Source Code for TOOLBOX

Version 5.2

Lundh, Michael

1988

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7385)/1-79/(1988)

Source Code for TOOLBOX – Version 5.2

Michael Lundh

Department of Automatic Control
Lund Institute of Technology
March 1988

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Report

Date of issue
March 1988

Document Number
CODEN: LUTFD2/(TFRT-7385)/1-79/(1988)

Author(s)
Michael Lundh

Supervisor

Sponsoring organisation

Title and subtitle
Source Code for TOOLBOX – Version 5.2

Abstract

This report contains the source code for the program TOOLBOX.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
79

Recipient's notes

Security classification

1. Introduction

This paper contains the source code for the program *TOOLBOX*, version 5.2 linked 88-03-23. It is described in Lundh (1988).

2. Modules

The program consists of the main program module on file TOOLBOX.MOD and 24 pairs of definition and implementation modules.
Files are grouped by giving them names starting with same character combinations. Files with names starting with PRC contains code for the concurrent processes of the program. Combination MNY means code for menus and combination AD means code for the adaptive controller.
The modules are

ADAPTIVE	General interface for adaptive control.
ADISTR	Indirect adaptive controller.
BODEPLOT	Routines for drawing the Bode diagram of the process model.
DESIGN	Pole placement design routine.
GLOBALS	Data structures for global modes and man-machine interface.
MNYDESGN	Procedures called from Design-Menu.
MNYESTIM	Procedures called from Estim-Menu.
MNYMODEL	Procedures called from Model-Menu.
MNYPLOT	Procedures called from Plot-Menu.
MNYREFS	Procedures called from Refsig-Menu.
MNYREGUL	Procedures called from Regul-Menu.
MNYSETUP	Procedures called from Setup-Menu.
OPCOM	Auxiliary routines for man-machine interface.
PCALC	Procedures for polynomial calculations.
PRCEXEC	Process Execute and Update.
PRCKEYB	Process Keyboard.
PRCMOUSE	Process MouseHigh and MouseLow.
PRCPLOT	Process Plotter.
REGULMON	Monitor for regulator.
RLS	Code for the recursive least-squares estimator.
SCREEND	Defines how the screen is split in different areas.
SHOWPAR	Procedures for displaying the history of estimation.
STOREDM	Declaration of array for D-Matrix history.
STORETH	Declaration of array for history of A and B in the model.

The file name extensions are .DEF and .MOD.

Compilation

The software development system used comes from LOGITECH. Most modules have been compiled using the compiler command line

```
m2c/C/2
```

To increase speed of procedures executed every sample some modules have been compiled so that no run time checks are made. The compiler command line then was

```
m2c/C/2/A-/R-/S-/T-
```

The modules compiled for fast execution are

ADAPTIVE
ADISTR
DESIGN
GLOBALS
PCALC
PRCEXEC
PRCPLOT
REGULMON
RLS

3. References

LUNDH, M. (1988): "A TOOLBOX for Discrete Time Design and On-Line Control," Report CODEN: LUTFD2/TFRT-7382, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

