Algebraic multivariate GCD, Lucas Hu, July 2018.

```
> restart;

> currentdir();
                "/Users/lucas/Desktop/number_field_online"                    (1)
> read("recden");
Warning, `C` is implicitly declared local to procedure 
`interprpoly`

> evaltotaltime:=0:


> # Kronecker substitution.
> Kron:=proc(poly,L::list,V::list)
>     local result:
>    result:=subs([seq(V[k]=V[1]^mul(L[i],i=1..k-1),k=1..nops(V))
  ],poly):
>    return result:
> end:

> # Inverse Kronecker substitution.
> Kroninv:=proc(poly,L::list,V::list,main)  # can use algsubs
  because e1+e2*r1+e3*r1*r2 < r1*r2*r3.
>    local result, d, dm, i, j, r, C, M, temp:    # e1+e2(e1+1)+e3
  (e1+1)(e2+1) < (e1+1)(e2+1)(e3+1)= (e1+1)(e2+1)e3+(e1+1)e2+e1+1
>    result:=1:
>    if whattype(poly) = `*` then
>        d:=degree(poly,V[1]):
>       dm:=degree(poly,main):
>        for i to nops(V) do
>            r:=d mod L[i]:
>            result:=result*V[i]^r:
>             d:=(d-r)/L[i]:
>          od:
>       return eval(poly,[main=1,V[1]=1])*main^(dm)*result:
>     fi:
>    if whattype(poly) = `+` then
>       C:=coeffs(poly,[main,V[1]],`M`):
>        M:=[M]:
>        result:=0:
>       for i to nops(M) do
>           temp:=1:
>           d:=degree(M[i],V[1]):
>           for j to nops(V) do
>               r:=d mod L[j]:
>              temp:=temp*V[j]^r:
```

```
>                 d:=(d-r)/L[j]:
>              od:
>           result:=result+C[i]*x^(degree(M[i],x))*temp:
>         od:
>      fi:
>    return result:
> end:
```

```
> # monic subresultant GCD algorithm
> SRGCD := proc(A,B,main,av,m,p)   # A and B are univariate
  polynomial with coefficients in Z_p[z]/<m,p> in recden stucture.


>    # main is the main variable, X[1].  av is the algebaric
  number X[-1].


>    local r,i,d,c,b,h,ZERO,lcr,lcri;
>    if degree(A,main) >= degree(B,main) then r[1], r[2]:= A, B:
>     else r[1], r[2] :=  B, A: fi:
>
>    r[1]:=A; r[2]:=B;
>     i:=3:
>    d[2]:=degrpoly(r[1]) - degrpoly(r[2]):
>    c[2]:=lcrpoly(r[2]):
>    b[2]:=rpoly((-1)^(d[2]+1),av,m,p):
>    h[2]:=rpoly(-1,av,m,p):
>    divrpoly(premrpoly(r[1],r[2]),b[2],'q'):    # b[2] is an
  integer, no try.
>     r[3]:=q;
>    ZERO:=rpoly(0,[main,av],m,p):
>
>     try

>    while r[i] <> ZERO  do
>
>        d[i]:=degrpoly(r[i-1])-degrpoly(r[i]):
>         c[i]:=lcrpoly(r[i]):
>       divrpoly( powrpoly(scarpoly(-1,c[i-1]),d[i-1]), powrpoly
  (h[i-1],d[i-1]-1), 'Q'):
>         h[i]:=Q:
>        b[i]:=mulrpoly(scarpoly(-1,c[i-1]),powrpoly(h[i],d[i])):
>
>        divrpoly(premrpoly(r[i-1],r[i]),b[i],'q'):
>         r[i+1]:=q:
>          i:=i+1:
```

```
>     od:
>
>     catch:
>         return FAIL:
>     end try:

>     lcr:=lcrpoly(r[i-1]):
>
>      try
>         lcri:=invrpoly(lcr):
>         return mulrpoly(lcri,r[i-1]):
>     catch:
>         return FAIL:
>     end try:

> end:
```

```
> ugcd:=proc(K1,K2,LC,X,m,p)  # Compute the univariate GCD over
  number field. X[1] is the main variable and X[2] is the algebraic
  number.
>                           # inputs of ugcd are in recden format.

>     local G, st:
