BruhatDecomposition

Computes the Bruhat decomposition of matrices in classical groups.

0.1

27 March 2020

Daniel Rademacher

Alice Niemeyer

Daniel Rademacher
Email: rademacher@art.rwth-aachen.de
Address: Daniel Rademacher
Chair of Algebra and Representation Theory
RWTH Aachen
Pontdriesch 10/16
52062 Aachen
Germany

Alice Niemeyer
Email: alice.niemeyer@mathb.rwth-aachen.de
Homepage: http://www.math.rwth-aachen.de/~Alice.Niemeyer/
Address: Alice Niemeyer
Chair of Algebra and Representation Theory
RWTH Aachen
Pontdriesch 10/16
52062 Aachen
Germany

Contents

Chapter 1

Foreword

Let G be one of the classical groups SL, Sp, SU or SO over a finite field of size q and dimension d. Let g be an element in G. We want to write $g = u_1 \cdot w \cdot u_2$ with u_1 and u_2 lower unitriangular matrices and w a monomial matrix.

This is already implemented for:

- Special linear group (SL) (see Chapter 3)
- Symplectic group (Sp) (see Chapter 5)
- Special unitary group (SU) (see Chapter 4)
- Special orthogonal group (SO) (see Chapter 2)

1.1 Main Function

1.1.1 BruhatDecomposition

▷ `BruhatDecomposition(g)` (function)

Returns: pgr (A SLP to compute u_1, u_2, p_{sign} and $diag$ and the matrices u_1, u_2, p_{sign} and $diag$ itself.)

Checks whether g is an element of one of the classical groups in their natural representation. If yes, the corresponding Bruhat decomposition of the group and the element g is calculated. Otherwise the function prints a warning.

Chapter 2

Special Orthogonal Group

This chapter deals with the special orthogonal group

2.1 Introduction and Quick Start of functions for SO

TODO

2.2 Functions for SO

2.2.1 FindPrimePowerDecomposition

▷ `FindPrimePowerDecomposition(n)` (function)

Returns: $[a,b]$ (a and b are natural numbers such that $n-1=2^a\cdot b$)

n: Natural number Computes two natural numbers a and b such that $n-1=2^a\cdot b$.

2.2.2 LGOStandardGensSO

▷ `LGOStandardGensSO(d, q, e)` (function)

▷ `__LGOStandardGensSOPlus(arg)` (function)

▷ `__LGOStandardGensSOCircle(arg)` (function)

▷ `__LGOStandardGensSOMinus(arg)` (function)

Returns: stdgens (the LGO standard-generators of SO(e,d,q))

d: the dimension of our matrices,

q: A prime power $q=p^f$, where $\mathbb{F}_q$ ist the field whereover the matrices are defined. q has to be odd

e: 1 for plus type, 0 for zero type, -1 for minus type This function computes the standard generators of SO as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in odd characteristic" Depending on e and p (notice $q=p^f$ with p prime), the functions `__LGOStandardGensSOPlus(d,q)`, `__LGOStandardGensSOCircle(d,q)` or `__LGOStandardGensSOMinus(d,q)` are called.

2.2.3 LGOStandardGensOmega

▷ `LGOStandardGensOmega(d, q, e)` (function)

▷ `__LGOStandardGensOmegaPlus(arg)` (function)

▷ `__LGOStandardGensOmegaPlusEvenChar(arg)` (function)
▷ `__LGOStandardGensOmegaCircle(arg)` (function)
▷ `__LGOStandardGensOmegaCircleEvenChar(arg)` (function)
▷ `__LGOStandardGensOmegaMinus(arg)` (function)
▷ `__LGOStandardGensOmegaMinusEvenChar(arg)` (function)

Returns: stdgens (the LGO standard-generators of $\Omega(e,d,q)$)

d: the dimension of our matrices,
q: A prime power $q = p^f$, where $\mathbb{F}_q$ ist the field whereover the matrices are defined.
e: 1 for plus type, 0 for zero type, -1 for minus type This function computes the standard generators of Ω as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in odd characteristic" and "Constructive Recognition of Classical Groups in even characteristic" Depending on e, the functions `__LGOStandardGensOmegaPlus(d,q)`, `__LGOStandardGensOmegaPlusEvenChar(d,q)`, `__LGOStandardGensOmegaCircle(d,q)`, `__LGOStandardGensOmegaCircleEvenChar(d,q)` `__LGOStandardGensOmegaMinus(d,q)` or `__LGOStandardGensOmegaMinusEvenChar(d,q)` are called.

2.2.4 MSO

▷ `MSO(d, q, e)` (function)

Returns: G (where $G = \mathrm{SO}(e,d,q)$)

d: the dimension of our matrices,
q: A prime power $q = p^f$, where $\mathbb{F}_q$ ist the field whereover the matrices are defined. q has to be odd
e: 1 for plus type, 0 for zero type, -1 for minus type
This function returns the special orthogonal group of type e. The generators of the group are the LGO standard generators and the size of the group is already stored as an attribute.

2.2.5 UnitriangularDecompositionSOPlus

▷ `UnitriangularDecompositionSOPlus(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators of $\mathrm{SO}^+(d,q)$
g: A matrix in $\mathrm{SO}^+(d,q)$
Computes the Unitriangular decomposition of the matrix g.

2.2.6 UnitriangularDecompositionSOCircle

▷ `UnitriangularDecompositionSOCircle(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators of $\mathrm{SO}^\circ(d,q)$
g: A matrix in $\mathrm{SO}^\circ(d,q)$
Computes the Unitriangular decomposition of the matrix g.

