Self-tuners with Automatic Adjustment of the Sampling Period for Processes with Time Delays

Åström, Karl Johan; Zhou, Zhao-Ying

1981

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

CODEN:LUTFD2/(TFRT-7229)/1-074/(1981)

SELF-TUNERS WITH AUTOMATIC ADJUSTMENT OF THE SAMPLING PERIOD FOR PROCESSES WITH TIME DELAYS

KARL JOHAN ÅSTRÖM
ZHOU ZHAO-YING

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL

AUGUST 1981

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
REPORT

Date of issue
August 1981

Document number
CODEN:LUTFD2/(TFRT-7229)/1-074/(1981)

Author(s)
Karl Johan Åström
Zhou Zhao-ying

Supervisor

Sponsoring organization
STU-78-3763

Title and subtitle
Self-Tuners with Automatic Adjustment of the Sampling Period for Processes with Time Delays.

Abstract
Simpe self-tuners for processes with time delay and low order dynamics are developed. Methods for automatic adjustment of the sampling period are discussed. The performance of the regulators are illustrated by simulation, and experiments on laboratory processes.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
74

Recipient's notes

Security classification

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

SELF-TUNERS WITH AUTOMATIC ADJUSTMENT OF THE SAMPLING PERIOD FOR PROCESSES WITH TIME DELAYS

Karl Johan Åström

Zhou Zhao-ying

Department of Automatic Control

Lund Institute of Technology

August 1981

ABSTRACT

Simple self-tuners for processes with time delay and low order dynamics are developed. Methods for automatic adjustment of the sampling period are discussed. The performance of the regulators are illustrated by simulation, and experiments on laboratory processes.

CONTENTS

APPENDIX A: SIMNON PROGRAM

Self-tuner with Two More Extra Parameters of B

B: PASCAL PROGRAM 1

C: PASCAL PROGRAM 2

1. INTRODUCTION

There are many industrial processes whose dynamics can be approximated with a time delay T_d and a first order lag with time constant T. If the time delay T_d is much shorter than T the process is easy to control. A proportional regulator with a high gain can for example be used. If the time delay T_d is larger than the time constant T then the process has a considerable phase shift already for very low frequencies. The gain of a proportional controller must then be chosen so small that the total loop gain is less than one. This means that the proportional gain has to be less than the inverse value of the process gain. To control such processes in a reasonable way it is necessary to have controllers with dead time compensation e.g. of the type proposed by Smith (1958). Dead time compensation in essence amounts to storing past control actions and taking them into account when determining the control signal. In this way it is possible to maintain a high loop gain.

The problem formulation is given in section 2. The design problem for systems with known parameters is stated and solved in section 3. Section 4 deals with the parameter estimation problem. Different ways of adjusting the sampling period are covered in section 5. Results from simulations and experiments using DEC LSI 11/03 to control processes simulated on an analog computer are given in section 6.

2. THE CONTROL PROBLEM

The problem formulation is discussed in this section. The regulator structure chosen is the same as for the regulator R2 in the DDC package PPCP which was developed at the IBM Nordic laboratory. It is proposed that the regulator should be modified a little to avoid the inherent difficulties with the one-degree-of- freedom structure used. The processes to be controlled are then characterized and a formulation of the control problem is given.

Regulator Structure

The regulator R2 is characterized by

$$\nabla u(t) + r_1 \nabla u(t-h) + r_2 \nabla u(t-2h) = - s_0 e(t) - s_1 e(t-h)$$

where u(t) is the control signal and

$$e(t) = u_c(t) - y(t)$$

is the errror, and ∇ a difference operator. The regulator R2 is a one-degree-of-freedom controller according to Horowitz (1963). Because of this the regulator will always introduce a zero $z = - s_1 / s_0$ in the transfer function. This is a serious limitation because it may result in an unnecessarily large overshoot. It is thus useful to replace the regulator with a two degree of freedom configuration characterized by

$$\nabla u(t) + r_1 \nabla u(t-h) + r_2 \nabla u(t-2h) = t_0 u_c(t) + t_0 t_1 u_c(t-h) - s_0 y(t) - s_1 y(t-h) \quad (2.1)$$

The control algorithm is incremental. This is because the computer control actions are introduced via increase and decrease functions in the analog part of the system. This can not be changed without drastic modification of the system.

Process Model

The processes to be controlled are described as a combination of a time delay T_d and a first order lag. Such an approximation is sufficiently accurate for the purpose of control in many cases. The process model is thus

$$\frac{dy(t)}{dt} = -\alpha y(t) + \beta u(t-T_d) \qquad (2.2)$$

The corresponding transfer function is

$$G(s) = \frac{k}{1+sT} e^{-sT_d} \qquad (2.3)$$

where

$$\left.\begin{aligned} k &= \beta / \alpha \\ T &= 1 / \alpha \end{aligned}\right\} \qquad (2.4)$$

Assume that the process is sampled with the sampling period h. The behaviour of the sampled system at times which are synchronized to the sampling instants can be described by the equation

$$y(t) = ay(t-h) + b_1 u(t-kh) + b_2 u(t-kh-h) \qquad (2.5)$$

where

$$\left.\begin{aligned} a &= e^{-\alpha h} \\ b_1 &= \frac{\beta}{\alpha}\left[1 - e^{-\alpha(h - T_d \bmod h)}\right] \\ b_2 &= \frac{\beta}{\alpha}\left[e^{-\alpha(h - T_d \bmod h)} - e^{-\alpha h}\right] \\ k &= 1 + T_d \operatorname{div} h \end{aligned}\right\} \qquad (2.6)$$

If the sampled model (2.5) is given the corresponding parameters of the continous time system is given by

$$\left.\begin{aligned} \alpha &= -\frac{1}{h} \ln a \\ \beta &= -\frac{b_1 + b_2}{h(1-a)} \ln a \\ T_d &= \left[k - 1 - \frac{\ln(ab_1 + b_2)/(ab_1 + ab_2)}{\ln a}\right] h \end{aligned}\right\} \qquad (2.7)$$

The process gain K and the time constant is given by

$$\left.\begin{aligned} K &= \frac{b_1 + b_2}{1 - a} \\ T &= -h / \ln a \end{aligned}\right\} \qquad (2.8)$$

Problem Formulation

It is intended to design a regulator with the structure (2.1) which will work well under a variety of circumstances. Some of the applications can be captured by an optimization criterion like minimum variance but for many other applications the goal is more loosely defined e.g. to obtain control loops with reasonable gain and settling times. In the applications considered it is also desirable to have responses without overshoot. In some cases there are also problems with disturbances. It is therefore decided to design for a specified settling time and to include filtering by specifying a first order observer with a given time constant.

3. CONTROL DESIGN

The control system design problem will now be solved. The essence of the problem can be captured by a pole placement design. Since the specifications call for settling time without overshoot and since they are not very strict, it is reasonable to require that the sampled system has zeros at the origin and at

$$z_1 = \exp(-h/T_c)$$

where T_c is the specified settling time. Similarly the observer polynomial is specified by requiring that it has zeros at the origin and at

$$z_2 = \exp(-h/T_0)$$

where T_0 is specified. The desired performance is thus controlled by the two parameters T_c and T_0. The regulator is designed so that the transfer function from the command signal to the output is given by

$$H_d(z) = \frac{1 + p_1}{b_1 + b_2} \cdot \frac{b_1 z + b_2}{z^k (z + p_1)} \qquad (3.1)$$

where

$$p_1 = -\exp(-h/T_c)$$

The regulator will thus have a zero at $z = -b_2 / b_1$. There will also be a delay of at least two sampling periods. This can not be avoided because of the delay in the process. If the process zeros is well damped it can be attemped to cancel the process zero to obtain the transfer function

$$H_d(z) = \frac{1 + p_1}{b_1} \cdot \frac{b_1}{z^{k-1} (z + p_1)} \qquad (3.2)$$

The cases (3.1) and (3.2) will be considered separately.

No Cancellation of Process Zeros

Using z-transform notation the sampled process model (2.5)

can be described by

$$H(z) = \frac{B(z)}{A(z)}$$

where

$$A(z) = z^k (z - a)$$
$$B(z) = b_1 z + b_2$$

Only two cases, $k = 1$ and $k = 2$ are considered. They represent the situations of $h > T_d$ and $2h > T_d > h$ respectively. Since the desired regulator structure (2.1) is such that it contains an integrator the basic design equation for the pole placement synthesis is

$$\left.\begin{aligned} &(z-a)(z-1)(z^2+r_1 z+r_2) + (b_1 z+b_2)(s_0 z+s_1) \\ &\qquad = z^2(z+p_1)(z+t_1) \qquad \text{for } k = 2 \\ &(z-a)(z-1)(z-r_1) + (b_1 z + b_2)(s_0 z+s_1) \\ &\qquad = z(z+p_1)(z+t_1) \qquad \text{for } k = 1 \end{aligned}\right\} \quad (3.3)$$

This means that one observer pole is at the origin and that the closed loop system has two additional poles at origin for $k = 2$, or one for $k = 1$.

Equating coefficients of equal powers of z the following equations are obtained for $k = 2$

$$z^3: \quad -a - 1 + r_1 = p_1 + t_1$$

$$z^2: \quad -r_1(a+1) + a + r_2 + b_1 s_0 = p_1 t_1$$

$$z^1: \quad a r_1 - r_2(1+a) + b_1 s_1 + b_2 s_0 = 0$$

$$z^0: \quad a r_2 + b_2 s_1 = 0$$

This set of linear equations can be solved if $b_1 + b_2 \neq 0$ and $ab_1 + b_2 \neq 0$. The solution is

$$
\left.
\begin{aligned}
r_1 &= 1 + a + p_1 + t_1 \\
r_2 &= \{ b_2^2[(p_1 t_1 - a) + (1+a) r_1] + b_1 b_2 a r_1 \} / N \\
s_0 &= \{ b_2[(1+a+a^2) r_1 + (1+a)(p_1 t_1 - a)] \\
&\quad + b_2[a(1+a) r_1 + a(p_1 t_1 - a)] \} / N \\
s_1 &= \{ b_2[-a(1+a) r_1 - a(p_1 t_1 - a)] - a^2 b_1 r_1 \} / N \\
t_0 &= (1+p_1) / (b_1 + b_2) \\
N &= (b_1 + b_2)(a b_1 + b_2)
\end{aligned}
\right\} \qquad (3.4)
$$

For k = 1 we have similarly

$$
\left.
\begin{aligned}
s_0 &= \frac{1}{1-a} \left[\frac{(1+p_1)(1+t_1)}{b_1 + b_2} - \frac{a(a+p_1)(a+t_1)}{a b_1 + b_2} \right] \\
s_2 &= \frac{1}{1-a} \left[\frac{a(a+p_1)(a+t_1)}{a b_1 + b_2} - \frac{a(1+p_1)(1+t_1)}{b_1 + b_2} \right] \\
&\text{and} \\
r_1 &= p_1 + t_1 + 1 + a - b_1 s_0
\end{aligned}
\right\} \qquad (3.5)
$$

The condition

$$b_1 + b_2 = 0 \qquad (3.6)$$

implies that the process model has a zero at z = 1, which cancels the zero in the regulator. To obtain the controller in that case the factor (z-1) in (3.3) should be cancelled before the polynomials R and S are determined. If this is done the regulator will, however, no longer contain an integrator. This means that if (3.6) holds then there is no regulator of the form (2.1) which will do the job.

The condition

$$a b_1 + b_2 = 0 \qquad (3.7)$$

implies that there is a common factor in the process model. This factor should then be cancelled before the equation (3.3) is solved. If (3.7) holds the process model reduced to

$$H(z) = \frac{b_1}{z^k}$$

or

$$y(t) = b_1 u(t-kh)$$

The design equation (3.3) reduces to

$$(z-1)(z+r_1) + b_1 s_0 = (z+p_1)(z+t_1) \quad \text{for } k = 2 \qquad (3.8a)$$

$$(z-1)(z+r_1) + zb_1 s_0 = (z+p_1)(z+t_1) \quad \text{for } k = 1 \qquad (3.8b)$$

Equating coefficients of equal powers of z in (3.8a) gives

$$z^1 : r_1 - 1 = p_1 + t_1$$

$$z^0 : -r_1 + b_1 s_0 = p_1 t_1$$

Hence

$$\left.\begin{aligned} r_1 &= 1 + p_1 + t_1 \\ s_1 &= (r_1 + p_1 t_1) / b_1 \end{aligned}\right\} \qquad (3.9a)$$

From (3.8b) we have

$$\left.\begin{aligned} r_1 &= -p_1 t_1 \\ s_0 &= (p_1 + t_1 + 1 - p_1 t_1) / b_1 \end{aligned}\right\} \qquad (3.9b)$$

In the particular case (3.7) the coefficients r_2 and s_2 in the control law are thus equal to zero.

<u>The Process Zero is Cancelled</u>

If the process zero is cancelled $b_1 z + b_2$ must divide $z^2 + r_1 z + r_2$. Hence

$$z^2 + r_1 z + r_2 = (z+b_2/b_1)(z+r_1')$$

The design equation then becomes

$$(z-a)(z-1)(z+r_1') + b_1(s_0 z + s_1) = z(z+p_1)(z+t_1) \quad \text{for } k = 2 \qquad (3.10a)$$

$$(z-a)(z-1) + b_1(s_0 z + s_1) = (z+p_1)(z+t_1) \quad \text{for } k = 1 \qquad (3.10b)$$

Equating coefficients of equal powers of z in (3.10a) gives

$$z^2 : -1 - a + r_1' = p_1 + t_1$$

$$z^1 : -(1+a)r_1' + a + b_1 s_0 = p_1 t_1$$

$$z^0 : a r_1' + b_1 s_1 = 0$$

This linear equation has the solution

$$\left.\begin{aligned} r_1' &= p_1 + t_1 + a + 1 \\ s_0 &= [(1+a)r_1' - a + t_1 p_1] / b_1 \\ s_1 &= -a r_1' / b_1 \end{aligned}\right\} \qquad (3.11a)$$

The remaining parameters of the regulator are then given by

$$r_1 = -b_2 / b_1 + r_1'$$

$$r_2 = r_1' b_2 / b_1$$

For k = 1 we have

$$\left.\begin{aligned} r_1 &= b_2 / b_1 \\ r_2 &= 0 \\ s_0 &= (1 + a + t_1 + p_1) / b_1 \\ s_1 &= (t_1 p_1 - a) / b_1 \end{aligned}\right\} \qquad (3.11b)$$

Regulator Design for B Polynomials of higher order

When the sampling period h is smaller than time delay T_d the the degree of polynomial B in the estimated model

$$A(z)y(t) = B(z)u(t) + e(t) \qquad (3.12)$$

is required such that

$$\deg B \geq 1 + T_d \text{ div } h \qquad (3.13)$$

If the time delay is unknown the safe way to hand the control problem is to us a polynomial B of high order. As an example we consider a process with two extra terms of B as follows.

$$y(t) = a\,y(t-h) + b_1 u(t-h) + b_2 u(t-2h) + b_3 u(t-3h) + b_4 u(t-4h) \qquad (3.14)$$

The linear regulator is given by

$$R(z)u(t) = T(z)y_r(t) - S(z)y(t) \qquad (3.15)$$

where observer $T(z) = t_0 T_1(z)$

$$t_0 = \begin{cases} p_s / b_s & \text{if no common factor} \\ p_s / (b_1 + b_{2c} + b_{3c}) & \text{for a common factor} \end{cases}$$

$$b_1 z^3 + b_2 z^2 + b_3 z + b_4 = (z-a)(b_1 z^2 + b_{2c} z + b_{3c})$$

and R, S and T satisfy the basic design equation

$$A(z)\nabla R(z) + B(z)S(z) = P(z)T_1(z) \qquad (3.16)$$

equating coefficients with same power of z (3.16) can be rewritten as a parameter equation

$$D\Theta = C \qquad (3.17)$$

When there is no cancellation (3.17) has the form

$$\Theta = [\, r_1 \quad r_2 \quad r_3 \quad s_0 \quad s_1 \,]^T$$

$$D = \begin{bmatrix} 1 & 0 & 0 & b_1 & 0 \\ -a-1 & 1 & 0 & b_2 & b_1 \\ a & -a-1 & 1 & b_3 & b_2 \\ 0 & a & -a-1 & b_4 & b_3 \\ 0 & 0 & a & 0 & b_4 \end{bmatrix}$$

$$C = [\, t_1 + p_1 + 1 + a \quad t_1 p_1 - a \quad 0 \quad 0 \quad 0 \,]^T$$

If there is a common factor then (3.17) becomes

$$\Theta = [\, r_1 \quad r_2 \quad s_0 \,]^T$$

$$D = \begin{bmatrix} 1 & 0 & b_1 \\ -1 & 1 & b_{2c} \\ 0 & -1 & b_{3c} \end{bmatrix}$$

$$C = [\, t_1 + p_1 + 1 \quad t_1 p_1 \quad 0 \,]^T$$

The control signal is given by

$$u(t) = t_0 yr(t) + t_0 t_1 y(t-h) - s_0 y(t) - s_1 y(t-h)$$
$$+ (1-r_1)u(t-h) + (r_1-r_2)u(t-2h) + (r_2-r_3)u(t-3h) + r_3 u(t-4h)$$

The test value

$$T_{com} = eps\ (b_{max})^4$$

is used to determine whether there is a common factor.

