Conjugacy Classes in Finite Unitary Groups

D. E. Taylor

Version of 26 June, 2021
TEXed: 3 August, 2021

This treatment of the general unitary groups is modelled on the reports 'Conjugacy Classes in Finite Special and General Linear Groups' and 'Conjugacy Classes in Finite Symplectic Groups'. It also uses ideas from the preprint [2] and the C code of Sergei Haller implementing conjugacy classes for the conformal unitary groups (see the file `classes_classical.c`).

The conjugacy classes are obtained by first computing a complete collection of invariants, determining a representative matrix for each invariant and then computing the order of its centraliser directly from the invariant. This construction is based on the work of Wall [4].

1 Finite unitary groups

For a prime power q, let $k = \mathrm{GF}(q^2)$ and let $\sigma : k \to k : x \mapsto \bar{x}$ denote the automorphism of k defined by $\bar{x} = x^q$. If $f(t) = a_0 + a_1 t + \cdots + a_n t^n \in k[t]$, let $\overline{f}(t) = \bar{a}_0 + \bar{a}_1 t + \cdots + \bar{a}_n t^n$.

The general unitary group

The *general unitary group* $\mathrm{GU}(n,q)$ considered here is the set of $n \times n$ matrices A over the field k such that $A\Lambda\overline{A}^{\mathrm{tr}} = \Lambda$, where A^{tr} is the transpose of A and Λ is the 'standard' hermitian form

$$\Lambda_n = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ & & \iddots & & \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

The description of the conjugacy classes of $\mathrm{GU}(n,q)$ closely parallels the description of the conjugacy classes of $\mathrm{GL}(n,q)$ and $\mathrm{Sp}(2n,q)$.

The group $\mathrm{GU}(n,q)$ acts on $V = k^n$ and for $g \in \mathrm{GU}(n,q)$, the space V becomes a $k[t]$-module V_g by defining $vf(t) = vf(g)$ for all $v \in V$ and all $f(t) \in k[t]$. The hermitian form on V preserved by $\mathrm{GU}(n,q)$ is defined by $\beta(u,v) = u\Lambda\overline{v}^{\mathrm{tr}}$.

The conformal unitary group

The *conformal unitary group* $\mathrm{CGU}(n,q)$ is the group of linear transformations g such that $\beta(ug, vg) = \lambda(g)\beta(u,v)$ for some $\lambda(g) \in k^\times$ and all $u, v \in V$. In matrix terms this is equivalent to $A\Lambda\overline{A}^{\mathrm{tr}} = \lambda(g)\Lambda$. On applying the automorphism σ we see that $\lambda(g) \in k_0$, the fixed field of σ.

If we identify $\theta \in k$ with the corresponding scalar matrix we have $\beta(u\theta, v\theta) = \theta\bar{\theta}$ and since the norm map is onto we see that $\mathrm{CGU}(n,q) = Z\ \mathrm{GU}(n,q)$, where Z is the centre of $\mathrm{CGU}(n,q)$.

Therefore, if $g_1, g_2, \ldots, g_r$ represent the conjugacy classes of $\mathrm{GU}(n,q)$, the conjugacy classes of $\mathrm{CGU}(n,q)$ are represented by the elements $\zeta^i g_j$, where ζ is a primitive element of k, $1 \le i < q$ and $1 \le j \le r$. Thus from now on we restrict our attention to the general unitary group.

```
import "common.m" : centralJoin, convert;
import "Classes/translate/translateU.m" : tagToNameGU, tagToNameSU;
```

2 Polynomials and partitions

If $\mathcal{P}$ is the set of all partitions and Φ is the set of all monic irreducible polynomials (other than t), then for $g \in \mathrm{GL}(n,q^2)$ there is a function $\mu : \Phi \to \mathcal{P}$ such that

$$V_g = \bigoplus_{f\in\Phi,i} k[t]/(f)^{\mu_i(f)} \tag{2.1}$$

and $\mu(f) = (\mu_1(f), \mu_2(f), \ldots,)$ is a partition such that

$$\sum_{f\in\Phi} \deg(f)|\mu(f)| = n.$$

If $g \in \mathrm{GU}(n,q)$ there are restrictions—to be determined in the sections which follow—on the polynomials that can occur in this decomposition.

Polynomials

Definition 2.1.

(i) Let $f(t) \in k[t]$ be a monic polynomial of degree d such that $f(0) \neq 0$. The *twisted dual* of $f(t)$ is the polynomial

$$\tilde{f}(t) = \overline{f(0)}^{-1} t^d \overline{f}(t^{-1}).$$

(ii) The polynomial $f(t)$ is $\sim$*symmetric* if $\tilde{f}(t) = f(t)$.

(iii) A polynomial $f(t)$ is $\sim$*irreducible* if it is $\sim$symmetric and has no proper $\sim$symmetric factors.

The monic polynomial $f(t) = a_0 + a_1 t + \cdots + a_{d-1}t^{d-1} + t^d$ is $\sim$symmetric if and only if

$$a_0\overline{a}_0 = 1 \quad \text{and} \quad a_{d-i} = a_0\overline{a}_i \quad \text{for } 0 < i < d. \tag{2.2}$$

It is clear that for monic polynomials f and g we have $\tilde{\tilde{f}} = f$ and $\widetilde{(fg)} = \tilde{f}\tilde{g}$.

```
intrinsic TildeDualPolynomial( f :: RngUPolElt ) -> RngUPolElt
{ The twisted dual of the polynomial f }
  eseq := Coefficients(f);
  a0 := eseq[1];
```

```
    require a_0 ne 0 : "Polynomial must have non-zero constant term";
    K := CoefficientRing(f);
    flag, q := IsSquare(#K);
    require flag : "Field size must be a square";
    cseq := [e^q : e in eseq ];
    return cseq[1]^-1 * Parent(f) ! Reverse(cseq);
end intrinsic;
```

If g preserves the hermitian form β, then for all $u, v \in V$ we have

$$\beta(ug, v) = \beta(u, vg^{-1})$$

and thus for $f(t) \in k[t]$

$$\beta(uf(g), v) = \beta(u, v\overline{f}(g^{-1})). \tag{2.3}$$

In particular, if $m(t)$ is the minimal polynomial of g, then $v\overline{m}(g^{-1}) = 0$ for all v and therefore $g^d\overline{m}(g^{-1}) = 0$, where d is the degree of $m(t)$. Thus $\tilde{m}(g) = 0$ and it follows that $\tilde{m}(t) = m(t)$; that is, the minimal polynomial of g is $\sim$symmetric.

Lemma 2.2. *Let $f(t)$ be a monic $\sim$irreducible polynomial.*

(i) *If $f(t)$ is reducible, there exists an irreducible polynomial $h(t)$ such that $f(t) = h(t)\tilde{h}(t)$ and $h(t) \neq \tilde{h}(t)$.*

(ii) *If $f(t)$ is irreducible, the degree d of $f(t)$ is odd and $\xi^{q^d+1} = 1$ for all ξ such that $f(\xi) = 0$.*

Proof. (i) Suppose that $h(t)$ is an irreducible factor of $f(t)$. Then $\tilde{g}(t)$ divides $\tilde{f}(t) = f(t)$ and since $f(t)$ is $\sim$irreducible $f(t) = h(t)\tilde{h}(t)$ and $\tilde{h}(t) \neq h(t)$.

