```
> restart;with(LinearAlgebra):
>
```

Procédure dans le cas où il y a un carré

```
> sicarre:=proc(indice::integer,T0::Matrix,p::integer)::list(Matri
  x);
  local T::Matrix,P::Matrix,i::integer,j::integer,n::integer;
  T:=T0;
  n:=RowDimension(T);
  P:=Matrix(n,n,0);
  for i from 1 to n do
    P[indice,i]:=T[indice,i]/T[indice,indice];
  od;

  T[[1..(indice-1),(indice+1)..n],[1..(indice-1),(indice+1)..n]]

  :=T[[1..(indice-1),(indice+1)..n],[1..(indice-1),(indice+1)..n]]

  -((1)/T[indice,indice])*T[[1..indice-1,indice+1..n],indice].Tran
  spose(T[[1..indice-1,indice+1..n],indice]);

  for i from 1 to n do
    if (i<>indice) then
      T[indice,i]:=0;
      T[i,indice]:=0;
    fi;
  od;
  RETURN([T,P]);
  end proc:
```

Procédure dans le cas où il n'y a pas de facteur carré

```
> sipascarre:=proc(indice::list(integer),T0::Matrix,p::integer)::l
  ist(Matrix);
  local
  T::Matrix,P::Matrix,B1::Vector,B2::Vector,B3::Matrix,i::integer,
  j::integer,n::integer,a;
  T:=T0;
  n:=RowDimension(T);
  P:=Matrix(n,n,0);
  B1:=2*T[[1..indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1..n]
  ,indice[1]];
  B2:=2*T[[1..indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1..n]
  ,indice[2]];
  #f_1 = t[1,2] en [1] + b2
  #f_2 = t[1,2]/t[1,2] en [2] + b1/t[1,2]
  B3:=B1.Transpose(B2);
  a:=2*T[indice[1],indice[2]];

  # matrice de changement de base:
  P[indice[1],[1..indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1
  ..n]]:=(B2+((1)/a)*B1);
  P[indice[2],[1..indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1
  ..n]]:=(B2-((1)/a)*B1);
  P[indice[1],[indice[1],indice[2]]]:=< a | 1 >;
  P[indice[2],[indice[1],indice[2]]]:=< a | -1 >;
```

```
# nouvelle forme quadratique:

T[[1..indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1..n],[1..i
ndice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1..n]]

:=T[[1..indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1..n],[1.
.indice[1]-1,indice[1]+1..indice[2]-1,indice[2]+1..n]]
      -((1)/a)*(1/2)*(Transpose(B3)+B3);
T[[1..indice[1]-1,indice[1]+1..n],indice[1]]:=Vector(n-1);
T[indice[1],[1..indice[1]-1,indice[1]+1..n]]:=Transpose(Vector(n
-1));
T[indice[1],indice[1]]:=(1)/4;
T[[1..indice[2]-1,indice[2]+1..n],indice[2]]:=Vector(n-1);
T[indice[2],[1..indice[2]-1,indice[2]+1..n]]:=Transpose(Vector(n
-1));
T[indice[2],indice[2]]:=-(1)/4;

RETURN([T,P]);
end proc:
```

décomposition de gauss

```
> GaussDec:=proc(T0::Matrix,p::integer)::list(Matrix);
  local
  T::Matrix,P::Matrix,ListeResult1::list(Matrix),ListeResult2::lis
  t(Matrix),i::integer,j::integer,n::integer;
  T:=T0;
  n:=RowDimension(T);
  i:=1;
  while (i <= n and T[i,i] = 0) do i:=i+1; od;
  if (i <= n)
    then
      ListeResult1:=sicarre(i,T,p);
      T:=ListeResult1[1];
      P:=ListeResult1[2];

  ListeResult2:=GaussDec(T[[1..(i-1),(i+1)..n],[1..(i-1),(i+1)..n]
  ],p);
      T[[1..(i-1),(i+1)..n],[1..(i-1),(i+1)..n]]:=ListeResult2[1];
      P[[1..(i-1),(i+1)..n],[1..(i-1),(i+1)..n]]:=ListeResult2[2];
    else
      i:=1;
      j:=1;
      while (j <= n and T[i,j]=0) do
        j:=j+1;
        i:=1;
        while (i < j-1 and j <= n and T[i,j]=0) do i:=i+1; od;
      od;
      if (j <= n)
        then
          ListeResult1:=sipascarre([i,j],T,p);
          T:=ListeResult1[1];
          P:=ListeResult1[2];
```

```
ListeResult2:=GaussDec(T[[1..(i-1),(i+1)..(j-1),(j+1)..n],[1..(i
-1),(i+1)..(j-1),(j+1)..n]],p);

T[[1..(i-1),(i+1)..(j-1),(j+1)..n],[1..(i-1),(i+1)..(j-1),(j+1).
.n]]:=ListeResult2[1];

P[[1..(i-1),(i+1)..(j-1),(j+1)..n],[1..(i-1),(i+1)..(j-1),(j+1).
.n]]:=ListeResult2[2];
      else
        # fournir le rangN:=n;
    fi;

fi;
RETURN([T,P]);
end proc:
```