2.2.7 UnitriangularDecompositionSOMinus

▷ `UnitriangularDecompositionSOMinus(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices

of the Bruhat-Decomposition)
stdgens: The LGO standard-generators of $\mathrm{SO}^-(d,q)$
g: A matrix in $\mathrm{SO}^-(d,q)$
Computes the Unitriangular decomposition of the matrix g.

2.2.8 MonomialSLPSOPlus

▷ `MonomialSLPSOPlus(stdgens, mat, slp)` (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)
stdgens: The LGO standard-generators of $\mathrm{SO}^+(d,q)$
mat: A monomial matrix (ie w) in $\mathrm{SO}^+(d,q)$
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{SO}^+(d,q)$ into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

2.2.9 MonomialSLPSOCircle

▷ `MonomialSLPSOCircle(stdgens, mat, slp)` (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)
stdgens: The LGO standard-generators of $\mathrm{SO}^\circ(d,q)$
mat: A monomial matrix (ie w) in $\mathrm{SO}^\circ(d,q)$
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{SO}^\circ(d,q)$ into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

2.2.10 MonomialSLPSOMinus

▷ `MonomialSLPSOMinus(stdgens, mat, slp)` (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)
stdgens: The LGO standard-generators of $\mathrm{SO}^-(d,q)$
mat: A monomial matrix (ie w) in $\mathrm{SO}^-(d,q)$
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{SO}^-(d,q)$ into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

2.2.11 FindCorrectCycel

▷ `FindCorrectCycel(perm, j)` (function)

Returns: A permutation

perm: A list of cycles

j: A natural number

This is a help function for `MonomialSLPSOPlus`. Checks whether there is a cycle c in perm such that $j^c \neq j$. If there is such an cycle, the cycle is returned. Otherwise the identity permutation is returned.

2.2.12 TestPermutationProd

▷ `TestPermutationProd(op, np, l, n)` (function)

Returns: true or false

op: A list of cycle

np: A list of cycle

l: A list of natural numbers

n: A natural number

This is a help function for `MonomialSLPSOPlus`. This function checks whether the new permutation np destorys an already considered element of op. The already considered elements are stored in l.

2.2.13 TestPermutationProd2

▷ `TestPermutationProd2(op, np, tn, l, n)` (function)

Returns: true or false

op: A list of cycle

np: A list of cycle

tn: A natural number

l: A list of natural numbers

n: A natural number

This is a help function for `MonomialSLPSOPlus`. This function checks whether the probability to continue with np is higher than with op depending on the element tn.

2.2.14 MonomialMatrixToEasyForm

▷ `MonomialMatrixToEasyForm(M)` (function)

Returns: [list,perm] (list is a list of the non-zero elements of each column of M, perm is the permutation corresponding to M)

M: A monomial matrix

This is a help function for `MonomialSLPSOPlus` and `MonomialSLPSOCircle`. This function calcultes a list of size 2. The first entry is a list of the non-zero elements of each column of M. The second entry is a permutation which corresponds to M as a permutation matrix.

2.2.15 EasyFormToMonomialMatrix

▷ `EasyFormToMonomialMatrix(tupel, n, fld)` (function)

Returns: M (A monomial matrix)

$tupel$: A 2-tupel as in `MonomialMatrixToEasyForm`

n: A natural number

fld: A finite field
This is a help function for `MonomialSLPSOPlus` and `MonomialSLPSOCircle`. This function computes a monomial matrix M of size n over fld such that `MonomialMatrixToEasyForm`$(M) = tupel$.

2.2.16 MultiplicationOfEasyForm

▷ `MultiplicationOfEasyForm(`*tupel1, tupel2*`)` (function)

Returns: [list,perm] (list is a list of the non-zero elements of each column of M, perm is the permutation corresponding to M)

*tupel*1: A 2-tupel as in `MonomialMatrixToEasyForm`
*tupel*2: A 2-tupel as in `MonomialMatrixToEasyForm`
This is a help function for `MonomialSLPSOPlus` and `MonomialSLPSOCircle`. Let M_1 be a monomial matrix which corresponds to *tupel*1 and M_2 be a monomial matrix which corresponds to *tupel*2. This function computes a tupel [list,perm] such that for the corresponding monomial matrix M holds $M = M_1 \cdot M_2$.

2.2.17 DiagSLPSOPlus

▷ `DiagSLPSOPlus(`*stdgens, diag, slp*`)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators of $\mathrm{SO}^+(d,q)$
diag: A diagonal matrix (eg diag) in $\mathrm{SO}^+(d,q)$
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

2.2.18 DiagSLPSOCircle

▷ `DiagSLPSOCircle(`*stdgens, diag, slp*`)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators of $\mathrm{SO}^\circ(d,q)$
diag: A diagonal matrix (eg diag) in $\mathrm{SO}^\circ(d,q)$
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

2.2.19 DiagSLPSOMinus

▷ `DiagSLPSOMinus(`*stdgens, diag, slp*`)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators of $\mathrm{SO}^-(d,q)$
diag: A diagonal matrix (eg diag) in $\mathrm{SO}^-(d,q)$
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

2.2.20 BruhatDecompositionSO

▷ `BruhatDecompositionSO(`*`stdgens, g`*`)` (function)

Returns: pgr (A SLP to compute u_1, u_2, p_{sign} and $diag$ and the matrices u_1, u_2, p_{sign} and $diag$ itself.)

stdgens: The LGO standard-generators
g: A matrix in $\mathrm{SO}(e,d,q)$
Uses `UnitriangularDecompositionSOPlus()`, `MonomialSLPSOPlus()` and `DiagSLPSOPlus()` for the plus type, `UnitriangularDecompositionSOCircle()`, `MonomialSLPSOCircle()` and `DiagSLPSOCircle()` for the circle type or `UnitriangularDecompositionSOMinus()`, `MonomialSLPSOMinus()` and `DiagSLPSOMinus()` for the minus type to write a matrix $g \in \mathrm{SO}(e,d,q)$ as $g = u_1^{-1} \cdot p_{sign} \cdot diag \cdot u_2^{-1}$ where u_1, u_2 are lower unitriangular matrices, p_{sign} is a monomial matrix and $diag$ a diagonal matrix. It furthermore yields an SLP that returns the above matrices if evaluated with the LGO standard-generators.

