liegt.

- Der nächste Schritt besteht darin herauszufinden, ob G zu einer Untergruppe von $\Gamma\mathrm{L}(1, \mathrm{q}^2)$ konjugiert ist, wofür uns Methode 6.1.4 dient. Da jedes Element von $\Gamma\mathrm{L}(1, \mathrm{q}^2)$ in einem Singerzykel von G liegt, ist es entweder irreduzibel oder skalar. Wir wissen, dass h^2 nicht skalar ist, da es nicht im Zentrum von G liegt. Falls h^2 reduzibel ist, dann ist G sicherlich nicht zu einer Untergruppe von $\Gamma\mathrm{L}(1, \mathrm{q}^2)$ konjugiert und wir können den Algorithmus abbrechen. An sonsten nehmen wir an dass h^2 irreduzibel ist und testen ob h^2 sogar primitiv irreduzibel ist. Dabei heißt in unserem Fall ein Element von G primitiv irreduzibel, falls es irreduzibel ist und seine Ordnung durch eine Primzahl geteilt wird, die zwar $q^2 - 1$ teilt, jedoch nicht $q - 1$. Ist h^2 primitiv irreduzibel dann untersuchen wir wie folgt, ob G zu einer Untergruppe von $\Gamma\mathrm{L}(1, \mathrm{q}^2)$ konjugiert ist oder auch nicht: G ist genau dann zu einer Untergruppe von $\Gamma\mathrm{L}(1, \mathrm{q}^2)$ konjugiert, wenn $\langle h^4 \rangle$ normal in G ist. Da h^4 zu einem Element eines Singerzyklus konjugiert ist und der Normalisator dieser zyklischen Gruppen genau $\Gamma\mathrm{L}(1, \mathrm{q}^2)$ ist, können wir die obige Frage genau dann mit „ja“ beantworten, wenn für alle $i = 1, \ldots, k$ stets $g_i^{-1} h^4 g_i h^4 = h^4 g_i^{-1} h^4 g_i$ gilt.

  Trifft dies zu wird der Algorithmus abgebrochen, da G in dem Fall die $\mathrm{SL}(2, \mathrm{q})$ nicht enthält.

- Methode 6.1.5 entscheidet, ob G imprimitiv ist.

  Falls dies der Fall ist, wird der Algorithmus abgebrochen, da eine solche Matrizengruppe die spezielle lineare Gruppe nicht enthält.

- Als nächstes überprüfen wir in Methode 6.1.6, ob G/Z isomorph zu Alt(5), Alt(4) oder Sym(4) ist.

  Falls dies zutrifft, und G nicht eine der wenigen Ausnahmen — nämlich $\mathrm{GL}(2, 3)$, $\mathrm{SL}(2, 3)$, $\mathrm{GL}(2, 4)$, $\mathrm{SL}(2, 4)$, $\mathrm{SL}(2, 5)$, $< Z\big(\mathrm{GL}(2, 5)\big), \mathrm{SL}(2, 5) >$ — ist, so enthält G die $\mathrm{SL}(2, \mathrm{q})$ nicht und der Algorithmus wird abgebrochen.

- Falls G all die bisherigen Tests bestanden hat, dann ist die Gruppe entweder modulo Z über einem echten Unterkörper von $\mathbb{F}_q$ realisierbar oder $\mathrm{SL}(2, \mathrm{q}) \leq \mathrm{G}$. Dies entscheiden wir mit Methode 6.1.7.

6 Methoden

In diesem Kapitel werden alle im Programm für Dimension 2 relevanten Methoden beschrieben. Dabei findet man einerseits neu definierte Funktionen, andererseits aber auch Methoden, die bereits für den Fall $d \geq 3$ vorhanden waren und mitbenutzt werden.

6.1 Neu definierte Methoden

In allen hier angeführten Methoden wird zu allererst die Dimension der Matrizengruppe *grp* bestimmt. Ist diese ungleich 2, so wird die jeweilige Methode mit dem Ausgabewert $false$ verlassen und nicht nochmal aufgerufen.

