```
> restart; with(LinearAlgebra): interface(rtablesize=[infinity,
  infinity]):
```

Code to accompany paper "Equivariant isomorphism of quantum lens spaces of low dimension" by S. Eilers and S.E. Zegers.

We study only QLS in the original form as defined by Hong and Szymanski, given by parameters r and (m_1,...,m_(k+1)). with m_i prime to r. We always represent them

```
> (r,M);
```

$$r, M \tag{1}$$

with m a Maple list with k+1 entries that are all units in Z_r, such as

```
> (3,[1,2,1,1]); (4,[3]); (10,[3,3,3]);
```

$$3, [1, 2, 1, 1]$$
$$4, [3]$$
$$10, [3, 3, 3] \tag{2}$$

Running code with m_i that are not prime to r will cause undesired effects, even infinite loops in some cases.

Matrices

In this section, we define various matrices to organize our computations. A main goal is to define a workable class of matrices H describing all possible isomorphisms as explained in Proposition 4.4.

We work with forward shift in Maple but mainly with backward shift (**S**, cf. Section 4) in the paper. Inverse or transpose gets one from the other.

```
> Sforward:=n->Matrix(n,n,{(1,n)=1,seq((i,i-1)=1,i=2..n)});
Warning, (in Sforward) `i` is implicitly declared local
```

$$Sforward := n \mapsto Matrix(n, n, \{seq((i, i-1) = 1, i = 2..n), (1, n) = 1\}) \tag{1.1}$$

This is the adjacency matrix for the graph **E_{r;M}:**

```
> QLSadjacency:=proc(r,M) local work,i,j; work:=Matrix([[Sforward
  (r)^M[1]],seq([ZeroMatrix(r,r)],i=2..nops(M))]); for j from 2 to
  nops(M) do work:=<work|Matrix([seq([Sforward(r)^M[i]],i=1..j),seq
  ([ZeroMatrix(r,r)],i=j+1..nops(M))])>; end; end;
```

$$QLSadjacency := \mathbf{proc}(r, M) \tag{1.2}$$
$$\quad \mathbf{local}\ work, i, j;$$
$$\quad work := Matrix([[Sforward(r)^M[1]], seq([LinearAlgebra:-ZeroMatrix(r, r)], i = 2..nops(M))]);$$
$$\quad \mathbf{for}\ j\ \mathbf{from}\ 2\ \mathbf{to}\ nops(M)\ \mathbf{do}$$
$$\quad\quad work := <work|Matrix([seq([Sforward(r)^M[i]], i = 1..j), seq([LinearAlgebra:-ZeroMatrix(r, r)], i = j+1..nops(M))])>$$
$$\quad \mathbf{end\ do}$$
$$\mathbf{end\ proc}$$

This is the map induced by **lt_*.** We decribe it via the matrix **B[r;M]** just as in Section 4. Note transposition and signs.

```
> DGshiftmatrix:=proc(r,M) local work,i,j; work:=Matrix([[Sforward
```

```
  (r)^M[1]],seq([ZeroMatrix(r,r)],i=2..nops(M))]); for j from 2 to
  nops(M) do work:=<work|Matrix([seq([-Sforward(r)^M[j]],i=1..j-1),
  [Sforward(r)^M[j]],seq([ZeroMatrix(r,r)],i=j+1..nops(M))])>; end;
  return Transpose(work); end;
```

$$DGshiftmatrix := \mathbf{proc}(r, M) \tag{1.3}$$

```
    local work, i, j;
    work := Matrix([[Sforward(r)^M[1]], seq([LinearAlgebra:-ZeroMatrix(r, r)], i = 2
     ..nops(M))]);
    for j from 2 to nops(M) do
        work := <work|Matrix([seq([-Sforward(r)^M[j]], i = 1..j-1),
        [Sforward(r)^M[j]], seq([LinearAlgebra:-ZeroMatrix(r, r)], i = j+1
        ..nops(M))])>
    end do;
    return LinearAlgebra:-Transpose(work)
end proc
```

```
> QLSadjacency(4,[1,3,1]), DGshiftmatrix(4,[1,3,1]);
```

$$\begin{bmatrix}
0&0&0&1&0&0&0&1&0&0&0&1\\
1&0&0&0&1&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0
\end{bmatrix}, \tag{1.4}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The following threematrices extract relevant subentries for various use. They only depend on r and the number of entries of M.

```
> Pspecial:=(r,M)->DiagonalMatrix(ListTools[Flatten]([seq([1,seq
  (0,j=2..r)],i=1..nops(M))]));
```

Warning, (in Pspecial) `i` is implicitly declared local

Warning, (in Pspecial) `j` is implicitly declared local

$$Pspecial := (r, M) \mapsto LinearAlgebra:-DiagonalMatrix(ListTools[Flatten]([seq([1, seq(0, j = 2..r)], i = 1..nops(M))])) \quad \textbf{(1.5)}$$

```
> Ccompress:=(r,M)->Matrix(nops(M),r*nops(M),{seq((i,1+(i-1)*r)=1,
  i=1..nops(M))});
```

Warning, (in Ccompress) `i` is implicitly declared local

$$Ccompress := (r, M) \mapsto Matrix(nops(M), r \cdot nops(M), \{seq((i, 1 + (i-1) \cdot r) = 1, i = 1..nops(M))\}) \quad \textbf{(1.6)}$$

```
> theUnit:=(r,M)->Matrix(r*nops(M),1,{seq((i*r+1,1)=1,i=0..nops(L)
  -1)});
```

Warning, (in theUnit) `i` is implicitly declared local

$$theUnit := (r, M) \mapsto Matrix(r \cdot nops(M), 1, \{seq((i \cdot r + 1, 1) = 1, i = 0..nops(L) - 1)\}) \quad \textbf{(1.7)}$$

```
> Pspecial(4,[1,3,1]), Ccompress(4,[1,3,1]), theUnit(4,[1,3,1]);
```

(1.8)

$$\begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}, \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0 \end{bmatrix}, \begin{bmatrix} 1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix} \quad (1.8)$$

Now comes the matrix **P_m/n** from the paper's Section 4.

```
> Pdiagonal:=(r,m,n)->Matrix(r,r,{seq(((i*n mod r)+1,(i*m mod r)
  +1)=1,i=0..r-1)});
```

Warning, (in Pdiagonal) `i` is implicitly declared local

$$Pdiagonal := (r, m, n) \mapsto Matrix(r, r, \{seq(((i \cdot n \bmod r) + 1, (i \cdot m \bmod r) + 1) = 1, i = 0 \ldots r - 1)\}) \quad (1.9)$$

