```
with(LinearAlgebra) :
with(QuillenSuslin) :
with(ListTools) :
with(MatrixPolynomialAlgebra) :
with(combinat) :
```

1.primitive decomposition algorithm

```
> gcdvector :=proc (L)
    # this algo aims to compute the g.c.d. of a list of given polynomials;
    # L is a list of polynomials;
    local n, g, i;
    n := numelems(L);
    g := L[1];
    if n = 1 then
        return g;
    else
        for i from 2 to n do
            g := gcd(g, L[i]);
        end do;
        return g;
    end if;
  end proc;
```

```
gcdvector := proc(L)                                                    (1.1)
   local n, g, i;
   n := numelems(L);
   g := L[1];
   if n = 1 then
      return g
   else
      for i from 2 to n do g := gcd(g, L[i]) end do; return g
   end if
end proc
```

```
> modpx :=proc (M :: Matrix, p, xvar)

    # this algo outputs a matrix whose elements mod a polynomial of a single
    variable xvar;
    # M a matrix;
    # p a polynomial of variable xvar;
    # xvar a variable;
    local m, n, i, j, A;
    m := RowDimension(M);
    n := ColumnDimension(M);
    A := Matrix(m, n);
    for i from 1 to m do
        for j from 1 to n do
```

```
                A[i, j] := rem(M[i, j], p, xvar);
            end do;
        end do ;
        return A;
    end proc;
```
modpx := proc(M::Matrix, p, xvar) **(1.2)**
```
    local m, n, i, j, A;
    m := LinearAlgebra:-RowDimension(M);
    n := LinearAlgebra:-ColumnDimension(M);
    A := Matrix(m, n);
    for i to m do for j to n do A[i, j] := rem(M[i, j], p, xvar) end do end do;
    return A
end proc
```
```
> factorirrfactors :=proc(p)
        # factor a polynomial into irreducible parts;
        # p a polynomial;
        local L, LL, l, s, i, j;
        L := factors(p);
        l := numelems(L[2]);
        LL := [ ];
        for i from 1 to l do
            s := L[2][i][2];
            for j from 1 to s do
                LL := [op(LL), L[2][i][1]];
            end do;
        end do;
        return LL;
    end proc;
```
factorirrfactors := proc(p) **(1.3)**
```
    local L, LL, l, s, i, j;
    L := factors(p);
    l := numelems(L[2]);
    LL := [ ];
    for i to l do
        s := L[2][i][2]; for j to s do LL := [op(LL), L[2][i][1]] end do
    end do;
    return LL
end proc
```
```
> diagdenomcol :=proc(M)
        # extract the lcm of each column elements' denom;
        # M a matrix;
        local n, m, den, i, j, ll;
        m := RowDimension(M);
```

```
    n := ColumnDimension(M);
    den := [ ];
    for i from 1 to n do
        l1 := denom(M[1, i]);
        for j from 1 to m do
            l1 := lcm(l1, denom(M[j, i]));
        end do;
        den := [op(den), l1];
    end do;
    return DiagonalMatrix(den);
end proc;
```

diagdenomcol := proc(M) **(1.4)**

```
local n, m, den, i, j, l1;
m := LinearAlgebra:-RowDimension(M);
n := LinearAlgebra:-ColumnDimension(M);
den := [ ];
for i to n do
    l1 := denom(M[1, i]);
    for j to m do l1 := lcm(l1, denom(M[j, i])) end do;
    den := [op(den), l1]
end do;
return LinearAlgebra:-DiagonalMatrix(den)
end proc
```

```
> checkzerorow := proc(M :: Matrix, row)

    # Check if a matrix has zero rows, 'row' returns the corresponding row
    number if there are zero row with extries all 0
    local m, n, i, j;
    m := RowDimension(M);
    n := ColumnDimension(M);
    for i from row to m do
        j := 1;
        while M[i, j] = 0 do
            if j < n then
                j := j + 1;
            else
                return i;
            end if ;
        end do;
    end do;
    return 0;
end proc;
```