(ii) (Ennola [1, Lemma 2]) Let ξ be a root of f and let e be the order of ξ. Then $f = \tilde{f}$ implies $\overline{f}(\xi^{-1}) = 0$ and so $f(\xi^{-q}) = 0$. The Galois group $\mathrm{Gal}(k[\xi], k)$ is generated by

$$\tau : x \mapsto x^{q^2}$$

and therefore $\xi^{-q} = \tau^i(\xi) = \xi^{q^{2i}}$ for some i such that $0 \leq i < d$. Since $|k[\xi]| = q^{2d}$ we also have $\xi^{q^{2d}} = \xi$. Thus $q^{2i-1} \equiv -1 \pmod{e}$ and $q^{2d} \equiv 1 \pmod{e}$.

Let $s = \gcd(2i-1, 2d)$. There exist integers a and b such that

$$s = a(2i-1) + 2bd$$

and it follows that $q^s = q^{a(2i-1)+2bd} \equiv (-1)^a \pmod{e}$. But s is odd, hence s divides d and since a is odd, $q^s \equiv -1 \pmod{e}$. Thus if d is even, $q^d \equiv 1 \pmod{e}$. But then $\tau^{d/2}(\xi) = \xi$, which is a contradiction. Therefore d is odd, $q^d \equiv -1$ and so $\xi^{q^d+1} = 1$. □

```
intrinsic TildeIrreduciblePolynomials( q :: RngIntElt, d :: RngIntElt ) -> SeqEnum
{All monic polynomials of degree d with no proper
 tilde-symmetric factors}
  K := GF(q^2);
  P := PolynomialRing(K);
```

ξ is a generator of the group of field elements of norm 1.

```
  xi := PrimitiveElement(K)^(q-1);
```

```
    if d eq 1 then return [P| P.1−ξ^i : i in [0..q] ]; end if;
    m := d div 2;
    if IsEven(d) then
        pols := (m eq 1) select [P| P.1−a : a in K | a ne 0 ]
          else AllIrreduciblePolynomials(K, m);
        pols := SetSeq({ h*hh : h in pols | h ne hh where hh is TildeDualPolynomial(h) });
    else    // d is odd
        pols := [P| ];
        V := VectorSpace(K, m);
```

Construct all $\sim$symmetric polynomials of degree d.

```
        for i := 0 to q do
            a0 := ξ^i;
            for v in V do
                eseq := [a0] cat EltSeq(v);
                eseq cat:= [a0*eseq[m+1−j]^q : j in [0..m−1]] cat [K ! 1];
                f := P ! eseq;
                if IsIrreducible(f) then Append(~pols, f); end if;
            end for;
        end for;
    end if;
    return pols;
end intrinsic;
```

Conjugacy class invariants

Proof required!

We shall see that elements $g_1, g_2 \in \mathrm{GU}(n,q)$ are conjugate in $\mathrm{GU}(n,q)$ if and only if they are conjugate in $\mathrm{GL}(n,q^2)$ and the conjugacy classes in $\mathrm{GU}(n,q)$ are parametrised by sequences $[\ \langle\, f, \lambda \,\rangle \mid f \in \widetilde{\Phi}, \lambda \in \mathcal{P}\ ]$, where $\widetilde{\Phi}$ is the set of $\sim$irreducible polynomials.

The class invariants can be constructed in several steps. Firstly, choose a partition $\nu = [n_1, n_2, \ldots, n_k]$ of d. If ν has m parts of size n, choose m polynomials of degree n (with repetition) represented as a list ξ of pairs, where $\langle\, f, r \,\rangle$ indicates that the polynomial f of degree n has been chosen r times.

Secondly, refine ξ by replacing each pair $\langle\, f, r \,\rangle$ by $\langle\, f, \lambda \,\rangle$, where λ is a partition of r. This refinement step is carried out by the following function.

```
refine := function(ξ)
    Λ := [{@ @}];
    for η in ξ do
        Γ := [];
        f, r := Explode(η);
        for λ in Partitions(r) do
            β := convert(λ);
            for π in Λ do Append(~Γ, Include(π, <f, β> )); end for;
        end for;
        Λ := Γ;
    end for;
```

```
    return Λ;
end function;

intrinsic InternalClassInvariantsGU( d :: RngIntElt, q :: RngIntElt : Subset := "All")
    -> SeqEnum
{The sequence of conjugacy class invariants for the general
 unitary group GU(d,q)}
    if Subset eq "Unipotent" then
        t := PolynomialRing(GF(q^2)).1;
        return [ {@ <t - 1, convert(part)> @} : part in Partitions(d)];
    end if;
    pols := [TildeIrreduciblePolynomials(q, k) : k in [1..d]];
    polsz := [{ 1..#pols[k] } : k in [1..d]];
    ptnz := [convert(λ) : λ in Partitions(d)];
    inv := [];
    for δ in ptnz do
        prev := [{@ @}];
        for term in δ do
            ss := [];
            n, m := Explode(term);
            pp := pols[n];
            for S in Multisets(polsz[n], m) do
                if Subset eq "Semisimple" then
                    Ξ := [{@ @}];
                    for i -> r in S do
                        Ξ := [ Include(π, <pp[i], [<1, r>]>) : π in Ξ];
                    end for;
                else
                    ξ := [ < pp[i], r > : i -> r in S ];
                    Ξ := refine(ξ);
                end if;
                for stub in prev do
                    for π in Ξ do Append(~ss, stub join π); end for;
                end for;
            end for;
            prev := ss;
        end for;
        inv cat:= ss;
    end for;
    return inv;
end intrinsic;
```

Given the sequence *pif* of primary invariant factors $\langle f, m \rangle$ of a unitary matrix, for each polynomial f combine the multiplicities m into a partition and replace f by $f\tilde{f}$ if $f \neq \tilde{f}$.

```
primaryTildeFactors := function(pif)
    P := Parent(pif[1][1]);
    pols := [P| ];
```

```
    parts := [];
    duals := [P| ];
    j := 1;
    for i := 1 to #pif do
      f := pif[i][1]; ndx := pif[i][2];
      h := TildeDualPolynomial(f);
      if f eq h then
        if j eq 1 or pols[j−1] ne f then
          pols[j] := f;
          parts[j] := [];
          j +:= 1;
        end if;
        Append(∼parts[j−1], ndx);
      elif f notin duals then    // skip if in duals
        if IsEmpty(duals) or h ne duals[#duals] then
          Append(∼duals, h);
          pols[j] := h∗f;
          parts[j] := [];
          j +:= 1;
        end if;
        Append(∼parts[j−1], ndx);
      end if;
    end for;
    return pols, parts;
  end function;
```

3 An orthogonal decomposition

In this section we show that for $g \in \mathrm{GU}(n,q) = \mathrm{GU}(V)$ and the corresponding function $\mu : \Phi \to \mathcal{P}$, the direct sum decomposition (2.1)

$$V_g = \bigoplus_{f\in\Phi,i} k[t]/(f)^{\mu_i(f)} \tag{3.1}$$

can be converted to an orthogonal decomposition and the calculation of the conjugacy class of g can be reduced to studying the restriction of g to each component.