The next procedure is used to extract the entries that must vanish for positivity.

```
> manyDiagonals:=proc(r,n) local i,j; Matrix(r,r,{seq(seq(((i mod
  r)+1,(n*j+i mod r)+1)=1,i=0..r-1),j=0..r/n-1)}); end;
```

$$manyDiagonals := \mathbf{proc}(r, n) \quad (1.10)$$

local i, j;

$Matrix(r, r, \{seq(seq(((i \bmod r) + 1, (n*j + i \bmod r) + 1) = 1, i = 0 \ldots r - 1), j = 0 \ldots r/n - 1)\})$

end proc

```
> manyDiagonals(8,1), manyDiagonals(8,2), manyDiagonals(8,4);
```

$$\begin{bmatrix} 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1 \end{bmatrix}, \begin{bmatrix} 1&0&1&0&1&0&1&0\\ 0&1&0&1&0&1&0&1\\ 1&0&1&0&1&0&1&0\\ 0&1&0&1&0&1&0&1\\ 1&0&1&0&1&0&1&0\\ 0&1&0&1&0&1&0&1\\ 1&0&1&0&1&0&1&0\\ 0&1&0&1&0&1&0&1 \end{bmatrix}, \begin{bmatrix} 1&0&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0\\ 0&0&1&0&0&0&1&0\\ 0&0&0&1&0&0&0&1\\ 1&0&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0\\ 0&0&1&0&0&0&1&0\\ 0&0&0&1&0&0&0&1 \end{bmatrix} \quad (1.11)$$

Block matrix created from this, identity in diagonal, zero above.

```
> findPmatrix:=proc(r,L) local i,work; work:=<IdentityMatrix(r,r)
  |ZeroMatrix(r,r*(nops(L)-1))>; for i from 2 to nops(L) do work:=
  <work,<ZeroMatrix(r,r*(i-2))|manyDiagonals(r,gcd(L[i]-L[i-1],r))
  |IdentityMatrix(r,r)|ZeroMatrix(r,r*(nops(L)-i))>>; end; return
  map(x->min(1,x),work^(nops(L)-1)); end;
```

```
findPmatrix := proc(r, L)                                              (1.12)
    local i, work;
    work := < LinearAlgebra:-IdentityMatrix(r, r)|LinearAlgebra:-ZeroMatrix(r, r
    * (nops(L) - 1)) > ;
    for i from 2 to nops(L) do
        work := < work, < LinearAlgebra:-ZeroMatrix(r, r * (i - 2))
        |manyDiagonals(r, gcd(L[i] - L[i - 1], r))|LinearAlgebra:-IdentityMatrix(r, r)
        |LinearAlgebra:-ZeroMatrix(r, r * (nops(L) - i)) > > >
    end do;
    return map(x→min(1, x), work^(nops(L) - 1))
end proc
```

Computes the P_0 set as defined just before Lemma 2.8.

```
> findP0set:=proc(r,L) local work,i; work:=Ccompress(r,L).Pspecial
  (r,L).findPmatrix(r,L); map(x->min(x,1),add([seq(SubMatrix(work,
  1..nops(L),1+(i-1)*r..i*r),i=1..nops(L))])); end;
```

```
findP0set := proc(r, L)                                                (1.13)
    local work, i;
    work := `.`(Ccompress(r, L), Pspecial(r, L), findPmatrix(r, L));
    map(x→min(x, 1), add([seq(LinearAlgebra:-SubMatrix(work, 1 ..nops(L), 1
    + (i - 1) * r ..i * r), i = 1 ..nops(L))]))
end proc
```

```
> findPmatrix(6,[5,1,1]), findP0set(6,[5,1,1]);
```

(1.14)

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0\\
1&0&1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&1&0&1&0&1&0&0&0&1&0&0&0&0&0&1&0&0\\
1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&1
\end{bmatrix},
\begin{bmatrix}
1&0&0&0&0&0\\
1&0&1&0&1&0\\
1&0&1&0&1&0
\end{bmatrix} \qquad (1.14)$$

Generic block of free variables, named to avoid repetitions

```
> Yblock:=(r,k,l)->Matrix(r,r,{seq(seq((i+1,j+1)=y[k,l,i,j],i=0..
  r-1),j=0..r-1)});
```

Warning, (in Yblock) `j` is implicitly declared local

Warning, (in Yblock) `i` is implicitly declared local

$$Yblock := (r,k,l) \mapsto Matrix\left(r,r,\left\{seq\left(seq\left((i+1,j+1)=y_{k,l,i,j},i=0..r-1\right),j=0..r-1\right)\right\}\right) \qquad (1.15)$$

Combined to ensure zeros needed for positivity below diagonal

```
> Yindeterminates:=proc(r,L) local work,i,j; work:=Matrix([
  [ZeroMatrix(r,r)],seq([Yblock(r,i,0)],i=1..nops(L)-1)]); for j
  from 2 to nops(L) do work:=<work|Matrix([seq([ZeroMatrix(r,r)],i=
  1..j-1),[ZeroMatrix(r,r)],seq([Yblock(r,i,j-1)],i=j..nops(L)-1)])
  >; end; Zip(`*`,findPmatrix(r,L),work); end;
```

$$Yindeterminates := \mathbf{proc}(r,L) \qquad (1.16)$$

local *work*, *i*, *j*;

work := *Matrix*([[*LinearAlgebra*:-*ZeroMatrix*(*r*, *r*)], *seq*([*Yblock*(*r*, *i*, 0)], *i* = 1 ..*nops*(*L*) − 1)]);

for *j* **from** 2 **to** *nops*(*L*) **do**

work := < *work*|*Matrix*([*seq*([*LinearAlgebra*:-*ZeroMatrix*(*r*, *r*)], *i* = 1 ..*j* − 1),

```
        [LinearAlgebra:-ZeroMatrix(r, r)], seq([Yblock(r, i, j - 1)], i = j..nops(L) - 1)]
        ) >
    end do;
    LinearAlgebra:-Zip(`*`, findPmatrix(r, L), work)
end proc
```

```
> Yindeterminates(6,[5,1,1]);
```