6.1.1 RECOG.TestRandomElementCase2 (*recognise*, *grp*)

Die Methode erzeugt zunächst ein Pseudozufallselement der Gruppe *grp*, berechnet sein charakteristisches Polynom und speichert dies im globalen Datensatz *recognise* unter *recognise.g* bzw. *recognise.cpol*. Weiterhin wird die Zählvariable *recognise.n* um eins erhöht. Falls noch kein Element *recognise.h* gefunden worden ist, dessen Quadrat nicht im Zentrum der Gruppe *grp* liegt, wird die projektive Ordnung von *recognise.g* berechnet und in die Menge *recognise.porders* dazugenommen. Falls die projektive Ordnung von *recognise.g* größer 2 ist, das heißt *recognise.*g^2 ist keine Skalarmatrix und liegt somit nicht im Zentrum von *grp*, wird *recognise.g* als *recognise.h* gespeichert. Der Ausgabewert ist $fail$, womit diese Methode nach Erhöhung des Toleranzzählers (vergleiche Kapitel 3.2) immer wieder aufgerufen wird.

```
RECOG.TestRandomElementCase2 := function ( recognise, grp )
 local g, porder;
 if recognise.d <> 2 then
   return false;
 fi;
 recognise.g := PseudoRandom( grp );
 recognise.cpol := CharacteristicPolynomial( recognise.g );
 recognise.n := recognise.n + 1;
 g := recognise.g;
 if recognise.needPOrders then
   porder := ProjectiveOrder( g );
   AddSet( recognise.porders, porder );
   if porder[1] > 2 then
     recognise.h := g;
     recognise.hasExp2 := false;
     recognise.needPOrders := false;
   fi;
 fi;
 return fail;
end;
```

6.1.2 RECOG.IsAbelian (*recognise*, *grp*)

Die Methode untersucht, ob die Gruppe *grp* abelsch ist und setzt *recognise.isAbelian* auf *true*, falls alle Gruppenelemente kommutieren, andernfalls auf $false$. Im Falle von Kommutativität wird *recognise.*$IsSLContained$ auf $false$ gesetzt und der Algorithmus mit dem Ausgabewert *true* beendet, da eine abelsche Gruppe die $\mathrm{SL}(2, q)$ nicht enthält. Bei nicht abelschen Gruppen ist der Ausgabewert $false$. Demnach wird in beiden Fällen diese Methode nicht nochmal aufgerufen.

```
RECOG.IsAbelian := function ( recognise, grp )
 if recognise.d <> 2 then
   return false;
 fi;
 if IsAbelian( grp ) then
   recognise.isAbelian := true;
   Info( InfoClassical, 2,
   ``The group is abelian and thus doesn't contain a classical group'');
   recognise.IsSLContained := false;
   return true;
 fi;
 recognise.isAbelian := false;
 return false;
end;
```

6.1.3 RECOG.HasExp2 (*recognise*, *grp*)

Falls dies nach 15 Durchläufen immernoch unbekannt ist, untersucht die Methode, ob die Gruppe *grp* modulo ihres Zentrums den Exponenten 2 hat oder nicht, und setzt im ersten Fall $recognise.hasExp2$ auf *true*, im zweiten auf $false$. Falls die Antwort bereits bekannt ist, wird die Methode mit dem Ausgabewert $false$ verlassen und nicht nochmal aufgerufen. Falls die Antwort zwar unbekannt ist, aber weniger als 16 Schleifendurchläufe ausgeführt wurden, wird die Methode mit dem Ausgabewert $fail$ beendet und zu einem späteren Zeitpunkt wieder aufgerufen.

Zunächst wird die projektive Ordnung der Generatoren der Gruppe bestimmt. Falls sich darunter ein Element mit projektiver Ordnung größer 2 befindet, ist auch der Exponent der Gruppe modulo des Zentrums größer 2. Das entsprechende Element wird sodann unter *recognise.h* im globalen Datensatz gespeichert und die Methode mit dem Ausgabewert $false$ verlassen und nicht nochmal aufgerufen. Falls alle Generatoren projektive Ordnung kleiner gleich 2 besitzen, so ist der Exponent der (nicht trivialen) Gruppe modulo des Zentrums gleich 2. Da Eine Gruppe, die modulo des Zentrums Exponent 2 hat, die $\mathrm{SL}(2, q)$ nicht enthält, wird in diesem Fall $recognise.IsSLContained$ auf $false$ gesetzt und der Algorithmus mit dem Ausgabewert *true* beendet.

```
RECOG.HasExp2 := function ( recognise, grp )
 local generators, gen, porder;
 if recognise.d <> 2 then
   return false;
 fi;
 if recognise.hasExp2 <> ``unknown'' then
   return false;
 fi;
```

```
generators := recognise.generators;
if recognise.n > 15 then
  for gen in generators do
   porder := ProjectiveOrder( gen );
   if porder[1] > 2 then
    recognise.h := gen;
    recognise.hasExp2 := false;
    return false;
   fi;
  od;
  recognise.hasExp2 := true;
  Info( InfoClassical, 2, ``The group modulo scalars has exponent 2
  and thus doesn't contain a classical group'');
  recognise.IsSLContained := false;
  return true;
fi;
return fail;
end;
```

