



LUND UNIVERSITY

A Microcomputer Implementation of LQG Self-Tuner

Zhou, Zhao-Ying; Åström, Karl Johan

1981

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Zhou, Z.-Y., & Åström, K. J. (1981). *A Microcomputer Implementation of LQG Self-Tuner*. (Technical Reports TFRT-7226). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:

2

General rights

Unless other specific re-use rights are stated the following general rights apply:

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

CODEN: LUTFD2/(TFRT-7226)/1-052/(1981)

A MICROCOMPUTER IMPLEMENTATION OF LOG SELF-TUNER

ZHOU ZHAO-YING

KARL JOHAN ÅSTRÖM

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
JULY 1981

| | | |
|---|-----------------------|---|
| LUND INSTITUTE OF TECHNOLOGY DEPARTMENT OF AUTOMATIC CONTROL Box 725 S 220 07 Lund 7 Sweden | | Document name Internal report |
| Author(s) Zhou Zhao-ying Karl Johan Aström | | Date of issue July 1981 |
| | | Document number CODEN: LUTFD2/(TFRT-7226)/1-052/(1981) |
| | | Supervisor |
| | | Sponsoring organization STU 78-3763 |
| Title and subtitle A Microcomputer Implementation of LQG Self-tuner | | |
| Abstract This report describes an interactive program for a single-input single-output LQG self-tuner and some results of experiments. The program is based on recursive parameter estimation and regulator design based on spectral factorization and a solution of a linear polynomial equation. The program also allows regulator design based on pole-placement. There is also a code for interactive control of the experiment. The program is written in PASCAL. The experiments have been run on a DEC LSI 11/03. In the experiments the self-tuner has been used to control a laboratory process for concatenated control. Different processes have also been simulated on an analog computer. This includes ship steering simulations and stochastic control experiments. | | |
| Key words | | |
| Classification system and/or index terms (if any) | | |
| Supplementary bibliographical information | | |
| ISSN and key title | | ISBN |
| Language English | Number of pages 52 | 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.

A MICROCOMPUTER IMPLEMENTATION

OF

LOG SELF-TUNER

Zhou Zhao-ying

Karl Johan Aström

Department of Automatic Control

Lund Institute of Technology

July 1981

CONTENTS

- 1. INTRODUCTION**
- 2. THE LOG DESIGN**
- 3. PARAMETER ESTIMATION**
- 4. IMPLEMENTATION AND CODING**
- 5. EXPERIMENTS**
- 6. CONCLUSIONS**
- 7. REFERENCES**

- APPENDIX: A PROGRAM LISTING**

1. INTRODUCTION

A block diagram of a self-tuning regulator is shown in Fig.1. The regulator can be thought of as composed of two loops. An ordinary control loop with a process and a regulator. A tuning loop composed of a parameter estimation and a control design.

In the particular implementation described in this paper the control design is based on spectral factorization and solution of a linear polynomial equation. A useful property of the LQG self-tuner is that the only parameters which have to be externally adjusted are the weighting between output and control error and the sampling period. There are also possibilities to determine the sampling period automatically.

The general principles of these types of self-tuners are discussed in Astrom (1980a). An LQG self-tuner for a special case which admits analytical solution of the spectral factorization problem is given in Astrom (1980b). Other implementations of the LQG self-tuner based on solution of the Riccati equation are given in Astrom (1974) and Gustavsson (1980).

The report is organized as follows. A review of the LQG design for systems with known parameters are given in section 2. A brief summary of the recursive parameter estimation used is given in section 3. Section 4 deals with practical problem related to implementation and coding.

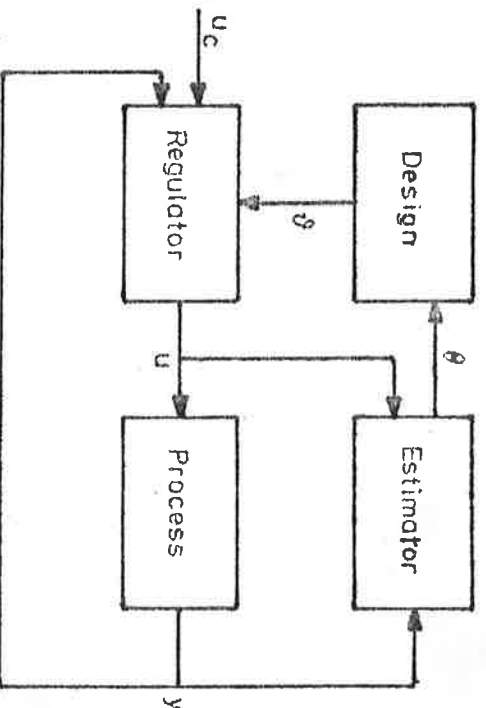


Fig.1. A block diagram of a self-tuner

Results of some experiments performed with the self-tuner are given in section 5. Conclusions are given in section 6.

This report is the last in a series of three reports on LQG self-tuners for SISO processes. The other two are listed in the references, see Zhou and Astrom (1981a) and (1981b).

The authors would like to thank Rolf Braun who provided us with photographs for this report.

2. THE LOG DESIGN

Consider a single-input single-output system whose dynamics is described by

$$A(z^{-1})y(t) = B(z^{-1})u(t) + C(z^{-1})e(t) \quad (2.1)$$

Let the criterion be to minimize

$$E \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^t [y^2(k) + \rho u^2(k)] \quad (2.2)$$

The optimal feedback law can be expressed as follows

$$R(z^{-1})u(t) = C(z^{-1})u_c(t) - S(z^{-1})y(t) \quad (2.3)$$

The polynomials R and S are determined by a two step procedure. The characteristic polynomial of the optimal closed loop system is first obtained as a solution to the spectral factorization problem.

$$P(z)P(z^{-1}) = \rho A(z)A(z^{-1}) + B(z)B(z^{-1}) \quad (2.4)$$

where ρ is the weighting factor in the criterion (2.2). The polynomials R and S are then obtained from the solution of the linear polynomial equation.

$$A(z^{-1})R(z^{-1}) + B(z^{-1})S(z^{-1}) = P(z^{-1})C(z^{-1}) \quad (2.5)$$

The equation (2.5) has many solutions. The particular solution with

$$\deg S = \deg A - 1$$

$$\deg R = \deg B - 1$$

is chosen if the degree condition

$$\deg (P + C) < \deg A + \deg B - 1$$

holds. See Astrom (1979) and Zhou and Astrom (1981b).

Spectral Factorization

The spectral factorization problem is solved iteratively using the recursion

$$P_{j+1} = \frac{1}{2} [P_j + X_j] \quad (2.7)$$

where X_j is given by

$$P_j(z)X_j(z^{-1}) + X_j(z)P_j(z^{-1}) = \rho A(z)A(z^{-1}) + B(z)B(z^{-1}) \quad (2.8)$$

If the iteration is started with a stable polynomial the sequence $\{P_j\}$ will converge to the desired solution. In practice only a few iterations are required. One to three iterations are used in the program. The polynomial P from the previous step is used as a starting point for the iteration.

Equation (2.8) is solved using the complex integrals

$$I(k) = \frac{1}{2\pi i} \oint \frac{gA(z)A(z^{-1}) + B(z)B(z^{-1})}{P_j(z)P_j(z^{-1})} \frac{dz}{z^{k+1}} \quad (2.9)$$

Introducing

$$X = [x_0, \dots, x_k]^T$$

$$\tilde{P}_j = \begin{bmatrix} P_{0,j} & & & & 0 \\ & P_{1,j} & & & \\ & & 2P_{0,j} & & \\ & & & \ddots & \\ P_{k,j} & 2P_{k-1,j} & \dots & 2P_{0,j} & \end{bmatrix}$$

and

$$I = [I(0), \dots, I(k)]^T$$

the solution can be written as

$$X = \tilde{P}_j^{-1} I \quad (2.10)$$

Formula $I(k)$ has two versions; one is accurate and another is a reduced version. See Zhou and Astrom (1981a).

