```
> with(LinearAlgebra):
  interface(rtablesize=50);
```

50 **(1)**

```
> #Maple returns degree(0,x) as -inf,
  #we need it to return 0 for indexing
  degree2 := proc(f,x)

  local d;
  d := degree(f,x);
  if d = -infinity then return 0; fi;

  return d;


  end proc:
> expandSpace := proc(F,Space,dCur,dNew,k)

  local i,A;

  if(dNew+1 <= Space) then
     return F,Space;
  fi;

  #create new array
  A := Array(1..2^k);

  #move elements over
  for i from 1 to dCur+1 do
     A[i] := F[i];
  od;

  return A,2^k;

  end proc:
> #Calculates M,S polynomials for Diophantine  Equations
  #Verifies the initial n input polynomials are relatively prime
  MultiEEA := proc(U::list,x::name,p::prime)
  local n,M,sis,i,g,s,t,Mpolys,Spolys;

     #local variables
     n := nops(U);
     Mpolys := Array(2..n);
     Spolys := Array(1..n-1);
     M := 1;

     #Generate M
     for i from n by -1 to 2 do
        M := Expand(M*U[i]) mod p;
        Mpolys[i] := M;
     od;

     Mpolys := convert(Mpolys,list);
     sis := NULL;

     #Generate S, verify polynomials relatively prime
     for i from 1 to n-1 do
        g := Gcdex(Mpolys[i],U[i],x,'Spolys[i]') mod p;
        if g=1 then sis := sis,s; else return FAIL,FAIL,FAIL; fi;
     od;
     [sis],Mpolys,convert(Spolys,list);
```

```
end:
> #Solves the polynomial Diophantine Equation
DiophantineN := proc(U,c,M,S,p,x)

local n,q,g,ck,i,s,t,Sigmas;

n := nops(U);
ck := c;
Sigmas := Array(1..n);
for i from 1 to n-1 do
   Sigmas[i] := Rem(ck*S[i],U[i],x) mod p;
   ck := Quo(ck-Sigmas[i]*M[i],U[i],x) mod p;
end do;
Sigmas[n] := ck;
return(convert(Sigmas,list));
end proc:
> #calculates coeff(f_1*f_2*...*f_n,(y-alpha)^k) and stores the
results in G
coeffExtract := proc(p,alpha,k,H,Space,Degrees,n,dy)

   local i,j,m,s,t,L,D,MIN,MAX,Delta;

   t := n; s := 1;
   while(t > 1) do
      for i from 0 to floor(t/2)-1 do
         H[s+t+i],Space[s+t+i] := expandSpace(H[s+t+i],Space[s+t+
i],Degrees[s+t+i],Degrees[s+2*i]+Degrees[s+2*i+1],ceil(log[2]
(Degrees[s+2*i]+Degrees[s+2*i+1]+1)));
         Degrees[s+t+i] := Degrees[s+2*i] + Degrees[s+2*i+1];
         if k <= Degrees[s+2*i] + Degrees[s+2*i+1] then
            MIN := max(0,k-Degrees[s+2*i+1]);
            MAX := min(k,Degrees[s+2*i]);
            for L from MIN to MAX do
               H[s+t+i][k+1] := H[s+t+i][k+1] + H[s+2*i][L+1]*H
[s+2*i+1][k-L+1] mod p;