>    # RD:=Algebraic[RecursiveDensePolynomials]:
>
>    G := SRGCD(K1,K2,X[1],X[2],m,p):
>    if G <> FAIL then
>        G := mulrpoly(LC,G):
>         return G:
>     else
>         return FAIL:
>      fi:
>
> end:
```

```
> # Berlekamp Massey algorithm.
> BM := proc(s, N, P, x)
>    local C,B,T,L,k,i,n,d,b,binv,safemod;
>    #safemod := (exp, P) -> `if`(P=0, exp, exp mod P);
>     B := 1;
>     C := 1;
>     L := 0;
>     k := 1;
>     b := 1;
>    for n from 0 to 2*N-1 do
>         #d := s[n];
>       #for i from 1 to L do d := d + coeff(C,x,i)*s[n-i] mod P;
```

```
  od;
>       binv := 1/b mod P;
>       d := s[n] + add( coeff(C,x,i)*s[n-i], i=1..L ) mod P;
>       if d=0 then k := k+1 fi;
>       if (d <> 0 and 2*L > n) then
>         C := C - modp(binv*d,P)*expand(x^k*B);
>           k := k+1;
>         fi;
>       if (d <> 0 and 2*L <= n) then
>           T := C mod P;
>         C := C - modp(binv*d,P)*expand(x^k*B);
>           B := T;
>           L := n+1-L;
>           k := 1;
>           b := d;
>         fi;
>     od;
>    return C mod P;
> end:
```

```
> # Shifted Vandermonde system solver. p is the master polynomial
  which is also the reciprocal of the result of the BMA call.
> Vandermonde_Solve_Mod :=proc( A, W, MasterPoly, s, n, p )

>    local a, i, invs, j, P, q, Q, x, deno:

>    x := Array(1..n,0):

>    if n = 1 then
>      x[1] := modp( W[1]/(A[1]&^s), p ) :   # Special case when
  n=1.
>       return x:

>     fi:
>    Q := modp1( ConvertIn( MasterPoly, v ), p ):  # Master
  polynomial.
>    for i to n do
>      q := modp1( Quo( Q , modp1( ConvertIn( v-A[i] , v ), p )
  ) , p ):
>       a := modp1( Eval( q, A[i] ), p ):
>       invs := 1/a mod p:
>      P := modp( invs * modp1( ConvertOut( q ), p ), p ):
>      deno:= 1/(A[i]&^s) mod p:
>       x[i] := modp( add( W[j]*P[j]*deno, j=1..n ), p ):
>    od:

>    return x:
```

```
> end:

> PGCD:=proc(KA,KB,KL,X,M,p,ybound,d0,tau)   #ybound is for the
  support degree check.
>    local RD, result, si, dm, dx, dv, cont, coeffscube, s, j, w,
  E, A, B, LC, KR, i, k, K, CX1, unstable, BMtemp, AE, BE, LCE, AR,
  BR,LCR;
>    local rootsfail, LAMBDA, ROOTS, W1, n, Q1, ey, SUPPORTE, D0,
  m;
>    # RD:=Algebraic[RecursiveDensePolynomials]:
>    # KA and KB are in [x,y]. KL is in y.
>    printf("PGCD is called  p=%d\n",p);
>    if lcoeff(KA,X[1]) mod p = 0  or lcoeff(KB,X[1]) mod p = 0 or
  lcoeff(m,Z[1]) mod p = 0 then
>        return FAIL:
>     fi:

>    dm:=degree(M,X[-1]):
>    coeffscube:=Array(0..d0, 0..dm, -3..1000, 0):   # The first
  index denotes the degree of x,
>                                                    # The second
  index denotes the degree of z,
>                                                     # the third
  index denotes the array.
>                                                    # coeffscube
  [i,m,-3] stores the final monomials (exponents form).
>                                                    # coeffscube
  [i,m,-2] stores the feedback polynomial or the final answer: the
  coefficient of x^m*z^k.
>                                                    # coeffscube