6.1.4 RECOG.IsSubgroupOfGammaL (*recognise*, *grp*)

Die Methode bestimmt, ob die Gruppe *grp* zu einer Untergruppe von $\Gamma\mathrm{L}(1,\mathrm{q}^2)$ konjugiert ist. Falls das Element *recognise.h* noch nicht gefunden wurde oder *recognise.*h^2 nicht primitiv irreduzibel ist, wird $fail$ ausgegeben und die Methoden erst in einem Schleifendurchlauf mit höherer Toleranz aufgerufen — in der Hoffnung, dass dann ein geeignetes Element *recognise.h* gefunden worden ist. Um ein neues Element zu finden, wird im zweiten Fall *recognise.needPOrders* gleich *true* gesetzt.

Falls das charakteristische Polynom von *recognise.h* reduzibel ist und somit *recognise.*h^2 reduzibel auf $\mathcal{V} = \mathbb{F}_q^2$ operiert, ist *grp* nicht zu einer Untergruppe von $\Gamma\mathrm{L}(1,\mathrm{q}^2)$ konjugiert. Wir setzen *recognise.isSubgroupOfGammaL* auf $false$ und verlassen die Methode mit dem Ausgabewert $false$. An sonsten nehmen wir an dass h^2 irreduzibel ist und testen im weiteren Verlauf von Methode 6.1.4, ob G zu einer Untergruppe von $\Gamma\mathrm{L}(1,\mathrm{q}^2)$ konjugiert ist oder auch nicht, indem wir überprüfen, ob das Element *recognise.*h^4 für alle Erzeuger *gen* von *grp* mit gen^{-1}∗*recognise.*h^4∗*gen* kommutiert. Trifft dies zu, wird *recognise.IsSubgroupOfGammaL* auf *true* und *recognise.IsSLContained* auf $false$ gesetzt und der Algorithmus mit dem Ausgabewert *true* beendet. Ist *grp* nicht zu einer Untergruppe von $\Gamma\mathrm{L}(1,\mathrm{q}^2)$ konjugiert, so setzen wir *recognise.isSubgroupOfGammaL* auf $false$ und verlassen die Methode mit dem Ausgabewert $false$.

```
RECOG.IsSubgroupOfGammaL := function ( recognise, grp )
 local h, gen, generators, order, x, x2, charPol;
 if recognise.d <> 2 then
   return false;
 fi;
```

```
  if recognise.h = fail then
    return fail;
  fi;
  h := recognise.h;
  q := recognise.q;
  charPol := CharacteristicPolynomial( h^2 ); x := h^4;
  generators := recognise.generators;
  if not IsIrreducible( charPol ) then
    recognise.isSubgroupOfGammaL := false;
    return false;
  fi;
  order := Order(h^2);
  if ( Order( h^2 ) in Factors( q^2 - 1 ) ) = false or
  Order( h^2 ) in Factors( q-1 ) then
    recognise.needPOrdes := true;
    return fail;
  fi;
  for gen in generators do
    x2 := x^gen;
    if (x2 * x <> x * x2) then
      recognise.isSubgroupOfGammaL := false;
      return false;
    fi;
  od;
  recognise.isSubgroupOfGammaL := true;
  Info( InfoClassical, 2, ``The group is conjugate to a subgroup of
  GammaL(1,'', q, ``) and thus doesn't contain a classical group'');
  recognise.IsSLContained := false;
  return true;
end;
```

6.1.5 RECOG.IsImprimitive (*recognise*, *grp*)

Methode 6.1.5 entscheidet, ob die Gruppe *grp* imprimitiv ist. Falls dies der Fall ist, wird *recognise.isImprimitive* gleich *true* und *recognise.IsSLContained* auf *false* gesetzt, da eine solche Matrizengruppe die spezielle lineare Gruppe nicht enthält. Der Ausgabewert ist dann *true*, das heißt der Algorithmus wird beendet. Falls die *grp* nicht imprimitiv ist, so setzen wir *recognise.isImprimitive* auf *false* und verlassen die Methode mit dem Ausgabewert *false*.