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
y_{1,0,0,0} & 0 & y_{1,0,0,2} & 0 & y_{1,0,0,4} & 0 & 0 & 0 & 0 & \cdots \\
0 & y_{1,0,1,1} & 0 & y_{1,0,1,3} & 0 & y_{1,0,1,5} & 0 & 0 & 0 & \cdots \\
y_{1,0,2,0} & 0 & y_{1,0,2,2} & 0 & y_{1,0,2,4} & 0 & 0 & 0 & 0 & \cdots \\
0 & y_{1,0,3,1} & 0 & y_{1,0,3,3} & 0 & y_{1,0,3,5} & 0 & 0 & 0 & \cdots \\
y_{1,0,4,0} & 0 & y_{1,0,4,2} & 0 & y_{1,0,4,4} & 0 & 0 & 0 & 0 & \cdots \\
0 & y_{1,0,5,1} & 0 & y_{1,0,5,3} & 0 & y_{1,0,5,5} & 0 & 0 & 0 & \cdots \\
y_{2,0,0,0} & 0 & y_{2,0,0,2} & 0 & y_{2,0,0,4} & 0 & y_{2,1,0,0} & 0 & 0 & \cdots \\
0 & y_{2,0,1,1} & 0 & y_{2,0,1,3} & 0 & y_{2,0,1,5} & 0 & y_{2,1,1,1} & 0 & \cdots \\
y_{2,0,2,0} & 0 & y_{2,0,2,2} & 0 & y_{2,0,2,4} & 0 & 0 & 0 & y_{2,1,2,2} & \cdots \\
0 & y_{2,0,3,1} & 0 & y_{2,0,3,3} & 0 & y_{2,0,3,5} & 0 & 0 & 0 & y_{2,} \cdots \\
y_{2,0,4,0} & 0 & y_{2,0,4,2} & 0 & y_{2,0,4,4} & 0 & 0 & 0 & 0 & \cdots \\
0 & y_{2,0,5,1} & 0 & y_{2,0,5,3} & 0 & y_{2,0,5,5} & 0 & 0 & 0 & \cdots
\end{bmatrix} \quad \textbf{(1.17)}$$

Diagonal matrix with shifts allowed

```
> PSdiagonal:=proc(r,M,N,L) local work,i,j; work:=Matrix([
  [Sforward(r)^(-L[1]).Pdiagonal(r,M[1],N[1])],seq([ZeroMatrix(r,r)
  ],i=1..nops(M)-1)]); for j from 2 to nops(M) do work:=
  <work|Matrix([seq([ZeroMatrix(r,r)],i=1..j-1),[Sforward(r)^(-L[j]
  ).Pdiagonal(r,M[j],N[j])],seq([ZeroMatrix(r,r)],i=j..nops(M)-1)])
  >; end; end;
```

```
PSdiagonal := proc(r, M, N, L)                                          (1.18)
    local work, i, j;
    work := Matrix([[`.`(Sforward(r)^(-L[1]), Pdiagonal(r, M[1], N[1]))],
    seq([LinearAlgebra:-ZeroMatrix(r, r)], i = 1..nops(M) - 1)]);
```

```
    for j from 2 to nops(M) do
        work := < work|Matrix([seq([LinearAlgebra:-ZeroMatrix(r, r)], i = 1 ..j - 1),
        [`.`(Sforward(r)^(-L[j]), Pdiagonal(r, M[j], N[j]))], seq([LinearAlgebra:-
        ZeroMatrix(r, r)], i = j ..nops(M) - 1)]) >
    end do
end proc
```

Combined to get the general form of the H matrix from Proposition 4.4. Requires, to make sense, that the gcd values for M and N match up and that the entries in L come from P_0. (The two procedures will be used indidually to save time).

```
> Hmatrix:=(r,M,N,L)->Yindeterminates(r,M)+PSdiagonal(r,M,N,L);
```

$$Hmatrix := (r, M, N, L) \mapsto Yindeterminates(r, M) + PSdiagonal(r, M, N, L) \tag{1.19}$$

```
> Hmatrix(5,[1,2,3],[4,3,2],[0,1,1]), Hmatrix(5,[1,1,2],[2,4,1],
  [0,0,0]);
```

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
y_{1,0,0,0} & y_{1,0,0,1} & y_{1,0,0,2} & y_{1,0,0,3} & y_{1,0,0,4} & 0 & 0 & 0 & 0 & \cdots \\
y_{1,0,1,0} & y_{1,0,1,1} & y_{1,0,1,2} & y_{1,0,1,3} & y_{1,0,1,4} & 0 & 0 & 0 & 1 & \cdots \\
y_{1,0,2,0} & y_{1,0,2,1} & y_{1,0,2,2} & y_{1,0,2,3} & y_{1,0,2,4} & 0 & 0 & 1 & 0 & \cdots \\
y_{1,0,3,0} & y_{1,0,3,1} & y_{1,0,3,2} & y_{1,0,3,3} & y_{1,0,3,4} & 0 & 1 & 0 & 0 & \cdots \\
y_{1,0,4,0} & y_{1,0,4,1} & y_{1,0,4,2} & y_{1,0,4,3} & y_{1,0,4,4} & 1 & 0 & 0 & 0 & \cdots \\
y_{2,0,0,0} & y_{2,0,0,1} & y_{2,0,0,2} & y_{2,0,0,3} & y_{2,0,0,4} & y_{2,1,0,0} & y_{2,1,0,1} & y_{2,1,0,2} & y_{2,1,0,3} & y_{2,} \cdots \\
y_{2,0,1,0} & y_{2,0,1,1} & y_{2,0,1,2} & y_{2,0,1,3} & y_{2,0,1,4} & y_{2,1,1,0} & y_{2,1,1,1} & y_{2,1,1,2} & y_{2,1,1,3} & y_{2,} \cdots \\
y_{2,0,2,0} & y_{2,0,2,1} & y_{2,0,2,2} & y_{2,0,2,3} & y_{2,0,2,4} & y_{2,1,2,0} & y_{2,1,2,1} & y_{2,1,2,2} & y_{2,1,2,3} & y_{2,} \cdots \\
y_{2,0,3,0} & y_{2,0,3,1} & y_{2,0,3,2} & y_{2,0,3,3} & y_{2,0,3,4} & y_{2,1,3,0} & y_{2,1,3,1} & y_{2,1,3,2} & y_{2,1,3,3} & y_{2,} \cdots \\
y_{2,0,4,0} & y_{2,0,4,1} & y_{2,0,4,2} & y_{2,0,4,3} & y_{2,0,4,4} & y_{2,1,4,0} & y_{2,1,4,1} & y_{2,1,4,2} & y_{2,1,4,3} & y_{2,} \cdots
\end{bmatrix}, \tag{1.20}$$