4. PARAMETER ESTIMATION

To adjust the sampling period automatically an extra parameter of polynomial B is also estimated. The model used is thus

$$y(t+1) = ay(t) + b_1 u(t) + b_2 u(t-1) + b_3 u(t-2) \qquad (4.1)$$

where the time unit is chosen as the sampling period h. To describe the algorithm the following notation is introduced

$$\varphi(t) = [\ y(t-1)\ u(t-1)\ u(t-2)\ u(t-3)\ ]$$

$$\theta(t) = [\ \hat{a}(t)\quad \hat{b}_1(t)\quad \hat{b}_2(t)\quad \hat{b}_3(t)\ ]^T$$

The recursive least squares estimate of the parameters is then given by

$$\begin{aligned}
\theta(t+1) &= \theta(t) + P(t+1)\varphi^T(t+1)\varepsilon(t+1)\\
P(t+1) &= [\ P(t) - P(t)\varphi(t+1)R^{-1}(t+1)\varphi^T(t+1)P(t)\ ] / \lambda\\
R(t) &= \lambda + \varphi(t+1)P(t)\varphi^T(t+1)\\
\varepsilon(t+1) &= y(t+1) - \varphi(t+1)\theta(t)
\end{aligned} \qquad (4.2)$$

where λ is the forgetting factor.

Because the bias of the process is not estimated the differences of measurement between two neighbourhood are provided to estimation. A filter is also used to improve the quality of signals

$$\left.\begin{aligned}
yf(t) &= t_1 yf(t-1) + y(t)\\
uf(t) &= t_1 uf(t-1) + u(t)
\end{aligned}\right\} \qquad (4.3)$$

$\varphi(t)$ and $\varepsilon(t)$ are repleced by

$$\left.\begin{aligned}
\varphi(t) &= [\ yf(t-2)-yf(t-1)\quad uf(t-1)-uf(t-2)\\
&\qquad uf(t-2)-uf(t-3)\quad uf(t-3)-uf(t-4)\ ]\\
\varepsilon(t) &= yf(t) - yf(t-1) - \varphi(t)\theta(t-1)
\end{aligned}\right\} \qquad (4.4)$$

An instrumental variable method has also been tested to avoid the bias problem.

5. ADJUSTING THE SAMPLING PERIOD

The sampling period h is a critical parameter. It is clearly desirable to adjust h automatically. The principles for adjusting h are discussed in this section. A block diagram of the self-tuner with automatic adjustment of the sampling period is shown in Fig.1. A simulation result is given by Fig.2.

The Case of Known Parameters

Since the regulator R2 is based on the model (2.3) the sampling interval h should be smaller than the time delay T_d. If the parameter b_2 is different from zero the sampling period should also be larger than $T_d / 2$. We thus obtain the following rule for choosing the sampling period

$$0.5\, T_d < h < T_d \qquad (5.1)$$

If the regulator R2 should be used the ratio between the time delay T_d and the time constant T should not be too small. A reasonable condition is

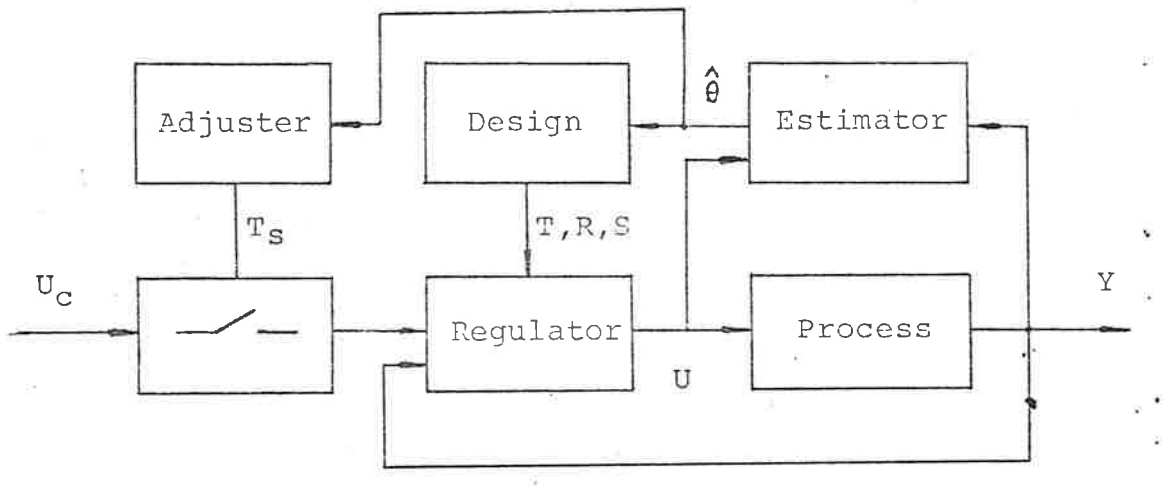


Fig 1. Structure of a self-tuner with automatic adjustment of sampling period.

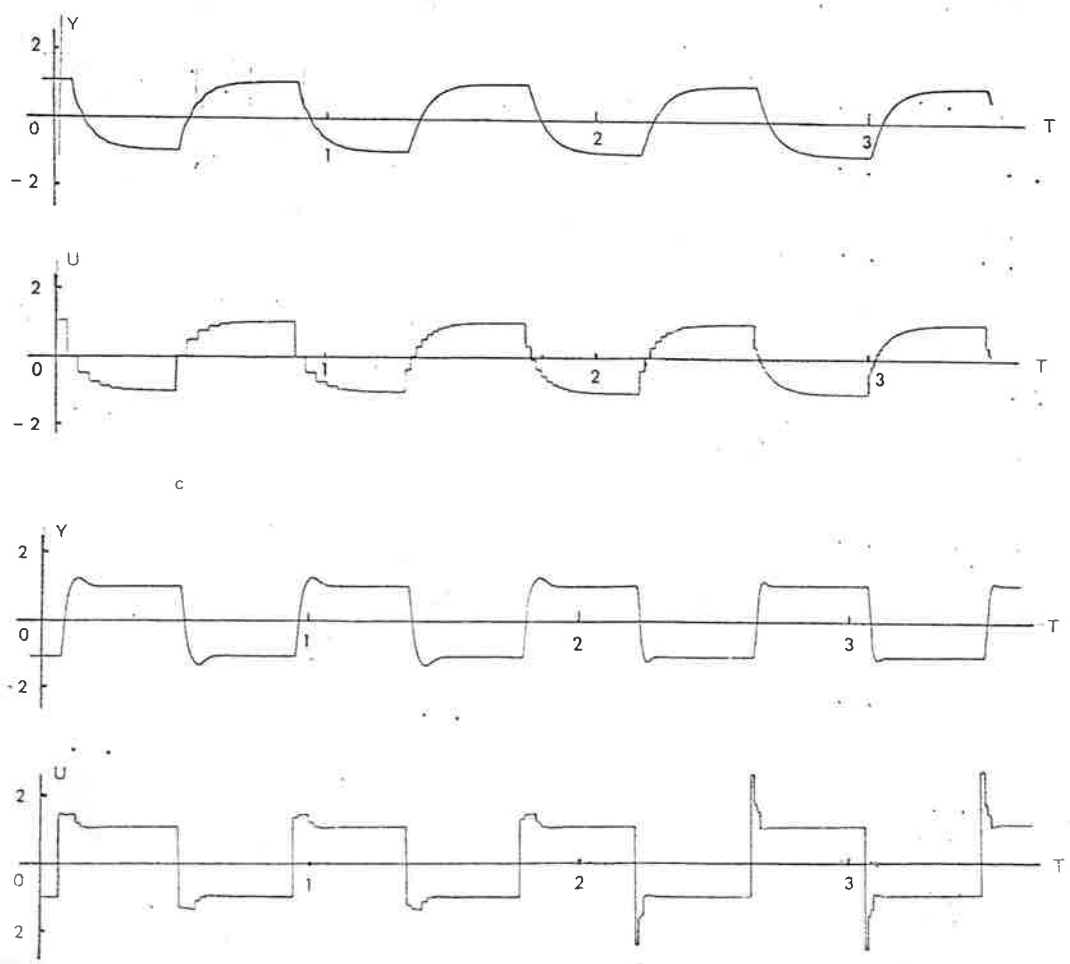


Fig 2. Automatic adjustment of the sampling period by a simple self-tuner. Process exp(-0.4s)/(s+1). The sampling period is adjusted from 2.4 to 1.2, then to 0.6 s.

A. Desired pole p = 0.5.

B. Desired pole p = 5.

$$T_d > 0.2\ T \qquad (5.2)$$

If this equality is not satisfied it is better to use some other design method. Some methods for automatic adjustment of the sampling period are given below.

Automatic Adjustment 1

It is natural to use the inequality (5.1) as a basis for an automatic adjustment of the sampling period. A reasonable choice is thus

$$h = 0.75\ \hat{T}_d \qquad (5.3)$$

If a reliable estimate of the time delay can be produced it is thus straightforward to determine a suitable value of the sampling period. If good apriori estimates are provide the time delay can be estimated from (2.7). It is, however, not

entirely trivial to provide a good estimate of time delay if the initial estimate of h is far from the true value. To see this let w be the impulse response of the system and let $\{ w_i , i = 0, 1, 2, \ldots \}$ be the pulse response. It follows that

$$w_i = \int_{(i-1)h}^{ih} w(t)\, dt$$

$$= \begin{cases} 0 & i = 0 \\ b_0 & i = 1 \\ b_1 + ab_0 & i = 2 \\ a^{i-3}(b_2+ab_1+a^2b_0) & i >= 3 \end{cases} \qquad (5.4)$$

For a system with a time delay these equations imply that

$$b_0 = 0 \qquad \text{if } h <= T_d$$

$$b_0 = b_1 = 0 \qquad \text{if } h <= T_d / 2$$

$$b_0 = b_1 = b_2 = 0 \qquad \text{if } h <= T_d / 3$$

Since the least squares estimates are asymptotically consistent it follows that the estimates of the parameters b_1, b_2 and b_3 will all be zero if the initial value of the sampling period h is smaller than $T_d / 3$. In that case the estimates can not be used to estimate the time delay. However, if the initial value of the sampling period is larger than $T_d / 3$ the time delay can be estimated from (2.7). Several different cases can be distinguished.

Case 1: $| b_1 | > \alpha \max(| b_2 |, | b_3 |)$

$$T_d = - \frac{\ln (b_2+ab_1) / (b_1+ b_2)}{\ln a} h \qquad (5.5)$$

Case 2: $| b_1 | < \alpha \max(| b_2 |, | b_3 |)$

$| b_2 | > \alpha | b_3 |$

$$T_d = \left[1 - \frac{\ln (b_3+ab_2) / (b_2+ b_3)}{\ln a} \right] h \qquad (5.6)$$

Case 3: If all parameters b_1, b_2, and b_3 are small the time delay can not be easily estimated. Neither is it easy to find good test quantities. One possibility is to increase the number of estimated parameters. If an apriori estimate k of the DC-gain is available the variable

$$\frac{b_1+ b_2+ b_3}{k(1-a)}$$

could be used to test if $h < T_d / 3$. A third possibility is to use the observation that if the sampling interval is too small then the estimated parameters b_1, b_2 and b_3 will be small. The process gain will than be very large and the closed loop system will be unstable. If the control signal is bounded the system will then have a limit cycle which could be detected. A simple describing function analysis indicates that the period of the limit cycle will be between $2 T_d$ and $4 T_d$. A determination of the frequency of the limit cycle will thus provide an estimate of T_d. The automatic adjustment procedure can then be summed up as follows.

Step_1. Choose a sufficient large sampling period $h > T_d / 2$. Run the self-tuner based on

$$Ay = Bu$$

$$AR + BS = PT_1$$

Step_2. When the parameters have converged. determine the time delay from estimates with an extra parameter of B. Use (5.5) or (5.6).

Step_3. Adjust h to T_s

$$T_s = \alpha T_d$$

where α is a constant. Before the adjustment, update $\hat{\theta}(h)$ to $\hat{\theta}(T_s)$ according to (2.6).

Step_4. Keep T_s in an admissible range $[T_{sl}, T_{sh}]$, where

$T_{sl} = \alpha_1 T_d$

$T_{sh} = \alpha_2 T_d$

α_1 and α_2 are constant.

Automatic_Adjustment_2

If the choice of the sampling period is not too strict it is unnecessary to compute the time delay and the adjustment can be arranged to double or half the sampling period. The procedure is given by

Step_1. Choose a sufficient large sampling period $h > T_d / 2$.

Run a self-tuner with estiomation of two different models.

$A_1 y = B_1 u$

$A_2 y = B_2 u$

$A_1 R + B_1 S = PT_1$

where

A_1 and B_1 are estimated with sampling period h and

A_2 and B_2 are estimated with sampling period h/2.

Step_2. Determination of the range of T_d.

Case 1. When $h > 2T_d$ then

$\min [b_1, b_2, b_3]_h = b_3(h)$ and

$\min [b_1, b_2, b_3]_{h/2} = b_3(h/2)$

Choose $T_s = h/2$.

Case 2. When $2T_d > h > T_d$ then

$\min [b_1, b_2, b_3]_h = b_3(h)$ and

$\min [b_1, b_2, b_3]_{h/2} = b_1(h/2)$

Keep $T_s = h$.

Case 3. When $T_d > h$ then

$\min [b_1, b_2, b_3]_h = b_1(h)$

Choose $T_s = 2h$.

Step_3. Adjusting the sampling period.

If case 1 or case 2 repeats n consequtive times a decision of decreasing or increasing the sampling period will be given. Before the adjustment update $\hat{\theta}(h)$ to $\hat{\theta}(T_s)$. In case 1 $\hat{\theta}(h/2)$ has been estimated the adjustment will be very smooth.