  [i,m,-1] = 0 indicates coeff(x^m z^k)=0, no BMA required.
>                                                    # coeffscube
  [i,m,-1] = 1 indicates coeff(x^m z^k)<>0, run BMA on the array
  coeffscube[m,k].
>
>
>     s:=rand(p)():                   # Pick a random shift
>     j:=s:
>    w:=numtheory[primroot](p):      # Pick a generator
>    E:=w&^s mod p:                  # Construct an evaluation
  point.

>    for i from 0 to 2*tau-1 do
>
>       if Eval(lcoeff(KA,X[1]), X[2]=E) mod p = 0 and Eval
```

```
(lcoeff(KB,X[1]), X[2]=E) mod p = 0 then  # bad evaluation
>            return FAIL:
>          fi:
>        AE:=Eval(KA,y=E) mod p:
>        BE:=Eval(KB,y=E) mod p:
>        LCE:=Eval(KL,y=E) mod p:
>
>        AR:=rpoly(AE,[op(X[1]),op(X[-1])],M,p):
>        BR:=rpoly(BE,[op(X[1]),op(X[-1])],M,p):
>        LCR:=rpoly(LCE,[op(X[1]),op(X[-1])],M,p):

>        KR[i]:=ugcd(AR,BR,LCR,[op(X[1]),op(X[-1])],M,p):   #
compute the univariate GCD
>
>        if KR[i] = FAIL then return FAIL fi:    # Zero divisor
>
>        K[i]:=rpoly(KR[i]):  # convert the recden polynomial
structure to the Maple polynomial structure.

>        E:=E*w mod p:
>
>      od:
>
>     D0 := degree(K[0],X[1]):
>     for i from 1 to 2*tau-1 do
>        if degree(K[i],X[1]) <> D0 then return FAIL fi:    #check
degree of the GCD images.
>     od:

>     # record the coefficients of K[i] to coeffscube, Supp(K[i])
may not be the same as Supp(K[j]) if i <> j,
>
>     for i from 0 to 2*tau-1 do
>        for m from 0 to D0 do
>            CX1[m] := coeff(K[i],X[1],m):
>            if CX1[m] <> 0 then
>                for k from 0 to dm do
>                    if coeff(CX1[m],X[-1],k) <> 0 then
>                        if coeffscube[m,k,-1] = 0 then
>                            coeffscube[m,k,-1] := 1:   # 1 at
index -1 indicates that the coefficient is non-zero and need BMA
action.
>                            fi:
>                        coeffscube[m,k,i] := coeff(CX1[m],X[-1],
k):
>                        fi:
>                    od:
>                fi:
```

```
>         od:
>     od:

>    # Run the BMA.
>
>    for m from 0 to D0 do
>       for k from 0 to dm do
>           if coeffscube[m,k,-1] = 1 then
>             BMtemp:=BM(coeffscube[m,k], tau, p, v):   # use v
  as variable in the feedback polynomial.
>             coeffscube[m,k,-2]:=BMtemp:    # store the
  feedback polynomial to coeffscube at index -2.
>             fi:
>          dv:=degree(BMtemp,v):
>          LAMBDA:=add(v^(dv-n)*coeff(BMtemp,v,n), n=0..dv):  #
  get Lambda polynomial by computing the recripcal of BMtemp.
>          ROOTS:=Roots(LAMBDA) mod p:     # compute the roots.

>          if nops(ROOTS) = dv then    # otherwiseBM terminates
  early.
>             SUPPORTE:=[]:
>             ROOTS:=Array(1..dv, [seq(ROOTS[n][1],n=1..dv)]):
   # Convert to the array structure.
>               for n to dv do
>                ey:=numtheory[mlog](ROOTS[n],w,p):  # solve
  discrete log
>               SUPPORTE:=[op(SUPPORTE),ey]:   #SUPPORTE
  contains the exponents of y, not monomials of y.
>                od:
>