checkzerorow := proc(M::Matrix, row) **(1.5)**

```
local m, n, i, j;
```

```
    m := LinearAlgebra:-RowDimension(M);
    n := LinearAlgebra:-ColumnDimension(M);
    for i from row to m do
        j := 1;
        while M[i, j] = 0 do
            if j < n then j := j + 1 else return i end if
        end do
    end do;
    return 0
end proc
```

```
> checkzerorowfromrc := proc(M :: Matrix, row, r, c)
    # Starting from the r'th row and c'th column of matrix M, check whether the
    submatrix at the lower right corner of M has a zero row, 'row' returns the
    corresponding row number when there is a row in which all elements are 0
     local m, n, i, j;
     m := RowDimension(M);
     n := ColumnDimension(M);
     for i from r to m do
         j := c;
         while M[i, j] = 0 do
             if j < n then
                 j := j + 1;
             else
                 row := i;
                 return true;
             end if ;
         end do;
     end do;
     return false;
  end proc;
```

```
checkzerorowfromrc := proc(M::Matrix, row, r, c)                    (1.6)
    local m, n, i, j;
    m := LinearAlgebra:-RowDimension(M);
    n := LinearAlgebra:-ColumnDimension(M);
    for i from r to m do
        j := c;
        while M[i, j] = 0 do
            if j < n then j := j + 1 else row := i; return true end if
        end do
    end do;
    return false
end proc
```

```
> findcolumn :=proc(M, r, c)
    # Starting from the r'th row and c'th column of matrix M, check whether the
    submatrix at the lower right corner of M has a column which has nonzero
    element,if exists, return the corresponding column number.
     local m, n, i, j;
     m := RowDimension(M);
     n := ColumnDimension(M);
     for j from c to n do
         for i from r to m do
             if M[i, j] ≠ 0 then
                 return j;
             end if;
         end do;
     end do;
     return false;
  end proc;
findcolumn := proc(M, r, c)                                            (1.7)
   local m, n, i, j;
   m := LinearAlgebra:-RowDimension(M);
   n := LinearAlgebra:-ColumnDimension(M);
   for j from c to n do
      for i from r to m do
         if M[i, j]<>0 then return j end if
      end do
   end do;
   return false
end proc
> degree00 :=proc(f, vv)

    # return the degree of a poly f with respect to the variable vv, return +
    ∞ if the degree is less than 0.
     local d;
     d := degree(f, vv);
     if d < 0 then
         d :=+∞;
     end if;
     return d;
  end proc;
degree00 := proc(f, vv)                                                (1.8)
   local d;
   d := degree(f, vv); if d < 0 then d := infinity end if; return d
end proc
> interchangerow :=proc(L, R, Rmod, r, j0, yvar)
```

```
# The input matrix L and R are required to make M = LR. The returned M' is
the result of swapping some rows of the matrix M so that the element
position at the j0th column and rth row of M has the smallest degree in the
current column.
 local m, n, LL, i, k, mini, D;
 m := RowDimension(R);
 n := ColumnDimension(R);
 LL := [ ];
 k := m - r + 1;
 for i from r to m do
     LL := [op(LL), degree00(Rmod[i, j0], yvar)];
 end do;
 mini := LL[1];
 for i from 1 to k do
     if LL[i] < mini then
         mini := LL[i];
     end if;
 end do;
 i := Search(mini, LL);
 D := RowOperation(IdentityMatrix(m), [r, r+ i-1]);
 return L·D, D^-1·R;
end proc;
```

interchangerow := proc(L, R, Rmod, r, j0, yvar) **(1.9)**

```
   local m, n, LL, i, k, mini, D;
   m := LinearAlgebra:-RowDimension(R);
   n := LinearAlgebra:-ColumnDimension(R);
   LL := [ ];
   k := m - r + 1;
   for i from r to m do LL := [op(LL), degree00(Rmod[i, j0], yvar)] end do;
   mini := LL[1];
   for i to k do if LL[i] < mini then mini := LL[i] end if end do;
   i := ListTools:-Search(mini, LL);
   D := LinearAlgebra:-RowOperation(LinearAlgebra:-IdentityMatrix(m), [r, r
   + i - 1]);
   return Typesetting:-delayDotProduct(L, D), Typesetting:-delayDotProduct(1
   /D, R)
end proc
```

```
> euclidcolumnreduce :=proc(L, R, Rmod, r, c, p, xvar, yvar)

   # Perform column Gauss reduction on the matrix and return L, R such that M
   = LR

    local m, n, i, k, a1, b1, bb, ss, tt, aa, q1, r1, qq, rr, E, DD;
```

```
    m := RowDimension(R);
    n := ColumnDimension(R);
    al := [ ];
    k := m − r + 1;
    for i from r to m do
        al := [op(al), Rmod[i, c]];
    end do;
    bb := lcoeff(al[1], yvar);
    gcdex(bb, p, xvar, 'ss', 'tt');
    aa := ss·al[1];
    aa := rem(aa, p, xvar);
    ql := [ ];
    rl := [ ];
    for i from 2 to k do
        rr := rem(al[i], aa, yvar, 'qq');
        rl := [op(rl), rr];
        ql := [op(ql), qq];
    end do;
    E := IdentityMatrix(k);
    for i from 2 to k do
        E := RowOperation(E, [i, 1], -ss·ql[i − 1]) ;
    end do;
    DD := DiagonalMatrix([IdentityMatrix(r − 1), E]);
    return L • DD^-1, DD • R;
end proc;
```