In case 2 the value $\hat{\theta}(2h)$ must be calculated. The formulae is given as follows. Consider a least squares model for sampling period h

$$y = \varphi(h)\theta(h) \tag{5.7}$$

Then the model for sampling period 2h is given as

$$y = \varphi(2h)\theta(2h) + f(\theta(h), \nabla u) \tag{5.8}$$

where ∇u is the variation of control signal in 2h. The proof of (5.8) is straight forward but tedious. It is given elsewhere. Let it suffice here to show the formula in a special case only. Consider the process with sampling period h

$$y(t+2) = -ay(t+1) + b_1 u(t-k+2) + b_2 u(t-k+1)$$

For $2h > T_d$ we have

$$y(t+2) = \varphi(2h)\theta(2h) + \nabla\varphi(h)C(h)$$

where

$$\theta(2h) = [\, a(2h) \quad b_1(2h) \quad b_2(2h) \,]^T$$

$$\varphi(2h) = [\, -y(t) \quad u(t) \quad u(t-2) \,]$$

$$C(h) = [\, c_1(h) \quad c_2(h) \,]^T$$

$$\nabla\varphi(h) = [\, u(t+1)-u(t) \quad u(t-1)-u(t-2) \,]$$

Consider several different cases.

1). $T_d = 0$, i.e. $k = 1$, $b_2 = 0$. It results in

$$a(2h) = a^2$$
$$b_1(2h) = b_1(1-a)$$
$$b_2(2h) = 0$$
$$c_1(2h) = b_1$$
$$c_2(2h) = 0$$

2). $h > T_d$, i.e. $k = 1$. We have

$$a(2h) = a^2$$
$$b_1(2h) = -ab_1 + b_1 + b_2$$
$$b_2(2h) = -ab_2$$
$$c_1(2h) = b_1$$
$$c_2(2h) = -ab_1$$

3). $2h > T_d \geq h$, i.e. $k = 2$. then

$$a(2h) = a^2$$
$$b_1(2h) = b_1$$
$$b_2(2h) = -ab_2 - a_1b_1 + b_2$$
$$c_1(2h) = 0$$
$$c_2(2h) = -ab_1 + b_2$$

4). $h = T_d$, i.e. $k = 2$ and $b_2 = 0$. then

$$a(2h) = a^2$$
$$b_1(2h) = b_1$$
$$b_2(2h) = -ab_1$$
$$c_1(2h) = 0$$
$$c_2(2h) = -ab_1$$

6. SIMULATION AND EXPERIMENTS

The behaviour of the self-tuner is shown by simulations using SIMNON and by experiments on an analog computer and DEC LSI 11/03 using an interactive program written in Pascal.

Changing time constant

The first experiments illustrate how the self-tuner reacts to changes in the time constants of the process. Fig.3 shows the behaviour of a process without control and with self-tuning control. The time constant T of the desired closed loop system can be adjusted very well according to requirements, where T = 1/p.

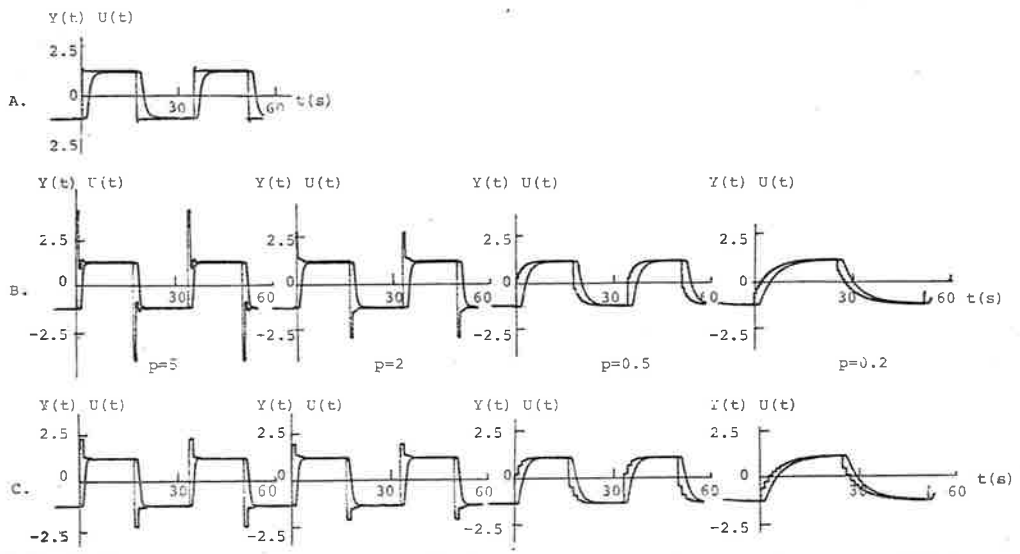


Fig 3. Changing time constant of a closed loop system by self-tuner.

Process exp(-0.9s)/(s+1).

A. without control.

B. self-tuner with different p, h = 0.7 s.

C. self-tuner with different p, h = 1.2 s.

Initial Conditions

When all initial values of estimates are set to zero the control signals in transient will reach its limits and the output changes drastically. See Fig.4. If reasonable initial values are given the transient is however also reasonable.

Load change

The self-tuner can tolerate large load disturbances. Fig.5 shows the performance of a self-tuner under load changes.

Automatic adjustment of sampling period

The performance of algorithm 1 is illustrated in Fig.5. The adjustment takes place in both case $h > T_d$ and case $h < T_d$.

Similar results for algorithm 2 are shown in Fig.6.

High order process dynamics

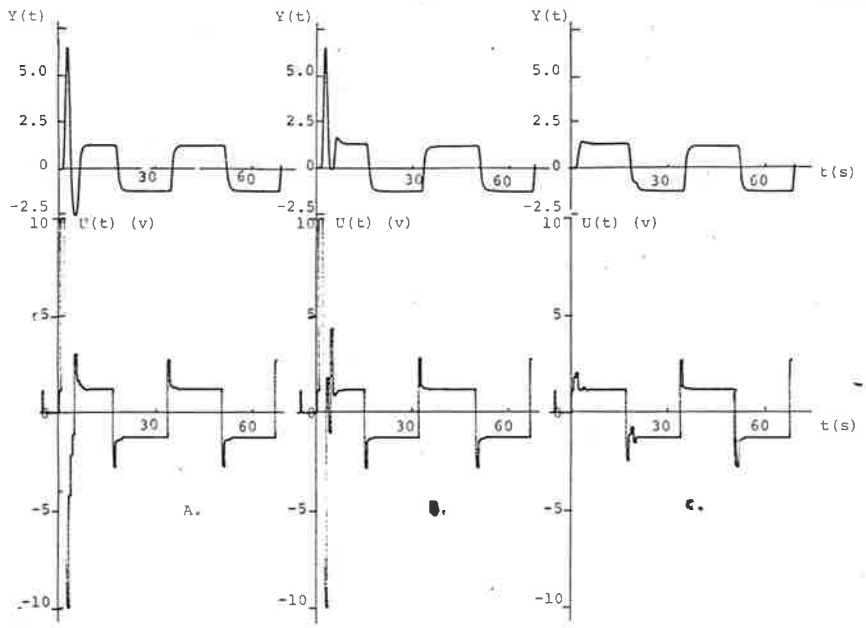


Fig 4. A self-tuner starts from different initial conditions. Process exp(0.9s)/(s+1), desired closed loop system p = 2.

A. θ=0.01, B. a= -0.5, b1 = b2 = b3 = 0.01,

C. a = -0.5, b1 = 0.01, b2 = 0.33, b3 = 0.14.

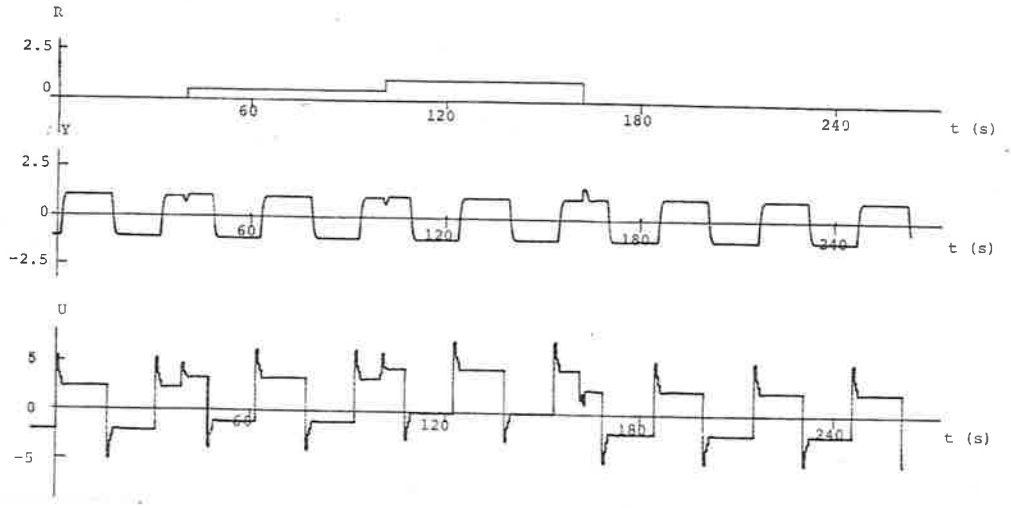


Fig_5. A self-tuner working under load (R) change. The Process transfer function is exp(0.4s)/(s+1). The parameters are p = 4, h = 0.6 s.

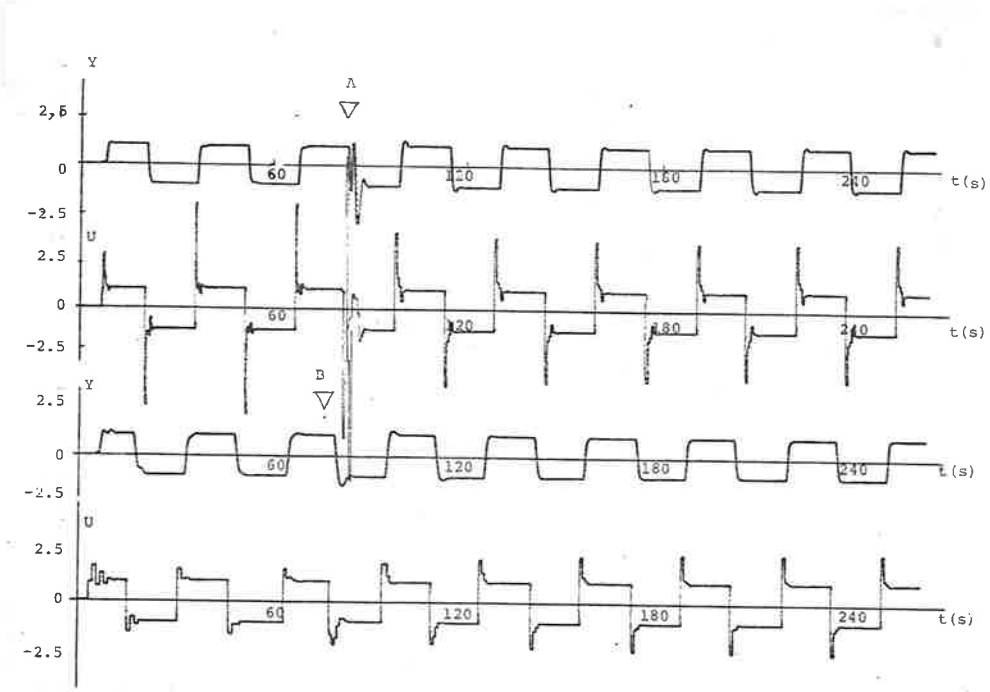


Fig_6. Simulation of algorithm 1 and a process with the transfer function exp(-0.5s)/(s+1).

A. The sampling period is adjusted automatically from 0.35 s. to 0.76 s. after 70 s.

B. The sampling period is adjusted automatically from 1.2 s. to 0.6 s. after 60 s.

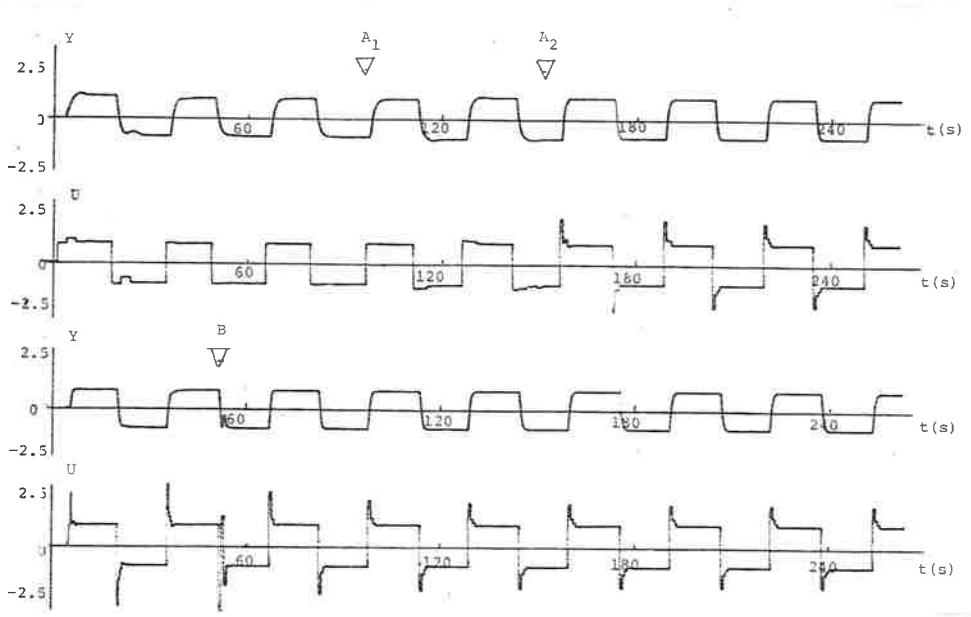
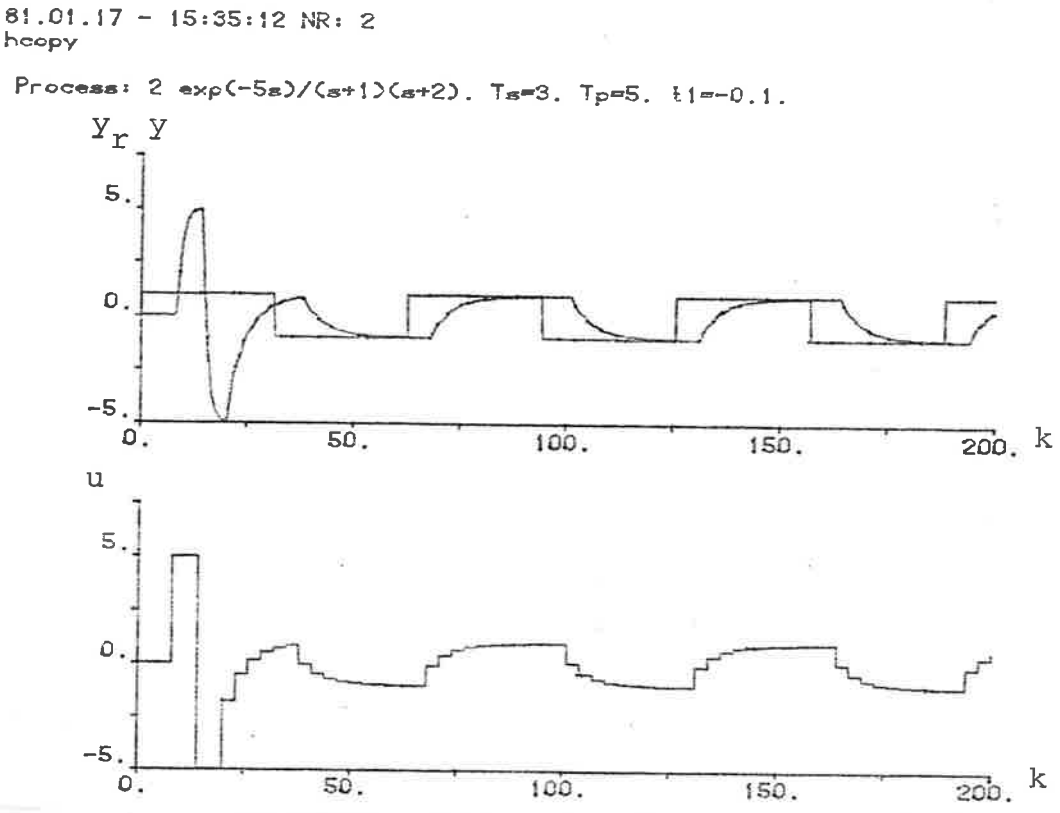


Fig_7. Simulation of algorithm 2 and a Process with the transfer function exp(-0.5s)/(s+1).