>             if max(SUPPORTE) <= ybound then   # degree of
  support is checked.
>                  W1:=Array(1..dv,0):

>                for n to dv do W1[n]:=coeffscube[m,k,n-1] od:
>              Q1:=Vandermonde_Solve_Mod(ROOTS, W1, LAMBDA,
  s, dv, p):
>                coeffscube[m,k,-2]:=add(Q1[n]*X[1]^m*X[-1]^k*
  X[2]^SUPPORTE[n], n=1..dv):
>                 coeffscube[m,k,-1]:=2:   # this coefficient
  is all done.
>                coeffscube[m,k,-3]:=SUPPORTE:
>                else
>                 return FAIL:      # feedback polynomial
  stabilized too early or all images are unlucky
>                 fi:
>             fi:
>         od:
```

```
>     od:
>
>    result:=0:
>    for m from 0 to D0 do
>       for k from 0 to dm do
>          if coeffscube[m,k,-1] = 2 then
>             result:=result+coeffscube[m,k,-2]:
>             fi:
>        od:
>     od:
>    return result:
>
> end:
```

```
> # Termbound can help to determine the term bound in the provided
  example.
> # You need to also know the number of terms in the scaling factor
  to determine the final term bound.
> termbound:=proc(G,main,alg)

>    local result, CX, i, CXZ, j:
>    result:=0:
>    CX:=[coeffs(g,x)]:
>    for i to nops(CX) do
>       CXZ[i]:=[coeffs(CX[i],z)]:
>       for j to nops(CXZ[i]) do
>          if whattype(CXZ[i][j]) <> `+` and result = 0 then
  result:=1 fi:
>          if whattype(CXZ[i][j]) = `+` and result < nops(CXZ[i]
  [j]) then result:=nops(CXZ[i][j]) fi:
>        od:
>     od:
>    return result:
> end:
```

```
> # Input polynomials must be primitive with respect to the main
  variable X[1].
```

```
> MGCD:=proc(A,B,tau,X,m,Ssize)        # X is a list of
  variables.
>                                      # X[1] is the main
  variable.
>                                      # X[-1] is the algebraic
  number.
>                                      # X[2..-2] are variables
  to be interpolated.
```

```
>                                             # m is the minimal
  polynomial.
>                                             # Ssize controls the size
  of the prime set S because |S| may be too large.
>




>    local SA, SB, SM, d, dm, Dm, h, n, LCA, LCB, LC, r, i, KA,
  KB, KL, s, Be, Ns, Bel, S;
>    local q, d0, M, YB, H, Hp, CR, Rki, Rkir, R, G, p:
>
>    # Convert inputs to their semi-associate representations.
>    SA:=expand(1/icontent(A)*A):
>    SB:=expand(1/icontent(B)*B):
>    SM:=expand(1/icontent(m)*m):
>
>    d:=max(seq(max(degree(A,X[i]),degree(B,X[i])),i=1..nops(X)-1)
  ):
>    dm:=degree(m,z):
>    Dm:=denom(m):
>    h:=max(norm(SA,infinity),norm(SB,infinity)):   # h >= hc in
  the thesis, because the conversion only introduces denominators
  to SA and SB.
>    n:=nops(X-1):

>    # cheat to get the scaling factor LC. We can use LCA or LCB
  instead which do not need GCD computation.
>    LCA:=lcoeff(a,X[1]):
>    LCB:=lcoeff(b,X[1]):
>    LC:=subs(RootOf(m)=z,evala(Gcd(subs(z=RootOf(m),LCA),subs(z=
  RootOf(m),LCB)))):

>     r:=[]:
>    for i from 2 to nops(X)-1 do
>       r:=[op(r),1+degree(SA,x[i])*degree(SB,X[1])+degree(SB,X
  [i])*degree(SA,X[1])]:
>     od:

>    KA:=Kron(SA,r,X[2..-2]):
>    KB:=Kron(SB,r,X[2..-2]):
>    KL:=Kron(LC,r,X[2..-2]):

>    # Construct S set.
>    s:=add(coeff(m,z,i)*Dm^(dm-i)*v^i, i=0..dm):
>    Be:= 2*(2*d^2+1)^n+2*d*(2*d^2+1)^n+2*d*(d+1)*(2*d^2+1)^n*dm +
  d*dm: printf("Primes p >= Be = %d \n", Be):
```

```
>    #W:=(2*d)^d*((2*d^2+1)^n*dm)^(2*d)*h^(2*d)*(Dm+norm(SM,
  infinity))^(2*d*(dm-1)+1):
>    #BZDP:=((2*dm)^dm*(dm*(d+1))^(2*dm)*h^(2*dm))*((2*dm)^dm*(2*
  d*(2*d^2+1)^n)^(2*dm)*W^(2*dm))^(d+1):

>     #dis:=resultant(diff(s,v),s,v):
>    #dc:=Dm^(dm-1):
>    #E:=evalf(exp(n*d)*dm*(dm-1)^((dm-1)/2)*norm(s,2)^(dm-1)*dis^
  (-1/2)*add(norm(s,2)^i,i=0..(dm-1))):  # note n, not n+1 since X
  starting with index 1.
>    #BUP:=(2*d)^d*(dm*(dm-1)*(2*d^2+1)^n*E*dis*dc^2*h^2)^(2*d)*
  (1+norm(s,infinity))^((4*d-1)*(dm-1)):

>    #BB:=max(BZDP,BUP):
>    #Ns:=evalf(4*log[4*4*tau*Be](BB)):    # X=4 and Y=4 in
  Theorem 21.
>
>     #if Ns > Ssize then Ns := Ssize fi:
>     Ns := Ssize:
>     Bel:=ceil(evalf(log[2](Be))):
>      S:=[]:
>    while nops(S) < Ns do
>        for q from 1 by 2 to 10000 while nops(S) < Ns do   #
  Choose 10000 to get smooth primes.
>            if isprime(2^Bel*q+1) then
>                 S:=[op(S),2^Bel*q+1]:
>                fi:
>          od:
>         Bel:=Bel+1:
>     od:
>
>    d0:=max(degree(A,X[1]),degree(B,X[1])):
>     M:=1:
>
>    YB:=max(degree(KA,y),degree(KB,y)):
>     H:=0:
>    p:=S[rand(nops(S))()]:

>     # Loop
>
>     do


>      while M mod p = 0  do p:=S[rand(nops(S))()] od:    #
  randomly pick a prime.
>      #printf("The prime for the PGCD call is %d.\n", p):
>      Hp:=PGCD(KA, KB, KL, X, SM, p, YB, d0, tau):
```

```
>       if Hp <> FAIL then
>          if degree(Hp,X[1]) < d0 then
>             d0:=degree(Hp,X[1]):
>             M:=p:    # reset modulus
>             H:=Hp:   # reset Chinese remaindering image.
>             p:=S[rand(nops(S))()]:                    #
  randomly pick a prime.
>          elif degree(Hp,X[1]) = d0 then
>             CR:=chrem([H,Hp],[M,p]):
>             M:=M*p:
>             H:=CR:
>             printf("Rational number reconstruction is called
  and the result is.\n"):
>             R:=iratrecon(H,M):  print(R):
>             if R <> FAIL and Hp = R mod p then
>                Rki := Kroninv(R,r,X[2..-2],X[1]):
>                Rkir := subs(X[-1]=RootOf(m),Rki):
>                G := evala(Primpart(Rkir, X[1])):   # Cheating
  step, use Maple Primpart to remove contents.
>                if evala(Divide(subs(X[-1]=RootOf(m),A),G))
  then
>                   return subs(RootOf(m)=X[-1],G):
>                   fi:
>                fi:
>             p:=S[rand(nops(S))()]:                # randomly
  pick a prime.
>             fi:
>          else
>          p:=S[rand(nops(S))()]:                   # randomly
  pick a prime.
>          fi:
>     od:

> end:
> # trace(MGCD);
```

```
> # First example, two variables, [x,y], no Kroncker substitution
  is required, the size of coefficients is large and it
> # requires more than 2 primes.
```

```
> m:=z^3 + 2*z^2 + 1;  irreduc(m);
```