Chapter 3

Special Linear Group

This chapter deals with the special linear group

3.1 Introduction and Quick Start of functions for SL

Concept: This implementation follows the ideas of "Straight-line programs with memory and matrix Bruhat decomposition" by Alice Niemeyer, Tomasz Popiel and Cheryl Praeger. In the following all references will mean this paper and in case we differ from this paper (due to readability or bug-fixing) this will also be remarked.

Let $g \in \mathrm{SL}(d, p^f)$ Bruhat Decomposition computes $g = u_1 \cdot w \cdot u_2$, where

- u_1, u_2 are lower triangular matrices
- w is monomial matrix

In this algorithm we want to compute the Bruhat-Decomposition of g and give g (respectively u_1, w and u_2) as word in the so called "LGO standard generators" (REF TODO).

1) While computing u_1 (resp u_2) with some kind of Gauß-Algorithm, we express the matrices as product of so called transvections

- For $1 \leq j < i \leq d$: $t_{i,j}(\alpha)$ is the matrix T with 1-entries on diagonal, $T_{i,j} = \alpha$, 0 elsewhere
  Each $t_{i,j}(\alpha)$ can be computed from $t_{2,1}(\alpha)$ via recursion, where we have to distinguish the odd and even dimensons (p12 Lemma 4.2). This again can be expressed as a product of $t_{2,1}(\omega^\ell)$ (where omega is a primitive element and $0 \leq \ell < f$). The transvections as words in the standard generators are described in (p12 Lemma 4.2).
  This yields a decomposition of u_1 and u_2 in standard generators.

2) In a further step we will decompose the monomial Matrix w in a signed permutation matrix p_sign and a diagonal Matrix diag. (How to associate p_sign with a product of generators is further described in (PART I b) and (PART III))

3) The last step is the decomposition of the diagonal Matrix in 2) as word in the standard generators.

We won't do this matrix multiplications directly, but write them in a list to evaluate in a Straight-LineProgram. (Section 2) Although described differently in the paper, we sometimes will allow instructions to multiply more than two elements (eg during conjugating). This doesn't affect the optimality of an slp much, but higly increases the readability of our implementation.

3.2 Implemented Subfunctions (Part I)

Later we will need some additional functions. Why they are needed and where they are needed is described here.

- `MakeSLP()`: After the `BruhatDecompositionSL()` we get a list of instructions to calculate the matrices we want using the LGO standard generators. `MakeSLP()` is used to get a SLP out of these instructions.
- `CoefficientsPrimitiveElement()`: It expresses an element w in a field fld as a linear combination of a Primitive Element. This is important for the transvections. (TODO Add Reference!)
- `MyPermutationMat()`: Turns a permutation into a permutation matrix. We need it to calculate the LGO standard generator.
- `LGOStandardGensSL()`: This function computes the standard generators of SL as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in odd characteristic". (TODO Add Reference!)
- `HighestSlotOfSLP()`: The following function determines the highest slot of a SLP constructed from the list slp will write in. This is important to glue SLPs together.
- `MatToWreathProd()` and `WreathProdToMat()`: In `PermSLP()` [3.8.14] we want to transform the monomial matrix w given by `UnipotentDecomposition()` into a diagonal matrix. (The exact procedure is described in `PermSLP()` [3.8.14])

  Since multiplying the LGO standard-generators s, v and x not only involves permutations but we also have to consider which non-zero entries are $+1$ and which -1, we want to associate this matrices with permutations on $2d$ points. (cf. Wreath-Product)

  $[s,v,x] \to Sym(2d), M \to$ `Mwr` where $i^{Mwr} = j$ and $(i+d)^{Mwr} = j+d$ if $M_{i,j} = 1$ and $i^{Mwr} = j+d$ and $(i+d)^{Mwr} = j$ if $M_{i,j} = -1$ for $1 \leq i \leq d$.

  Due to their relation to wreath-products, we will call denote the image of a matrix `M`$\in [s,v,x]$ by `Mwr`.

  In fact the association from `MatToWreathProd()` [3.8.6] is an isomorphism and we can associate to each permutation we compute during `PermSLP()` [3.8.14] a signed permutation matrix (a monomial matrix with only $+1$ and -1 as non-zero entries).

  $M_{i,j} = 1$ if $i^{Mwr} = j \leq d$ and $M_{i,j} = -1$ if $i^{Mwr} = j+d$
- `AEM()`: Write instructions for Ancient Egyptian Multiplication in slp. At several occasions we will need to compute a high power of some value saved in a memory slot.
- `TestIfMonomial()`: Tests if a given matrix M is monomial matrix. We use it to decide whether we are already finished in `UnipotentDecomposition()`.

For some functions also exist a NC version. See [3.7].

3.3 UnipotentDecomposition (Part II - a)

In this section is the `UnipotentDecomposition()` described. This method is used to compute the Unitriangular decomposition of the matrix g. [3.8.11]

For this we use five local functions in the `UnipotentDecomposition()`. They are `TransvecAtAlpha()`, `ShiftTransvections()`, `FastShiftTransvections()`, `BackShiftTransvections()` and `FastBackShiftTransvections()`.

