```
> restart:
> (* PACKAGES: *)

> with(LinearAlgebra):
  with(ArrayTools):
  with(Groebner):

> (* HELPER FUNCTIONS: *)

> DIVALGO := proc(f::polynom,F::list(polynom),X::list(symbol),
  ord::symbol)
  local Q,R,P,L,i,j,LTP,sizeF,flag:

  sizeF := nops(F):
  Q := Array(1..sizeF):
  R := 0: (* Remainder Column. *)
  P := f: (* Intermediate Dividend. *)
  L := [seq(LT(F[j],X,ord),j=1..sizeF)]:

  while P<>0 do:
     flag := false:
     LTP := LT(P,X,ord):
     for i from 1 to sizeF while not flag do
         if L[i]<>0 then:
            if divide(LTP,L[i],'q') then
                Q[i] := Q[i]+q:
               P := P-(expand(q*F[i])):
                flag := true:
              fi:
          fi:
     od:
     if flag=false then:
        R := R+LTP:
        P := P-LTP:
      fi:
  od:

  return R:
  end proc:

> LT := proc(f::polynom,X::list(symbol),ord::symbol)
  local lm,lc:
```

```
  lm,lc := LeadingTerm(f,ord(op(X))):
  return (lm*lc):
  end proc:

> BBALGO := proc(F::list(polynom),X::list(symbol),ord::symbol)
  ::list(polynom):
  local G,GPrev,LMF,LMG,hash,size,i,j,sPol,sPolRem,reduceG:

  G := F:
  hash := table():
  do
     GPrev := G:
     size := nops(G):
     for i from 1 to size do
        for j from (i+1) to size do
          LMF := LeadingMonomial(GPrev[i],ord(op(X))):
          LMG := LeadingMonomial(GPrev[j],ord(op(X))):
           if gcd(LMF,LMG)=1 then
                next:
              fi:
            if assigned(hash[i,j]) then
                next:
              fi:
             hash[i,j] := true:
           sPol := SPolynomial(GPrev[i],GPrev[j],ord(op(X))):
          sPolRem := DIVALGO(sPol,G,X,ord):
           if sPolRem<>0 then
             (* sPolRem := expand(sPolRem/lcoeff(sPolRem,
  order=ord(op(X)))): *)
               G := [op(G),sPolRem]:
              fi:
         od:
      od:
  until G=GPrev:
  reduceG := REDUCEGB(GPrev,X,ord):
  return reduceG:
  end proc:

> REDUCEGB := proc(G::list(polynom),X::list(symbol),ord::symbol)
  ::list(polynom):
  local temp,i,j,LTF,LTFtemp,LTGtemp,LTG,k,newGB:
```

```
  temp := G:

  i := 1:
  while i<= nops(temp) do:
     LTF := LT(temp[i],X,ord):
      j := i+1:
     while j<= nops(temp) do:
        LTG := LT(temp[j],X,ord):
        if divide(LTG,LTF) then:
           temp := subsop(j=NULL,temp):
         else:
             j++:
          fi:
      od:
      i++:
  od:

  for i to nops(temp) do:
     temp[i] := DIVALGO(temp[i],subsop(i=NULL,temp),X,ord):
     if temp[i]<>0 then:
        newGB[i] := primpart(expand(temp[i]/lcoeff(temp[i]))):
      fi:
  od:

  newGB := convert(newGB,list):

  return newGB:
  end proc:

> REDUCEGBrev := proc(G::list(polynom),X::list(symbol),ord::symbol)
  ::list(polynom):
  local temp,i,j,LTF,LTFtemp,LTGtemp,LTG,k,newGB:

  temp := G:

  i := 1:
  while i<= nops(temp) do:
     LTF := LT(temp[i],X,ord):
      j := i+1:
     while j<= nops(temp) do:
        LTG := LT(temp[j],X,ord):
        if divide(LTG,LTF) then:
```

```
          temp := subsop(j=NULL,temp):
        else:
            j++:
         fi:
    od:
    i++:
od:

for i to nops(temp) do:
   temp[i] := DIVALGO(temp[i],subsop(i=NULL,temp),X,ord):
    if temp[i]<>0 then:
       newGB[i] := expand(temp[i]/lcoeff(temp[i])):
     fi:
od:

newGB := convert(newGB,list):

return newGB:
end proc:
> GETCOEFF := proc(p,m,vars::list(name))
local q, v, e;

q := p;
for v in vars do:
   e := degree(m,v):
   if e=-infinity then
       e := 0:
    fi:
   q := coeff(q,v,e);
   if q=0 then
   return 0:
    fi:
od:
return q:
end proc:

> MAKEARR := proc(p::polynom,M,vars::list(name))
local m,res:

res := Array(1..nops(M),[seq(GETCOEFF(p,m,vars),m in M)]):
return res:
end proc:
```

```
> NextMon := module()
  option package:
  export Init,Next,Reset:

  local X,n,D,L,p;
  local MonFromExp := proc(e)
  local i:
  mul(X[i]^e[i],i=1..n):

  end proc:

  local CompsRevLexLast := proc(D::nonnegint,m::posint)
  local res,k,tails,t:

     if m=1 then:
        return [[D]];
     else
        res := [];
       for k from D to 0 by -1 do:
         tails := CompsRevLexLast(D-k,m-1):
           for t in tails do:
              res := [op(res),[op(t),k]]:
            od:
        od:
       return res:
     fi:

  end proc:

  Init := proc(vars::{list,set}(name))

  X := [op(vars)]:
  n := nops(X):
  D := 1:
  L := CompsRevLexLast(D,n):
  p := 1:

  end proc:

  Next := proc()
  local e;
```

```
  if p>nops(L) then:
      D := D+1:
     L := CompsRevLexLast(D,n):
      p := 1:
  f i :
  e := L[p]:
  p++:

  return MonFromExp(e):
  end proc;

  Reset := proc()  D := 1;  L := CompsRevLexLast(D, n);  p := 1;
  end proc;

  end module:
> (* INPUTS *):

> IDE := [x^2+(2*y^2)-y-(2*z),x^2-(8*y^2)+(10*z)-1,x^2-(7*y*z)]:
  IDE:
> IDE2 := [(x*y)+z-(x*z),x^2-z,(2*x^3)-(x^2*y*z)-1]:
  IDE2:
> IDE3 := [(x*y)+y^2-3,x^2+y^2-4,(2*y^3)+(3*x)-(7*y)]:
  IDE3:
> IDE4 := [(x*y)+z-(x*z),x^2-z,(2*x^3)-(x^2*y*z)-1]:
  IDE4:
> TRINKS := [45*p+35*s-165*b-36,35*p+40*z+25*t-27*s,15*w+25*p*s+30*
  z-18*t-165*b^2,-9*w+15*p*t+20*z*s,w*p+2*z*t-11*b^3,99*w-11*s*b+3*
  b^2]:
  TRINKS:
> oldGB := Groebner[Basis](TRINKS,tdeg(t,z,s,b,p,w)):
  oldGB:
> LTI := LeadingMonomial(oldGB,tdeg(t,z,s,b,p,w) ) :
  LTI:
> with(PolynomialIdeals):
> BasisElement,B2 := NormalSet(oldGB,tdeg(t,z,s,b,p,w) ) :
  BasisElement:

> (* FGLM ALGORITHM IMPLEMENTATION: *)

> repFGLM := proc(oldGB::list(polynom),Xold::list(symbol),
  Xnew::list(symbol),ord::symbol,newOrd::symbol)
```

```
local GLEX,BLEX,gCount,bCount,monIP,k,rowCount,mon2Pow,divG,
tempMat,bVec,res,addG,i,iFlag,LTGi,counter,temp1,temp2,dim,
monList:
global constMat:

GLEX := table():
BLEX := table():
gCount := 1:
bCount := 1:
monIP := 1:
rowCount := 1:
counter := 1:
NextMon:-Init(Xnew):

while true do:
   if counter=1 then:
       k := 0:
      counter++:
    else:
       k := 1:
    fi:
   monIP := NextMon:-Next():
   iFlag := false:
   while iFlag<>true do:
     mon2Pow := monIP^k:
       k++:
     divG := DIVALGO(mon2Pow,oldGB,Xold,ord):
      if rowCount>1 then:
        constMat := Concatenate(1,constMat,MAKEARR(divG,
BasisElement,Xold)):
       else:
          constMat := MAKEARR(divG,BasisElement,Xold):
         rowCount++:
        fi:
     if Rank(constMat)<>RowDimension(constMat) then:
        dim := RowDimension(constMat):
        tempMat := Transpose(constMat[1..dim-1]):
        bVec := Transpose(constMat[dim]):
        res := LinearSolve(tempMat,bVec):
        addG := mon2Pow-add(res[i]*op(Xnew[-1])^(i-1),i=1..
RowDimension(res)):
        GLEX[gCount] := addG:
```

```
            gCount++:
          LTGi := LT(addG,Xnew,newOrd):
          constMat := Transpose(tempMat):
           if divide(LTGi,op(Xnew[1])) then:
              GLEX := convert(GLEX,list):
              BLEX := convert(BLEX,list):
            GLEX := REDUCEGB(GLEX,Xnew,newOrd):
              return (GLEX,BLEX):
             else:
                iFlag := true:
                 break:
              fi:
          fi:
       BLEX[bCount] := mon2Pow:
        bCount++:
    od:
  od:

  end proc:
> (* EXAMPLE 1: *)

> oldGB := Groebner[Basis](IDE3,tdeg(x,y)):
  oldGB;
```