LOGITECH INC. (1986): *LOGITECH MODULA 2 – Software Development System*, Logitech Inc., Redwood City, CA.

```
DEFINITION MODULE Adaptive;

EXPORT QUALIFIED
  StartAdaptation, AdaptiveDesign;

PROCEDURE StartAdaptation(VAR ok:BOOLEAN);

PROCEDURE AdaptiveDesign(u,y,r,v:REAL);

END Adaptive.
```

```
IMPLEMENTATION MODULE Adaptive;

FROM Screend  IMPORT ErrorMessage;
FROM Globals  IMPORT GlobalData, AdaptDataType, SetEstAdapt,
                     EstAdaptType, AdaptRegType;
FROM AdISTR   IMPORT StartIndirectSTR, IndirectSTR;
(* FROM Debug    IMPORT DebugMouse; *)

PROCEDURE StartAdaptation(VAR ok:BOOLEAN);
BEGIN
  CASE GlobalData.AdaptData.AdaptReg OF
    indstr:  StartIndirectSTR(ok);
    ELSE ErrorMessage('No valid algorithm');
         ok := FALSE;
  END;

  IF ok THEN    (* limitation of control signal *)
    GlobalData.AdaptData.ulimit := 0.02;
  END;
END StartAdaptation;

PROCEDURE AdaptiveDesign(u,y,r,v:REAL);
BEGIN
(*  DebugMouse(10,'Adaptive',1); *)
  CASE GlobalData.AdaptData.AdaptReg OF
    indstr:  IndirectSTR(u,y);
  ELSE HALT;
  END;

  WITH GlobalData.AdaptData DO (* limitation of control signal *)
    IF ulimit<1.0 THEN
      ulimit := ulimit + 0.02; (* OK ??? *)
    END;
  END;
END AdaptiveDesign;

END Adaptive.
```

```
DEFINITION MODULE AdISTR;

EXPORT QUALIFIED
  StartIndirectSTR, IndirectSTR;


PROCEDURE StartIndirectSTR(VAR ok:BOOLEAN);
PROCEDURE IndirectSTR(u,y:REAL);

END AdISTR.
```

```
IMPLEMENTATION MODULE AdISTR;

FROM Pcalc    IMPORT mdegree, cpoly, Polnorm, Polmul, Diophantine, FixDegree;
FROM Globals  IMPORT ModelType, DesignType, RegulatorType,
                     GlobalData, AdaptDataType, TempData;
FROM RLS      IMPORT RestartEstimation, Estimation;
FROM Screend  IMPORT ErrorMessage;
FROM Regulmon IMPORT NewAdaptiveRegulator,
                     GetPolyInRegulator, NewPolyInRegulator;

CONST
  relative =1.0E-8;

VAR
  AoAm, Integr, Ttmp,
  Aext, AextIntegr  : cpoly;
  amqeq1 : REAL;

PROCEDURE StartIndirectSTR(VAR ok:BOOLEAN);
VAR i:CARDINAL;
BEGIN
  (* ??????????? hur ar det med modellen -- behover tester goras *)
  WITH GlobalData.Design DO
    Polmul(Ao, Am, relative, AoAm);
    IF Polnorm(AoAm)<0.0001 THEN
      ErrorMessage(' Am * Ao = 0 ');
      ok := FALSE;
      RETURN;
    END;

    IF Polnorm(Bmprim)<0.0001 THEN
      ErrorMessage(" Bm' = 0 ");
      ok := FALSE;
      RETURN;
    END;

    IF IntegralAction THEN
      WITH Integr DO
        degree:=1;  coeffs[0]:=1.0;  coeffs[1]:=-1.0;
        FOR i:=2 TO mdegree DO  coeffs[i]:=0.0;  END;
      END;
    ELSE
      WITH Integr DO
        degree:=0;  coeffs[0]:=1.0;
        FOR i:=1 TO mdegree DO  coeffs[i]:=0.0;  END;
      END;
    END;

    WITH Aext DO (* handle addtional poles in the origin *)
      degree:=GlobalData.Model.AdditionalPolesInOrigin;
      coeffs[0]:=1.0;
      FOR i:=1 TO mdegree DO  coeffs[i]:=0.0;  END;
    END;
    Polmul(Integr,Aext,relative,AextIntegr);

    (* degree test  deg(Ao)+deg(Am) >= 2deg(A)-deg(B+)+1-1 *)
    WITH GlobalData.Model DO
      IF AoAm.degree < 2*(A.degree+Aext.degree) -0 +Integr.degree -1 THEN
        ok:=FALSE;
        ErrorMessage(' Degree fault ');
        RETURN;
      END;
    END;

    amqeq1 := 0.0;
    FOR i:=0 TO Am.degree DO (* Am(1) *)
      amqeq1 := amqeq1 + Am.coeffs[i];
    END;
    IF ABS(amqeq1) < 0.001 THEN
      ok:=FALSE;
      ErrorMessage(' Am(1) = 0 ');
      RETURN;
    END;

    Polmul(Ao,Bmprim,relative,Ttmp);

    (* tilldela regulator polynom gradtal *)
    GetPolyInRegulator(TempData.Regulator);
    TempData.Regulator.RegType := 'ISTR';
    TempData.Regulator.T := Ttmp;
    TempData.Regulator.R.degree := AoAm.degree - Aext.degree
                                   - GlobalData.Model.A.degree;
    TempData.Regulator.S1.degree:= TempData.Regulator.R.degree;
    TempData.Regulator.T.degree := TempData.Regulator.R.degree;
    TempData.Regulator.Ao.degree:= TempData.Regulator.R.degree;

    TempData.Regulator.S2.degree:= 0;
    TempData.Regulator.S3.degree:= 0;
    FOR i:=0 TO mdegree DO
      TempData.Regulator.S2.coeffs[i] := 0.0;
      TempData.Regulator.S3.coeffs[i] := 0.0;
    END;

    NewPolyInRegulator(TempData.Regulator);

    ok:=TRUE;
  END;
  RestartEstimation(FALSE,ok);
END StartIndirectSTR;

PROCEDURE IndirectSTR(u,y:REAL);
VAR i:CARDINAL;
    bmqeq1, t0, r0: REAL;
    a1, bm, r1: cpoly;
BEGIN
  Estimation(u,y);

  WITH GlobalData DO
    Polmul(Model.A,AextIntegr,relative,a1);

    (* B- = B    B+ = 1 *)
    Polmul(Model.B,Design.Bmprim,relative,bm);

    bmqeq1 := 0.0;
    FOR i:=0 TO bm.degree DO (* bm(1) *)
      bmqeq1 := bmqeq1 + bm.coeffs[i];
    END;
    t0 := amqeq1/bmqeq1;

    WITH TempData.Regulator DO
      Diophantine(a1,Model.B,AoAm,r1,S1);
      Polmul(r1,Integr,relative,R);
```

```
      FixDegree(S1,R.degree);

      r0:=R.coeffs[0];  (* = 1.0 always ??? *)

      FOR i:=0 TO R.degree DO
        R.coeffs[i]  := R.coeffs[i] / r0;
        S1.coeffs[i] := S1.coeffs[i] / r0;
        T.coeffs[i]  := Ttmp.coeffs[i] / r0 * t0;
      END;
    END;
  END;

  NewAdaptiveRegulator(TempData.Regulator);
END IndirectSTR;

END AdISTR.
```

```
DEFINITION MODULE BodePlot;

FROM Pcalc    IMPORT cpoly;

EXPORT QUALIFIED DBode, DBode2;

PROCEDURE DBode(a,b:cpoly; ExtraPolesInOrigin:CARDINAL; wlow,whigh,h:REAL);

PROCEDURE DBode2(a1,b1:cpoly; ExtraPolesInOrigin1:CARDINAL;
                 a2,b2:cpoly; ExtraPolesInOrigin2:CARDINAL;
                 wlow,whigh,h:REAL);

END BodePlot.
```

```
IMPLEMENTATION MODULE BodePlot;

FROM Pcalc    IMPORT mdegree, coeffvector, cpoly;
FROM Graphics IMPORT SetWindow,rectangle,point, SetTextColor, SetLineColor,
                     PolyLine, color, SetFillColor,
                     FillRectangle, HideCursor, ShowCursor;
FROM Screend  IMPORT ScreenData;
FROM Opcom    IMPORT WriteReal,WriteText;
FROM MathLib  IMPORT exp,ln,cos,sin,sqrt,arctan;

CONST
  pi = 3.14159;
TYPE
  complex = RECORD
              re, im : REAL;
            END;
  plotvec = ARRAY [0..50] OF point;
VAR
  LogMagn1, Phase1,
  LogMagn2, Phase2 : plotvec;
  box              : rectangle;

  logaMax, logaMin,
  phaMax,  phaMin,
  logwMax, logwMin : REAL;

PROCEDURE ComMul(a,b:complex; VAR ab:complex);
BEGIN
  ab.re := a.re*b.re - a.im*b.im;
  ab.im := a.re*b.im + a.im*b.re;
END ComMul;

PROCEDURE ComAbs(a:complex):REAL;
BEGIN
  RETURN  sqrt(a.re*a.re + a.im*a.im);
END ComAbs;

PROCEDURE ComDiv(a,b:complex; VAR ab:complex);
BEGIN
  ab.re := (a.re*b.re + a.im*b.im)/(b.re*b.re + b.im*b.im);
  ab.im := (a.im*b.re - a.re*b.im)/(b.re*b.re + b.im*b.im);
END ComDiv;

PROCEDURE atan2(x,y:REAL):REAL;
BEGIN
  IF x>0.0 THEN
    RETURN arctan(y/x);
  ELSIF x<0.0 THEN
    IF y>0.0 THEN
      RETURN arctan(y/x) + pi;
    ELSE
      RETURN arctan(y/x) - pi;
    END;
  ELSE
    IF y>0.0 THEN
      RETURN  pi/2.0;
    ELSE
      RETURN -pi/2.0;
    END;
  END;
END atan2;

PROCEDURE Floor(x:REAL):REAL;
(* round towards -infinity *)
BEGIN
  IF x>=0.0 THEN
    RETURN  FLOAT( TRUNC( x ) );
  ELSE
    RETURN -FLOAT( TRUNC( 1.0-x) );
  END;
END Floor;

PROCEDURE Ceil(x:REAL):REAL;
(* round towards +infinity *)
BEGIN
  IF x>0.0 THEN
    RETURN  FLOAT( TRUNC( x+0.999 ) );
  ELSE
    RETURN -FLOAT( TRUNC( -x ) );
  END;
END Ceil;

PROCEDURE FixPhase(VAR Phase:plotvec);
VAR i,iw:CARDINAL;
BEGIN
  FOR iw:=1 TO 50 DO
    IF (Phase[iw].v-Phase[iw-1].v)>100.0 THEN
      FOR i:=iw TO 50 DO
        Phase[i].v:=Phase[i].v - 360.0;
      END;
    END;
  END;
END FixPhase;

PROCEDURE GetLimits(NrOfPlots:CARDINAL);
VAR iw:CARDINAL;
BEGIN
(*  DebugMouse(11,'GetLimit',1); *)
  logwMin := Floor( Phase1[0].h  );
  logwMax := Ceil(  Phase1[50].h );
  IF (logwMax-logwMin)<2.0 THEN
    logwMax:=logwMin+2.0;
  END;

  logaMax := -1.0E-10;  phaMax := -1.0E+10;
  logaMin :=  1.0E+10;  phaMin :=  1.0E+10;

  FOR iw:=0 TO 50 DO
    IF LogMagn1[iw].v>logaMax THEN logaMax:=LogMagn1[iw].v; END;
    IF LogMagn1[iw].v<logaMin THEN logaMin:=LogMagn1[iw].v; END;

    IF Phase1[iw].v>phaMax THEN phaMax:=Phase1[iw].v; END;
    IF Phase1[iw].v<phaMin THEN phaMin:=Phase1[iw].v; END;
  END;

  IF NrOfPlots=2 THEN
    FOR iw:=0 TO 50 DO
      IF LogMagn2[iw].v>logaMax THEN logaMax:=LogMagn2[iw].v; END;
      IF LogMagn2[iw].v<logaMin THEN logaMin:=LogMagn2[iw].v; END;

      (* IF Phase2[iw].v>phaMax THEN phaMax:=Phase2[iw].v; END;
         IF Phase2[iw].v<phaMin THEN phaMin:=Phase2[iw].v; END; *)
```

```
    END;
  END;

  logaMin := Floor( logaMin );
  logaMax := Ceil(  logaMax );
  IF (logaMax-logaMin)<1.0 THEN
    logaMax:=logaMin+1.0;
  END;

  phaMin := 90.0*Floor( phaMin/90.0 );
  phaMax := 90.0*Ceil(  phaMax/90.0 );
  IF (phaMax-phaMin)<90.0 THEN
    phaMax:=phaMin+90.0;
  END;
END GetLimits;

PROCEDURE FRD(a,b:cpoly; ExtraPolesInOrigin:CARDINAL; wlow,whigh,h:REAL;
              VAR LogMagnitude, Phase:plotvec);
VAR i,iw      : CARDINAL;
    az,bz,
    bzaz,eiwh : complex;
    w         : REAL;
BEGIN
  w := wlow;
(*  DebugMouse(12,'FRD',1); *)
  FOR iw:=0 TO 50 DO
    eiwh.re:=cos(w*h);
    eiwh.im:=sin(w*h);

    bz.re:=b.coeffs[0];
    bz.im:=0.0;
    FOR i:=1 TO b.degree DO
      ComMul(bz,eiwh, bz);
      bz.re:=bz.re+b.coeffs[i];
    END;

    az.re:=a.coeffs[0];
    az.im:=0.0;
    FOR i:=1 TO a.degree DO
      ComMul(az,eiwh, az);
      az.re:=az.re+a.coeffs[i];
    END;
    FOR i:=1 TO ExtraPolesInOrigin DO
      ComMul(az,eiwh, az);
    END;

    ComDiv(bz,az, bzaz);
    LogMagnitude[iw].h := ln(w)/ln(10.0);
    LogMagnitude[iw].v := ln( ComAbs(bzaz) )/ln(10.0);

    Phase[iw].h := ln(w)/ln(10.0);
    Phase[iw].v := atan2(bzaz.re,bzaz.im)*180.0/pi;

    (* w:=w*(whigh/wlow)^(1.0/50.0); *)
    w:= w* exp( ln(whigh/wlow) / 50.0)
  END;

  FixPhase(Phase);
END FRD;

PROCEDURE PlotDottedLine(y:REAL);
VAR i:CARDINAL;
    p:ARRAY[0..1] OF point;
BEGIN
  SetLineColor(ScreenData.Opcomhandle,green);
  p[0].v:=y;  p[1].v:=y;
  FOR i:=0 TO 30 DO
    p[0].h:=logwMin+(logwMax-logwMin)*FLOAT(i)/31.0;
    p[1].h:=p[0].h+(logwMax-logwMin)/90.0;
    PolyLine(ScreenData.Opcomhandle, p, 2);
  END;
END PlotDottedLine;

PROCEDURE AmplitudePlot(NrOfPlots:CARDINAL);
BEGIN
(*  DebugMouse(13,'AmPlot',1); *)
  box.loleft.h:=logwMin;                    box.upright.h:=logwMax;
  box.loleft.v:=logaMin-(logaMax-logaMin); box.upright.v:=logaMax;
  SetWindow(ScreenData.Opcomhandle,box);
  SetFillColor(ScreenData.Opcomhandle,black);
  FillRectangle(ScreenData.Opcomhandle,box);
  SetTextColor(ScreenData.Opcomhandle,green);
  WriteText(logwMin+0.75*(logwMax-logwMin),
            logaMin+0.90*(logaMax-logaMin),'BODE PLOT');

  SetTextColor(ScreenData.Opcomhandle,red);

  WriteReal(logwMin+0.02*(logwMax-logwMin),
            logaMin+0.90*(logaMax-logaMin),'',exp(logaMax*ln(10.0)) );
  WriteReal(logwMin+0.02*(logwMax-logwMin),
            logaMin+0.03*(logaMax-logaMin),'',exp(logaMin*ln(10.0)) );
  WriteText(logwMin+0.15*(logwMax-logwMin),
            logaMin+0.90*(logaMax-logaMin),'Magnitude (B/A)');

  SetLineColor(ScreenData.Opcomhandle,red);
  PolyLine(ScreenData.Opcomhandle, LogMagn1, 50);
  IF NrOfPlots=2 THEN
    SetTextColor(ScreenData.Opcomhandle,blue);
    WriteText(logwMin+0.35*(logwMax-logwMin),
              logaMin+0.90*(logaMax-logaMin),' (Bf/Af)');

    SetLineColor(ScreenData.Opcomhandle,blue);
    PolyLine(ScreenData.Opcomhandle, LogMagn2, 50);
  END;

  IF (logaMin<0.0) AND (0.0<logaMax) THEN (* gain=1.0 *)
    PlotDottedLine(0.0);
  END;
END AmplitudePlot;

PROCEDURE PhasePlot(NrOfPlots:CARDINAL);
BEGIN
(*  DebugMouse(14,'PhPlot',1); *)
  box.loleft.h:=logwMin;  box.upright.h:=logwMax;
  box.loleft.v:=phaMin;   box.upright.v:=phaMax+(phaMax-phaMin);
  SetWindow(ScreenData.Opcomhandle,box);
  SetTextColor(ScreenData.Opcomhandle,red);

  WriteReal(logwMin+0.02*(logwMax-logwMin),
            phaMin+0.90*(phaMax-phaMin),'',phaMax);
  WriteReal(logwMin+0.02*(logwMax-logwMin),
            phaMin+0.03*(phaMax-phaMin),'',phaMin);
```

```
  WriteText(logwMin+0.15*(logwMax-logwMin),
            phaMin+0.90*(phaMax-phaMin),'Phase (degrees)');

  SetLineColor(ScreenData.Opcomhandle,red);
  PolyLine(ScreenData.Opcomhandle, Phase1, 50);
(*  IF NrOfPlots=2 THEN
    SetLineColor(ScreenData.Opcomhandle,blue);
    PolyLine(ScreenData.Opcomhandle, Phase2, 50);
  END; *)

  IF (phaMin<-180.0) AND (-180.0<phaMax) THEN (* phase=-180.0 *)
    PlotDottedLine(-180.0);
  END;
END PhasePlot;

PROCEDURE FrequencyAxis;
VAR ap:ARRAY [0..1] OF point;
    w :REAL;
BEGIN
(*  DebugMouse(14,'FrAxis',1); *)
  ap[0].h:=logwMin; ap[1].h:=logwMax;
  ap[0].v:=phaMax;  ap[1].v:=phaMax;
  SetLineColor(ScreenData.Opcomhandle,green);
  SetTextColor(ScreenData.Opcomhandle,green);
  PolyLine(ScreenData.Opcomhandle, ap, 2);

  ap[0].v:=phaMax-0.02*(phaMax-phaMin);
  ap[1].v:=phaMax+0.02*(phaMax-phaMin);
  w:=logwMin+1.0;
  WHILE w<(logwMax-0.5) DO
    ap[0].h:=w;    ap[1].h:=w;
    PolyLine(ScreenData.Opcomhandle, ap, 2);
    WriteReal(w-0.05*(logwMax-logwMin),
              phaMax+0.02*(phaMax-phaMin),'',exp(w*ln(10.0)) );
    w:=w+1.0;
  END;
  WriteText(logwMax-0.1*(logwMax-logwMin),
            phaMax+0.02*(phaMax-phaMin),'rad/s');
END FrequencyAxis;

PROCEDURE DBode(a,b:cpoly; ExtraPolesInOrigin:CARDINAL; wlow,whigh,h:REAL);
BEGIN
  FRD(a,b, ExtraPolesInOrigin, wlow,whigh,h, LogMagn1,Phase1);
  GetLimits(1);

  HideCursor;
  AmplitudePlot(1);
  PhasePlot(1);
  FrequencyAxis;
  ShowCursor;
END DBode;

PROCEDURE DBode2(a1,b1:cpoly; ExtraPolesInOrigin1:CARDINAL;
                 a2,b2:cpoly; ExtraPolesInOrigin2:CARDINAL;
                 wlow,whigh,h:REAL);
BEGIN
  FRD(a1,b1, ExtraPolesInOrigin1, wlow,whigh,h, LogMagn1,Phase1);
  FRD(a2,b2, ExtraPolesInOrigin2, wlow,whigh,h, LogMagn2,Phase2);
  GetLimits(2);

  HideCursor;
  AmplitudePlot(2);
  PhasePlot(2);
  FrequencyAxis;
  ShowCursor;
END DBode2;

END BodePlot.
```

```
DEFINITION MODULE Design;

FROM Pcalc   IMPORT cpoly;

EXPORT QUALIFIED RSTD;

PROCEDURE RSTD(a,bplus,bminus,am,bm1,ao,ar:cpoly; VAR r,s,t:cpoly);

(* PROCEDURE LQ(a,bplus,bminus,am,bm1,ao,ar:cpoly; VAR r,s,t:cpoly); *)

END Design.
```

```
IMPLEMENTATION MODULE Design;

FROM Screend  IMPORT ErrorMessage;
FROM Pcalc    IMPORT mdegree, polytypes, complex, coeffvector, cpoly, zpoly,
                     poly, Polnorm, Normalize, Cutpoly,
                     Poladd, Poldiv, Polmul, Polsub,
                     Gcd, Diophantine, FixDegree;

CONST
  relative =1.0E-8;

(*---------------------------------------------------------------------------*)

PROCEDURE PolyEval(p:cpoly; arg:REAL):REAL;
(* Evaluates a polynomial with the argument arg. *)
VAR
  sum : REAL;
  i   : CARDINAL;
BEGIN
  sum := p.coeffs[0];
  FOR i := 1 TO p.degree DO
    sum := sum*arg + p.coeffs[i];
  END;
  RETURN sum;
END PolyEval;

PROCEDURE RSTD(a,bplus,bminus,am,bm1,ao,ar:cpoly; VAR r,s,t:cpoly);
VAR a1,bm,r1,c, tmp,alfa :cpoly;
    r0,t0  : REAL;
    i,deg  : CARDINAL;
BEGIN
  Polmul(a,ar,relative,a1);
  Polmul(am,ao,relative,c);
  Polmul(bminus,bm1,relative,bm);

  t0 := PolyEval(am,1.0)/PolyEval(bm,1.0);

  Diophantine(a1,bminus,c,r1,s);

  IF (r1.degree+ar.degree+bplus.degree)>s.degree THEN (* fix degrees *)
    IF ABS(bminus.coeffs[bminus.degree]) > 1.0E-4 THEN
      alfa.degree    := 0;
      alfa.coeffs[0] := -r1.coeffs[r1.degree]/bminus.coeffs[bminus.degree];
      FOR i:=1 TO mdegree DO  alfa.coeffs[i] := 0.0; END;

      Polmul(alfa,bminus,relative,tmp);
      Poladd(r1,tmp,relative,r1);

      Polmul(alfa,a1,relative,tmp);
      Polsub(s,tmp,relative,s);
    ELSE
      ErrorMessage("can't fix degrees");
    END;
  END;

  Polmul(r1,ar,relative,r);
  Polmul(r,bplus,relative,r);

  r0:=r.coeffs[0];
  Polmul(ao,bm1,relative,t);

  FOR i:=0 TO r.degree DO
    r.coeffs[i] := r.coeffs[i] / r0;
  END;
  FOR i:=0 TO s.degree DO
    s.coeffs[i] := s.coeffs[i] / r0;
  END;
  FOR i:=0 TO t.degree DO
    t.coeffs[i] := t.coeffs[i] / r0 * t0;
  END;

  (* CHECK ORDER OF R S AND T *)
  deg:=0;
  IF t.degree>deg THEN deg:=t.degree; END;
  IF s.degree>deg THEN deg:=s.degree; END;
  IF r.degree<deg THEN
    ErrorMessage('Noncausal Regulator');
  ELSE
    deg:=r.degree;
    FixDegree(r, deg);
    FixDegree(s, deg);
    FixDegree(t, deg);
  END;
END RSTD;

END Design.
```

```
DEFINITION MODULE Globals;

FROM Pcalc    IMPORT (* mdegree, coeffvector, *) cpoly;
(* FROM Graphics IMPORT buttonset; *)

EXPORT QUALIFIED
  free, busy,

  ModeType, EstAdaptType, PlotTypeType, SignalTypeType, OpcomModeType,
  PlottingType, AdaptRegType,

  SetupDataType, ModelType, DesignType, RegulatorType, RefSignalType,
  AdaptDataType, EstimDataType, ActualDataType,

  GlobalData, TempData,

  GetModes, SetMode, SetEstAdapt, SetPlotting,
  SetOpcomMode, GetOpcomMode,
  SendActualData, ReceiveActualData, ResetPlotBuffer, InitGlobals;

CONST
  free = 1;
  busy = 0;

TYPE
  ModeType       = (Stop,Zero,Run);
  PlottingType   = (PlotOff,PlotOn);
  EstAdaptType   = (Off,Estimate,Adaptive);
  AdaptRegType   = (fixed,indstr,indstr2,dirstr);
  PlotTypeType   = (Redraw, PlotBetween,
                    PlotEvery, PlotEveryTwo, PlotEveryFour);
  SignalTypeType = (external, square, triangle, sine, step, ramp);
  OpcomModeType  = (SetupMode, ModelMode,  DesignMode,
                    RegulMode, RefSigMode, PlotMode,
                    EstimMode, AdaptMode,  AnalysMode);

  SetupDataType  = RECORD
                     chanRef,
                     chanY1,
                     chanY2,
                     chanY3,
                     chanU          : CARDINAL;  (* IO channels *)
                     NumberOfInputs : CARDINAL;
                     PlotWhen       : PlotTypeType;
                     HorizontalTime : REAL;
                     Tsamp          : CARDINAL;
                     RealTsamp      : REAL;
                     Dt1, Dt2, Dt3  : CARDINAL;  (* For intersample plotting *)
                     Ulow, Uhigh    : REAL;
                   END;

  ModelType      = RECORD
                     B,A,C  : cpoly;
                     AdditionalPolesInOrigin : CARDINAL;
                     delay : CARDINAL; (* redundant information *)
                     BodeWlow, BodeWhigh:REAL;
                   END;

  DesignType     = RECORD
                     Bplus,Bminus,
                     Bmprim,Am,Ao   : cpoly;
                     IntegralAction : BOOLEAN;
                   END;

  RegulatorType  = RECORD
                     RegType         : ARRAY[0..12] OF CHAR;
                     R,S1,S2,S3,T,Ao : cpoly;
                   END;

  RefSignalType  = RECORD
                     SignalType   : SignalTypeType;
                     Mean,
                     Amplitude,
                     Period       : REAL;
                     TimeInPeriod : REAL;
                   END;

  EstimDataType  = RECORD
                     Bf, Af       : cpoly;
                     D0           : REAL;
                     Lambda       : REAL;
                     HistoryTime  : REAL;
                     HistoryCard  : CARDINAL;
                     EstPurpose   : ARRAY[0..8] OF CHAR;
                                    (* MODELPOLY or REGULPOLY *)
                   END;

  AdaptDataType  = RECORD
                     AdaptReg     : AdaptRegType;
                   (*   RegulatorNr : CARDINAL; *)
                     ulimit       : REAL;
                   END;

  ActualDataType = RECORD
                     PlotWhen                    : PlotTypeType;
                     y1t0,y2t0,y3t0, ut0,reft0,
                     y1t1,y2t1,y3t1,
                     y1t2,y2t2,y3t2,
                     y1t3,y2t3,y3t3              : REAL;
                   END;

VAR
  GlobalData     : RECORD
                     SetupData       : SetupDataType;
                     Model           : ModelType;
                     Design          : DesignType;
                     (* Regulator    : RegulatorType; *)
                     RefSignal       : RefSignalType;
                     PlotWhichCurve  : BITSET;
                     EstimData       : EstimDataType;
                     AdaptData       : AdaptDataType;
                   END;

  TempData       : RECORD
                     SetupData       : SetupDataType;
                     Model           : ModelType;
                     Design          : DesignType;
                     Regulator       : RegulatorType;
                     RefSignal       : RefSignalType;
                     (* PlotWhichCurve : BITSET; *)
                     AdaptData       : AdaptDataType;
                     EstimData       : EstimDataType;
```

```
                END;

(* monitor *) PROCEDURE GetModes(VAR m:ModeType; VAR pm:PlottingType;
                                 VAR ea:EstAdaptType);
(* monitor *) PROCEDURE SetMode(m:ModeType);
(* monitor *) PROCEDURE SetPlotting(pm:PlottingType);
(* monitor *) PROCEDURE SetEstAdapt(ea:EstAdaptType);

(* monitor *) PROCEDURE SetOpcomMode(opmode:OpcomModeType);
(* monitor *) PROCEDURE GetOpcomMode():OpcomModeType;

PROCEDURE SendActualData(VAR ad:ActualDataType);
PROCEDURE ReceiveActualData(VAR ad:ActualDataType);
PROCEDURE ResetPlotBuffer;

(* initial *) PROCEDURE InitGlobals;

END Globals.
```

```
IMPLEMENTATION MODULE Globals;

FROM Kernel   IMPORT Event, InitEvent, Await, Cause,
                     Semaphore, InitSem, Wait, Signal;
FROM Pcalc    IMPORT mdegree, coeffvector, cpoly;
FROM Screend  IMPORT WriteMouseBox, ScreenData, ErrorMessage;
FROM Graphics IMPORT buttontype, buttonset, EraseChar,
                     point, WriteString, color, SetFillColor, handle,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;
FROM RLS      IMPORT RestartEstimation;
FROM Adaptive IMPORT StartAdaptation;
FROM Strings  IMPORT Concat;
CONST
  BufferSize= 400;
  ActiveCol     = red;