Definition 3.1. For each irreducible polynomial $f(t)$, the f-primary component of (3.1) is

$$V_{(f)} = \bigoplus_{i\geq 1} k[t]/(f)^{\mu_i(f)} = \{\, v \mid vf(g)^i = 0 \text{ for sufficiently large } i \,\}.$$

Lemma 3.2. *$V_{(f)}$ is orthogonal to $V_{(h)}$ unless $h(t) = \tilde{f}(t)$.*

Proof. Choose i large enough so that $uf(g)^i = 0$ for all $u \in V_{(f)}$. Then for all $u \in V_{(f)}$ and $v \in V$

$$\beta(u, v\overline{f}(g^{-1})^i) = \beta(uf(g)^i, v) = 0,$$

whence $V_{(f)}$ is orthogonal to $V\tilde{f}(g)^i$.

If $\tilde{f}(t) \neq h(t)$, then by irreducibility there are polynomials $r(t)$ and $s(t)$ such that $1 = r(t)h(t)^i + s(t)\tilde{f}(t)$. It follows that for large i and $v \in V_{(h)}$ we have $v = vs(g)\tilde{f}(g)$ and therefore the map

$$V_{(h)} \to V_{(h)} : v \mapsto v\tilde{f}(g)$$

is a bijection. Hence $V_{(f)}$ is orthogonal to $V_{(h)}$. □

Corollary 3.3. *$V_g = \perp_f \widetilde{V}_{(f)}$, where f ranges over all $\sim$irreducible polynomials and where*

$$\widetilde{V}_{(f)} = \begin{cases} V_{(h)} \oplus V_{(\tilde{h})} & f = h\tilde{h} \text{ and } h \neq \tilde{h}; \\ V_{(f)} & f = \tilde{f} \text{ is irreducible.} \end{cases}$$

Corollary 3.4. *If $h(t)$ is irreducible but not $\sim$symmetric, then $V_{(h)}$ and $V_{(\tilde{h})}$ are totally isotropic and $V_{(h)} \oplus V_{(\tilde{h})}$ is non-degenerate.*

Proof. It follows from the lemma that $V_{(h)}$ and $V_{(\tilde{h})}$ are totally isotropic and from the previous corollary $V_{(h)} \oplus V_{(\tilde{h})}$ is non-degenerate. □

Lemma 3.5. *If f is $\sim$symmetric, $V_{(f)}$ splits as an orthogonal sum $V_{(f)} = V^1 \perp V^2 \perp \cdots \perp V^r$, where each V^i is annihilated by $f(g)^i$ and is free as a module over $k[t]/(f^i)$.*

Proof. (Milnor [3]) The primary rational decomposition of $V_{(f)}$ is $V_{(f)} = W_1 \oplus W_2 \oplus \cdots \oplus W_r$ with W_i free as a $k[t]/(f^i)$-module but where the decomposition may not be orthogonal. Suppose that $W_r \cap W_r^\perp \neq 0$. Since $W_r \cap W_r^\perp$ is g-invariant we may choose $u \in W_r \cap W_r^\perp$ such that $u \neq 0$ and $uf(g) = 0$. But then $u = vf(g)^{r-1}$ for some $v \in W_r$. For $i < r$ and $w \in W_i$ we have

$$\beta(u, w) = \beta(vf(g)^{r-1}, w) = \beta(v, w\overline{f}(g^{-1})^{r-1}) = 0$$

because $f = \tilde{f}$ and $i < r$. Thus u is in the radical of β, which is a contradiction. It follows that $V_{(f)} = W_r^\perp \perp W_r$ and the proof is complete by induction. □

The polynomials f such that $\mu(f)$ is non-trivial are divisors of the minimal polynomial of g, which is $\sim$symmetric. Therefore we may restrict μ to the set $\widetilde{\Phi}$ of $\sim$irreducible polynomials.

Represent μ as a sequence $[\ \langle f, \lambda \rangle \mid f \in \widetilde{\Phi}, \lambda = \mu(f) \neq \varnothing\ ]$, where $\varnothing$ denotes the trivial partition.

```
intrinsic InternalConjugacyInvariantGU( g :: GrpMatElt ) -> SeqEnum
{The conjugacy class invariant of the unitary matrix A}
  F := BaseRing(g);
  std, _ := StandardHermitianForm(Nrows(g), F);
  e := Degree(F) div 2;
  require g*std*FrobeniusImage(Transpose(g), e) eq std :
    "matrix is not in the standard unitary group";
  pif := PrimaryInvariantFactors(g);
  pols, parts := primaryTildeFactors(pif);
  return {@ <pols[i], convert(parts[i])> : i in [1..#pols] @};
end intrinsic;
```

4 Unitary companion matrices

Given $g \in \mathrm{GU}(V)$, suppose $V = V_{(h)} \oplus V_{(\tilde{h})}$ where $h(t) \neq \tilde{h}(t)$ and $h(t)$ is irreducible. Choose a basis $v_1, v_2, \ldots, v_r$ for $V_{(h)}$ and a basis $w_1, w_2, \ldots, w_r$ for $V_{(\tilde{h})}$ such that $\beta(v_i, w_{r-j+1}) = \delta_{ij}$. The matrices of β and g with respect to this basis of V are

$$\begin{pmatrix} 0 & \Lambda \\ \Lambda & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} A & 0 \\ 0 & \Lambda \overline{A}^{-\mathrm{tr}} \Lambda \end{pmatrix}.$$

The minimal polynomial of A is $h(t)^s$ for some s and the minimal polynomial of $\overline{A}^{-1}$ is $\tilde{h}(t)^s$. If $\mu(h) = (\mu_1, \mu_2, \ldots)$ is the partition determined by A (in the general linear group), the conjugacy class of g is completely determined by the pair $\langle f, \mu(h) \rangle$, where $f(t) = h(t)\tilde{h}(t)$. Note that $\Lambda^{-1} = \Lambda^{\mathrm{tr}} = \Lambda$.

```
splitCompanion := function(h)
  d := Degree(h);
  A := CompanionMatrix(h);
  F := BaseRing(h);
  Λ, _ := StandardHermitianForm(d, F);
  e := Degree(F) div 2;
  return GU(2*d, F) ! DiagonalJoin(A, Λ*FrobeniusImage(Transpose(A^-1), e)*Λ);
end function;
```

Next suppose that V is a cyclic g-module and that the characteristic polynomial $h(t)$ of g is a power of a $\sim$symmetric irreducible polynomial.

Even degree

If the degree of $h(t)$ is $2d$, then $h(t)$ can be written as

$$h(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_{d-1} t^{d-1} + a_0 t^d (\overline{a}_d + \overline{a}_{d-1} t + \overline{a}_{d-2} t^2 + \cdots + \overline{a}_1 t^{d-1} + \overline{a}_0 t^d)$$

where $a_d = a_0 \overline{a}_d$ and $a_0 \overline{a}_0 = 1$.

Let v be a generator of the g-module V. The matrix of g with respect to the basis v, vg, $\ldots$, vg^{2d-1} is the 'standard' companion matrix

$$C_h = \left(\begin{array}{cccc|cccccc} 0 & 1 & & & & & & & & \\ & 0 & \ddots & & & & & & & \\ & & \ddots & 1 & & & & & & \\ & & & 0 & 1 & & & & & \\ \hline & & & & 0 & 1 & & & & \\ & & & & & 0 & 1 & & & \\ & & & & & & \ddots & \ddots & & \\ & & & & & & & 0 & 1 & \\ -a_0 & -a_1 & \cdots & -a_{d-1} & -a_0\overline{a}_d & -a_0\overline{a}_{d-1} & \cdots & -a_0\overline{a}_2 & -a_0\overline{a}_1 \end{array}\right).$$

Now set $J = \begin{pmatrix} 0 & \overline{P}^{\mathrm{tr}} \\ P & 0 \end{pmatrix}$ where P is the $d \times d$ upper triangular matrix

$$\begin{pmatrix} b_0 & b_1 & b_2 & \cdots & b_{d-1} \\ & b_0 & b_1 & \cdots & b_{d-2} \\ & & \ddots & \ddots & \vdots \\ & & & b_0 & b_1 \\ & & & & b_0 \end{pmatrix},$$

with b_0 chosen so that $b_0/\overline{b}_0 = -a_0$ and where $b_i = -a_i\overline{b}_0$ for $1 \le i < d$.