The difference to `UnipotentDecompositionWithTi()` [3.4] is that this version won't store all the transvections $t_{i,i-1}(\omega^l)$. This will increase the runtime but reduce the memory usage by $(d-3)\cdot f$ compared to the `UnipotentDecompositionWithTi()`.

The function can be called for example by

Example

```
gap> d := 3;;
gap> q := 5;;
gap> L := SL(d, q);;
gap> m := PseudoRandom(L);;
gap> stdgens := LGOStandardGensSL(d, q);;
gap> UnipotentDecomposition( stdgens, g);;
```

3.4 UnipotentDecomposition saving Transvections (Part II - b)

In this section is the `UnipotentDecompositionWithTi()` described. This method is used to compute the Unitriangular decomposition of the matrix g. [3.8.12]

In this version we will store all the transvections $t_{i,i-1}(\omega^l)$. This will increase the memory usage by $(d-3)\cdot f$ but reduce runtime.

In `UnipotentDecompositionWithTi()` we use two local functions. They are `TransvectionAtAlpha()` and `ComputeAllTransvections()`.

The function can be called for example by

Example

```
gap> d := 3;;
gap> q := 5;;
gap> L := SL(d, q);;
gap> m := PseudoRandom(L);;
gap> stdgens := LGOStandardGensSL(d, q);;
gap> UnipotentDecompositionWithTi( stdgens, g);;
```

3.5 Decomposing the Monomial Matrix (Part III)

We use three functions to decompose the monomial matrix w we get from `UnipotentDecomposition()`. They are:

- `PermutationMonomialMatrix()`: Find the permutation (in Sym(d) corresponding to the monomial matrix w) and $diag$ a diagonal matrix, where $diag[i]$ is the non-zero entry of row i. [3.8.13]
- `PermSLP()`: In this function we will transform a monomial matrix $w \in \mathrm{SL}(d,q)$ into a diagonal matrix $diag$. Using only the standard-generators s,v,x. This will lead to a monomial matrix p_{sign} with only ± 1 in non-zero entries and $p_{sign}\cdot diag = w$ (i.e. $diag = (p_{sign})^{-1}\cdot w$).

Furthermore we will return list `slp` of instructions which will (when evaluated at the LGO standard-generators) yield *diag*.

It is sufficient for *diag* to be diagonal, if the permutation associated with w (i.e. $i^{\pi_w} = j$ if $M_{i,j} \neq 0$) is the inverse of the permutation associated to p_{sign} (again only to Sym(d)).

In `PermSLP()` we thus transform π_w to () using only $\{\pi_s, \pi_v, \pi_x\}$. In order to know *diag* without computing all matrix multiplications, (we don't know the signs of p_{sign}), we compute a second permutation simultaneously (here using their identification with permutations in Sym($2d$) and identifying $\{\pi_s, \pi_v, \pi_x\}$ with $\{s, v, x\}$). [3.8.14]

- `DiagonalDecomposition()`: Writes a list of instructions which evaluated on LGO standard-generators yield the diagonal matrix of the input. [3.8.15]

To these three functions is also a NC version implemented. See [3.7].

3.6 Main Function (Part IV)

In `BruhatDecompositionSL()` [3.8.16] we put everything together. We use the three functions `UnipotentDecomposition()` [3.8.11], `PermSLP()` [3.8.14] and `DiagonalDecomposition()` [3.8.15] to compute matrices with $u_1^{-1} \cdot p_{sign} \cdot diag \cdot u_2^{-1} = g$ and a SLP `pgr` that computes these matrices with the LGO standard generators.

Here is an exampel:

Example

```
gap> mat := [ [ Z(5)^2, Z(5)^0, Z(5)^2 ],
>             [ Z(5)^3, 0*Z(5), 0*Z(5) ],
>             [ 0*Z(5), Z(5)^2, Z(5)^2 ] ];;#!
gap> L := BruhatDecompositionSL(LGOStandardGensSL(3,5), mat);
gap> result := ResultOfStraightLineProgram(L[1], LGOStandardGensSL(3,5));
```

`BruhatDecompositionSLWithTi()` [3.8.17] works like `BruhatDecompositionSL()` [3.8.16] but uses `UnipotentDecompositionWithTi()` [3.8.12] instead of `UnipotentDecomposition()` [3.8.11].

You can use it in the same way like `BruhatDecompositionSL()`:

Example

```
gap> mat := [ [ Z(5)^2, Z(5)^0, Z(5)^2 ],
>             [ Z(5)^3, 0*Z(5), 0*Z(5) ],
>             [ 0*Z(5), Z(5)^2, Z(5)^2 ] ];;
gap> L := BruhatDecompositionSLWithTi(LGOStandardGensSL(3,5), mat);
gap> result := ResultOfStraightLineProgram(L[1], LGOStandardGensSL(3,5));
```

To both functions is also a NC version implemented. See [3.7].

3.7 NC Version

Here is the NC version of the Bruhat Decomposition described. In all implemented functions are all used functions replaced through their NC version (if one exists). Moreover are all checks from functions of MyBruhatDecomposition removed.