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
y_{1,0,0,0} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\
0 & y_{1,0,1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & y_{1,0,2,2} & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\
0 & 0 & 0 & y_{1,0,3,3} & 0 & 0 & 0 & 1 & 0 & \cdots \\
0 & 0 & 0 & 0 & y_{1,0,4,4} & 0 & 1 & 0 & 0 & \cdots \\
y_{2,0,0,0} & y_{2,0,0,1} & y_{2,0,0,2} & y_{2,0,0,3} & y_{2,0,0,4} & y_{2,1,0,0} & y_{2,1,0,1} & y_{2,1,0,2} & y_{2,1,0,3} & y_{2,} \cdots \\
y_{2,0,1,0} & y_{2,0,1,1} & y_{2,0,1,2} & y_{2,0,1,3} & y_{2,0,1,4} & y_{2,1,1,0} & y_{2,1,1,1} & y_{2,1,1,2} & y_{2,1,1,3} & y_{2,} \cdots \\
y_{2,0,2,0} & y_{2,0,2,1} & y_{2,0,2,2} & y_{2,0,2,3} & y_{2,0,2,4} & y_{2,1,2,0} & y_{2,1,2,1} & y_{2,1,2,2} & y_{2,1,2,3} & y_{2,} \cdots \\
y_{2,0,3,0} & y_{2,0,3,1} & y_{2,0,3,2} & y_{2,0,3,3} & y_{2,0,3,4} & y_{2,1,3,0} & y_{2,1,3,1} & y_{2,1,3,2} & y_{2,1,3,3} & y_{2,} \cdots \\
y_{2,0,4,0} & y_{2,0,4,1} & y_{2,0,4,2} & y_{2,0,4,3} & y_{2,0,4,4} & y_{2,1,4,0} & y_{2,1,4,1} & y_{2,1,4,2} & y_{2,1,4,3} & y_{2,} \cdots
\end{bmatrix}$$

Other tools

Compute all possible choices of M vectors of length k+1, all starting with 1.

```
> GetUniverse:=proc(r,k) local units,i,j,u,nextup,lastup,work;
  units:=[]; for i to r do if gcd(i,r)=1 then units:=[op(units),i];
  end; end; lastup:=[[1]];  for j to k do nextup:=[]; for work in
  lastup do for u in units do nextup:=[op(nextup),[op(work),u]];
  end; end; lastup:=nextup; end; return nextup; end;
```

$$GetUniverse := \mathbf{proc}(r, k) \qquad \textbf{(2.1)}$$

```
    local units, i, j, u, nextup, lastup, work;
    units := [ ];
    for i to r do if gcd(i, r) = 1 then units := [op(units), i] end if end do;
    lastup := [[1]];
    for j to k do
        nextup := [ ];
        for work in lastup do
            for u in units do
                nextup := [op(nextup), [op(work), u]]
```

```
            end do
        end do;
        lastup := nextup
    end do;
    return nextup
end proc
```

```
> GetUniverse(8,2);
```

$$[[1,1,1],[1,1,3],[1,1,5],[1,1,7],[1,3,1],[1,3,3],[1,3,5],[1,3,7],[1,5,1],[1,5,3],[1,5,5],[1,5,7],[1,7,1],[1,7,3],[1,7,5],[1,7,7]] \quad (2.2)$$

Compute the **W bar** vector [Definition 1.4]

```
> ReduceDegree:=proc(p,r) local work; work:=p; while degree(work)
  >=r do work:=quo(work,x^r,x)+rem(work,x^r,x); end; return sort
  (work); end;
```

$$ReduceDegree := \mathbf{proc}(p, r) \quad (2.3)$$

```
    local work;
    work := p;
    while r <= degree(work) do work := quo(work, x^r, x) + rem(work, x^r, x) end do;
    return sort(work)
end proc
```

```
> Wbar:=(r,M)->map(p->ReduceDegree(p,r),x*Ccompress(r,M).Pspecial
  (r,M).QLSadjacency(r,M).(IdentityMatrix(r*nops(M))-x*
  (IdentityMatrix(r*nops(M))-Pspecial(r,M)).QLSadjacency(r,M))^(-1)
  .Pspecial(r,M).Transpose(Ccompress(r,M)));
```

$$Wbar := (r, M) \mapsto map\left(p \mapsto ReduceDegree(p, r), (x \cdot Ccompress(r, M)) \cdot Pspecial(r, M) \cdot QLSadjacency(r, M) \cdot 1 \Big/ (LinearAlgebra:-IdentityMatrix(r \cdot nops(M)) - (x \cdot (LinearAlgebra:-IdentityMatrix(r \cdot nops(M)) - Pspecial(r, M))) \cdot QLSadjacency(r, M)) \cdot Pspecial(r, M) \cdot Ccompress(r, M)^{+}\right) \quad (2.4)$$

The complete invariant for ideal lattices. Will be used as a quick test in more advanced computations, to speed things up.

```
> IdealInvariant:=proc(r,L) local i; [seq(gcd(L[i+1]-L[i],r),i=1..
  nops(L)-1)]; end;
```

$$IdealInvariant := \mathbf{proc}(r, L) \quad (2.5)$$

```
    local i; [seq(gcd(L[i + 1] - L[i], r), i = 1 ..nops(L) - 1)]
end proc
```

Step through all possibilities of shifts allow by the P_0 set.

```
> getNext:=proc(r,L,A) local i,j,k; for i from nops(L) to 1 by -1
  do for j from L[i]+1 to r-1 do if A[i][j+1]=1 then return [op(L
  [1..i-1]),j,seq(0,k=i+1..nops(L))]; end; end;  end;  return [];
  end;
```

$$getNext := \mathbf{proc}(r, L, A) \quad (2.6)$$

```
    local i, j, k;
    for i from nops(L) by -1 to 1 do
        for j from L[i]+1 to r-1 do
            if A[i][j+1]=1 then
                return [op(L[1..i-1]), j, seq(0, k=i+1..nops(L))]
            end if
        end do
    end do;
    return [ ]
end proc
```

Normalizing up to trivial isomorphism

```
> modrinverse:=(r,x)->subs(msolve(x*y=1,r),y);
```

$$modrinverse := (r, x) \mapsto subs(msolve(x \cdot y = 1, r), y) \quad \textbf{(2.7)}$$

```
> overlayData:=proc(r,L,oldlist) local Ladjusted, i; Ladjusted:=
  map(x->x*modrinverse(r,L[1]) mod r,L);  for i to nops(oldlist) do
  if Ladjusted in oldlist[i] then return i; end; end; end;
```

$$overlayData := \textbf{proc}(r, L, oldlist) \quad \textbf{(2.8)}$$

```
    local Ladjusted, i;
    Ladjusted := map(x->x*modrinverse(r, L[1]) mod r, L);
    for i to nops(oldlist) do
        if Ladjusted in oldlist[i] then return i end if
    end do
end proc
```

Key tests

We now give tests for the properties II,IV,V,VI,VII from Section 7. They only work when the first parameter of M and N are 1 (or at least identical).