A direct calculation shows that $\overline{C}_h^{\mathrm{tr}} J C_h = J$ and so g preserves the hermitian form whose matrix is J^{-1}. Furthermore $J = Q\Lambda_{2d}\overline{Q}^{\mathrm{tr}}$, where

$$Q = \begin{pmatrix} I & \\ & P\Lambda_d \end{pmatrix} = Q^{\mathrm{tr}}.$$

Therefore $U_h = \overline{Q}C_h\overline{Q}^{-1}$ satisfies $U_h\Lambda_{2d}\overline{U}_h^{\mathrm{tr}} = \Lambda_{2d}$. Consequently U_h is the matrix of g with respect to the basis $u_1, u_2, \ldots, u_{2d}$ where $Q = (q_{ij})$ and $u_i = \sum_{j=1}^{2d} \overline{q}_{ij} v g^{j-1}$. Setting $v_i = u_{2d-i+1}$ for $1 \le i \le d$ the pairs (u_i, v_i) are mutually orthogonal hyperbolic pairs and we have

$$u_i = v g^{i-1} \quad \text{and} \quad v_i = \sum_{j=1}^{i} \overline{b}_{i-j} v g^{d+j-1} \quad 1 \le i \le d.$$

Writing U_h with respect to the basis $u_1, u_2, \ldots, u_d, v_d, v_{d-1}, \ldots, v_1$ and setting $b_d = -a_d\overline{b}_0$ we have

$$U_h = \left(\begin{array}{cccc|ccccc} 0 & 1 & & & & & & & \\ & 0 & \ddots & & & & & & \\ & & \ddots & 1 & & & & & \\ & & & 0 & & & & & 1/\overline{b}_0 \\ \hline b_0 & b_1 & \cdots & b_{d-1} & 0 & 0 & \cdots & 0 & -\overline{b}_d/\overline{b}_0 \\ & & & & 1 & 0 & \cdots & 0 & -\overline{b}_{d-1}/\overline{b}_0 \\ & & & & & \ddots & \ddots & & \vdots \\ & & & & & & 1 & 0 & -\overline{b}_2/\overline{b}_0 \\ & & & & & & & 1 & -\overline{b}_1/\overline{b}_0 \end{array}\right). \tag{4.1}$$

This is the *unitary companion matrix* of $h(t)$.

For future reference we note that

$$\begin{aligned} \det(tI - U_h) = &-(b_0/\overline{b}_0) - (b_1/\overline{b}_0)t - \cdots - (b_{d-1}/\overline{b}_0)t^{d-1} \\ &+ (\overline{b}_d/\overline{b}_0)t^d(\overline{b}_{d-1}/\overline{b}_0)t^{d+1} + \cdots + (\overline{b}_1/\overline{b}_0)t^{2d-1} + t^{2d}. \end{aligned} \tag{4.2}$$

```
evenCompanion := function( f )
  error if f ne TildeDualPolynomial(f), "polynomial must be ~symmetric";
  e := Degree( f ); assert IsEven(e);
```

```
    d := e div 2;
    F := BaseRing(f);
    flag, q := IsSquare(#F); assert flag;
```

Transform the coefficients.

```
    a0 := Coefficient(f, 0);
    flag, b0 := Hilbert90(−1/a0, q); assert flag;
    bbar0 := b0^q;
    a := Coefficients( f )[2..d+1];
    b := [−a[i]*bbar0 : i in [1..d] ];
    C := ZeroMatrix( BaseRing(f), e, e );
    for i := 1 to d−1 do
      C[i, i+1] := 1;
      C[d+1, i+1] := b[i];
      C[d+i+1, d+i] := 1;
      C[e−i+1, e] := −b[i]^q/bbar0;
    end for;
    C[d, e] := 1/bbar0;
    C[d+1, 1] := b0;
    C[d+1, e] := −b[d]^q/bbar0;
    return C;
  end function;
```

Odd degree

Suppose that the degree of $h(t)$ is $2d+1$. Then

$$h(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_d t^d + a_0 t^{d+1}(\overline{a}_d + \overline{a}_{d-1} t + \overline{a}_{d-2} t^2 + \cdots + \overline{a}_1 t^{d-1} + \overline{a}_0 t^d)$$

where $a_0 \overline{a}_0 = 1$.

Taking our cue from [2, p. 16] and extending (4.1) we set

$$U_h = \left(\begin{array}{ccccc|c|cccc} 0 & 1 & & & & & & & & \\ & 0 & \ddots & & & & & & & \\ & & \ddots & 1 & & & & & & \\ & & & 0 & & & & & & 1/\overline{b}_0 \\ \hline & & & & & u & & & & -\overline{c}/\overline{b}_0 \\ \hline b_0 & b_1 & \cdots & b_{d-1} & & cu & 0 & 0 & \cdots & 0 & -\overline{b}_d/\overline{b}_0 \\ & & & & & & 1 & 0 & \cdots & 0 & -\overline{b}_{d-1}/\overline{b}_0 \\ & & & & & & & \ddots & \ddots & & \vdots \\ & & & & & & & & 1 & 0 & -\overline{b}_2/\overline{b}_0 \\ & & & & & & & & & 1 & -\overline{b}_1/\overline{b}_0 \end{array}\right).$$

Choose the values of $c, u, b_0, \ldots, b_d$ so that U_h preserves the standard hermitian form and satisfies $\det(tI - U_h) = h(t)$.

A direct calculation shows that $U_h \Lambda \overline{U}_h^{\mathrm{tr}} = \Lambda$ if and only if $u\overline{u} = 1$ and $c\overline{c} = b_d + \overline{b}_d$. Furthermore,

$$\det(tI - U_h) = (t-u)\,p(t) + (c\overline{c}u/\overline{b}_0)t^d,$$

where $p(t)$ is the polynomial defined in (4.2). Equating coefficients with $h(t)$ leads to the equations

$$\begin{aligned}
a_0\overline{b}_0 &= ub_0 \\
a_i\overline{b}_0 &= ub_i - b_{i-1} && \text{for } 1 \le i < d \\
a_d\overline{b}_0 &= -u\overline{b}_d - b_{d-1} + c\overline{c}u \\
a_0\overline{a}_{d+1-i}\overline{b}_0 &= -u\overline{b}_{d-i} + \overline{b}_{d+1-i} && \text{for } 1 \le i \le d
\end{aligned}$$

which simplify to

$$\begin{aligned}
a_0\overline{b}_0 &= ub_0 \\
a_i\overline{b}_0 &= ub_i - b_{i-1} && \text{for } 1 \le i \le d\,.
\end{aligned}$$

Therefore

$$b_i = \overline{b}_0 \sum_{j=0}^{i} a_j/u^{i-j+1} \quad \text{for } 1 \le i \le d\,.$$

