```
--> load ( gcdex ) $ load ( grobner ) $ load ( eigen ) $ load ( "nchrpl" ) $
    /* load(sym)$  load("sym/compile")$ */

--> /*==========       subroutine 1 ==========================*/

--> /*
    以下は逆元を求めるサブルーチンプログラム
    使い方

    A:    逆元を求めたい元
    Const: 代数体を規定しているリスト

    例えば    Const:[ [v,gv[0]],[a[1],B[1]]] など
    gv[0]: 最小多項式    変数v
    B[1]:  添加される数a[1]が満たす冪根方程式

    具体的には    Const:[[v,v^8-v^6+v^4-v^2+1],[a[1],a[1]^2-5/4]]$

    */
--> Inverse ( A , Const ) : = block (

      [ Nconst , vn , Tn , rwn , rpn ] ,

    Nconst : length ( Const ) ,
    vn : Nconst ,
    varL : [ ] ,
    maxpL : [ ] ,

    Tn : 1 ,
    for i : 1 thru Nconst do (

      con : Const [ i ] ,
      y [ i ] : con [ 1 ] , varL : endcons ( y [ i ] , varL ) ,
      D [ i ] : con [ 2 ] ,
      pw [ i ] : hipow ( D [ i ] , y [ i ] ) , maxpL : endcons ( pw [ i ] , maxpL ) , Tn : Tn * pw [ i ]

      ) ,
      /*    基底の次数の組み合わせ



    Tn:    pw[1]*pw[2]*pw[3]   2*3*2=12
    pwL:   基底次数の組み合わせのリスト
```

```
        [[0,0,0],[0,0,1],[0,1,0],[0,1,1],[0,2,0],[0,2,1],
          [1,0,0],[1,0,1],[1,1,0],[1,1,1],[1,2,0],[1,2,1]]

rwn:   繰り返し同じ数を書く数(re-wright number)
rpn:   (repetition number)
       例えば上の例ではリスト[x1,x2,x3]のx3の例でいうとx2だけをみると
              [0,0,1,1,2,2,0,0,1,1,2,2]となっているが、この
       [0,0,1,1,2,2]の繰り返しが2回繰り返されている。
       この数を指定する数
     Tn    pw[i]   rwn  rpr
     12    ÷ 2    =  6    1=Tn/(pw[1]*rwn)
      6    ÷ 3    =  2    2=Tn/(pw[2]*rwn)
      2    ÷ 2    =  1    6=Tn/(pw[3]*rwn)

     という関係式があるので下記のプログラムとなる

   */

pwL : [ ] ,
for i : 1 thru Tn do ( pwL : endcons ( [ ] , pwL ) ) ,

rwn : Tn ,
for i : 1 thru Nconst do (

  rwn : rwn / pw [ i ] ,
  rpn : Tn / ( pw [ i ] * rwn ) ,
  n : 1 ,

for j : 1 thru rpn do (

for k : 1 thru pw [ i ] do (

for m : 1 thru rwn do (
       pwL [ n ] : endcons ( k - 1 , pwL [ n ] ) , n : n + 1
       )
     )
     )
     ) ,
baseL : [ ] ,

for i : 1 thru Tn do (
  b : 1 ,
  for j : 1 thru vn do ( b : b * varL [ j ] ^ pwL [ i ] [ j ] ) ,
```

```
    baseL : endcons ( b , baseL )

  ) ,
  coefL : makelist ( c [ i ] , i , 0 , Tn - 1 ) ,
  F : sum ( coefL [ i ] * baseL [ i ] , i , 1 , Tn ) ,
  W : F * A ,

  for i : 1 thru Nconst do (
  W : remainder ( W , Const [ i ] [ 2 ] , varL [ i ] )
  ) ,
  W : expand ( W ) ,
  CeL : [ ] ,
  for i : 1 thru Tn do (
    ce : W ,