The Linear Polynomial Equation

The diophantine equation (2.5) has a general solution:

$$\left. \begin{aligned} R(z^{-1}) &= E(z^{-1})P(z^{-1})C(z^{-1}) + G(z^{-1})V(z^{-1}) \\ S(z^{-1}) &= F(z^{-1})P(z^{-1})C(z^{-1}) + H(z^{-1})V(z^{-1}) \end{aligned} \right\} \quad (2.11)$$

where E , F , G and H come from a linear transformation

$$\begin{bmatrix} E & F \\ G & H \end{bmatrix} \begin{bmatrix} A & 1 & 0 \\ B & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & E & F \\ 0 & G & H \end{bmatrix} \quad (2.12)$$

with the relationships

$$\left. \begin{aligned} A(z^{-1})E(z^{-1}) + B(z^{-1})F(z^{-1}) &= 1 \\ A(z^{-1})G(z^{-1}) + B(z^{-1})H(z^{-1}) &= 0 \end{aligned} \right\} \quad (2.13)$$

which is solved by an Euclidean algorithm. See Kucera(1979).
If a common factor exists in $A(z^{-1})$ and $B(z^{-1})$, say $B_0(z^{-1})$,

the linear transformation gives

$$\begin{bmatrix} E_1 & F_1 \\ G_1 & H_1 \end{bmatrix} \begin{bmatrix} A & B & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ B_0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} B_0 & E & F & 1 \\ 0 & 1 & 1 & 1 \\ 0 & G & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

E_1 , F_1 , G_1 and H_1 also suit for the system A_1 and B_1 after cancelling the common factor from original system.

The minimum degree solution of (1.4) is given by

$$\left. \begin{aligned} R(z^{-1}) &= E(z^{-1})P(z^{-1})C(z^{-1}) \operatorname{div} G(z^{-1}) \\ S(z^{-1}) &= F(z^{-1})P(z^{-1})C(z^{-1}) \operatorname{div} H(z^{-1}) \end{aligned} \right\} \quad (2.14)$$

If the degree condition

$$\operatorname{deg} P < \operatorname{deg} A + \operatorname{deg} B - 1 \quad (2.15)$$

holds, then $R(z^{-1})$ and $S(z^{-1})$ are unique. If (2.15) does not hold, the minimum degree solution for $S(z^{-1})$ is chosen.

Pole Placement

It is clear from the description of the design algorithm that a pole placement self-tuner is obtained as a by-product, simply by specifying the polynomial P instead of determining it from spectral factorization.

3. PARAMETER ESTIMATION

The estimation is based on the mode (2.1) by means of an extended least squares algorithm, Panuska (1969). To describe the algorithm introduce the notations

$$\hat{\theta} = [a_1 \dots a_n \ b_1 \dots b_m \ c_1 \dots c_k]^T$$

$$\varphi(t) = [-y(t-1), \dots, -y(t-n), u(t-1), \dots, u(t-m), \dots, e(t-1), \dots, e(t-k)]^T$$

where $e(t+1) = y(t+1) - \varphi(t+1)^T \hat{\theta}(t)$.

The extended RLS algorithm is given by

$$\begin{aligned} \hat{\theta}(t+1) &= \hat{\theta}(t) + K(t+1) e(t+1) \\ K(t+1) &= \frac{P(t) \varphi(t+1)}{1 + \varphi(t+1)^T P(t) \varphi(t+1)} \\ P(t+1) &= \left\{ P(t) - \frac{P(t) \varphi(t+1) \varphi(t+1)^T P(t)}{1 + \varphi(t+1)^T P(t) \varphi(t+1)} \right\} / \lambda \end{aligned} \quad (3.1)$$

where the factor λ is introduced to discount past data when performing the estimation.

To avoid the problem of levels the estimation can also be based on differences of the parameter of the model

$$A(z^{-1}) \nabla y(t) = B(z^{-1}) \nabla u(t) + C(z^{-1}) \nabla e(t) \quad (3.2)$$

where ∇ is a difference operator.

Introduce

$$\varphi(t) = [-\nabla y(t-1), \dots, -\nabla y(t-n), \nabla u(t-1), \dots, \nabla u(t-m), \dots, \nabla e(t-1), \dots, \nabla e(t-k)]^T$$

then (3.1) also holds.

4. IMPLEMENTATION

All programs required for the self-tuner are written in Pascal for DEC LSI 11/03. A simple real time scheduler, Mattsson (1978), which allows the use of a foreground program for on-line control and a background program for operator communication, was used. The program listings are given in an Appendix to this report. A brief description of the program is given here.

The foreground program performs the recursive parameter estimation and the regulator design.

The background program initializes the algorithm. It also allows for operator communication using a simple command driven interaction.

The following commands are available:

HELP lists available commands, parameters and input-output channels.

DISP displays the current values of experimental parameters on screen.

PAR changes the parameters.

RUN starts the self-tuner.

STOP stops the self-tuner.

STORE stores the reference, control and output signals in a file.

RESULT stores the estimated parameters and regulator parameters in another file.

Size_of_Program_and_Code

The source code is about 1400 lines of Pascal, including comments and declarations. Some details are given in Table 1. The total program size is about 40 kbytes. Examples of computing times which executes three iterations of spectral factorization are given in Table 2.

In the coding flexibility and readability has been emphasized rather than compactness and speed. It is of interest to compare this implementation of the LQG self-tuner with others. An implementation based on the solution of Riccati equation was given in Astrom (1974). In

this implementation there was no operator communication. The pure foreground code compiled into about 8 kbytes on the PDP-15. The program was transferred to PDP 11/03 by Gustavsson (1980). Operator communication was also added in this implementation. The source code for the program was about 1400 lines, half of them are comments. The compiled code required about 40 kbytes. Of these about 8 kbytes was required for the pure foreground.

It thus appears that implementations based on Riccati equations and polynomial manipulations require about the same amount of code. The minimum size of a dedicated implementation with no operator communication is about 8 kbytes.

Table 1 -- Size of source code including comments and declarations.

| Program | Number of lines |
|----------------------------|-----------------|
| Foreground | 770 |
| Estimation | 37 |
| Spectral factorization | 144 |
| Linear polynomial equation | 275 |
| Background | 604 |

Table 2 -- Execution time.

| deg A = deg B = deg C = n | t (sec.) |
|---------------------------|----------|
| n | |
| 1 | 0.18 |
| 2 | 0.28 |
| 3 | 0.50 |
| 4 | 0.80 |
| 5 | 1.22 |
| 6 | 1.78 |
| 7 | 2.50 |
| 8 | 3.38 |

5. EXPERIMENTS

The LOG self-tuner was tested extensively by controlling several laboratory processes. A selection of the results are given in this section.

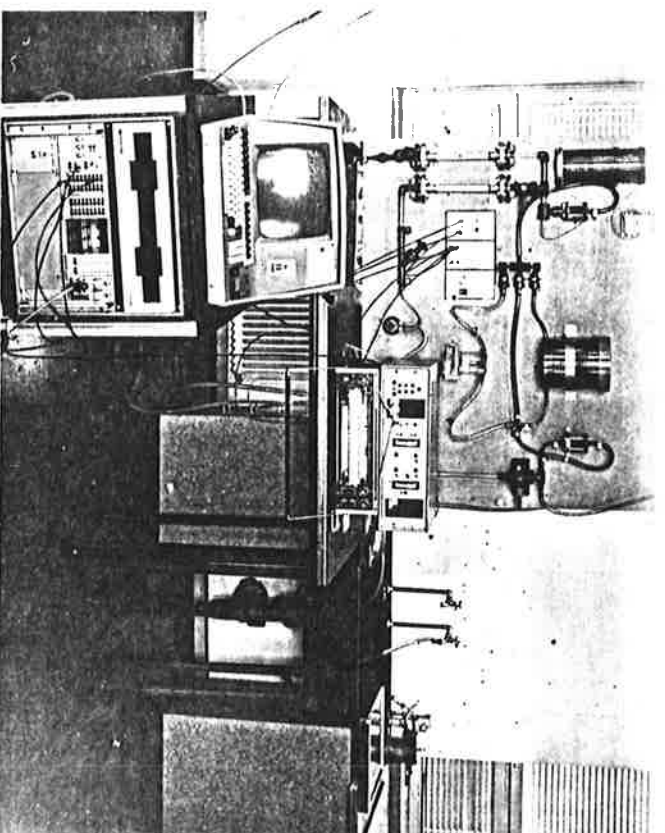
Concentration_Control

A simple laboratory process for concentration control is shown in Fig.2.

The process consist of a tank with mixing and pipes. The dynamics of the process can be approximated by time delay and first order dynamics. The time constant is only slightly larger than the time delay, see Fig.3.

The static gain of the process also varies with the operating conditions. A static characteristic of the process is shown in Fig.4. The absolute value of the gain will also vary with the concentration of the salt solution in the storage tank and the flow of fresh water.

To control a process with the dynamics of the salt process



Fig_2. Laboratory process for concentration control with the DEC LSI 11/03 computer.