The space V is a cyclic U_h-module if and only if $c \neq 0$. Therefore, in order to ensure that V is cyclic we choose $u = 1$ if $f(1) \neq 0$ and otherwise choose $u = \overline{\zeta}/\zeta$, where ζ is a primitive element in GF(q^2).

oddCompanion := **function**(*f*)

The polynomial f should be a power of a $\sim$symmetric irreducible but we don't bother to check this.

```
error if f ne TildeDualPolynomial(f), "polynomial must be ~symmetric";
e := Degree( f ); assert IsOdd(e);
F := BaseRing(f);
flag, q := IsSquare(#F); assert flag;

a0 := Coefficient(f, 0);
if e eq 1 then return Matrix(F, 1, 1, [-a0]); end if;
d := e div 2;
a := Coefficients( f );
ζ := PrimitiveElement(F);
F0 := sub<F | ζ^(q+1) >;
```

When f is a power of $t-1$ choose $u \neq 1$.

```
if Evaluate(f, 1) eq 0 then
   u := ζ^(q-1);
   flag, b0 := Hilbert90(u/a0, q); assert flag;
   bbar0 := b0^q;
   b := [ &+[ bbar0*a[j]/u^(i-j+2) : j in [1..i+1]] : i in [1..d] ];
else
```

```
    u := 1;
    flag, b0 := HILBERT90(1/a0, q); assert flag;
    bbar0 := b0^q;
    b := [ &+[ bbar0*a[j] : j in [1..i+1]] : i in [1..d] ];
  end if;
  flag, c := NORMEQUATION(F, F0 ! (b[d] + b[d]^q) : DETERMINISTIC); assert flag;
  C := ZEROMATRIX( BASERING(f), e, e );
  for i := 1 to d-1 do
    C[i,i+1] := 1;
    C[d+2,i+1] := b[i];
    C[d+i+2,d+i+1] := 1;
    C[e-i+1,e] := -b[i]^q/bbar0;
  end for;
  C[d,e] := 1/bbar0;
  C[d+1,d+1] := u;
  C[d+1,e] := -c^q/bbar0;
  C[d+2,d+1] := c*u;
  C[d+2,1] := b0;
  C[d+2,e] := -b[d]^q/bbar0;
  return C;
end function;
```

5 Conjugacy classes in unitary groups

Unitary direct sums

If $A \in \mathrm{GU}(2m,q)$ and $B \in \mathrm{GU}(n,q)$ we may write A as the block matrix

$$A = \begin{pmatrix} P & Q \\ R & S \end{pmatrix}$$

and then the 'central join'

$$A \circ B = \begin{pmatrix} P & 0 & Q \\ 0 & B & 0 \\ R & 0 & S \end{pmatrix}$$

belongs to $\mathrm{GU}(2m+n,q)$.

The code for *centralJoin* is in the file `common.m` but we record it here for completeness.

```
centralJoin := function( A, B )
  d := NROWS(A);
  if d eq 0 then return B; end if;
  e := NROWS(B);
  if e eq 0 then return A; end if;
  assert ISEVEN(d);
  m := d div 2;
  X := ZEROMATRIX(BASERING(A), d+e, d+e);
```

```
    InsertBlock(~X, SubMatrix(A, 1,1, m,m), 1,1);
    InsertBlock(~X, SubMatrix(A, 1,m+1, m,m), 1,m+e+1);
    InsertBlock(~X, SubMatrix(A, m+1,1, m,m), m+e+1,1);
    InsertBlock(~X, SubMatrix(A, m+1,m+1, m,m), m+e+1,m+e+1);
    InsertBlock(~X, B, m+1,m+1);
    return X;
end function;
```

In order to form a unitary join of an $m \times m$ matrix and an $n \times n$ matrix with both m and n odd we first find the permutation that converts

$$\begin{pmatrix} \Lambda_m & 0 \\ 0 & \Lambda_n \end{pmatrix} \quad \text{to} \quad \begin{pmatrix} & & & \Lambda_{m-1} \\ & & 1 & 0 \\ & & 0 & 1 \\ \Lambda_{n-1} & & & \end{pmatrix}$$

```
antidiagPerm := function(m1, m2)
    X := [ i : i in [1..m1] ] cat [ 2*m1+1+i : i in [1..m2] ] cat
         [ m1+1, 2*m1+m2+2 ] cat
         [ 2*m1+m2+2+i : i in [1..m2] ] cat [ m1+1+i: i in [1..m1] ];
    return Sym(2*m1+2*m2+2) ! X;
end function;
```

Now observe that if $n\bar{n} = -1$ and $t + \bar{t} = 1$, then

$$\begin{pmatrix} 1 & n \\ \bar{t} & -nt \end{pmatrix} \begin{pmatrix} 1 & t \\ \bar{n} & -\bar{n}\bar{t} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

```
uTransform := function( F, mA, mB )
    flag, q := IsSquare(#F); assert flag;
    ζ := PrimitiveElement(F);
    t := ζ/(ζ + ζ^q);
    e := IsOdd(q) select (q-1) div 2 else q-1;
    n := ζ^e;
    M := Matrix(F,2,2, [1,n, 1-t,-n*t]);
    Z := IdentityMatrix(F, mA+mB);
    return DiagonalJoin(<Z, M, Z>);
end function;

unitaryJoin := function(A, B)
    nA := Nrows(A); nB := Nrows(B);
    mA := nA div 2; mB := nB div 2;
```

mA and *mB* are Witt indices.

```
    if nA eq 0 then return B; end if;
    if nB eq 0 then return A; end if;
    if nA eq 2*mA then return centralJoin(A, B); end if;
    if nB eq 2*mB then return centralJoin(B, A); end if;
    F := BaseRing(A);
```

```
    π := antidiagPerm(mA, mB);
    C0 := DiagonalJoin(A, B);
    C := Matrix(F, n, n, [C0[i^π, j^π] : i, j in [1..n]]) where n is nA+nB;
    X := uTransform(F, mA, mB);
    return X*C*X^-1;
  end function;
```

With *unitaryJoin* at our disposal we can construct the matrix of the restriction of g to each primary component. In the following function f is a $\sim$symmetric *reducible* polynomial and μ is a partition.

```
  splitComponent := function( f, plist )
    factors := Factorisation(f);
    h := factors[1][1];
    assert f eq h*factors[2][1];
    X := ZeroMatrix( BaseRing(f), 0, 0 );
    for μ in plist do
      e, m := Explode(μ);
      for i := 1 to m do
        X := centralJoin(X, splitCompanion(h^e));
      end for;
    end for;
    return X;
  end function;
```

Now deal with a $\sim$symmetric *irreducible* polynomial f and a partition μ.

```
  nonsplitComponent := function(f, plist)
    assert IsIrreducible(f);
    d := Degree(f);
    X := ZeroMatrix( BaseRing(f), 0, 0 );
    for μ in plist do
      e, m := Explode(μ);
      for i := 1 to m do
        X := IsEven(d*e) select unitaryJoin(X, evenCompanion(f^e))
                         else unitaryJoin(X, oddCompanion(f^e));
      end for;
    end for;
    return X;
  end function;

  intrinsic InternalRepMatrixGU( inv :: SetIndx[Tup] ) → GrpMatElt
  { A representative matrix of the unitary conjugacy class with
    invariant inv }
    F := BaseRing(Parent(inv[1][1]));
    X := ZeroMatrix( F, 0, 0 );
    for polpart in inv do
      f, plist := Explode(polpart);
      X := IsIrreducible(f) select unitaryJoin(X, nonsplitComponent(f, plist))
                            else unitaryJoin(X, splitComponent(f, plist));
```

```
    end for;
    return GeneralUnitaryGroup(Nrows(X), F) ! X;
end intrinsic;
```