  MenyLineCol   = blue;
  MenyChoiceCol = blue;
  SZRCol        = blue;
  EstAdCol      = blue;
  PlotCol       = blue;

VAR
  PlotBuffer : RECORD
                 Mutex    : Semaphore;
                 Change   : Event;
                 Buffer   : ARRAY[1..BufferSize] OF ActualDataType;
                 Count    : [0..BufferSize];
                 inp,outp : [1..BufferSize];
               END;

  RedrawPlot  : ActualDataType;

  ModeMonitor : RECORD
                  Mutex     : Semaphore;
                  Mode      : ModeType;
                  Plotting  : PlottingType;
                  EstAdapt  : EstAdaptType;
                  OpcomMode : OpcomModeType;
                END;

(* ------------------------------------------------------------------------ *)
(* monitor *) PROCEDURE SetOpcomMode(opmode:OpcomModeType);
VAR i:CARDINAL;
    pkt:point;
BEGIN
  WITH ModeMonitor DO
    Wait(Mutex);
    HideCursor;
    IF (OpcomMode=PlotMode) OR (opmode=PlotMode) THEN
      OpcomMode:=opmode;
      ResetPlotBuffer;
      SendActualData(RedrawPlot);  (* fel Logik ??*)
    ELSE
      OpcomMode:=opmode;
    END;

    WITH ScreenData DO (* faster if case-statement, requires init *)
      WriteMouseBox(2,0,' SETUP  ',MenyChoiceCol);
      WriteMouseBox(2,1,' MODEL  ',MenyChoiceCol);
      WriteMouseBox(2,2,' DESIGN ',MenyChoiceCol);
      WriteMouseBox(2,3,' REGUL  ',MenyChoiceCol);
      WriteMouseBox(2,4,' REFSIG ',MenyChoiceCol);
      WriteMouseBox(2,5,' PLOT   ',MenyChoiceCol);
      WriteMouseBox(2,6,' ESTIM  ',MenyChoiceCol);
(*      WriteMouseBox(2,7,' ADAPT  ',MenyChoiceCol);
      WriteMouseBox(2,8,' ANALYS ',MenyChoiceCol);  *)
      WriteMouseBox(2,9,' EXIT ',  MenyChoiceCol);

      CASE OpcomMode OF
        SetupMode  : WriteMouseBox(2,0,' SETUP  ',ActiveCol)|
        ModelMode  : WriteMouseBox(2,1,' MODEL  ',ActiveCol)|
        DesignMode : WriteMouseBox(2,2,' DESIGN ',ActiveCol)|
        RegulMode  : WriteMouseBox(2,3,' REGUL  ',ActiveCol)|
        RefSigMode : WriteMouseBox(2,4,' REFSIG ',ActiveCol)|
        PlotMode   : WriteMouseBox(2,5,' PLOT   ',ActiveCol)|
        EstimMode  : WriteMouseBox(2,6,' ESTIM  ',ActiveCol);
(*      AdaptMode  : WriteMouseBox(2,7,' ADAPT  ',ActiveCol)|
        AnalysMode : WriteMouseBox(2,8,' ANALYS ',ActiveCol); *)
      END;
    END;
    ShowCursor;
    Signal(Mutex);
  END;
END SetOpcomMode;

(* monitor *) PROCEDURE GetOpcomMode():OpcomModeType;
VAR op:OpcomModeType;
BEGIN
  RETURN ModeMonitor.OpcomMode;
(*  WITH ModeMonitor DO
    Wait(Mutex);
    op:=OpcomMode;
    Signal(Mutex);
  END;
  RETURN op; *)
END GetOpcomMode;

(* ------------------------------------------------------------------------ *)

(* monitor *) PROCEDURE GetModes(VAR m:ModeType; VAR pm:PlottingType;
                                 VAR ea:EstAdaptType);
BEGIN
  WITH ModeMonitor  DO
    (* Wait(Mutex); *)
    m  := Mode;
    pm := Plotting;
    ea := EstAdapt;
    (* Signal(Mutex); *)
  END;
END GetModes;

(* monitor *) PROCEDURE SetPlotting(pm:PlottingType);
BEGIN
  WITH ModeMonitor  DO
    Wait(Mutex);
    Plotting := pm;

    CASE Plotting OF
      PlotOff: WriteMouseBox(3,5,' PLOTON ',PlotCol);
               WriteMouseBox(3,6,' NOPLOT ',ActiveCol)|
      PlotOn : WriteMouseBox(3,5,' PLOTON ',ActiveCol);
               WriteMouseBox(3,6,' NOPLOT ',PlotCol);
```

```
    END;

    Signal(Mutex);
  END;
END SetPlotting;

(* monitor *) PROCEDURE SetEstAdapt(ea:EstAdaptType);
VAR ok:BOOLEAN;
BEGIN
  WITH ModeMonitor  DO
    Wait(Mutex);
    EstAdapt := ea;
    WriteMouseBox(3,3,' ESTIM  ',EstAdCol);
    WriteMouseBox(3,4,' ADAPT  ',EstAdCol);
    CASE EstAdapt OF
      Off:       (* no action *) |
      Estimate: RestartEstimation(FALSE,ok);
                IF ok THEN
                  WriteMouseBox(3,3,' ESTIM  ',ActiveCol);
                ELSE
                  EstAdapt := Off;
                END|
      Adaptive: StartAdaptation(ok);
                IF ok THEN
                  WriteMouseBox(3,4,' ADAPT  ',ActiveCol);
                ELSE
                  EstAdapt := Off;
                END;
    END;

    Signal(Mutex);
  END;
END SetEstAdapt;

(* monitor *) PROCEDURE SetMode(m:ModeType);
BEGIN
  WITH ModeMonitor  DO
    Wait(Mutex);
    Mode :=m;

    IF Mode=Run THEN
      SendActualData(RedrawPlot);      (* Redraw PlotArea *)
      (* InitAnalyzeBuffer (not yet implemented) *)
      IF EstAdapt<>Adaptive THEN (* no more limitation of controlsignal *)
        GlobalData.AdaptData.ulimit :=1.0;
      END;
    END;

    WriteMouseBox(3,0,' STOP   ',SZRCol);
    WriteMouseBox(3,1,' ZERO   ',SZRCol);
    WriteMouseBox(3,2,' RUN    ',SZRCol);
    CASE ModeMonitor.Mode OF
      Stop: EstAdapt := Off;
            WriteMouseBox(3,3,' ESTIM  ',EstAdCol);
            WriteMouseBox(3,4,' ADAPT  ',EstAdCol);
            WriteMouseBox(3,0,' STOP   ',ActiveCol)|
      Zero: WriteMouseBox(3,1,' ZERO   ',ActiveCol)|
      Run:  WriteMouseBox(3,2,' RUN    ',ActiveCol);
    END;

    Signal(Mutex);
  END;
END SetMode;

(* ------------------------------------------------------------------ *)
(* procedures for sending data to plotter *)

PROCEDURE SendActualData(VAR ad:ActualDataType);
BEGIN          (* VAR-decl to increase speed *)
  WITH PlotBuffer DO
    Wait(Mutex);
    IF Count=BufferSize THEN  (* stop when buffer full *)
      SetMode(Stop);
      ErrorMessage(' Plot Buffer Full ');
      Cause(Change);
    ELSE
      Buffer[inp]:=ad;
      inp:=(inp MOD BufferSize) +1;
      Count:=Count+1;
      Cause(Change);
    END;
    Signal(Mutex);
  END;
END SendActualData;

PROCEDURE ReceiveActualData(VAR ad:ActualDataType);
BEGIN
  WITH PlotBuffer DO
    Wait(Mutex);
    WHILE Count=0 DO  Await(Change);  END;
    ad:=Buffer[outp];
    outp:=(outp MOD BufferSize) +1;
    Count:=Count-1;
    Cause(Change);
    Signal(Mutex);
  END;
END ReceiveActualData;

PROCEDURE InitPlotBuffer;
BEGIN
  WITH PlotBuffer DO
    InitSem(Mutex,free);
    InitEvent(Change,Mutex);
    Count:=0;
    inp:=1;
    outp:=1;
  END;
END InitPlotBuffer;

PROCEDURE ResetPlotBuffer;
BEGIN
  WITH PlotBuffer DO
    Wait(Mutex);
    Count:=0;
    inp:=1;
    outp:=1;
    Signal(Mutex);
  END;
END ResetPlotBuffer;

(* ------------------------------------------------------------------ *)
```

```
(* initial *) PROCEDURE InitGlobals;
VAR i:CARDINAL;
BEGIN
  WITH ModeMonitor DO
    InitSem(Mutex,free);
    Mode     := Stop;
    EstAdapt := Off;
  END;

  WITH RedrawPlot DO
    PlotWhen:=Redraw;
    y1t0 := 0.0; y2t0 := 0.0; y3t0 := 0.0;  ut0 := 0.0;  reft0 := 0.0;
    y1t1 := 0.0; y2t1 := 0.0; y3t1 := 0.0;
    y1t2 := 0.0; y2t2 := 0.0; y3t2 := 0.0;
    y1t3 := 0.0; y2t3 := 0.0; y3t3 := 0.0;
  END;

  InitPlotBuffer;
  SetOpcomMode(SetupMode);
  SetPlotting(PlotOn);

  (*------------------------------------ initialize GlobalData *)
  WITH GlobalData.SetupData DO
    chanRef:=0;  chanY1 :=1;  chanY2 :=2;  chanY3 :=3;  chanU  :=0;
    Tsamp := 200; RealTsamp := 0.2; Dt1:=50; Dt2:=100; Dt3:=150;
    HorizontalTime := 20.0;  NumberOfInputs :=1;
    PlotWhen := PlotEvery;
    Ulow:=-1.0; Uhigh:=1.0;
  END;

  WITH GlobalData.Model DO
    FOR i:=0 TO mdegree DO
      A.coeffs[i]  := 0.0;
      B.coeffs[i]  := 0.0;
      C.coeffs[i]  := 0.0;
    END;
    A.coeffs[0]  := 1.0;
    C.coeffs[0]  := 1.0;
    A.degree:=0;
    B.degree:=0;
    C.degree:=0;
    AdditionalPolesInOrigin := 0;
    delay := 1;
    BodeWlow  := 0.01;
    BodeWhigh := 10.0;
  END;

  WITH GlobalData.Design DO
    Bplus.coeffs[0]  := 1.0;
    Bminus.coeffs[0] := 1.0;
    Bmprim.coeffs[0] := 1.0;
    Am.coeffs[0] := 1.0;
    Ao.coeffs[0] := 1.0;
    FOR i:=1 TO mdegree DO
      Bplus.coeffs[i]  := 0.0;
      Bminus.coeffs[i] := 0.0;
      Bmprim.coeffs[i] := 0.0;
      Am.coeffs[i] := 0.0;
      Ao.coeffs[i] := 0.0;
    END;
    IntegralAction:=FALSE;
    Bplus.degree  := 0;
    Bminus.degree := 0;
    Bmprim.degree := 0;
    Am.degree     := 0;
    Ao.degree     := 0;
  END;

  WITH GlobalData.RefSignal DO
    SignalType := square;  Mean := 0.0;  Amplitude := 0.1;  Period := 10.0;
    TimeInPeriod := 0.0;
  END;

  GlobalData.PlotWhichCurve := {1};

  WITH GlobalData.EstimData DO
    Af.degree:=2;                  (* Filter Parameters *)
    Af.coeffs[0] :=  1.0;
    Af.coeffs[1] := -1.1707;
    Af.coeffs[2] :=  0.6400;
    Bf.degree:=2;
    Bf.coeffs[0] :=  0.0;
    Bf.coeffs[1] :=  1.0;
    Bf.coeffs[2] := -1.0;
    FOR i:=3 TO mdegree DO Af.coeffs[i]:=0.0;  Bf.coeffs[i]:=0.0; END;

    Lambda := 0.99;
    D0     := 100.0;
    HistoryTime := 50.0;
    HistoryCard := 50;
  END;

  WITH GlobalData.AdaptData DO
    AdaptReg    := fixed;
    ulimit      := 1.0;
  END;

  (*------------------------------------ initialize TempData *)
  TempData.SetupData  := GlobalData.SetupData;
  TempData.Model      := GlobalData.Model;
  TempData.Design     := GlobalData.Design;
  WITH TempData.Regulator DO
    RegType:='None';
    FOR i:=0 TO mdegree DO
      R.coeffs[i]  := 0.0;
      S1.coeffs[i] := 0.0;
      S2.coeffs[i] := 0.0;
      S3.coeffs[i] := 0.0;
      T.coeffs[i]  := 0.0;
      Ao.coeffs[i] := 0.0;
    END;
    R.coeffs[0]  := 1.0;
    Ao.coeffs[0] := 1.0;
    R.degree  := 0;
    S1.degree := 0;
    S2.degree := 0;
    S3.degree := 0;
    T.degree  := 0;
    Ao.degree := 0;
  END;

  TempData.RefSignal  := GlobalData.RefSignal;
```

```
  TempData.EstimData  := GlobalData.EstimData;
  TempData.AdaptData  := GlobalData.AdaptData;
END InitGlobals;

END Globals.
```

```
DEFINITION MODULE MNYdesgn;

EXPORT QUALIFIED StartDesignMeny, MouseOnDesignMeny, KeybOnDesignMeny;

PROCEDURE StartDesignMeny;
PROCEDURE MouseOnDesignMeny(rect:CARDINAL);
PROCEDURE KeybOnDesignMeny(str:ARRAY OF CHAR);

END MNYdesgn.
```

```
IMPLEMENTATION MODULE MNYdesgn;

FROM Globals  IMPORT SetMode, SetEstAdapt, ModeType, GlobalData, TempData,
                     EstAdaptType, GetModes, PlottingType,
                     RegulatorType, AdaptRegType;
FROM Pcalc    IMPORT mdegree, coeffvector, cpoly, Polnorm,
                     Polmul, Poldiv, Gcd, Sfactorize;
FROM Graphics IMPORT SetWindow,rectangle,point,EraseChar, buttonset,
                     WriteString,color,SetFillColor,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;
FROM Screend  IMPORT ScreenData, ErrorMessage, WriteMenyLine;
FROM Opcom    IMPORT GetWord, WriteInt, WriteReal,WriteText,
                     WritePolynomCoeff, ReadPolynomCoeff, ReadContinuousSpecs;
FROM Strings  IMPORT CompareStr;
FROM ConvReal IMPORT StringToReal;
FROM Regulmon IMPORT NewPolyInRegulator;
FROM RLS      IMPORT RestartEstimation;
FROM Adaptive IMPORT StartAdaptation;
FROM Design   IMPORT RSTD;
FROM MathLib  IMPORT exp, cos, sqrt;

TYPE
  DesignSet      = SET OF AdaptRegType;
VAR
  DesignRect : rectangle;
(*  DesignType : DesignTypeType; *)

PROCEDURE DesignCheck(ds:DesignSet):BOOLEAN;
BEGIN
  IF TempData.AdaptData.AdaptReg IN ds THEN
    RETURN TRUE;
  ELSE
    ErrorMessage('May not be changed now');
    RETURN FALSE;
  END;
END DesignCheck;

PROCEDURE DOdesign;
VAR integ,Aext : cpoly;
    i          : CARDINAL;
BEGIN
  WITH GlobalData DO
    IF Polnorm(Design.Am)*Polnorm(Design.Ao)<0.0001 THEN
      ErrorMessage(' Am * Ao = 0 ');
      RETURN;
    END;

    WITH integ DO
      coeffs[0]:=1.0;
      FOR i:=1 TO mdegree DO
        coeffs[i]:=0.0;
      END;
      IF Design.IntegralAction THEN
        degree:=1;
        coeffs[1]:=-1.0;
      ELSE
        degree:=0;
      END;
    END;

    WITH TempData.Regulator DO
      Aext:=Model.A; (* add additional poles to A *)
      FOR i:=Aext.degree+1 TO mdegree DO
        Aext.coeffs[i]:=0.0;
      END;
      Aext.degree:=Model.A.degree+Model.AdditionalPolesInOrigin;
      RSTD(Aext,      Design.Bplus,  Design.Bminus,
           Design.Am, Design.Bmprim, Design.Ao,     integ,   R,S1,T);

      RegType:='PolePlac'; RegType[8]:=CHR(0);

      S2.degree:=0;
      FOR i:=0 TO mdegree DO
        S2.coeffs[i]:=0.0;
      END;

      S3.degree:=0;
      FOR i:=0 TO mdegree DO
        S3.coeffs[i]:=0.0;
      END;
    END;
  END;
END DOdesign;

PROCEDURE FactorB;
CONST rel = 1.0E-8;
VAR breci,bbstar,bstab, s11,s12,s13,s14 : cpoly;
    i    : CARDINAL;
    bpl0 : REAL;
BEGIN
  WITH GlobalData.Model DO
    breci.degree:=B.degree;      (* create recprocal polynomial *)
    FOR i:=0 TO B.degree DO
      breci.coeffs[breci.degree-i] := B.coeffs[i];
    END;
    Polmul(B,breci,rel,bbstar);

    Sfactorize(bbstar,bstab);   (* stable part of B B* *)
    WITH TempData.Design DO
      Gcd(B,bstab,rel,Bplus, s11,s12,s13,s14);  (* stable part of B *)

      bpl0:=Bplus.coeffs[0];
      FOR i:=0 TO Bplus.degree DO
        Bplus.coeffs[i] := Bplus.coeffs[i]/bpl0;
      END;

      Poldiv(B,Bplus,rel,Bminus, s11);          (* unstable part of B *)
    END;
  END;
END FactorB;

PROCEDURE WriteDesignScreen;
VAR pkt:point;
BEGIN
  IF TempData.AdaptData.AdaptReg=fixed THEN
    WriteMenyLine(' SAVE   ',' COMPU  ',' TOREG  ',' INTON  ',
                  ' INTOFF ',' ',' ',' ',' ',' DEMO ',blue);
  ELSE
   WriteMenyLine(' SAVE   ',' ',' ',' ',' INTON  ',
                 ' INTOFF ',' ',' ',' ',' ',' ',' ',blue);
  END;
```

```
  FillRectangle(ScreenData.Opcomhandle,DesignRect);

  (* write in opcom area *)
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);
  WITH TempData.Design DO
    CASE TempData.AdaptData.AdaptReg OF
      fixed:
        WriteText(0.3,13.6,'Design Menu: Fixed Controller');
        WritePolynomCoeff(1.5,11.0,' B+(q)  = ',Bplus);
        WritePolynomCoeff(1.5,10.0,' B-(q)  = ',Bminus)|
      indstr:
       WriteText(0.3,13.6,'Design Menu: Indirect Adaptive Controller (ISTR)');
       WriteText(1.5,11.0,' B+(q)  =   1.00  (fixed)');
       WriteText(1.5,10.0,' B-(q)  =   B(q)  (estimated)')|
      indstr2:
       WriteText(0.3,13.6,'Design Menu: Indirect Adaptive Controller (ISTR2)');
       WriteText(1.5,11.0,' B+(q)  =   1.00  (fixed)');
       WriteText(1.5,10.0,' B-(q)  =   B(q)  (estimated)')|
      dirstr:
       WriteText(0.3,13.6,'Design Menu: Direct Adaptive Controller (DSTR)');
       WriteText(1.5,11.0,' B+(q)  =   B(q)  (not estimated)');
       WriteText(1.5,10.0,' B-(q)  =   1.00  (fixed)');
    END;

    WritePolynomCoeff(1.5, 9.0," Bm'(q) = ",Bmprim);
    WritePolynomCoeff(1.5, 8.0,' Am(q)  = ',Am);
    WritePolynomCoeff(1.5, 7.0,' Ao(q)  = ',Ao);
    IF IntegralAction THEN
      WriteText(1.5,6.0,' Integrating Regulator');
    END;
  END;

  (* ShowCursor; *)
END WriteDesignScreen;

PROCEDURE StartDesignMeny;
BEGIN
  (* set opcom area *)
  DesignRect.loleft.v  :=  0.0;   DesignRect.loleft.h  :=  0.0;
  DesignRect.upright.v := 14.5;   DesignRect.upright.h := 10.0;
  SetWindow(ScreenData.Opcomhandle,DesignRect);
  SetFillColor(ScreenData.Opcomhandle,black);
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);

  TempData.Design             := GlobalData.Design;
  TempData.AdaptData.AdaptReg := GlobalData.AdaptData.AdaptReg;
  WriteDesignScreen;
END StartDesignMeny;

PROCEDURE MouseOnDesignMeny(rect:CARDINAL);
VAR pkt:point;
    ok : BOOLEAN;
    EstAd : EstAdaptType;
    Mode  : ModeType;
    Plotting : PlottingType;
BEGIN
  GetModes(Mode,Plotting,EstAd);
  CASE rect OF
    10: IF EstAd=Adaptive THEN
          ErrorMessage('Adaptive Regulator Running');
        ELSE
          GlobalData.Design             := TempData.Design;
          GlobalData.AdaptData.AdaptReg := TempData.AdaptData.AdaptReg;
          pkt.h:=0.0; pkt.v:=0.0;
          EraseChar(ScreenData.Opcomhandle,pkt,30);
          WriteText(0.0,0.1,' Data Saved  ');
        END|
    11: IF TempData.AdaptData.AdaptReg IN DesignSet{fixed} THEN
          IF EstAd=Adaptive THEN
            ErrorMessage('Adaptive Regulator Running');
          ELSE
            GlobalData.Design := TempData.Design;
            DOdesign;
            pkt.h:=0.0; pkt.v:=0.0;
            EraseChar(ScreenData.Opcomhandle,pkt,30);
            WriteText(0.0,0.1,' New Regulator Ready  ');
          END;
        END|
    12: IF TempData.AdaptData.AdaptReg IN DesignSet{fixed} THEN
          IF EstAd=Adaptive THEN
            ErrorMessage('Adaptive Regulator Running');
          ELSE
            GlobalData.Design := TempData.Design;
            DOdesign;
            NewPolyInRegulator(TempData.Regulator);
            pkt.h:=0.0; pkt.v:=0.0;
            EraseChar(ScreenData.Opcomhandle,pkt,30);
            WriteText(0.0,0.1,' New Regulator Running');
          END;
        END|
    13: TempData.Design.IntegralAction := TRUE;
        WriteDesignScreen|
    14: TempData.Design.IntegralAction := FALSE;
        WriteDesignScreen|
    19: IF TempData.AdaptData.AdaptReg IN DesignSet{fixed} THEN
          WITH TempData.Design DO
            ReadContinuousSpecs('10 0.7',Am);
            ReadContinuousSpecs('15 0.9',Ao);
            IntegralAction := TRUE;
          END;
          GlobalData.Design := TempData.Design;
          WriteDesignScreen;
          DOdesign;
          TempData.Regulator.RegType :='DemoD';
          NewPolyInRegulator(TempData.Regulator);
          WriteText(0.0,0.1,' Demo Regulator Running');
        END;
  ELSE
    (* no action *)
  END;
END MouseOnDesignMeny;

PROCEDURE KeybOnDesignMeny(str:ARRAY OF CHAR);
VAR cmd,chb:ARRAY [0..35] OF CHAR;
(*    EstAd  : EstAdaptType;
    Mode   : ModeType;
    Plotting : PlottingType; *)
BEGIN
(*  GetModes(Mode,Plotting,EstAd);
  IF (EstAd=Adaptive) AND (Mode<>Stop) THEN
    ErrorMessage('Adaptive Regulator Running');
    RETURN;
```

```
  END;
*)
  GetWord(str,cmd);
  IF    CompareStr(cmd,'BPLUS')  =0 THEN
    IF NOT DesignCheck(DesignSet{fixed}) THEN RETURN; END;
    ReadPolynomCoeff(str,TempData.Design.Bplus);
    GetWord(str,chb);
    IF CompareStr(chb,'B')=0 THEN
      TempData.Design.Bplus := GlobalData.Model.B;
    END;
  ELSIF CompareStr(cmd,'BMINUS') =0 THEN
    IF NOT DesignCheck(DesignSet{fixed}) THEN RETURN; END;
    ReadPolynomCoeff(str,TempData.Design.Bminus);
    GetWord(str,chb);
    IF CompareStr(chb,'B')=0 THEN
      TempData.Design.Bminus := GlobalData.Model.B;
    END;
  ELSIF CompareStr(cmd,'BMPRIM') =0 THEN
    ReadPolynomCoeff(str,TempData.Design.Bmprim);
  ELSIF CompareStr(cmd,'AM') =0 THEN
    ReadPolynomCoeff(str,TempData.Design.Am);
  ELSIF CompareStr(cmd,'AO') =0 THEN
    ReadPolynomCoeff(str,TempData.Design.Ao);
  ELSIF CompareStr(cmd,'CSAM')    =0 THEN
    ReadContinuousSpecs(str,TempData.Design.Am);
  ELSIF CompareStr(cmd,'CSAO')    =0 THEN
    ReadContinuousSpecs(str,TempData.Design.Ao);
  ELSIF CompareStr(cmd,'FACTORB') =0 THEN
    IF NOT DesignCheck(DesignSet{fixed}) THEN RETURN; END;
    FactorB;
  ELSIF CompareStr(cmd,'FIXED') =0 THEN
    TempData.AdaptData.AdaptReg := fixed;
  ELSIF CompareStr(cmd,'ISTR') =0 THEN
    TempData.AdaptData.AdaptReg := indstr;
(* ELSIF CompareStr(cmd,'ISTR2') =0 THEN
    TempData.AdaptData.AdaptReg := indstr2;
  ELSIF CompareStr(cmd,'DSTR') =0 THEN
    TempData.AdaptData.AdaptReg := dirstr; *)
  ELSIF CompareStr(cmd,'HELP')    =0 THEN
  ELSE  ErrorMessage('Error in Command');
  END;
  HideCursor;
  WriteDesignScreen;
  ShowCursor;
END KeybOnDesignMeny;

(* BEGIN
  TempData.AdaptData.AdaptReg := fixed; *)
END MNYdesgn.
```

```
DEFINITION MODULE MNYEstim;

EXPORT QUALIFIED StartEstimMeny, MouseOnEstimMeny, KeybOnEstimMeny;

PROCEDURE StartEstimMeny;
PROCEDURE MouseOnEstimMeny(rect:CARDINAL);
PROCEDURE KeybOnEstimMeny(str:ARRAY OF CHAR);

END MNYEstim.
```

```
IMPLEMENTATION MODULE MNYEstim;

FROM Globals  IMPORT GlobalData, TempData, SetupDataType, EstimDataType,
                     ModeType, PlottingType, EstAdaptType, GetModes;
FROM Pcalc    IMPORT cpoly, FixDegree;
FROM RLS      IMPORT MaxStore, maxindex, col, matr,
                     EstimateParameters, RestartEstimation;
FROM Graphics IMPORT SetWindow,rectangle,point, buttonset, PolyLine,
                     WriteString,color,SetFillColor, SetLineColor,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;
FROM Screend  IMPORT ScreenData, ErrorMessage, WarningMessage, WriteMenyLine;
FROM Opcom    IMPORT GetWord, WriteInt, WriteReal, WriteText,
                     WritePolynomCoeff, ReadPolynomCoeff, ReadContinuousSpecs;
FROM ShowPar  IMPORT ShowStoredParameters;
FROM ConvReal IMPORT StringToReal, RealToString;
FROM NumberConversion IMPORT CardToString;
FROM Strings  IMPORT CompareStr, Concat;

TYPE
  ParVect = ARRAY[0..MaxStore] OF point;
VAR
  EstimRect  : rectangle;
  (* th         : ARRAY[1..6] OF ParVect;
  thmin,thmax: ARRAY[1..6] OF REAL; *)

(* ------------------------------------------------------------------------ *)

PROCEDURE WriteEstimScreen;
VAR pkt: ARRAY[0..1] OF point;
    num : ARRAY[0..10] OF CHAR;
    str : ARRAY[0..80] OF CHAR;
    i,j : CARDINAL;
BEGIN
  (* set opcom area *)
  EstimRect.loleft.v  :=  0.0;   EstimRect.loleft.h  :=  0.0;
  EstimRect.upright.v := 12.0;   EstimRect.upright.h := 10.0;
  WITH ScreenData DO
    SetWindow(Opcomhandle,EstimRect);
    SetFillColor(Opcomhandle,black);
    SetTextColor(Opcomhandle,Textcolor);
  END;

  WriteMenyLine(' SAVE   ',' DMATR  ',' REPRIN ',' ',' ',
                '        ',' ',' ',' ',' ',      ',blue);

  FillRectangle(ScreenData.Opcomhandle,EstimRect);
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);
  WriteText(0.3,11.3,'Estim Menu');
  WITH TempData.EstimData DO
    WriteReal(2.2,10.5,'Lambda= ',Lambda);
    WriteReal(2.2, 9.5,'D0     = ',D0);
    WriteText(4.7,10.5,'Filter');
    WritePolynomCoeff(5.7,10.5,'Bf(q)= ',Bf);
    WritePolynomCoeff(5.7, 9.5,'Af(q)= ',Af);

    WITH EstimateParameters DO
      FOR i:=1 TO N DO     (* Write D and L *)
        IF i=1 THEN
          str := 'L=';
        ELSE
          str := '  ';
        END;
        str[5] :=CHR(0);
        FOR j:=1 TO i DO