    for j : 1 thru vn do ( ce : coeff ( ce , varL [ j ] , pwL [ i ] [ j ] ) ) ,

    CeL : endcons ( ce , CeL )

    ) ,
    CeL [ 1 ] : CeL [ 1 ] - 1 ,
  SLV : solve ( CeL , coefL ) ,
  IA : sum ( coefL [ i ] * baseL [ i ] , i , 1 , Tn ) ,
  IA : ev ( IA , SLV )
  ) $

--> /*==========      subroutine 2 ==========================*/

--> /*
  以下は、代数体の次数を低減させるサブルーチンプログラム

  Const: 代数体を規定しているリスト

  例えば　Const:[ [v,gv],[a[1],B[1]]] など
  gv[i]: 現在の体で使用されている v　の最小多項式
  B[1]: 添加される数a[1]が満たす冪根方程式

  具体的には　Const:[[v,v^8-v^6+v^4-v^2+1],[a[1],a[1]^2-5/4]]$


    */

--> Reduce ( A , Const ) : = block (

  crn : length ( Const ) ,
```

```
    for i : 1 thru crn do ( A : remainder ( A , Const [ i ] [ 2 ] , Const [ i ] [ 1 ] ) ) ,

    A

    ) $
--> /*==========       main program ==========================*/
--> fx : x ^ 4 + 4 * x + 2 ;
--> N : 4 ; NN : N ! ;
--> rtL : [ α , β , γ , δ ] ;
    fx1 : divide ( fx , x - α , x ) ;
    fx2 : divide ( fx1 [ 1 ] , x - β , x ) ;
    fx3 : divide ( fx2 [ 1 ] , x - γ , x ) ;
    fx4 : divide ( fx3 [ 1 ] , x - δ , x ) ;
--> /*  v[i]の計算の時に必要*/
--> cL : [ 1 , 2 , 3 , 4 ] ; zL : [ u1 , u2 , u3 , u4 ] ;
    V ( cL , rtL ) : = sum ( cL [ i ] * rtL [ i ] , i , 1 , N ) ;
--> V ( cL , rtL ) ;
--> /* α,β,γの計算の時に必要*/
--> prtL : listify ( permutations ( rtL ) ) ;
--> viL : makelist ( v [ i ] , i , 1 , NN ) ;
--> pVrtL : makelist ( V ( cL , prtL [ i ] ) , i , 1 , NN ) ;
--> virtL : map ( lambda ( [ x , y ] , x = y ) , viL , pVrtL ) ;
--> awL : [ 1 , 0 , 0 , 0 ] ; bwL : [ 0 , 1 , 0 , 0 ] ; cwL : [ 0 , 0 , 1 , 0 ] ; dwL : [ 0 , 0 , 0 , 1 ] ;
--> pawL : makelist ( V ( awL , prtL [ i ] ) , i , 1 , NN ) ;
    pbwL : makelist ( V ( bwL , prtL [ i ] ) , i , 1 , NN ) ;
    pcwL : makelist ( V ( cwL , prtL [ i ] ) , i , 1 , NN ) ;
    pdwL : makelist ( V ( dwL , prtL [ i ] ) , i , 1 , NN ) ;
--> vxL : makelist ( x - v [ i ] , i , 1 , NN ) ;
--> awxL : makelist ( pawL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;
    bwxL : makelist ( pbwL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;
    cwxL : makelist ( pcwL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;
    dwxL : makelist ( pdwL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;
--> Vx : apply ( "*" , vxL ) ;
--> /*   α, β, γ ,δ の計算   の準備   */
```

```
--> Pax : 0 $
    for i : 1 thru NN do ( Pax : Pax + Vx * awxL [ i ] ) $
    Pax $
    Pbx : 0 $