$$m := z^3 + 2\,z^2 + 1$$

$$true \quad (2)$$

```
> X:=[x,y,z];
```

$$X := [x, y, z] \quad (3)$$

```
> g:=z^2*x^2 + (randpoly([y],terms=5,degree=5,coeffs=rand(-2^32.
```

```
.2^32)))*z^2/6*x + (randpoly([y],terms=5, degree=5,coeffs=rand
(-2^32..2^32)))*z/17 + 1;
```

$$g := z^2 x^2 + \frac{1}{6}\left(\left(545404204\, y^5 + 2715962298\, y^4 + 1196140740\, y^3 - 182506777\, y^2 - 150802599\right) z^2 x\right) + \frac{1}{17}\left(\left(3117454609\, y^5 - 867128743\, y^4 - 3019235525\, y^2 - 4274422387\, y + 483031418\right) z\right) + 1 \quad (4)$$

```
> abar:=randpoly(X,terms=3);
```

$$abar := 72\, x^2 y^3 + 37\, x y^3 z + 74\, y^2 \quad (5)$$

```
> bbar:=randpoly(X,terms=3);
```

$$bbar := -23\, x^2 z + 10\, x z^2 + 98 \quad (6)$$

```
> a:=expand(abar*g):
> a:=rem(a,m,z):
> b:=expand(bbar*g):
> b:=rem(b,m,z):
> # trace(MGCD);
> # trace(PGCD);
> # trace(ugcd);
> # trace(SRGCD);
> st:=time(): GG:=MGCD(a,b,15,X,m,20); time()-st;     # Generate
only 20 primes in this example, the term bound is 15.
```

```
Primes p >= Be = 7488474
PGCD is called  p=3112173569
PGCD is called  p=1484783617
Rational number reconstruction is called and the result is.
```

FAIL

```
PGCD is called  p=595591169
Rational number reconstruction is called and the result is.
```

$$-\frac{3117454609}{17} z^2 y^5 + \frac{272702102}{3} x y^5 - \frac{6234909218}{17} y^5 z + \frac{867128743}{17} z^2 y^4 + 452660383\, x y^4 + \frac{1734257486}{17} y^4 z + 199356790\, x y^3 + \frac{3019235525}{17} z^2 y^2 - \frac{182506777}{6} x y^2 + \frac{6038471050}{17} y^2 z + 251436611\, z^2 y + x^2 + 502873222\, y z - \frac{483031418}{17} z^2 - \frac{50267533}{2} x - \frac{966062853}{17} z - 2$$

$$GG := -18704727654\, z^2 y^5 + 9271871468\, x y^5 - 37409455308\, y^5 z + 5202772458\, z^2 y^4 + 46171359066\, x y^4 + 10405544916\, y^4 z$$

$$+20334392580\,x\,y^3+18115413150\,z^2y^2-3102615209\,x\,y^2+36230826300\,y^2z+25646534322\,z^2y+102\,x^2+51293068644\,y\,z-2898188508\,z^2-2563644183\,x-5796377118\,z-204$$

$$0.474 \tag{7}$$

```
> g;
```

$$z^2x^2+\frac{1}{6}\left(\left(545404204\,y^5+2715962298\,y^4+1196140740\,y^3-182506777\,y^2-150802599\right)z^2x\right)+\frac{1}{17}\left(\left(3117454609\,y^5-867128743\,y^4-3019235525\,y^2-4274422387\,y+483031418\right)z\right)+1 \tag{8}$$

```
> lcoeff(GG,x);    # the Leading coefficient computed by our code. There should be no z in the leading coefficient, integer is fine.
```

$$102 \tag{9}$$

```
> lcoeff(g,x);     # the leading coefficient of g which is used to create this problem and it is not monic with respect to x and y.
```