A. The sampling period is adjusted automatically from 2.8 to 1.4, then to 0.7 s.

B. The sampling period is adjusted automatically from 0.35 to 0.7 s.

The simple self-tuner has also been applied to processes having higher order dynamics. Systems with the transfer functions exp(-5s)/(s+1)(s+2) and exp(-5s)/(s+1)(s+2)(s+3) are simulated using SIMNON. Fig.8 gives some results.



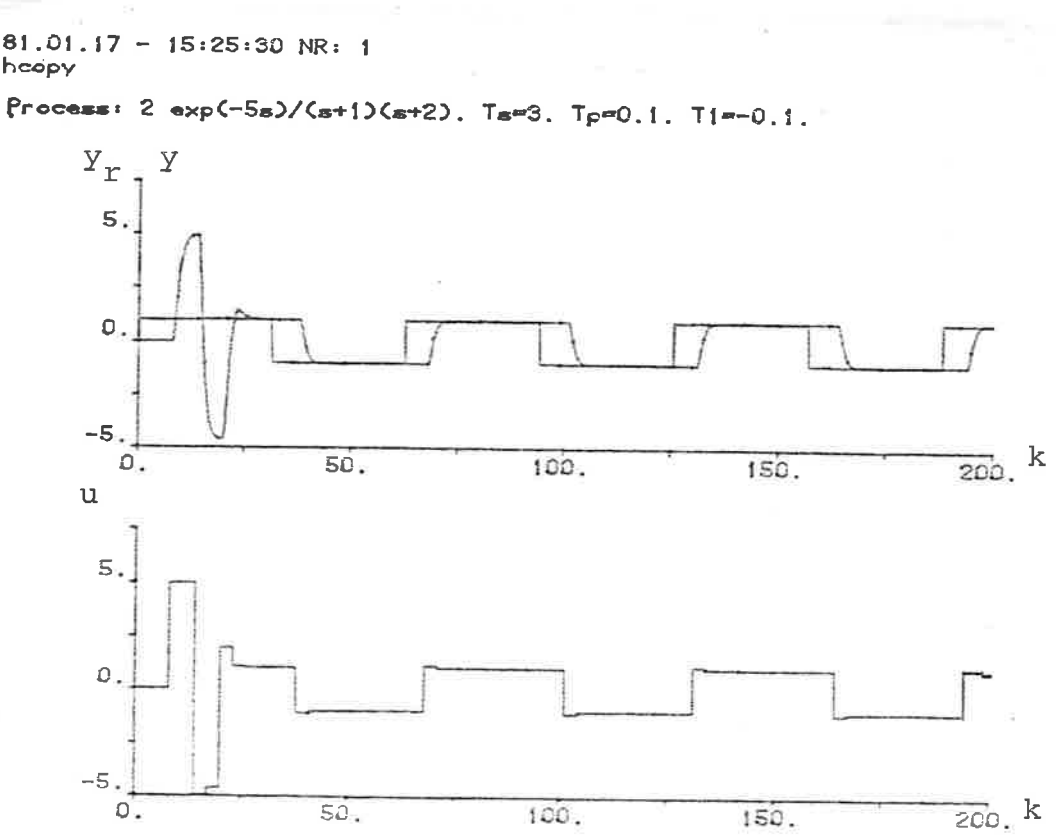


Fig_8. Simulation results for a process with the transfer function exp(-5s)/(s+1)(s+2).

7. REFERENCES

1.Horowitz I.M. (1963): Synthesis of Feedback Systems. Academic Press. New York.

2.Astrom K.J (1979): Simple Self-tuners (1). CODEN: LUTFD2 / (TFRT-7184) / 1-063 / (1979).

3.Wittenmark B, Hagander P. and Gustavsson I. (1980): STUPID Implementation of a Self-tuning PID-controller. CODEN: LUTFD2 / (TFRT-7201) / 1-030 / (1980).

4.Smith O. J. M. (1957): Control of Loops with Dead Time. Chemical Engineering Progress. 53 217-219.

5.Young P. C. (1970): An Instrumental Variable Method for Real-time Identification of a Noisy Process. Automatica. 6 271-287.

6.Kurz H. (1979): Digital Parameter-adaptive Control of Processes with Unknown Constant or Timevarying Dead Time. Proc. 5th IFAC Symposium on Identification and System Parameter Estimation. Darmstadf.

APPENDIX A: SIMNON PROGRAM

Self-tuner with Two More Extra Parameters of B.

```
CONTINUOUS SYSTEM PROC1

"Self-tuner for a first order process with a time delay.
"The algorithm is based on pole placement and the regulator
can hand the polynomial B with two more extra parameters.

INPUT  U
OUTPUT Y
TIME   T
STATE  X
DER    DX

"Process

y=if t<nk then x else (x+ak)
dx=(-x+k*u)/tc

tc:1
k:1
nk:1000
ak:0.5

END


DISCRETE SYSTEM REG1

INPUT  Y YR
OUTPUT U
TIME   T
TSAMP  TS
STATE  T1  T2  T3  T4  T5  F1  F2  F3  F4  F5
STATE  P11 P12 P13 P14 P15 P22 P23 P24 P25
STATE  P33 P34 P35 P44 P45 P55
STATE  YR1 Y1  U1  U2  U3  U4  YF1 UF1
NEW    NT1  NT2  NT3  NT4  NT5  NF1  NF2  NF3  NF4  NF5
NEW    NP11 NP12 NP13 NP14 NP15 NP22 NP23 NP24 NP25
NEW    NP33 NP34 NP35 NP44 NP45 NP55
NEW    NYR1 NY1  NU1  NU2  NU3  NU4  NYF1 NUF1

"Part 2 Regulator Design

p1=-exp(-dt/tp)
ps=1+p1
a1=t1
b1=t2
b2=t3
b3=t4
b4=t5
bs=b1+b2+b3+b4
bm1=max(abs(b1),abs(b2))
bm2=max(abs(b3),abs(b4))
bmax=max(bm1,bm2)
w1=a1-1
w3=tt1+p1-w1
w4=tt1*p1+a1
```

```
d12=b2-w1*b1
d22=b3+a1*b1
d23=b2+b4/a1
d33=b3+w1*b4/a1
n=-a1*(d22*(d33+a1*b1)-d23*(d12*a1+b4)+w1*(b1*b4-d33*d12))

"Test common factor

an=abs(n)
tcom=0.001*bmax*bmax*bmax

bc2=b2-a1*b1
bc3=b3-a1*bc2
nc=b1+bc2+bc3

t0=if an<tcom then ps/(b1+bc2+bc3) else ps/bs
dr11=w1-w4/w3
dr12=b2-b1*w4/w3
r11=-a1*w3*(b3*(d33+a1*b1)-d23*(dr12*a1+b4)+w1*(b1*b4-d33*dr12))
r12=(tt1+ps)*(bc3+bc2)-b1*tt1*p1
r1=if an>tcom then r11/n else r12/nc
s01=-a1*w3*(dr11*(w1*d33+a1*d23)+a1*(a1*b1+d33))
so2=tt1+ps+tt1*p1
s0=if an>tcom then s01/n else so2/nc
r21=a1*w3*(dr11*(d33*b3-d23*b4)-a1*(b1*b4-dr12*d33))
r2=if an>tcom then r21/n else bc3*s0
s1=if an>tcom then a1*w3*(dr11*(w1*b4+a1*b3)+a1*(a1*dr12+b4))/n else 0
r3=if an>tcom then b4*w3*(a1*(b4+a1*dr12)+dr11*(a1*b3+b4*w1))/n else 0

"Control signal

u01=(1-r1)*u1+(r1-r2)*u2+(r2-r3)*u3+r3*u4
u0=t0*yr+t0*tt1*yr1-s0*y-s1*y1+u01
u=if u0<ulow then ulow else if u0>uhigh then uhigh else u0

k0=if abs(s0+s1)>0 then t0*(1+tt1)/(s0+s1) else 0

"Part 1 Estimation

yf=-tt1*yf1+y
uf=-tt1*uf1+u
e=yf-yf1-t1*f1-t2*f2-t3*f3-t4*f4-t5*f5

"Estimator Gain

k1=p11*f1+p12*f2+p13*f3+p14*f4+p15*f5
k2=p12*f1+p22*f2+p23*f3+p24*f4+p25*f5
k3=p13*f1+p23*f2+p33*f3+p34*f4+p35*f5
k4=p14*f1+p24*f2+p34*f3+p44*f4+p45*f5
k5=p15*f1+p25*f2+p35*f3+p45*f4+p55*f5

d=1+f1*k1+f2*k2+f3*k3+f4*k4+f5*k5

"update Estimate

nt1=t1+k1*e/d
nt2=t2+k2*e/d
nt3=t3+k3*e/d
nt4=t4+k4*e/d
nt5=t5+k5*e/d
```

```
"Update Covariance

np11=(p11-k1*k1/d)/lam
np12=p12-k1*k2/d
np13=p13-k1*k3/d
np14=p14-k1*k4/d
np15=p15-k1*k5/d
np22=(p22-k2*k2/d)/lam
np23=p23-k2*k3/d
np24=p24-k2*k4/d
np25=p25-k2*k5/d
np33=(p33-k3*k3/d)/lam
np34=p34-k3*k4/d
np35=p35-k3*k5/d
np44=(p44-k4*k4/d)/lam
np45=p45-k4*k5/d
np55=(p55-k5*k5/d)/lam

"Update Fi-vector

nf1=yf1-yf
nf2=uf-uf1
nf3=f2
nf4=f3
nf5=f4
ny1=y
nu1=u
nu2=u1
nu3=u2
nu4=u3
nyr1=yr
nyf1=yf
nuf1=uf

"Update Sampling Time

ts=t+dt

dt:1.5
lam:0.98
t1:1.E-3
t2:1.E-3
t3:1.E-3
t4:1.E-3
t5:1.E-3
p11:100
p22:100
p33:100
p44:100
p55:100

tt1:0.2
ulow:-5
uhigh:5
tp:2

END
```

```
CONNECTING SYSTEM CONR1

TIME T

yref=sign(sin(a0*t))
yr[reg1]=yref
y[reg1]=y[proc1]
u[proc1]=y1[delay]
td1[delay]=t-td
u1[delay]=u[reg1]

a0:0.05
td:5

END
```