euclidcolumnreduce := proc(L, R, Rmod, r, c, p, xvar, yvar) **(1.10)**

```
  local m, n, i, k, al, bl, bb, ss, tt, aa, ql, rl, qq, rr, E, DD;
  m := LinearAlgebra:-RowDimension(R);
  n := LinearAlgebra:-ColumnDimension(R);
  al := [ ];
  k := m − r + 1;
  for i from r to m do al := [op(al), Rmod[i, c]] end do;
  bb := lcoeff(al[1], yvar);
  gcdex(bb, p, xvar, 'ss', 'tt');
  aa := ss*al[1];
  aa := rem(aa, p, xvar);
  ql := [ ];
  rl := [ ];
  for i from 2 to k do
     rr := rem(al[i], aa, yvar, 'qq'); rl := [op(rl), rr]; ql := [op(ql),
     qq]
  end do;
  E := LinearAlgebra:-IdentityMatrix(k);
```

```
    for i from 2 to k do
        E := LinearAlgebra:-RowOperation(E, [i, 1], - ss*ql[i - 1])
    end do;
    DD := LinearAlgebra:-DiagonalMatrix([LinearAlgebra:-
    IdentityMatrix(r - 1), E]);
    return Typesetting:-delayDotProduct(L, 1 / DD), Typesetting:-
    delayDotProduct(DD, R)
end proc
> judgezeroele :=proc(R, r, c)

     #Determine whether there is a non-zero element in the cth column of matrix
     R from the rth row to the end of this column. If there is a non-zero
     element, return 0, otherwise return 1

      local i, m;
      m := RowDimension(R);
      for i from r + 1 to m do
          if R[i, c] ≠ 0 then
              return 0;
          end if;
      end do;
      return 1;
  end proc;
judgezeroele := proc(R, r, c)                                             (1.11)
    local i, m;
    m := LinearAlgebra:-RowDimension(R);
    for i from r + 1 to m do if R[i, c] <> 0 then return 0 end if end do;
    return 1
end proc
> factorsingleirrfactor :=proc(L, R, p, xvar, yvar)


     #The procedure for performing a primitive decomposition of the matrix M
     with respect to a single irreducible polynomial

      local m, n, i0, j0, LL, RR, Rmod, r1, D0;
      m := RowDimension(R);
      n := ColumnDimension(R);
      i0 := 1;
      j0 := 1;
      LL := L;
      RR := R;
      Rmod := modpx(RR, p, xvar);
      r1 := checkzerorow(Rmod, i0);
```

```
        D0 := IdentityMatrix(m);
        while r1 = 0 do
            j0 := findcolumn(Rmod, i0, j0);
            while judgezeroele(Rmod, i0, j0) = 0 do
                LL, RR := interchangerow(LL, RR, Rmod, i0, j0, yvar);
                Rmod := modpx(RR, p, xvar);
                LL, RR := euclidcolumnreduce(LL, RR, Rmod, i0, j0, p, xvar, yvar);
                Rmod := modpx(RR, p, xvar);
            end do;
            i0 := i0 + 1;
            j0 := j0 + 1;
            r1 := checkzerorow(Rmod, i0);
        end do;
        D0 := RowOperation(D0, r1, p);
        RR := D0^(-1) · RR;
        return simplify(LL · D0), simplify(RR);
    end proc;
factorsingleirrfactor := proc(L, R, p, xvar, yvar)                                    (1.12)
    local m, n, i0, j0, LL, RR, Rmod, r1, D0;
    m := LinearAlgebra:-RowDimension(R);
    n := LinearAlgebra:-ColumnDimension(R);
    i0 := 1;
    j0 := 1;
    LL := L;
    RR := R;
    Rmod := modpx(RR, p, xvar);
    r1 := checkzerorow(Rmod, i0);
    D0 := LinearAlgebra:-IdentityMatrix(m);
    while r1 = 0 do
        j0 := findcolumn(Rmod, i0, j0);
        while judgezeroele(Rmod, i0, j0) = 0 do
            LL, RR := interchangerow(LL, RR, Rmod, i0, j0, yvar);
            Rmod := modpx(RR, p, xvar);
            LL, RR := euclidcolumnreduce(LL, RR, Rmod, i0, j0, p, xvar, yvar);
            Rmod := modpx(RR, p, xvar)
        end do;
        i0 := i0 + 1;
        j0 := j0 + 1;
        r1 := checkzerorow(Rmod, i0)
    end do;
    D0 := LinearAlgebra:-RowOperation(D0, r1, p);
```

```
    RR := Typesetting:-delayDotProduct(1 / D0, RR);
    return simplify(Typesetting:-delayDotProduct(LL, D0)), simplify(RR)
end proc
```

```
> preliminaryfactor :=proc(M, xvar, yvar)

   # Compute the primitive decomposition of the matrix M

      local  m, n, L, R, gg, pl, lpl, ii, p;
      m := RowDimension(M);
      n := ColumnDimension(M);
      L := IdentityMatrix(m);
      R := M;
      gg := content(gcdvector(MaxMinors(M)), yvar);
      pl := factorirrfactors(gg);
      lpl := numelems(pl);
      for ii from 1 to lpl do
          p := pl[ii];
          L, R := factorsingleirrfactor(L, R, p, xvar, yvar);
      end do;
      return L, R;
  end proc;
```

```
preliminaryfactor := proc(M, xvar, yvar)                                   (1.13)
    local m, n, L, R, gg, pl, lpl, ii, p;
    m := LinearAlgebra:-RowDimension(M);
    n := LinearAlgebra:-ColumnDimension(M);
    L := LinearAlgebra:-IdentityMatrix(m);
    R := M;
    gg := content(gcdvector(MaxMinors(M)), yvar);
    pl := factorirrfactors(gg);
    lpl := numelems(pl);
    for ii to lpl do
        p := pl[ii]; L, R := factorsingleirrfactor(L, R, p, xvar, yvar)
    end do;
    return L, R
end proc
```