6 Centraliser orders

The centraliser orders of elements of the unitary group can be computed using the functions $A(\varphi^\mu)$ and $B(\varphi)$ from Wall [4]. Here f is a polynomial and $\langle e, m \rangle$ is a term from the partition list.

```
A_fn := func< f, m, Q | IsIrreducible(f) select OrderGU(m, Q) else OrderGL(m, Q) >;

κ := function(plist, d)
    val := 0;
    for μ in plist do
        e, m := Explode(μ);
        val +:= (e-1)*m^2;
    end for;
    for i := 1 to #plist-1 do
        e := plist[i][1];
        m := plist[i][2];
        val +:= &+[ 2*e*m*plist[j][2] : j in [i+1..#plist]];
    end for;
    return val*d;
end function;
```

Here *plist* has the form $[\ \ldots\ ,\ \langle e, m \rangle,\ \ldots\ ]$.

```
B_fn := function(pol_part)
    f, plist := Explode(pol_part);
    flag, q := IsSquare(#BaseRing(f)); assert flag;
    d := Degree(f);
    return q^κ(plist, d) * &*[A_fn(f, μ[2], q^d) : μ in plist];
end function;
```

The order of the centraliser of an element in the unitary group whose conjugacy invariant is μ.

```
centraliserOrderGU := func< inv | &*[ B_fn(pol_part) : pol_part in inv ] >;
```

The conjugacy classes of $\mathrm{GU}(n, q)$

```
classesGU := function(d, q)
    ord := OrderGU(d, q);
    L := InternalClassInvariantsGU(d, q);
    cc := [car<Integers(), Integers(), GU(d, q)>|
        < Order(M), ord div centraliserOrderGU(μ), M > : μ in L | true
            where M is InternalRepMatrixGU(μ) ];
    ParallelSort(~cc, ~L);
```

```
    return cc, L;
end function;
```

7 Test code for general unitary groups

```
AttachSpec("Conjugacy.spec");

testDual := procedure()
    print "Test TildeDualPolynomial";
    for q in [3,4,5] do
        print "q =", q;
        F := GF(q^2);
        σ := iso< F -> F | x :-> x^q >;
        P<t> := PolynomialRing(F);
        for i := 1 to 5 do
            lst := [ Random(F) : i in [1..6]];
            if lst[1] ne 0 and lst[6] ne 0 then
                assert Dual(P ! lst : t := F ! 1, σ := σ) eq TildeDualPolynomial(P ! lst);
            end if;
        end for;
    end for;
    print "Passed\n";
end procedure;

testDual();

test0gu := procedure(n, q)
    printf "Test 0: compare with ";
    printf "Classes(GU(%o,%o) : Al := \"Lifting\")\n", n, q;
    G := GU(n, q);
    time reps := Classes(G);
    delete G;
    G := GU(n, q);
    time cc := Classes(G : Al := "Lifting");
    assert #Set(reps) eq #cc;
    ndx := [];
    for X in reps do
        assert exists(i){ i : i in [1..#cc] | IsConjugate(G, X[3], cc[i][3]) };
        Append(~ndx, i);
    end for;
    assert #reps eq #SequenceToSet(ndx);
    print "Passed\n";
end procedure;

test0gu(3,2);
test0gu(3,3);
test0gu(3,4);
```

```
test0gu(4,3);

test1gu := procedure(n,q)
  printf "Test 1: class sizes for GU(%o,%o)\n", n,q;
  f := NCLASSES("unitary",n);
  assert #INTERNALCLASSINVARIANTSGU(n,q) eq EVALUATE(f,q);
  print "Passed\n";
end procedure;

test1gu(5,4);
test1gu(6,4);
test1gu(5,5);
test1gu(6,5);

test2gu := procedure(n,q)
  printf "Test 2: conjugacy invariants for GU(%o,%o)\n", n,q;
  for μ in INTERNALCLASSINVARIANTSGU(n,q) do
    g := INTERNALREPMATRIXGU(μ);
    c := INTERNALCONJUGACYINVARIANTGU(g);
    assert μ eq c;
  end for;
  print "Passed\n";
end procedure;

test2gu(4,3);
test2gu(5,4);
test2gu(5,5);
test2gu(6,5);

test3gu := procedure(n,r)
  printf "Test 3: centraliser orders for GU(%o,%o)\n", n,r;
  S := GU(n,r);
  for μ in INTERNALCLASSINVARIANTSGU(n,r) do
    g := INTERNALREPMATRIXGU(μ);
    assert #CENTRALISER(S,g) eq centraliserOrderGU(μ);
  end for;
  print "Passed\n";
end procedure;

test3gu(4,3);
test3gu(4,4);
test3gu(5,3);
```

Conjugacy invariants (randomised)

```
test4gu := procedure(n,r)
  printf "Randomised conjugacy invariants for GU(%o,%o)\n", n,r;
  G := GU(n,r);
  for μ in INTERNALCLASSINVARIANTSGU(n,r) do
```

```
    g := InternalRepMatrixGU(μ);
    h := Random(G);
    c := ConjugacyInvariantGU(g^h);
    assert μ eq c;
  end for;
  print "Passed\n";
end procedure;

test4gu(4,5);
test4gu(5,5);
test4gu(6,4);
```

8 Extended special unitary groups

The *extended* special unitary group $\mathrm{ExtSU}(n,q,m)$ of index m is the unique group G such that $\mathrm{SU}(n,q) \subseteq G \subseteq \mathrm{GU}(n,q)$ and $m = |G : \mathrm{SU}(n,q)|$.

Return a diagonal matrix whose image in $\mathrm{GU}(n,q)/\mathrm{SU}(n,q)$ has order $(q+1)/r$

```
diagMatrixGU := function(n, q, r)
  F := GF(q^2);
  ξ := PrimitiveElement(F);
  A := IdentityMatrix(F, n);
  A[1,1] := ξ^r;
  A[n,n] := ξ^(-q*r);
  return A;
end function;
```

Construct an extended special unitary group

```
intrinsic ExtendedSU( n :: RngIntElt, q :: RngIntElt, m :: RngIntElt)
    -> GrpMat
{The subgroup of GU(n,q) that contains SU(n,q) as a subgroup
 of index m}
  require m gt 0 : "the index should be positive";
  if m eq 1 then G := SU(n,q);
  elif m eq q + 1 then G := GU(n,q);
  else
    divides, r := IsDivisibleBy(q + 1, m);
    require divides : "the index should divide q + 1";
    G := sub< GU(n,q) | SU(n,q), diagMatrixGU(n,q,r) >;
    G`Order := OrderSU(n,q) * m;
  end if;
  return G;
end intrinsic;
```

```
intrinsic IndexOfSU( G :: GrpMat ) → RngIntElt
{The index of the special unitary group in G}
  F := BaseRing(G);
  require IsA(Type(F), FldFin) : "the base field should be finite";
  msg := "G should contain the standard special unitary
    group and be a subgroup of the conformal unitary group";
  count := 0;
  repeat    // at most 4 times
    flag := RecognizeClassical(G);
    count +:= 1;
  until flag or count gt 3;
  require flag and ClassicalType(G) eq "unitary" : msg;
  d := Degree(G);
  flag, q := IsSquare(#F); assert flag;
  ord := assigned G`Order select G`Order else LMGOrder(G);
  return ord div OrderSU(d, q);
end intrinsic;
```