APPENDIX B: PROGRAM LISTING 1.

```
PROGRAM SELFTUNER2(1);

   { Author   Zhou Z.Y. and Astrom K.J
     Data     Nov. 1980.
     References
       Astrom K.J. and Zhou (1981): Self-tuners with Automatic
         Adjustment of the Sampling period for processes
         with time delays.
       Zhou Z.Y. (1980) A microcomputer implementation of
         datalogging and recursive least squares estimation.
       Wittenmark B., Hagander P. and Gustavsson I. (1980):
         STUPID Implementation of a self-tuning PID-controller.
       Mattsson S.E.(1978):A simple real-time schedurer.
         report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root  (SR)  estimation  and  several
     regulator calculations (RC) for simple  processes with
     time delay.  All  results  can  be  stored in files if
     respective logging command is  given.
     The  control  algorithm  based  on  explicit adaptive
     pole-zero-placement is described  in Astrom (1979).
     The time delay is calculated from estimated parameters
     and the sampling period is adjusted automatically.

     The program consists of these files:
     RLS   -- estimator.
     RC    -- regulator calculation.
     FG    -- foreground program.
     OPCOM -- operator communication and main program.   }

{ GLOBAL DATA DECLARATIONS }

const n=4;
      m=7;
      l=180;

type  specificationtype=record
        pole,tsamp,ta,ni,hhi,hlo,dampl,damph,lambda,po,
        lolim,hilim,aoa,boa,bob,boc:real;
        nd,ns,nta,ntb,np,yrchan,ychan,uchan,outchan:integer;
        log:boolean;
        end;
      estpartype=record
        th,fi:array[1..n] of real;
        p:    array[1..n,1..n] of real;
        end;
      regpartype=record
        t0,r1,r2,s0,s1:real;
        end;
      logpartype=record
        logth:    array[1..n,1..l] of real;
        logregpar:array[1..m,1..l] of real;
        loguy:    array[0..2,1..l] of real;
        end;

var   specif,compar:specificationtype;
      estpar:estpartype;
```

```
        regpar:regpartype;
        logpar:logpartype;
        i,j,k,s,w,period:integer;
        yr,yrold,y,yold,u,u1,uold1,uold2,uold3,td,t:real;
        yf,yfold,uf,ufold:real;
        newpar:boolean;
        delu:array[1..100] of real;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
Procedure estimation(y:real; var estpar:estpartype);
                                                        external;
Procedure regdesign(specif:specificationtype;estpar:estpartype;
                              var regpar:regpartype);external;
Procedure delayu;external;

{=============================================================FOREGROUND}
Procedure foreground;

   { The procedure FG is organized as follows:
     Adin    inputs yr(t) and y(t);
     Control calculates control signal u(t) from measurements
             and reglator parameters of given process;
     Daout   outputs u(t);
     Estimation estimates the parameters of the process;
     Regdesign  calculates the reglator parameters ;
     Display display parameters of estimation and reglator
             on screen;
     Logdata prepares data for store in files after running;
     delayu  delays the output of control signal;
     Updata  updates input and output for next step.          }


{-------------------------------------------------------------control}
Procedure control;
   { calculates control signals u(t) from measurements ( yr
     and y ),  reglator parameters ( t0, s0, s1, r1 )  and
     integral compensation ub. }
 var tpoly,spoly,rpoly:real;
 begin
   with regpar,specif,estpar do begin
     tpoly:=t0*yr+t0*ta*yrold;
     spoly:=s0*y+s1*yold;
     rpoly:=(1-r1)*uold1+(r1-r2)*uold2+r2*uold3;
     u1:=tpoly-spoly+rpoly;
     u:=u1;
     if u1<lolim then u:=lolim;
     if u1>hilim then u:=hilim;
   end
 end; { of control }

{-------------------------------------------------------------display}
Procedure display;
   { displays parameters of the process and the regulator
     on screen }
 begin
   with specif,estpar,regpar do begin
     if s=ns then begin
        writeln('point',k);
        for i:=1 to n do begin
            write('   ');   write(th[i]:8:4)
        end;
        writeln;
        write(' ');         write(t0:8:4);
        write('  ');        write(ta:8:4);
        write('  ');        write(r1:8:4);
        write('  ');        write(r2:8:4);
        write('  ');        write(s0:8:4);
        write('  ');        write(s1:8:4);
        write('  ');        write(u1:8:4);
        write('  ');        write(u:8:4);
```

```
        writeln;
        write(' ':3,'tsamp=',tsamp:8:4,' ':7);
        if (abs(td)<1000) and (abs(t)<1000) then
        writeln('td=',td:8:4,' ':8,'t=',t:8:4);
        writeln;
        s:=0
      end;
      s:=s+1
    end
  end; { of display }

{-------------------------------------------------------------logdata}
Procedure logdata;
    { prepares data and parameters from k=nta to k=ntb for
      store in files after running }
  begin
    with logpar,estpar,regpar do begin
      for i:=1 to n do logth[i,w]:=th[i];
      loguy[0,w]:=yr;       loguy[1,w]:=y;
      loguy[2,w]:=u;
      logregpar[1,w]:=t0; logregpar[2,w]:=r1;
      logregpar[3,w]:=r2; logregpar[4,w]:=s0;
      logregpar[5,w]:=s1; logregpar[6,w]:=u1;
      logregpar[7,w]:=u;  w:=w+1
    end
  end; { of logpar }

{-------------------------------------------------------------updata}
Procedure updata;
    { updates input and output preparing for next step }
  var b1,b2,bs,nxm:real;
  begin
    with estpar,specif do begin
      fi[4]:=fi[3];
      fi[3]:=fi[2];
      fi[1]:=yfold-yf;
      fi[2]:=uf-ufold;
      yrold:=yr;
      yfold:=yf;
      ufold:=uf;
      yold:=y;
      uold3:=uold2;
      uold2:=uold1;
      uold1:=u;
    end
  end; { of update }

{-----------------------------------------------CODE OF FOREGROUND}
{ CODE FOREGROUND }

begin
    with specif do begin
      yr:=adin(yrchan);
      y:= adin(ychan);
      control;
      daout(outchan,u);
      delayu;
      estimation(y,estpar);
      regdesign(specif,estpar,regpar);
      display;
      if log then
```

```
         if (k)=nta) and (k(=ntb) then logdata;
      updata;
      k:=k+1;
      if newpar then begin
         specif:=compar;
         period:=round(50*tsamp);
         newpar:=false
      end
   end
end; { of foreground }
```

```
PROGRAM SELFTUNER2(1);

   { Author   Zhou Z.Y. and Astrom K.J
     Data     Nov. 1980.
     References
       Astrom K.J. and Zhou (1981): Self-tuners with Automatic
         Adjustment of the Sampling period for processes
         with time delays.
       Zhou Z.Y. (1980) A microcomputer implementation of
         datalogging and recursive least squares estimation.
       Wittenmark B., Hagander P. and Gustavsson I. (1980):
         STUPID Implementation of a self-tuning PID-controller.
       Mattsson S.E.(1978):A simple real-time schedurer.
         report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root (SR)  estimation  and  several
     regulator calculations (RC) for simple  processes with
     time delay.  All  results  can  be  stored in files if
     respective logging command is  given.
     The  control  algorithm  based  on  explicit adaptive
     pole-zero-placement is described  in Astrom (1979).
     The time delay is calculated from estimated parameters
     and the sampling period is adjusted automatically.

     The program consists of these files:
     RLS    -- estimator.
     RC     -- regulator calculation.
     FG     -- foreground program.
     OPCOM -- operator communication and main program.    }

{ GLOBAL DATA DECLARATIONS }

const n=4;
      m=7;
      l=180;

type  specificationtype=record
        pole,tsamp,ta,ni,hhi,hlo,dampl,damph,lambda,po,
        lolim,hilim,aoa,boa,bob,boc:real;
        nd,ns,nta,ntb,np,yrchan,ychan,uchan,outchan:integer;
        log:boolean;
        end;
      estpartype=record
        th,fi:array[1..n] of real;
        p:    array[1..n,1..n] of real;
        end;
      regpartype=record
        t0,r1,r2,s0,s1:real;
        end;
      logpartype=record
        logth:    array[1..n,1..l] of real;
        logregpar:array[1..m,1..l] of real;
        loguy:    array[0..2,1..l] of real;
        end;

var   specif,compar:specificationtype;
      estpar:estpartype;
      regpar:regpartype;
      logpar:logpartype;
      i,j,k,s,w,period:integer;
```

```
      yr,yrold,y,yold,u,u1,uold1,uold2,uold3,td,t:real;
      yf,yfold,uf,ufold:real;
      newpar:boolean;
      delu:array[1..100] of real;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
{=============================================================OPCOM}
Procedure Opcom;

   { The procedure OPCOM is organized as follows:
     Initialize    recognizes identifiers and initializes:
        initestpar initializes estimated parameters;
        initregpar initializes reglator parameters;
        inituy     initializes input and output;
        initspecif gives specifications;
        desiredpol calculates desired polynomial.
     Help          lists available commands.
     Par           modefies parameters:
        getiden    reads the identifiers;
        get        reads parameter if it is a real;
        take       reads parameter if it is a integer;
        askboolean reads command if it is a boolean;
        initestpar
        desiredpol
        error      gives a sensible orror message if in error.
     Run           runs the selftuner.
     Stop          stops the selftuner.
     Disp          display the current parameters.
     Store         logs inputs and output.
        error
     Resultp       logs estimated parameters.
        error
     Resultu       logs reglator parameters.
        error
     Exit          stops the whole program and exits it into computer.}

{ DECLARATIONS OF OPCOM }

label 999;
const idlength=8;
      blank=' ';
type  identifiertype=array[1..idlength] of char;
      opindex=(xhelp,xpar,xrun,xstop,xdisp,xstore,xresultp,
               xresultu, xexit,xlastop);
      asindex=(spole,stsamp,sdampl,sdamph,snp,sni,slambda,
               spo,sta,shhi,shlo,slolim,shilim,snd,sns,snta,
               sntb,saoa,sboa,sbob,sboc,slog,syrchan,sychan,
               suchan,soutchan,slastas);
      errors=(fewarg,manyarg,noname,prierr,illfile,nolog);
var   identifier:identifiertype;
      operation: array[opindex] of identifiertype;
      assignment:array[asindex] of identifiertype;
      xop:opindex;
      sas:asindex;
      ch: char;
      logdata,logtheta,logreg:boolean;

{ PROCEDURE FOR OPCOM }

{-------------------------------------------------------------help}
Procedure help;
   { lists available commands and information on screen }
 begin
   for i:=1 to 5 do writeln;
   writeln('         Simple Self-tuner');
   writeln;
```

```
   writeln('The available commands are as follows:');
   writeln;
   write(' ':4,'DISP');          writeln(' ':12,'STORE');
   write(' ':4,'PAR');           writeln(' ':13,'RESULTP');
   write(' ':4,'RUN');           writeln(' ':13,'RESULTU');
   write(' ':4,'STOP');          writeln(' ':12,'EXIT');
   writeln;
   writeln('The parameters can be changed:');
   writeln;
   writeln(' ':4,'POLE,TSAMP,TA,NI,HHI,HLO,LAMDA,PO,
    ND,NS,NTA,NTB,NP,DAMPL,DAMPH,LOLIM,HILIM.');
   writeln;
   writeln('The initial values can be assigned:');
   writeln;
   writeln(' ':4,'AOA,BOA,BOB,BOC.');
   writeln;
   writeln('The channels can be chosen:');
   writeln;
   writeln(' ':4,'YRCHAN,YCHAN,UCHAN,OUTCHAN'); writeln
 end; { of help }

{-------------------------------------------------------------error}
Procedure error(err:errors);
   { gives a sensible error message }
 begin
   case err of
     fewarg: writeln('too few arguments');
     manyarg:writeln('too many arguments');
     prierr: writeln(identifier,':illegal value');
     noname: writeln(identifier,':illegal argument');
     nolog:  writeln(identifier,':log do not be reguired');
     illfile:writeln(identifier,':ill file')
   end;
   goto 999
 end; { of error }

{-----------------------------------------------------initestpar}
Procedure initestpar;
   { initializes th, fi and p }
 begin
   with compar,estpar do begin
     for i:=1 to n do
         for j:=1 to n do
             if i=j then p[i,j]:=po else p[i,j]:=0.0;
     th[1]:=aoa;   th[2]:=boa;
     th[3]:=bob;   th[4]:=boc;
   end;
   k:=1;       s:=1;
   w:=1;
 end; { of initestpar }

{-------------------------------------------------------------inituy}
Procedure inituy;
   { initializes input and output }
 begin
   with estpar do
     for i:=1 to n do fi[i]:=0.0;
   for i:=1 to 20 do delu[i]:=0.0;
   yrold:=0.0;
   yfold:=0.0;     ufold:=0.0;
   yold:=0.0;      uold1:=0.0;
```

```
    uold2:=0.0;    uold3:=0.0;
    daout(0,0.0);  daout(1,0.0)
  end; { of inituy }

{-------------------------------------------------------initspecif}
Procedure initspecif;
    { assigns specification }
  begin
    with compar,estpar do begin
      pole:=2;         tsamp:=1.2;
      ni:=0.001;       np:=50;
      lambda:=0.98;    po:=100;
      ta:=-0.05;       hhi:=0.78;
      hlo:=0.72;       dampl:=0.2;
      damph:=0.9;      lolim:=-1;
      hilim:=1;        nta:=1;
      ntb:=180;        ns:=50;
      aoa:=-0.01;      boa:=0.01;
      bob:=0.01;       boc:=0.01;
      log:=false;      yrchan:=0;
      ychan:=1;        uchan:=2;
      outchan:=1;      nd:=5;
    end
  end; { of initspecif }

{-------------------------------------------------------initregpar}
Procedure initregpar;
    { initializes reglator parameters }
  begin
    with regpar do begin
      t0:=1.0;
      r1:=0.0;       r2:=0.0;
      s0:=0.0;       s1:=0.0;
    end
  end;

{-------------------------------------------------------initialize}
Procedure initialize;
    { recognizes identifiers and initializes }
  begin
    operation[xhelp]:=     'HELP    ';
    operation[xdisp]:=     'DISP    ';
    operation[xpar]:=      'PAR     ';
    operation[xrun]:=      'RUN     ';
    operation[xstop]:=     'STOP    ';
    operation[xexit]:=     'EXIT    ';
    operation[xstore]:=    'STORE   ';
    operation[xresultp]:=  'RESULTP ';
    operation[xresultu]:=  'RESULTU ';
    assignment[spole]:=    'POLE    ';
    assignment[stsamp]:=   'TSAMP   ';
    assignment[sdampl]:=   'DAMPL   ';
    assignment[sdamph]:=   'DAMPH   ';
    assignment[snp]:=      'NP      ';
    assignment[sni]:=      'NI      ';
    assignment[slambda]:=  'LAMBDA  ';
    assignment[spo]:=      'PO      ';
    assignment[sta]:=      'TA      ';
    assignment[shhi]:=     'HHI     ';
    assignment[shlo]:=     'HLO     ';
    assignment[slolim]:=   'LOLIM   ';
```

```
    assignment[shilim]:=  'HILIM   ';
    assignment[snd]:=     'ND      ';
    assignment[sns]:=     'NS      ';
    assignment[snta]:=    'NTA     ';
    assignment[sntb]:=    'NTB     ';
    assignment[saoa]:=    'AOA     ';
    assignment[sboa]:=    'BOA     ';
    assignment[sbob]:=    'BOB     ';
    assignment[sboc]:=    'BOC     ';
    assignment[slog]:=    'LOG     ';
    assignment[syrchan]:= 'YRCHAN  ';
    assignment[sychan]:=  'YCHAN   ';
    assignment[suchan]:=  'UCHAN   ';
    assignment[soutchan]:='OUTCHAN ';
    with compar do begin
      initspecif;
      initestpar;
      initregpar;
      inituy;
      specif:=compar;
      newpar:=true
    end;