2.new algo

```
> newsyzalg :=proc(M, svar, tvar)
      # outputs the syzygy module generated by the column of the matrix M;
      # M a matrix with entries of two variables;
```

```
    # svar  tvar variable
    local V, Q, n, m, V2, L, R, W;
    V2 := (Matrix(SyzygyModule(M^%T, [tvar]))); V2 := V2^%T;
    m := RowDimension(M);
    n := ColumnDimension(M);
    W := simplify(V2 • diagdenomcol(V2));
    L, R := preliminaryfactor(W^%T, svar, tvar);
    return R^%T;
  end proc;
newsyzalg := proc(M, svar, tvar)                                    (2.1)
  local V, Q, n, m, V2, L, R, W;
  V2 := Matrix(SyzygyModule(M^`%T`, [tvar]));
  V2 := V2^`%T`;
  m := LinearAlgebra:-RowDimension(M);
  n := LinearAlgebra:-ColumnDimension(M);
  W := simplify(Typesetting:-delayDotProduct(V2, diagdenomcol(V2)));
  L, R := preliminaryfactor(W^`%T`, svar, tvar);
  return R^`%T`
end proc
```

3.previous algorithm by chen et al

```
> checknonzerominor :=proc(M)

  # Determine whether a matrix M has a non-zero minor
  # Requires matrix rows to be smaller than columns and full rank

    local m, l, r, cc, k, i;
    l := RowDimension(M);
    m := ColumnDimension(M);
    r := m − l;
    cc := choose(m, l);
    k := numelems(cc);
    for i from 1 to k do
        if simplify(Determinant(SubMatrix(M, [1..l], cc[i]))) ≠ 0 then
            return cc[i];
        end if;
    end do;
  end proc;
checknonzerominor := proc(M)                                        (3.1)
  local m, l, r, cc, k, i;
  l := LinearAlgebra:-RowDimension(M);
  m := LinearAlgebra:-ColumnDimension(M);
```

```
    r := m - l;
    cc := combinat:-choose(m, l);
    k := numelems(cc);
    for i to k do
        if simplify(LinearAlgebra:-Determinant(LinearAlgebra:-SubMatrix(M, [1
        .. l], cc[i]))) <> 0 then
            return cc[i]
        end if
    end do
end proc
```

```
> shiftmat :=proc(M, li)
       local m, l, coli, F1, F2;
       l := RowDimension(M);
       m := ColumnDimension(M);
       coli := [op({`$`(1 .. m)} \ {op(li)})];
       F1 := SubMatrix(M, [1..l], li);
       F2 := SubMatrix(M, [1..l], coli);
       return F1, F2;
  end proc;
shiftmat := proc(M, li)                                                  (3.2)
    local m, l, coli, F1, F2;
    l := LinearAlgebra:-RowDimension(M);
    m := LinearAlgebra:-ColumnDimension(M);
    coli := [op({`$`(1..m)}minus{op(li)})];
    F1 := LinearAlgebra:-SubMatrix(M, [1..l], li);
    F2 := LinearAlgebra:-SubMatrix(M, [1..l], coli);
    return F1, F2
end proc
```

```
> step1 :=proc(F1, F2)

   # step1 in previous algorithm by chen et al

       local   F22, A, B, i, r, l1, dd ;
       r := ColumnDimension(F2);
       F22 :=-1·F2;
       A := simplify(Adjoint(F1) . F22);
       l1 := [ ];