The determinant of the matrix representing the conjugacy class invariant μ.

```
weight := func< ξ | &+[π[1]*π[2] : π in ξ ] >;

classdet := func< μ | &*[ ((-1)^Degree(f)*Coefficient(f, 0))^weight(tpl[2])
  where f is tpl[1] : tpl in μ ]>;
```

The order of the image (under the determinant map) of the centralizer of the matrix representing μ.

```
CentOrderModSU := func< q, μ | (q+1) div
  GCD(Append(Flat([[em[1] : em in pt[2]] : pt in μ]), q+1)) >;
```

Extended conjugacy invariants

Suppose that m divides $q+1$, $S = \mathrm{SU}(n,q)$, $G = \mathrm{ExtSU}(n,q,m)$ and $U = \mathrm{GU}(n,q)$. Then $G \trianglelefteq U$ and for $g \in G$ the conjugacy class $\mathrm{ccl}_U(g)$ of g in U splits into G-classes $\mathrm{ccl}_G(g^u)$ for suitable $u \in U$. If $u \in C_U(g)G$, then $u = ch$ for some $c \in C_U(g)$ and $h \in G$ whence $g^u = g^h$ is conjugate to g and the converse is clear. Thus the conjugacy classes of G contained in $\mathrm{ccl}_U(g)$ are represented by the elements g^u, where u runs through a transversal of $C_U(g)G$ in U. A similar calculation can be found in Wall [5].

If ζ is a primitive element of $\mathrm{GF}(q^2)$, the image of the determinant map $\det : U \to \mathrm{GF}(q^2)^\times$ is the cyclic subgroup of order $q+1$ generated by $\xi = \overline{\zeta}/\zeta$. The determinant map identifies U/S with the cyclic group $\langle \xi \rangle$. Thus $\{ \xi^i \mid 0 \le i < r \}$ is transversal to $C_U(g)G/G$ in U/G, where $r = |U/C_U(g)G|$.

If μ is the conjugacy invariant of g in U we label the conjugacy class of g^u in G with $\langle \mu, i \rangle$, where ξ^i is the determinant of u.

```
intrinsic InternalClassInvariantsExtSU( d :: RngIntElt, q :: RngIntElt, m :: RngIntElt )
    → SeqEnum
{The sequence of conjugacy class invariants for the subgroup of
 GU(d,q) containing SU(d,q) as a subgroup of index m}
```

```
    error if not IsDivisibleBy( q+1, m ), "m must divide q+1";
    cGU := InternalClassInvariantsGU( d, q );
    if m eq q + 1 then return [ <μ,0> : μ in cGU ]; end if;
    conj_inv := [];
    for μ in cGU do
      if classdet(μ)^m eq 1 then
        r := (q+1) div LCM(m, CentOrderModSU(q,μ));
        for i := 0 to r−1 do
          Append( ~conj_inv, <μ,i> );
        end for;
      end if;
    end for;
    return conj_inv;
  end intrinsic;
```

The matrices

$$\begin{pmatrix} \overline{\zeta}^i & & \\ & I & \\ & & \zeta^{-i} \end{pmatrix} \quad \text{for } 0 \le i \le q$$

are transversal to S in U.

The sequence of conjugacy class invariants for $\mathrm{SU}(d,q)$.

```
  intrinsic InternalClassInvariantsSU( d :: RngIntElt, q :: RngIntElt : Subset := "All" )
    → SeqEnum
  {The sequence of conjugacy class invariants for the subgroup of
   GU(d,q) containing SU(d,q) as a subgroup of index m}
    if Subset eq "Unipotent" then
      t := PolynomialRing(GF(q^2)).1;
      return [ {@ <t − 1, convert(part)> @} : part in Partitions(d)];
    end if;
    cGU := InternalClassInvariantsGU( d, q : Subset := Subset);
    conj_inv := [];
    for μ in cGU do
      if classdet(μ) eq 1 then
        r := (q+1) div CentOrderModSU(q,μ);
        for i := 0 to r−1 do
          Append( ~conj_inv, <μ,i> );
        end for;
      end if;
    end for;
    return conj_inv;
  end intrinsic;

  intrinsic InternalRepMatrixExtSU( inv :: Tup ) → GrpMatElt
  {Given an extended conjugacy invariant, return a matrix which
   represents the conjugacy class in ExtSU}
    μ, a := Explode(inv);
    M := InternalRepMatrixGU(μ);
```

```
    F := BaseRing(M);
    flag, q := IsSquare(#F); assert flag;
    n := Nrows(M);
    if a ne 0 then
      η := PrimitiveElement(F)^a;
      D := IdentityMatrix(F, n);
      D[1,1] := η^q;
      D[n,n] := η^-1;
      R := Parent(M) ! (D^-1*M*D);
    else
      R := M;
    end if;
    return R;
  end intrinsic;
```

The conjugacy classes of ExtSU(d, q, m)

There is no need to coerce the representatives into the subgroup. This will be done when the classes are assigned to the group.

```
  classesExtSU := function(d, q, m)
    if m eq q+1 then return classesGU(d, q); end if;
    U := GL(d, q^2);
    ord := OrderGU(d, q);
    cl := [car<Integers(), Integers(), U>|];
    L := InternalClassInvariantsExtSU(d, q, m);
    for inv in L do
      μ := inv[1];
      M := InternalRepMatrixExtSU(inv);
      len := (ord*LCM(m, CentOrderModSU(q, μ))) div
             (centraliserOrderGU(μ)*(q+1));
      Append(~cl, < Order(M), len, M > );
    end for;
    ParallelSort(~cl, ~L);
    return cl, L;
  end function;
```

The following intrinsic is called by InternalClassesClassical which itself is called by the C code `matg/access.c/matg_ensure_classes`.

```
  intrinsic InternalClassesExtendedSU( G :: GrpMat ) → BoolElt
  {Internal function: attempt to assign the conjugacy classes
   of the extended special unitary group. Return true if
   successful}
    /*
      It is assumed that this function is called only when it is known
      that G is a finite unitary group.
    */
    F := BaseRing(G);
```

```
flag, q := IsSquare(#F); assert flag;
d := Degree(G);
e := Degree(F) div 2;
M, _ := StandardHermitianForm(d, F);
conformal := false;
if forall{g : g in Generators(G) | g*M*FrobeniusImage(Transpose(g), e) eq M} then
   X := IdentityMatrix(F, d);
   std := true;
else
   forms := InvariantSesquilinearForms(G);
   if IsEmpty(forms) then
      vprint Classes: "No invariant hermitian form";
      forms := SemiInvariantSesquilinearForms(G);
      if IsEmpty(forms) then
         vprint Classes: "No semi-invariant hermitian form";
         return false;
      else
         factors := forms[1, 1];
         F_0 := Universe(factors);
         U, phi := UnitGroup(F);
         U_0, psi := UnitGroup(F_0);
         D, p_1, p_2 := DirectSum(U_0, U);
         X := sub< D | [ p_1(factors[i]@@psi) + p_2(Determinant(G.i)@@phi) :
                                 i in [1..Ngens(G)] ] >;
         m := #X;
         if m ne #F - 1 then
            vprint Classes: "proper subgroup of CGU", d, q;
            return false;
         end if;
         J := forms[1, 2, 1];
         conformal := true;
      end if;
   else
      J := forms[1];
   end if;
   X := TransformForm(J, "unitary"); assert Type(X) ne BoolElt;
   std := false;
end if;
vprint Classes: "Standard:", std;
vprint Classes: "Conformal:", conformal;
m := LCM([Order(Determinant(g)) : g in Generators(G)]);
if conformal then
   cc := ConjugacyClasses(CGU(d, q));
   L := [];
else
   cc, L := classesExtSU(d, q, m);
end if;
```

```
    if not assigned G`ClassicalType then
      if conformal then
        G`ClassicalType := (m eq q*q - 1) select "CGU" else "AltSU";
      elif m eq q + 1 then
        G`ClassicalType := "GU";
      elif m eq 1 then
        G`ClassicalType := "SU";
      else
        G`ClassicalType := "ExtSU";
      end if;
    end if;
    if not std then
      cc := [ < t[1], t[2], X*t[3]*X^-1 > : t in cc ];
    end if;
    vprint Classes: "assigning unitary classes";
    G`Classes := cc;
    G`Labels_S := {@ x : x in L @};
    return true;
end intrinsic;
```