Tests II,IV,VII use the isolve facilities of Maple, but are sped ud by testing first with solve. Only if solutions over Q are found, we try for solutions over Z.

```
> testII:=proc(r,M,N) local i,j,UU,YY,PP0,DGM,DGN,LL,PP,sols,
  isols; if not is(IdealInvariant(r,M)=IdealInvariant(r,N)) then
  return false; end; UU:=theUnit(r,M); YY:=Yindeterminates(r,M);
  PP0:=findP0set(r,M); DGM:=DGshiftmatrix(r,M);  DGN:=DGshiftmatrix
  (r,N); LL:=[seq(0,j=1..nops(M))]; while(nops(LL)>0) do PP:=
  PSdiagonal(r,M,N,LL);  sols:=solve(convert((YY+PP).DGM-DGN.(YY+
  PP),set) union convert((YY+PP).UU-UU,set)); if not is(sols=NULL)
  then isols:=isolve(convert((YY+PP).DGM-DGN.(YY+PP),set) union
  convert((YY+PP).UU-UU,set)); if not is(isols=NULL) then return
  true; end; end; LL:=getNext(r,LL,PP0); end;  return false; end;
```

(3.1)

```
testII := proc(r, M, N)                                                    (3.1)
    local i, j, UU, YY, PP0, DGM, DGN, LL, PP, sols, isols;
    if not is(IdealInvariant(r, M) = IdealInvariant(r, N)) then return false end if;
    UU := theUnit(r, M);
    YY := Yindeterminates(r, M);
    PP0 := findP0set(r, M);
    DGM := DGshiftmatrix(r, M);
    DGN := DGshiftmatrix(r, N);
    LL := [seq(0, j = 1 ..nops(M))];
    while 0 < nops(LL) do
        PP := PSdiagonal(r, M, N, LL);
        sols := solve(convert(`.`(YY + PP, DGM) - `.`(DGN, YY + PP), set) union
        convert(`.`(YY + PP, UU) - UU, set));
        if not is(sols = NULL) then
            isols := isolve(convert(`.`(YY + PP, DGM) - `.`(DGN, YY + PP), set) union
            convert(`.`(YY + PP, UU) - UU, set));
            if not is(isols = NULL) then return true end if
        end if;
        LL := getNext(r, LL, PP0)
    end do;
    return false
end proc
```

```
> testIV:=
> proc(r,M,N) local i,j,YY,PP0,DGM,DGN,LL,PP,sols,isols; if not is
  (IdealInvariant(r,M)=IdealInvariant(r,N)) then return false; end;
  YY:=Yindeterminates(r,M); PP0:=findP0set(r,M); DGM:=DGshiftmatrix
  (r,M);  DGN:=DGshiftmatrix(r,N); LL:=[seq(0,j=1..nops(M))]; while
  (nops(LL)>0) do PP:=PSdiagonal(r,M,N,LL);  sols:=solve(convert(
  (YY+PP).DGM-DGN.(YY+PP),set)); if not is(sols=NULL) then isols:=
  isolve(convert((YY+PP).DGM-DGN.(YY+PP),set)); if not is(isols=
  NULL) then return true; end; end; LL:=getNext(r,LL,PP0); end;
  return false; end;
```

```
testIV := proc(r, M, N)                                                    (3.2)
    local i, j, YY, PP0, DGM, DGN, LL, PP, sols, isols;
    if not is(IdealInvariant(r, M) = IdealInvariant(r, N)) then return false end if;
    YY := Yindeterminates(r, M);
    PP0 := findP0set(r, M);
    DGM := DGshiftmatrix(r, M);
    DGN := DGshiftmatrix(r, N);
    LL := [seq(0, j = 1 ..nops(M))];
```

```
    while 0 < nops(LL) do
        PP := PSdiagonal(r, M, N, LL);
        sols := solve(convert(`.`(YY + PP, DGM) - `.`(DGN, YY + PP), set));
        if not is(sols = NULL) then
            isols := isolve(convert(`.`(YY + PP, DGM) - `.`(DGN, YY + PP), set));
            if not is(isols = NULL) then return true end if
        end if;
        LL := getNext(r, LL, PP0)
    end do;
    return false
end proc
```

```
> testV:=(r,M,N)->is(convert(Wbar(r,M),listlist)=convert(Wbar(r,
  N),listlist));
```

$$testV := (r, M, N) \mapsto is(convert(Wbar(r, M), listlist) = convert(Wbar(r, N), listlist)) \quad (3.3)$$

```
> testVI:=proc(r,M,N) if nops(convert(M,set))<nops(M) or nops
  (convert(N,set))<nops(M) then return is(M=N); else return is
  (IdealInvariant(r,M)=IdealInvariant(r,N)); end; end;
```

$$testVI := \mathbf{proc}(r, M, N) \quad (3.4)$$

```
    if nops(convert(M, set)) < nops(M) or nops(convert(N, set)) < nops(M) then
        return is(M = N)
    else
        return is(IdealInvariant(r, M) = IdealInvariant(r, N))
    end if
end proc
```

```
> testVII:=
> proc(r,M,N) local i,j,k,YY,PP0,DGM,DGN,LL,PP,sols,isols,
  shortcut; if not is(IdealInvariant(r,M)=IdealInvariant(r,N)) then
  return false; end; YY:=Yindeterminates(r,M); shortcut:={seq(seq(y
  [i,j,0,0],j=0..i-1),i=1..nops(M)-1)} union {seq(seq(seq(y[i,j,k,
  0],j=0..i-1),i=1..nops(M)-1),k=1..r-1)};DGM:=DGshiftmatrix(r,M);
  DGN:=DGshiftmatrix(r,N); LL:=[seq(0,j=1..nops(M))];  PP:=
  PSdiagonal(r,M,N,LL);  sols:=solve(convert((YY+PP).DGM-DGN.(YY+
  PP),set) union shortcut); if not is(sols=NULL) then isols:=isolve
  (convert((YY+PP).DGM-DGN.(YY+PP),set) union shortcut); if not is
  (isols=NULL) then return true; end; end; return false; end;
```

$$testVII := \mathbf{proc}(r, M, N) \quad (3.5)$$

```
    local i, j, k, YY, PP0, DGM, DGN, LL, PP, sols, isols, shortcut;
    if not is(IdealInvariant(r, M) = IdealInvariant(r, N)) then return false end if;
    YY := Yindeterminates(r, M);
    shortcut := {seq(seq(y[i, j, 0, 0], j = 0 ..i - 1), i = 1 ..nops(M) - 1)} union
```

```
    {seq(seq(seq(y[i,j,k,0],j=0..i-1),i=1..nops(M)-1),k=1..r-1)};
    DGM := DGshiftmatrix(r,M);
    DGN := DGshiftmatrix(r,N);
    LL := [seq(0,j=1..nops(M))];
    PP := PSdiagonal(r,M,N,LL);
    sols := solve(convert(`.`(YY+PP,DGM) - `.`(DGN,YY+PP),set) union
    shortcut);
    if not is(sols=NULL) then
        isols := isolve(convert(`.`(YY+PP,DGM) - `.`(DGN,YY+PP),set) union
        shortcut);
        if not is(isols=NULL) then return true end if
    end if;
    return false
end proc
```

Systematic testing

We define here two procedures for systematic studies of our equivalence relations, singleClassify and multipleClassify. Both come with an optional verboselevel parameter which controls output in the following way. Level 1 (default) just returns the number of elements in each class, as a list where x entries of value y is returned as [y,x]. Level 2 also prints those partitions that have more than one element. Level 3 prints all partitions. For the multipleClassify there are also levels -1, -2, -3 that suppress the output from lower dimensions and gives data just for the highest one as governed by the absolute value of the verboselevel.