       dd := simplify(Determinant(F1));
       for i from 1 to r do
           l1 := [op(l1), dd];
       end do;
       B := DiagonalMatrix(l1);
       return Matrix([[A], [B]]);
```

```
    end proc;
step1 := proc(F1, F2)                                                  (3.3)
    local F22, A, B, i, r, l1, dd;
    r := LinearAlgebra:-ColumnDimension(F2);
    F22 := - F2;
    A := simplify(Typesetting:-delayDotProduct(LinearAlgebra:-Adjoint(F1),
    F22));
    l1 := [ ];
    dd := simplify(LinearAlgebra:-Determinant(F1));
    for i to r do l1 := [op(l1), dd] end do;
    B := LinearAlgebra:-DiagonalMatrix(l1);
    return Matrix([[A], [B]])
end proc
```

```
> oldsyzalg :=proc(M, svar, tvar)
      local l, m, r, li, Hx, HH, F1, F2, G1, H1, E, HH1, V, H2, L, R, H3, V1, Ht;
      l := RowDimension(M);
      m := ColumnDimension(M);
      r := m - l;
      li := checknonzerominor(M);
      F1, F2 := shiftmat(M, li);
      Hx := step1(F1, F2);
      HH := Hx^%T;
      G1, H1 := preliminaryfactor(HH, svar, tvar);
      HH1, E := HermiteForm_column(H1, tvar, output = ['H', 'U']);
      V := simplify(E . diagdenomcol(E));
      H2 := SubMatrix(simplify(H1 . V), [1..r], [1..r]);
      L, R := preliminaryfactor(H2^%T, svar, tvar);
      H3 := L^%T;
      V1 := simplify(SubMatrix(Adjoint(V), [1..r], [1..m]));
      Ht := simplify((1/Determinant(V)) . (H3 . V1));
      return Ht^%T;
  end proc;
oldsyzalg := proc(M, svar, tvar)                                       (3.4)
    local l, m, r, li, Hx, HH, F1, F2, G1, H1, E, HH1, V, H2, L, R, H3, V1, Ht;
    l := LinearAlgebra:-RowDimension(M);
    m := LinearAlgebra:-ColumnDimension(M);
    r := m - l;
    li := checknonzerominor(M);
    F1, F2 := shiftmat(M, li);
    Hx := step1(F1, F2);
    HH := Hx^`%T`;
```

```
    G1, H1 := preliminaryfactor(HH, svar, tvar);
    HH1, E := MatrixPolynomialAlgebra:-HermiteForm[column](H1, tvar, output = [
    'H', 'U']);
    V := simplify(Typesetting:-delayDotProduct(E, diagdenomcol(E)));
    H2 := LinearAlgebra:-SubMatrix(simplify(Typesetting:-delayDotProduct(H1,
    V)), [1..r], [1..r]);
    L, R := preliminaryfactor(H2^`%T`, svar, tvar);
    H3 := L^`%T`;
    V1 := simplify(LinearAlgebra:-SubMatrix(LinearAlgebra:-Adjoint(V), [1
    ..r], [1..m]));
    Ht := simplify(Typesetting:-delayDotProduct(H3, V1) / LinearAlgebra:-
    Determinant(V));
    return Ht^`%T`
end proc
```

4.example

```
test1 := Matrix([[2·s·t, 2·t, 2·s, s^2 + t^2 + 1]]);
F := newsyzalg(test1, s, t);
simplify(test1 • F);
gcdvector(MaxMinors(F));
```

$$test1 := \begin{bmatrix} 2\,s\,t & 2\,t & 2\,s & s^2+t^2+1 \end{bmatrix}$$

$$F := \begin{bmatrix} 1 & 0 & 0 \\ -s & s\,t & -t^2-1 \\ 0 & s^2+1 & -s\,t \\ 0 & -2\,s & 2\,t \end{bmatrix}$$