The following intrinsic is a variant of the old ClassRepresentativesGU.

```
intrinsic InternalUnitaryClasses(G :: GrpMat : Subset := "All")
    -> SeqEnum, SeqEnum
{Conjugacy class representatives and labels for the standard
 unitary or special unitary group. The parameter Subset is either
 "Unipotent", "Semisimple" or "All" (the default)}
    require Subset in {"Unipotent", "Semisimple", "All"}: "invalid Subset";
    F := BaseRing(G);
    d := Dimension(G);
    flag, q := IsSquare(#F);
    require flag: "G is not a standard unitary group";
    e := Degree(F) div 2;

    M, _ := StandardHermitianForm(d, F);
    require forall{g : g in Generators(G) | g*M*FrobeniusImage(Transpose(g), e) eq M}:
      "G is not a standard unitary group";
    m := LCM([Order(Determinant(g)) : g in Generators(G)]);
    if m eq 1 then
      L := InternalClassInvariantsSU(d, q : Subset := Subset);
      ord := OrderGU(d, q);
      cc := [car<Integers(), Integers(), G>|];
      for inv_ in L do
        if Subset eq "Unipotent" then
          mu := inv_;
          inv := < inv_, 0 >;
        else
          mu := inv_[1];
```

```
            inv := inv_;
        end if;
        M := G ! InternalRepMatrixExtSU(inv);
        len := (ord*LCM(m, CentOrderModSU(q, μ))) div
                (centraliserOrderGU(μ)*(q+1));
        Append(~cc, < Order(M), len, M > );
      end for;
      L := [ tagToNameSU(μ) : μ in L ];
    elif m eq q+1 then
      L := InternalClassInvariantsGU(d, q : Subset := Subset);
      ord := OrderGU(d, q);
      cc := [car<Integers(), Integers(), GU(d, q)>|
        < Order(M), ord div centraliserOrderGU(μ), M > : μ in L | true
          where M is InternalRepMatrixGU(μ) ];
      ParallelSort(~cc, ~L);
      L := [ tagToNameGU(μ) : μ in L ];
    else
      error "neither SU not GU";
    end if;
    return cc, {@ x : x in L @};
end intrinsic;
```

9 Test code for special unitary groups

```
AttachSpec("Conjugacy.spec");

test0su := procedure(n, q)
    printf "Test 0: compare with ";
    printf "Classes(SU(%o,%o) : Al := \"Lifting\")\n", n, q;
    G := SU(n, q);
    reps := Classes(G);
    delete G;
    G := SU(n, q);
    cc := Classes(G : Al := "Lifting");
    ndx := [];
    for X in reps do
      assert exists(i){ i : i in [1..#cc] | IsConjugate(G, X[3], cc[i][3]) };
      Append(~ndx, i);
    end for;
    assert #reps eq #SequenceToSet(ndx);
    print "Passed\n";
end procedure;

test0su(2, 3);
test0su(2, 5);
test0su(2, 7);
```

```
test0su(3,2);
test0su(3,3);
test0su(3,4);
test0su(4,3);

test1su := procedure(n,r)
   printf "Test 1: class sizes for SU(%o,%o)\n", n,r;
   f := #CLASSES(SU(n,r) : AL := "Lifting");
   assert #INTERNALCLASSINVARIANTSEXTSU(n,r,1) eq f;
   print "Passed\n";
end procedure;

test1su(3,2);
test1su(3,3);
test1su(4,3);
test1su(3,4);
test1su(4,4);

test2su := procedure(n,r,m)
   printf "Test 2: conjugacy invariants for ExtSU(%o,%o,%o)\n", n,r,m;
   for inv in INTERNALCLASSINVARIANTSEXTSU(n,r,m) do
      g := INTERNALREPMATRIXEXTSU(inv);
      c := INTERNALCONJUGACYINVARIANTGU(g);
      assert inv[1] eq c;
   end for;
   print "Passed\n";
end procedure;

test2su(3,2,1);
test2su(3,3,1);
test2su(3,3,2);
test2su(3,4,1);
test2su(3,8,3);
test2su(4,3,1);
test2su(4,3,2);
test2su(4,4,1);
test2su(4,4,5);
```

References

[1] V. Ennola. On the conjugacy classes of the finite unitary groups. *Ann. Acad. Sci. Fenn. Ser. A I No.*, 313:13, 1962.

[2] S. Haller and S. H. Murray. Computing conjugacy in finite classical groups 1: similarity in unitary groups. preprint, January 2009.

[3] J. Milnor. On isometries of inner product spaces. *Invent. Math.*, 8:83–97, 1969.

[4] G. E. Wall. On the conjugacy classes in the unitary, symplectic and orthogonal groups. *J. Aust. Math. Soc.*, 3:1–62, 1963.

[5] G. E. Wall. Conjugacy classes in projective and special linear groups. *Bull. Austral. Math. Soc.*, 22(3):339–364, 1980.

Revision history

2020-08-07 Revised the code for IndexOfSU so that it accepts a subgroup of the conformal unitary group.

2020-08-15 Use FrobeniusImage(Transpose(X), e) instead of ConjugateTranspose.

2020-09-07 Avoid calling TransformForm on a group.

2020-09-18 Fast version of ClassInvariantsGU.

2021-03-26 InternalUnitaryClasses.

2021-04-01 Removed all calls to ConjugateTranspose(d, σ) and replaced them by the faster FrobeniusImage(Transpose(g), e).

2021-04-04 Changed the partitions within a conjugacy invariant to multiplicity format and changed the type of the conjugacy invariant itself from SeqEnum to SetIndx.

2021-04-28 Changed the intrinsic ClassInvariantsGU to InternalClassInvariantsGU and also changed CentraliserOrderGU to a function *centraliserOrderGU*. Similarly for the special unitary groups.

2021-05-02 Removed ClassRepresentativesGU and ClassRepresentativesSU. Also changed RepresentativeMatrixGU and RepresentativeMatrixExtSU to InternalRepMatrixGU and InternalRepMatrixExtSU.

2021-05-06 Changed ConjugacyInvariantGU to InternalConjugacyInvariantGU.

2021-05-19 Extend the fast algorithm to conformal unitary groups.

2021-06-05 Correction to IndexOfSU.

2021-06-26 Minor changes to InternalClassesExtendedSU.