Falls das Element *recognise.h* noch nicht gefunden wurde, wird *fail* ausgegeben und die Methode erst in einem Schleifendurchlauf mit höherer Toleranz aufgerufen. Um die Frage nach der Imprimitivität zu beantworten, betrachten wir die Eigenräume von *recognise.*h^2. Dazu berechnen wir die Eigenvektoren von *recognise.*h^2 und speichern diese im globalen Datensatz unter *recognise.eigenvectors*. Falls das Quadrat eines Elementes x von $\mathrm{GL}(1,\mathrm{q}) \,\mathrm{wr}\, \mathbb{Z}_2$ keine

Skalarmartix ist, ist $\mathcal{V}$ eine direkte Summe der Eigenräume von x^2 und die Faktoren dieser Zerlegung werden von x^2 erhalten. Falls *recognise*.h^2 nicht zwei verschieden Eigenräume besitzt, ist *grp* nicht imprimitiv. Anderenfalls, da *recognise*.h^2 nicht skalar ist, müssen wir überprüfen, ob die Eigenräume $\mathcal{V}_1$ und $\mathcal{V}_2$ von *recognise*.h^2 erhalten werden. Falls alle Generatoren das Paar $\{\mathcal{V}_1, \mathcal{V}_2\}$ erhalten — das heißt jeder der Generatoren ist bezüglich einer Basis aus Eigenvektoren eine monomiale Matrix — so ist *grp* imprimitiv. Sonst ist *grp* primitiv.

```
RECOG.IsImprimitive := function ( recognise, grp )
 local h, f, gen, genNew, generators, eigenvectors;

 if recognise.d <> 2 then
   return false;
 fi;
 if recognise.h = fail then
   return fail;
 fi;
 h := recognise.h;
 f := recognise.field;
 generators := recognise.generators;
 eigenvectors := Eigenvectors( f, h^2 );
 if Length(eigenvectors) <> 2 then
   recognise.isImprimitive := false;
   return false;
 fi;
 for gen in generators do
   genNew := gen^eigenvectors;
   if not IsMonomialMatrix( genNew ) then
     recognise.isImprimitive := false;
     return false;
   fi;
 od;
 recognise.isImprimitive := true;
 Info( InfoClassical, 2, ''The group is imprimitive
 and thus doesn't contain a classical group'');
 recognise.IsSLContained := false;
 return true;
end;
```

6.1.6 RECOG.IsAlt5Alt4Sym4 (*recognise*, *grp*)

Die Methode untersucht, ob die Gruppe *grp* modulo ihres Zentrums isomorph zur Gruppe Alt(5), Alt(4) oder Sym(4) ist. Falls dies zutrifft, und *grp* nicht eine der wenigen Ausnahmen, das heißt $\mathrm{GL}(2,3), \mathrm{SL}(2,3), \mathrm{GL}(2,4), \mathrm{SL}(2,4), \mathrm{SL}(2,5), <\mathrm{Z}\big(\mathrm{GL}(2,5)\big), \mathrm{SL}(2,5)>$ ist, so enthält *grp* die $\mathrm{SL}(2, \mathrm{q})$ nicht. Wir setzen *recognise.IsSLContained* auf *false* und der Algorithmus wird mit dem Ausgabewert *true* abgebrochen. Falls dies nicht zutrifft, oder *grp* unter eine der Ausnahmen fällt, dann ist der Ausgabewert *false*.