    for i : 1 thru NN do ( Pbx : Pbx + Vx * bwxL [ i ] ) $
    Pbx $
    Pcx : 0 $
    for i : 1 thru NN do ( Pcx : Pcx + Vx * cwxL [ i ] ) $
    Pcx $
    Pdx : 0 $
    for i : 1 thru NN do ( Pdx : Pdx + Vx * dwxL [ i ] ) $
    Pdx $

--  eVx : ev ( Vx , virtL ) $ eP : eVx $ R1 : remainder ( eP , fx4 [ 2 ] , δ ) $ R2 : remainder ( R1 , fx3 [
>   2 ] , γ ) $ R3 : remainder ( R2 , fx2 [ 2 ] , β ) $ R23 : remainder ( R3 , fx1 [ 2 ] , α ) $ Vx : R23 ;
    ePax : ev ( Pax , virtL ) $ eP : ePax $ R1 : remainder ( eP , fx4 [ 2 ] , δ ) $ R2 : remainder ( R1 ,
    fx3 [ 2 ] , γ ) $ R3 : remainder ( R2 , fx2 [ 2 ] , β ) $ R23 : remainder ( R3 , fx1 [ 2 ] , α ) $ Pax :
    R23 ;
    ePbx : ev ( Pbx , virtL ) $ eP : ePbx $ R1 : remainder ( eP , fx4 [ 2 ] , δ ) $ R2 : remainder ( R1 ,
    fx3 [ 2 ] , γ ) $ R3 : remainder ( R2 , fx2 [ 2 ] , β ) $ R23 : remainder ( R3 , fx1 [ 2 ] , α ) $ Pbx :
    R23 ;
    ePcx : ev ( Pcx , virtL ) $ eP : ePcx $ R1 : remainder ( eP , fx4 [ 2 ] , δ ) $ R2 : remainder ( R1 ,
    fx3 [ 2 ] , γ ) $ R3 : remainder ( R2 , fx2 [ 2 ] , β ) $ R23 : remainder ( R3 , fx1 [ 2 ] , α ) $ Pcx :
    R23 ;
    ePdx : ev ( Pdx , virtL ) $ eP : ePdx $ R1 : remainder ( eP , fx4 [ 2 ] , δ ) $ R2 : remainder ( R1 ,
    fx3 [ 2 ] , γ ) $ R3 : remainder ( R2 , fx2 [ 2 ] , β ) $ R23 : remainder ( R3 , fx1 [ 2 ] , α ) $ Pdx :
    R23 ;

--> /* Vxは最小多項式gx[0]の候補となる。 Vx が因数分解されたかどうかチェックする。
    因数分解されないとき　gx[0]=Vx　とする
    Vxが因数分解されたとき　　gx[0]=part(Vx,1)　として因数分解の第１因子をgx[0]とする
    以上の判断をして以下の議論をする。
            参考：式" f "の1番目(n=1)の因数を取り出す命令　part(f,n)

    hipow(Vx,x)
    と言うコマンドも面白い

    */

--> Vx : factor ( Vx ) ;

--> Vpw : hipow ( Vx , x ) $
    if Vpw = NN then gx [ 0 ] : Vx else gx [ 0 ] : part ( Vx , 1 ) $
    gx [ 0 ] ;
```

```
--> gv [ 0 ] : subst ( v , x , gx [ 0 ] ) ;

--> /*ここ重要　拘束条件のリスト作成

    Const:[]$  巡回拡大の際に現れる2項方程式B[i]とその根として添加される数a[i]を
                巡回拡大が現れる度に拘束条件リストに付け加えられるリスト
              Const:cons([a[i],B[i]],const); という命令で積み重ねる

    CurConst:[]$　高速条件Consにその時点での最小多項式の条件を加えるた高速条件
                CurConst:cons([v,gv[i]],Const);　*/

    /*現時点での単拡大体F(v)上での拘束条件作成は以下の通り*/ ;

--> Const : [ ] $
    CurConst : [ ] $
    CurConst : cons ( [ v , gv [ 0 ] ] , Const ) ;

--> dV : diff ( Vx , x ) ;

--> dVx : diff ( Vx , x ) ;  dVv : subst ( v , x , dVx ) ; IdV : Inverse ( dVv , CurConst ) ;