          RealToString(L[j,i],num,5);
          Concat(str,' ',str);
          Concat(str,num,str);
        END;
        IF i=1 THEN
          Concat(str,'      D=',str);
        ELSE
          Concat(str,'        ',str);
        END;
        RealToString(D[i],num,7);
        Concat(str,num,str);
        WriteText(0.2,9.0-0.8*FLOAT(i),str);
      END;
    END;
  END;
END WriteEstimScreen;

PROCEDURE StartEstimMeny;
BEGIN
  WITH GlobalData.EstimData DO
    HistoryCard := TRUNC( HistoryTime/GlobalData.SetupData.RealTsamp+0.5);
    IF HistoryCard>=MaxStore-10 THEN (* -10 for safety *)
      HistoryCard:= MaxStore-11;
      HistoryTime:= FLOAT(HistoryCard)*GlobalData.SetupData.RealTsamp;
      WarningMessage('History Changed');
    END;
  END;

  TempData.EstimData := GlobalData.EstimData;
  WriteEstimScreen;
END StartEstimMeny;

PROCEDURE MouseOnEstimMeny(rect:CARDINAL);
VAR EstAd : EstAdaptType;
    Mode  : ModeType;
    ok    : BOOLEAN;
    Plotting : PlottingType;
BEGIN
  GetModes(Mode,Plotting,EstAd);
  CASE rect OF
    10: IF EstAd=Off THEN
          GlobalData.EstimData := TempData.EstimData;
          WriteText(0.0,0.1,' Data saved');
        ELSE
          ErrorMessage('No Changes allowed now');
        END|
    11: WITH EstimateParameters DO
          ShowStoredParameters('D-MATRIX',1,N,'D',1,N,' ',0,0,' ',0,0);
        END|
    12: WriteEstimScreen;
  ELSE
    (* no action *)
  END;
END MouseOnEstimMeny;

PROCEDURE KeybOnEstimMeny(str:ARRAY OF CHAR);
VAR cmd    : ARRAY [0..21] OF CHAR;
    tmppol : cpoly;
```

```
    valr   : REAL;
    pos,i  : CARDINAL;
    valc   : CARDINAL;
    ok     : BOOLEAN;
BEGIN
  GetWord(str,cmd);

  pos :=0;
  valr:=0.0;
  IF CompareStr(cmd,'AF')  =0 THEN
    ReadPolynomCoeff(str,tmppol);
    IF ABS(tmppol.coeffs[0])<0.001 THEN
      ErrorMessage('Af[0] must be nonzero');
    ELSIF tmppol.degree>2 THEN
      ErrorMessage('degree Af <= 2');
    ELSE
      FOR i:=tmppol.degree+1 TO 2 DO
        tmppol.coeffs[i]:=0.0;
      END;
      tmppol.degree := 2;
      TempData.EstimData.Af := tmppol;
    END;
  ELSIF CompareStr(cmd,'CSAF')  =0 THEN
    ReadContinuousSpecs(str,tmppol);
    IF tmppol.degree>2 THEN
      ErrorMessage('degree Af <= 2');
    ELSE
      FOR i:=tmppol.degree+1 TO 2 DO
        tmppol.coeffs[i]:=0.0;
      END;
      tmppol.degree := 2;
      TempData.EstimData.Af := tmppol;
    END;
  ELSIF CompareStr(cmd,'BF')  =0 THEN
    ReadPolynomCoeff(str,tmppol);
    IF tmppol.degree>2 THEN
      ErrorMessage('degree Bf <= 2');
    ELSE
      TempData.EstimData.Bf := tmppol;
      FixDegree(TempData.EstimData.Bf,2);
    END;
  ELSIF CompareStr(cmd,'LAMBDA') =0 THEN
    StringToReal(str,pos,valr);
    IF (0.1<valr) AND (valr<=1.0) THEN
      TempData.EstimData.Lambda := valr;
    ELSE
      ErrorMessage('Lambda out of range');
    END;
  ELSIF CompareStr(cmd,'D0') =0 THEN
    StringToReal(str,pos,valr);
    IF 0.0<valr THEN
      TempData.EstimData.D0 := valr;
    ELSE
      ErrorMessage('D0 out of range');
    END;
  ELSIF CompareStr(cmd,'HIST') =0 THEN
    pos := 0;
    StringToReal(str,pos,valr);
    valc := TRUNC(valr/GlobalData.SetupData.RealTsamp+0.5);
    IF (5<valc) AND (valc< MaxStore-10) THEN (* -10 for safety *)
      WITH GlobalData.EstimData DO
        HistoryTime := valr;
        HistoryCard := valc;
      END;
    ELSE
      ErrorMessage('History too short or long');
    END;
    RETURN;
  ELSE  ErrorMessage('Error in Command');
  END;
  HideCursor;
  WriteEstimScreen;
  ShowCursor;
END KeybOnEstimMeny;

END MNYEstim.
```

```
DEFINITION MODULE MNYmodel;

EXPORT QUALIFIED StartModelMeny, MouseOnModelMeny, KeybOnModelMeny;

PROCEDURE StartModelMeny;
PROCEDURE MouseOnModelMeny(rect:CARDINAL);
PROCEDURE KeybOnModelMeny(str:ARRAY OF CHAR);

END MNYmodel.
```

```
IMPLEMENTATION MODULE MNYmodel;

FROM Pcalc     IMPORT mdegree, coeffvector, cpoly;
FROM Globals   IMPORT GetModes, SetMode, ModeType, GlobalData, TempData,
                      EstAdaptType, RegulatorType, PlottingType;
FROM Graphics  IMPORT SetWindow,rectangle,point, buttonset,
                      WriteString,color,SetFillColor,
                      FillRectangle, SetTextColor, HideCursor, ShowCursor;
FROM Screend   IMPORT ScreenData, ErrorMessage, WarningMessage, WriteMenyLine;
FROM Opcom     IMPORT GetWord, WriteInt, WriteReal,WriteText,
                      WritePolynomCoeff, ReadPolynomCoeff;
FROM Strings   IMPORT CompareStr;
FROM ConvReal  IMPORT StringToReal;
FROM NumberConversion IMPORT StringToCard;
FROM MathLib   IMPORT exp;
FROM ShowPar   IMPORT ShowStoredParameters;
FROM RLS       IMPORT MaxStore, RestartEstimation;
(* FROM DProc     IMPORT DebugProcessPoly; *)
FROM BodePlot IMPORT DBode, DBode2;

CONST
  pi=3.14159;

VAR
  ModelRect:rectangle;

PROCEDURE WriteModelScreen;
VAR pkt:point;
BEGIN
  WriteMenyLine(' SAVE   ',' SAMPLE ',' SHOWB  ',' SHOWA ',' SHOWAB ',
                ' BODE  ',' REPRIN ',' RESET ',' ',' ',blue);

  SetWindow(ScreenData.Opcomhandle,ModelRect);
  SetFillColor(ScreenData.Opcomhandle,black);
  FillRectangle(ScreenData.Opcomhandle,ModelRect);
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);
  WriteText(0.3,13.6,'Model Menu');

  (* write in opcom area *)
  WITH TempData.Model DO
    WritePolynomCoeff(1.5, 9.0,' B(q)  = ',B);
    WritePolynomCoeff(1.5, 8.0,' A(q)  = ',A);
    WritePolynomCoeff(1.5, 7.0,' C(q)  = ',C);
    WriteInt(1.5,6.0,' Additional Poles In Origin = ',AdditionalPolesInOrigin);
  END;

(* for debugging with internal process
  WITH DebugProcessPoly DO
    WritePolynomCoeff(1.5, 4.0,' MB(q)  = ',B);
    WritePolynomCoeff(1.5, 3.0,' MA(q)  = ',A);
    WritePolynomCoeff(1.5, 2.0,' MC(q)  = ',C);
  END;
  --------------- *)
  (* ShowCursor; *)
END WriteModelScreen;

PROCEDURE StartModelMeny;
BEGIN
  (* set opcom area *)
  ModelRect.loleft.v  :=  0.0;   ModelRect.loleft.h  :=  0.0;
  ModelRect.upright.v := 14.5;   ModelRect.upright.h := 10.0;
  TempData.Model := GlobalData.Model;
  WriteModelScreen;
END StartModelMeny;

PROCEDURE MouseOnModelMeny(rect:CARDINAL);
CONST a=0.12; b=24.7; (* motor model *)
VAR i    : CARDINAL;
    a0,h : REAL;
    EstAd: EstAdaptType;
    Mode : ModeType;
    Plotting: PlottingType;
    ok   : BOOLEAN;
BEGIN
  GetModes(Mode,Plotting,EstAd);
  CASE rect OF
    10: IF EstAd<>Off THEN
          ErrorMessage('Estimator Runing');
          RETURN;
        END;
        WITH TempData.Model DO
          IF B.degree>=(A.degree+AdditionalPolesInOrigin) THEN
            ErrorMessage(' No delay in process ' );
            RETURN;
          END;
          delay := A.degree + AdditionalPolesInOrigin - B.degree;
          IF delay>4 THEN
            ErrorMessage(' Too Many Added Poles ' );
            RETURN;
          END;
          a0:=A.coeffs[0];   (* norm coeffecients *)
          FOR i:=0 TO A.degree DO
            A.coeffs[i] :=  A.coeffs[i] / a0;
            B.coeffs[i] :=  B.coeffs[i] / a0;
          END;
        END;
        GlobalData.Model := TempData.Model;
        WriteModelScreen;
        WriteText(0.0,0.1,' Data Saved')|
    11: IF EstAd<>Off THEN
          ErrorMessage('Estimator Runing');
          RETURN;
        END;
        h:=GlobalData.SetupData.RealTsamp;
        WITH TempData.Model DO
          B.coeffs[0] := (b/a)*(a*h-1.0+exp(-a*h))/a;
          B.coeffs[1] := (b/a)*(1.0-exp(-a*h)-a*h*exp(-a*h))/a;
          A.coeffs[0] := 1.0;
          A.coeffs[1] := -(1.0+exp(-a*h));
          A.coeffs[2] := exp(-a*h);
          B.degree := 1;
          A.degree := 2;
          AdditionalPolesInOrigin := 0;
          delay    := 1;
        END;
        GlobalData.Design.Bminus := TempData.Model.B;
        GlobalData.Design.Bplus  := TempData.Model.B;
        WITH GlobalData.Design.Bplus DO
          degree:=0;
          coeffs[0]:=1.0;
        END;
```

```
    WITH GlobalData.Design.Bmprim DO
      degree:=0;
      coeffs[0]:=1.0;
    END;

    GlobalData.Model := TempData.Model;
    WriteModelScreen;
    WriteText(0.0,0.0,'Data Saved for Servo')|
12: WITH GlobalData.Model DO
      IF CompareStr(GlobalData.EstimData.EstPurpose,'MODELPOLY')=0 THEN
        ShowStoredParameters('Theta', A.degree+1,A.degree+B.degree+1,
                             'B',0,B.degree,' ',0,0,' ',0,0);
      ELSE
        ErrorMessage('Not estimating A and B');
      END;
    END|
13: WITH GlobalData.Model DO
      IF CompareStr(GlobalData.EstimData.EstPurpose,'MODELPOLY')=0 THEN
        ShowStoredParameters('Theta',1,A.degree,
                             'A',1,A.degree,' ',0,0,' ',0,0);
      ELSE
        ErrorMessage('Not estimating A and B');
      END;
    END|
14: WITH GlobalData.Model DO
      IF CompareStr(GlobalData.EstimData.EstPurpose,'MODELPOLY')=0 THEN
        ShowStoredParameters('Theta',1,A.degree+B.degree+1,
                             'A',1,A.degree,'B',0,B.degree,' ',0,0);
      ELSE
        ErrorMessage('Not estimating A and B');
      END;
    END|
15: IF EstAd<>Off THEN
      TempData.Model := GlobalData.Model;
    END;
    WITH TempData DO (* variable wlow and whigh *)
      IF (CompareStr(GlobalData.EstimData.EstPurpose,'MODELPOLY')=0)
         AND (EstAd<>Off) THEN
        DBode2(Model.A,Model.B,Model.AdditionalPolesInOrigin,
               EstimData.Af,EstimData.Bf,0,
               GlobalData.Model.BodeWlow,GlobalData.Model.BodeWhigh,
               GlobalData.SetupData.RealTsamp);

      ELSE
        DBode(Model.A,Model.B,Model.AdditionalPolesInOrigin,
              GlobalData.Model.BodeWlow,GlobalData.Model.BodeWhigh,
              GlobalData.SetupData.RealTsamp);
      END;
    END|
16: IF EstAd<>Off THEN
      TempData.Model := GlobalData.Model;
    END;
    WriteModelScreen|
17: (* are there any modes when command not shold be invoked *)
    RestartEstimation(TRUE,ok);
    IF ok THEN
      TempData.Model := GlobalData.Model;
      WriteModelScreen;
      WriteText(0.0,0.0,'Model Reset and Saved');
    ELSE
      ErrorMessage('deg(A)+deg(B) too large');
        END;
    ELSE
      (* no action *)
    END;
END MouseOnModelMeny;

PROCEDURE KeybOnModelMeny(str:ARRAY OF CHAR);
VAR cmd,cmd1:ARRAY [0..35] OF CHAR;
    tmppol:cpoly;
    EstAd: EstAdaptType;
    Mode : ModeType;
    Plotting  : PlottingType;
    valc,pos,degold : CARDINAL;
    valr,valr2: REAL;
    ok : BOOLEAN;
BEGIN
  GetWord(str,cmd);

  IF CompareStr(cmd,'HIST') =0 THEN (* always allowed changes *)
    pos := 0;
    StringToReal(str,pos,valr);
    valc := TRUNC(valr/GlobalData.SetupData.RealTsamp+0.5);
    IF (5<valc) AND (valc< MaxStore-10) THEN (* -10 for safety *)
      WITH GlobalData.EstimData DO
        HistoryTime := valr;
        HistoryCard := valc;
      END;
    ELSE
      ErrorMessage('History too short or long');
    END;
    RETURN;
  ELSIF CompareStr(cmd,'BODEFREQ')    =0 THEN
    valr:=-2.0; valr2:=-2.0;
    pos:=0;
    StringToReal(str,pos,valr);
    valc := pos;
    StringToReal(str,pos,valr2);
    IF pos=valc THEN
      GetWord(str,cmd1);  (* dummy call to blank first argument *)
      GetWord(str,cmd1);
      IF CompareStr(cmd1,'PI')=0 THEN
        valr2:=pi/GlobalData.SetupData.RealTsamp*0.999;
      ELSE
        ErrorMessage('Not valid frequency');
        RETURN;
      END;
    END;
    IF (0.0<valr) AND (valr<valr2) THEN
      IF GlobalData.SetupData.RealTsamp*valr2>pi THEN
        ErrorMessage('whigh*Tsamp not <= pi');
        RETURN;
      ELSE
        GlobalData.Model.BodeWlow  := valr;
        GlobalData.Model.BodeWhigh := valr2;
        RETURN;
      END;
    ELSE
      ErrorMessage('not 0<wlow<whigh');
      RETURN;
    END;
(* debug process setting
```

```
ELSIF CompareStr(cmd,'MA') =0 THEN
  ReadPolynomCoeff(str,DebugProcessPoly.A);
  RETURN;
ELSIF CompareStr(cmd,'MB') =0 THEN
  ReadPolynomCoeff(str,DebugProcessPoly.B);
  RETURN;
ELSIF CompareStr(cmd,'MC') =0 THEN
  ReadPolynomCoeff(str,DebugProcessPoly.C);
  RETURN;
---------- *)
END;

GetModes(Mode,Plotting,EstAd);
IF EstAd<>Off THEN
  ErrorMessage('Estimator Runing');
  RETURN;
END;

IF    CompareStr(cmd,'A')  =0 THEN
  ReadPolynomCoeff(str,tmppol);
  IF ABS(tmppol.coeffs[0])<0.001 THEN
    ErrorMessage('A[0] must be nonzero');
  ELSE
    TempData.Model.A := tmppol;
  END;
ELSIF CompareStr(cmd,'B') =0 THEN
  ReadPolynomCoeff(str,TempData.Model.B);
ELSIF CompareStr(cmd,'C') =0 THEN
  ReadPolynomCoeff(str,TempData.Model.C);
ELSIF CompareStr(cmd,'ADDPOL') =0 THEN
  StringToCard(str,valc,ok);
  IF ok THEN
    TempData.Model.AdditionalPolesInOrigin := valc;
  ELSE
    ErrorMessage(' Number out of range ');
  END;
ELSIF CompareStr(cmd,'DEG') =0 THEN
  GetWord(str,cmd1);
  StringToCard(str,valc,ok);
  IF ok AND (valc<=5) THEN
    WITH TempData.Model DO
      CASE cmd1[0] OF
        'A': degold := A.degree;
             FOR pos:=degold+1 TO valc DO A.coeffs[pos]:=0.0; END;
             A.degree := valc|
        'B': degold := B.degree;
             FOR pos:=degold+1 TO valc DO B.coeffs[pos]:=0.0; END;
             B.degree := valc|
        'C': degold := C.degree;
             FOR pos:=degold+1 TO valc DO C.coeffs[pos]:=0.0; END;
             C.degree := valc;
      ELSE
        ErrorMessage(' not Valid Polynomial');
        degold:=0;
      END;
    END;
    IF valc<degold THEN
      WarningMessage('Polynomial chopped');
    END;
  ELSE
    ErrorMessage(' not (degree<=5)');
    END;
  ELSIF CompareStr(cmd,'HELP')    =0 THEN
  ELSE  ErrorMessage('Error in Command');
  END;
  HideCursor;
  WriteModelScreen;
  ShowCursor;
END KeybOnModelMeny;

END MNYmodel.
```

```
DEFINITION MODULE MNYplot;

EXPORT QUALIFIED StartPlotMeny, MouseOnPlotMeny, KeybOnPlotMeny;

PROCEDURE StartPlotMeny;
PROCEDURE MouseOnPlotMeny(rect:CARDINAL);
PROCEDURE KeybOnPlotMeny(str:ARRAY OF CHAR);

END MNYplot.
```

```
IMPLEMENTATION MODULE MNYplot;

FROM Screend  IMPORT ScreenData, ErrorMessage, WriteMenyLine;
FROM Globals  IMPORT SetMode, ModeType, GlobalData;
FROM Graphics IMPORT SetWindow,rectangle, color, SetFillColor, buttonset,
                     FillRectangle, HideCursor, ShowCursor;
VAR
  OpcRect : rectangle;

PROCEDURE WritePlotScreen;
BEGIN
  WriteMenyLine(' RY1    ',' RY2    ',' RY3    ',' RY12   ',' RY13   ',
                ' RY23   ',' RY123  ',' ',' ',' ',blue);
END WritePlotScreen;

PROCEDURE StartPlotMeny;
BEGIN
  OpcRect.loleft.v  :=  0.0;   OpcRect.loleft.h  :=  0.0;
  OpcRect.upright.v :=  1.0;   OpcRect.upright.h :=  1.0;
  SetWindow(ScreenData.Opcomhandle,OpcRect);
  SetFillColor(ScreenData.Opcomhandle,white);

  WritePlotScreen;
END StartPlotMeny;

PROCEDURE MouseOnPlotMeny(rect:CARDINAL);
BEGIN
  CASE rect OF
    10: GlobalData.PlotWhichCurve := {1}    |
    11: GlobalData.PlotWhichCurve := {2}    |
    12: GlobalData.PlotWhichCurve := {3}    |
    13: GlobalData.PlotWhichCurve := {1,2}  |
    14: GlobalData.PlotWhichCurve := {1,3}  |
    15: GlobalData.PlotWhichCurve := {2,3}  |
    16: GlobalData.PlotWhichCurve := {1,2,3};
  ELSE
    (* no action *)
  END;
END MouseOnPlotMeny;

PROCEDURE KeybOnPlotMeny(str:ARRAY OF CHAR);
VAR cmd:ARRAY [0..35] OF CHAR;
BEGIN
  ErrorMessage('Commands not Defined');
END KeybOnPlotMeny;

END MNYplot.
```

```
DEFINITION MODULE MNYrefs;

EXPORT QUALIFIED StartRefSigMeny, MouseOnRefSigMeny, KeybOnRefSigMeny;

PROCEDURE StartRefSigMeny;
PROCEDURE MouseOnRefSigMeny(rect:CARDINAL);
PROCEDURE KeybOnRefSigMeny(str:ARRAY OF CHAR);

END MNYrefs.
```

```
IMPLEMENTATION MODULE MNYrefs;

FROM Globals  IMPORT SetMode, ModeType, GlobalData, TempData,
                     RefSignalType, SignalTypeType;
FROM Graphics IMPORT SetWindow,rectangle,point, buttonset,
                     WriteString,color,SetFillColor,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;
                     (* buttontype   *)
FROM Screend  IMPORT ScreenData, ErrorMessage, WriteMenyLine;
FROM Opcom    IMPORT GetWord, WriteInt, WriteReal,WriteText,
                     WritePolynomCoeff, ReadPolynomCoeff;
FROM ConvReal IMPORT StringToReal;
FROM Strings  IMPORT CompareStr;
FROM NumberConversion IMPORT StringToCard;
(* FROM Debug IMPORT Write; *)

VAR
  SignalRect   : rectangle;

PROCEDURE WriteSignalScreen;
VAR pkt:point;
BEGIN              (* Write these also in *OnSetupMeny ?? ==> faster *)
  WriteMenyLine(' SAVE   ',' EXTERN ',' SQUARE ',' TRIAN  ',' SINE   ',
                ' STEP   ',' RAMP   ',' ',' ',' ',blue);

  FillRectangle(ScreenData.Opcomhandle,SignalRect);
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);
  WriteText(0.3,10.3,'Reference Signal Menu');

  (* write in opcom area *)
  WITH TempData.RefSignal DO
    CASE SignalType OF
      external: WriteText(2.0,7.0,' SignalType =  external ')|
      square  : WriteText(2.0,7.0,' SignalType =  square   ')|
      triangle: WriteText(2.0,7.0,' SignalType =  triangle ')|
      sine    : WriteText(2.0,7.0,' SignalType =  sine     ')|
      step    : WriteText(2.0,7.0,' SignalType =  step     ')|
      ramp    : WriteText(2.0,7.0,' SignalType =  ramp     ');
    END;
    WriteReal(2.0,6.0,' Mean       = ',Mean);
    WriteReal(2.0,5.0,' Amplitude  = ',Amplitude);
    WriteReal(2.0,4.0,' Period     = ',Period);
  END;

  (* ShowCursor; *)
END WriteSignalScreen;

PROCEDURE StartRefSigMeny;
BEGIN
  (* set opcom area *)
  SignalRect.loleft.v  :=  0.0;   SignalRect.loleft.h  :=  0.0;
  SignalRect.upright.v := 11.0;   SignalRect.upright.h := 10.0;
  WITH ScreenData DO
    SetWindow(Opcomhandle,SignalRect);
    SetFillColor(Opcomhandle,black);
    SetTextColor(Opcomhandle,Textcolor);
  END;

  TempData.RefSignal := GlobalData.RefSignal;
  WriteSignalScreen;
END StartRefSigMeny;
```

```
PROCEDURE MouseOnRefSigMeny(rect:CARDINAL);
BEGIN
  CASE rect OF
    10: WITH TempData.RefSignal DO
          IF ABS(Mean)+ABS(Amplitude)>1.0 THEN
            ErrorMessage('Mean + Amplitude > 1.0');
          ELSE
            TempData.RefSignal.TimeInPeriod :=0.0;
            GlobalData.RefSignal := TempData.RefSignal;
            WriteText(0.0,0.0,' Data Saved ');
          END;
        END|
    11: TempData.RefSignal.SignalType := external;
        WriteSignalScreen|
    12: TempData.RefSignal.SignalType := square;
        WriteSignalScreen|
    13: TempData.RefSignal.SignalType := triangle;
        WriteSignalScreen|
    14: TempData.RefSignal.SignalType := sine;
        WriteSignalScreen|
    15: TempData.RefSignal.SignalType := step;
        WriteSignalScreen|
    16: TempData.RefSignal.SignalType := ramp;
        WriteSignalScreen;
  ELSE
    (* no action *)
  END;
END MouseOnRefSigMeny;

PROCEDURE KeybOnRefSigMeny(str:ARRAY OF CHAR);
VAR cmd1 : ARRAY [0..21] OF CHAR;
    valr : REAL;
    pos  : CARDINAL;
BEGIN
  GetWord(str,cmd1);
  pos :=0;
  valr:=0.0;
  StringToReal(str,pos,valr);

  HideCursor;
  IF CompareStr(cmd1,'MEAN')    =0 THEN
    IF (pos>0) AND (-1.0<=valr) AND (valr<=1.0) THEN
      TempData.RefSignal.Mean :=valr;
      WriteSignalScreen;
    ELSE
      ErrorMessage('Mean out of range');
    END;
  ELSIF CompareStr(cmd1,'AMPLITUDE')    =0 THEN
    IF (pos>0) AND (-1.0<=valr) AND (valr<=1.0) THEN
      TempData.RefSignal.Amplitude :=valr;
      WriteSignalScreen;
    ELSE
      ErrorMessage('Amplitude out of range');
    END;
  ELSIF CompareStr(cmd1,'PERIOD')     =0 THEN
    IF (pos>0) AND (0.0<valr) AND (valr<=1000.0) THEN
      TempData.RefSignal.Period :=valr;
      WriteSignalScreen;
    ELSE
      ErrorMessage('Period out of range');
```

```
    END;
  ELSE  ErrorMessage('Error in Command');
  END;
  ShowCursor;
END KeybOnRefSigMeny;

END MNYrefs.
```

```
DEFINITION MODULE MNYregul;

EXPORT QUALIFIED StartRegulMeny, MouseOnRegulMeny, KeybOnRegulMeny;

PROCEDURE StartRegulMeny;
PROCEDURE MouseOnRegulMeny(rect:CARDINAL);
PROCEDURE KeybOnRegulMeny(str:ARRAY OF CHAR);

END MNYregul.
```

```
IMPLEMENTATION MODULE MNYregul;

FROM Globals  IMPORT GetModes, SetMode, ModeType, PlottingType,
                     TempData, RegulatorType, EstAdaptType,
                     GlobalData,SetupDataType, EstimDataType;
FROM Pcalc    IMPORT mdegree, coeffvector, cpoly, Poladd;
FROM Regulmon IMPORT NewPolyInRegulator, GetPolyInRegulator, InitRegulmonitor;
FROM Graphics IMPORT SetWindow,rectangle,point, buttonset,
                     WriteString,color,SetFillColor,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;
FROM Screend  IMPORT ScreenData, ErrorMessage, WriteMenyLine;
FROM Opcom    IMPORT GetWord, WriteInt, WriteReal,WriteText,
                     WritePolynomCoeff, ReadPolynomCoeff;
FROM Strings  IMPORT CompareStr;
FROM ConvReal IMPORT StringToReal;
FROM ShowPar  IMPORT ShowStoredParameters;
FROM RLS      IMPORT MaxStore;

VAR
  RunningRegulator : RegulatorType;
  RegulRect        : rectangle;

PROCEDURE MakePoly(deg:CARDINAL; gain, c0,c1,c2:REAL; VAR p:cpoly);
VAR i:CARDINAL;
BEGIN
  WITH p DO
    degree:=deg;
    coeffs[0] := gain*c0;
    coeffs[1] := gain*c1;
    coeffs[2] := gain*c2;
    FOR i:=degree+1 TO mdegree DO
      coeffs[i] := 0.0;
    END;
  END;
END MakePoly;

PROCEDURE WriteRegulScreen;
VAR pkt:point;
BEGIN
  WriteMenyLine(' SAVE   ',' INIT   ',' SHOWR  ',' SHOWS1 ',' SHOWS2 ',
                ' SHOWRS ',' REPRIN ',' ',' ',' DEMO ',blue);

  SetWindow(ScreenData.Opcomhandle,RegulRect);
  SetFillColor(ScreenData.Opcomhandle,black);
  FillRectangle(ScreenData.Opcomhandle,RegulRect);
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);

  (* write in opcom area *)
  WITH RunningRegulator DO
    WriteText( 0.2,13.0,'Running:   ');
    WriteText( 0.2,12.0,RegType);
    WritePolynomCoeff(1.5,13.0,' R(q)  = ',R);
    WritePolynomCoeff(1.5,12.0,' S1(q) = ',S1);
    WritePolynomCoeff(1.5,11.0,' S2(q) = ',S2);
    WritePolynomCoeff(1.5,10.0,' S3(q) = ',S3);
    WritePolynomCoeff(1.5, 9.0,' T(q)  = ',T);
    WritePolynomCoeff(1.5, 8.0,' Ao(q) = ',Ao);
  END;