Um die Frage zu beantworten, betten wir *grp* in die symmetrische Gruppe Sym(n) ein, wobei n der Anzahl der 1-dimensionalen Untervektorräume gleicht und speichern das Resultat im globalen Datensatz unter *recognise.pgrp*. Falls die Anzahl der von *recognise.pgrp* bewegten Punkte kleiner gleich 5 ist, die Ordnung der Gruppe $|\mathrm{Alt}(5)| = 60$, $|\mathrm{Alt}(4)| = 12$ oder $|\mathrm{Sym}(4)| = 24$ gleicht und *recognise.pgrp* eine alternierende oder symmetrische Gruppe ist, wird bis auf die oben erwähnten Ausnahmen *recognise.isAlt5Alt4Sym4* auf *true* gesetzt — anderenfalls auf *false*.

```
RECOG.IsAlt5Alt4Sym4 := function ( recognise, grp )
 local pgrp, q;
 if recognise.d <> 2 then
  return false;
 fi;
 pgrp := ProjectiveActionOnFullSpace( grp, recognise.field, 2 );
 recognise.pgrp := pgrp; q := recognise.q;
 if NrMovedPoints( pgrp ) <= 5 then
   if ( Size( pgrp ) = 12 and q <> 3 ) or
    ( Size( pgrp ) = 24 and q <> 3 ) or
    ( Size( pgrp ) = 60 and q <> 4 and q <> 5 ) then
    if IsAlternatingGroup( pgrp ) or IsSymmetricGroup( pgrp ) then
     recognise.isAlt5Alt4Sym4 := true;
     Info( InfoClassical, 2, ``The group modulo scalars is isomorphic
     Alt5, ALt4 or Sym4 and thus doesn't contain a classical group'';
     recognise.IsSLContained := false;
     return true;
    fi;
  fi;
 fi;
 recognise.isAlt5Alt4Sym4 := false;
 return false;
end;
```

6.1.7 RECOG.IsSL2Contained (*recognise*, *grp*)

Die Methode überprüft den jeweiligen Informationsstand, das heißt sie testet, ob bereits genügend Informationen zu Verfügung stehen, um zu entscheiden, dass die Gruppe *grp* die spezielle lineare Gruppe $\mathrm{SL}(2, \mathrm{q})$ enthält.

Falls die Gruppe *grp* reduzibel ist oder mindestens einer der Werte *recognise.isAbelian*, *recognise.hasExp2*, *recognise.isSubgroupOfGammaL*, *recognise.isImprimitive* oder auch *recognise.isAlt5Alt4Sym4* nicht gleich *false* ist, so wird die Methode mit dem Ausgabewert *fail* verlassen. Anderenfalls ist *grp* modulo des Zentrums entweder über einem echten Unterkörper von $\mathbb{F}_q$ realisierbar oder enthält die spezielle lineare Gruppe $\mathrm{SL}(2, \mathrm{q})$. Die

Gruppe *grp* ist modulo ihres Zentrums genau dann über einem echten Unterkörper von $\mathbb{F}_q$ realisierbar, falls *grp* intransitiv auf den $q+1$ eindimensionalen Untervektorräumen von $\mathcal{V}$ operiert, das heißt falls *recognise.pgrp* intransitiv ist. Ist dies der Fall, so setzen wir *recognise.isRepresentableOverSubfield* auf $false$ und *recognise.IsSLContained* auf *true* und verlassen die Methode mit dem Ausgabewert $false$. Anderenfalls setzen wir umgekehrt *recognise.isRepresentableOverSubfield* auf *true*, *recognise.IsSLContained* auf $false$ und brechen den Algorithmus mit dem Ausgabewert *true* ab, da eine solche Gruppe die $\mathrm{SL}(2, \mathrm{q})$ nicht enthält.

```
RECOG.IsSL2Contained := function( recognise, grp )
 if recognise.d <> 2 then
   return false;
 fi;
 if recognise.isReducible = true then
   return false;
 fi;
 if (recognise.isReducible = false and
  recognise.isAbelian = false and
  recognise.hasExp2 = false and
  recognise.isSubgroupOfGammaL = false and
  recognise.isImprimitive = false and
  recognise.isAlt5Alt4Sym4 = false ) then
   if IsTransitive( recognise.pgrp ) then
    recognise.isRepresentableOverSubfield := false;
    Info(InfoClassical,2,``The group is not generic and containes'');
    Info(InfoClassical,2,``SL('', 2, ``, '', recognise.q, ``);'');
    recognise.IsSLContained := true;
    return true;
   fi;
   recognise.isRepresentableOverSubfield := true;
   Info( InfoClassical, 2, ``The group is representable over a proper
   subfield and thus doesn't contain a classical group'');
   recognise.IsSLContained := false;
   return true;
 fi;
 return fail;
end;
```

6.2 Vorhandene Methoden

Der Vollständigkeit halber listen wir hier zwei Methoden auf, die bereits für den Fall $d > 2$ implementiert wurden und nun mitverwendet werden.