            od;
         fi;
         #D[i] := D[2*i] + D[2*i+1];
      od;

      if t mod 2 = 1 then
         H[s+t+i],Space[s+t+i] := expandSpace(H[s+t+i],Space[s+t+
i],Degrees[s+t+i],Degrees[s+2*i],ceil(log[2](Degrees[s+2*i]+1)));
         if Degrees[s+2*i] >= k then
              Degrees[s+t+i] := Degrees[s+2*i];
              H[s+t+i][k+1] := H[s+2*i][k+1] mod p;
         fi;
      fi;

      s := s+t; t := ceil(t/2);
   od;

   if k+1 > Space[s] then
      Delta := 0;
   else
      Delta := H[s][k+1];
   fi;

   return Delta,H,Space,Degrees;
```

```
end proc:
> #Fills in missing values in G
coeffUpdate := proc(p,alpha,k,H,Space,D,n)

   local i,s,t,L,T;

   s := 1; t := n;
   while(t > 1) do
      for i from 0 to floor(t/2)-1 do

         if s=1 then
            T[i] := 0;
            if D[2*i+1] = k then T[i] := H[s+2*i][k+1]*H[s+2*i+1]
[0+1] mod p; fi;
            if D[2*i+2] = k then T[i] := T[i] + H[s+2*i][0+1]*H
[s+2*i+1][k+1] mod p; fi;
         else
            T[i] := H[s+2*i+1][0+1]*T[2*i] + H[s+2*i][0+1]*T[2*i+1]
mod p;
         fi;
         H[s+t+i],Space[t+s+i] := expandSpace(H[s+t+i],Space[t+s+
i],D[s+t+i],D[s+2*i] + D[s+2*i+1],ceil(log[2](D[s+2*i] + D[s+2*i+1]
+1)));

         D[s+t+i] := D[s+2*i] + D[s+2*i+1];

         H[s+t+i][k+1] := H[s+t+i][k+1] + T[i] mod p;
      od;
      if(t mod 2 = 1) then
         T[i] := H[s+2*i][k+1];
         H[s+t+i],Space[t+s+i] := expandSpace(H[s+t+i],Space[t+s+
i],D[s+t+i],D[s+2*i],ceil(log[2](D[s+2*i]+1)));
         D[s+t+i] := D[s+2*i];
         H[t+s+i][k+1] := T[i];
      fi;
      s := s+t; t := ceil(t/2);
   od;

   return H,Space,D;

end proc:
> CubicBivariateHensel := proc(A::polynom,F0::list,x::name,y::name,
alpha::integer,p::prime)
#A(x,y) - polynomial to factor
#F0 - list of monic, relatively prime polynomials in x s.t. the
product is equal to...
#x - variable 1 (usually x)
#y - variable 2 (usually y)
#alpha - integer to use for calculating the taylor coeff.
#p - prime

#local variables
local n,m,B,F,W,G,i,j,k,t,ck,sigmas,deltas,Delta,Coeffs,evalPoints,
dx,dy,T,M,S,Space,Degrees,numPolys;

n := nops(F0);
F := F0;
dx := degree(A,x);
dy := degree(A,y);

#find the number of polynomials we will calculate
```

```
t := n;
numPolys := 1;
while t > 1 do
   numPolys := numPolys + t;
   t := ceil(t/2);
od;

#initial arrays/lists
deltas := [seq(0,i=1..dx)];