These functions has been modified by this actions and got a NC Version:

- `MakeSLP()`[3.8.1] → `MakeSLPNC()`[3.8.1] (uses the NC version of `StraightLineProgram`)
- `MyPermutationMat()` [3.8.3] → `MyPermutationMatNC()` [3.8.3] (uses the NC version of `ConvertToMatrixRep`)
- `LGOStandardGensSL()` [3.8.4] → `LGOStandardGensSLNC()` [3.8.4] (uses the NC version of `MyPermutationMat()`)
- `MatToWreathProd()` [3.8.6] → `MatToWreathProdNC()` [3.8.6] (no checks for user input)
- `TestIfMonomial()` [3.8.9] → `TestIfMonomialNC()` [3.8.9] (no checks for user input)
- `UnipotentDecomposition()` [3.8.11] → `UnipotentDecompositionNC()` [3.8.11] (no checks for user input)
- `UnipotentDecompositionWithTi()` [3.8.12] → `UnipotentDecompositionWithTiNC()` [3.8.12] (no checks for user input)
- `PermutationMonomialMatrix()` [3.8.13] → `PermutationMonomialMatrixNC()` [3.8.13] (no checks for user input)
- `PermSLP()` [3.8.14] → `PermSLPNC()` [3.8.14] (no checks for unser input and uses `PermutationMonomialMatrixNC()`)
- `DiagonalDecomposition()` [3.8.15] → `DiagonalDecompositionNC()` [3.8.15] (no checks for user input)
- `BruhatDecompositionSL()` [3.8.16] → `BruhatDecompositionSLNC()` [3.8.16] (uses `UnipotentDecompositionNC()`, `PermSLPNC()` and `DiagonalDecompositionNC`)
- `BruhatDecompositionSLWithTi()` [3.8.17] → `BruhatDecompositionSLWithTiNC()` [3.8.17] (uses `UnipotentDecompositionWithTiNC()`, `PermSLPNC()` and `DiagonalDecompositionNC()`)

3.8 Functions for SL

3.8.1 MakeSLP

▷ `MakeSLP(`*`slp, genlen`*`)` (function)
▷ `MakeSLPNC(`*`arg`*`)` (function)

Returns: An SLP using the instructions of slp and genlen inputs
slp: A list of instructions for a straight-line program,
genlen: The number of inputs for our SLP (ie the number of generators)
To increase readability, the lists slp as defined later (see Unipotent-, Diagonal-, BruhatDecompositionSL and PermSLP) start with [1,1],[2,1],.. [5,1]. However this represents the LGO standard-generators and is the input of our straight-line program. Defining and SLP we thus have to exclude this instructions from our list.

3.8.2 CoefficientsPrimitiveElement

▷ `CoefficientsPrimitiveElement(fld, alpha)` (function)

Returns: Coefficients (A vector c sth for omega primitive element alpha = sum c[i] omega^(i-1))

fld: A field,

alpha: An element of fld

The following function has been written by Thomas Breuer. It expresses an element alpha in a field fld as a linear combination of a Primitive Element.

3.8.3 MyPermutationMat

▷ `MyPermutationMat(perm, dim, fld)` (function)

▷ `MyPermutationMatNC(arg)` (function)

Returns: The permutation matrix of perm over $M_{dxd}(fld)$ (ie $res_{i,j} = One(fld)$ if $i^{perm} = j$)

perm: A permutation,

dim: A natural number,

fld: A field

Given a permutation an integer $d > 0$ and a field fld, this function computes the permutation matrix P in $M_{dxd}(fld)$.

3.8.4 LGOStandardGensSL

▷ `LGOStandardGensSL(d, q)` (function)

▷ `LGOStandardGensSLNC(arg)` (function)

Returns: stdgens (the LGO standard-generators of SL(d,q))

d: the dimension of our matrix,

q: A prime power $q = p^f$, where F_q ist the field whereover the matrices are defined

This function computes the standard generators of SL as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in odd characteristic" (This matrices can also be found in the paper ch 3.1 ps 6-7)

3.8.5 HighestSlotOfSLP

▷ `HighestSlotOfSLP(slp)` (function)

Returns: highestslot (The number of slots this SLP will need if evaluated)

slp: A list of instructions satisfying the properties for an SLP

The following function determines the highest slot a SLP constructed from the list slp will write in.

3.8.6 MatToWreathProd

▷ `MatToWreathProd(M)` (function)

▷ `MatToWreathProdNC(arg)` (function)

Returns: perm (the permutation Mwr)

M: A monomial matrix with only +1 and -1 entries

In PermSLP we want to transform the monomial matrix w given by UnipotentDecomposition() into a diagonal matrix. (The exact procedure is described in PermSLP) Since multiplying the LGO standard-generators s, v and x not only involves permutations but we also have to consider which non-zero entries are +1 and which -1, we want to associate this matrices with permutations on $2d$ points. (cf

Wreath-Product) $\langle s,v,x\rangle \to \mathrm{Sym}(2d), M \to Mwr$ where $i^{Mwr} = j$ and $(i+d)^{Mwr} = j+d$ if $M_{i,j} = 1$ and $i^{Mwr} = j+d$ and $(i+d)^{Mwr} = j$ if $M_{i,j} = -1$ for $1 \le i \le d$ Due to their relation to wreath-products, we will call denote the image of a matrix $M \in \langle s,v,x\rangle$ by Mwr

3.8.7 WreathProdToMat

▷ `WreathProdToMat(perm, dim, fld)` (function)

Returns: res (The Matrix M satisfying the below properties)

perm: A permutation in Sym($2d$) sth. $i, i+d_1 : 1 \le i \le d$ are blocks,
dim: The dimension of the matrix we want perm send to,
fld: The field whereover the matrix is defined.
In fact the association above is an isomorphism and we can associate to each permutation we compute during PermSLP a unique monomial matrix whose non-zero entries are +1 or -1. $M_{i,j} = 1$ if $i^{Mwr} = j \le d$ and $M_{i,j} = -1$ if $i^{Mwr} = j+d$

3.8.8 AEM

▷ `AEM(spos, respos, tmppos, k)` (function)

Returns: instr (Lines of an SLP that will (when evaluated) take the value b saved in spos and write b^k in respos)

AEM (Ancient Egytian Multiplication)
spos: The memory slot, where a value b is saved in,
respos: The memory slot we want the exponentation to be written in,
tmppos: A memory slot for temporary results,
k: An integer
At several occasions we will need to compute a high power of some value saved in a memory slot. For this purpose there is a variaton of AEM implemented below. Remarks: tmpos and respos must differ. If spos = respos or spos = tmpos it will be overwritten.