6.2.1 RECOG.IsReducible (*recognise*, *grp*)

Die Methode untersucht, ob das derzeitige Zufallselement bereits auf die Irreduzibilität der Gruppe *grp* schließen lässt.

```
RECOG.IsReducible := function( recognise, grp )
 local deg, dims, g;
 deg := List( Factors( recognise.cpol ), i-> Degree( i ) );
 dims := [ 0 ];
 for g in deg do
   UniteSet( dims, dims + g );
 od;
 if IsEmpty( recognise.dimsReducible ) then
   recognise.dimsReducible := dims;
 else
   IntersectSet( recognise.dimsReducible, dims );
 fi;
 if Length( recognise.dimsReducible ) = 2 then
   recognise.isReducible := false;
   return false;
 fi;
 return fail;
end;
```

6.2.2 RECOG.MeatAxe (*recognise*, *grp*)

Falls dies nach 15 Schleifendurchläufen immer noch nicht bekannt ist, untersucht diese Methode die Gruppe *grp* auf Irreduzibilität, indem sie den MeatAxe Algorithmus anwendet.

```
RECOG.MeatAxe := function( recognise, grp )
 if recognise.n > 15 then
   recognise.needMeataxe := true;
 fi;
 if recognise.needMeataxe <> true
   then return NotApplicable;
 fi;
 if MTX.IsIrreducible( recognise.module ) then
   recognise.isReducible := false;
   return false;
 else
   Info( InfoClassical, 2, ``The group acts reducibly
   and thus doesn't contain a classical group'');
   recognise.isReducible := true;
   recognise.IsSLContained := false;
```

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

7 Beispiele

7.1 $\mathrm{SL}(2,\mathrm{q}) \nleq \mathrm{G}$

Beispiel einer abelschen Gruppe

```
gap> LoadPackage(``recog'');;
gap> m1 := [ [ 0*Z(8), Z(8)^3 ], [ Z(8)^2, 0*Z(8) ] ];;
gap> g := Group([m1]);;
gap> RecogniseClassical(g);


#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 80 method ``IsReducible'': fail.
#I Finished rank 69 method ``IsAbelian'': success.
rec( field := GF(2^3), d := 2, p := 2, a := 3, q := 8, E := [ ], LE := [
    ],
BE := [ ], LB := [ ], LS := [ ], E2 := [ ], LE2 := [ ], BE2 := [ ],
g := [ [ Z(2^3)^3, 0*Z(2) ], [ 0*Z(2), Z(2^3)^3 ] ], cpol := x_1^2+Z(2^3)^6,
isppd := fail, n := 1,
module := rec( field := GF(2^3), isMTXModule := true, dimension := 2,
   generators := [ [ [ 0*Z(2), Z(2^3)^3 ], [ Z(2^3)^2, 0*Z(2) ] ] ] ),
currentgcd := 2, isReducible := ``unknown'', isGeneric := false,
isNotExt := ``unknown'', hint := ``unknown'', hintIsWrong := false,
isNotMathieu := ``unknown'', isNotAlternating := ``unknown'',
isNotPSL := ``unknown'', possibleNearlySimple := [ ],
dimsReducible := [ 0, 1, 2 ], orders := [ ], porders := [ ],
hasSpecialEle := false, bc := ``unknown'', kf := ``unknown'', plusminus :=
    [ ],
sq1 := [ [ [ Z(2^3), 0*Z(2) ], [ 0*Z(2), Z(2^3) ] ] ],
sq2 := [ [ [ Z(2^3), 0*Z(2) ], [ 0*Z(2), Z(2^3) ] ] ],
scalars := Group([ [ [ Z(2^3), 0*Z(2) ], [ 0*Z(2), Z(2^3) ] ] ]),
needMeataxe := false, needForms := false, needOrders := false,
needPOrders := false, needBaseChange := false, needKF := false,
needPlusMinus := false, needDecompose := false, needLB := false,
needE2 := false, maybeDual := true, maybeFrobenius := false,
ClassicalForms := [ ], isAbelian := true, h := fail, hasExp2 := ``unknown'',
isSubgroupOfGammaL := ``unknown'', isImprimitive := ``unknown'',
generators := [ [ [ 0*Z(2), Z(2^3)^3 ], [ Z(2^3)^2, 0*Z(2) ] ] ],
isAlt5Alt4Sym4 := ``unknown'', isRepresentableOverSubfield := ``unknown'',
pgrp := ``unknown'', IsSLContained := false, IsSpContained := ``unknown'',
IsSUContained := ``unknown'', IsSOContained := ``unknown'' )
```