evalPoints := [seq(i,i=0..dx-1)];

W := Array(1..dx);
G := Array(1..dx);
Degrees := Array(1..dx);
Space := Array(1..dx);
for i from 1 to dx do
   G[i] := Array(1..numPolys);
   for j from 1 to numPolys do
      G[i][j] := Array(1..1);
   od;
   Space[i] := Array([seq(1,i=1..numPolys)]);
   Degrees[i] := Array(1..numPolys);
od;


#get a taylor series around (y-alpha) (does NOT use Shaw and
Traub's method)
B := taylor(A,y=alpha,dy+1);

#Solve for M polynomials to use for the Diophantine Equation
(Optimization)
T,M,S := MultiEEA(F0,x,p); # Solve this once for re-use

#do initial evaluations
for i from 1 to dx do
   for j from 1 to n do
      G[i][j][1] := Eval(F0[j],x=evalPoints[i]) mod p;
   od;
   deltas[i],G[i],Space[i],Degrees[i] := coeffExtract(p,alpha,0,G
[i],Space[i],Degrees[i],n);
od;

#print(G[1]);

#if the EEA failed
if (T,M) = (FAIL,FAIL) then
   return "Initial Factors are not relatively prime";
fi;

#main loop
for k from 1 to dy do

   #printf("k=%d\n",k);

   #CoefficientExtraction
   for i from 1 to dx do
      deltas[i],G[i],Space[i],Degrees[i] := coeffExtract(p,alpha,k,
G[i],Space[i],Degrees[i],n);
   od;

   #Interpolation
   Delta := Interp(evalPoints,deltas,x) mod p;
```

```
    #print("got here");

    ck := Expand(coeff(B,(y-alpha),k) - Delta) mod p;

    if add(degree(F[i],y),i=1..n) = dy and ck <> 0 then
       return (FAIL);
    fi;

    if ck <> 0 then

       #Solve Diophantine Equation for coefficients
       sigmas := DiophantineN(F0,ck,M,S,p,x);

       #Update the values of F
       for i from 1 to n do
          F[i] := F[i] + sigmas[i]*(y-alpha)^k;
       od;

       #Update Evaluations
       for i from 1 to dx do
          for j from 1 to n do
             t := Eval(sigmas[j],x=evalPoints[i]) mod p;
             if t <> 0 then
                G[i][j],Space[i][j] := expandSpace(G[i][j],Space[i]
 [j],Degrees[i][j],k,ceil(log[2](k+1)));
                Degrees[i][j] := k;
                G[i][j][k+1] := t;
             fi;
          od;
       od;

       #Perform CoefficientUpdate
       for i from 1 to dx do
          G[i],Space[i],Degrees[i] := coeffUpdate(p,alpha,k,G[i],
 Space[i],Degrees[i],n);
       od;


    fi;
 od;

 #return bivar polynomials or fail
 if add(degree(F[i],y),i=1..n) = dy then
    return(F);
 else
    return(FAIL);
 fi;

 end proc:
```

```
> #`mod` := mods;
> p := 1009;
```