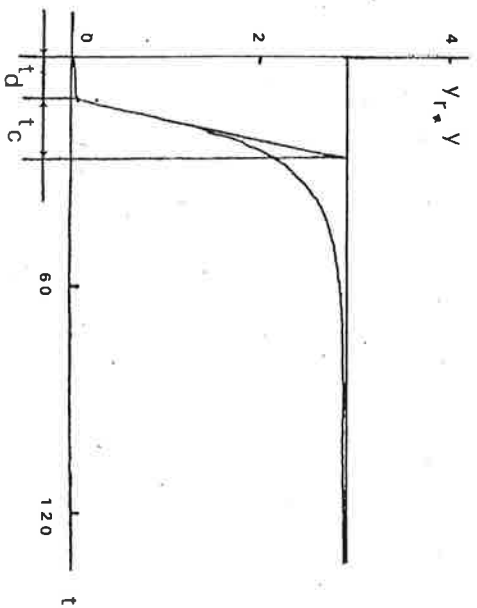


Fig. 3. Step response of the process.

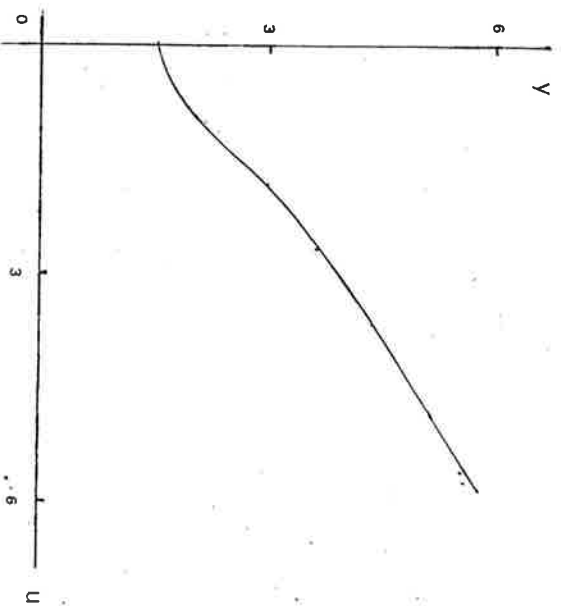


Fig. 4. Static characteristic of the process.

it is necessary to have integral action. This is forced into the LQG self-tuner by changing the ordinary diophantine equation by

$$A \nabla R + B^{-1} S = P T$$

where ∇ is the difference operator

$$\nabla = 1 - z^{-1}$$

Because air bubbles in the pipe cause a measurement disturbance, a filter is used:

$$F(s) = \frac{0.2}{s + 0.2}$$

The experimental parameters were chosen as follows: $h = 10$ s,

$\rho = 5$, and $\lambda = 0.96$. The estimated mathematic model and the corresponding regulator parameters obtained at state condition are given below.

| | | | | |
|---------------|--------|---------|---------|---------|
| A(z) | 1 | -0.4195 | | |
| B(z) | 0 | -0.0458 | 0.0714 | 0.1709 |
| P(z) | 2.2649 | -1.1826 | -0.0009 | |
| C(z) | 1 | | | |
| $\nabla R(z)$ | 2.2554 | -0.2064 | -0.7812 | -1.6746 |
| S(z) | 9.5852 | -4.1107 | | |

The zeros of polynomial $P(z)$ are 0.5229 and -0.0008. Typical experimental results are shown in Fig.5. The experiment shows responses to changes in the command signals, to load disturbances and to flow disturbances.

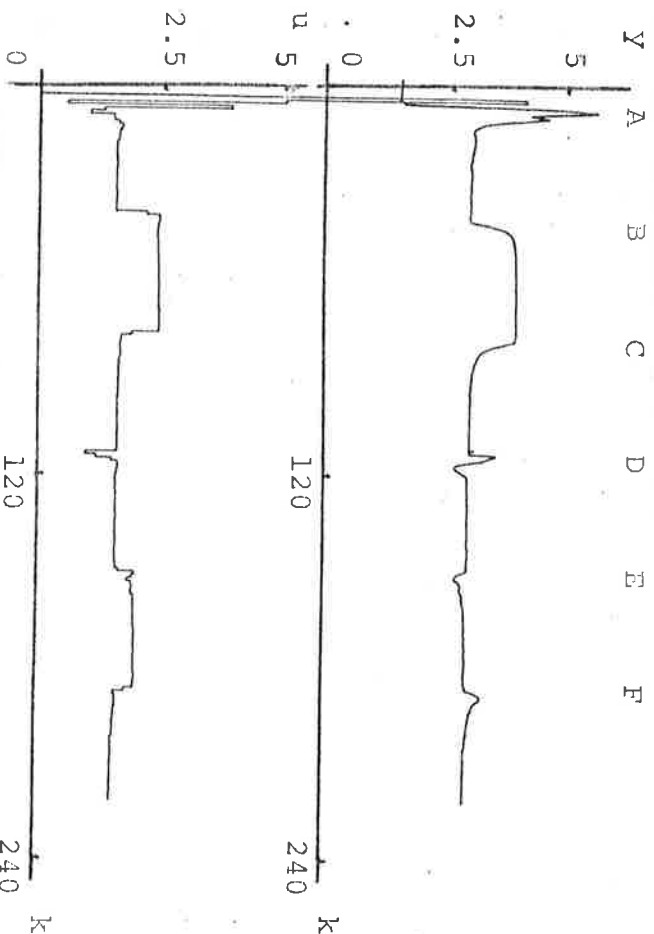


Fig.5. Experiment results of concentration control.

At A the closed loop system starts from zero initial values. At B the reference value is changed from 0.285 to 0.39 and at C from 0.39 to 0.29. The curves show clearly that the system designed by LQG responds well to step commands. At D a load disturbance is generated by injecting a high concentration salt solution for 2 seconds. The response to flow changes is shown in E and F. At E the flow is increased by 30% and at F it is again reduced to the nominal value.

The curves in Fig.5. show clearly that the regulator works very satisfactory in all cases.

Process_Simulation_on_Analog_Computer

The concentration control problem is by no means the most natural application of LQG theory. After all the LQG control attempts to minimize a lossfunction for a linear system in a noisy environment. To allow a flexible way of testing the algorithm under a large variety of circumstances several experiments were performed by simulating a process on an analog computer. The experimental set up is shown in Fig.6.

The process with the transfer function

$$G(s) = \frac{1}{s(s+1)} \quad (5.1)$$

which may represent a motor drive or simple ship steering

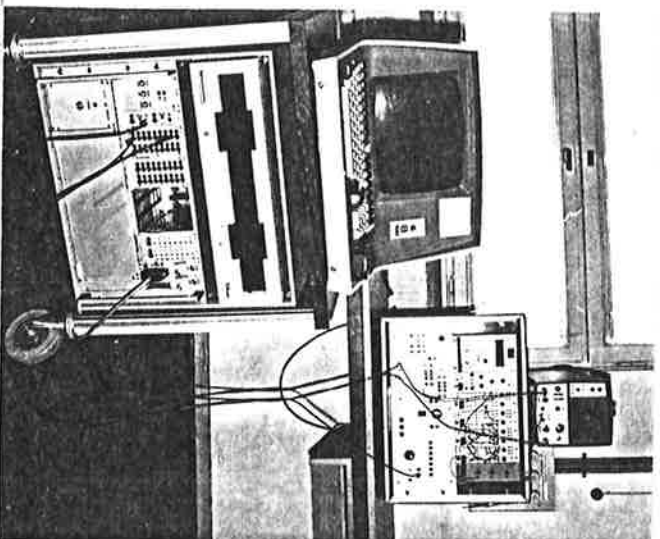


Fig. 6. Experimental set up for simulation of process dynamics on an analog computer

dynamics is easy to control. A proportional regulator with unit gain gives e.g. a good response. Now consider the process with the transfer function

$$G(s) = \frac{1}{\omega^2 (s+1)(s^2 + 2\zeta\omega s + \omega^2)} \quad (5.2)$$

This can be regarded as the system (5.1) with additional high frequency dynamics added. If the system with the transfer function (5.2) is controlled with unit gain feedback, the following criterion is obtained with the Routh-hurwitz criterion.

$$(2\zeta\omega - \omega)(\omega^2 - 1) > 0$$

For $\omega > 1$ we thus get

$$\zeta > \frac{1}{2\omega}$$

The LQG self-tuner works very well even for $\zeta < \frac{1}{2\omega}$.

A simulation with parameters $h = 0.4$ s, $\omega = 5$, $\zeta = 0.01$, $\varrho = 0.01$, and $\lambda = 0.98$ is made. The simulation results is shown in Fig. 7. The step response of the closed loop system for process (5.2) when $\varrho = 0.05$ is given in Fig. 8.