$$[xy + y^2 - 3,\ x^2 + y^2 - 4,\ 2y^3 + 3x - 7y]$$

```
> LTI := LeadingMonomial(oldGB,tdeg(x,y)):
  LTI;
```

$$[xy,\ x^2,\ y^3]$$

```
> BasisElement,B2 := NormalSet(oldGB,tdeg(x,y)):
  BasisElement;
```

$$[1,\ y,\ x,\ y^2]$$

```
> res1,res2 := repFGLM(oldGB,[x,y],[x,y],tdeg,plex):
  res1;
  res2;
```

$$[2y^4 - 10y^2 + 9,\ 2y^3 + 3x - 7y]$$

$$[1,\ y,\ y^2,\ y^3]$$

```
> mapGB := Groebner[Basis](IDE3,plex(x,y)):
  mapGB;
```

$$[2y^4 - 10y^2 + 9,\ 2y^3 + 3x - 7y]$$

```
> (* EXAMPLE 2: *)
```

```
> oldGB := Groebner[Basis](IDE4,tdeg(x,y,z)):
  oldGB;
```

$$[x+y-z,\ y^2-2yz+z^2-z,\ yz^2+2yz-2z^2+1,\ z^4-3z^3-4yz+2z^2-y+2z-2]$$

```
> LTI := LeadingMonomial(oldGB,tdeg(x,y,z)):
  LTI;
```

$$[x,\ y^2,\ yz^2,\ z^4]$$

```
> BasisElement,B2 := NormalSet(oldGB,tdeg(x,y,z)):
  BasisElement;
```

$$[1,\ z,\ y,\ z^2,\ yz,\ z^3]$$

```
> res1,res2 := repFGLM(oldGB,[x,y,z],[x,y,z],tdeg,plex):
  res1;
  res2;
```

$$[z^6-z^5-4z^4-2z^3+1,\ -4z^5+5z^4+13z^3+10z^2+7y-6z-2,\ 4z^5-5z^4-13z^3 - 10z^2+7x-z+2]$$

$$[1,\ z,\ z^2,\ z^3,\ z^4,\ z^5]$$

```
> mapGB := Groebner[Basis](IDE4,plex(x,y,z)):
  mapGB;
```

$$[z^6-z^5-4z^4-2z^3+1,\ -4z^5+5z^4+13z^3+10z^2+7y-6z-2,\ 4z^5-5z^4-13z^3 - 10z^2+7x-z+2]$$

```
> (* EXAMPLE 3: TRINKS SYSTEM *)

> oldGB := Groebner[Basis](TRINKS,tdeg(t,z,s,b,p,w)):
  oldGB:

  (* OUTPUT TOO BIG. *)

> LTI := LeadingMonomial(oldGB,tdeg(t,z,s,b,p,w) ) :
  LTI;
```

$$[s,\ t,\ pb,\ pz,\ b^2,\ bz,\ z^2,\ w^3,\ w^2p,\ w^2b,\ w^2z,\ wp^2,\ p^3]$$

```
> (* Bound on |V|. *)

  BasisElement,B2 := NormalSet(oldGB,tdeg(t,z,s,b,p,w) ) :
  BasisElement;
```

$$[1,\ w,\ p,\ b,\ z,\ w^2,\ pw,\ bw,\ wz,\ p^2]$$

```
> res1,res2 := repFGLM(oldGB,[t,z,s,b,p,w] , [t,z,s,b,p,w],tdeg,plex)
  :
```

```
(* OUTPUT IS TOO BIG. UNCOMPRESS TO CHECK. *)
res1:
nops(res1);
res2;
```

$$6$$

$$\left[1, w, w^2, w^3, w^4, w^5, w^6, w^7, w^8, w^9\right]$$

```
> mapGB := Groebner[Basis](TRINKS,plex(t,z,s,b,p,w)):
mapGB:
nops(mapGB);
```

$$6$$

```
> if mapGB=res1 then:
    print("TRUE"):
fi:
```

$$\text{"TRUE"}$$

```
> (* Bound on |V|. *)

BasisElement2,B2 := NormalSet(mapGB,plex(t,z,s,b,p,w) ) :
BasisElement;
```

$$\left[1, w, p, b, z, w^2, p\,w, b\,w, w\,z, p^2\right]$$