teste si les matrices de la décomposition de gauss sont bien celles attendues.

```
> Teste0:=proc(LR::list(Matrix),p)::Matrix;
  Transpose(LR[2]).LR[1].LR[2];
  RETURN(%);
  end proc:
```

Même chose modulo p

```
> Teste:=proc(LR::list(Matrix),p)::Matrix;
  Transpose(LR[2]).LR[1].LR[2] mod p;
  RETURN(%);
  end proc:
```

Forme quadratique associée à une matrice

```
> Quad:=proc(Mat::Matrix);
  expand(Transpose(Vector(RowDimension(Mat),symbol=x)).Mat.Vector(
  RowDimension(Mat),symbol=x));
  RETURN(%);
  end proc:
```

produit la matrice GG

```
> GenerateGG:=proc(S::Matrix,q::integer,e::integer,N::integer,s::i
  nteger)::Matrix;
  local GG::Matrix,i::integer,j::integer,k::integer;
  GG:=Matrix(e+floor((e-s)/q),(N+1)*floor((e-s)/q),0):
  for i from 1 to e+1 do
     for j from 1 to N+1 do
        for k from 1 to floor((e-s)/q) do
          GG[i+k-1,(k-1)*(N+1)+j]:=S[j,i];
        od;
     od;
  od;
  RETURN(GG);
  end proc:
```

liste des déterminents des mineurs maximaux de la matrice M, en caractéristique p
(lignes < col)
(colenlevee:=array(1..(col-lignes)))

```
> suitedetmineurs:=proc(p::integer,M::Matrix,lignes::integer,col::
  integer,colenlevee::array(integer),edt::list)::array;
  local edt1::list,colenlevee1::array(integer),l::integer,zorglub;
  edt1:=edt;
  colenlevee1:=copy(colenlevee);
```

```
  if
    lignes=col
  then
    zorglub:=evala(Determinant(M)) mod p;
    if zorglub <> 0 then
  edt1:=[op(edt1),[copy(colenlevee1),zorglub]]; fi;
  else
    if
      op(op(2,eval(colenlevee1))[1])[2]=col-lignes
    then
      for l from 1 to col do
        colenlevee1[col-lignes]:=l;

  edt1:=suitedetmineurs(p,M[1..lignes,[1..(l-1),(l+1)..col]],ligne
  s,col-1,colenlevee1,edt1);
      end do:
    else
      for l from colenlevee1[col-lignes+1] to col do

  colenlevee1[col-lignes]:=l+op(op(2,eval(colenlevee1))[1])[2]-(co
  l-lignes);

  edt1:=suitedetmineurs(p,M[1..lignes,[1..(l-1),(l+1)..col]],ligne
  s,col-1,colenlevee1,edt1);
      end do:
    fi;
  fi;
  RETURN(edt1);
  end proc:
```

engendre T a partir d'un bloc A **symetrique**, pour e=2q seulement (un chevauchement)
(degré minimal pour avoir une courbe libre (s=0))

```
> GenerateT:=proc(A::Matrix)::Matrix;
  local T::Matrix,n::integer;
  n:=RowDimension(A);
  T:=Matrix(2*n+1);
  T[[2..n+1],[2..n+1]]:=A;
  T[[1..n],[n+2..2*n+1]]:=-A;
  T[[n+2..2*n+1],[1..n]]:=-A;
  T[[n+1..2*n],[n+1..2*n]]:= T[[n+1..2*n],[n+1..2*n]] + A;
  RETURN(T);
  end proc:
```

engendre T a partir d'un bloc A **symetrique**, pour e=2q+1
(degré minimal pour avoir une courbe très libre (s=1))

```
>
```