  end; { of initialize }

{-------------------------------------------------------skipblank}
Procedure skipblank;
  begin
    repeat read(ch) until (ch<>blank) or eoln
  end; { of skipblank }

{-------------------------------------------------------getident}
Procedure getident;
    { reads a identifier from the terminal }
  begin
    for i:=1 to idlength do identifier[i]:=blank;
    skipblank;
    i:=1;
    while (ch>='A') and (ch<='Z') and (i<9) and not eoln do
    begin identifier[i]:=ch;
          i:=i+1;
          read(ch)
    end;
    if (ch>='A') and (ch<='Z') then identifier[i]:=ch
  end; { of getident }

{-----------------------------------------------------------take}
Procedure take(var rn:real);
    { reads a number if it is a real }
  begin
    if ch=blank then read(rn);
    if (rn<-1000) or (rn>1000) then error(prierr) else read(ch);
    if ch<>',' then read(ch)
  end; { of take }

{-----------------------------------------------------------get}
Procedure get(var inn:integer);
    { reads a number if it is a integer }
  begin
    if ch=blank then read(inn);
    if inn<0 then error(prierr) else read(ch);
    if ch<>',' then read(ch)
```

```
 end; { of get }

{-------------------------------------------------------askboolean}
Procedure askboolean(var command:boolean);
   { reads a command if it is a boolean }
 var comname:array[1..idlength] of char;
 begin
     comname:=identifier;
     if ch=blank then getident;
     if identifier='TRUE    ' then command:=true
     else command:=false;
     identifier:=comname
 end; { of askboolean }

{-----------------------------------------------------------------par}
Procedure par;
   { updates parameters }
 var pp:specificationtype;
 begin
   if eoln then error(fewarg);
   pp:=compar;
   repeat getident;
     assignment[slastas]:=identifier;
     sas:=spole;
     while assignment[sas]<>identifier do sas:=succ(sas);
     case sas of
       spole:   take(pp.pole);
       stsamp:  take(pp.tsamp);
       sdampl:  take(pp.dampl);
       sdamph:  take(pp.damph);
       snp:     get(pp.np);
       sni:     take(pp.ni);
       slambda: take(pp.lambda);
       spo:     take(pp.po);
       sta:     take(pp.ta);
       shhi:    take(pp.hhi);
       shlo:    take(pp.hlo);
       slolim:  take(pp.lolim);
       shilim:  take(pp.hilim);
       snd:     get(pp.nd);
       sns:     get(pp.ns);
       snta:    get(pp.nta);
       sntb:    get(pp.ntb);
       saoa:    take(pp.aoa);
       sboa:    take(pp.boa);
       sbob:    take(pp.bob);
       sboc:    take(pp.boc);
       syrchan: get(pp.yrchan);
       sychan:  get(pp.ychan);
       suchan:  get(pp.uchan);
       soutchan:get(pp.outchan);
       slog:    askboolean(pp.log);
       slastas: error(noname)
     end;
     writeln(identifier,'has been read')
   until eoln;
   newpar:=false;
   compar:=pp;
   with compar do begin
     initestpar;
   end;
```

```
   newpar:=true
 end; { of par }

{-------------------------------------------------------------disp}
Procedure disp;
   { displays current parameters }
 var rr:real;

 Procedure dataform(rr:real);
    { formats a real number }
  var w:integer;
  begin
    w:=rform(rr,6);  write(rr:10:w)
  end; { of dataform }

 begin { disp }
   with compar do begin
     writeln('   The current parameters are:');
     write('pole=   ');  dataform(pole);    write('   ');
     write('tsamp=  ');  dataform(tsamp);   writeln;
     write('nd=     ');  write(nd:6);       write(' ':7);
     write('ns=     ');  writeln(ns:6);
     write('np=     ');  write(np:6);       write(' ':7);
     write('nta=    ');  writeln(nta:6);
     write('ni=     ');  dataform(ni);      write('   ');
     write('ntb=    ');  writeln(ntb:6);
     write('lambda= ');  dataform(lambda);  write('   ');
     write('aoa=    ');  dataform(aoa);     writeln;
     write('po=     ');  dataform(po);      write('   ');
     write('boa=    ');  dataform(boa);     writeln;
     write('ta=     ');  dataform(ta);      write('   ');
     write('bob=    ');  dataform(bob);     writeln;
     write('hhi=    ');  dataform(hhi);;    write('   ');
     write('boc=    ');  dataform(boc);     writeln;
     write('hlo=    ');  dataform(hlo);     write('   ');
     write('yrchan= ');  writeln(yrchan:6);
     write('damph=  ');  dataform(damph);   write('   ');
     write('ychan=  ');  writeln(ychan:6);
     write('dampl=  ');  dataform(dampl);   write('   ');
     write('uchan=  ');  writeln(uchan:6);
     write('hilim=  ');  dataform(hilim);   write('   ');
     write('outchan=');  writeln(outchan:6);
     write('lolim=  ');  dataform(lolim);   write('   ');
     write('log=    ');  writeln(' ':4,log);
   end
 end; { of disp }

{-----------------------------------------------------------buildfile}
Procedure buildfile;
   { stores data of input-output, estimated parameters and
     regulator parameters in files respectively }
 var outfile:file of char;
     filename:array[1..14] of char;
     len:integer;
 begin
   with specif do
     if log=false then error(nolog) else begin
     read(filename);   len:=-1;
     rewrite(outfile,filename,'DAT',len);
     if len=-1 then error(illfile);
     with logpar do begin
```

```
        for w:=1 to (ntb-nta) do begin
          if logdata then  begin
             for i:=0 to 2 do write(outfile,loguy[i,w]:8:4);
             writeln(outfile)
          end;
          if logtheta then begin
             for i:=1 to n do write(outfile,logth[i,w]:8:4);
             writeln(outfile)
          end;
          if logreg then begin
             for i:=1 to m do write(outfile,logregpar[i,w]:8:4);
             writeln(outfile)
          end;
        end;
        close(outfile)
      end
    end
 end; { of buildfile }

{-------------------------------------------------------------------stop}
Procedure stop;
 begin
   period:=0; writeln(k:6,' data points have been collected.')
 end; { of stop }

{-----------------------------------------------------CODE OF OPCOM}
{ CODE OPCOM }

begin
   initialize;
   repeat
     write(')');
     getident;
     operation[xlastop]:=identifier;
     xop:=xhelp;
     while operation[xop]<>identifier do xop:=succ(xop);
     case xop of
          xhelp:   help;
          xdisp:   disp;
          xpar:    par;
          xrun:    period:=10;
          xstop,
          xexit:   stop;
          xstore:  begin logdata:=true;
                     logtheta:=false;    logreg:=false;
                     buildfile
                   end;
          xresultp:begin logtheta:=true;
                     logdata:=false;     logreg:=false;
                     buildfile
                   end;
          xresultu:begin logreg:=true;
                     logdata:=false;     logtheta:=false;
                     buildfile
                   end;
          xlastop: error(noname)
     end;
999: readln
   until xop=xexit
end; { of opcom }
```

```
{================================================CODE OF MAIN PROGRAM}
{ CODE MAIN PROGRAM }
Procedure foreground;external;

begin
   period:=0;
   clksave;
   schedule(foreground,period);
   opcom;
   clkrestore
end. { of main program }
```

```
PROGRAM SELFTUNER2(1);

   { Author   Zhou Z.Y. and Astrom K.J
     Data     Nov. 1980.
     References
       Astrom K.J. and Zhou (1981): Self-tuners with Automatic
         Adjustment of the Sampling period for processes
         with time delays.
       Zhou Z.Y. (1980) A microcomputer implementation of
         datalogging and recursive least squares estimation.
       Wittenmark B., Hagander P. and Gustavsson I. (1980):
         STUPID Implementation of a self-tuning PID-controller.
       Mattsson S.E.(1978):A simple real-time schedurer.
         report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root  (SR)  estimation  and  several
     regulator calculations (RC) for simple  processes with
     time delay.  All  results  can  be  stored in files if
     respective logging command is  given.
     The  control  algorithm  based  on  explicit adaptive
     pole-zero-placement is described  in Astrom (1979).
     The time delay is calculated from estimated parameters
     and the sampling period is adjusted automatically.

     The program consists of these files:
     RLS    -- estimator.
     RC     -- regulator calculation.
     FG     -- foreground program.
     OPCOM -- operator communication and main program.    }

{ GLOBAL DATA DECLARATIONS }

const n=4;
      m=7;
      l=180;

type  specificationtype=record
        pole,tsamp,ta,ni,hhi,hlo,dampl,damph,lambda,po,
        lolim,hilim,aoa,boa,bob,boc:real;
        nd,ns,nta,ntb,np,yrchan,ychan,uchan,outchan:integer;
        log:boolean;
        end;
      estpartype=record
        th,fi:array[1..n] of real;
        p:    array[1..n,1..n] of real;
        end;
      regpartype=record
        t0,r1,r2,s0,s1:real;
        end;
      logpartype=record
        logth:    array[1..n,1..l] of real;
        logregpar:array[1..m,1..l] of real;
        loguy:    array[0..2,1..l] of real;
        end;

var   specif,compar:specificationtype;
      estpar:estpartype;
      regpar:regpartype;
      logpar:logpartype;
      i,j,k,s,w,period:integer;
```

```
      yr,yrold,y,yold,u,u1,uold1,uold2,uold3,td,t:real;
      yf,yfold,uf,ufold:real;
      newpar:boolean;
      delu:array[1..100] of real;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
{=============================================================estimation}
Procedure estimation(y:real; var estpar:estpartype);
   { The procedure RLS calculates a Recursive Least Squares
     estimation for a first order process and the differences
     beteewn the filterred input and output signals are used.}
var r,e:real;
    fi1:array[1..n] of real;
begin
  with specif,estpar do begin
    yf:=-ta*yfold+y;
    uf:=-ta*ufold+u;
    r:=1;
    e:=yf-yfold;
    for i:=1 to n do begin
        fi1[i]:=0;
        for j:=1 to n do fi1[i]:=fi1[i]+p[i,j]*fi[j];
        r:=r+fi[i]*fi1[i];
        e:=e-fi[i]*th[i]
    end;
    for i:=1 to n do
        th[i]:=th[i]+fi1[i]*e/r;
    for i:=1 to n do
        for j:=i to n do begin
            p[i,j]:=p[i,j]-fi1[i]*fi1[j]/r;
            p[j,i]:=p[i,j];
            if i=j then p[i,j]:=p[i,j]/lambda;
    end;
  end;
end;

{=============================================================regdesign}
Procedure regdesign(specif:specificationtype; estpar:estpartype;
                    var regpar:regpartype);
   { The procedure RC designs a regulator according to Astrom's
     explicit adaptive pole-zero-placement algorithm for a first
     order process with a time delay.
     The inequalities imply two test values:
     tcomf=a*b1+b2 for test of common factor, and
     tdampz=-b2/b1 for test of well damped zero.      }
 var a1,b1,b2,b3,bmaxv,bminv,tsampd:real;
     bmax,bmin:integer;
     adjust:boolean;

{-----------------------------------------------------------maxmin}
Procedure maxmin;
   { Detemnine the bmax and bmin from b1,b2 and b3. }
 begin
   with specif,estpar do begin
     a1:=-th[1];        b1:=th[2];
     b2:=th[3];         b3:=th[4];
     if abs(b1))abs(b2) then begin
        bmax:=1;        bmaxv:=b1;
        bmin:=2;        bminv:=b2;
     end
     else begin
            bmax:=2;      bmaxv:=b2;
            bmin:=1;      bminv:=b1;
          end;
     if abs(b3))abs(bmaxv) then begin
        bmax:=3;          bmaxv:=b3;
     end;
```

```
      if abs(b3)(abs(bminv) then bmin:=3;
      if bmin=1 then begin
         b1:=th[3];     b2:=th[4];
      end;
    end;
  end;

{-------------------------------------------------------calculator}
Procedure calculator;
  { Calculates the regulator parameters of Ts(Td and Ts)Td
    respectively, where Ts is the sample period and Td is
    the time delay of the controlled process. }
  var p1,ps,as1,bs,w1,nsum:real;
  begin
  with specif do begin
    as1:=1+a1;         bs:=b1+b2;
    p1:=-exp(-pole*tsamp);
    ps:=1+p1;
    w1:=p1*ta-a1;
    nsum:=(b1+b2)*(a1*b1+b2);
    with regpar do begin
      if bmin=1 then
           if abs(nsum)(abs(0.01*bmaxv*bmaxv) then begin
              t0:=ps/b1;
              r1:=ps+ta;
              r2:=0.0;
              s0:=(r1+ta*p1)/b1;
              s1:=0.0
           end
           else if (-b2)dampl*b1) and (-b2(damph*b1) then begin
                   t0:=ps/b1;
                   r1:=ps+a1+ta+b2/b1;
                   r2:=r1*b2/b1;
                   s0:=(r1+a1*r1-a1+ta*p1)/b1;
                   s1:=-a1*r1/b1
                end
                else begin
                  t0:=ps/bs;
                  r1:=as1+p1+ta;
                  r2:=(b2*b2*(w1+as1*r1)+b1*b2*a1*r1)/nsum;
                  s0:=(b2*((as1+a1*a1)*r1+as1*w1)+b1*(a1*as1*r1+a1*w1))/ns
                  s1:=(b2*(-a1*as1*r1-a1*w1)-a1*a1*b1*r1)/nsum
                end;
       if bmin()1 then
           if abs(nsum)(abs(0.01*bmaxv*bmaxv) then begin
              t0:=ps/b1;
              r1:=-ta*p1;
              r2:=0.0;
              s0:=(ps+ta-r1)/b1;
              s1:=0.0;
           end
           else if (-b2)dampl*b1) and (-b2(damph*b1) then begin
                   t0:=ps/b1;
                   r1:=b2/b1;
                   r2:=0.0;
                   s0:=(as1+ta+p1)/b1;
                   s1:=(ta*p1-a1)/b1;
                end
                else begin
                        t0:=ps/bs;
                        r1:=((as1+ta+p1)*b2*b2+(a1-ta*p1)*b1*b2)/nsum;
```

```
                        r2:=0.0;
                        s0:=((as1+ta+p1)-r1)/b1;
                        s1:=-a1*r1/b2;
                end;
      end;
    end;
 end;

{-------------------------------------------------------modifytsamp}
Procedure modifytsamp;
   { Adjusts the sample period according to estimates. }
 var tdi:real;
 begin
   with specif do begin
     tdi:=(a1*b1+b2)/(a1*(b1+b2));
     if (tdi)0) and (a1)0) then begin
        td:=-tsamp*ln(tdi)/ln(a1);
        if bmin=1 then td:=td+tsamp;
        t:=-tsamp/ln(a1);
     end;
     tsampd:=tsamp;
     adjust:=false;
     if k)=np then begin
        if (tsamp)(hhi*td)) or (tsamp((hlo*td)) then
           tsampd:=(hhi+hlo)*td/2;
        adjust:=true;
     end;
     if k)np then begin
        if tsamp)(hhi*td) then tsampd:=tsamp-ni;
        if tsamp((hlo*td) then tsampd:=tsamp+ni;
        if tsamp()tsampd then adjust:=true;
     end;
   end;
 end;

{-------------------------------------------------------modifyestpar}
Procedure modifyestpar;
 var afa,bata,tmd:real;
     tdt,tt:integer;
 begin
   with specif,estpar do begin
     afa:=ln(a1)/tsamp;
     bata:=(b1+b2)*ln(a1)/(tsamp*(1-a1));
     th[1]:=-exp(-afa*tsampd);
     tdt:=trunc(1000*td);
     tt:=trunc(1000*tsampd);
     tmd:=0.01*(tdt mod tt);
     b1:=bata*(1-exp(-afa*(tsampd-tmd)))/afa;
     b2:=-bata*th[1]*(exp(afa*tmd)-1)/afa;
     if bmin=1 then begin
        th[2]:=0.01;
        th[3]:=b1;
        th[4]:=b2;
     end
     else begin
            th[2]:=b1;
            th[3]:=b2;
            th[4]:=0.01;
          end;
   end;
 end;
```

```
{================================================code of regdesign}
{ code of regdesign }
begin
  with specif do begin
    maxmin;
    calculator;
    modifytsamp;
    if adjust then begin
       modifyestpar;
       newpar:=false;
       with compar do tsamp:=tsampd;
       newpar:=true;
    end;
  end;
end;
```