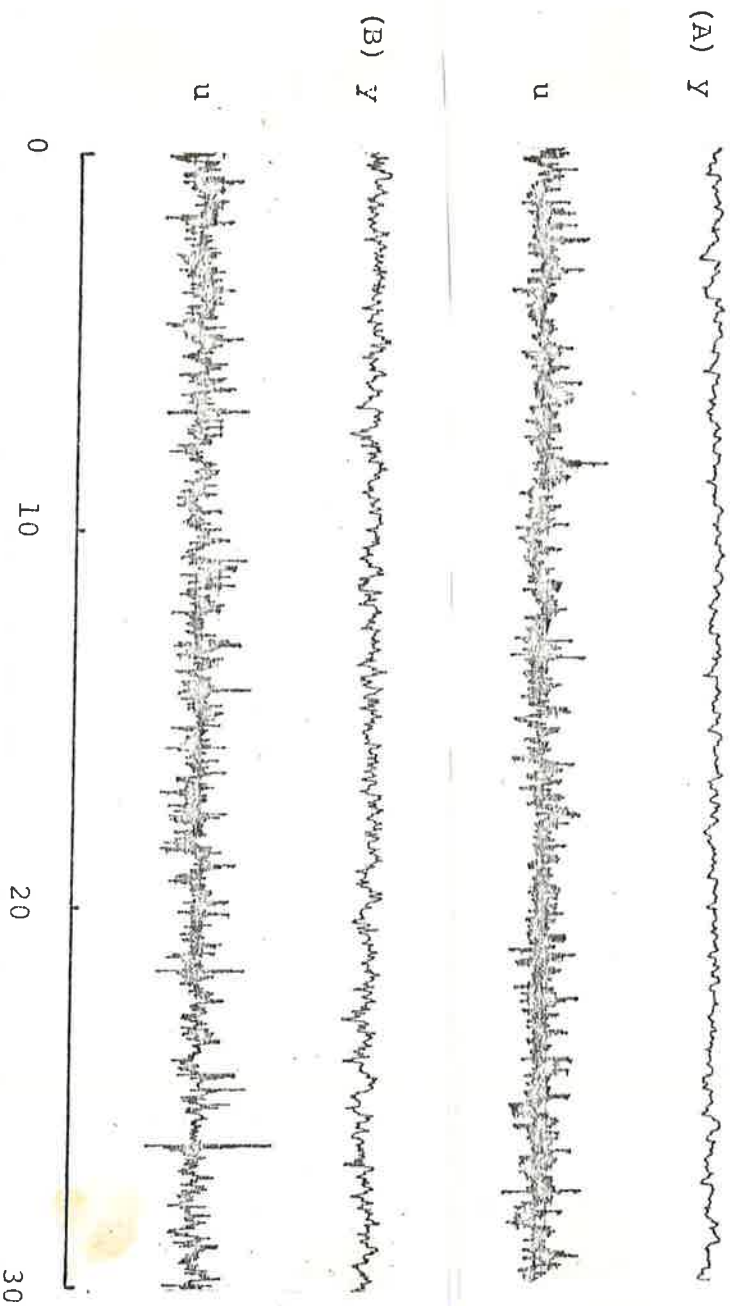


Fig. 7. Simulation results.

- A. Output and control signals of Process (5.1).
 B. Output and control signals of process (5.2).

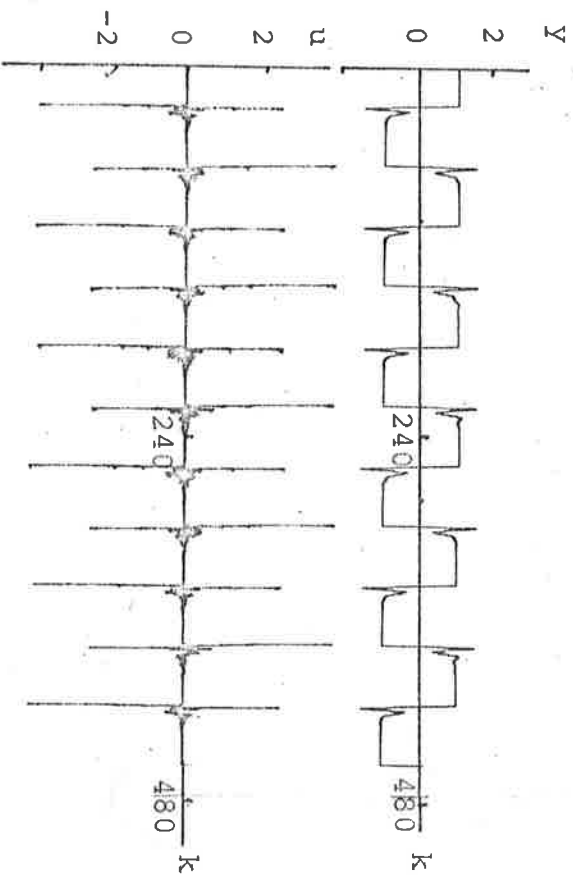


Fig.8. Closed loop response of process (5.2) .

Simulation of Ship steering Dynamics

Ship steering is a good candidate for LQG control. The ship dynamics, Astrom (1981), is given by a state equation.

$$\frac{d}{dt} \begin{bmatrix} v \\ r \\ \psi \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v \\ r \\ \psi \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ 0 \end{bmatrix} \delta$$

The transfer function from rudder δ to heading ψ is given by

$$G(s) = \frac{k(s+b)}{s(s+a)(s+c)} = \frac{1}{s(s+a)} \left[1 - \frac{c-b}{s+c} \right] \quad (5.3)$$

where the constants k , a , b and c used in the simulation are chosen as: $k = 1$, $a = 0.37$, $b = 1.4$, and $c = 2.34$. The simulation is performed with $h = 0.5$ s, as in Fig.9.

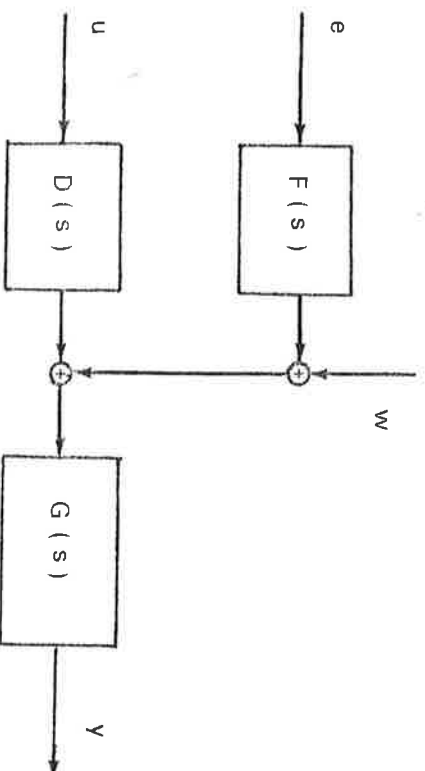
where $F(s) = \frac{0.2}{s + 0.2}$

$D(s) = 1$

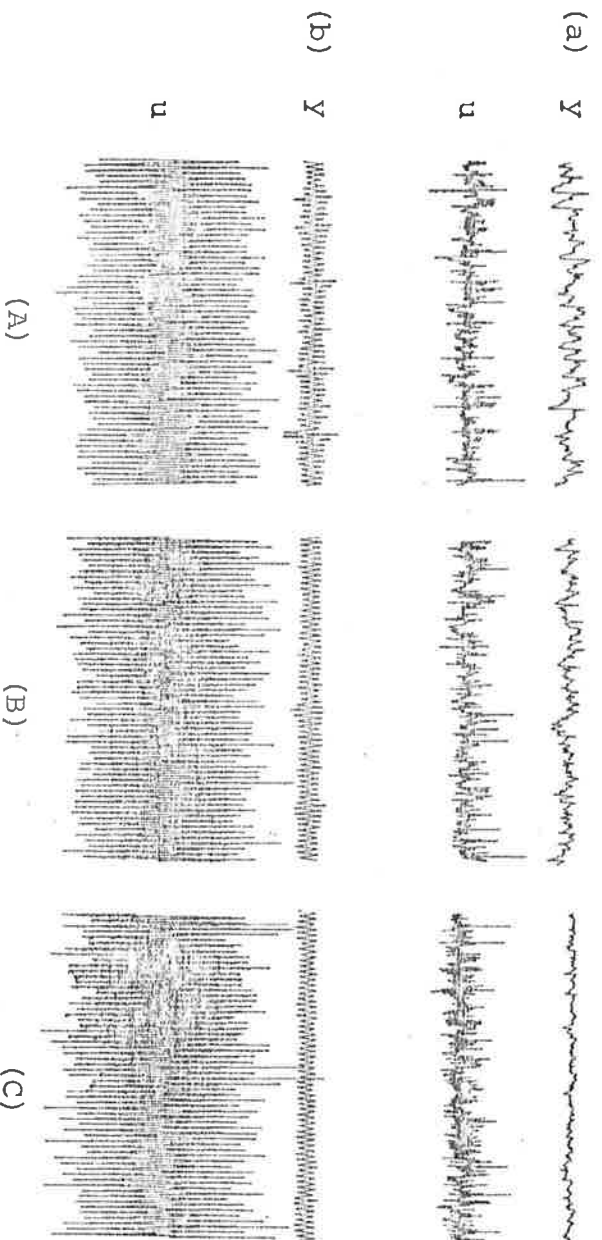
$e(t)$ -- white noise.