  WITH TempData.Regulator DO
    WriteText( 0.2, 6.5,'Ready  :   ');
    WriteText( 0.2, 5.5,RegType);
    WritePolynomCoeff(1.5,6.5,' R(q)  = ',R);
    WritePolynomCoeff(1.5,5.5,' S1(q) = ',S1);
    WritePolynomCoeff(1.5,4.5,' S2(q) = ',S2);
    WritePolynomCoeff(1.5,3.5,' S3(q) = ',S3);
    WritePolynomCoeff(1.5,2.5,' T(q)  = ',T);
    WritePolynomCoeff(1.5,1.5,' Ao(q) = ',Ao);
  END;

  (* ShowCursor; *)
END WriteRegulScreen;

PROCEDURE StartRegulMeny;
BEGIN
  (* set opcom area *)
  RegulRect.loleft.v  :=  0.0;   RegulRect.loleft.h  :=  0.0;
  RegulRect.upright.v := 14.5;   RegulRect.upright.h := 10.0;

  GetPolyInRegulator(RunningRegulator);
  WriteRegulScreen;
END StartRegulMeny;

PROCEDURE MouseOnRegulMeny(rect:CARDINAL);
VAR EstAd:EstAdaptType;
    Mode: ModeType;
    Plotting :PlottingType;
BEGIN
  GetModes(Mode,Plotting,EstAd);
  CASE rect OF
    10: IF EstAd=Adaptive THEN
          ErrorMessage('Adaptive Regulator Runing');
        ELSE
          NewPolyInRegulator(TempData.Regulator);
          GetPolyInRegulator(RunningRegulator);
          WriteRegulScreen;
        END|
    11: IF EstAd=Adaptive THEN
          ErrorMessage('Adaptive Regulator Runing');
        ELSIF Mode<>Stop THEN
          ErrorMessage('Not Stopped Regulator');
        ELSE
          InitRegulmonitor;
          GetPolyInRegulator(RunningRegulator);
          WriteRegulScreen;
        END|
    12: WITH RunningRegulator DO
          IF CompareStr(GlobalData.EstimData.EstPurpose,'REGULPOLY')=0 THEN
            ShowStoredParameters('RST',1,R.degree,
                                 'R',1,R.degree,' ',0,0,' ',0,0);
          ELSE
            ErrorMessage('Not estimating R and S');
          END;
        END|
    13: WITH RunningRegulator DO
          IF CompareStr(GlobalData.EstimData.EstPurpose,'REGULPOLY')=0 THEN
            ShowStoredParameters('RST',R.degree+1,R.degree+S1.degree+1,
                                 'S1',0,S1.degree,' ',0,0,' ',0,0);
          ELSE
            ErrorMessage('Not estimating R and S');
          END;
        END|
```

```
    14: WITH RunningRegulator DO
          IF CompareStr(GlobalData.EstimData.EstPurpose,'REGULPOLY')=0 THEN
            ShowStoredParameters('RST',R.degree+S1.degree+2,
                                 R.degree+S1.degree+S2.degree+2,
                                 'S2',0,S2.degree,' ',0,0,' ',0,0);
          ELSE
            ErrorMessage('Not estimating R and S');
          END;
        END|
    15: WITH RunningRegulator DO
          IF CompareStr(GlobalData.EstimData.EstPurpose,'REGULPOLY')=0 THEN
            ShowStoredParameters('RST',1,R.degree+S1.degree+S2.degree+2,
                                 'R', 1,R.degree,
                                 'S1',0,S1.degree,
                                 'S2',0,S2.degree);
          ELSE
            ErrorMessage('Not estimating R and S');
          END;
        END|
    16: IF EstAd<>Off THEN
          GetPolyInRegulator(RunningRegulator);
        END;
        WriteRegulScreen|
    19: IF EstAd=Adaptive THEN
          ErrorMessage('Adaptive Regulator Runing');
        ELSE
          WITH TempData.Regulator DO
          RegType := 'DemoR';
          R.degree:=2; S1.degree:=2; S2.degree:=0; S3.degree:=0; T.degree:=2;
          R.coeffs[0] := 1.0;    R.coeffs[1] :=-0.3731; R.coeffs[2] :=-0.6269;
          S1.coeffs[0]:= 9.7487; S1.coeffs[1]:=-13.615; S1.coeffs[2]:= 5.1909;
          S2.coeffs[0]:= 0.0;    S2.coeffs[1]:= 0.0;    S2.coeffs[2]:= 0.0;
          S3.coeffs[0]:= 0.0;    S3.coeffs[1]:= 0.0;    S3.coeffs[2]:= 0.0;
          T.coeffs[0] := 2.0206; T.coeffs[1] :=-0.8316; T.coeffs[2] := 0.1358;
          END;
          NewPolyInRegulator(TempData.Regulator);
          GetPolyInRegulator(RunningRegulator);
          WriteRegulScreen;
        END;
  ELSE
    (* no action *)
  END;
END MouseOnRegulMeny;

PROCEDURE KeybOnRegulMeny(str:ARRAY OF CHAR);
VAR cmd:ARRAY [0..35] OF CHAR;
    tmppol,p1,p2,p3  : cpoly;
    EstAd            : EstAdaptType;
    Mode             : ModeType;
    Plotting         : PlottingType;
    K,Ti,Td,N,h,valr : REAL;
    pos,posold ,valc : CARDINAL;
BEGIN
  GetModes(Mode,Plotting,EstAd);
  IF EstAd=Adaptive THEN
    ErrorMessage('Adaptive Regulator Runing');
    RETURN;
  END;

  GetWord(str,cmd);
  IF    CompareStr(cmd,'R')  =0 THEN
    ReadPolynomCoeff(str,tmppol);
    IF ABS(tmppol.coeffs[0])<0.001 THEN
      ErrorMessage('R[0] must be nonzero');
    ELSE
      TempData.Regulator.R := tmppol;
      TempData.Regulator.RegType := 'HandMade';
    END;
  ELSIF CompareStr(cmd,'S1') =0 THEN
    ReadPolynomCoeff(str,TempData.Regulator.S1);
    TempData.Regulator.RegType := 'HandMade';
  ELSIF CompareStr(cmd,'S2') =0 THEN
    ReadPolynomCoeff(str,TempData.Regulator.S2);
    TempData.Regulator.RegType := 'HandMade';
  ELSIF CompareStr(cmd,'S3') =0 THEN
    ReadPolynomCoeff(str,TempData.Regulator.S3);
    TempData.Regulator.RegType := 'HandMade';
  ELSIF CompareStr(cmd,'T')  =0 THEN
    ReadPolynomCoeff(str,TempData.Regulator.T);
    TempData.Regulator.RegType := 'HandMade';
  ELSIF CompareStr(cmd,'AO') =0 THEN
    ReadPolynomCoeff(str,TempData.Regulator.Ao);
    TempData.Regulator.RegType := 'HandMade';
  ELSIF CompareStr(cmd,'PID')    =0 THEN
    pos := 0;
    StringToReal(str,pos,K);
    IF pos=0 THEN
      ErrorMessage('Must have arguments');
      RETURN;
    END;
    posold:=pos; StringToReal(str,pos,Ti); IF pos=posold THEN Ti:=-1.0; END;
    posold:=pos; StringToReal(str,pos,Td); IF pos=posold THEN Td:= 0.0; END;
    posold:=pos; StringToReal(str,pos,N);  IF pos=posold THEN N :=30.0; END;

    h := GlobalData.SetupData.RealTsamp;
    WITH TempData.Regulator DO
      MakePoly(0, 0.0, 0.0,0.0,0.0,S2);
      MakePoly(0, 0.0, 0.0,0.0,0.0,S3);
      IF Ti<=0.0 THEN
        IF ABS(Td)<1.0E-10 THEN              (* P-reg   *)
          MakePoly(0, 1.0, 1.0,0.0,0.0, R);
          MakePoly(0,  K , 1.0,0.0,0.0, T);
          MakePoly(0,  K , 1.0,0.0,0.0, S1);
          RegType := 'P  ';
        ELSE                                 (* PD-reg  *)
          MakePoly(1, 1.0, 1.0,-Td/(N*h+Td),0.0, R);
          MakePoly(1,  K , 1.0,-Td/(N*h+Td),0.0, T);
          MakePoly(1,  K , 1.0+Td*N/(N*h+Td), -(Td*N+Td)/(N*h+Td),0.0,S1);
          RegType := 'PD ';
        END;
      ELSE
        IF ABS(Td)<1.0E-10 THEN              (* PI-reg  *)
          MakePoly(1, 1.0, 1.0,-1.0,0.0,R);
          MakePoly(1,  K , 1.0,-1.0+h/Ti,0.0,T);
          S1 := T;
          RegType := 'PI ';
        ELSE                                 (* PID-reg *)
          MakePoly(2, 1.0, 1.0,-1.0-Td/(N*h+Td),Td/(N*h+Td),R);
          MakePoly(2,  K , 1.0,-1.0-Td/(N*h+Td),Td/(N*h+Td),p1);
          MakePoly(1, K*h/Ti,1.0,-Td/(N*h+Td),0.0,p2);
          MakePoly(2, K*N*Td/(N*h+Td), 1.0, -2.0,1.0,p3);
          Poladd(p1,p2,1.0E-8,T);
```

```
          Poladd(T,p3,1.0E-8,S1);
          RegType := 'PID';
        END;
      END;
    END;

  ELSIF CompareStr(cmd,'HIST') =0 THEN
    pos := 0;
    StringToReal(str,pos,valr);
    valc := TRUNC(valr/GlobalData.SetupData.RealTsamp+0.5);
    IF (5<valc) AND (valc< MaxStore-10) THEN (* -10 for safety *)
      WITH GlobalData.EstimData DO
        HistoryTime := valr;
        HistoryCard := valc;
      END;
    ELSE
      ErrorMessage('History too short or long');
    END;

  ELSIF CompareStr(cmd,'HELP')    =0 THEN
  ELSE  ErrorMessage('Error in Command');
  END;
  HideCursor;
  WriteRegulScreen;
  ShowCursor;
END KeybOnRegulMeny;

END MNYregul.
```

```
DEFINITION MODULE MNYsetup;

EXPORT QUALIFIED StartSetupMeny, MouseOnSetupMeny, KeybOnSetupMeny;

PROCEDURE StartSetupMeny;
PROCEDURE MouseOnSetupMeny(rect:CARDINAL);
PROCEDURE KeybOnSetupMeny(str:ARRAY OF CHAR);

END MNYsetup.
```

```
IMPLEMENTATION MODULE MNYsetup;

FROM Screend  IMPORT ScreenData, ErrorMessage, WriteMouseBox, WriteMenyLine;
FROM Globals  IMPORT SetMode, ModeType, PlotTypeType,
                     GlobalData, TempData, SetupDataType;
FROM PRCexec  IMPORT SetPlotWhen;
FROM Graphics IMPORT SetWindow,rectangle,point,
                     WriteString,color,SetFillColor,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;
                     (* buttontype   *)
FROM Opcom    IMPORT GetWord, WriteText, WriteInt, WriteReal;
FROM ConvReal IMPORT StringToReal;
FROM Strings  IMPORT Assign, CompareStr;
FROM NumberConversion IMPORT StringToCard;
(* FROM Debug IMPORT Write; *)

VAR
  SetupRect       : rectangle;

PROCEDURE WriteSetupScreen;
VAR pkt:point;
BEGIN              (* Write these also in *OnSetupMeny ?? ==> faster *)
  HideCursor;
  WriteMenyLine(' SAVE   ',' BETWE  ',' EVERY  ',' EVERY2 ',' EVERY4 ',
                ' ',' ',' ',' ',' DEMO ',blue);

  FillRectangle(ScreenData.Opcomhandle,SetupRect);
  SetTextColor(ScreenData.Opcomhandle,ScreenData.Textcolor);
  WriteText(0.3,11.3,'Setup Menu');

  (* write in opcom area *)
  WITH TempData.SetupData DO
    WITH ScreenData DO
      SetTextColor(Opcomhandle,Refcolor);
      WriteInt(2.0,10.0,' chanRef = ',chanRef);
      SetTextColor(Opcomhandle,Y1color);
      WriteInt(2.0, 9.0,' chanY1  = ',chanY1 );
      SetTextColor(Opcomhandle,Y2color);
      WriteInt(2.0, 8.0,' chanY2  = ',chanY2 );
      SetTextColor(Opcomhandle,Y3color);
      WriteInt(2.0, 7.0,' chanY3  = ',chanY3 );
      SetTextColor(Opcomhandle,Ucolor);
      WriteInt(2.0, 6.0,' chanU   = ',chanU  );

      SetTextColor(Opcomhandle,Textcolor);
      WriteInt(2.0, 5.0,' Number of Inputs (NUMINP) = ',NumberOfInputs);

      CASE PlotWhen OF
        PlotBetween:   WriteText(2.0,4.0,' Plot Y between samples     ')|
        PlotEvery:     WriteText(2.0,4.0,' Plot Y every sample        ')|
        PlotEveryTwo:  WriteText(2.0,4.0,' Plot Y every second sample ')|
        PlotEveryFour: WriteText(2.0,4.0,' Plot Y every fourth sample ')|
        Redraw:        WriteText(2.0,4.0,' ERROR Redraw not here      ');
      END;

      WriteReal(2.0,3.0,' Horizontal Plot Time (HPT) = ',HorizontalTime);
      WriteReal(2.0,2.0,' Sampling Interval (TSAMP)  = ',RealTsamp);

      WriteReal(4.15,1.0,'          ',Uhigh);
      WriteReal(2.0,1.0,' ULimits (ULIM) : ',Ulow);
    END;
    ShowCursor;
END WriteSetupScreen;

PROCEDURE StartSetupMeny;
BEGIN
  (* set opcom area *)
  SetupRect.loleft.v  :=  0.0;   SetupRect.loleft.h  :=  0.0;
  SetupRect.upright.v := 12.0;   SetupRect.upright.h := 10.0;
  WITH ScreenData DO
    SetWindow(Opcomhandle,SetupRect);
    SetFillColor(Opcomhandle,black);
  END;

  TempData.SetupData := GlobalData.SetupData;
  SetMode(Stop);
  WriteSetupScreen;
END StartSetupMeny;

PROCEDURE MouseOnSetupMeny(rect:CARDINAL);
BEGIN
  CASE rect OF
    10: GlobalData.SetupData := TempData.SetupData;
        SetPlotWhen(GlobalData.SetupData.PlotWhen);
        WriteText(0.0,0.1,' Data Saved ')|
    11: TempData.SetupData.PlotWhen := PlotBetween;
        WriteSetupScreen|
    12: TempData.SetupData.PlotWhen := PlotEvery;
        WriteSetupScreen|
    13: TempData.SetupData.PlotWhen := PlotEveryTwo;
        WriteSetupScreen|
    14: TempData.SetupData.PlotWhen := PlotEveryFour;
        WriteSetupScreen|
    19: (* demo setup *)
        WITH TempData.SetupData DO
          HorizontalTime := 20.0;
          chanRef:=0; chanY1:=1; chanY2:=2; chanY3:=3; chanU:=0;
          NumberOfInputs:=2;
          PlotWhen  :=  PlotEvery;
          Tsamp     :=  100; RealTsamp :=0.1;
          Ulow      := -1.0;
          Uhigh     :=  1.0;
        END;
        GlobalData.SetupData := TempData.SetupData;
        WriteSetupScreen;
        WriteText(0.0,0.1,' Demo Data Saved ');
  ELSE
    (* no action *)
  END;
END MouseOnSetupMeny;

PROCEDURE KeybOnSetupMeny(str:ARRAY OF CHAR);
VAR cmd1 : ARRAY [0..21] OF CHAR;
    valc : CARDINAL;
    valr,valr2 : REAL;
    pos  : CARDINAL;
    ok   : BOOLEAN;
BEGIN
  GetWord(str,cmd1);
  IF cmd1[0]='C' THEN      (* Read I/O Channel *)
    StringToCard(str,valc,ok);
```

```
    IF cmd1[4]='U' THEN
      ok:=ok AND (0<=valc) AND (valc<=1);
    ELSE
      ok:=ok AND (0<=valc) AND (valc<=3);
    END;

    IF ok THEN
      IF CompareStr(cmd1,'CHANREF') =0 THEN
        TempData.SetupData.chanRef :=valc;
      ELSIF CompareStr(cmd1,'CHANY1')  =0 THEN
        TempData.SetupData.chanY1  :=valc;
      ELSIF CompareStr(cmd1,'CHANY2')  =0 THEN
        TempData.SetupData.chanY2  :=valc;
      ELSIF CompareStr(cmd1,'CHANY3')  =0 THEN
        TempData.SetupData.chanY3  :=valc;
      ELSIF CompareStr(cmd1,'CHANU')   =0 THEN
        TempData.SetupData.chanU   :=valc;
      ELSE
        ErrorMessage('Unknown Command');
        ok:=FALSE;
      END;
    ELSE
      ErrorMessage('Chan out of range');
    END;
    IF ok THEN
      WriteSetupScreen;
    END;
  ELSE
    pos :=0;
    valr:=0.0;

    IF CompareStr(cmd1,'HPT')   =0 THEN
      StringToReal(str,pos,valr);
      IF (pos>0) AND (2.0<=valr) AND (valr<=100.0) THEN
        TempData.SetupData.HorizontalTime :=valr;
        WriteSetupScreen;
      ELSE
        ErrorMessage('HPT out of range');
      END;
    ELSIF CompareStr(cmd1,'TSAMP')   =0 THEN
      StringToReal(str,pos,valr);
      valc:=10*TRUNC(100.0*valr+0.5);
      IF (pos>0) AND (9<valc) AND (valc<10001) THEN  (* 14 OK ?! *)
        WITH TempData.SetupData DO
          Tsamp :=valc;
          Dt1:=Tsamp DIV 4;
          Dt2:=(Tsamp-Dt1) DIV 3;
          Dt3:=(Tsamp-Dt1-Dt2) DIV 2;
          RealTsamp:=FLOAT(Tsamp)/1000.0;
        END;
        WriteSetupScreen;
      ELSE
        ErrorMessage('Tsamp out of range');
      END;
    ELSIF CompareStr(cmd1,'ULIM')    =0 THEN
      valr:=-2.0; valr2:= 2.0;
      StringToReal(str,pos,valr);
      StringToReal(str,pos,valr2);
      IF (-1.0<=valr)  AND (valr<=1.0)  AND
         (-1.0<=valr2) AND (valr2<=1.0) AND (valr<valr2) THEN
        TempData.SetupData.Ulow  := valr;
        TempData.SetupData.Uhigh := valr2;
        WriteSetupScreen;
      ELSE
        ErrorMessage('-1 <= Umin < Umax <= 1');
      END;
    ELSIF CompareStr(cmd1,'NUMINP')  =0 THEN
      StringToCard(str,valc,ok);
      ok:=ok AND (1<=valc) AND (valc<=3);
      IF ok THEN
        TempData.SetupData.NumberOfInputs := valc;
        WriteSetupScreen;
      ELSE
        ErrorMessage(' 1 <= NUMINP <= 3 ');
      END;
    ELSIF CompareStr(cmd1,'HELP')    =0 THEN
    ELSE  ErrorMessage('Error in Command');
    END;
  END;
END KeybOnSetupMeny;

END MNYsetup.
```

```
DEFINITION MODULE Opcom;

FROM Pcalc    IMPORT cpoly;

EXPORT QUALIFIED WriteReal, WriteInt, WriteText,
                 WritePolynomCoeff, ReadPolynomCoeff, GetWord,
                 ReadContinuousSpecs,
                 IdentifyCommand, CommandMatrix;

CONST
  CommandNumberMax = 20;
TYPE
  CommandMatrix = ARRAY [1..CommandNumberMax],[0..12] OF CHAR;

PROCEDURE WriteReal(h,v:REAL;txt:ARRAY OF CHAR;val:REAL);
PROCEDURE WriteInt(h,v:REAL;txt:ARRAY OF CHAR;val:INTEGER);
PROCEDURE WriteText(h,v:REAL;txt:ARRAY OF CHAR);

PROCEDURE WritePolynomCoeff(h,v:REAL;txt:ARRAY OF CHAR;p:cpoly);
PROCEDURE ReadPolynomCoeff(str:ARRAY OF CHAR;VAR p:cpoly);
PROCEDURE ReadContinuousSpecs(str:ARRAY OF CHAR; VAR p:cpoly);

PROCEDURE GetWord(VAR str,com:ARRAY OF CHAR);

PROCEDURE IdentifyCommand(str:ARRAY OF CHAR; com:CommandMatrix):CARDINAL;

END Opcom.
```

```
IMPLEMENTATION MODULE Opcom;

FROM Screend  IMPORT ScreenData, ErrorMessage;
FROM Globals  IMPORT GlobalData, SetupDataType;
FROM Pcalc    IMPORT mdegree, coeffvector, cpoly, Polmul;
FROM Graphics IMPORT SetWindow,rectangle, SetFillColor,
                     point, WriteString, color, FillRectangle, SetTextColor;
FROM Strings  IMPORT Concat, Assign, Length;
FROM ConvReal IMPORT StringToReal, RealToString;
FROM NumberConversion IMPORT IntToString;
FROM MathLib  IMPORT exp, cos, sqrt;

PROCEDURE GetWord(VAR str,com:ARRAY OF CHAR);
VAR len, is, ic : CARDINAL;
BEGIN
  len    := Length(str);
  is     := 0;

  WHILE (is<len) AND (str[is]=' ') DO  (* Skip Blanks *)
    is:=is+1;
  END;

  ic := 0;
  WHILE (is<len) AND (str[is]<>' ') DO  (* Get Word *)
    com[ic] := CAP( str[is] );
    str[is] := ' ';
    is      := is+1;
    ic      := ic+1;
  END;
  IF ic<=HIGH(com) THEN
    com[ic] := CHR(0);
  END;
END GetWord;

PROCEDURE WriteReal(h,v:REAL;txt:ARRAY OF CHAR;val:REAL);
VAR pkt:point;
    num:ARRAY[0..40] OF CHAR;
BEGIN
  pkt.h:=h; pkt.v:=v;
  RealToString(val,num,7); (* maybe not correct   - parameter ?? *)
  Concat(txt,num,num);
  WriteString(ScreenData.Opcomhandle,pkt,num);
END WriteReal;

PROCEDURE WriteInt(h,v:REAL;txt:ARRAY OF CHAR;val:INTEGER);
VAR pkt:point;
    num:ARRAY[0..40] OF CHAR;
BEGIN
  pkt.h:=h; pkt.v:=v;
  IntToString(val,num,4); (* maybe not correct   - parameter ?? *)
  Concat(txt,num,num);
  WriteString(ScreenData.Opcomhandle,pkt,num);
END WriteInt;

PROCEDURE WriteText(h,v:REAL;txt:ARRAY OF CHAR);
VAR pkt:point;
BEGIN
  pkt.h:=h; pkt.v:=v;
  WriteString(ScreenData.Opcomhandle,pkt,txt);
END WriteText;

PROCEDURE WritePolynomCoeff(h,v:REAL;txt:ARRAY OF CHAR;p:cpoly);
VAR pkt : point;
    num : ARRAY[0..10] OF CHAR;
    str : ARRAY[0..70] OF CHAR;
    i   : CARDINAL;
BEGIN
  pkt.h:=h; pkt.v:=v;
  Assign(txt,str);
  WITH p DO
    FOR i:=0 TO degree DO
      RealToString(coeffs[i],num,7); (* maybe not correct   - parameter ?? *)
      Concat(str,' ',str);
      Concat(str,num,str);
    END;
  END;
  WriteString(ScreenData.Opcomhandle,pkt,str);
END WritePolynomCoeff;

PROCEDURE ReadPolynomCoeff(str:ARRAY OF CHAR;VAR p:cpoly);
VAR i,pos,posold:CARDINAL;
    ptmp:cpoly;
BEGIN
  i:=0;
  pos:=0;
  LOOP
    posold:=pos;
    StringToReal(str,pos,ptmp.coeffs[i]);
    IF pos=posold THEN
      IF i=0 THEN
        ptmp.degree:=0;
      ELSE
        ptmp.degree:=i-1;
      END;
      EXIT;
    ELSIF i=mdegree THEN
      ptmp.degree:=mdegree;
      EXIT;
    END;
    i:=i+1;
  END;

  FOR i:=ptmp.degree+1 TO mdegree DO
    ptmp.coeffs[i]:=0.0;
  END;

  IF ptmp.degree<=5 THEN
    p:=ptmp;
  ELSE
    ErrorMessage('Polynomial degree <= 5');
  END;
END ReadPolynomCoeff;

PROCEDURE ReadContinuousSpecs(str:ARRAY OF CHAR; VAR p:cpoly);
VAR z,w,a,h        : REAL;
    pos,posold,i   : CARDINAL;
    p1,p2          : cpoly;
BEGIN
  h      := GlobalData.SetupData.RealTsamp;
  pos    := 0;
  posold := 0;
  StringToReal(str,pos,w);
```

```
  IF (pos=posold) OR (w<0.0) THEN
    ErrorMessage('Out of range: w > 0');
    RETURN
  END;

  posold := pos;
  StringToReal(str,pos,z);
  IF pos=posold THEN (* first order polynomial *)
    WITH p DO
      degree:=1;
      coeffs[0] := 1.0;
      coeffs[1] := -exp(-w*h);
      FOR i:=2 TO mdegree DO coeffs[i]:=0.0; END;
    END;
    RETURN
  END;

  IF (0.0>z) OR (z>1.0) THEN
    ErrorMessage('Out of range: 0 < z <= 1');
    RETURN
  END;

  posold := pos;
  StringToReal(str,pos,a);
  IF pos=posold THEN (* second order polynomial *)
    WITH p DO
      degree:=2;
      coeffs[0] := 1.0;
      coeffs[1] := -2.0*exp(-z*w*h)*cos(w*h*sqrt(1.0-z*z));
      coeffs[2] := exp(-2.0*z*w*h);
      FOR i:=3 TO mdegree DO coeffs[i]:=0.0; END;
    END;
    RETURN
  END;

  (* third order polynomial *)
  WITH p2 DO
    degree:=2;
    coeffs[0] := 1.0;
    coeffs[1] := -2.0*exp(-z*w*h)*cos(w*h*sqrt(1.0-z*z));
    coeffs[2] := exp(-2.0*z*w*h);
    FOR i:=3 TO mdegree DO coeffs[i]:=0.0; END;
  END;
  WITH p1 DO
    degree:=1;
    coeffs[0] := 1.0;
    coeffs[1] := -exp(-a*w*h);
    FOR i:=2 TO mdegree DO coeffs[i]:=0.0; END;
  END;
  Polmul(p1,p2,1.0E-8,p);
END ReadContinuousSpecs;

PROCEDURE IdentifyCommand(str:ARRAY OF CHAR; com:CommandMatrix):CARDINAL;
VAR ic,is, index : CARDINAL;
    match        : BOOLEAN;
BEGIN
  index := 0;
  FOR ic:=1 TO CommandNumberMax DO
    match := TRUE;
    FOR is:=1 TO Length(str) DO
      match:=match AND (str[is-1]=com[ic,is]);
    END;
    IF match THEN
      IF index=0 THEN
        index := ic;
      ELSE
        ErrorMessage('Ambiguos Command');
        RETURN 0;
      END;
    END;
  END;

  IF index=0 THEN
    ErrorMessage('Unknown Command');
  END;

  RETURN index;
END IdentifyCommand;

END Opcom.
```

```
DEFINITION MODULE Pcalc;

EXPORT QUALIFIED mdegree, polytypes, complex, coeffvector, cpoly, zpoly, poly,
                 Polnorm, Normalize, Cutpoly,
                 Poladd, Poldiv, Polmul, Polsub,
                 Gcd, Diophantine, Sfactorize, FixDegree;

CONST mdegree = 10;

TYPE
        polytypes       = (bycoeffs, byzeros);
        complex         = RECORD
                            re, im : REAL;
                          END;
        coeffvector     = ARRAY [0..mdegree] OF REAL;
        cpoly           = RECORD
                            degree        : CARDINAL;
                            coeffs        : coeffvector;
                          END;
        zpoly           = RECORD
                            degree        : INTEGER;
                            leadingcoeff  : REAL;
                            zeros         : ARRAY [1..mdegree] OF complex;
                          END;
        poly            = RECORD
                            CASE representation : polytypes OF
                              bycoeffs : cp : cpoly|
                              byzeros  : zp : zpoly
                            END;
                          END;

PROCEDURE Polnorm(p: cpoly): REAL;
PROCEDURE Normalize(reps: REAL; VAR p: cpoly);
PROCEDURE Cutpoly(reps: REAL; VAR p: cpoly);
PROCEDURE Poladd(term1, term2: cpoly; reps: REAL; VAR sum: cpoly);
PROCEDURE Poldiv(numerator, denominator: cpoly; reps: REAL;
                 VAR quotient, reminder: cpoly);
PROCEDURE Polmul(factor1, factor2: cpoly; reps: REAL; VAR product: cpoly);
PROCEDURE Polsub(term1, term2: cpoly; reps: REAL; VAR difference: cpoly);
PROCEDURE Gcd(a, b: cpoly; reps: REAL; VAR g, x, y, u, v: cpoly);
PROCEDURE Diophantine(a, b, c :cpoly; VAR xx,yy:cpoly);