APPENDIX C: PROGRAM LISTING 2.

```
PROGRAM SELFTUNER2(2);

   { Authors  Zhou Z.Y. and Astrom K.J.
     Data     Dec. 1980.
     References
       Astrom K.J. and Zhou (1979): Self-tuners with Automatic
         Adjustment of the Sampling Period for Processes
         with Time delay.
       Zhou Z.Y.(1980):A microcomputer implementation of
         datalogging and recursive least sqares estimatoion.
       Wittenmark B., Hagander and Gustavsson I. (1980):
         STUPID Implementation of a self-tuning PID-controller.
       Mattsson S.E.(1978):A simple real-time schedurer.
         report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root  (SR)  estimation  and  several
     regulator calculations (RC) for simple  processes with
     time delay.  All  results  can  be  stored in files if
     respective logging command is  given.
     The  control  algorithm  based  on  explicit adaptive
     pole-zero-placement is described  in Astrom (1980).
     A group of estimates with two priods are used to sense
     the time delay of a controlled process and the sampling
     period is adjusted automatically.

     The program consists of these files:
     RLS    -- estimator.
     RC     -- regulator calculation.
     FG     -- foreground program.
     OPCOM -- operator communication and main program.   }

{ GLOBAL DATA DECLARATIONS }

const n=4;
      m=7;
      l=180;

type  specificationtype=record
        pole,tsamp,ta,ni,hhi,hlo,dampl,damph,lambda,po,
        lolim,hilim,aoa,boa,bob,boc:real;
        nd,ns,nta,ntb,np,yrchan,ychan,uchan,outchan:integer;
        log:boolean;
        end;
      estpartype=record
        th,fi:array[1..2,1..n] of real;
        p:    array[1..2,1..n,1..n] of real;
        end;
      regpartype=record
        t0,r1,r2,s0,s1:real;
        end;
      logpartype=record
        logth:    array[1..n,1..l] of real;
        logregpar:array[1..m,1..l] of real;
        loguy:    array[0..2,1..l] of real;
        end;

var   specif,compar:specificationtype;
```

```
        estpar:estpartype;
        regpar:regpartype;
        logpar:logpartype;
        q,i,j,k,s,w,period:integer;
        yr,yrold1,y,yold1,u,u1,uold1,uold2,uold3:real;
        yf,uf,yfold,ufold:array[1..2] of real;
        bmin:array[1..2] of integer;
        newpar,insample,desample:boolean;
        delu:array[1..100] of real;
        ss:array[1..100] of integer;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
Procedure estimation(y:real; var estpar:estpartype);
                                                        external;
Procedure regdesign(specif:specificationtype;estpar:estpartype;
                              var regpar:regpartype);external;
Procedure delayu;external;

{=============================================================FOREGROUND}
Procedure foreground;

   { The procedure FG is organized as follows:
     Adin     inputs yr(t) and y(t);
     Control calculates control signal u(t) from measurements
             and reglator parameters of given process;
     Daout    outputs u(t);
     Estimation estimates the parameters of the process;
     Regdesign  calculates the reglator parameters ;
     Display display parameters of estimation and reglator
             on screen;
     Logdata prepares data for store in files after running;
     delayu   delays the output of control signal;
     Updata   updates input and output for next step.          }


{-------------------------------------------------------------control}
Procedure control;
   { calculates control signals u(t) from measurements ( yr
     and y );  reglator parameters ( t0, s0, s1, r1 )  and
     integral compensation ub. }
 var tpoly,spoly,rpoly:real;
 begin
   with regpar,specif,estpar do begin
     if odd(k) then begin
        tpoly:=t0*yr+t0*ta*yrold1;
        spoly:=s0*y+s1*yold1;
        rpoly:=(1-r1)*uold1+(r1-r2)*uold2+r2*uold3;
        u1:=tpoly-spoly+rpoly;
        u:=u1;
        if u1<lolim then u:=lolim;
        if u1>hilim then u:=hilim;
     end
     else u:=uold1;
   end;
 end; { of control }

{-------------------------------------------------------------display}
Procedure display;
   { displays parameters of the process and the regulator
     on screen }
 begin
   with specif,estpar,regpar do begin
     if s=ns then begin
        writeln('point',k);
        for q:=1 to 2 do begin
          for i:=1 to n do begin
              write('    ');   write(th[q,i]:8:4)
          end;
          if bmin[q]=1 then if q=1 then
               write('     h(Td,   h=',tsamp:8:4)
                 else write('   2h(Td,  Tc=',2*tsamp:8:4)
            else if bmin[q]=3 then if q=1 then
              write('     h)Td,   h=',tsamp:8:4)
```

```
                else write('   2h)Td,  Tc=',2*tsamp:8:4);
          writeln;
        end;
        write(' ');         write(t0:8:4);
        write('  ');         write(ta:8:4);
        write('  ');         write(r1:8:4);
        write('  ');         write(r2:8:4);
        write('  ');         write(s0:8:4);
        write('  ');         write(s1:8:4);
        write('  ');         write(u1:8:4);
        write('  ');         write(u:8:4);
        writeln;
        s:=0
      end;
      s:=s+1
    end
 end; { of display }

{-------------------------------------------------------------logdata}
Procedure logdata;
   { prepares data and parameters from k=nta to k=ntb for
     store in files after running }
 begin
   with logpar,estpar,regpar do begin
     for i:=1 to n do logth[i,w]:=th[2,i];
     loguy[0,w]:=yr;       loguy[1,w]:=y;
     loguy[2,w]:=u;
     logregpar[1,w]:=t0; logregpar[2,w]:=r1;
     logregpar[3,w]:=r2; logregpar[4,w]:=s0;
     logregpar[5,w]:=s1; logregpar[6,w]:=u1;
     logregpar[7,w]:=u;  w:=w+1
   end
 end; { of logpar }

{-------------------------------------------------------------updata}
Procedure updata;
   { updates input and output preparing for next step }
 var ym,um:real;
 begin
   with estpar,specif do begin
     fi[1,4]:=fi[1,3];
     fi[1,3]:=fi[1,2];
     fi[1,1]:=yfold[1]-yf[1];
     fi[1,2]:=uf[1]-ufold[1];
     yfold[1]:=yf[1];
     ufold[1]:=uf[1];
     if odd(k) then begin
        fi[2,4]:=fi[2,3];
        fi[2,3]:=fi[2,2];
        fi[2,1]:=yfold[2]-yf[2];
        fi[2,2]:=uf[2]-ufold[2];
        yfold[2]:=yf[2];
        ufold[2]:=uf[2];
        yrold1:=yr;
        yold1:=y;
        uold3:=uold2;
        uold2:=uold1;
        uold1:=u;
        if insample then begin
           for i:=1 to n do th[1,i]:=th[2,i];
           th[2,1]:=-th[1,1]*th[1,1];
```

```
            th[2,2]:=th[1,2];
            th[2,3]:=-th[1,1]*th[1,3]-th[1,1]*th[1,2]+th[1,3];
            th[2,4]:=0.0;
            insample:=false;
         end;
         if desample then begin
            for i:=1 to n do th[2,i]:=th[1,i];
            if th[2,1]<0 then th[1,1]:=-sqrt(-th[2,1]);
            th[1,2]:=0.0;
            th[1,3]:=th[2,2];
            th[1,4]:=(th[2,3]+th[1,1]*th[1,3])/(1-th[1,1]);
            desample:=false;
         end;
      end;
   end
 end; { of update }

{-------------------------------------------------CODE OF FOREGROUND}
{ CODE FOREGROUND }

begin
   with specif do begin
      yr:=adin(yrchan);
      y:= adin(ychan);
      control;
      daout(outchan,u);
      delayu;
      q:=1;
      estimation(y,estpar);
      if odd(k) then begin
         q:=2;
         estimation(y,estpar);
         regdesign(specif,estpar,regpar);
      end;
      display;
      if log then
         if (k)=nta) and (k<=ntb) then logdata;
      updata;
      k:=k+1;
      if newpar then begin
         specif:=compar;
         period:=round(50*tsamp);
         newpar:=false
      end
   end
end; { of foreground }
```

```
PROGRAM SELFTUNER2(2);

   { Author   Zhou Z.Y.
     Data     Dec. 1980.
     References
       Astrom K.J. and Zhou (1979): Self-tuners with Automatic
         Adjustment of the Sampling Period for Processes
         with Time delay.
       Zhou Z.Y.(1980):A microcomputer implementation of
         datalogging and recursive least sqares estimatoion.
       Wittenmark B., Hagander and Gustavsson I. (1980):
         STUPID Implementation of a self-tuning PID-controller.
       Mattsson S.E.(1978):A simple real-time schedurer.
         report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root (SR)  estimation  and  several
     regulator calculations (RC) for simple  processes with
     time delay.  All  results  can  be  stored in files if
     respective logging command is  given.
     The  control  algorithm  based  on  explicit adaptive
     pole-zero-placement is described  in Astrom (1980).
     A group of estimates with two priods are used to sense
     the time delay of a controlled process and the sampling
     period is adjusted automatically.

     The program consists of these files:
     RLS   -- estimator.
     RC    -- regulator calculation.
     FG    -- foreground program.
     OPCOM -- operator communication and main program.   }

{ GLOBAL DATA DECLARATIONS }

const n=4;
      m=7;
      l=180;

type  specificationtype=record
        pole,tsamp,ta,ni,hhi,hlo,dampl,damph,lambda,po,
        lolim,hilim,aoa,boa,bob,boc:real;
        nd,ns,nta,ntb,np,yrchan,ychan,uchan,outchan:integer;
        log:boolean;
        end;
      estpartype=record
        th,fi:array[1..2,1..n] of real;
        p:    array[1..2,1..n,1..n] of real;
        end;
      regpartype=record
        t0,r1,r2,s0,s1:real;
        end;
      logpartype=record
        logth:    array[1..n,1..l] of real;
        logregpar:array[1..m,1..l] of real;
        loguy:    array[0..2,1..l] of real;
        end;

var   specif,compar:specificationtype;
      estpar:estpartype;
      regpar:regpartype;
      logpar:logpartype;
```

```
        q,i,j,k,s,w,period:integer;
        yr,yrold1,y,yold1,u,u1,uold1,uold2,uold3:real;
        yf,uf,yfold,ufold:array[1..2] of real;
        bmin:array[1..2] of integer;
        newpar,insample,desample:boolean;
        delu:array[1..100] of real;
        ss:array[1..100] of integer;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
{=============================================================OPCOM}
Procedure Opcom;

   { The procedure OPCOM is organized as follows:
     Initialize    recognizes identifiers and initializes:
        initestpar initializes estimated parameters;
        initregpar initializes reglator parameters;
        inituy     initializes input and output;
        initspecif gives specifications;
        desiredpol calculates desired polynomial.
     Help          lists available commands.
     Par           modefies parameters:
        getiden    reads the identifiers;
        get        reads parameter if it is a real;
        take       reads parameter if it is a integer;
        askboolean reads command if it is a boolean;
        initestpar
        desiredpol
        error      gives a sensible orror message if in error.
     Run           runs the selftuner.
     Stop          stops the selftuner.
     Disp          display the current parameters.
     Store         logs inputs and output.
        error
     Resultp       logs estimated parameters.
        error
     Resultu       logs reglator parameters.
        error
     Exit          stops the whole program and exits it into computer.}

{ DECLARATIONS OF OPCOM }

label 999;
const idlength=8;
      blank=' ';
type  identifiertype=array[1..idlength] of char;
      opindex=(xhelp,xpar,xrun,xstop,xdisp,xstore,xresultp,
               xresultu, xexit,xlastop);
      asindex=(spole,stsamp,sdampl,sdamph,snp,sni,slambda,
               spo,sta,shhi,shlo,slolim,shilim,snd,sns,snta,
               sntb,saoa,sboa,sbob,sboc,slog,syrchan,sychan,
               suchan,soutchan,slastas);
      errors=(fewarg,manyarg,noname,prierr,illfile,nolog);
var   identifier:identifiertype;
      operation: array[opindex] of identifiertype;
      assignment:array[asindex] of identifiertype;
      xop:opindex;
      sas:asindex;
      ch: char;
      logdata,logtheta,logreg:boolean;

{ PROCEDURE FOR OPCOM }

{-------------------------------------------------------------help}
Procedure help;
   { lists available commands and information on screen }
 begin
   for i:=1 to 5 do writeln;
   writeln('         Simple Self-tuner');
   writeln;
```

```
   writeln('The available commands are as follows:');
   writeln;
   write(' ':4,'DISP');          writeln(' ':12,'STORE');
   write(' ':4,'PAR');           writeln(' ':13,'RESULTP');
   write(' ':4,'RUN');           writeln(' ':13,'RESULTU');
   write(' ':4,'STOP');          writeln(' ':12,'EXIT');
   writeln;
   writeln('The parameters can be changed:');
   writeln;
   writeln(' ':4,'POLE,TSAMP,TA,NI,HHI,HLO,LAMDA,PO,
    ND,NS,NTA,NTB,NP,DAMPL,DAMPH,LOLIM,HILIM.');
   writeln;
   writeln('The initial values can be assigned:');
   writeln;
   writeln(' ':4,'AOA,BOA,BOB,BOC.');
   writeln;
   writeln('The channels can be chosen:');
   writeln;
   writeln(' ':4,'YRCHAN,YCHAN,UCHAN,OUTCHAN'); writeln
 end; { of help }

{-------------------------------------------------------------error}
Procedure error(err:errors);
   { gives a sensible error message }
 begin
   case err of
     fewarg: writeln('too few arguments');
     manyarg:writeln('too many arguments');
     prierr: writeln(identifier,':illegal value');
     noname: writeln(identifier,':illegal argument');
     nolog:  writeln(identifier,':log do not be reguired');
     illfile:writeln(identifier,':ill file')
   end;
   goto 999
 end; { of error }

{-----------------------------------------------------initestpar}
Procedure initestpar;
   { initializes th, fi and p }
 begin
   with compar,estpar do begin
     for q:=1 to 2 do
       for i:=1 to n do
         for j:=1 to n do
             if i=j then p[q,i,j]:=po else p[q,i,j]:=0.0;
     th[2,1]:=aoa;   th[2,2]:=boa;
     th[2,3]:=bob;   th[2,4]:=boc;
   end;
   insample:=false;  desample:=false;
   k:=1;       s:=1;
   w:=1;
 end; { of initestpar }

{-------------------------------------------------------------inituy}
Procedure inituy;
   { initializes input and output }
 begin
   with estpar do
     for q:=1 to 2 do
       for i:=1 to n do begin
           th[q,i]:=0.01;  fi[q,i]:=0.0;
```

```
      end;
   for i:=1 to 20 do delu[i]:=0.0;
   yrold1:=0.0;       yold1:=0.0;
   for q:=1 to 2 do begin
      yfold[q]:=0.0;   ufold[q]:=0.0;
   end;
   uold1:=0.0;     uold2:=0.0;
   uold3:=0.0;
   for i:=1 to 100 do ss[i]:=0;
   daout(0,0.0);  daout(1,0.0)
 end; { of inituy }

{-------------------------------------------------------initspecif}
Procedure initspecif;
   { assigns specification }
 begin
   with compar,estpar do begin
     pole:=2;        tsamp:=1.2;
     ni:=0.001;      np:=50;
     lambda:=0.98;   po:=100;
     ta:=-0.05;      hhi:=0.78;
     hlo:=0.72;      dampl:=0.2;
     damph:=0.9;     lolim:=-1;
     hilim:=1;       nta:=1;
     ntb:=180;       ns:=50;
     aoa:=-0.01;     boa:=0.01;
     bob:=0.01;      boc:=0.01;
     log:=false;     yrchan:=0;
     ychan:=1;       uchan:=2;
     outchan:=1;     nd:=5;
   end
 end; { of initspecif }

{-------------------------------------------------------initregpar}
Procedure initregpar;
   { initializes reglator parameters }
 begin
   with regpar do begin
     t0:=1.0;
     r1:=0.0;      r2:=0.0;
     s0:=0.0;      s1:=0.0;
   end
 end;

{-------------------------------------------------------initialize}
Procedure initialize;
   { recognizes identifiers and initializes }
 begin
   operation[xhelp]:=     'HELP    ';
   operation[xdisp]:=     'DISP    ';
   operation[xpar]:=      'PAR     ';
   operation[xrun]:=      'RUN     ';
   operation[xstop]:=     'STOP    ';
   operation[xexit]:=     'EXIT    ';
   operation[xstore]:=    'STORE   ';
   operation[xresultp]:=  'RESULTP ';
   operation[xresultu]:=  'RESULTU ';
   assignment[spole]:=    'POLE    ';
   assignment[stsamp]:=   'TSAMP   ';
   assignment[sdampl]:=   'DAMPL   ';
   assignment[sdamph]:=   'DAMPH   ';