$w(t)$ -- triangular wave.

Output and control signals from some of the experiments are



Fig_9. Process $G(s)$ in simulation.



Fig_10. Simulation results of ship steering dynamics.

- A. $N_a=2$, $N_b=2$, $N_c=0$, $\rho=0.14$.
 - B. $N_a=2$, $N_b=3$, $N_c=0$, $\rho=0.14$.
 - C. $N_a=2$, $N_b=3$, $N_c=0$, $\rho=0.01$.
- a. White noise disturbance only.
 - b. White noise + sea wave.

shown in Fig.10. These results include simulation without (a) and with wave disturbances (b). Results using different model structures and different control weighting are also shown. The simulated wave disturbance had a period of 12 s. An attenuation of the wave disturbance by a factor of 3-5 was observed. The sampling period used was 0.3 s. A comparison of Fig.10A and Fig.10B shows that there is an incentive in increasing N_b from 2 to 3. The reason for this is that $N_b = 3$ corresponds to a time-delay which admits an approximation of higher order dynamics. The benefits desired from increasing N_b from 2 to 3 are smaller if the sampling period is increased. Fig.10C finally shows that tighter control is easily obtained by reducing the control weighting ρ . Notice that the increase of control action is very marginal.

If there is an integral action in the regulator the simulation results in the case of long period sea wave is very satisfactory. Fig.11 shows the results with different periods of sea wave. The experimental parameters were: $h = 0.5$ s, $\rho = 0.14$, $N_a = 2$ and $N_b = 3$.

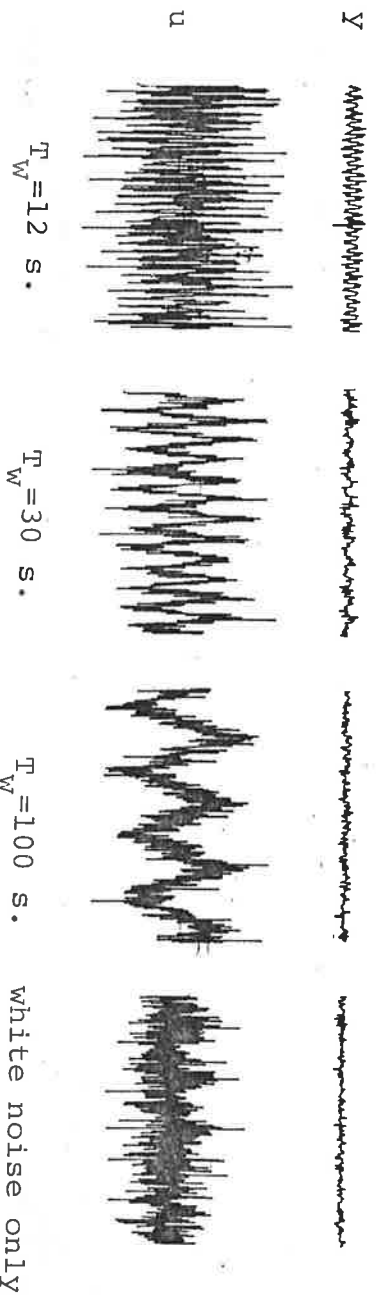


Fig--11. Simulation results of ship steering by LOG self-tuner with an integral action.

Some More Simulations

Experiments on deterministic processes: Two processes with the transfer functions

$$G_1(s) = \frac{1}{(s+1)^2}$$

and

$$G_2(s) = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2}$$

are tested. The effect of different weighting factors ρ for the processes are shown in Fig.12 and Fig.13. An interesting behaviour of the self-tuner appears when the parameters of tested processes vary. Fig.14 shows that the tuning is very satisfactory when the process 1 changes to $\frac{1}{s(s+1)}$ and then to s^{-2} . The process 2 with $\omega_0=1$ is tuned even more successfully when changing the damping factor ζ with values from 0.2 to 0.0 and then finally to -0.2. See Fig.15.

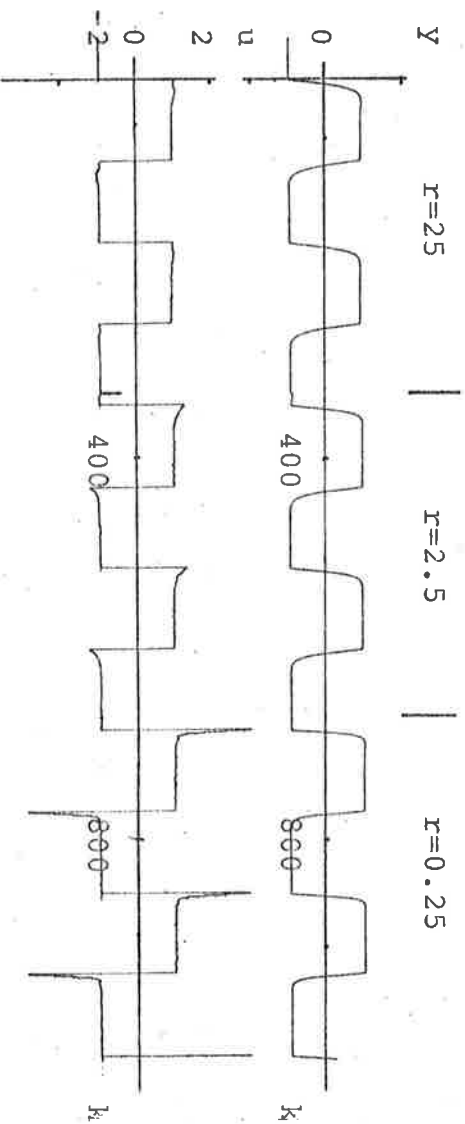


Fig.12. Change of weighting factor ρ for process 1.

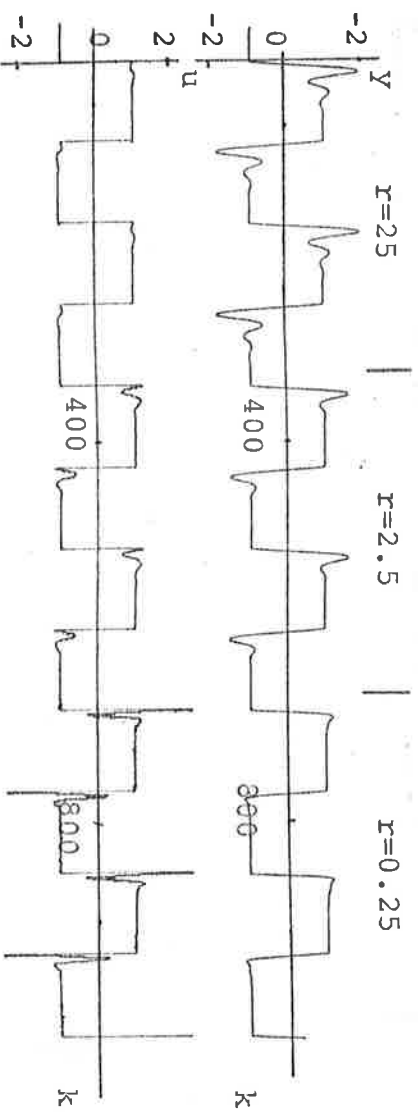


Fig. 13. Change of weighting factor ρ for process 2.