3.8.9 TestIfMonomial

▷ `TestIfMonomial(M)` (function)
▷ `TestIfMonomialNC(arg)` (function)

Returns: True if M is a monomial matrix, otherwise false.

M: A Matrix
Tests if a given matrix M is a monomial matrix. There is function in GAP, however it does not seem to work for SL(d,q).

3.8.10 Transvections2

▷ `Transvections2(stdgens, omega, slp, pos)` (function)

Returns: slp: The list of instruction with additional instructions writing $t_{2,1}(\omega^\ell)$ in Slot pos[l+1] $0 \le \ell \le f-1$.

stdgens: The LGO standard-generators of SL(d,q)
omega: A primitive element of GF(q)
slp: A list of instructions
pos: A list of numbers, denoting where to save the transvections $t_{2,1}(\omega^\ell)$ for $0 \le \ell \le f-1$

Let stdgens be the list of standard generators for SL(d,p^f) and let omega be a primitive element of G(p^f). This function computes $T_2 := \{t_{2,1}(\omega^\ell) \mid 0 \leq \ell \leq f-1\}$ Record what we do in slp This function coincides with eq (6) p12.

3.8.11 UnipotentDecomposition

▷ `UnipotentDecomposition(stdgens, g)` (function)
▷ `UnipotentDecompositionNC(arg)` (function)

Returns: slp (A list of instructions yielding u_1,u_2 if evaluated as SLP), $[u_1,g,u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators
g: A matrix in SL(d,q)
Computes the Unitriangular decomposition of the matrix g.

3.8.12 UnipotentDecompositionWithTi

▷ `UnipotentDecompositionWithTi(stdgens, g)` (function)
▷ `UnipotentDecompositionWithTiNC(arg)` (function)

Returns: slp (A list of instructions yielding u_1,u_2 if evaluated as SLP), $[u_1,g,u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators
g: A matrix in SL(d,q)
Computes the Bruhat decomposition of the matrix g, given the standard generators for the group. In this version we will store all the transvections $t_{i,i-1}(\omega^\ell)$. this will increase the memory usage by $(d-3)\cdot f$ but reduce the runtime.

3.8.13 PermutationMonomialMatrix

▷ `PermutationMonomialMatrix(M)` (function)
▷ `PermutationMonomialMatrixNC(arg)` (function)

Returns: diag (The vector of non-zero entries, where diag$[i]$ is the non-zero entry of row i.), perm (The permutation associated to M, i.e. $i^{perm} = j$ if $M_{i,j}$ is not 0)

M: A monomial matrix.
Find the permutation (in Sym(d)) corresponding to the input monomial matrix.

3.8.14 PermSLP

▷ `PermSLP(stdgens, mat, slp)` (function)
▷ `PermSLPNC(arg)` (function)

Returns: slp (A list of instructions to evaluate p_sign if slp was Input then this instructions are added to slp), p_sign (The signed permutation matrix), mat (The diagonal matrix diag)

stdgens: The LGO standard-generators
mat: A monomial matrix (ie w)
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $w \in$ SL(d,q) into a diagonal matrix diag. Using only the standard-generators s,v,x this will lead to a monomial matrix p_sign with only +-1 in non-zero entries and p_sign*diag = w (i.e. diag = p_sign^-1*w). Furthermore we will return list slp of

instructions which will (when evaluated at the LGO standard-generators) yield diag.
It is sufficient for diag to be diagonal, if the permutation associated with w (i.e. $i_w^\pi = j$ if $M_{i,j}$ not 0) is the inverse of the permutation associated to p_sign (again only to Sym(d))
In PermSLP we thus transform π_w to () using only $\{\pi_s, \pi_v, \pi_x\}$ In order to know diag without computing all matrix multiplications, (we don't know the signs of p_sign), we compute a second permutation simultaneously (here using their identification with permutations in Sym($2d$) and identifying $\{\pi_s, \pi_v, \pi_x\}$ with $\{s, v, x\}$)

3.8.15 DiagonalDecomposition

▷ `DiagonalDecomposition(stdgens, diag, slp)` (function)
▷ `DiagonalDecompositionNC(arg)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp), hres (The the identity matrix)

stdgens: The LGO standard-generators
diag: A diagonal matrix (eg diag)
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated on LGO standard-generators yield the diagonal matrix of the input.

3.8.16 BruhatDecompositionSL

▷ `BruhatDecompositionSL(stdgens, g)` (function)
▷ `BruhatDecompositionSLNC(arg)` (function)

Returns: pgr (A SLP to compute u_1, u_2, p_{sign} and $diag$ and the matrices u_1, u_2, p_{sign} and $diag$ itself.)

stdgens: The LGO standard-generators
g: A matrix in SL(d, q)
Uses `UnipotentDecomposition()`, `PermSLP()` and `DiagonalDecomposition()` to write a matrix $g \in \mathrm{SL}(d,q)$ as $g = u_1^{-1} \cdot p_{sign} \cdot diag \cdot u_2^{-1}$ where u_1, u_2 are lower unitriangular matrices, p_{sign} is a monomial matrix with only +1 and -1 as non-zero entries and $diag$ a diagonal matrix. It furthermore yields an SLP that returns the above matrices if evaluated with the LGO standard-generators.