The singleClassify procedure divides the universe of k+1-vectors starting with 1 into partitions using a test chosen amongst the ones defined above.

```
> singleClassify:=proc(r,k,test::procedure,verboselevel:=1) local
  i,mylist; mylist:=[ListTools[Categorize]((x,y)->test(r,x,y),
  GetUniverse(r,k))];  mylist:=sort(map(x->[-nops(x),x],mylist));
  if(verboselevel=2) then for i in mylist do if(i[1]<-1) then print
  (i[2]); end; end; end; if(verboselevel=3) then for i in mylist do
  print(i[2]); end; end; return ListTools[Collect](map(x->-x[1],
  mylist)); end;
```

```
singleClassify := proc(r,k,test::procedure,verboselevel := 1)                    (4.1)
    local i,mylist;
    mylist := [ListTools[Categorize]((x,y)->test(r,x,y),GetUniverse(r,k))];
    mylist := sort(map(x->[-nops(x),x],mylist));
    if verboselevel=2 then
        for i in mylist do if i[1]<-1 then print(i[2]) end if end do
    end if;
    if verboselevel=3 then for i in mylist do print(i[2]) end do end if;
```

```
    return ListTools[Collect](map(x→ − x[1], mylist))
end proc
```

The compareClassification procedure compares the partitions obtained by two supplied test procedures to the universe of all choices of vectors of lenght k+1 describing a quantum lense space with r given. If they agree, 0 is returned. If the first is found to be coarser than the second, -1 and an example where the realtions differ is returned. If the first is found to be finer than the second, 1 and an example where the relations differ is returned. The order of the procedures matters significantly for run times; the first procedure should be the fastest one. Also, if the tests are incomparable, a -1 result will always be reported.

```
> compareClassification:=proc(r,k,fast::procedure,slow::procedure)
  local onePartition,i,j; onePartition:=[ListTools[Categorize]((x,
  y)->fast(r,x,y),GetUniverse(r,k))]; for i to nops(onePartition)
  do for j to nops(onePartition[i])-1 do if(not slow(r,onePartition
  [i][j],onePartition[i][j+1])) then return -1,onePartition[i][j],
  onePartition[i][j+1]; end; end; end; for i to nops(onePartition)
  do for j from i+1 to nops(onePartition) do if slow(r,onePartition
  [i][1],onePartition[j][1]) then return 1, onePartition[i][1],
  onePartition[j][1]; end; end; end; return 0; end;
```

```
compareClassification := proc(r, k, fast::procedure, slow::procedure)                    (4.2)
    local onePartition, i, j;
    onePartition := [ListTools[Categorize]( (x, y)→fast(r, x, y), GetUniverse(r, k) )];
    for i to nops(onePartition) do
        for j to nops(onePartition[i]) − 1 do
            if not slow(r, onePartition[i][j], onePartition[i][j + 1]) then
                return − 1, onePartition[i][j], onePartition[i][j + 1]
            end if
        end do
    end do;
    for i to nops(onePartition) do
        for j from i + 1 to nops(onePartition) do
            if slow(r, onePartition[i][1], onePartition[j][1]) then
                return 1, onePartition[i][1], onePartition[j][1]
            end if
        end do
    end do;
    return 0
end proc
```

The multipleClassify procedure divides the universe of k+1-vectors starting with 1 into partitions using a test chosen from the list above. It goes through all shorter vectors as well. and uses the results to speed up computation. This is possible because if [r_0,...,r_k+1] and [s_0,...,s_k+1] agree wrt.

some test, then so do [r_0,...,r_k] and [s_0,...,s_k], and so do [r_1,...,r_k+1] and [s_1,...,s_k+1].
when k>1 this is often faster than running singleClassify.

```
> multipleClassify:=proc(r,k,test::procedure,verboselevel:=1)
  local subdivs,i,mytable,range,halfway,mylist,sel; subdivs:=Array
  (1..k);  for i to k do if(i=1) then subdivs[1]:=[ListTools
  [Categorize]((x,y)->test(r,x,y),GetUniverse(r,1))]; else
  mytable:=ListTools[Classify](x->[overlayData(r,x[1..-2],subdivs
  [i-1]),overlayData(r,x[2..-1],subdivs[i-1])],GetUniverse(r,i));
  subdivs[i]:=[]; range:=indices(mytable); for halfway in range do
  subdivs[i]:=[op(subdivs[i]),ListTools[Categorize]((x,y)->test(r,
  x,y),mytable[op(halfway)])];  end;  end; if(verboselevel>0) then
  print(i); end; mylist:=sort(map(x->[-nops(x),x],subdivs[i])); if
  (verboselevel>1 or (verboselevel<-1 and i=k)) then for sel in
  mylist do if(-sel[1]>1 or verboselevel>2 or (verboselevel<-2 and
  i=k)) then print(sel[2]);  end; end; end; if(verboselevel>0 or i=
  k) then print(ListTools[Collect](map(x->-x[1],mylist))); end;
  end; end;
```

$$multipleClassify := \mathbf{proc}(r, k, test::procedure, verboselevel := 1) \tag{4.3}$$

```
local subdivs, i, mytable, range, halfway, mylist, sel;
subdivs := Array(1 ..k);
for i to k do
    if i = 1 then
        subdivs[1] := [ListTools[Categorize]((x, y)→test(r, x, y), GetUniverse(r, 1))]
    else
        mytable := ListTools[Classify](x→[overlayData(r, x[1 .. -2], subdivs[i - 1
        ]), overlayData(r, x[2 .. -1], subdivs[i - 1])], GetUniverse(r, i));
        subdivs[i] := [];
        range := indices(mytable);
        for halfway in range do
            subdivs[i] := [op(subdivs[i]), ListTools[Categorize]((x, y)→test(r, x, y),
            mytable[op(halfway)])]
        end do
    end if;
    if 0 < verboselevel then print(i) end if;
    mylist := sort(map(x→[-nops(x), x], subdivs[i]));
    if 1 < verboselevel or verboselevel < -1 and i = k then
        for sel in mylist do
            if 1 < -sel[1] or 2 < verboselevel or verboselevel < -2 and i = k
            then
                print(sel[2])
            end if
```

```
            end do
        end if;
        if 0 < verboselevel or i = k then
            print(ListTools[Collect](map(x→ − x[1], mylist)))
        end if
    end do
end proc
```