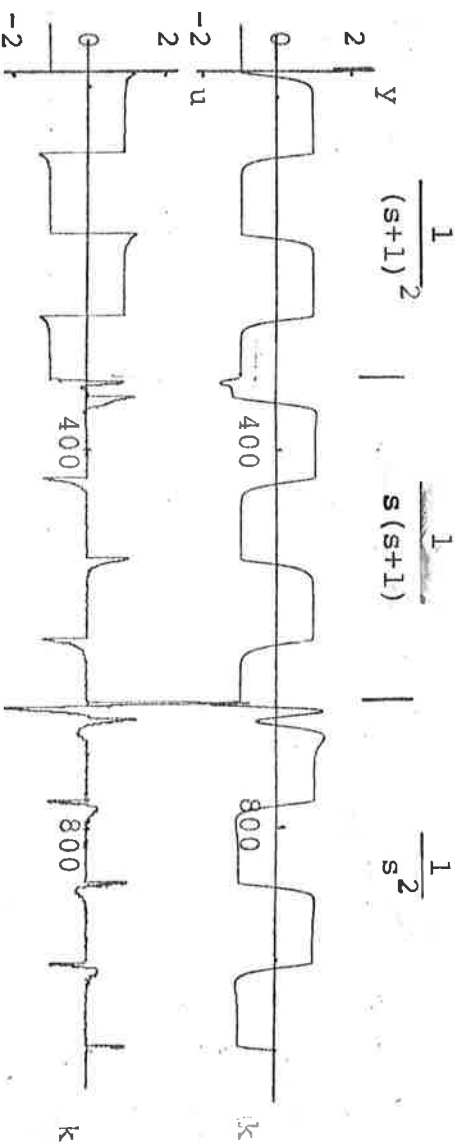


Fig. 14. LOG self-tuner working on the process 1 when parameters vary.

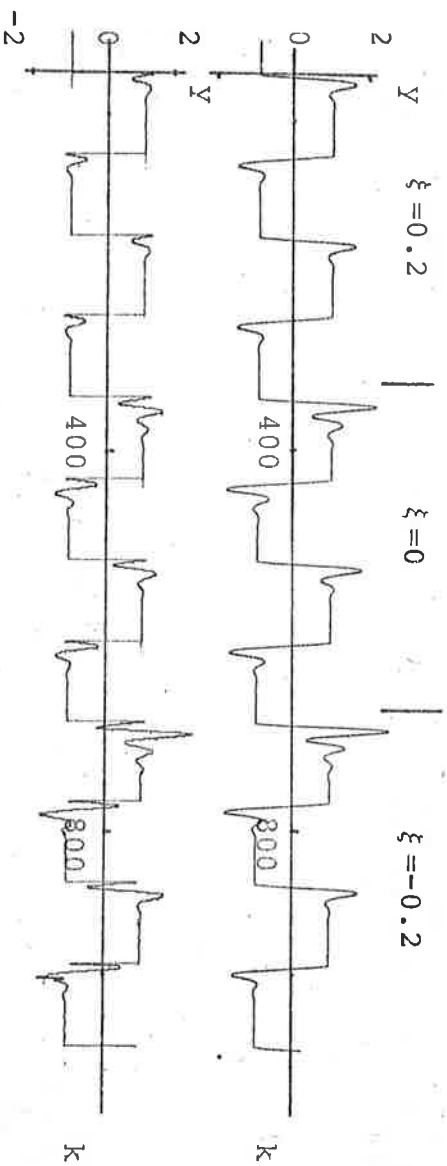


Fig. 15. LOG self-tuner working on the process 2 with different ζ .

Experiments on Stochastic Processes: The disturbance in processes simulated is a continuous white noise. These examples are

$$G_1(s) = (s+1)^{-2} \quad \text{and} \quad D_1(s) = 1.$$

$$G_2(s) = (s+1)^{-2} \quad \text{and} \quad D_2(s) = 10^4 (s+10)^{-4}$$

$$\text{and} \quad G_3(s) = s^{-1} (s+1)^{-2} \quad \text{and} \quad D_3(s) = 1.$$

See Fig. 9.

Fig. 16 shows the effect of changing weighting ρ . The estimation is based on model (2.1) with $N_a = 2$, $N_b = 2$, $N_c = 1$ and $h = 0.32$ s.

Fig. 17 shows how the system in example 2 behaves without and under control. The parameters are $N_a = 2$, $N_b = 3$, $\rho = 0.01$ and $h = 0.4$ s.

Fig. 18 shows a simulation of adaptive control of the process G_3 . The following parameters were used: $N_a = 2$,

$N_b = 4$, $\rho = 0.01$ and $h = 0.4$ s.

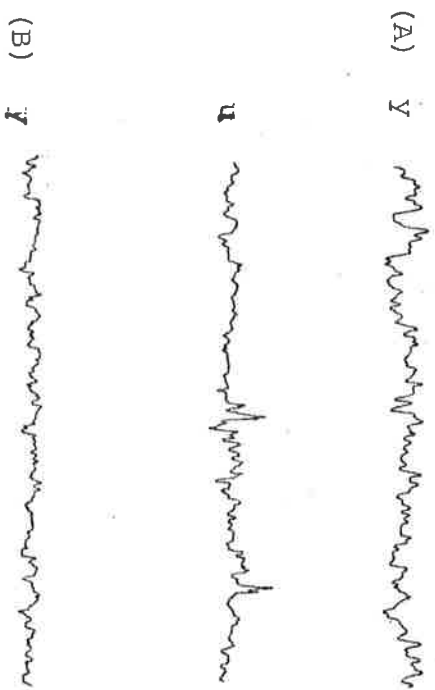


Fig. 16. Simulation results of example 1.
A. $\varrho = 1$, C. $\varrho = 0.01$.

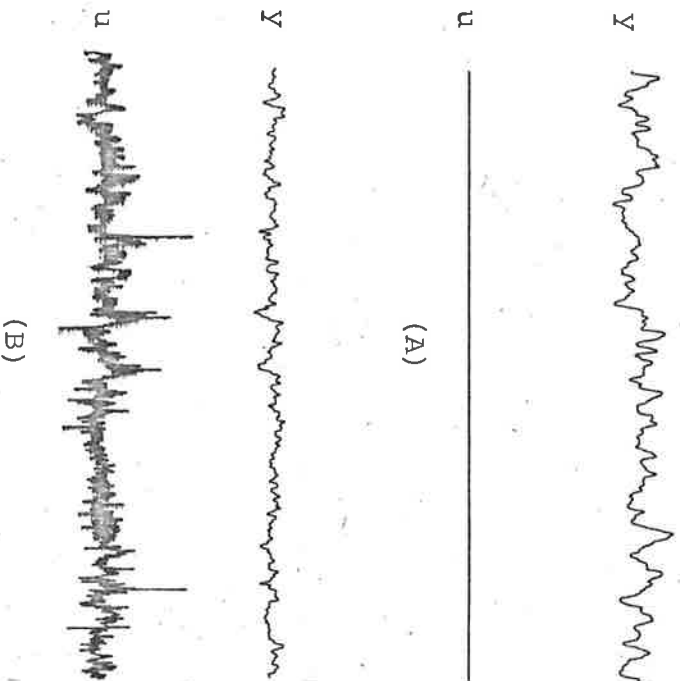
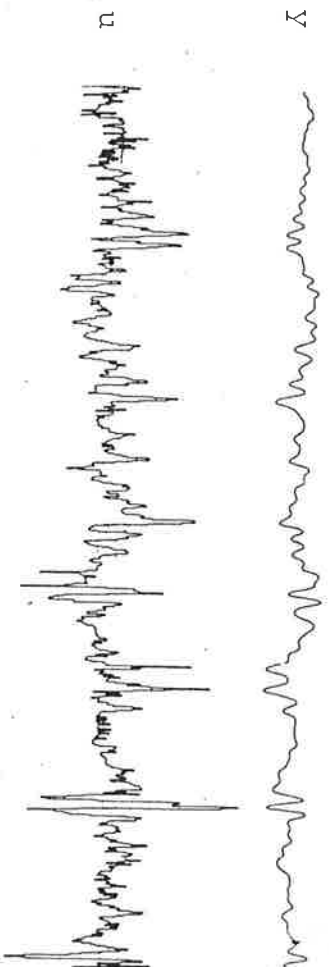


Fig. 17. Simulation results of example 2.
A. Without control, B. $\varrho = 0.01$.



Fig_18. Simulation results of example 3.

6. CONCLUSIONS

The experiments derived from this implementation shows that the LQG design is a fruitful approach to self-tuning control. The polynomial approach is very convenient since it admits simultaneous implementation of pole-placement and LQG control. It is clear that there may be numerical problems. Selfguards must be introduced to obtain a robust algorithm. Examples of such safe-guards have been given. When programmed in DNSI Pascal on DEC LSI 11/03 the code is about 40 kbytes. This includes an interactive operator communication which is about 20 kbytes. It is thus a reasonable balance between the operator communication and the control algorithm. A stripped version of the pure foreground algorithm can probably be squeezed into 10 kbytes. Execution time admits a sampling rate of 3 Hz for a second order model. For an eighths order model the sampling period must reduced to 0.3 Hz. No attempts have been made to speed up the code but there are some margins for improvement. The experiments performed show that it was easy to put the algorithm in operation.