pivot de gauss, donne le rang dans F_p PROBLEMES

```
> Rankp:=proc(p::integer,M0::Matrix)::integer;
  local
  M::Matrix,n::integer,m::integer,i0::integer,i::integer,Ligne::Ve
  ctor;
  m:=RowDimension(M0);
  n:=ColumnDimension(M0);
  if n=0 then RETURN(0);
    else
```

```
    i0:=1;
    while (i0 <= m and M0[i0,1] = 0) do i0:=i0+1; od;
    if i0=m+1 then RETURN(Rankp(p,M0[[1..m],[2..n]]));
      else
      M:=M0;
      if (i0 > 1) then
        Ligne:=M[i0,1..n];
        M[i0,1..n]:=M[1,1..n];
        M[1,1..n]:=Ligne;
        fi;
      for i from i0+1 to m do
        if (M[i,1] <> 0) then
          M[i,1..n]:=Ligne[1].M[i,1..n]-M[i,1].Ligne;
          fi;
        od;
      RETURN(1+Rankp(p,M[[2..m],[2..n]] mod p));
      fi;
    fi;
  end proc:
```

Teste dans le cas p=5

```
>
```

Valeurs des constantes

```
> p:=5;
  q:=p;
  e:=2*q;
  N:=q+1;
  s:=0;
```

$$p := 5$$

$$q := 5$$

$$e := 10$$

$$N := 6$$

$$s := 0$$

```
> #A:=RandomMatrix(q,q,generator=-(p-1)/2..(p-1)/2,outputoptions=[
  shape=symmetric]);Rank(A);
  A:=Matrix(q,q,0,shape=symmetric):
  for i from 1 to (q-1)/2-1 do
    A[i,(q-1)/2-i+1]:=-1;
  od:
  for i from 1 to (q-1)/2-1 do
    A[q+1-i,(q-1)/2+i+1]:=-1;
  od:
  #A[1,(q+1)/2]:=0:
  #A[4,4]:=0:
  for i from 1 to q-1 do
    A[i+1,q+1-i]:=1;
    A[i,q-i]:=1;
  od:
  ##### AUTRE #####
  A:=Matrix(q,q,0,shape=symmetric):
  for i from 1 to q do
    A[i,i]:=1;
    A[q+1-i,i]:=1;
```

```
  od:
  for i from 1 to (q-1)/2 do
    A[i,(q+1)/2-i+1]:=1;
    A[(q+1)-i-1,(q+1)/2+i]:=1;
  od:

  A;
  Rank(A);
  #latex(A);
```

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

$$2$$

```
> T:=GenerateT(A);
  Rank(T)-q-2;
  #evalm(T);
```

$$T := \begin{bmatrix} \text{11 x 11 Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

$$0$$

```
> LR:=GaussDec(copy(T),p):T:
> if Rank(T-Teste0(LR))=0 then true else T-Teste0(LR) ;fi;
```

$$true$$

```
> evalm(LR[1]);evalm(LR[2]);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & P & P & 0 & 0 & 0 & P & P & 0 \\ 0 & 0 & 0 & P & P & 0 & 0 & 0 & P & P & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & P & P & 0 & 0 & 0 & P & P & 0 \\ 0 & 0 & 0 & P & P & 0 & 0 & 0 & P & P & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

```
> Pzero:=LR[2][[1..3,6..8,11],[1..2*q+1]];
  Dzero:=LR[1][[1..3,6..8,11],[1..3,6..8,11]];
```

$$Pzero := \begin{bmatrix} \text{7 x 11 Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

$$Dzero := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

```
> if Rank(T-Teste0([Dzero,Pzero]))=0 then true else
  T-Teste0([Dzero,Pzero]) ;fi;
```

$$true$$

```
> Dun:=DiagonalMatrix([1,1,1,1,1,1,2]);if Rank(Dun.Dun-Dzero mod
  p)=0 then true else Dun.Dun-Dzero mod p; fi;
```

$$Dun := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

true

```
> S:=Dun.Pzero;
  if Rank(T-Teste0([IdentityMatrix(q+2),S]) mod 5)=0 then true
  else evalm(T-Teste0([IdentityMatrix(q+2),S]) mod 5) ;fi;
  #latex(S);
```

$$S := \begin{bmatrix} \text{7 x 11 Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

true

```
> G:=GenerateGG(S,q,e,N,s);
  #latex(G);
```

$$G := \begin{bmatrix} \text{12 x 14 Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

```
> Rank(G);
```

$$10$$

```
> evalm(G mod 5);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 1 & 0 & 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 4 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 4 & 0 & 1 & 0 & 2 \\ 0 & 0 & 4 & 0 & 1 & 0 & 2 & 0 & 4 & 0 & 2 & 0 & 1 & 0 \\ 0 & 4 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 4 & 0 & 1 & 0 & 2 \\ 0 & 0 & 4 & 0 & 0 & 0 & 2 & 0 & 4 & 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 2 \end{bmatrix}$$

```
>
>
> ?Identity
>
Error, reserved word `end` unexpected
```