$$z^2 \tag{10}$$

```
> # Second example, four variables, [x,y,v,u], Kronecker substitution is used.
```

```
> m:=z^2 + 2;  irreduc(m);
```

$$m := z^2+2$$

$$true \tag{11}$$

```
> X:=[x,y,v,u,z];
```

$$X := [x, y, v, u, z] \tag{12}$$

```
> g:=((1/2*z+4/3*v)*y)*x^2 + (randpoly([y,v,u],terms=5,degree=10))*z^2/6*x + (randpoly([y,v,u],terms=5, degree=10))*z/17 + 1;
```

$$g := \left(\frac{z}{2}+\frac{4\,v}{3}\right)y\,x^2+\frac{\left(45\,y^5u^5+49\,v^9u-31\,y^2v^6-44\,v^4u^3-85\,y\,v\,u^3\right)z^2x}{6}+\frac{\left(-54\,y^2v^4u^4-8\,y^4u^5-96\,y\,v^7+83\,y^2v^4u-14\,v\right)z}{17}+1 \tag{13}$$

```
> abar:=(z*y+1)*x+y+z+v;
```

$$abar := (y\,z+1)\,x+y+z+v \tag{14}$$

```
> bbar:=(z*y+1)*x+y+2*z*u;
```

$$bbar := (y\,z+1)\,x+y+2\,z\,u \tag{15}$$

```
> a:=expand(abar*g):
```

```
> a:=rem(a,m,z):
> b:=expand(bbar*g):
> b:=rem(b,m,z):
> # trace(MGCD);
> # trace(PGCD);
> # trace(ugcd);
> st:=time(): GG:=MGCD(a,b,15,X,m,20); time()-st;     # Generate
  only 20 primes in this example, the term bound is 15.
```

Primes p >= Be = 18665282
PGCD is called p=15065939969
PGCD is called p=11173625857
Rational number reconstruction is called and the result is.

$$\frac{3}{8}x^2y-\frac{21}{34}y^{19}-\frac{6}{17}y^{2664}-\frac{72}{17}y^{134}-\frac{45}{4}y^{2666}x+\frac{31}{4}y^{117}x-\frac{6}{17}y^{2665}z-\frac{21}{34}y^{20}z+y^{21}x^2+\frac{249}{68}y^{610}-\frac{81}{34}y^{2206}-\frac{72}{17}y^{135}z-\frac{85}{8}y^{1616}xz+\frac{85}{4}y^{1617}x+\frac{49}{8}y^{703}xz-\frac{31}{8}y^{116}xz+11y^{1673}x-\frac{11}{2}y^{1672}xz+\frac{3}{4}y-\frac{3}{8}z-\frac{1}{2}y^{20}x^2z-\frac{81}{34}y^{2207}z+\frac{249}{68}y^{611}z+\frac{45}{8}y^{2665}xz-\frac{49}{4}y^{704}x+\frac{3}{8}x^2y^2z$$

$$GG := -1530u^5xy^5-324zy^2v^4u^4-1666uv^9x-48zy^4u^5-576zyv^7+1054v^6xy^2+1496u^3v^4x+498zy^2v^4u+2890u^3vxy+136x^2yv+51x^2yz-84vz+102$$

$$0.359 \quad (16)$$

```
> g;
```

$$\left(\frac{z}{2}+\frac{4v}{3}\right)yx^2+\frac{(45y^5u^5+49v^9u-31y^2v^6-44v^4u^3-85vyu^3)z^2x}{6}+\frac{(-54y^2v^4u^4-8y^4u^5-96yv^7+83y^2v^4u-14v)z}{17}+1 \quad (17)$$

```
> lcoeff(GG,x);     # the Leading coefficient computed by my code.
  (The term 136vy does not have the variable z)
```

$$136vy+51yz \quad (18)$$

```
> lcoeff(g,x);      # the leading coefficient of g which is used to
  create this problem.
```

$$\left(\frac{z}{2}+\frac{4v}{3}\right)y \quad (19)$$