7. REFERENCES

1. Astrom K.J. (1981): Ship Steering - A Test Example for Robust Regulator Design. CODEN: LUTFD2 / (TFRT-7222) / 1-009 / (1981).
2. Astrom K.J. (1980a): Self-tuning regulator - design principles and applications. In Narendra and Monopoli. (1980).
3. Astrom K.J. (1980b): Design of Fixed Gain and Adaptive Ship Steering Autopilots Based on the Nomoto Model. Symposium on Ship Steering Automatic Control. Genova Italy. 1980.
4. Astrom K.J. (1979): Algebraic System Theory as a Tool for Regulator Design. Report. CODEN: LUTFD2 / (TFRT-7164) / 1-023 / (1979).
5. Astrom K.J. (1974): A Self-tuning Regulator for Non-minimum Phase Systems. Report. CODEN: LUTFD2 / (TFRT-3113) / 1-048 / (1974).
6. Zhou Z.Y, Astrom K.J. (1981a): Polynomial Factorization Using Recursive Formulas for Complex Integrals. Report. CODEN: LUTFD2 / (TFRT-7223) / 1-030 / (1981).
7. Zhou Z.Y, Astrom K.J. (1981b): Regulator Synthesis Based on Polynomial Manipulation. Report. CODEN: LUTFD2 / (TFRT-7225) / 1-044 / (1981).
8. Wittenmark B, Hagander P. and Gustavsson I. (1980): STUPID Implementation of a self-tuning PID-controller. CODEN: LUTFD2 / (TFRT-7201) / 1-030 / (1980)
9. Gustavsson I. (1980): Implementation of a Self-tuning Regulator for Non-minimum Phase Systems on PDP 11/03. Report. CODEN: LUNTFD2 / (TFRT-7209) / 1-093 / (1980).
10. Mattsson S.E. (1978): A Simple Real-time Scheduler. Report. CODEN: LUTFD2 / (TFRT-7156) / 1-010 / (1978).
11. Kucera V. (1979): Discrete Linear Control. The Polynomial Equation Approach. Academica Prague.
12. Parnuska V. (1969): An Adaptive Recursive Least Squares Identification Algorithm. Proc. 1969. IEEE Symposium on Adaptive Processes. Pennsylvania State University.

APPENDIX

PROGRAM LISTING

PROGRAM SELF-TUNER;

```
{ Author   Zhou Z.Y.
  Data     May. 1981.
References
```

```
  Zhou Z.Y.(1980): A Microcomputer Implementation of
    Datalogging and Recursive Least Squares Estimation.
  Zhou Z.Y. and Astrom K.J.(1981): Polynomial Factorization
    Using Recursive Formulas for Complex Integral.
  Zhou Z.Y. and Astrom K.J.(1981): Regulator Synthesis
    Based on Polynomial Manipulation.
  Mattsson S.E.(1978): A Simple Real-time Scheduler.
  Kucera V.(1979): Discrete Linear Control. Academia
    Prague.
```

```
The program performs on-line recursive least squares
(RLS) or square root (SR) estimation, spectral
factorization (SF) and regulator design (RD) for SISO
system. All results can be stored in files
if respective logging command is given. The control
algorithm is based on pole-placement or linear
quadratic control.
```

```
The program consists of these files:
```

```
FG  -- foreground program.
OPCOM -- operator communication and main program.
RD  -- regulator design which contains
      ES - estimator,
      SF - spectral factorization, and
      RC - regulator calculation.
}

```

{ GLOBAL DATA DECLARATIONS }

```
const n=10;
      m= 5;
      l=30;

type  specificationtype=record
      tsamp,lambda,po,py,eps,lo,loim,hilim,aa,ab,ba,bb,
      pz,pa,pb,pc,pd,pe:real;
      na,nd,ni,np,ns,nt,nta,ntb,yrchan,ychan,uchan,
      outchan,delayb:integer;
      log,minr,inita,initb,initp:boolean;
      end;
      polytype=record
      zi:array[0..n] of real;
      d:integer;
      end;
var   specif,compar:specificationtype;
      a,b,c,p,r,s,t:polytype;
      i,j,k,m,nab,q,w,period,ivr,sumorder:integer;
      yr,yu,fi:array[0..n] of real;
      loga,logb,logu,logt,logr,logs:array[0..m,1..l] of real;
      ppi:array[0..n,0..n] of real;
      ul,ey: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 c1ksave;external;  
Procedure c1krestore;external;  
  
{ END OF GLOBAL }
```

```

Procedure estimation(y:real; var a,b:polytype);external;
Procedure specfac(a,b:polytype; var p:polytype);external;
Procedure regdesign(a,b:polytype; var r:s:polytype);external;
{=====FOREGROUND}
Procedure foreground;
{ The procedure FG is organized as follows:
  Adin  inputs yr(t) and y(t);
  Control compute control signal u(t) from measurements
        and regulator parameters of given process;
  Daout  outputs u(t);
  Estimation estimates the parameters of the process;
  Spefac  performs a spectral factorization;
  Regdesign calculates the regulator parameters ;
  Display display parameters of estimation and regulator
        on screen;
  Logdata prepares data for store in files after running;
  delayu  delays the output of control signal;
  Udata  updates input and output for next step.
}
}

{-----control}
Procedure control;
{ computes control signals u(t) from measurements ( yr
  and y ), regulator parameters T(z) , R(z) and S(z). }
var t0,ps,bs:real;
begin
  with specif do begin
    u1:=0;
    bs:=0;   for i:=1 to b.d do bs:=bs+b.z[i];
    ps:=0;   for i:=0 to p.d do ps:=ps+p.z[i];
    t0:=ps/bs;
    for i:=0 to t.d do u1:=u1+t.z[i]*yr[i];
    u1:=t0*u1;
    for i:=0 to s.d do u1:=u1-s.z[i]*y[i];
    for i:=1 to r.d do u1:=u1-r.z[i]*u[i];
    u1:=u1/r.z[0];
    u[0]:=u1;
    if u1<101im then u[0]:=101im;
    if u1>hilim then u[0]:=hilim;
  end;
end; { of control }

{-----display}
Procedure display;
{ displays parameters of the process and the regulator
  on screen }
begin
  with specif do begin
    if q=ns then begin
      writeln('point',k);
      write(' A(z) ');
      for i:=1 to a.d do begin
        write(' '); write(a.z[i]:8:4);
      end;
      writeln;
      write(' B(z) ');
      for i:=1 to b.d do begin
        write(' '); write(b.z[i]:8:4);
      end;
    end;
  end;
end;

```

```

        writeln;
        write(' P(z) ');
        for i:=0 to p.d do begin
            write(' '); write(p.z[i]:8:4);
        end;
        writeln;
        write(' T(z) ');
        for i:=0 to t.d do begin
            write(' '); write(t.z[i]:8:4);
        end;
        writeln;
        write(' R(z) ');
        for i:=0 to r.d do begin
            write(' '); write(r.z[i]:8:4);
        end;
        writeln;
        write(' S(z) ');
        for i:=0 to s.d do begin
            write(' '); write(s.z[i]:8:4);
        end;
        writeln;
        write(' u1 u ');
        write(' '); write(u1:8:4);
        write(' '); write(u[0]:8:4);
        writeln;
        q:=0
    end;
    q:=q+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 specif do begin
        for i:=1 to a.d do loga[i,w]:=a.z[i];
        for i:=1 to b.d do logb[i,w]:=b.z[i];
        loguy[0,w]:=yr[0]; loguy[1,w]:=y[0];
        loguy[2,w]:=u1; loguy[3,w]:=u[0];
        for i:=0 to t.d do logt[i,w]:=t.z[i];
        for i:=0 to r.d do logr[i,w]:=r.z[i];
        for i:=0 to s.d do logs[i,w]:=s.z[i];
        w:=w+1
    end
end; { of logdata }

-----updata}
Procedure updata;
{ updates input and output preparing for next step }
var b1,b2,bs,nxm:real;
begin
    with specif do begin
        for i:=t.d downto 1 do yr[i]:=yr[i-1];
        for i:=a.d downto 1 do y[i]:=y[i-1];
        for i:=b.d downto 1 do u[i]:=u[i-1];
        for i:=sumorder downto 2 do fil[i]:=fil[i-1];
        fil[1]:=y[2]-y[1];
        fila.d+1:=u[1]-u[2];
        filnab+1]:=ey;
    end
end