--  /*  逆元がvの多項式として求まったので、α,β,γ,δ の計算が出来るようになった  */
>
    Pav : subst ( v , x , Pax ) ; Pbv : subst ( v , x , Pbx ) ; Pcv : subst ( v , x , Pcx ) ; Pdv : subst ( v , x ,
    Pdx ) ;
    /*  wa1:expand(Pav*IdV);  HTML用*/
    av : Reduce ( Pav * IdV , CurConst ) ; bv : Reduce ( Pbv * IdV , CurConst ) $ cv : Reduce ( Pcv *
    IdV , CurConst ) $ dv : Reduce ( Pdv * IdV , CurConst ) $
    SoL : [ α = av , β = bv , γ = cv , δ = dv ] $

--> VivL : expand ( ev ( virtL , SoL ) ) $

--> check : [ ] $ GgrL : [ ] $
    for i : 1 thru NN do (
      z : remainder ( subst ( rhs ( VivL [ i ] ) , x , Vx ) , gv [ 0 ] ) ,
      check : endcons ( z , check ) , if z = 0 then GgrL : endcons ( i , GgrL )
                    ) $
    check ; GgrL ;

--> /*
    これで準備が整った。後は組成列に従って巡回拡大に分解して
    gx[0] - > gx{1} -. gx[2] -> gx[3] -> gx[4]　と
    最小多項式の次数を提言してゆく計算
    */

--> /*  第４ステップ　 P=2　*/
```

```
--> Gal11:[1,4,5,8,9,12,13,16,17,20,21,24]$
    Gal12:[2,3,6,7,10,11,14,15,18,19,22,23]$

--> N1:length(Gal11);

--> vx1L:makelist(x-v[Gal11[i]],i,1,N1);
    vx2L:makelist(x-v[Gal12[i]],i,1,N1);

--> h0:apply("*",vx1L);
    h1:apply("*",vx2L);

--> h0:ev(h0,VivL)$  h0:Reduce(h0,CurConst);
    h1:ev(h1,VivL)$  h1:Reduce(h1,CurConst);

--> t0:ratexpand((h0+h1)/2)$ t0:Reduce(t0,CurConst)$
    t1:ratexpand((h0-h1)/2)$ t1:Reduce(t1,CurConst)$

--> ratvars(x)$ t0;t1:ratsimp(t1);
    /* t1:expand(t1)$ t1:ratsimp(t1); */

--> t1:expand(t1)$
    cv:coeff(t1,x,hipow(t1,x));
    cv:Reduce(cv,CurConst);
    cv2:cv^2$ cv2:Reduce(cv2,CurConst);
    A[1]:cv2$ B[1]:a[1]^2-A[1];

--> icv:Inverse(cv,CurConst);
    q1:Reduce(icv*t1,CurConst)$ q1:expand(q1);

--> t1:a[1]*q1;
    h0:t0+t1$ gx[1]:h0; gv[1]:subst(v,x,h0);

--> /*
    新たな制約条件B[1]と新たな最小多項式gv[1]が生成されたので、
    Const  CurConst   を後進する必要があるので以下の命令をする
    */

    cn:[a[1],B[1]]$ Const:cons(cn,Const);
    Ω:ω^2+ω+1$
    cn:[ω,Ω]$ Const:cons(cn,Const)$
    CurConst:cons([v,gv[1]],Const);

    /*拘束条件作成は終*/


--> /*  第５ステップ      P=3  */

--> /*  Gal11:[1,4,5,8,9,12,13,16,17,20,21,24]$  */
```

```
--> Gal21:[1,8,17,24]$
    Gal22:[4,12,13,21]$
    Gal23:[5,9,16,20]$

--> N2:length(Gal21);

--> vx1L:makelist(x-v[Gal21[i]],i,1,N2);
    vx2L:makelist(x-v[Gal22[i]],i,1,N2);
    vx3L:makelist(x-v[Gal23[i]],i,1,N2);

--> h0:apply("*",vx1L);
    h1:apply("*",vx2L);
    h2:apply("*",vx3L);

--> h0:ev(h0,VivL)$h0:Reduce(h0,CurConst);