Examples of use

We first use test VI to partition the universe of vectors [1,m_1,m_2] relative to r=5. Note how the verboselevel controls the amount of detail in the output.

```
> singleClassify(5,2,testVI);
```

$$[[1, 10], [6, 1]] \quad \textbf{(5.1)}$$

```
> singleClassify(5,2,testVI,2);
```

$$[[1, 2, 3], [1, 2, 4], [1, 3, 2], [1, 3, 4], [1, 4, 2], [1, 4, 3]]$$

$$[[1, 10], [6, 1]] \quad \textbf{(5.2)}$$

```
> singleClassify(5,2,testVI,3);
```

$$[[1, 2, 3], [1, 2, 4], [1, 3, 2], [1, 3, 4], [1, 4, 2], [1, 4, 3]]$$

$$[[1, 1, 1]]$$
$$[[1, 1, 2]]$$
$$[[1, 1, 3]]$$
$$[[1, 1, 4]]$$
$$[[1, 2, 1]]$$
$$[[1, 2, 2]]$$
$$[[1, 3, 1]]$$
$$[[1, 3, 3]]$$
$$[[1, 4, 1]]$$
$$[[1, 4, 4]]$$
$$[[1, 10], [6, 1]] \quad \textbf{(5.3)}$$

We note the dramatic variation in time usage amongst the five procedures listed above.

```
> map(x->time(singleClassify(5,2,x)),[testII,testIV,testV,testVI,
  testVII]);
```

$$[18.851, 22.260, 4.173, 0.003, 7.011] \quad \textbf{(5.4)}$$

Because the multipleClassify procedure pre-partitions the studied universe using shorter vectors, it may give substantial speedup and might be convenient even when one cares only about the last dimension. One makes the multipleClassify procedure behave like the singleClassify prodecure by using negative verboselevels.

```
> time(singleClassify(5,3,testV,2));
```

$$[[1, 2, 4, 3], [1, 3, 4, 2], [1, 4, 2, 3], [1, 4, 3, 2]]$$

[[1, 2, 3, 1], [1, 3, 2, 1]]
[[1, 2, 3, 4], [1, 3, 2, 4]]
[[1, 2, 4, 1], [1, 4, 2, 1]]
[[1, 3, 4, 1], [1, 4, 3, 1]]
183.190 **(5.5)**

```
> time(multipleClassify(5,3,testV,-2));
```

[[1, 2, 4, 3], [1, 3, 4, 2], [1, 4, 2, 3], [1, 4, 3, 2]]
[[1, 2, 3, 1], [1, 3, 2, 1]]
[[1, 2, 3, 4], [1, 3, 2, 4]]
[[1, 2, 4, 1], [1, 4, 2, 1]]
[[1, 3, 4, 1], [1, 4, 3, 1]]
[[1, 52], [2, 4], [4, 1]]
4.392 **(5.6)**

Showing more data requires very little extra time.

```
> time(multipleClassify(5,3,testV,2));
```

1
[[1, 2], [1, 3], [1, 4]]
[[1, 1], [3, 1]]
2
[[1, 2, 3], [1, 2, 4], [1, 3, 2], [1, 3, 4], [1, 4, 2], [1, 4, 3]]
[[1, 10], [6, 1]]
3
[[1, 2, 4, 3], [1, 3, 4, 2], [1, 4, 2, 3], [1, 4, 3, 2]]
[[1, 2, 3, 1], [1, 3, 2, 1]]
[[1, 2, 3, 4], [1, 3, 2, 4]]
[[1, 2, 4, 1], [1, 4, 2, 1]]
[[1, 3, 4, 1], [1, 4, 3, 1]]
[[1, 52], [2, 4], [4, 1]]
4.407 **(5.7)**

Note that tests V and VII disagree at k=3:

```
> multipleClassify(5,3,testVII,2);
```

1
[[1, 2], [1, 3], [1, 4]]
[[1, 1], [3, 1]]
2
[[1, 2, 3], [1, 2, 4], [1, 3, 2], [1, 3, 4], [1, 4, 2], [1, 4, 3]]
[[1, 10], [6, 1]]
3

[[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 3, 4, 2], [1, 4, 2, 3], [1, 4, 3, 2]]
[[1, 2, 3, 1], [1, 3, 2, 1]]
[[1, 2, 4, 1], [1, 4, 2, 1]]
[[1, 3, 4, 1], [1, 4, 3, 1]]
[[1, 52], [2, 3], [6, 1]] **(5.8)**

One can use compareClassification to compare two tests on a universe. It speeds up computation to order the tests so the fastest comes first:

```
> compareClassification(5,3,testV,testVII);
```

1, [1, 2, 3, 4], [1, 2, 4, 3] **(5.9)**

This output says that the pair found is equivalent wrt. V but not wrt. VII.

Performed experiments and findings

These are the experiments reported on in the paper's last section. Combined run time on a standard laptop was approximately 13 hours.

Comparing V, VI, VII at k=1 (corresponding to d=5) for composite r<=32

```
> for rr to 32 do if not isprime(rr) then print(rr,
  compareClassification(rr,1,testVI,testV)); end; end;
```

1, 0
4, 0
6, 0
8, 0
9, 0
10, 0
12, 0
14, 0
15, 0
16, 0
18, 0
20, 0
21, 0
22, 0
24, 0
25, 0
26, 0
27, 0
28, 0
30, 0
32, 0 **(6.1)**

```
> for rr to 32 do if not isprime(rr) then print(rr,
```

```
compareClassification(rr,1,testVI,testVII)); end; end;
```

1, 0

4, 0

6, 0

8, 0

9, 0

10, 0

12, 0

14, 0

15, 0

16, 0

18, 0

20, 0

21, 0

22, 0

24, 0

25, 0

26, 0

27, 0

28, 0

30, 0

32, 0 **(6.2)**

Finding differences between V, VI, VII at k<=3 (i.e. d<=7) for r=5 (see Example 7.3)

```
> for dd to 3 do dd,compareClassification(5,dd,testVI,testV); end;
```

1, 0

2, 0

3, −1, [1, 2, 3, 4], [1, 2, 4, 3] **(6.3)**

```
> testV(5,[1,2,3,4], [1,2,4,3]); testVI(5,[1,2,3,4], [1,2,4,3]);
  testVII(5,[1,2,3,4], [1,2,4,3]);
```

false

true

true **(6.4)**

```
> for dd to 3 do dd,compareClassification(5,dd,testVI,testVII);
  end;
```