3.8.17 BruhatDecompositionSLWithTi

▷ `BruhatDecompositionSLWithTi(stdgens, g)` (function)
▷ `BruhatDecompositionSLWithTiNC(arg)` (function)

Returns: pgr (A SLP to compute u_1, u_2, p_{sign} and $diag$ and the matrices u_1, u_2, p_{sign} and $diag$ itself.)

stdgens: The LGO standard-generators
g: A matrix in SL(d, q)
As `BruhatDecompositionSL()` but replaces `UnipotentDecomposition()` by `UnipotentDecompositionWithTi()`.

Chapter 4

Special Unitary Group

This chapter deals with the special unitary group

4.1 Introduction and Quick Start of functions for SU

TODO

4.2 Functions for SU

4.2.1 MakePermutationMat

▷ `MakePermutationMat(perm, dim, fld)` (function)

Returns: The permutation matrix of perm over $M_{dxd}(fld)$ (ie $res_{i,j} = One(fld)$ if $i^{perm} = j$)

perm: A permutation,
dim: A natural number,
fld: A field
This is the same function as MyPermutationMat.

4.2.2 LGOStandardGensSU

▷ `LGOStandardGensSU(d, q)` (function)

Returns: stdgens (the LGO standard-generators of SU(d,q))

d: The dimension of our matrix.
q: A prime power $q = p^f$, where F_q ist the field whereover the matrices are defined
This function computes the standard generators of SU as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in odd characteristic". If q is even, `LGOStandardGensSUEvenChar(d,q)` is called automatically.

4.2.3 LGOStandardGensSUEvenChar

▷ `LGOStandardGensSUEvenChar(d, q)` (function)

Returns: stdgens (the LGO standard-generators of SU(d,q)) for q even

d: The dimension of our matrix.
q: A 2 power $q = 2^f$, where F_q ist the field whereover the matrices are defined

This function computes the standard generators of Sp as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in even characteristic"

4.2.4 CoefficientsPrimitiveElementS

▷ `CoefficientsPrimitiveElementS(fld, alpha, basis)` (function)

Returns: Coefficients (A vector c sth alpha = sum c[i] b[i])

fld: A field,
alpha: An element of fld
basis: A F_p basis of fld
It expresses an element alpha in a field fld as a linear combination of the basis elements.

4.2.5 UnitriangularDecompositionSUEven

▷ `UnitriangularDecompositionSUEven(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators
g: A matrix in SU(d,q) where d is even and q is odd
Computes the Unitriangular decomposition of the matrix g.

4.2.6 UnitriangularDecompositionSUEvenAndEvenChar

▷ `UnitriangularDecompositionSUEvenAndEvenChar(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators
g: A matrix in SU(d,q) where d is even and q is even
Computes the Unitriangular decomposition of the matrix g.

4.2.7 UnitriangularDecompositionSUOdd

▷ `UnitriangularDecompositionSUOdd(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators
g: A matrix in SU(d,q) where d is odd and q is odd
Computes the Unitriangular decomposition of the matrix g.

4.2.8 UnitriangularDecompositionSU

▷ `UnitriangularDecompositionSU(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators
g: A matrix in SU(d,q)
Computes the Unitriangular decomposition of the matrix g. Depending on q and d the correct

function of UnitriangularDecompositionSUEven, UnitriangularDecompositionSUOdd and UnitriangularDecompositionSUOdd is choosen.

4.2.9 MonomialSLPSUOdd

▷ MonomialSLPSUOdd(*stdgens, mat, slp*) (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)

stdgens: The LGO standard-generators
mat: A monomial matrix (ie w) in $\mathrm{SU}(d,q)$ with d even and q odd
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{SU}(d,q)$ with d even and q odd into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

4.2.10 MonomialSLPSUEvenAndEvenChar

▷ MonomialSLPSUEvenAndEvenChar(*stdgens, mat, slp*) (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)

stdgens: The LGO standard-generators
mat: A monomial matrix (ie w) in $\mathrm{SU}(d,q)$ with d even and q even
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{SU}(d,q)$ with d even and q even into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

4.2.11 MonomialSLPSUEven

▷ MonomialSLPSUEven(*stdgens, mat, slp*) (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)

stdgens: The LGO standard-generators
mat: A monomial matrix (ie w) in $\mathrm{SU}(d,q)$ with d odd and q odd
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{SU}(d,q)$ with d odd and q odd into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

4.2.12 CheckContinue

▷ `CheckContinue(perm, m)` (function)

Returns: True or false

perm: A permutation

m: A natural number. If this function is called by `MonomialSLPSU` then $m = \frac{d}{2}$ or $m = \frac{(d-1)}{2}$

This is a help function for `MonomialSLPSU`. This function checks whether for all cycle c of perm holds: LargestMovedPoint(c) $\leq$ m or SmallestMovedPoint(c) > m. Notice that this is the condition for the main loop of `MonomialSLPSU`.

4.2.13 CycleFromPermutation

▷ `CycleFromPermutation(g)` (function)

Returns: List of permutations

g: A permutation

This is a help function for `MonomialSLPSUOdd`. This function computes the cycles of g and stores them in the output list.

4.2.14 CycleFromListMine

▷ `CycleFromListMine(nc, h)` (function)

Returns: TODO

nc: A subset of [1,...,h]

h: A natural number (the largest moved point of a permutation)

This is a help function for `CycleFromPermutation`. This function computes a cycle in Sym_h which corresponds to nc.

