```
> with(LinearAlgebra):
  interface(rtablesize=50);
                                  10                                  (1)
> #This function calculates degree(f,x) except it returns 0 from
  degree(0,x) instead of -infinity
  degree2 := proc(f,x)

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

  return d;


  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,F,G,n,dy)

   local i,j,H,Delta,d,D,MIN,MAX;

   H := G;
   Delta := 0;

   #if the number of factors
      MIN := max(0,k-degree2(F[2],y));
      MAX := min(k,degree2(F[1],y));
      for j from MIN to MAX do
         H[2,1+k] := H[2,1+k] + coeff(F[1],y-alpha,j)*coeff(F[2],
y-alpha,k-j) mod p;
      od;
      d := degree2(F[1],y) + degree2(F[2],y);
      for i from 3 to n do
         D := d;
         d := d + degree2(F[i],y);
         if k <= d then
            MIN := max(0,k-D);
            MAX := min(k,degree2(F[i],y));
            for j from MIN to MAX do
               H[i,1+k] := H[i,1+k] + H[i-1,k-j+1]*coeff(F[i],y-
alpha,j) mod p;
            od;
         fi;
      od;


   return H[n,k+1],H;


end proc:
> #Fills in missing values in G
coeffUpdate := proc(p,alpha,k,F,G,n,dy)

   local i,t,H;

   H := G;

   if n > 2 then
      t := coeff(F[1],y-alpha,k);
      H[1,k+1] := t;
      for i from 2 to n do
         t := coeff(F[i],y-alpha,0)*t + coeff(F[i],y-alpha,k)*H
[i-1,1] mod p;
         H[i,k+1] := H[i,k+1] + t mod p;
      od;
   fi;

   return H;

end proc:
```

```
> QuarticBivariateHensel := 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. A -
  F0_1*F0_2*...*F0_n equiv 0 mod (y-alpha)
  #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,E,Evals,G,i,j,k,t,ck,sigmas,Delta,Coeffs,CoeffsMul,
  dx,dy,T,M,S;

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


  #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


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

  #set up G (CoeffExtract matrix)
  G := Matrix(n,dy+1);
  G[1,1] := F0[1];
  for i from 1 to n-1 do
     G[i+1,1] := Expand(G[i,1]*F0[i+1]) mod p;
  od;

  #main loop
  for k from 1 to dy do

     #CoefficientExtraction
     Delta,G := coeffExtract(p,alpha,k,F,G,n,dy);

     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);
```

```
        #print(sigmas);

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

        #Perform CoefficientUpdate
        #print(F); print(G);
        G := coeffUpdate(p,alpha,k,F,G,n,dy);


     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;
```

$$p := 1009 \quad \textbf{(2)}$$

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

$$\alpha := 3 \quad \textbf{(3)}$$

```
> #Calculate the polynomial to factor: A
  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);
  A := Expand(f1*f2*f3*f4) mod p:
```

$$f1 := x^4 - 7x^3 + 22x^2y - 94xy^2 - 55x^2 + 87xy - 62y^2 - 56x + 97y - 73$$
$$f2 := x^4 - 4x^3 - 83x^2y + 62xy^2 - 44y^3 - 10x^2 - 82xy + 71y^2 + 80x - 17y - 75$$
$$f3 := x^4 - 10x^3 - 7x^2y + 42xy^2 + 75y^3 - 40x^2 - 50xy - 92y^2 + 23x + 6y + 74$$
$$f4 := x^4 + 72x^3 + 37x^2y + 87xy^2 + 98y^3 - 23x^2 + 44xy - 23y^2 + 29x + 10y - 61 \quad \textbf{(4)}$$

```
> #Calculate the initial factors (A - f10*f20*f30*f40) mod (y-
  alpha) = 0
  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;
  F0 := [f10,f20,f30,f40]:
```

$$f10 := x^4 + 1002x^3 + 11x^2 + 368x + 669$$
$$f20 := x^4 + 1005x^3 + 750x^2 + 392x + 334$$
$$f30 := x^4 + 999x^3 + 948x^2 + 251x + 280$$
$$f40 := x^4 + 72x^3 + 88x^2 + 944x + 390 \quad \textbf{(5)}$$

```
> #check (A - f10*f20*f30*f40) mod (y-alpha) = 0
  Rem(Expand(A - mul(F0)) mod p,(y-3),y) mod p;
```

$$0 \tag{6}$$

```
> #Run the Quartic algorithm and check that it worked
  F := QuarticBivariateHensel(A,F0,x,y,alpha,p):
  Expand(A - mul(F)) mod p;
```

$$0 \tag{7}$$