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

$$1 \qquad (4.1)$$

```
st := time[real]() :
F := newsyzalg(test1, s, t) :
print("Computing Syzygy module using new algo costs:", time[real]() - st, "s");

st := time[real]() :
F := oldsyzalg(test1, s, t) :
print("Computing Syzygy module using previous algo costs:", time[real]() - st, "s");
```

```
st := time[real]() :
F := BasisOfKernelModule(test1^%T, [s, t], true) :
print("Computing Syzygy module using QSAlgorithm algo costs:", time[real]() − st,
    "s");
```

"Computing Syzygy module using new algo costs:", 0.017, "s"

"Computing Syzygy module using previous algo costs:", 0.039, "s"

"Computing Syzygy module using QSAlgorithm algo costs:", 0.088, "s" **(4.2)**

```
test2 := Matrix([[t^2 + s·t + 2·s^2 − 2·s^2·t, t^2 + 2·s·t + s·t^2 + 2·s^2 − s^2·t + 2·s^2·t^2, -t^2
    + s·t + 2·s·t^2 + 2·s^2 − s^2·t − 2·s^2·t^2, 2·s·t − 2·s·t^2 − 2·s^2·t − s^2·t^2]]);
F := newsyzalg(test2, s, t);
simplify(test2 • F);
gcdvector(MaxMinors(F));
```

$$test2 := \left[-2\, s^2\, t + 2\, s^2 + s\, t + t^2,\ 2\, s^2\, t^2 - s^2\, t + t^2\, s + 2\, s^2 + 2\, s\, t + t^2,\right.$$
$$\left. -2\, s^2\, t^2 - s^2\, t + 2\, t^2\, s + 2\, s^2 + s\, t - t^2,\ -s^2\, t^2 - 2\, s^2\, t - 2\, t^2\, s + 2\, s\, t\right]$$

$$F := \left[\left[\frac{(4936\, s^2 - 5451\, s + 695)\,(2797\, s^2 + 388\, s + 796)}{209856},\ \frac{13805992}{138305}\, s^4\right.\right.$$
$$- \frac{12224839}{138305}\, s^3 + \frac{2236628}{138305}\, s^2 - \frac{3516116}{138305}\, s,\ \frac{10339020934466}{417411555249}\, s^4$$
$$- \frac{17294200417877}{128434324692}\, s^3 + \frac{78976}{3279}\, s^2\, t + \frac{182436801594859}{2226194961328}\, s^2 - \frac{29072}{1093}\, s\, t$$
$$\left. - \frac{15899956275703}{1669646220996}\, s + \frac{11120}{3279}\, t + \frac{1554400694933}{42811441564}\right],$$
$$\left[-\frac{8628745}{209856}\, s^4 + \frac{6822019}{209856}\, s^3 - \frac{6648983}{104928}\, s^2 + \frac{525117}{17488}\, s - \frac{138305}{52464},\ -\frac{1725749}{27661}\, s^4\right.$$
$$+ \frac{6130494}{138305}\, s^3 - \frac{12468136}{138305}\, s^2 + \frac{5194964}{138305}\, s,\ -\frac{25847552336165}{1669646220996}\, s^4$$
$$+ \frac{181322742315229}{2226194961328}\, s^3 - \frac{49360}{3279}\, s^2\, t - \frac{196186886482601}{3339292441992}\, s^2 + \frac{45872}{3279}\, s\, t$$
$$\left. + \frac{29232213501799}{278274370166}\, s - \frac{63584}{3279}\, t - \frac{1554400694933}{42811441564}\right],$$
$$\Bigg[$$
$$-\frac{1}{209856}\big((5177247\, s^3 - 6509260\, s^2 - 1678848\, s\, t - 9539983\, s - 3357696\, t$$
$$+ 2232068)\, s\big),$$
$$-\frac{1}{138305}\big(s\, (5177247\, s^3 - 6094345\, s^2 - 1678848\, s\, t - 10231508\, s$$

$$- 3357696\ t + 1678848)), \ -\frac{5169510467233\ s^4}{556548740332} + \frac{355330194783917\ s^3}{6678584883984}$$
$$+ \frac{(-40205054422272\ t - 154936631819375)\ s^2}{6678584883984}$$
$$+ \frac{(20323137211904\ t - 637973298940364)\ s}{6678584883984} - 16\ t\Big],$$
$$\Big[-\frac{1725749}{52464}\ s^4 - \frac{1517243}{69952}\ s^3 - 16\ s^2\ t - \frac{1340863}{209856}\ s^2 + 16\ s\ t - \frac{1298741}{104928}\ s - 8\ t$$
$$+ \frac{138305}{104928}, \ -\frac{6902996\ s^4}{138305} - \frac{5104949\ s^3}{138305} + \frac{(-3357696\ t - 1479168)\ s^2}{138305}$$
$$+ \frac{(3357696\ t - 2597482)\ s}{138305} - \frac{1678848\ t}{138305}, \ -\frac{5169510467233\ s^4}{417411555249}$$
$$+ \frac{3031172487661\ s^3}{642171662346} + \frac{(-120660308888576\ t + 333686920694963)\ s^2}{6678584883984}$$
$$+ \frac{(206590901700864\ t + 105414662675448)\ s}{6678584883984} - \frac{12801708880\ t}{381895293} + \frac{1554400694933}{85622883128}$$
$$\Big]\Big]$$