4.2.15 DiagSLPSUOdd

▷ `DiagSLPSUOdd(stdgens, diag, slp)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators

diag: A diagonal matrix (eg diag) in SU(d,q) with d odd and q odd

slp: An already existing list of instructions *optional

Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

4.2.16 DiagSLPSU

▷ `DiagSLPSU(stdgens, diag, slp)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators

diag: A diagonal matrix (eg diag) in SU(d,q)

slp: An already existing list of instructions *optional

Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input. Depending on q and d the correct function of `DiagSLPSUEven`, `DiagSLPSUEvenAndEvenChar` and `DiagSLPSUOdd` is choosen.

4.2.17 DiagSLPSUEven

▷ `DiagSLPSUEven(`*`stdgens, diag, slp`*`)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators
diag: A diagonal matrix (eg diag) in SU(d,q) with d even and q odd
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

4.2.18 DiagSLPSUEvenAndEvenChar

▷ `DiagSLPSUEvenAndEvenChar(`*`stdgens, diag, slp`*`)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators
diag: A diagonal matrix (eg diag) in SU(d,q) with d even and q even
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

4.2.19 BruhatDecompositionSU

▷ `BruhatDecompositionSU(`*`stdgens, g`*`)` (function)

Returns: pgr (A SLP to compute u_1, u_2, p_{sign} and $diag$ and the matrices u_1, u_2, p_{sign} and $diag$ itself.)

stdgens: The LGO standard-generators
g: A matrix in SU(d,q)
Uses `UnitriangularDecompositionSU()`, `MonomialSLPSU()` and `DiagSLPSU()` to write a matrix $g \in$ SU(d,q) as $g = u_1^{-1} \cdot p_{sign} \cdot diag \cdot u_2^{-1}$ where u_1, u_2 are lower unitriangular matrices, p_{sign} is a monomial matrix and $diag$ a diagonal matrix. It furthermore yields an SLP that returns the above matrices if evaluated with the LGO standard-generators.

Chapter 5

Symplectic Group

This chapter deals with the symplectic group

5.1 Introduction and Quick Start of functions for Sp

TODO

5.2 Functions for Sp

5.2.1 LGOStandardGensSp

▷ `LGOStandardGensSp(d, q)` (function)

Returns: stdgens (the LGO standard-generators of Sp(d,q))

d: The dimension of our matrix. Notice that d needs to be even for symplectic groups.

q: A prime power $q = p^f$, where F_q ist the field whereover the matrices are defined

This function computes the standard generators of Sp as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in odd characteristic"

5.2.2 LGOStandardGensSpEvenChar

▷ `LGOStandardGensSpEvenChar(d, q)` (function)

Returns: stdgens (the LGO standard-generators of Sp(d,q)) for q even

d: The dimension of our matrix. Notice that d needs to be even for symplectic groups.

q: A 2 power $q = 2^f$, where F_q ist the field whereover the matrices are defined

This function computes the standard generators of Sp as given by C. R. Leedham-Green and E. A. O'Brien in "Constructive Recognition of Classical Groups in even characteristic"

5.2.3 UnitriangularDecompositionSp

▷ `UnitriangularDecompositionSp(stdgens, g)` (function)

Returns: slp (A list of instructions yielding u_1, u_2 if evaluated as SLP), $[u_1, g, u_2]$ (The matrices of the Bruhat-Decomposition)

stdgens: The LGO standard-generators

g: A matrix in Sp(d,q)

Computes the Unitriangular decomposition of the matrix g.

5.2.4 MonomialSLPSp

▷ `MonomialSLPSp(stdgens, mat, slp)` (function)

Returns: slp (A list of instructions to evaluate tmpvalue. If slp is also given as input then this instructions are added to slp), [tmpvalue,diag] (tmpvalue is a monomial matix such that tmpvalue*mat = diag where diag is a diagonal matrix)

stdgens: The LGO standard-generators
mat: A monomial matrix (ie w)
slp: An already existing list of instructions *optional
In this function we will transform a monomial matrix $mat \in \mathrm{Sp}(d,q)$ into a diagonal matrix diag. Using only the standard-generators s,u,v this will lead to a monomial matrix tmpvalue and $tmpvalue^{-1} \cdot diag = mat$ (i.e. diag = tmpvalue*mat). Furthermore we will return list slp of instructions which will (when evaluated at the LGO standard-generators) yields diag.

5.2.5 DiagSLPSp

▷ `DiagSLPSp(stdgens, diag, slp)` (function)

Returns: slp (A list of instructions to evaluate diag if slp was Input then this instructions are added to slp)

stdgens: The LGO standard-generators
diag: A diagonal matrix (eg diag)
slp: An already existing list of instructions *optional
Writes a list of instructions which evaluated with LGO standard-generators yield the diagonal matrix of the input.

5.2.6 BruhatDecompositionSp

▷ `BruhatDecompositionSp(stdgens, g)` (function)

Returns: pgr (A SLP to compute u_1, u_2, p_{sign} and $diag$ and the matrices u_1, u_2, p_{sign} and $diag$ itself.)

stdgens: The LGO standard-generators
g: A matrix in $\mathrm{Sp}(d,q)$
Uses `UnitriangularDecompositionSp()`, `MonomialSLPSp()` and `DiagSLPSp()` to write a matrix $g \in \mathrm{Sp}(d,q)$ as $g = u_1^{-1} \cdot p_{sign} \cdot diag \cdot u_2^{-1}$ where u_1, u_2 are lower unitriangular matrices, p_{sign} is a monomial matrix and $diag$ a diagonal matrix. It furthermore yields an SLP that returns the above matrices if evaluated with the LGO standard-generators.

Index