    h1:ev(h1,VivL)$h1:Reduce(h1,CurConst);
    h2:ev(h2,VivL)$h2:Reduce(h2,CurConst)$

    t0:(h0+h1+h2)/3$          t0:Reduce(t0,CurConst);
    t1:(h0+ω*h1+ω^2*h2)/3$t1:Reduce(t1,CurConst)$
    t2:(h0+ω^2*h1+ω*h2)/3$  t2:Reduce(t2,CurConst)$

--> /*HTML用出力*/
    t1:expand(t1)$t2:expand(t2)$
    mp:hipow(t1,x);
    a2:coeff(t1,x,mp);
    b2:coeff(t2,x,mp);
    /*出力終了*/

--> t1:expand(t1)$a2:coeff(t1,x,hipow(t1,x))$
    t2:expand(t2)$b2:coeff(t2,x,hipow(t2,x))$

    ia2:Inverse(a2,CurConst)$
    ib2:Inverse(b2,CurConst)$
    q1:Reduce(ia2*t1,CurConst)$q1:expand(q1);
    q2:Reduce(ib2*t2,CurConst)$q2:expand(q2);

--> /*HTML 出力用*/
    ia2:Inverse(a2,CurConst);
    ib2:Inverse(b2,CurConst);

--> d2:a2^3$d2:Reduce(d2,CurConst)$d2:expand(d2);
    A[2]:d2$B[2]:a[2]^3-A[2];

    id2:Inverse(d2,CurConst);
    ev2:a2*b2$ev2:Reduce(ev2,CurConst);
    ev2:Reduce(ev2*id2,CurConst)$ev2:expand(ev2)*a[2]^2;
```

```
--> t1 : a [ 2 ] * q1 ; t2 : ev2 * q2 ;

--> t2 : Reduce ( t2 , CurConst ) $ t2 : expand ( t2 ) ;

--> ratvars ( a [ 2 ] ) $ ratsimp ( t1 ) ; ratsimp ( t2 ) ;
    gx [ 2 ] : t0 + t1 + t2 ; gv [ 2 ] : subst ( v , x , gx [ 2 ] ) ;

--> /*      以上で3乗根多項式が求められた    */

--  hipow ( gx [ 2 ] , x ) ;
>   c4 : coeff ( gx [ 2 ] , x , 4 ) ; c3 : coeff ( gx [ 2 ] , x , 3 ) ; c2 : coeff ( gx [ 2 ] , x , 2 ) ; c1 : coeff ( gx [
    2 ] , x , 1 ) ; c0 : coeff ( gx [ 2 ] , x , 0 ) ;

--> /*   拘束条件の更新    */

    cn : [ a [ 2 ] , B [ 2 ] ] $ Const : cons ( cn , Const ) ;

    CurConst : cons ( [ v , gv [ 2 ] ] , Const ) ;


    /*拘束条件の更新完了*/

--> /*  第6ステップ       P=2 */

--> ratvars ( x ) $

--> Gal31 : [ 1 , 8 ] $
    Gal32 : [ 17 , 24 ] $

--> N3 : length ( Gal31 ) ;

--> vx1L : makelist ( x - v [ Gal31 [ i ] ] , i , 1 , N3 ) ;
    vx2L : makelist ( x - v [ Gal32 [ i ] ] , i , 1 , N3 ) ;

--> h0 : apply ( "*" , vx1L ) ;
    h1 : apply ( "*" , vx2L ) ;

--> h0 : ev ( h0 , VivL ) $ h0 : Reduce ( h0 , CurConst ) ;
    h1 : ev ( h1 , VivL ) $ h1 : Reduce ( h1 , CurConst ) $
    t0 : expand ( ( h0 + h1 ) / 2 ) ; t1 : expand ( ( h0 - h1 ) / 2 ) ;

--> pw : hipow ( t1 , x ) ;
    t1 : expand ( t1 ) $ cv1 : coeff ( t1 , x , hipow ( t1 , x ) ) ; cv0 : coeff ( t1 , x , 0 ) ;

--> t1 : expand ( t1 ) $ cv1 : coeff ( t1 , x , hipow ( t1 , x ) ) $

    dv1 : cv1 ^ 2 $ dv1 : Reduce ( dv1 , CurConst ) $ dv1 : expand ( dv1 ) ;