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

512 **(4.3)**

```
st := time[real]( ) :
F := newsyzalg(test2, s, t) :
print("Computing Syzygy module using new algo costs:", time[real]( ) − st, "s");

st := time[real]( ) :
F := oldsyzalg(test2, s, t) :
print("Computing Syzygy module using previous algo costs:", time[real]( ) − st, "s");


st := time[real]( ) :
F := BasisOfKernelModule(test2^%T, [s, t], true) :
print("Computing Syzygy module using QSAlgorithm algo costs:", time[real]( ) − st,
    "s");
```

"Computing Syzygy module using new algo costs:", 0.117, "s"

"Computing Syzygy module using previous algo costs:", 0.114, "s"

"Computing Syzygy module using QSAlgorithm algo costs:", 0.483, "s" **(4.4)**

```
test3 := Matrix([[-3·s^2·t^2 + 5·s^2·t − 5·t^2 − 4·s·t + 5, -3·s^2·t^2 + 3·s^2·t + s^2 + s·t^2 − s
    − 2·t^2 − 5·s·t + 1, -5·s^2·t^2 + 6·s^2·t + 2·s·t − t^2 − t − 5, -4·s^2·t^2 + 3·s^2·t − s·t + 6·t^2
    − t + 1]]);
F := newsyzalg(test3, s, t) :
simplify(test3 • F);
gcdvector(MaxMinors(F));
```

$$test3 := [-3\ s^2\ t^2 + 5\ s^2\ t - 4\ s\ t - 5\ t^2 + 5,\ -3\ s^2\ t^2 + 3\ s^2\ t + t^2\ s + s^2 - 5\ s\ t - 2\ t^2 - s + 1,\ -5\ s^2\ t^2 + 6\ s^2\ t + 2\ s\ t - t^2 - t - 5,\ -4\ s^2\ t^2 + 3\ s^2\ t - s\ t + 6\ t^2 - t + 1]$$

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

$$2916 \tag{4.5}$$

```
st := time[real]() :
F := newsyzalg(test3, s, t) :
print("Computing Syzygy module using new algo costs:", time[real]() − st, "s");

st := time[real]() :
F := oldsyzalg(test3, s, t) :
print("Computing Syzygy module using previous algo costs:", time[real]() − st, "s");
```

"Computing Syzygy module using new algo costs:", 0.089, "s"

"Computing Syzygy module using previous algo costs:", 0.105, "s" **(4.6)**

```
test4 := [[-s t − s + 2 t, 3 s^2 + 3 s t − 3 t, s^2 − 2 t − 3, -s^2 + 2 t^2, 2 + 3 s, 3 s t − t
    + 2, -3 s^2 − s + 1, -3 t^2 − 2 s − 3 t],
    [-3 t^2 + 3 s − 1, s^2 + s t − 3 t^2, 2 s t + t^2 + 2 s, -s^2 + 2 t^2 − 2, 2 + 2 s − t, s t
    + 2 t^2 + 3 t, -3 t^2 − 3 t, -2 s^2 − 2 s t − 3],
    [s^2 − 2 t^2 − t, 2 s^2 + 3 s t − 3 s, -3 t^2 − 2, 1 + 3 s + 3 t, -2 s t + 3 t^2 + t, -3 s t
    − 2 t + 3, -t^2 − 2 t, s t − t]];
F := newsyzalg(test4, s, t) :
simplify(test4 • F);
 #gcdvector(MaxMinors(F));
```