Beispiel einer primitiven Gruppe

```
gap> LoadPackage(‘‘recog’’);
gap> m1 := [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7)^2 ] ];;
gap> m2 := [ [ 0*Z(7), Z(7)^3 ], [ Z(7)^2, 0*Z(7) ] ];;
gap> g := Group([m1,m2]);;
gap> RecogniseClassical(g);

#I Finished rank 99 method ‘‘TestRandomElementCase2’’: fail.
#I Finished rank 80 method ‘‘IsReducible’’: fail.
#I Finished rank 68 method ‘‘HasExp2’’: fail.
#I Finished rank 67 method ‘‘IsSubgroupOfGammaL’’: fail.
#I Finished rank 66 method ‘‘IsImprimitive’’: fail.
#I Finished rank 15 method ‘‘IsSL2Contained’’: fail.
#I Increasing tolerance to 0
#I Finished rank 99 method ‘‘TestRandomElementCase2’’: fail.
#I Finished rank 80 method ‘‘IsReducible’’: fail.
#I Finished rank 66 method ‘‘IsImprimitive’’: success.
rec( field := GF(7), d := 2, p := 7, a := 1, q := 7, E := [ ], LE := [ ],
BE := [ ], LB := [ ], LS := [ ], E2 := [ ], LE2 := [ ], BE2 := [ ],
g := [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7)^2 ] ],
cpol := x_1^2+Z(7)^2*x_1-Z(7)^0, isppd := fail, n := 2,
module := rec( field := GF(7), isMTXModule := true, dimension := 2,
generators := [ [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7)^2 ] ],
[ [ 0*Z(7), Z(7)^3 ], [ Z(7)^2, 0*Z(7) ] ] ] ), currentgcd := 2,
isReducible := ‘‘unknown’’, isGeneric := false, isNotExt := ‘‘unknown’’,
hint := ‘‘unknown’’, hintIsWrong := false, isNotMathieu := ‘‘unknown’’,
isNotAlternating := ‘‘unknown’’, isNotPSL := ‘‘unknown’’,
possibleNearlySimple := [ ], dimsReducible := [ 0, 1, 2 ], orders := [ ],
porders := [ [ 6, Z(7)^0 ] ], hasSpecialEle := false, bc := ‘‘unknown’’,
kf := ‘‘unknown’’, plusminus := [ ],
sq1 := [ [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7) ] ] ],
sq2 := [ [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7) ] ] ],
scalars := Group([ [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7) ] ] ]),
needMeataxe := false, needForms := false, needOrders := false,
needPOrders := false, needBaseChange := false, needKF := false,
needPlusMinus := false, needDecompose := false, needLB := false,
needE2 := false, maybeDual := true, maybeFrobenius := false,
ClassicalForms := [ ], isAbelian := false,
h := [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7)^2 ] ], hasExp2 := false,
isSubgroupOfGammaL := false, isImprimitive := true,
generators := [ [ [ Z(7), 0*Z(7) ], [ 0*Z(7), Z(7)^2 ] ],
[ [ 0*Z(7), Z(7)^3 ], [ Z(7)^2, 0*Z(7) ] ] ], isAlt5Alt4Sym4 := false,
isRepresentableOverSubfield := ‘‘unknown’’,
pgrp := Group([ (3,4,5,6,7,8), (1,2)(3,4)(5,8)(6,7) ]),
IsSLContained := false, IsSpContained := ‘‘unknown’’,
IsSUContained := ‘‘unknown’’, IsSOContained := ‘‘unknown’’ )
```