1, 0

2, 0

3, 1, [1, 2, 3, 1], [1, 3, 2, 1] **(6.5)**

```
> testV(5,[1,2,3,1], [1,3,2,1]); testVI(5,[1,2,3,1], [1,3,2,1]);
  testVII(5,[1,2,3,1], [1,3,2,1]);
```

true

false

true **(6.6)**

Complete analyses at r=5 and r=8 as far as practical (k=6, d=13 for r=5 and k=5, d=11 for r=8). Note these are done with VII even though we do not know that VII agrees with the more relevant II og IV.

```
> multipleClassify(5,6,testVII,2);
```

1

[[1, 2], [1, 3], [1, 4]]

[[1, 1], [3, 1]]

2

[[1, 2, 3], [1, 2, 4], [1, 3, 2], [1, 3, 4], [1, 4, 2], [1, 4, 3]]

[[1, 10], [6, 1]]

3

[[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 3, 4, 2], [1, 4, 2, 3], [1, 4, 3, 2]]

[[1, 2, 3, 1], [1, 3, 2, 1]]

[[1, 2, 4, 1], [1, 4, 2, 1]]

[[1, 3, 4, 1], [1, 4, 3, 1]]

[[1, 52], [2, 3], [6, 1]]

4

[[1, 2, 3, 4, 1], [1, 2, 4, 3, 1], [1, 3, 2, 4, 1], [1, 3, 4, 2, 1], [1, 4, 2, 3, 1], [1, 4, 3, 2, 1]]

[[1, 2, 3, 1, 4], [1, 3, 2, 1, 4]]

[[1, 2, 3, 4, 2], [1, 2, 4, 3, 2]]

[[1, 2, 4, 1, 3], [1, 4, 2, 1, 3]]

[[1, 3, 2, 4, 3], [1, 3, 4, 2, 3]]

[[1, 3, 4, 1, 2], [1, 4, 3, 1, 2]]

[[1, 4, 2, 3, 4], [1, 4, 3, 2, 4]]

[[1, 238], [2, 6], [6, 1]]

5

[[1, 2, 3, 4, 1, 2], [1, 2, 4, 3, 1, 2], [1, 3, 2, 4, 1, 3], [1, 3, 4, 2, 1, 3], [1, 4, 2, 3, 1, 4], [1, 4, 3, 2, 1, 4]]

[[1, 2, 3, 1, 4, 2], [1, 3, 2, 1, 4, 3]]

[[1, 2, 3, 1, 4, 3], [1, 3, 2, 1, 4, 2]]

[[1, 2, 3, 4, 1, 3], [1, 3, 2, 4, 1, 2]]

[[1, 2, 3, 4, 2, 1], [1, 2, 4, 3, 2, 1]]

[[1, 2, 4, 3, 1, 4], [1, 4, 3, 2, 1, 3]]

[[1, 3, 2, 4, 3, 1], [1, 3, 4, 2, 3, 1]]

[[1, 3, 4, 2, 1, 4], [1, 4, 2, 3, 1, 2]]

[[1, 4, 2, 3, 4, 1], [1, 4, 3, 2, 4, 1]]

[[1, 1002], [2, 8], [6, 1]]

6

[[1, 2, 3, 4, 1, 2, 3], [1, 2, 4, 3, 1, 2, 4], [1, 3, 2, 4, 1, 3, 2], [1, 3, 4, 2, 1, 3, 4], [1, 4, 2, 3, 1, 4, 2], [1, 4, 3, 2, 1, 4, 3]]

[[1, 2, 3, 1, 4, 2, 3], [1, 3, 2, 1, 4, 3, 2]]

[[1, 2, 3, 1, 4, 3, 2], [1, 3, 2, 1, 4, 2, 3]]

[[1, 2, 3, 4, 1, 2, 4], [1, 3, 4, 2, 1, 3, 2]]

[[1, 2, 3, 4, 1, 3, 2], [1, 3, 2, 4, 1, 2, 3]]

[[1, 2, 4, 3, 1, 2, 3], [1, 3, 2, 4, 1, 3, 4]]

[[1, 2, 4, 3, 1, 4, 2], [1, 4, 3, 2, 1, 3, 4]]

[[1, 3, 4, 2, 1, 4, 3], [1, 4, 2, 3, 1, 2, 4]]

[[1, 4, 2, 3, 1, 4, 3], [1, 4, 3, 2, 1, 4, 2]]

[[1, 4, 2, 3, 4, 1, 2], [1, 4, 3, 2, 4, 1, 3]]

[[1, 4, 2, 3, 4, 1, 3], [1, 4, 3, 2, 4, 1, 2]]

[[1, 4070], [2, 10], [6, 1]] **(6.7)**

```
> multipleClassify(8,5,testVII,2);
```

1

[[1, 3], [1, 7]]

[[1, 2], [2, 1]]

2

[[1, 3, 5], [1, 7, 5]]

[[1, 3, 7], [1, 7, 3]]

[[1, 5, 3], [1, 5, 7]]

[[1, 10], [2, 3]]

3

[[1, 3, 5, 7], [1, 7, 5, 3]]

[[1, 3, 7, 1], [1, 7, 3, 1]]

[[1, 3, 7, 5], [1, 7, 3, 5]]

[[1, 5, 3, 7], [1, 5, 7, 3]]

[[1, 56], [2, 4]]

4

[[1, 3, 5, 7, 1], [1, 7, 5, 3, 1]]

[[1, 3, 7, 1, 5], [1, 7, 3, 1, 5]]

[[1, 3, 7, 5, 1], [1, 7, 3, 5, 1]]

[[1, 5, 3, 7, 1], [1, 5, 7, 3, 1]]

[[1, 5, 3, 7, 5], [1, 5, 7, 3, 5]]

[[1, 246], [2, 5]]

5

[[1, 3, 5, 7, 1, 3], [1, 7, 5, 3, 1, 7]]
[[1, 3, 7, 1, 5, 3], [1, 7, 3, 1, 5, 7]]
[[1, 3, 7, 1, 5, 7], [1, 7, 3, 1, 5, 3]]
[[1, 3, 7, 5, 1, 3], [1, 7, 3, 5, 1, 7]]
[[1, 3, 7, 5, 1, 7], [1, 7, 3, 5, 1, 3]]
[[1, 5, 3, 7, 1, 5], [1, 5, 7, 3, 1, 5]]
[[1, 5, 3, 7, 5, 1], [1, 5, 7, 3, 5, 1]]
[[1, 1010], [2, 7]] **(6.8)**

Located differences between II and VII. (Example 7.3, see discussion after Conjecture 7.4)

```
> testVII(5,[1,3,4,1,2,3],[1,4,3,1,2,4]); testII(5,[1,3,4,1,2,3],
  [1,4,3,1,2,4]);
```

false
true **(6.9)**

```
> testVII(5,[1,2,4,1,3,2],[1,4,2,1,3,4]); testII(5,[1,2,4,1,3,2],
  [1,4,2,1,3,4]);
```

false
true **(6.10)**