$$test4 := [[-s\ t - s + 2\ t,\ 3\ s^2 + 3\ s\ t - 3\ t,\ s^2 - 2\ t - 3,\ -s^2 + 2\ t^2,\ 2 + 3\ s,\ 3\ s\ t - t + 2,\ -3\ s^2 - s + 1,\ -3\ t^2 - 2\ s - 3\ t],$$
$$[-3\ t^2 + 3\ s - 1,\ s^2 + s\ t - 3\ t^2,\ 2\ s\ t + t^2 + 2\ s,\ -s^2 + 2\ t^2 - 2,\ 2 + 2\ s - t,\ s\ t + 2\ t^2 + 3\ t,\ -3\ t^2 - 3\ t,\ -2\ s^2 - 2\ s\ t - 3],$$
$$[s^2 - 2\ t^2 - t,\ 2\ s^2 + 3\ s\ t - 3\ s,\ -3\ t^2 - 2,\ 1 + 3\ s + 3\ t,\ -2\ s\ t + 3\ t^2 + t,\ -3\ s\ t - 2\ t + 3,\ -t^2 - 2\ t,\ s\ t - t]]$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.7)$$

```
st := time[real]() :
F := newsyzalg(test4, s, t) :
print("Computing Syzygy module using new algo costs:", time[real]() - st, "s");

st := time[real]() :
F := oldsyzalg(test4, s, t) :
print("Computing Syzygy module using previous algo costs:", time[real]() - st, "s");
```

"Computing Syzygy module using new algo costs:", 3.059, "s"
"Computing Syzygy module using previous algo costs:", 34.374, "s" **(4.8)**

```
test5 := [[3 s + t + 2, 2 t + 1, 2 s - 3, 3 s + t + 3, 3 s - t + 1, -3 s - 1, -s - 1, -s
    + t - 2, -s - 2 t - 1, -2 t + 1],
   [-2 s - 2 t - 3, 2 s + 2 t - 3, 2 s + 1, -s + t + 2, -3 s - 2, s - 2 t - 3, 3 t - 3,
   -2 s + 3 t + 3, s + 3 t - 3, -2 s],
   [-s + 3 t + 3, -s - 3, -s + 3 t - 1, -3 s - t + 2, 3 t + 1, 2 s - t + 1, 3 s + 3 t
   - 2, -2 s + 1, 2 s + 3 t - 1, t - 3],
   [t + 1, s - 2 t, -2 s + 2 t + 1, 3 s - 3 t, s + 2 t + 2, -s + t + 1, 3 s - 2 t + 2,
   -3 s - t + 1, s + 3 t - 2, 2 s - 2],
   [s + t - 3, 2 s, 3 s + t + 2, -2 s - t - 1, -s - 2, -s - t - 1, 2 s - t + 1, 2 s - t
   - 3, s + 2, 2 s + 3],
   [-s - 1, -s + 3 t - 2, 3 s + 3 t + 1, -2 s + 2 t, -3 s + 3 t + 3, 2 s - 2 t - 2, -3 s
   + 2, -3 s + 3 t - 2, 2 s - t + 2, 3 s - 2 t]];
F := newsyzalg(test5, s, t) :
simplify(test5 · F);
 #gcdvector(MaxMinors(F));
```

test5 := [[3 s + t + 2, 2 t + 1, 2 s − 3, 3 s + t + 3, 3 s − t + 1, -3 s − 1,
   -s − 1, -s + t − 2, -s − 2 t − 1, -2 t + 1],
   [-2 s − 2 t − 3, 2 s + 2 t − 3, 2 s + 1, -s + t + 2, -3 s − 2, s − 2 t
   − 3, 3 t − 3, -2 s + 3 t + 3, s + 3 t − 3, -2 s],
   [-s + 3 t + 3, -s − 3, -s + 3 t − 1, -3 s − t + 2, 3 t + 1, 2 s − t + 1,
   3 s + 3 t − 2, -2 s + 1, 2 s + 3 t − 1, t − 3],
   [t + 1, s − 2 t, -2 s + 2 t + 1, 3 s − 3 t, s + 2 t + 2, -s + t + 1, 3 s
   − 2 t + 2, -3 s − t + 1, s + 3 t − 2, 2 s − 2],
   [s + t − 3, 2 s, 3 s + t + 2, -2 s − t − 1, -s − 2, -s − t − 1, 2 s − t
   + 1, 2 s − t − 3, s + 2, 2 s + 3],
   [-s − 1, -s + 3 t − 2, 1 + 3 s + 3 t, -2 s + 2 t, -3 s + 3 t + 3, 2 s − 2 t
   − 2, -3 s + 2, -3 s + 3 t − 2, 2 + 2 s − t, 3 s − 2 t]]

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.9}$$

```
st := time[real]() :
F := newsyzalg(test5, s, t) :
print("Computing Syzygy module using new algo costs:", time[real]() - st, "s");

st := time[real]() :
F := oldsyzalg(test5, s, t) :
print("Computing Syzygy module using previous algo costs:", time[real]() - st, "s");
```

"Computing Syzygy module using new algo costs:", 2.079, "s"

"Computing Syzygy module using previous algo costs:", 56.329, "s" **(4.10)**