PROCEDURE Sfactorize(b: cpoly; VAR a: cpoly);
PROCEDURE FixDegree(VAR p:cpoly; deg:CARDINAL);

END Pcalc.
```

```
IMPLEMENTATION MODULE Pcalc;
(*-----------------------------------------------------------------------------
 Subset of PCALC.PAS.                                              87.05.05

 Tranlated into MODULA by Michael Lundh.
-----------------------------------------------------------------------------*)
FROM Screend IMPORT ErrorMessage;

CONST
      relative = 1.0E-8;

PROCEDURE Polnorm(p: cpoly): REAL;
VAR     i               : CARDINAL;
        r, s            : REAL;
BEGIN
  r:=0.0;
  WITH p DO
    FOR i:=0 TO degree DO
      s:=ABS(coeffs[i]);
      IF s>r THEN
        r:=s;
      END;
    END;
  END;
  RETURN r;
END Polnorm;

PROCEDURE Normalize(reps: REAL; VAR p: cpoly);
VAR     aeps            : REAL;
        align, i        : CARDINAL;
BEGIN
  aeps:=reps*Polnorm(p);
  align:=0;
  WITH p DO
    WHILE (ABS(coeffs[align])<=aeps) AND (align<=degree) DO
      align:=align+1;
    END;
    IF degree>=align THEN
      degree:=degree-align;
      FOR i:=0 TO degree DO
        coeffs[i]:=coeffs[i+align];
      END;
    ELSE
      degree:=0;
      coeffs[0]:=0.0;
    END;
  END;
END Normalize;

PROCEDURE Cutpoly(reps: REAL; VAR p: cpoly);
BEGIN
  Normalize(reps,p);
  WITH p DO
    IF degree<0 THEN
      degree:=0;
      coeffs[0]:=0.0;
    END;
  END;
END Cutpoly;

PROCEDURE Poladd(term1, term2: cpoly; reps: REAL; VAR sum: cpoly);
  PROCEDURE Assym(a, b: cpoly; align: CARDINAL);
  VAR     i               : CARDINAL;
  BEGIN
    sum:=a;
    WITH sum DO
      FOR i:=0 TO b.degree DO
        coeffs[i+align]:=coeffs[i+align]+b.coeffs[i];
      END;
      FOR i:=b.degree+align+1 TO mdegree DO  (* ML add 861215 *)
        coeffs[i]:=0.0;
      END;
    END;
  END Assym;

BEGIN
  IF term1.degree>term2.degree THEN
    Assym(term1,term2, term1.degree-term2.degree);
  ELSE
    Assym(term2,term1, term2.degree-term1.degree);
  END;
  Cutpoly(reps,sum);
END Poladd;

PROCEDURE Poldiv(numerator, denominator: cpoly; reps: REAL;
                 VAR quotient, remainder: cpoly);
VAR     i, j            : CARDINAL;
BEGIN
  remainder:=numerator;
  WITH quotient DO
    FOR i:=0 TO mdegree DO
      coeffs[i]:=0.0;
    END;
    IF numerator.degree<denominator.degree THEN
      degree:=0;
      RETURN;
    END;
    degree:=numerator.degree-denominator.degree;
    FOR j:=0 TO degree DO
      coeffs[j]:=remainder.coeffs[0]/denominator.coeffs[0];
      FOR i:=0 TO denominator.degree DO
        remainder.coeffs[i]:= remainder.coeffs[i]
                              -coeffs[j]*denominator.coeffs[i];
      END;
      remainder.coeffs[0]:=0.0;
      Cutpoly(reps,remainder);
    END;
  END;
  Cutpoly(reps,quotient);
  Cutpoly(reps,remainder);
END Poldiv;

PROCEDURE Polmul(factor1, factor2: cpoly; reps: REAL; VAR product: cpoly);
VAR     i, j            : CARDINAL;
BEGIN
  WITH product DO
    degree:=factor1.degree+factor2.degree;
    FOR i:=0 TO mdegree DO (* ML861215 degree => mdegree *)
      coeffs[i]:=0.0;
    END;
    FOR i:=0 TO factor1.degree DO
```

```
      FOR j:=0 TO factor2.degree DO
        coeffs[i+j]:=coeffs[i+j]+factor1.coeffs[i]*factor2.coeffs[j];
      END;
    END;
  END;
  Cutpoly(reps,product);
END Polmul;

PROCEDURE Polsub(term1, term2: cpoly; reps: REAL; VAR difference: cpoly);
VAR     i                : CARDINAL;
BEGIN
  WITH term2 DO
    FOR i:=0 TO degree DO
      coeffs[i]:=-coeffs[i];
    END;
  END;
  Poladd(term1,term2,reps,difference);
END Polsub;

PROCEDURE Gcd(a, b: cpoly; reps: REAL; VAR g, x, y, u, v: cpoly);
VAR     swapped            : BOOLEAN;
        h, hold, uold,vold : cpoly;
        nothing            : REAL;

  PROCEDURE Polydef(r: REAL; VAR p: cpoly);
  VAR i:CARDINAL;
  BEGIN
    WITH p DO
      degree:=0;
      coeffs[0]:=r;
      FOR i:=1 TO mdegree DO  (* ML861215  add *)
        coeffs[i]:=0.0;
      END;
    END;
  END Polydef;

  PROCEDURE Swapconditionally(VAR p1, p2: cpoly);
  VAR     p : cpoly;
  BEGIN
    IF swapped THEN
      p:=p1;
      p1:=p2;
      p2:=p;
    END;
  END Swapconditionally;

BEGIN
  swapped:=b.degree>a.degree;
  Polydef(1.0,x);
  Polydef(0.0,y);
  Polydef(0.0,u);
  Polydef(1.0,v);
  Swapconditionally(a,b);
  Swapconditionally(x,y);
  Swapconditionally(u,v);
  g:=a;
  h:=b;
  nothing:=reps*Polnorm(g);
  WHILE Polnorm(h)>nothing DO
    hold:=h;
    uold:=u;
    vold:=v;
    Poldiv(g,h,reps,g,h);
    Polmul(g,u,reps,u);
    Polsub(x,u,reps,u);
    Polmul(g,v,reps,v);
    Polsub(y,v,reps,v);
    g:=hold;
    x:=uold;
    y:=vold;
  END;
  Swapconditionally(x,y);
  Swapconditionally(u,v);
END Gcd;

PROCEDURE Diophantine(a, b, c :cpoly; VAR xx,yy:cpoly);
VAR     g, x, y, u, v   : cpoly;
        nothing         : REAL;
BEGIN
  Gcd(a,b,relative,g,x,y,u,v);
  nothing:=relative*Polnorm(c);
  IF (g.degree>c.degree) OR (Polnorm(g)<nothing) THEN
    ErrorMessage('No solution 1');
  END;
  Poldiv(c,g,relative,g,c);
  IF Polnorm(c)>nothing THEN
    ErrorMessage('No solution 2');
  END;
  Polmul(g,y,relative,b);
  Poldiv(b,v,relative,a,b);
  Polmul(a,u,relative,a);
  Polmul(g,x,relative,g);
  Polsub(g,a,relative,a);
  yy:=b;
  xx:=a;
END Diophantine;

PROCEDURE Sfactorize(b: cpoly; VAR a: cpoly);
CONST
  maxiter = 20;
VAR
  i, iter, j, n   : CARDINAL;
  alfa, beta, cr,
  il, s, sq, x    : coeffvector;
  p, sr           : ARRAY [0..mdegree] OF coeffvector;

BEGIN
  n:=b.degree DIV 2;                        (*Initialize a*)
  WITH a DO
    degree:=n;
    coeffs[0]:=1.0;
    FOR i:=1 TO n DO
      coeffs[i]:=0.0;
    END;

    FOR iter:=1 TO maxiter DO               (*Iteration loop*)
      sq[0]:=b.coeffs[0]/coeffs[0];
      FOR i:=1 TO n-1 DO
        sq[i]:=b.coeffs[i];
        FOR j:=1 TO i DO
          sq[i]:=sq[i]-sq[i-j]*coeffs[j];
        END;
```

```
      sq[i]:=sq[i]/coeffs[0];
    END;
    FOR i:=0 TO n-1 DO
      s[i]:=b.coeffs[n-i];
      FOR j:=0 TO n-i-1 DO
        s[i]:=s[i]-sq[i+j]*coeffs[n-j];
      END;
    END;
    s[n]:=b.coeffs[0];

    FOR i:=0 TO n DO                    (*Table*)
      p[0,i]:=coeffs[i];
      sr[0,i]:=s[i];
    END;
    FOR i:=0 TO n-1 DO
      FOR j:=0 TO n-i DO
        cr[j]:=p[i,n-i-j];
      END;
      alfa[i]:=p[i,n-i]/cr[n-i];
      beta[i]:=sr[i,n-i]/cr[n-i];
      FOR j:=0 TO n-i-1 DO
        p[i+1,j]:=p[i,j]-alfa[i]*cr[j];
        sr[i+1,j]:=sr[i,j]-beta[i]*cr[j];
      END;
    END;
    beta[n]:=sr[n,0]/p[n,0];

    FOR i:=0 TO n DO                     (*Integral*)
      beta[i]:=beta[i]/p[0,0];
    END;
    il[0]:=beta[n];
    FOR i:=1 TO n DO
      il[i]:=beta[n-i];
      FOR j:=0 TO i-1 DO
        il[i]:=il[i]-p[n-i,i-j]*il[j]/p[n-i,0];
      END;
    END;

    x[0]:=coeffs[0]*il[0];               (*Solution*)
    FOR i:=1 TO n DO
      x[i]:=coeffs[i]*il[0];
      FOR j:=1 TO i DO
        x[i]:=x[i]+2.0*coeffs[i-j]*il[j];
      END;
    END;
    FOR i:=0 TO n DO
      coeffs[i]:=0.5*(coeffs[i]+x[i]);
    END;
  END;
 END;
END Sfactorize;

PROCEDURE FixDegree(VAR p:cpoly; deg:CARDINAL);
(* Change a polynomial so that it has a specified degree *)
VAR i:CARDINAL;
BEGIN
  IF (p.degree<deg) AND (Polnorm(p)>0.0) THEN
    FOR i:=p.degree TO 0 BY -1 DO
      p.coeffs[i+deg-p.degree] := p.coeffs[i];
    END;
    FOR i:=deg-p.degree-1 TO 0 BY -1 DO
      p.coeffs[i] := 0.0;
    END;
    p.degree := deg;
  END;
END FixDegree;

END Pcalc.
```

```
DEFINITION MODULE PRCexec;

FROM Globals IMPORT PlotTypeType;

EXPORT QUALIFIED SetPlotWhen, InitExecute, Execute, Update;

(* initial *) PROCEDURE InitExecute;
(* process *) PROCEDURE Execute;
(* process *) PROCEDURE Update;

PROCEDURE SetPlotWhen(pw:PlotTypeType);

END PRCexec.
```

```
IMPLEMENTATION MODULE PRCexec;

FROM Kernel   IMPORT WaitUntil, GetTime, SetPriority,
                     Event, InitEvent, Await, Cause,
                     Semaphore, InitSem, Wait, Signal, Time, IncTime;
FROM Globals  IMPORT free, busy, PlotTypeType, SetupDataType, RefSignalType,
                     SignalTypeType, SendActualData, EstAdaptType,
                     ActualDataType, ModeType, GetModes, SetMode,
                     GlobalData, PlottingType;
FROM Regulmon IMPORT NewControlSignal, UpdateRegulatorState;
FROM RLS      IMPORT Estimation;
FROM Adaptive IMPORT AdaptiveDesign;
FROM AnalogIO IMPORT ADIn,DAOut;
(* FROM DProc    IMPORT ADIn,DAOut; *)
FROM MathLib  IMPORT sin;

VAR
  ActualData       : ActualDataType;
  RegSyncSemaphore : Semaphore;
  EstAdapt         : EstAdaptType;

PROCEDURE SetPlotWhen(pw:PlotTypeType);
BEGIN
  ActualData.PlotWhen:=pw;
END SetPlotWhen;

(* ------------------------------------------------------------------ *)

PROCEDURE ADinputOFy(VAR y1,y2,y3:REAL);
BEGIN
  WITH GlobalData.SetupData DO
    CASE NumberOfInputs OF
      1: y1 := ADIn(chanY1);  y2 := 0.0;           y3 := 0.0|
      2: y1 := ADIn(chanY1);  y2 := ADIn(chanY2);  y3 := 0.0|
      3: y1 := ADIn(chanY1);  y2 := ADIn(chanY2);  y3 := ADIn(chanY3);
    END;
  END;
END ADinputOFy;

PROCEDURE GetRefValue():REAL;
CONST Pi = 3.14159;
VAR r,tip:REAL;
BEGIN
  WITH GlobalData.RefSignal DO
    WITH GlobalData.SetupData DO
      TimeInPeriod := TimeInPeriod + RealTsamp;
      CASE SignalType OF
        external: r := ADIn(chanRef);
                  TimeInPeriod:=0.0|
        square  : IF TimeInPeriod<0.5*Period THEN
                    r := Mean+Amplitude;
                  ELSIF TimeInPeriod<Period THEN
                    r := Mean-Amplitude;
                  ELSE
                    TimeInPeriod := TimeInPeriod - Period;
                    r := Mean+Amplitude;
                  END |
        triangle: IF TimeInPeriod>Period THEN
                    TimeInPeriod := TimeInPeriod - Period;
                  END;
                  IF TimeInPeriod<0.25*Period THEN
                    r := Mean+4.0*Amplitude*TimeInPeriod/Period;
                  ELSIF TimeInPeriod<0.75*Period THEN
                    r := Mean+4.0*Amplitude*(0.5-TimeInPeriod/Period);
                  ELSE
                    r := Mean+4.0*Amplitude*(TimeInPeriod/Period-1.0);
                  END |
        sine    : IF TimeInPeriod>Period THEN
                    TimeInPeriod := TimeInPeriod - Period;
                  END;
                  r := Mean+Amplitude*sin(2.0*Pi*TimeInPeriod/Period) |
        step    : IF TimeInPeriod<0.05*HorizontalTime THEN
                    r:=0.0;
                  ELSIF TimeInPeriod+RealTsamp<HorizontalTime THEN
                    r := Amplitude;
                  ELSE
                    r:=0.0;
                    TimeInPeriod := 0.0;
                    SetMode(Zero);
                  END|
        ramp    : IF TimeInPeriod<0.05*HorizontalTime THEN
                    r:=0.0;
                  ELSIF TimeInPeriod+RealTsamp<HorizontalTime THEN
                    r := Amplitude*TimeInPeriod/HorizontalTime;
                  ELSE
                    r:=0.0;
                    TimeInPeriod := 0.0;
                    SetMode(Zero);
                  END;
      END;
    END;
    RETURN r;
  END;
END GetRefValue;

(* ------------------------------------------------------------------ *)

(* initial *) PROCEDURE InitExecute;
BEGIN
  InitSem(RegSyncSemaphore,busy);

  WITH ActualData DO
    PlotWhen := GlobalData.SetupData.PlotWhen;
    y1t0 := 0.0; y2t0 := 0.0; y3t0 := 0.0;  ut0 := 0.0;  reft0 := 0.0;
    y1t1 := 0.0; y2t1 := 0.0; y3t1 := 0.0;
    y1t2 := 0.0; y2t2 := 0.0; y3t2 := 0.0;
    y1t3 := 0.0; y2t3 := 0.0; y3t3 := 0.0;
  END;

END InitExecute;

(* Process *) PROCEDURE Execute;
VAR
  NextTime, SubTime : Time;
  PlotCounter       : CARDINAL;
  Mode              : ModeType;
  Plotting          : PlottingType;
BEGIN
  PlotCounter :=0;
  SetPriority(20);
  GetTime(NextTime);
```

```
LOOP
  SubTime:=NextTime;

  GetModes(Mode, Plotting, EstAdapt);
  WITH GlobalData.SetupData DO
    WITH ActualData DO
      CASE Mode OF
        Stop: ut0 := 0.0;
              GlobalData.RefSignal.TimeInPeriod := 0.0;
              DAOut(GlobalData.SetupData.chanU,ut0) |
        Zero: GlobalData.RefSignal.TimeInPeriod := 0.0;
              ADinputOFy(y1t0,y2t0,y3t0);
              ut0:=NewControlSignal(0.0,y1t0,y2t0,y3t0);
              DAOut(GlobalData.SetupData.chanU,ut0);
              UpdateRegulatorState|
        Run : reft0:=GetRefValue();
              ADinputOFy(y1t0,y2t0,y3t0);
              ut0:=NewControlSignal(reft0,y1t0,y2t0,y3t0);
              DAOut(GlobalData.SetupData.chanU,ut0);

              IF Plotting=PlotOn THEN
                CASE PlotWhen OF
                  PlotBetween   : Signal(RegSyncSemaphore);
                                    (* Start Update-process *)
                                  IncTime(SubTime,Dt1);
                                  WaitUntil(SubTime);
                                  ADinputOFy(y1t1,y2t1,y3t1);
                                  IncTime(SubTime,Dt2);
                                  WaitUntil(SubTime);
                                  ADinputOFy(y1t2,y2t2,y3t2);
                                  IncTime(SubTime,Dt3);
                                  WaitUntil(SubTime);
                                  ADinputOFy(y1t3,y2t3,y3t3);
                                  SendActualData(ActualData)|
                  PlotEvery     : CASE EstAdapt OF
                                    Off      : (* no action *) |
                                    Estimate : Estimation(ut0,y1t0)|
                                    Adaptive : AdaptiveDesign(ut0,y1t0,
                                               reft0,0.0);
                                  END;
                                  UpdateRegulatorState;
                                  SendActualData(ActualData)|
                  PlotEveryTwo  : CASE EstAdapt OF
                                    Off      : (* no action *) |
                                    Estimate : Estimation(ut0,y1t0)|
                                    Adaptive : AdaptiveDesign(ut0,y1t0,
                                               reft0,0.0);
                                  END;
                                  UpdateRegulatorState;
                                  PlotCounter:=(PlotCounter+1) MOD 2;
                                  IF PlotCounter = 0 THEN
                                    SendActualData(ActualData);
                                  END|
                  PlotEveryFour : CASE EstAdapt OF
                                    Off      : (* no action *) |
                                    Estimate : Estimation(ut0,y1t0)|
                                    Adaptive : AdaptiveDesign(ut0,y1t0,
                                               reft0,0.0);
                                   END;
                                  UpdateRegulatorState;
                                  PlotCounter:=(PlotCounter+1) MOD 4;
                                  IF PlotCounter = 0 THEN
                                    SendActualData(ActualData);
                                  END;
                END;
              ELSE
                CASE EstAdapt OF
                  Off      : (* no action *) |
                  Estimate : Estimation(ut0,y1t0)|
                  Adaptive : AdaptiveDesign(ut0,y1t0,reft0,0.0);
                END;
                UpdateRegulatorState;
              END;

          END;
          IncTime(NextTime,Tsamp);
          WaitUntil(NextTime);
        END;
      END;
    END;
  END Execute;

  (* ------------------------------------------------------------------- *)

  (* Process *) PROCEDURE Update;
  BEGIN
    SetPriority(30);

    LOOP
      Wait(RegSyncSemaphore);
      WITH ActualData DO
        CASE EstAdapt OF
          Off      : (* no action *) |
          Estimate : Estimation(ut0,y1t0)|
          Adaptive : AdaptiveDesign(ut0,y1t0,reft0,0.0);
        END;
      END;
      UpdateRegulatorState;
    END;
  END Update;

  END PRCexec.
```

```
DEFINITION MODULE PRCkeyb;

EXPORT QUALIFIED InitKeyboard, Keyboard;

(* initial *) PROCEDURE InitKeyboard;

(* Process *) PROCEDURE Keyboard;

END PRCkeyb.
```

```
IMPLEMENTATION MODULE PRCkeyb;

FROM Kernel   IMPORT SetPriority;
FROM Graphics IMPORT VirtualScreen,SetViewPort,SetWindow,SetFillColor,
                     handle,point,rectangle,color,
                     FillRectangle, SetTextColor, EraseChar,
                     WriteString, ReadString;
                     (* HideCursor,ShowCursor; *)
FROM Strings  IMPORT Concat, Length, CompareStr;
FROM Globals  IMPORT OpcomModeType, GetOpcomMode;
FROM Screend  IMPORT ScreenData, ErrorMessage, ResetErrorMessage;
FROM Opcom    IMPORT GetWord;
FROM MNYsetup IMPORT KeybOnSetupMeny;
FROM MNYmodel IMPORT KeybOnModelMeny;
FROM MNYdesgn IMPORT KeybOnDesignMeny;
FROM MNYregul IMPORT KeybOnRegulMeny;
FROM MNYrefs  IMPORT KeybOnRefSigMeny;
FROM MNYplot  IMPORT KeybOnPlotMeny;
(* FROM MNYAdapt IMPORT KeybOnAdaptMeny; *)
FROM MNYEstim IMPORT KeybOnEstimMeny;
FROM ConvReal IMPORT StringToReal; (*, RealToString;  *)

CONST
  ComRowMax =  2;
  StrMaxInd = 35;
TYPE
  StrType = ARRAY[0..StrMaxInd] OF CHAR;
VAR
  Commandbox     : rectangle;
  Commandpoint,  Promptpoint  : point;
  ComIndex : CARDINAL;
  blankcom : StrType;
  OldCommand : ARRAY [1..ComRowMax] OF StrType;

PROCEDURE Scroll(str:ARRAY OF CHAR);
VAR i:CARDINAL;
BEGIN
  IF ComIndex>0 THEN
    Concat(str,blankcom,OldCommand[ComIndex]);
    Promptpoint.v:=FLOAT(ComIndex);
    EraseChar(ScreenData.Commandhandle,Promptpoint,StrMaxInd+5);
    WriteString(ScreenData.Commandhandle,Promptpoint,OldCommand[ComIndex]);
    ComIndex:=ComIndex-1;
  ELSE
    FOR i:=ComRowMax TO 2 BY -1 DO
      OldCommand[i] := OldCommand[i-1];
      Promptpoint.v:=FLOAT(i);
      EraseChar(ScreenData.Commandhandle,Promptpoint,StrMaxInd+3);
      WriteString(ScreenData.Commandhandle,Promptpoint,OldCommand[i]);
    END;
    Concat(str,blankcom,OldCommand[1]);
    Promptpoint.v:=1.0;
    EraseChar(ScreenData.Commandhandle,Promptpoint,StrMaxInd+3);
    WriteString(ScreenData.Commandhandle,Promptpoint,OldCommand[1]);
  END;
END Scroll;

PROCEDURE GetCommand(prompt:ARRAY OF CHAR; VAR cmdstr:ARRAY OF CHAR);
VAR tmpstr : StrType;
BEGIN
  Promptpoint.v:=FLOAT(ComIndex);
  EraseChar(ScreenData.Commandhandle,Promptpoint,StrMaxInd+3);
  WriteString(ScreenData.Commandhandle,Promptpoint,prompt);
  Commandpoint.v:=FLOAT(ComIndex);
  ReadString(ScreenData.Commandhandle,Commandpoint,cmdstr);

  ResetErrorMessage;

  Concat(prompt,cmdstr,tmpstr);
  Scroll(tmpstr);
END GetCommand;

(* ------------------------------------------------------------------ *)
PROCEDURE InitKeyboard;
VAR r  : rectangle ;
    i  : CARDINAL;
BEGIN
  (* Area for Commandhandle *)
  Commandbox.loleft.h := 0.0;  Commandbox.upright.h := 2.25;
  Commandbox.loleft.v := 0.0;  Commandbox.upright.v := FLOAT(ComRowMax)+1.1;
  SetWindow(ScreenData.Commandhandle,Commandbox);
  SetFillColor(ScreenData.Commandhandle,black);
  FillRectangle(ScreenData.Commandhandle,Commandbox);
  SetTextColor(ScreenData.Commandhandle,green);
  Promptpoint.h:= 0.01;  Promptpoint.v:=FLOAT(ComRowMax);
  Commandpoint.h:= 0.10; Commandpoint.v:=FLOAT(ComRowMax);
  ComIndex := ComRowMax;
  FOR i:=0 TO StrMaxInd DO blankcom[i] := ' '; END;
END InitKeyboard;

(* Process *) PROCEDURE Keyboard;
VAR
  ComString : StrType ;
BEGIN
  SetPriority(60);
  LOOP
    GetCommand('->',ComString);

    IF    Length(ComString)>0  THEN
      CASE GetOpcomMode() OF
        SetupMode  : KeybOnSetupMeny(ComString)  |
        ModelMode  : KeybOnModelMeny(ComString)  |
        DesignMode : KeybOnDesignMeny(ComString) |
        RegulMode  : KeybOnRegulMeny(ComString)  |
        RefSigMode : KeybOnRefSigMeny(ComString) |
        PlotMode   : KeybOnPlotMeny(ComString)   |
       (*  AdaptMode  : KeybOnAdaptMeny(ComString) | *)
        EstimMode  : KeybOnEstimMeny(ComString)  ;
      END;
    END;
  END;
END Keyboard;

END PRCkeyb.
```

```
DEFINITION MODULE PRCmouse;

EXPORT QUALIFIED InitMouse, MouseHigh, MouseLow, WaitTheEnd;

(* initial *) PROCEDURE InitMouse;

(* process *) PROCEDURE MouseHigh;

(* process *) PROCEDURE MouseLow;

(* entry *)   PROCEDURE WaitTheEnd;

END PRCmouse.
```

```
IMPLEMENTATION MODULE PRCmouse;

FROM Kernel   IMPORT InitSem, Semaphore, Wait, Signal, SetPriority;
FROM Graphics IMPORT rectangle,point,buttonset,GetMouse,
                     SetTextColor,WriteString,EraseChar,color,
                     SetMouseRectangle, WaitMouseRectangle, ShowCursor;
FROM Screend  IMPORT ScreenData, ResetErrorMessage, WriteMouseBox,
                     ErrorMessage;
FROM Globals  IMPORT OpcomModeType, GetOpcomMode, SetOpcomMode, busy,
                     ModeType, PlottingType, EstAdaptType,
                     SetMode, SetPlotting, SetEstAdapt;
FROM MNYsetup IMPORT MouseOnSetupMeny,  StartSetupMeny;
FROM MNYmodel IMPORT MouseOnModelMeny,  StartModelMeny;
FROM MNYdesgn IMPORT MouseOnDesignMeny, StartDesignMeny;
FROM MNYregul IMPORT MouseOnRegulMeny,  StartRegulMeny;
FROM MNYrefs  IMPORT MouseOnRefSigMeny, StartRefSigMeny;
FROM MNYplot  IMPORT MouseOnPlotMeny,   StartPlotMeny;
(* FROM MNYAdapt IMPORT MouseOnAdaptMeny,  StartAdaptMeny; *)
FROM MNYEstim IMPORT MouseOnEstimMeny,  StartEstimMeny;
FROM AnalogIO IMPORT DAOut;

VAR
  TheEndSem     : Semaphore;
  MouseSync     : Semaphore;
  WhichRect     : CARDINAL;
  markedpoint   : point;
  pressedbutton : buttonset;

(* Entry *) PROCEDURE TheEnd;
BEGIN
  WriteMouseBox(2,9,' EXIT ',red);
  DAOut(0,0.0);
  DAOut(1,0.0);
  Signal(TheEndSem);
END TheEnd;

(* Entry *) PROCEDURE WaitTheEnd;
BEGIN
  Wait(TheEndSem);
END WaitTheEnd;

(* ------------------------------------------------ *)
PROCEDURE InitMouse;
VAR r : rectangle ;
    i : CARDINAL;
BEGIN
  InitSem(TheEndSem,busy);
  InitSem(MouseSync,busy);