    A [ 3 ] : dv1 $ B [ 3 ] : a [ 3 ] ^ 2 - A [ 3 ] ;
```

```
    icv1 : Inverse ( cv1 , CurConst ) ;
    q1 : Reduce ( icv1 * t1 , CurConst ) $ q1 : expand ( q1 ) ;
-->  /*HTML出力用*/

    dv1 : expand ( dv1 ) ; ratvars ( a [ 2 ] ) $ dv1 : ratsimp ( dv1 ) ;

    /*HTML　終了*/
-->  t1 : a [ 3 ] * q1 ;
-->  gx [ 3 ] : t0 + t1 ;  gv [ 3 ] : subst ( v , x , gx [ 3 ] ) ;
-->  /*　拘束条件の更新　*/

    cn : [ a [ 3 ] , B [ 3 ] ] $ Const : cons ( cn , Const ) ;

    CurConst : cons ( [ v , gv [ 3 ] ] , Const ) ;


    /*拘束条件の更新完了*/
-->  /* 第7ステップ　*/
-->  Gal41 : [ 1 ] $
     Gal42 : [ 8 ] $
-->  N4 : length ( Gal41 ) ;
-->  vx1L : makelist ( x - v [ Gal41 [ i ] ] , i , 1 , N4 ) ;
     vx2L : makelist ( x - v [ Gal42 [ i ] ] , i , 1 , N4 ) ;
-->  h0 : apply ( "*" , vx1L ) ;
     h1 : apply ( "*" , vx2L ) ;
-->  h0 : ev ( h0 , VivL ) $ h0 : Reduce ( h0 , CurConst ) ;
     h1 : ev ( h1 , VivL ) $ h1 : Reduce ( h1 , CurConst ) ;

    t0 : expand ( ( h0 + h1 ) / 2 ) ; t1 : expand ( ( h0 - h1 ) / 2 ) ;
-->  t1 : expand ( t1 ) $ cv1 : coeff ( t1 , x , hipow ( t1 , x ) ) $

    dv1 : cv1 ^ 2 $ dv1 : Reduce ( dv1 , CurConst ) $ dv1 : expand ( dv1 ) ;
    A [ 4 ] : dv1 $ B [ 4 ] : a [ 4 ] ^ 2 - A [ 4 ] ;

    icv1 : Inverse ( cv1 , CurConst ) $
    q1 : Reduce ( icv1 * t1 , CurConst ) $ q1 : expand ( q1 ) ;
```

```
--> t1:a[4]*q1$
    gx[4]:expand(t0+t1);gv[4]:subst(v,x,gx[4]);
--> /*   拘束条件の更新   */

    cn:[a[4],B[4]]$Const:cons(cn,Const);

    CurConst:cons([v,gv[4]],Const);


    /*拘束条件の更新完了*/
--> W:solve(gx[4],x);
--> B[1];B[2];B[3];B[4];Ω;
--> rsoL:[]$
    for i:1 thru N do(

      s:rhs(SoL[i]),s:Reduce(s,CurConst),
      rsoL:endcons(s,rsoL)
    )$
--> rsoL;
--> rω:allroots(Ω)$fω:rhs(rω[1]);

    for i:1 thru 4 do(B[i]:subst(fω,ω,B[i]))$

    ra1:allroots(B[1])$fa1:rhs(ra1[1]);

    for i:2 thru 4 do(B[i]:subst(fa1,a[1],B[i]))$

    ra2:allroots(B[2])$fa2:rhs(ra2[1]);

    for i:3 thru 4 do(B[i]:subst(fa2,a[2],B[i]))$

    ra3:allroots(B[3])$fa3:rhs(ra3[1]);

    for i:4 thru 4 do(B[i]:subst(fa3,a[3],B[i]))$

    ra4:allroots(B[4])$fa4:rhs(ra4[1]);
--  AnsLf:[]$
>   for i:1 thru 4 do(
      z:rsoL[i],z:subst(fω,ω,z),z:subst(fa1,a[1],z),z:subst(fa2,a[2],z),z:
    subst(fa3,a[3],z),z:subst(fa4,a[4],z),
```

```
      z : expand ( z ) ,
      AnsLf : endcons ( z , AnsLf )
    ) $
    AnsLf ;

--> allroots ( fx ) ;
```