```

```
    assignment[snp]:=      'NP       ';
    assignment[sni]:=      'NI       ';
    assignment[slambda]:= 'LAMBDA   ';
    assignment[spo]:=      'PO       ';
    assignment[sta]:=      'TA       ';
    assignment[shhi]:=     'HHI      ';
    assignment[shlo]:=     'HLO      ';
    assignment[slolim]:=   'LOLIM    ';
    assignment[shilim]:=   'HILIM    ';
    assignment[snd]:=      'ND       ';
    assignment[sns]:=      'NS       ';
    assignment[snta]:=     'NTA      ';
    assignment[sntb]:=     'NTB      ';
    assignment[saoa]:=     'AOA      ';
    assignment[sboa]:=     'BOA      ';
    assignment[sbob]:=     'BOB      ';
    assignment[sboc]:=     'BOC      ';
    assignment[slog]:=     'LOG      ';
    assignment[syrchan]:= 'YRCHAN   ';
    assignment[sychan]:=   'YCHAN    ';
    assignment[suchan]:=   'UCHAN    ';
    assignment[soutchan]:='OUTCHAN ';
    with compar do begin
      initspecif;
      initestpar;
      initregpar;
      inituy;
      specif:=compar;
      newpar:=true
    end;
  end; { of initialize }

{-------------------------------------------------------skipblank}
Procedure skipblank;
  begin
    repeat read(ch) until (ch<>blank) or eoln
  end; { of skipblank }

{-------------------------------------------------------getident}
Procedure getident;
    { reads a identifier from the terminal }
  begin
    for i:=1 to idlength do identifier[i]:=blank;
    skipblank;
    i:=1;
    while (ch>='A') and (ch<='Z') and (i<9) and not eoln do
    begin identifier[i]:=ch;
          i:=i+1;
          read(ch)
    end;
    if (ch>='A') and (ch<='Z') then identifier[i]:=ch
  end; { of getident }

{-----------------------------------------------------------take}
Procedure take(var rn:real);
    { reads a number if it is a real }
  begin
    if ch=blank then read(rn);
    if (rn<-1000) or (rn>1000) then error(prierr) else read(ch);
    if ch<>',' then read(ch)
  end; { of take }
```

```
{-------------------------------------------------------------------get}
Procedure get(var inn:integer);
   { reads a number if it is a integer }
 begin
   if ch=blank then read(inn);
   if inn<0 then error(prierr) else read(ch);
   if ch<>',' then read(ch)
 end; { of get }

{-----------------------------------------------------------askboolean}
Procedure askboolean(var command:boolean);
   { reads a command if it is a boolean }
 var comname:array[1..idlength] of char;
 begin
     comname:=identifier;
     if ch=blank then getident;
     if identifier='TRUE    ' then command:=true
     else command:=false;
     identifier:=comname
 end; { of askboolean }

{-----------------------------------------------------------------par}
Procedure par;
   { updates parameters }
 var pp:specificationtype;
 begin
   if eoln then error(fewarg);
   pp:=compar;
   repeat getident;
     assignment[slastas]:=identifier;
     sas:=spole;
     while assignment[sas]<>identifier do sas:=succ(sas);
     case sas of
       spole:   take(pp.pole);
       stsamp:  take(pp.tsamp);
       sdampl:  take(pp.dampl);
       sdamph:  take(pp.damph);
       snp:     get(pp.np);
       sni:     take(pp.ni);
       slambda: take(pp.lambda);
       spo:     take(pp.po);
       sta:     take(pp.ta);
       shhi:    take(pp.hhi);
       shlo:    take(pp.hlo);
       slolim:  take(pp.lolim);
       shilim:  take(pp.hilim);
       snd:     get(pp.nd);
       sns:     get(pp.ns);
       snta:    get(pp.nta);
       sntb:    get(pp.ntb);
       saoa:    take(pp.aoa);
       sboa:    take(pp.boa);
       sbob:    take(pp.bob);
       sboc:    take(pp.boc);
       syrchan: get(pp.yrchan);
       sychan:  get(pp.ychan);
       suchan:  get(pp.uchan);
       soutchan:get(pp.outchan);
       slog:    askboolean(pp.log);
       slastas: error(noname)
```

```
      end;
      writeln(identifier,'has been read')
    until eoln;
    newpar:=false;
    compar:=pp;
    with compar do begin
      initestpar;
    end;
    newpar:=true
  end; { of par }

{-------------------------------------------------------------------disp}
Procedure disp;
    { displays current parameters }
  var rr:real;

  Procedure dataform(rr:real);
     { formats a real number }
   var w:integer;
   begin
     w:=rform(rr,6);  write(rr:10:w)
   end; { of dataform }

  begin { disp }
    with compar do begin
      writeln('   The current parameters are:');
      write('pole=   '); dataform(pole);    write('   ');
      write('tsamp=  '); dataform(tsamp);   writeln;
      write('nd=     '); write(nd:6);       write(' ':7);
      write('ns=     '); writeln(ns:6);
      write('np=     '); write(np:6);       write(' ':7);
      write('nta=    '); writeln(nta:6);
      write('ni=     '); dataform(ni);      write('   ');
      write('ntb=    '); writeln(ntb:6);
      write('lambda= '); dataform(lambda);  write('   ');
      write('aoa=    '); dataform(aoa);     writeln;
      write('po=     '); dataform(po);      write('   ');
      write('boa=    '); dataform(boa);     writeln;
      write('ta=     '); dataform(ta);      write('   ');
      write('bob=    '); dataform(bob);     writeln;
      write('hhi=    '); dataform(hhi);;    write('   ');
      write('boc=    '); dataform(boc);     writeln;
      write('hlo=    '); dataform(hlo);     write('   ');
      write('yrchan= '); writeln(yrchan:6);
      write('damph=  '); dataform(damph);   write('   ');
      write('ychan=  '); writeln(ychan:6);
      write('dampl=  '); dataform(dampl);   write('   ');
      write('uchan=  '); writeln(uchan:6);
      write('hilim=  '); dataform(hilim);   write('   ');
      write('outchan='); writeln(outchan:6);
      write('lolim=  '); dataform(lolim);   write('   ');
      write('log=    '); writeln(' ':4,log);
    end
  end; { of disp }

{-----------------------------------------------------------buildfile}
Procedure buildfile;
    { stores data of input-output, estimated parameters and
      regulator parameters in files respectively }
  var outfile:file of char;
      filename:array[1..14] of char;
```

```
    len:integer;
 begin
   with specif do
     if log=false then error(nolog) else begin
     read(filename);   len:=-1;
     rewrite(outfile,filename,'DAT',len);
     if len=-1 then error(illfile);
     with logpar do begin
       for w:=1 to (ntb-nta) do begin
         if logdata then  begin
            for i:=0 to 2 do write(outfile,loguy[i,w]:8:4);
            writeln(outfile)
         end;
         if logtheta then begin
            for i:=1 to n do write(outfile,logth[i,w]:8:4);
            writeln(outfile)
         end;
         if logreg then begin
            for i:=1 to m do write(outfile,logregpar[i,w]:8:4);
            writeln(outfile)
         end;
       end;
       close(outfile)
     end
   end
 end; { of buildfile }

{-------------------------------------------------------------stop}
Procedure stop;
 begin
   period:=0; writeln(k:6,' data points have been collected.')
 end; { of stop }

{-----------------------------------------------CODE OF OPCOM}
{ CODE OPCOM }

begin
   initialize;
   repeat
     write(')');
     getident;
     operation[xlastop]:=identifier;
     xop:=xhelp;
     while operation[xop]<>identifier do xop:=succ(xop);
     case xop of
          xhelp:   help;
          xdisp:   disp;
          xpar:    par;
          xrun:    period:=10;
          xstop,
          xexit:   stop;
          xstore:  begin logdata:=true;
                     logtheta:=false;    logreg:=false;
                     buildfile
                   end;
          xresultp:begin logtheta:=true;
                     logdata:=false;     logreg:=false;
                     buildfile
                   end;
          xresultu:begin logreg:=true;
                     logdata:=false;     logtheta:=false;
```

```
                    buildfile
                  end;
          xlastop: error(noname)
      end;
999: readln
   until xop=xexit
end; { of opcom }

{================================================CODE OF MAIN PROGRAM}
{ CODE MAIN PROGRAM }
Procedure foreground;external;

begin
   period:=0;
   clksave;
   schedule(foreground,period);
   opcom;
   clkrestore
end. { of main program }
```

```
PROGRAM SELFTUNER2(2);

   { Author   Zhou Z.Y.
     Data     Dec. 1980.
     References
       Astrom K.J. and Zhou (1979): Self-tuners with Automatic
         Adjustment of the Sampling Period for Processes
         with Time delay.
       Zhou Z.Y.(1980):A microcomputer implementation of
         datalogging and recursive least sqares estimatoion.
       Wittenmark B., Hagander and Gustavsson I. (1980):
         STUPID Implementation of a self-tuning PID-controller.
       Mattsson S.E.(1978):A simple real-time schedurer.
         report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root (SR)  estimation  and  several
     regulator calculations (RC) for simple  processes with
     time delay.  All  results  can  be  stored in files if
     respective logging command is  given.
     The  control  algorithm  based  on  explicit adaptive
     pole-zero-placement is described  in Astrom (1980).
     A group of estimates with two priods are used to sense
     the time delay of a controlled process and the sampling
     period is adjusted automatically.

     The program consists of these files:
     RLS    -- estimator.
     RC     -- regulator calculation.
     FG     -- foreground program.
     OPCOM -- operator communication and main program.    }

{ GLOBAL DATA DECLARATIONS }

const n=4;
      m=7;
      l=180;

type  specificationtype=record
        pole,tsamp,ta,ni,hhi,hlo,dampl,damph,lambda,po,
        lolim,hilim,aoa,boa,bob,boc:real;
        nd,ns,nta,ntb,np,yrchan,ychan,uchan,outchan:integer;
        log:boolean;
        end;
      estpartype=record
        th,fi:array[1..2,1..n] of real;
        p:    array[1..2,1..n,1..n] of real;
        end;
      regpartype=record
        t0,r1,r2,s0,s1:real;
        end;
      logpartype=record
        logth:    array[1..n,1..l] of real;
        logregpar:array[1..m,1..l] of real;
        loguy:    array[0..2,1..l] of real;
        end;

var   specif,compar:specificationtype;
      estpar:estpartype;
      regpar:regpartype;
      logpar:logpartype;
```

```
        q,i,j,k,s,w,period:integer;
        yr,yrold1,y,yold1,u,u1,uold1,uold2,uold3:real;
        yf,uf,yfold,ufold:array[1..2] of real;
        bmin:array[1..2] of integer;
        newpar,insample,desample:boolean;
        delu:array[1..100] of real;
        ss:array[1..100] of integer;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
{=============================================================estimation}
Procedure estimation(y:real; var estpar:estpartype);
   { The procedure RLS calculates a Recursive Least Squares
     estimation for a first order process and the differences
     beteewn the filterred input and output signals are used.}
var r,e:real;
    fi1:array[1..n] of real;
begin
  with specif,estpar do begin
    yf[q]:=-ta*yfold[q]+y;
    uf[q]:=-ta*ufold[q]+u;
    r:=1;
    if q=1 then e:=yf[1]-yfold[1]
       else e:=yf[2]-yfold[2];
    for i:=1 to n do begin
        fi1[i]:=0;
        for j:=1 to n do fi1[i]:=fi1[i]+p[q,i,j]*fi[q,j];
        r:=r+fi[q,i]*fi1[i];
        e:=e-fi[q,i]*th[q,i]
    end;
    for i:=1 to n do
        th[q,i]:=th[q,i]+fi1[i]*e/r;
    for i:=1 to n do
        for j:=i to n do begin
            p[q,i,j]:=p[q,i,j]-fi1[i]*fi1[j]/r;
            p[q,j,i]:=p[q,i,j];
            if i=j then p[q,i,j]:=p[q,i,j]/lambda;
    end;
  end;
end;

{=============================================================regdesign}
Procedure regdesign(specif:specificationtype; estpar:estpartype;
                    var regpar:regpartype);
   { The procedure RC designs a regulator according to Astrom's
     explicit adaptive pole-zero-placement algorithm for a first
     order process with a time delay.
     The inequalities imply two test values:
     tcomf=a*b1+b2 for test of common factor, and
     tdampz=-b2/b1 for test of well damped zero.      }
 var a1,b1,b2,b3,bmaxv,bminv,tsampd:real;
     bmax:integer;

{-------------------------------------------------------------maxmin}
Procedure maxmin;
   { Detemnine the bmax and bmin from b1,b2 and b3. }
 begin
   with specif,estpar do begin
     for q:=1 to 2 do begin
       a1:=-th[q,1];        b1:=th[q,2];
       b2:=th[q,3];         b3:=th[q,4];
       if abs(b1))abs(b2) then begin
          bmax:=1;        bmaxv:=b1;
          bmin[q]:=2;     bminv:=b2;
       end
       else begin
              bmax:=2;      bmaxv:=b2;
              bmin[q]:=1;   bminv:=b1;
            end;
       if abs(b3))abs(bmaxv) then begin
          bmax:=3;          bmaxv:=b3;
```

```
        end;
        if abs(b3)<abs(bminv) then bmin[q]:=3;
        if bmin[q]=1 then begin
           b1:=th[q,3];     b2:=th[q,4];
        end;
      end;
    end;
  end;

{-----------------------------------------------------------calculator}
Procedure calculator;
  { Calculates the regulator parameters of Ts<Td and Ts>Td
    respectively, where Ts is the sample period and Td is
    the time delay of the controlled process. }
  var p1,ps,as1,bs,w1,nsum:real;
  begin
  with specif do begin
    as1:=1+a1;         bs:=b1+b2;
    p1:=-exp(-pole*tsamp);
    ps:=1+p1;
    w1:=p1*ta-a1;
    nsum:=(b1+b2)*(a1*b1+b2);
    with regpar do begin
      if bmin[2]=1 then
           if abs(nsum)<abs(0.01*bmaxv*bmaxv) then begin
              t0:=ps/b1;
              r1:=ps+ta;
              r2:=0.0;
              s0:=(r1+ta*p1)/b1;
              s1:=0.0
           end
           else if (-b2)dampl*b1) and (-b2<damph*b1) then begin
                   t0:=ps/b1;
                   r1:=ps+a1+ta+b2/b1;
                   r2:=r1*b2/b1;
                   s0:=(r1+a1*r1-a1+ta*p1)/b1;
                   s1:=-a1*r1/b1
                end
                else begin
                  t0:=ps/bs;
                  r1:=as1+p1+ta;
                  r2:=(b2*b2*(w1+as1*r1)+b1*b2*a1*r1)/nsum;
                  s0:=(b2*((as1+a1*a1)*r1+as1*w1)+b1*(a1*as1*r1+a1*w1))/ns
                  s1:=(b2*(-a1*as1*r1-a1*w1)-a1*a1*b1*r1)/nsum
                end;
      if bmin[2]<>1 then
         if abs(nsum)<abs(0.01*bmaxv*bmaxv) then begin
            t0:=ps/b1;
            r1:=-ta*p1;
            r2:=0.0;
            s0:=(ps+ta-r1)/b1;
            s1:=0.0;
         end
         else if (-b2)dampl*b1) and (-b2<damph*b1) then begin
                 t0:=ps/b1;
                 r1:=b2/b1;
                 r2:=0.0;
                 s0:=(as1+ta+p1)/b1;
                 s1:=(ta*p1-a1)/b1;
              end
              else begin
```

```
                        t0:=ps/bs;
                        r1:=((as1+ta+p1)*b2*b2+(a1-ta*p1)*b1*b2)/nsum;
                        r2:=0.0;
                        s0:=((as1+ta+p1)-r1)/b1;
                        s1:=-a1*r1/b2;
                  end;
      end;
    end;
  end;

{-------------------------------------------------------decision}

Procedure decision;
   { make a decision if the sampling period must be adjusted. }
 var ssum:integer;
 begin
   with specif do begin
     for i:=np downto 2 do ss[i]:=ss[i-1];
     if bmin[2]=1 then ss[1]:=1
        else if (bmin[1]=3) and (bmin[2]=3) then ss[1]:=-1
             else ss[1]:=0;
     ssum:=0;
     for i:=1 to np do ssum:=ssum+ss[i];
     if ssum=np then begin
        insample:=true;   desample:=false;
        tsampd:=2*tsamp;
        for i:=1 to np do ss[i]:=0;
        end else if ssum=-np then begin
              desample:=true;     insample:=false;
              tsampd:=tsamp/2;
              for i:=1 to np do ss[i]:=0;
            end;
   end;
 end;

{=================================================code of regdesign}
{ code of regdesign }
begin
  with specif do begin
    maxmin;
    calculator;
    decision;
    if (insample or desample) then begin
       newpar:=false;
       with compar do tsamp:=tsampd;
       newpar:=true;
    end;
  end;
end;
```