7.2 $\mathrm{SL}(2, \mathrm{q}) \leq \mathrm{G}$

```
gap> LoadPackage(``recog'');
gap> m1 := [ [ Z(11), 0*Z(11) ], [ 0*Z(11), Z(11) ] ];;
gap> m2 := [ [ Z(11^2), 0*Z(11) ], [ 0*Z(11), Z(11^2)^119 ] ];;
gap> m3 := [ [ Z(11)^5, Z(11)^0 ], [ Z(11)^5, 0*Z(11) ] ];;
gap> g := Group( [ m1, m2, m3 ] );;
gap> RecogniseClassical(g);

#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 68 method ``HasExp2'': fail.
#I Finished rank 67 method ``IsSubgroupOfGammaL'': fail.
#I Finished rank 66 method ``IsImprimitive'': fail.
#I Finished rank 15 method ``IsSL2Contained'': fail.
#I Increasing tolerance to 0
#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 67 method ``IsSubgroupOfGammaL'': fail.
#I Finished rank 15 method ``IsSL2Contained'': fail.
#I Increasing tolerance to 1
#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 67 method ``IsSubgroupOfGammaL'': fail.
#I Finished rank 15 method ``IsSL2Contained'': fail.
#I Increasing tolerance to 2
#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 67 method ``IsSubgroupOfGammaL'': fail.
#I Finished rank 15 method ``IsSL2Contained'': fail.
#I Increasing tolerance to 3
#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 67 method ``IsSubgroupOfGammaL'': fail.
#I Finished rank 15 method ``IsSL2Contained'': fail.
#I Increasing tolerance to 4
#I Finished rank 99 method ``TestRandomElementCase2'': fail.
#I Finished rank 15 method ``IsSL2Contained'': success.
rec( field := GF(11^2), d := 2, p := 11, a := 2, q := 121, E := [ ],
LE := [ ], BE := [ ], LB := [ ], LS := [ ], E2 := [ ], LE2 := [ ],
BE2 := [ ], g := [ [ Z(11^2)^100, Z(11)^9 ], [ Z(11^2)^80, Z(11^2)^106 ]
    ],
cpol := x_1^2+Z(11^2)^45*x_1+Z(11)^0, isppd := fail, n := 6,
module := rec( field := GF(11^2), isMTXModule := true, dimension := 2,
generators := [ [ [ Z(11), 0*Z(11) ], [ 0*Z(11), Z(11) ] ],
[ [ Z(11^2), 0*Z(11) ], [ 0*Z(11), Z(11^2)^119 ] ],
[ [ Z(11)^5, Z(11)^0 ], [ Z(11)^5, 0*Z(11) ] ] ] ), currentgcd := 2,
isReducible := false, isGeneric := false, isNotExt := ``unknown'',
hint := ``unknown'', hintIsWrong := false, isNotMathieu := ``unknown'',
isNotAlternating := ``unknown'', isNotPSL := ``unknown'',
possibleNearlySimple := [ ], dimsReducible := [ 0, 2 ], orders := [ ],
porders := [ [ 61, Z(11)^4 ], [ 61, Z(11)^5 ], [ 61, Z(11)^6 ],
```

```
[ 61, Z(11)^8 ] ], hasSpecialEle := false, bc := ‘‘unknown’’,
kf := ‘‘unknown’’, plusminus := [ ],
sq1 := [ [ [ Z(11^2), 0*Z(11) ], [ 0*Z(11), Z(11^2) ] ] ],
sq2 := [ [ [ Z(11^2), 0*Z(11) ], [ 0*Z(11), Z(11^2) ] ] ],
scalars := Group([ [ [ Z(11^2), 0*Z(11) ], [ 0*Z(11), Z(11^2) ] ] ]),
needMeataxe := false, needForms := false, needOrders := false,
needPOrders := false, needBaseChange := false, needKF := false,
needPlusMinus := false, needDecompose := false, needLB := false,
needE2 := false, maybeDual := true, maybeFrobenius := true,
ClassicalForms := [ ], isAbelian := false,
h := [ [ Z(11^2)^100, Z(11)^9 ], [ Z(11^2)^80, Z(11^2)^106 ] ],
hasExp2 := false, isSubgroupOfGammaL := false, isImprimitive := false,
generators := [ [ [ Z(11), 0*Z(11) ], [ 0*Z(11), Z(11) ] ],
[ [ Z(11^2), 0*Z(11) ], [ 0*Z(11), Z(11^2)^119 ] ],
[ [ Z(11)^5, Z(11)^0 ], [ Z(11)^5, 0*Z(11) ] ] ],
isAlt5Alt4Sym4 := false, isRepresentableOverSubfield := false,
pgrp := <permutation group with 3 generators>, IsSLContained := true,
IsSpContained := ‘‘unknown’’, IsSUContained := ‘‘unknown’’,
IsSOContained := ‘‘unknown’’ )
```

8 Bibliografie

Literatur

[1] Peter M. Neumann and Cheryl E. Praeger. *A recognition algorithm for special linear groups*. Proc. London Math. Soc. (3), 65:555-603, 1992.

[2] Alice C. Niemeyer and Cheryl E. Praeger. *A recognition algorithm for classical groups over finite fields*. Proc. London Math. Soc., 77:117-169, 1998.

[3] The GAP Group, *GAP — Groups, Algorithms,and Programming*. Version 4.4.9, 2006. (http://www.gap-system.org)