  ShowCursor;
  StartSetupMeny;
END InitMouse;

(* process *) PROCEDURE MouseHigh;
BEGIN
  SetPriority(10);
  LOOP
    WhichRect := WaitMouseRectangle(ScreenData.Mousehandle);
    GetMouse(ScreenData.Opcomhandle,markedpoint,pressedbutton);

    IF WhichRect>=30 THEN
      IF GetOpcomMode() = SetupMode THEN
        ErrorMessage(' No Changes Now ');
      ELSE
        CASE WhichRect OF
          30: SetMode(Stop)|
          31: SetMode(Zero)|
          32: SetMode(Run)|
          33: SetEstAdapt(Estimate)|
          34: SetEstAdapt(Adaptive)|
          35: SetPlotting(PlotOn)|
          36: SetPlotting(PlotOff);
        ELSE (* no action *)
        END;
      END;
    ELSE
      Signal(MouseSync);
    END;
  END;
END MouseHigh;

(* process *) PROCEDURE MouseLow;
BEGIN
  SetPriority(50);
  LOOP
    Wait(MouseSync);
    ResetErrorMessage;
    CASE WhichRect OF
          0 : (* funktionsanrop med markedpoint *)        |
         20 : SetOpcomMode(SetupMode);   StartSetupMeny   |
         21 : SetOpcomMode(ModelMode);   StartModelMeny   |
         22 : SetOpcomMode(DesignMode);  StartDesignMeny  |
         23 : SetOpcomMode(RegulMode);   StartRegulMeny   |
         24 : SetOpcomMode(RefSigMode);  StartRefSigMeny  |
         25 : SetOpcomMode(PlotMode);    StartPlotMeny    |
         26 : SetOpcomMode(EstimMode);   StartEstimMeny   |
         27 : (* SetOpcomMode(AdaptMode);   StartAdaptMeny *) |
         28 : (* no action  - yet ! *) |
         29 : TheEnd |
      10..19 : CASE GetOpcomMode() OF
                 SetupMode  : MouseOnSetupMeny(WhichRect)   |
                 ModelMode  : MouseOnModelMeny(WhichRect)   |
                 DesignMode : MouseOnDesignMeny(WhichRect)  |
                 RegulMode  : MouseOnRegulMeny(WhichRect)   |
                 RefSigMode : MouseOnRefSigMeny(WhichRect)  |
                 PlotMode   : MouseOnPlotMeny(WhichRect)    |
                 AdaptMode  : (* MouseOnAdaptMeny(WhichRect) *) |
                 EstimMode  : MouseOnEstimMeny(WhichRect)   ;
               END;
    ELSE (* no action *)
    END;
  END;
END MouseLow;

END PRCmouse.
```

```
DEFINITION MODULE PRCplot;

EXPORT QUALIFIED InitPlotter, Plotter;

(* initial *) PROCEDURE InitPlotter;

(* process *) PROCEDURE Plotter;

END PRCplot.
```

```
IMPLEMENTATION MODULE PRCplot;

FROM Kernel   IMPORT SetPriority;
FROM Screend  IMPORT ScreenData;
FROM Globals  IMPORT OpcomModeType,GetOpcomMode, GlobalData, SetupDataType,
                     ActualDataType, PlotTypeType, ReceiveActualData;
FROM Graphics IMPORT handle,point,rectangle,color, SetWindow,SetTextColor,
                     HideCursor, ShowCursor, WriteString,SetLineColor,
                     SetFillColor, PolyLine,FillRectangle, DrawRectangle;
FROM ConvReal IMPORT RealToString;
FROM Strings  IMPORT Concat;
(* FROM Debug    IMPORT Write; *)

VAR
  PlotData                  : ActualDataType;
  Unitbox,Diffbox           : rectangle;
  y1po,y2po,y3po,upo,rpo,dpo : REAL; (* Old plotpoints *)
  pt1, pt2, pt3, pt4, pt4i  : REAL;
  ActualHorizontalTime,             (* Actual Horizontal Time *)
  LastPlotTime              : REAL; (* Last position on horizontal axis *)
  plotpkt8                  : ARRAY [0..8] OF point;
  plotpkt2                  : ARRAY [0..2] OF point;

(*-------------------------------------------------------------------------*)

PROCEDURE AssignPlotTime;  (* kan goras smartare *)
VAR tsamp, Dt1, Dt2, Dt3 : CARDINAL;
BEGIN
  tsamp := GlobalData.SetupData.Tsamp;
  Dt1:=tsamp DIV 4;
  Dt2:=(tsamp-Dt1) DIV 3;
  Dt3:=(tsamp-Dt1-Dt2) DIV 2;

  pt1:=FLOAT(Dt1)/1000.0;
  pt2:=FLOAT(Dt1+Dt2)/1000.0;
  pt3:=FLOAT(Dt1+Dt2+Dt3)/1000.0;
  pt4:=FLOAT(tsamp)/1000.0;
  pt4i:=pt4;
END AssignPlotTime;

PROCEDURE TimeBetween;
BEGIN
  plotpkt8[0].h := LastPlotTime;
  plotpkt8[1].h := LastPlotTime;
  plotpkt8[2].h := LastPlotTime + pt1;
  plotpkt8[3].h := LastPlotTime + pt1;
  plotpkt8[4].h := LastPlotTime + pt2;
  plotpkt8[5].h := LastPlotTime + pt2;
  plotpkt8[6].h := LastPlotTime + pt3;
  plotpkt8[7].h := LastPlotTime + pt3;
  plotpkt8[8].h := LastPlotTime + pt4;
END TimeBetween;

PROCEDURE PlBetween(h:handle; c:color; at0,at1,at2,at3:REAL; VAR aop:REAL);
BEGIN
  SetLineColor(h,c);
  plotpkt8[0].v := aop;
  plotpkt8[1].v := at0;
  plotpkt8[2].v := at0;
  plotpkt8[3].v := at1;
  plotpkt8[4].v := at1;
  plotpkt8[5].v := at2;
  plotpkt8[6].v := at2;
  plotpkt8[7].v := at3;
  plotpkt8[8].v := at3;
  PolyLine(h,plotpkt8,9);
  aop :=at3;
END PlBetween;

PROCEDURE TimeEvery;
BEGIN
  plotpkt2[0].h := LastPlotTime;
  plotpkt2[1].h := LastPlotTime;
  plotpkt2[2].h := LastPlotTime + pt4i;
END TimeEvery;

PROCEDURE PlEvery(han:handle; col:color; at0:REAL; VAR aop:REAL);
BEGIN
  SetLineColor(han,col);
  plotpkt2[0].v := aop;
  plotpkt2[1].v := at0;
  plotpkt2[2].v := at0;
  PolyLine(han,plotpkt2,3);
  aop :=at0;
END PlEvery;

(*-------------------------------------------------------------------------*)

PROCEDURE DrawSmallArea;
VAR txt:ARRAY [0..9] OF CHAR;
    Textpoint,Timepoint : point;
BEGIN
  LastPlotTime:=0.0;

  ActualHorizontalTime := GlobalData.SetupData.HorizontalTime;
  AssignPlotTime;
  RealToString(ActualHorizontalTime, txt , 5 );
  Concat(txt,' s',txt);
  Textpoint.h:=0.0;    Textpoint.v:= 0.48;  (* old 50 *)
  Timepoint.v:=-1.0;   Timepoint.h:=0.91*ActualHorizontalTime;
  Unitbox.upright.h := ActualHorizontalTime;

  WITH ScreenData DO
    SetWindow(SmallPlothandle,Unitbox);
    FillRectangle(SmallPlothandle,Unitbox);
    SetTextColor(SmallPlothandle,Refcolor);
    WriteString(SmallPlothandle,Textpoint,'    r');
    SetTextColor(SmallPlothandle,Y1color);
    WriteString(SmallPlothandle,Textpoint,' y1');
    SetTextColor(SmallPlothandle,Plotcolor);
    WriteString(SmallPlothandle,Timepoint,txt);
    SetLineColor(SmallPlothandle,Plotcolor);
    DrawRectangle(SmallPlothandle,Unitbox);
  END;
END DrawSmallArea;

PROCEDURE SmallPlotBetween;
BEGIN
  WITH PlotData DO
    IF (LastPlotTime + pt4)>ActualHorizontalTime THEN    (* Redraw *)
      DrawSmallArea;
    END;
```

```
    TimeEvery;
    TimeBetween;

    WITH ScreenData DO
      PlEvery(SmallPlothandle,Refcolor, reft0, rpo);
      PlBetween(SmallPlothandle,Ylcolor, ylt0,ylt1,ylt2,ylt3, ylpo);
    END;

    LastPlotTime:=LastPlotTime + pt4;
  END;
END SmallPlotBetween;

PROCEDURE SmallPlotEveryX;
BEGIN
  WITH PlotData DO
    IF (LastPlotTime + pt4i)>ActualHorizontalTime THEN    (* Redraw *)
      DrawSmallArea;
    END;
    TimeEvery;

    WITH ScreenData DO
      PlEvery(SmallPlothandle,Refcolor, reft0, rpo);
      PlEvery(SmallPlothandle,Ylcolor, ylt0, ylpo);
    END;

    LastPlotTime:=LastPlotTime + pt4i;
  END;
END SmallPlotEveryX;

(*-------------------------------------------------------------------------*)

PROCEDURE DrawBigArea;
VAR txt:ARRAY [0..9] OF CHAR;
    pli:ARRAY [0..1] OF point;
    Textpoint,Timepoint : point;

BEGIN
  LastPlotTime:=0.0;

  ActualHorizontalTime := GlobalData.SetupData.HorizontalTime;
  AssignPlotTime;
  RealToString(ActualHorizontalTime, txt , 5 );
  Concat(txt,' s',txt);
  Timepoint.v:=-1.0;   Timepoint.h:=0.91*ActualHorizontalTime;

  Unitbox.upright.h    := ActualHorizontalTime;
  Diffbox.upright.h    := ActualHorizontalTime;

  WITH ScreenData DO
    SetWindow(BigPlothandle,Unitbox);
    FillRectangle(BigPlothandle,Unitbox);
    Textpoint.h:=0.0; Textpoint.v:= 0.70;
    IF 3 IN GlobalData.PlotWhichCurve THEN
      SetTextColor(BigPlothandle,Y3color);
      WriteString(BigPlothandle,Textpoint,'              y3');
    END;
    IF 2 IN GlobalData.PlotWhichCurve THEN
      SetTextColor(BigPlothandle,Y2color);
      WriteString(BigPlothandle,Textpoint,'          y2');
    END;
    IF 1 IN GlobalData.PlotWhichCurve THEN
      SetTextColor(BigPlothandle,Y1color);
      WriteString(BigPlothandle,Textpoint,'        y1');
    END;
    SetTextColor(BigPlothandle,Refcolor);
    WriteString(BigPlothandle,Textpoint,'      r');

    SetTextColor(BigPlothandle,Plotcolor);
    WriteString(BigPlothandle,Textpoint,' 1.0');
    Textpoint.v:=-0.94; WriteString(BigPlothandle,Textpoint,'-1.0');

    SetLineColor(BigPlothandle,Plotcolor);
    DrawRectangle(BigPlothandle,Unitbox);

    SetWindow(DiffPlothandle,Diffbox);
    FillRectangle(DiffPlothandle,Diffbox);
    Textpoint.v:= 0.054;
    SetTextColor(DiffPlothandle,Ylcolor);
    WriteString(DiffPlothandle,Textpoint,'           -y1');
    SetTextColor(DiffPlothandle,Refcolor);
    WriteString(DiffPlothandle,Textpoint,'          r');
    SetTextColor(DiffPlothandle,Diffcolor);
    WriteString(DiffPlothandle,Textpoint,'      e =');
    SetTextColor(DiffPlothandle,Plotcolor);
    WriteString(DiffPlothandle,Textpoint,' 0.1');
    Textpoint.v:=-0.094; WriteString(DiffPlothandle,Textpoint,'-0.1');

    SetLineColor(DiffPlothandle,Plotcolor);
    DrawRectangle(DiffPlothandle,Diffbox);
    pli[0].h:=  0.0;                    pli[0].v:= 0.0;
    pli[1].h:=  ActualHorizontalTime;   pli[1].v:= 0.0;
    SetLineColor(DiffPlothandle,grey);
    PolyLine(DiffPlothandle,pli,2);

    SetWindow(UPlothandle,Unitbox);
    FillRectangle(UPlothandle,Unitbox);
    Textpoint.v:= 0.54;
    SetTextColor(UPlothandle,Ucolor);
    WriteString(UPlothandle,Textpoint,'      u');
    SetTextColor(UPlothandle,Plotcolor);
    WriteString(UPlothandle,Textpoint,' 1.0');
    Textpoint.v:=-0.94;  WriteString(UPlothandle,Textpoint,'-1.0');
    WriteString(UPlothandle,Timepoint,txt);
    SetLineColor(UPlothandle,Plotcolor);
    DrawRectangle(UPlothandle,Unitbox);
  END;
END DrawBigArea;

PROCEDURE BigPlotBetween;
BEGIN
  WITH PlotData DO
    IF (LastPlotTime + pt4)>ActualHorizontalTime THEN    (* Redraw *)
      DrawBigArea;
    END;
    TimeEvery;
    TimeBetween;

    WITH ScreenData DO
      PlEvery(BigPlothandle,Refcolor, reft0, rpo);
      PlEvery(UPlothandle,  Ucolor,   ut0,  upo);
      PlBetween(DiffPlothandle,Diffcolor,
                reft0-ylt0, reft0-ylt1, reft0-ylt2, reft0-ylt3 , dpo);
```

```
      IF 3 IN GlobalData.PlotWhichCurve THEN
        PlBetween(BigPlothandle,Y3color, y3t0,y3t1,y3t2,y3t3, y3po);
      END;
      IF 2 IN GlobalData.PlotWhichCurve THEN
        PlBetween(BigPlothandle,Y2color, y2t0,y2t1,y2t2,y2t3, y2po);
      END;
      IF 1 IN GlobalData.PlotWhichCurve THEN
        PlBetween(BigPlothandle,Y1color, y1t0,y1t1,y1t2,y1t3, y1po);
      END;
    END;

    LastPlotTime:=LastPlotTime + pt4;
  END;
END BigPlotBetween;

PROCEDURE BigPlotEveryX;
BEGIN
  WITH PlotData DO
    IF (LastPlotTime + pt4i)>ActualHorizontalTime THEN    (* Redraw *)
      DrawBigArea;
    END;
    TimeEvery;

    WITH ScreenData DO
      PlEvery(BigPlothandle,Refcolor, reft0, rpo);
      PlEvery(UPlothandle,  Ucolor,   ut0,  upo);
      PlEvery(DiffPlothandle,Diffcolor, reft0-y1t0, dpo);
      IF 3 IN GlobalData.PlotWhichCurve THEN
        PlEvery(BigPlothandle,Y3color, y3t0, y3po);
      END;
      IF 2 IN GlobalData.PlotWhichCurve THEN
        PlEvery(BigPlothandle,Y2color, y2t0, y2po);
      END;
      IF 1 IN GlobalData.PlotWhichCurve THEN
        PlEvery(BigPlothandle,Y1color, y1t0, y1po);
      END;
    END;

    LastPlotTime:=LastPlotTime + pt4i;
  END;
END BigPlotEveryX;

(*-------------------------------------------------------------------------*)

PROCEDURE InitPlotter;
VAR r : rectangle ;
BEGIN
  y1po:=0.0; y2po:=0.0; y3po:=0.0; upo:=0.0; rpo:=0.0; dpo:=0.0;

  WITH ScreenData DO
    SetFillColor(SmallPlothandle,black);
    SetFillColor(BigPlothandle,black);
    SetFillColor(DiffPlothandle,black);
    SetFillColor(UPlothandle,black);
  END;

  Unitbox.loleft.v  :=-1.0;  Unitbox.loleft.h  :=   0.0;
  Unitbox.upright.v := 1.0;
  Diffbox.loleft.v  :=-0.1;  Diffbox.loleft.h  :=   0.0;
  Diffbox.upright.v := 0.1;

  DrawSmallArea;
END InitPlotter;

(* Process *) PROCEDURE Plotter;
BEGIN
  SetPriority(40);

  LOOP
    ReceiveActualData(PlotData);
    CASE GetOpcomMode() OF
      PlotMode: CASE PlotData.PlotWhen OF
                  Redraw:        DrawBigArea    |
                  PlotBetween:   BigPlotBetween|
                  PlotEvery:     BigPlotEveryX |
                  PlotEveryTwo:  BigPlotEveryX ;  pt4i:=2.0*pt4|
                  PlotEveryFour: BigPlotEveryX ;  pt4i:=4.0*pt4;
                END;
      ELSE      CASE PlotData.PlotWhen OF
                  Redraw:        DrawSmallArea   |
                  PlotBetween:   SmallPlotBetween|
                  PlotEvery:     SmallPlotEveryX |
                  PlotEveryTwo:  SmallPlotEveryX ;  pt4i:=2.0*pt4|
                  PlotEveryFour: SmallPlotEveryX ;  pt4i:=4.0*pt4;
                END;
    END;
  END;
END Plotter;

END PRCplot.
```

```
DEFINITION MODULE Regulmon;

FROM Globals IMPORT RegulatorType;
FROM Pcalc   IMPORT cpoly;

EXPORT QUALIFIED InitRegulmonitor, NewControlSignal, UpdateRegulatorState,
                 NewAdaptiveRegulator, NewPolyInRegulator, GetPolyInRegulator;

(* initial *) PROCEDURE InitRegulmonitor;
(* monitor *) PROCEDURE NewControlSignal(r0,y01,y02,y03:REAL):REAL;
(* monitor *) PROCEDURE UpdateRegulatorState;

(* monitor *) PROCEDURE NewPolyInRegulator(rd:RegulatorType);
(* monitor *) PROCEDURE GetPolyInRegulator(VAR rd:RegulatorType);
(* monitor *) PROCEDURE NewAdaptiveRegulator(VAR rd:RegulatorType);

END Regulmon.
```

```
IMPLEMENTATION MODULE Regulmon;

FROM Kernel   IMPORT Semaphore, InitSem, Wait, Signal;
FROM Screend  IMPORT ErrorMessage;
FROM Pcalc    IMPORT mdegree,cpoly,Polnorm,FixDegree;
FROM Globals  IMPORT free, RegulatorType, SetupDataType, GlobalData,
                     AdaptDataType;

VAR
  RegulMonitor : RECORD
                   Mutex            : Semaphore;
                   RegData          : RegulatorType;
                   y1,y2,y3,ref,u,v : ARRAY[0..mdegree] OF REAL;
                 END;

  firstcall : BOOLEAN;

PROCEDURE LimitationOfU( u:REAL ):REAL;
BEGIN
  WITH GlobalData DO
    IF u < SetupData.Ulow*AdaptData.ulimit THEN
      RETURN SetupData.Ulow*AdaptData.ulimit;
    ELSIF u > SetupData.Uhigh*AdaptData.ulimit THEN
      RETURN SetupData.Uhigh*AdaptData.ulimit;
    ELSE
      RETURN u;
    END;
  END;
END LimitationOfU;

(* monitor *) PROCEDURE NewPolyInRegulator(rd:RegulatorType);
VAR i        : CARDINAL;
    DegreeOK : BOOLEAN;
    r0       : REAL;
BEGIN
  WITH rd DO                              (* Check degree of regulator *)
    DegreeOK := (S1.degree<=R.degree) AND (S2.degree<=R.degree) AND
                (S3.degree<=R.degree) AND (T.degree <=R.degree) AND
                (Ao.degree<=R.degree);
  END;
  IF NOT DegreeOK THEN
    ErrorMessage(' Non-Causal Regulator ');
  ELSE
    WITH RegulMonitor DO
      Wait(Mutex);
      WITH RegData DO
        RegulMonitor.RegData:=rd;

        FixDegree(S1,R.degree);
        FixDegree(S2,R.degree);
        FixDegree(S3,R.degree);
        FixDegree(T ,R.degree);

        FOR i:=Ao.degree+1 TO R.degree DO (* Fix degree of Ao *)
          Ao.coeffs[i]:=0.0;
        END;
        Ao.degree:=R.degree;

        r0 := R.coeffs[0];          (* make r[0]=1 *)
        FOR i:=0 TO R.degree DO
          R.coeffs[i]  := R.coeffs[i]  / r0;
          S1.coeffs[i] := S1.coeffs[i] / r0;
          S2.coeffs[i] := S2.coeffs[i] / r0;
          S3.coeffs[i] := S3.coeffs[i] / r0;
          T.coeffs[i]  := T.coeffs[i]  / r0;
        END;

        FOR i:=R.degree+1 TO mdegree DO
          y1[i] :=0.0;  y2[i]:=0.0;  y3[i]:=0.0;
          ref[i]:=0.0;  u[i] :=0.0;  v[i]:=0.0;
          R.coeffs[i]  := 0.0;
          S1.coeffs[i] := 0.0;
          S2.coeffs[i] := 0.0;
          S3.coeffs[i] := 0.0;
          T.coeffs[i]  := 0.0;
          Ao.coeffs[i] := 0.0;
        END;

        (* Bump less transfer - Perform new comp. for next sample *)
        v[0]:=0.0;
        FOR i:=1 TO R.degree DO
          v[0]:= v[0] - Ao.coeffs[i]*v[i]
                      + T.coeffs[i]*ref[i]
                      - S1.coeffs[i]*y1[i]
                      - S2.coeffs[i]*y2[i]
                      - S3.coeffs[i]*y3[i]
                      + ( Ao.coeffs[i] - R.coeffs[i] )*u[i];
        END;
      END;
      Signal(Mutex);
    END;
  END;
END NewPolyInRegulator;

(* monitor *) PROCEDURE GetPolyInRegulator(VAR rd:RegulatorType);
BEGIN
  Wait(RegulMonitor.Mutex);
  rd := RegulMonitor.RegData;
  Signal(RegulMonitor.Mutex);
END GetPolyInRegulator;

(* monitor *) PROCEDURE NewAdaptiveRegulator(VAR rd:RegulatorType);
(* Fast tranfer for adaptive case - no safety at all *)
BEGIN
  RegulMonitor.RegData.R  := rd.R;
  RegulMonitor.RegData.S1 := rd.S1;
  RegulMonitor.RegData.T  := rd.T;
END NewAdaptiveRegulator;

(* ------------------------------------------------------------------------ *)
PROCEDURE InitRegulmonitor;
VAR i : CARDINAL;
BEGIN
  WITH RegulMonitor DO       (* Init RegulMonitor *)
    IF firstcall THEN
      InitSem(Mutex, free);
      firstcall:=FALSE;
    END;

    FOR i:=0 TO mdegree DO
      y1[i]   := 0.0;
      y2[i]   := 0.0;
```

```
      y3[i]    := 0.0;
      ref[i]   := 0.0;
      u[i]     := 0.0;
      v[i]     := 0.0;
      RegData.R.coeffs[i]  := 0.0;
      RegData.S1.coeffs[i] := 0.0;
      RegData.S2.coeffs[i] := 0.0;
      RegData.S3.coeffs[i] := 0.0;
      RegData.T.coeffs[i]  := 0.0;
      RegData.Ao.coeffs[i] := 0.0;
    END;
    RegData.R.coeffs[0]  := 1.0;
    RegData.Ao.coeffs[0] := 1.0;
    RegData.R.degree  := 0;
    RegData.S1.degree := 0;
    RegData.S2.degree := 0;
    RegData.S3.degree := 0;
    RegData.T.degree  := 0;
    RegData.Ao.degree := 0;
    RegData.RegType   := 'none';
  END;
END InitRegulmonitor;

(* monitor *) PROCEDURE NewControlSignal(r0,y01,y02,y03:REAL):REAL;
VAR i : CARDINAL;
BEGIN
  WITH RegulMonitor DO
    Wait(Mutex);
    ref[0]:=r0;  y1[0]:=y01;  y2[0]:=y02;  y3[0]:=y03;

    WITH RegData DO
      v[0] := v[0]  + T.coeffs[0]*ref[0]
                    - S1.coeffs[0]*y1[0]
                    - S2.coeffs[0]*y2[0]
                    - S3.coeffs[0]*y3[0];
      u[0] := LimitationOfU( v[0] );
    END;
    Signal(Mutex);
    RETURN u[0];
  END;
END NewControlSignal;

(* monitor *) PROCEDURE UpdateRegulatorState;
VAR i : CARDINAL;
BEGIN
  WITH RegulMonitor DO
    Wait(Mutex);
    WITH RegData DO
      CASE R.degree OF (* straight code to increase speed *)
        0: v[0]:=0.0|
        1: y1[1]:=y1[0];     y2[1]:=y2[0];   y3[1]:=y3[0];
           ref[1]:=ref[0];   u[1]:=u[0];     v[1]:=v[0];
           v[0]:= T.coeffs[1]*ref[1]
                 -S1.coeffs[1]*y1[1] -S2.coeffs[1]*y2[1] -S3.coeffs[1]*y3[1]
                 + ( Ao.coeffs[1] - R.coeffs[1] )*u[1]  - Ao.coeffs[1]*v[1]|

        2: y1[2]:=y1[1];     y2[2]:=y2[1];   y3[2]:=y3[2];
           y1[1]:=y1[0];     y2[1]:=y2[0];   y3[1]:=y3[0];
           ref[2]:=ref[1];   u[2]:=u[1];     v[2]:=v[1];
           ref[1]:=ref[0];   u[1]:=u[0];     v[1]:=v[0];
           v[0]:= T.coeffs[1]*ref[1]
                 -S1.coeffs[1]*y1[1] -S2.coeffs[1]*y2[1] -S3.coeffs[1]*y3[1]
                 + ( Ao.coeffs[1] - R.coeffs[1] )*u[1]  - Ao.coeffs[1]*v[1]
                 + T.coeffs[2]*ref[2]
                 -S1.coeffs[2]*y1[2] -S2.coeffs[2]*y2[2] -S3.coeffs[2]*y3[2]
                 + ( Ao.coeffs[2] - R.coeffs[2] )*u[2]  - Ao.coeffs[2]*v[2]|

        3: y1[3]:=y1[2];     y2[3]:=y2[2];   y3[3]:=y3[2];
           y1[2]:=y1[1];     y2[2]:=y2[1];   y3[2]:=y3[2];
           y1[1]:=y1[0];     y2[1]:=y2[0];   y3[1]:=y3[0];
           ref[3]:=ref[2];   u[3]:=u[2];     v[3]:=v[2];
           ref[2]:=ref[1];   u[2]:=u[1];     v[2]:=v[1];
           ref[1]:=ref[0];   u[1]:=u[0];     v[1]:=v[0];
           v[0]:= T.coeffs[1]*ref[1]
                 -S1.coeffs[1]*y1[1] -S2.coeffs[1]*y2[1] -S3.coeffs[1]*y3[1]
                 + ( Ao.coeffs[1] - R.coeffs[1] )*u[1]  - Ao.coeffs[1]*v[1]
                 + T.coeffs[2]*ref[2]
                 -S1.coeffs[2]*y1[2] -S2.coeffs[2]*y2[2] -S3.coeffs[2]*y3[2]
                 + ( Ao.coeffs[2] - R.coeffs[2] )*u[2]  - Ao.coeffs[2]*v[2]
                 + T.coeffs[3]*ref[3]
                 -S1.coeffs[3]*y1[3] -S2.coeffs[3]*y2[3] -S3.coeffs[3]*y3[3]
                 + ( Ao.coeffs[3] - R.coeffs[3] )*u[3]  - Ao.coeffs[3]*v[3]|

        4: y1[4]:=y1[3];     y2[4]:=y2[3];   y3[4]:=y3[3];
           y1[3]:=y1[2];     y2[3]:=y2[2];   y3[3]:=y3[2];
           y1[2]:=y1[1];     y2[2]:=y2[1];   y3[2]:=y3[2];
           y1[1]:=y1[0];     y2[1]:=y2[0];   y3[1]:=y3[0];
           ref[4]:=ref[3];   u[4]:=u[3];     v[4]:=v[3];
           ref[3]:=ref[2];   u[3]:=u[2];     v[3]:=v[2];
           ref[2]:=ref[1];   u[2]:=u[1];     v[2]:=v[1];
           ref[1]:=ref[0];   u[1]:=u[0];     v[1]:=v[0];
           v[0]:= T.coeffs[1]*ref[1]
                 -S1.coeffs[1]*y1[1] -S2.coeffs[1]*y2[1] -S3.coeffs[1]*y3[1]
                 + ( Ao.coeffs[1] - R.coeffs[1] )*u[1]  - Ao.coeffs[1]*v[1]
                 + T.coeffs[2]*ref[2]
                 -S1.coeffs[2]*y1[2] -S2.coeffs[2]*y2[2] -S3.coeffs[2]*y3[2]
                 + ( Ao.coeffs[2] - R.coeffs[2] )*u[2]  - Ao.coeffs[2]*v[2]
                 + T.coeffs[3]*ref[3]
                 -S1.coeffs[3]*y1[3] -S2.coeffs[3]*y2[3] -S3.coeffs[3]*y3[3]
                 + ( Ao.coeffs[3] - R.coeffs[3] )*u[3]  - Ao.coeffs[3]*v[3]
                 + T.coeffs[4]*ref[4]
                 -S1.coeffs[4]*y1[4] -S2.coeffs[4]*y2[4] -S3.coeffs[4]*y3[4]
                 + ( Ao.coeffs[4] - R.coeffs[4] )*u[4]  - Ao.coeffs[4]*v[4]|

        5: y1[5]:=y1[4];     y2[5]:=y2[4];   y3[5]:=y3[4];
           y1[4]:=y1[3];     y2[4]:=y2[3];   y3[4]:=y3[3];
           y1[3]:=y1[2];     y2[3]:=y2[2];   y3[3]:=y3[2];
           y1[2]:=y1[1];     y2[2]:=y2[1];   y3[2]:=y3[2];
           y1[1]:=y1[0];     y2[1]:=y2[0];   y3[1]:=y3[0];
           ref[5]:=ref[4];   u[5]:=u[4];     v[5]:=v[4];
           ref[4]:=ref[3];   u[4]:=u[3];     v[4]:=v[3];
           ref[3]:=ref[2];   u[3]:=u[2];     v[3]:=v[2];