$$p := 1009 \quad (2)$$

```
> alpha := 3;
```

$$\alpha := 3 \quad (3)$$

```
> f1 := x^4 + randpoly([x,y],dense,degree=3);
  f2 := x^4 + randpoly([x,y],dense,degree=3);
  f3 := x^4 + randpoly([x,y],dense,degree=3);
  f4 := x^4 + randpoly([x,y],dense,degree=3);
```

$$f1 := x^4 + 71x^3 - 12x^2y + 22y^3 + 72x^2 + 32xy + 88y^2 + 14x - 68y + 95$$
$$f2 := x^4 - 57x^3 - 80x^2y - 64xy^2 + 14y^3 + 98x^2 - 97xy - 61y^2 - 86x + 83y - 41$$
$$f3 := x^4 + 66x^3 + 62x^2y + 12xy^2 + 12y^3 + 75x^2 - 22xy + 82y^2 - 19x + y + 31$$
$$f4 := x^4 - 10x^3 + 5x^2y + 94xy^2 + 94y^3 + 45x^2 - 30xy + 32y^2 - 74x + 38y - 32 \quad (4)$$

```
> A := Expand(f1*f2*f3*f4) mod p;
```

$$A := x^{16} + 70x^{15} + 984x^{14}y + 42x^{13}y^2 + 142x^{12}y^3 + 403x^{14} + 473x^{13}y + 741x^{12}y^2 + 931x^{11}y^3 + 45x^{10}y^4 + 907x^9y^5 + 207x^8y^6 + 199x^{13} + 101x^{12}y + 602x^{11}y^2 + 515x^{10}y^3 + 482x^9y^4 + 851x^8y^5 + 506x^7y^6 + 370x^6y^7 + 387x^5y^8 + 608x^4y^9 + 723x^{12} + 322x^{11}y + 228x^{10}y^2 + 596x^9y^3 + 538x^8y^4 + 304x^7y^5 + 62x^6y^6 + 857x^5y^7 + 654x^4y^8 + 215x^3y^9 + 154x^2y^{10} + 598xy^{11} + 328y^{12} + 643x^{11} + 788x^{10}y + 885x^9y^2 + 107x^8y^3 + 164x^7y^4 + 399x^6y^5 + 945x^5y^6 + 909x^4y^7 + 800x^3y^8 + 135x^2y^9 + 22xy^{10} + 133y^{11} + 704x^{10} + 105x^9y + 119x^8y^2 + 466x^7y^3 + 165x^6y^4 + 504x^5y^5 + 51x^4y^6 + 747x^3y^7 + 448x^2y^8 + 434xy^9 + 682y^{10} + 597x^9 + 453x^8y + 440x^7y^2 + 796x^6y^3 + 767x^5y^4 + 580x^4y^5 + 585x^3y^6 + 231x^2y^7 + 935xy^8 + 233y^9 + 20x^8 + 451x^7y + 304x^6y^2 + 198x^5y^3 + 875x^4y^4 + 74x^3y^5 + 581x^2y^6 + 90xy^7 + 557y^8 + 226x^7 + 8x^6y + 797x^5y^2 + 441x^4y^3 + 931x^3y^4 + 656x^2y^5 + 600xy^6 + 558y^7 + 990x^6 + 951x^5y + 318x^4y^2 + 14x^3y^3 + 270x^2y^4 + 166xy^5 + 770y^6 + 936x^5 + 968x^4y + 561x^3y^2 + 362x^2y^3 + 51xy^4 + 90y^5 + 628x^4 + 483x^3y + 332x^2y^2 + 682xy^3 + 600y^4 + 810x^3 + 453x^2y + 690xy^2 + 556y^3 + 766x^2 + 519xy + 900y^2 + 73x + 976y + 379 \quad (5)$$

```
> f10 := Eval(f1,y=alpha) mod p;
  f20 := Eval(f2,y=alpha) mod p;
  f30 := Eval(f3,y=alpha) mod p;
  f40 := Eval(f4,y=alpha) mod p;
```

$$f10 := x^4 + 71x^3 + 36x^2 + 110x + 268$$
$$f20 := x^4 + 952x^3 + 867x^2 + 56x + 37$$
$$f30 := x^4 + 66x^3 + 261x^2 + 23x + 87$$
$$f40 := x^4 + 999x^3 + 60x^2 + 682x + 890 \quad (6)$$

```
> F0 := [f10,f20,f30,f40];
```

$$F0 := [x^4 + 71x^3 + 36x^2 + 110x + 268, x^4 + 952x^3 + 867x^2 + 56x + 37, x^4 + 66x^3 + 261x^2 + 23x + 87, x^4 + 999x^3 + 60x^2 + 682x + 890] \quad (7)$$

```
> Rem(Expand(A - mul(F0)) mod p,(y-3),y) mod p;
```

$$0 \quad (8)$$

```
> C := CubicBivariateHensel(A,F0,x,y,alpha,p):
  Expand(C) mod p;
```

$$[x^4 + 71x^3 + 997x^2y + 22y^3 + 72x^2 + 32xy + 88y^2 + 14x + 941y + 95, x^4 + 952x^3 \quad (9)$$

$+ 929 x^2 y + 945 x y^2 + 14 y^3 + 98 x^2 + 912 x y + 948 y^2 + 923 x + 83 y + 968, x^4$
$+ 66 x^3 + 62 x^2 y + 12 x y^2 + 12 y^3 + 75 x^2 + 987 x y + 82 y^2 + 990 x + y + 31, x^4$
$+ 999 x^3 + 5 x^2 y + 94 x y^2 + 94 y^3 + 45 x^2 + 979 x y + 32 y^2 + 935 x + 38 y + 977]$

```
> Expand(A-mul(C)) mod p;
```

$$0 \tag{10}$$