           ref[2]:=ref[1];   u[2]:=u[1];     v[2]:=v[1];
           ref[1]:=ref[0];   u[1]:=u[0];     v[1]:=v[0];
           v[0]:= T.coeffs[1]*ref[1]
                 -S1.coeffs[1]*y1[1] -S2.coeffs[1]*y2[1] -S3.coeffs[1]*y3[1]
                 + ( Ao.coeffs[1] - R.coeffs[1] )*u[1]  - Ao.coeffs[1]*v[1]
                 + T.coeffs[2]*ref[2]
                 -S1.coeffs[2]*y1[2] -S2.coeffs[2]*y2[2] -S3.coeffs[2]*y3[2]
                 + ( Ao.coeffs[2] - R.coeffs[2] )*u[2]  - Ao.coeffs[2]*v[2]
                 + T.coeffs[3]*ref[3]
                 -S1.coeffs[3]*y1[3] -S2.coeffs[3]*y2[3] -S3.coeffs[3]*y3[3]
```

```
                + ( Ao.coeffs[3] - R.coeffs[3] )*u[3]  - Ao.coeffs[3]*v[3]
                + T.coeffs[4]*ref[4]
                -S1.coeffs[4]*y1[4] -S2.coeffs[4]*y2[4] -S3.coeffs[4]*y3[4]
                + ( Ao.coeffs[4] - R.coeffs[4] )*u[4]  - Ao.coeffs[4]*v[4]
                + T.coeffs[5]*ref[5]
                -S1.coeffs[5]*y1[5] -S2.coeffs[5]*y2[5] -S3.coeffs[5]*y3[5]
                + ( Ao.coeffs[5] - R.coeffs[5] )*u[5]  - Ao.coeffs[5]*v[5];
      END;
    END;
    Signal(Mutex);
  END;
END UpdateRegulatorState;

BEGIN
  firstcall :=TRUE;
END Regulmon.
```

```
DEFINITION MODULE RLS;

EXPORT QUALIFIED
  MaxStore, maxindex, col, matr, EstimateParameters,
  RestartEstimation, InitRLS, Estimation, DORLS;

CONST
  MaxStore = 710;
  maxindex = 10;    (* max number of estimated parameters *)

TYPE
  col  = ARRAY[0..maxindex] OF REAL;
  matr = ARRAY[0..maxindex] OF col;

  Filter2State = RECORD
                   x1,x2 : REAL;
                 END;

VAR
  EstimateParameters: RECORD
                        Theta : col;  (* estimated parameters *)
                        Phi   : col;  (* regression vector *)
                        L     : matr; (* lower triangular part of LD decompo-
                                         sition of P. Notice that the elements
                                         are stored columnwise *)
                        D     : col;  (* diag. part of LD decomposition of P *)
                        N     : CARDINAL; (* number of estimated param *)
                      END;

PROCEDURE RestartEstimation(zerotheta:BOOLEAN; VAR ok:BOOLEAN);

PROCEDURE InitRLS;

PROCEDURE Filter2(x:REAL; VAR states:Filter2State; VAR xf:REAL);

PROCEDURE Estimation(u,y:REAL);

PROCEDURE DORLS(VAR theta,d:col; VAR l:matr; phi:col; n:CARDINAL);

END RLS.
```

```
IMPLEMENTATION MODULE RLS;

FROM Pcalc    IMPORT mdegree, coeffvector, cpoly;
FROM Globals  IMPORT ModelType, GlobalData, EstimDataType;
FROM Screend  IMPORT ErrorMessage;
FROM ShowPar  IMPORT SetStoredParameters;

CONST
  relative =1.0E-8;

VAR
  uf1, uf2, uf3 : REAL;
  ufilterstate,
  yfilterstate  : Filter2State;

(*-------------------------------------------------------------------------*)

PROCEDURE Filter2(x:REAL; VAR states:Filter2State; VAR xf:REAL);
(* Second order filter for regressors *)
VAR nx1,nx2: REAL;
BEGIN
  WITH GlobalData.EstimData DO
    WITH states DO
      nx1 := -Af.coeffs[1]*x1 - Af.coeffs[2]*x2 + x;
      nx2 :=                 x1;
      xf  := (Bf.coeffs[1]-Bf.coeffs[0]*Af.coeffs[1])*x1
           + (Bf.coeffs[2]-Bf.coeffs[0]*Af.coeffs[2])*x2 + Bf.coeffs[0]*x;

      x1:=nx1;
      x2:=nx2;
    END;
  END;
END Filter2;

(*-------------------------------------------------------------------------*)
(*  Routines for RLS estimation based on dyadic reduction.                 *)
(*  Ref. Peterka IFAC-86.                                                  *)
(*                                                                         *)
(*    Karl-Johan Astrom and Michael Lundh                  87.05.05        *)
(*-------------------------------------------------------------------------*)

PROCEDURE
DyadicReduction(VAR a,b:col; VAR alpha,beta:REAL; i0,i1,i2 :CARDINAL);
CONST
  mzero = 1.0E-10;
VAR
  i            : CARDINAL;
  w1,w2,b1,gam : REAL;
BEGIN
  IF beta<mzero THEN beta:=0.0; END;
  b1 := b[i0];
  w1 := alpha;
  w2 := beta*b1;
  alpha := alpha + w2*b1;
  IF alpha > mzero THEN
    beta := w1*beta/alpha;
    gam := w2/alpha;
    FOR i:=i1 TO i2 DO
      b[i] := b[i] - b1*a[i];
      a[i] := a[i] + gam*b[i];
    END;
  END;
END DyadicReduction;

PROCEDURE
LDFilter(VAR theta,d:col; VAR l:matr; phi:col; lambda:REAL; n:CARDINAL);
VAR
  i,j  : CARDINAL;
  e,w  : REAL;
BEGIN
  d[0] := lambda;
  e    := phi[0];
  FOR i:=1 TO n DO
    e:=e-theta[i]*phi[i];
    w:=phi[i];
    FOR j:=i+1 TO n DO w:=w+phi[j]*l[i,j]; END;
    l[0,i]:=0.0;
    l[i,0]:=w;
  END;
  FOR i:=n TO 1 BY -1 DO (* n.b. backward loop *)
    DyadicReduction(l[0],l[i],d[0],d[i],0,i,n);
  END;

  FOR i:=1 TO n DO
    theta[i]:=theta[i]+l[0,i]*e;
    d[i]:=d[i]/lambda;
  END;
END LDFilter;

(*-------------------------------------------------------------------------*)

PROCEDURE RestartEstimation(zerotheta:BOOLEAN; VAR ok:BOOLEAN);
VAR i,j:CARDINAL;
BEGIN
  WITH GlobalData.Model DO   (* set model polynomials *)
    IF A.degree+B.degree+1>maxindex THEN
      ErrorMessage('Deg A + Deg B>9');
      ok := FALSE;
      RETURN;
    END;

    FOR i:=A.degree+1 TO mdegree DO  A.coeffs[i] :=0.0;  END;
    FOR i:=B.degree+1 TO mdegree DO  B.coeffs[i] :=0.0;  END;

    A.coeffs[0] := 1.0;

    IF zerotheta THEN  (* default settings of A and B *)
      FOR i:=1 TO mdegree DO  A.coeffs[i] :=0.0;  END;
      CASE A.degree OF
        1:  A.coeffs[1] := -0.5|
        2:  A.coeffs[1] := -1.5; A.coeffs[2] := 0.7;
      ELSE  A.coeffs[1] := -2.0; A.coeffs[2] := 1.4; A.coeffs[3] := 0.4;
      END;
      FOR i:=0 TO B.degree DO  B.coeffs[i] := 2.0;  END;
    END;

    WITH EstimateParameters DO  (* transfer parameters from model *)
      Theta[0] :=0.0;
      FOR i:=1 TO A.degree DO
        Theta[i] := A.coeffs[i];
      END;
      FOR i:=0 TO B.degree DO
```

```
        Theta[i+A.degree+1] := B.coeffs[i];
      END;
      N:=A.degree+B.degree+1;
      FOR i:=N+2 TO maxindex DO
        Theta[i] := 0.0;
      END;

      FOR i:=0 TO maxindex DO
        FOR j:=0 TO maxindex DO
          L[i,j] :=0.0;
        END;
        L[i,i] :=1.0;
        D[i]   := GlobalData.EstimData.D0;
        Phi[i] := 0.0;
      END;
    END;
  END;

  WITH ufilterstate DO x1:=0.0; x2:=0.0; END;
  WITH yfilterstate DO x1:=0.0; x2:=0.0; END;

  uf1 := 0.0;  uf2 := 0.0;  uf3 := 0.0;

  GlobalData.EstimData.EstPurpose:='MODELPOLY';
END RestartEstimation;

PROCEDURE InitRLS;
VAR i,j : CARDINAL;
    ok  : BOOLEAN;
BEGIN
  RestartEstimation(TRUE,ok);

  WITH ufilterstate DO x1:=0.0; x2:=0.0; END;
  WITH yfilterstate DO x1:=0.0; x2:=0.0; END;

  uf1 := 0.0;  uf2 := 0.0;  uf3 := 0.0;
END InitRLS;

PROCEDURE DORLS(VAR theta,d:col; VAR l:matr; phi:col; n:CARDINAL);
BEGIN
  LDFilter(theta,d,l,phi,GlobalData.EstimData.Lambda,n);
  SetStoredParameters(theta,d);
END DORLS;

PROCEDURE Estimation(u,y:REAL);
VAR i      : CARDINAL;
    uf0,yf0: REAL;
BEGIN
  Filter2( u, ufilterstate, uf0);
  Filter2( y, yfilterstate, yf0);

  WITH EstimateParameters DO
    Phi[0]:=yf0;
    DORLS(Theta,D,L,Phi,N);

    WITH GlobalData.Model DO
      (* Update regressors (phi-vector) and transfer parameters to model *)
      FOR i:=N TO 2 BY -1 DO
        Phi[i]:=Phi[i-1];
      END;
      Phi[1]          := -yf0;

      CASE delay OF
        1: Phi[A.degree+1] := uf0|
        2: Phi[A.degree+1] := uf1|
        3: Phi[A.degree+1] := uf2|
        4: Phi[A.degree+1] := uf3;
      END;
      FOR i:=1 TO A.degree DO
        A.coeffs[i] := Theta[i];
      END;
      FOR i:=0 TO B.degree DO
        B.coeffs[i] := Theta[i+A.degree+1];
      END;

    END;
  END;
  uf3 := uf2;
  uf2 := uf1;
  uf1 := uf0;
END Estimation;

END RLS.
```

```
DEFINITION MODULE Screend;

FROM Graphics IMPORT handle, rectangle, color;

EXPORT QUALIFIED InitScreen, ScreenData, ErrorMessage, ResetErrorMessage,
                 WarningMessage, WriteMouseBox, WriteMenyLine;
                (* ReadVersionNumberAndDate; *)

VAR
  ScreenData : RECORD
                 SmallPlothandle,
                 BigPlothandle,
                 DiffPlothandle,
                 UPlothandle,
                 Mousehandle,
                 Opcomhandle,
                 Commandhandle        : handle;
                 Refcolor, Y1color,
                 Y2color, Y3color,
                 Ucolor,   Diffcolor,
                 Plotcolor, Textcolor : color;
                END;

(* initial *) PROCEDURE InitScreen(ver:ARRAY OF CHAR);

PROCEDURE WriteMouseBox(row,col:CARDINAL;txt:ARRAY OF CHAR;co:color);
PROCEDURE WriteMenyLine(t1,t2,t3,t4,t5,t6,t7,t8,t9,t10:ARRAY OF CHAR;co:color);

PROCEDURE ErrorMessage(txt:ARRAY OF CHAR);
PROCEDURE WarningMessage(txt:ARRAY OF CHAR);
PROCEDURE ResetErrorMessage;

(* PROCEDURE ReadVersionNumberAndDate(ver:ARRAY OF CHAR); *)

END Screend.
```

```
IMPLEMENTATION MODULE Screend;

FROM Graphics IMPORT VirtualScreen,SetViewPort,SetWindow,handle,rectangle,
                     point, SetMouseRectangle, EraseChar,PolyLine,
                     WriteString,color,SetFillColor, SetLineColor,
                     FillRectangle, SetTextColor, HideCursor, ShowCursor;

VAR
  Errorhandle   : handle;
  Errorbox      : rectangle;
  MouseBoxRect  : ARRAY [1..3],[0..9] OF rectangle;
  MouseBoxCol   : ARRAY [1..3],[0..9] OF color;

PROCEDURE Message(txt1,txt2:ARRAY OF CHAR;co:color);
VAR Errorpoint:point;
BEGIN
  HideCursor;
  SetFillColor(Errorhandle,co);
  FillRectangle(Errorhandle,Errorbox);
  Errorpoint.h := 0.1;
  Errorpoint.v := 1.1;
  WriteString(Errorhandle,Errorpoint,txt1);
  Errorpoint.v := 0.2;
  WriteString(Errorhandle,Errorpoint,txt2);
  ShowCursor;
END Message;

PROCEDURE ErrorMessage(txt:ARRAY OF CHAR);
BEGIN
  Message(' ERROR ',txt,red);
END ErrorMessage;

PROCEDURE WarningMessage(txt:ARRAY OF CHAR);
BEGIN
  Message(' WARNING ',txt,lightred);
END WarningMessage;

PROCEDURE ResetErrorMessage;
BEGIN
  SetFillColor(Errorhandle,black);
  FillRectangle(Errorhandle,Errorbox);
END ResetErrorMessage;

PROCEDURE MenyGrid;
VAR p:ARRAY [0..1] OF point;
BEGIN
  SetLineColor(ScreenData.Mousehandle,black);
  p[0].h:=0.100;  p[1].h:=9.870;  p[0].v:=0.162;  p[1].v:=0.162; (* 160 163 *)
  PolyLine(ScreenData.Mousehandle,p,2);

  p[0].h:=3.060;  p[1].h:=3.060;  p[0].v:=0.120;  p[1].v:=0.162;
  PolyLine(ScreenData.Mousehandle,p,2);

  p[0].h:=5.060;  p[1].h:=5.060;
  PolyLine(ScreenData.Mousehandle,p,2);

  p[0].h:=7.060;  p[1].h:=7.060;
  PolyLine(ScreenData.Mousehandle,p,2);
END MenyGrid;

PROCEDURE WriteMouseBox(row,col:CARDINAL;txt:ARRAY OF CHAR;co:color);
VAR pkt:point;
BEGIN
  CASE row OF
    1: pkt.v := 0.210|
    2: pkt.v := 0.168|
    3: pkt.v := 0.125;
  END;
  pkt.h  := 0.120 + FLOAT(col);
  HideCursor;
  WITH ScreenData DO
    SetFillColor(Mousehandle,MouseBoxCol[row,col]);
    FillRectangle(Mousehandle,MouseBoxRect[row,col]);
    SetTextColor(Mousehandle,co);
    WriteString(Mousehandle,pkt,txt);
  END;
  MenyGrid;
  ShowCursor;
END WriteMouseBox;

PROCEDURE InitMouseBox(row,col:CARDINAL;background:color);
VAR r:rectangle;
BEGIN
  CASE row OF
    1: r.loleft.v := 0.205;  r.upright.v := 0.245|
    2: r.loleft.v := 0.163;  r.upright.v := 0.203|
    3: r.loleft.v := 0.120;  r.upright.v := 0.160;
  END;
  IF col=0 THEN
    r.loleft.h  := FLOAT(col)+0.1;
  ELSE
    r.loleft.h  := FLOAT(col)+0.015;
  END;
  IF col<9 THEN
    r.upright.h := FLOAT(col)+1.0;
  ELSE
    r.upright.h := FLOAT(col)+0.87;
  END;
  MouseBoxRect[row,col]:=r;
  MouseBoxCol[row,col]:=background;

  r.loleft.v  := r.loleft.v +0.004;
  r.upright.v := r.upright.v-0.004;
  r.loleft.h  := r.loleft.h +0.15;
  r.upright.h := r.upright.h-0.15;
  SetMouseRectangle(ScreenData.Mousehandle,r,10*row+col);
END InitMouseBox;

PROCEDURE WriteMenyLine(t0,t1,t2,t3,t4,t5,t6,t7,t8,t9:ARRAY OF CHAR;co:color);
BEGIN
  WriteMouseBox(1,0,t0,co);
  WriteMouseBox(1,1,t1,co);
  WriteMouseBox(1,2,t2,co);
  WriteMouseBox(1,3,t3,co);
  WriteMouseBox(1,4,t4,co);
  WriteMouseBox(1,5,t5,co);
  WriteMouseBox(1,6,t6,co);
  WriteMouseBox(1,7,t7,co);
  WriteMouseBox(1,8,t8,co);
  WriteMouseBox(1,9,t9,co);
END WriteMenyLine;
```

```
(* ------------------------------------------------------------- *)

PROCEDURE DefineArea(VAR h:handle; xl,xh,yl,yh:REAL);
VAR r : rectangle ;
BEGIN
  VirtualScreen(h);
  r.loleft.v:=  yl;   r.loleft.h:=  xl;
  r.upright.v:= yh;   r.upright.h:= xh;
  SetViewPort(h,r);
  (* dummy settings to prevent run time errors *)
  SetWindow(h,r);
  SetTextColor(h,red);
  SetLineColor(h,red);
  SetFillColor(h,red);
END DefineArea;

(* initial *) PROCEDURE InitScreen(ver:ARRAY OF CHAR);
VAR r:rectangle;
    pkt:point;
    i:CARDINAL;
BEGIN
  WITH ScreenData DO
    DefineArea(SmallPlothandle, 0.010, 1.490, 0.850, 0.990);
    DefineArea(BigPlothandle,   0.010, 1.490, 0.600, 0.990);
    DefineArea(DiffPlothandle,  0.010, 1.490, 0.422, 0.598);
    DefineArea(UPlothandle,     0.010, 1.490, 0.250, 0.420); (* old 280 *)

    DefineArea(Mousehandle,     0.000, 1.500, 0.120, 1.000);
    r.loleft.h  :=  0.0;        r.loleft.v  :=  0.12;
    r.upright.h := 10.0;        r.upright.v :=  1.000;
    SetWindow(Mousehandle,r);
    SetFillColor(Mousehandle,white);
    FillRectangle(Mousehandle,r);

    FOR i:=0 TO 9 DO
      InitMouseBox(1,i,grey);
      InitMouseBox(2,i,white);
    END;
    FOR i:=0 TO 6 DO
      InitMouseBox(3,i,white);
    END;
    SetTextColor(Mousehandle,blue);
    pkt.h:=7.2;  pkt.v:=0.125; (* old 0.120 *)
    WriteString(Mousehandle,pkt,ver);
(*  WriteString(Mousehandle,pkt,VersionNumber); *)

    r.loleft.v  := 0.25;    r.loleft.h  := 0.01;  (* Define box for work area *)
    r.upright.v := 0.84;    r.upright.h := 9.99;
    SetMouseRectangle(Mousehandle,r,0);

    DefineArea(Opcomhandle,   0.010, 1.490, 0.250, 0.848); (* old 280 *)
    DefineArea(Commandhandle, 0.010, 0.990, 0.000, 0.115);

    Refcolor  := green   ;
    Y1color   := yellow  ;
    Y2color   := cyan    ;
    Y3color   := magenta ;
    Ucolor    := blue    ;
    Diffcolor := red     ;
    Plotcolor := lightblue;
    Textcolor := green ;
  END;

  (* Area for Errorhandle *)
  DefineArea(Errorhandle, 1.000, 1.490, 0.001, 0.115);
  Errorbox.loleft.h := 0.0;   Errorbox.upright.h := 1.0;
  Errorbox.loleft.v := 0.0;   Errorbox.upright.v := 2.0;
  SetWindow(Errorhandle,Errorbox);
  SetFillColor(Errorhandle,black);
  FillRectangle(Errorhandle,Errorbox);
  SetTextColor(Errorhandle,black);
END InitScreen;

END Screend.
```

```
DEFINITION MODULE ShowPar;

FROM RLS IMPORT col;

EXPORT QUALIFIED SetStoredParameters, ShowStoredParameters;

PROCEDURE SetStoredParameters(th,d:col);

PROCEDURE ShowStoredParameters(typ:ARRAY OF CHAR; i1,ih:CARDINAL;
                               c1:ARRAY OF CHAR;  l1,h1:CARDINAL;
                               c2:ARRAY OF CHAR;  l2,h2:CARDINAL;
                               c3:ARRAY OF CHAR;  l3,h3:CARDINAL);
END ShowPar.
```

```
IMPLEMENTATION MODULE ShowPar;

FROM Globals  IMPORT GlobalData, EstimDataType;
FROM Graphics IMPORT SetWindow, rectangle,point, PolyLine,
                     color,SetFillColor, SetLineColor,
                     FillRectangle, SetTextColor (*HideCursor, ShowCursor *) ;
FROM RLS      IMPORT MaxStore, maxindex, col;
FROM Screend  IMPORT ScreenData, (*, ErrorMessage *) WarningMessage;
FROM Opcom    IMPORT WriteReal, WriteText;
FROM NumberConversion IMPORT CardToString;
FROM Strings  IMPORT Concat, Length, CompareStr;
FROM StoreTh  IMPORT StoredTheta;
FROM StoreDM  IMPORT StoredDmatrix;

TYPE
  TxtType = ARRAY [0..15] OF CHAR;
VAR
  i,j:CARDINAL;
  th : ARRAY[1..MaxStore] OF col;

  (* StoredTheta and StoredDmatrix declared in own modules *)
  StoreIndex   : CARDINAL;                  (* Index to Ring Buffer *)

PROCEDURE SetStoredParameters(th,d:col);
BEGIN
  StoreIndex:=(StoreIndex MOD MaxStore) + 1;
  StoredTheta[StoreIndex]   := th;
  StoredDmatrix[StoreIndex] := d;
END SetStoredParameters;

PROCEDURE Addstr(str:ARRAY OF CHAR; i:CARDINAL; VAR outstr:TxtType);
BEGIN
  Concat(str,' ',outstr);
  outstr[Length(str)]  :=CHR(48+i);
  outstr[Length(str)+1]:=CHR(0);
END Addstr;

PROCEDURE ShowStoredParameters(typ:ARRAY OF CHAR; il,ih:CARDINAL;
                               c1:ARRAY OF CHAR;  l1,h1:CARDINAL;
                               c2:ARRAY OF CHAR;  l2,h2:CARDINAL;
                               c3:ARRAY OF CHAR;  l3,h3:CARDINAL);
VAR
  PlTh        : ARRAY[0..MaxStore] OF point;
  VAxis       : ARRAY[0..1] OF point;
  box         : rectangle;
  max,min,
  maxp,minp,
  thmin,thmax : REAL;
  timestr     : TxtType;
  Labels      : ARRAY [1..maxindex+2] OF TxtType;
  i,j,k,
  si, index   : CARDINAL;
BEGIN
  (* Check history *)
  WITH GlobalData.EstimData DO
    HistoryCard := TRUNC( HistoryTime/GlobalData.SetupData.RealTsamp+0.5);
    IF HistoryCard>=MaxStore-10 THEN (* -10 for safety *)
      HistoryCard:= MaxStore-11;
      HistoryTime:= FLOAT(HistoryCard)*GlobalData.SetupData.RealTsamp;
      WarningMessage('History Changed');
    END;
  END;

  (*  Get Stored Parameters *)
  index := StoreIndex;
  IF index < GlobalData.EstimData.HistoryCard THEN
    si:=index + MaxStore - GlobalData.EstimData.HistoryCard;
  ELSE
    si:=index - GlobalData.EstimData.HistoryCard;
  END;
  IF si=0 THEN si:=1; END; (* safety ?? - run time error occurred once *)

  thmax:=-100000.0;
  thmin:= 100000.0;

  FOR i:=1 TO GlobalData.EstimData.HistoryCard DO
    WITH GlobalData.EstimData DO
      PlTh[i-1].h:= FLOAT(i-1)*HistoryTime/FLOAT(HistoryCard);
    END;

    k:=0;
    FOR j:=il TO ih DO
      k:=k+1;
      IF typ[0]='T' THEN
        th[i,k] := StoredTheta[si,j];
      ELSE
        th[i,k] := StoredDmatrix[si,j];
      END;
      IF th[i,k] > thmax THEN  thmax:=th[i,k]; END;
      IF th[i,k] < thmin THEN  thmin:=th[i,k]; END;
    END;
    si := (si MOD MaxStore) + 1;
  END;

  j:=0;
  FOR i:=l1 TO h1 DO j:=j+1; Addstr(c1,i,Labels[j]); END;
  FOR i:=l2 TO h2 DO j:=j+1; Addstr(c2,i,Labels[j]); END;
  FOR i:=l3 TO h3 DO j:=j+1; Addstr(c3,i,Labels[j]); END;

  IF ABS(thmax - thmin)>0.1 THEN (* vertical scaling *)
    min := thmin;
    max := thmax;
  ELSE
    min := 0.5*(thmin+thmax) - 0.05;
    max := 0.5*(thmin+thmax) + 0.05;
  END;

  WITH box DO
    loleft.v  := 0.5*(max + min) - 0.55*(max-min);
    upright.v := 0.5*(max + min) + 0.65*(max-min);

    minp      := 0.5*(max + min) - 0.54*(max-min);
    maxp      := 0.5*(max + min) + 0.58*(max-min);

    WITH GlobalData.EstimData DO
      loleft.h:= -0.02*HistoryTime; upright.h:= 1.02*HistoryTime;

      CardToString(TRUNC(HistoryTime+0.5),timestr,4);
      Concat(timestr,' s',timestr);

      (* HideCursor; *)
      SetWindow(ScreenData.Opcomhandle,box);
```

```
      SetTextColor(ScreenData.Opcomhandle,green);
      SetLineColor(ScreenData.Opcomhandle,green);
      SetFillColor(ScreenData.Opcomhandle,black);
      FillRectangle(ScreenData.Opcomhandle,box);

      VAxis[0].h := 0.0;  VAxis[0].v := min;
      VAxis[1].h := 0.0;  VAxis[1].v := max;
      PolyLine(ScreenData.Opcomhandle,VAxis,2);

      WriteReal(0.02*HistoryTime,max,'',max);
      WriteReal(0.02*HistoryTime,min,'',min);
      WriteText(0.90*HistoryTime,minp,timestr);
      IF typ[0]='T' THEN
        IF CompareStr(GlobalData.EstimData.EstPurpose,'MODELPOLY')=0 THEN
          WriteText(0.0*HistoryTime,maxp,'Model Menu: Estimated Parameters');
        ELSIF CompareStr(GlobalData.EstimData.EstPurpose,'REGULPOLY')=0 THEN
          WriteText(0.0*HistoryTime,maxp,'Regul Menu: Estimated Parameters');
        END;
      ELSE
        WriteText(0.0*HistoryTime,maxp,'Estim Menu: D-matrix Diagonal');
      END;

      FOR k:=1 TO ih-il+1 DO
        SetLineColor(ScreenData.Opcomhandle,VAL(color,k));
        SetTextColor(ScreenData.Opcomhandle,VAL(color,k));

        WriteText((0.46+0.05*FLOAT(k))*HistoryTime,maxp,Labels[k]);

        FOR j:=1 TO GlobalData.EstimData.HistoryCard DO
          PlTh[j-1].v:= th[j,k];
        END;

        PolyLine(ScreenData.Opcomhandle, PlTh,
                 GlobalData.EstimData.HistoryCard);
      END;
    END;
  END;

  (* ShowCursor; *)
END ShowStoredParameters;

BEGIN
  StoreIndex := 1;
  FOR i:=1 TO MaxStore DO
    FOR j:=0 TO maxindex DO
      StoredTheta[i,j]   := 0.0;
      StoredDmatrix[i,j] := 0.0;
    END;
  END;
END ShowPar.
```

```
DEFINITION MODULE StoreDM;

FROM RLS IMPORT col, MaxStore;

EXPORT QUALIFIED StoredDmatrix;

VAR
  StoredDmatrix: ARRAY[1..MaxStore] OF col; (* Ring buffers *)

END StoreDM.
```

```
IMPLEMENTATION MODULE StoreDM;


END StoreDM.
```

```
DEFINITION MODULE StoreTh;

FROM RLS IMPORT col, MaxStore;

EXPORT QUALIFIED StoredTheta;

VAR
  StoredTheta: ARRAY[1..MaxStore] OF col; (* Ring buffers *)

END StoreTh.
```

```
IMPLEMENTATION MODULE StoreTh;


END StoreTh.
```

```
MODULE ToolBox;

FROM RTMouse  IMPORT Init;
FROM Kernel   IMPORT CreateProcess;
FROM Globals  IMPORT InitGlobals;
FROM Screend  IMPORT InitScreen;
FROM Regulmon IMPORT InitRegulmonitor;
FROM RLS      IMPORT InitRLS;
FROM PRCplot  IMPORT InitPlotter,   Plotter;
FROM PRCexec  IMPORT InitExecute,   Execute, Update;
FROM PRCmouse IMPORT InitMouse,     MouseHigh, MouseLow, WaitTheEnd;
FROM PRCkeyb  IMPORT InitKeyboard,  Keyboard;

(* ------------------------------------------------------------------------ *)
CONST
  VersionNumberAndDate = 'Version 5.2   88.03.23';
(* ------------------------------------------------------------------------ *)

BEGIN
  Init;

  InitScreen(VersionNumberAndDate);
  InitGlobals;

  InitExecute;
  InitRegulmonitor;
  InitRLS;

  InitPlotter;
  InitMouse;
  InitKeyboard;

  CreateProcess( Execute,    4000 );
  CreateProcess( Update,     4000 );
  CreateProcess( Plotter,    4000 );
  CreateProcess( MouseHigh,  2000 );
  CreateProcess( MouseLow,  20000 );  (* old 10000 *)
  CreateProcess( Keyboard,  10000 );

  WaitTheEnd;
END ToolBox.
```