```

```

end
end; { of update }

-----delayu}
Procedure delayu;
{ delays output u for record. }
begin
  with specif do begin
    for i:=nd downto 2 do
      delu[i]:=delu[i-1];
      daout(0,delu[nd]);
    end;
  end;
end;

-----CODE OF FOREGROUND}
{ CODE FOREGROUND }
begin
  with specif do begin
    yr[0]:=adin(yrch);
    y[0]:= adin(ychan);
    estimation(y[0]-y[1],a,b);
    if initp=false specifac(a,b,p);
    regdesign(a,b,r,s);
    control;
    daout(outchan,u[0]);
    display;
    if log then
      if (k)=nta) and (k<=ntb) then logdata;
    delayu;
    updata;
    k:=k+1;
    if newpar then begin
      specif:=compar;
      period:=trunc(50*tsamp);
      newpar:=false
    end
  end
end
end; { of foreground }

```

PROGRAM SELF-TUNER;

```
{ Author   Zhou Z.Y.
  Data     May. 1981.
```

References

```
Zhou Z.Y.(1980): A Microcomputer Implementation of
  Data logging and Recursive Least Squares Estimation.
Zhou Z.Y. and Astrom K.J.(1981): Polynomial Factorization
  Using Recursive Formulas for Complex Integral.
Zhou Z.Y. and Astrom K.J.(1981): Regulator Synthesis
  Based on Polynomial Manipulation.
Mattsson S.E.(1978): A Simple Real-time Scheduler.
Kucera V.(1979): Discrete Linear Control. Academia
  Prague.
```

```
The program performs on-line recursive least squares
(RLS) or square root (SR) estimation, spectral
factorization (SF) and regulator design (RD) for SISO
system. All results can be stored in files
if respective logging command is given. The control
algorithm is based on pole-placement or linear
quadratic control.
```

```
The program consists of these files:
```

```
FG  -- foreground program.
OPCDM -- operator communication and main program.
RD  -- regulator design which contains
      ES - estimator,
      SF - spectral factorization, and
      RC - regulator calculation.
}
}
}
```

{ GLOBAL DATA DECLARATIONS }

```
const n=10;
      m= 5;
      l=30;
```

```
type  specificationtype=record
      tsamp,lambda,pp,py,eps,io,lo,lim,hi,lm,aa,ab,ba,bb,
          pz,pa,pb,pc,pe,pe:real;
      namb,nd,nin,pn,nt,nta,ntb,yrchan,ychan,uchan,
          outchan,delayb:integer;
      log,minr,inita,initb,initp:boolean;
      end;
      polytype=record
      zi:array[0..n] of real;
      d:integer;
      end;
var   specif,compar:specificationtype;
      a,b,c,p,r,s,t:polytype;
      i,j,k,m,namb,q,w,period,iyr,sumorder:integer;
      yr,yu,fi:array[0..n] of real;
      loga,logb,loguy,logt,logr,logs:array[0..m,1..l] of real;
      pi:array[0..n,0..n] of real;
      ul,ey: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 regulator parameters;
  initu          initializes input and output;
  initspecif     gives specifications;
  Help          lists available commands.
  Par           modifies 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;
  initabp;
  error        gives a sensible error message if in error.
  Run          runs the selftuner.
  Stop        stops the selftuner.
  Disp       display the current parameters.
  Store      logs inputs and output.
             error    logs estimated parameters.
  Resultp    logs regulator parameters.
             error    logs regulator parameters.
  Resultu    stops the whole program and exits it into
             error    computer.}

{ DECLARATIONS OF OPCOM }

label 999;
const idlength=8;
      blank=' ' ;
type  identifier=array[1..idlength] of char;
      opindex=(xhelp,xpar,xrun,xstop,xdisp,xstore,xresultp,
               xresultu,xexit,xlastop);
      asindex=(stsamp,slo,slambda,spo,spy,seps,slolim,shilim,
               snd,smi,smns,snta,sntb,slog,smirr,sinita,sinitb,
               sinitp,snps,spz,spa,spb,spc,spd,spe,sna,saa,sab,
               snb,sba,sbb,snt,sdelayb,syrchan,sychan,suchan,
               soutchan,slastas);
      errors=(fewarg,manyarg,noname,priterr,illfile,nolog);
var   identifier:identifier;
      operation:array[opindex] of identifier;
      assignment:array[asindex] of identifier;
      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('      SISO Self-tuner');
  writeln;
  writeln('The available commands are as follows:');

```

```

writeln';
write(' ':4,'DISP');
write(' ':4,'PAR');
write(' ':4,'RUN');
write(' ':4,'STOP');
writeln';
writeln('The parameters can be changed:');
writeln';
writeln(' ':4,'TSAMP,LO,LAMBDA,PO,PY,EPS,ND,NI,NS,NT,
NTA,NTB,LOLIM,HILIM');
writeln';
writeln('The initial values can be assigned:');
writeln';
writeln(' ':4,'NA,AA,AB,NB,DELAYB,BA,BB,NP,PZ,PA,PB,PC,PD,PE');
writeln';
writeln('The boolean variables are:');
writeln';
writeln(' ':4,'LOG,MINR,INITA,INITB,INITP');
writeln';
writeln('The channels can be chosen:');
writeln';
writeln(' ':4,'YRCHAN,YCHAN,UTCHAN');
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 required');
    illfile:writeln('identifier','ill file')
  end;
  goto 999
end; { of error }

```

```

-----initspecif}
Procedure initspecif;
{ assigns specification }
begin
  with compar do begin
    tsamp:=1;      1o:=1;
    na:=2;        nb:=2;
    delayb:=1;   np:=2;
    nt:=0;       ni:=3;
    ns:=50;      nd:=10;
    lambda:=0.98; po:=100;
    eps:=0.01;  py:=1;
    lolim:=-1;  hilim:=1;
    nta:=1;     ntb:=30;
    yrchan:=0;  ychan:=1;
    uchan:=2;   outchan:=1;
    log:=false; minr:=true;
    inita:=false;  initb:=false;
    initp:=false;  aa:=0.001;
    ab:=0.001;    ba:=0.001;
    bb:=0.001;    pz:=1;
    pa:=0.001;    pb:=0.001;
  end;
end;

```

```

pc:=0.001;      pd:=0.001;
pe:=0.001;
end;
end; { of initspecif }

-----inittestpar}
Procedure inittestpar;
  { initializes th, fi and p }
begin
  with compar do begin
    for i:=0 to n do begin
      a.z[i]:=0.001;
      b.z[i]:=0.001;
      p.z[i]:=0.001;
      f[i]:=0.0;
      for j:=1 to n do
        if i=j then p[i,j]:=po else p[i,j]:=0.0;
      end;
      a.d:=na;      a.z[0]:=1;
      b.d:=nb;      b.z[2]:=1;
      p.d:=np;      p.z[0]:=1;
      if na>nb then nm:=na else nm:=nb;
      sumorder:=4;  nab:=4;
      k:=1;         nab:=4;
      w:=1;         q:=1;
    end;
  end;
end;

-----initabp}
Procedure initabp;
begin
  with compar do begin
    if na>nb then nm:=na else nm:=nb;
    if nm<np then nm:=np;
    a.d:=na;
    if inita then begin
      a.z[1]:=aa;      a.z[2]:=ab;
      for i:=na+1 to nm do a.z[i]:=0;
    end;
    b.d:=nb;         b.z[b.d]:=1;
    if initb then begin
      for i:=0 to delayb-1 do b.z[i]:=0;
      b.z[delayb]:=ba;  b.z[delayb+1]:=bb;
      for i:=nb+1 to nm do b.z[i]:=0;
    end;
    p.d:=np;
    if initp then begin
      p.z[0]:=pz;      p.z[1]:=pa;
      p.z[2]:=pb;      p.z[3]:=pc;
      p.z[4]:=pd;      p.z[5]:=pe;
      for i:=np+1 to nm do p.z[i]:=0;
    end;
    t.d:=nt;
    sumorder:=na+nb+nt;
    nab:=na+nb;
    k:=1;      q:=1;
    w:=1;
  end; { of inittestpar }
end;
-----inituy}

```

```

Procedure inity;
  { initializes input and output }
begin
  with compar do begin
    for i:=0 to n do begin
      yr[i]:=0.0;
      y[i]:=0.0;
      uc[i]:=0.0;
    end;
    for i:=1 to nd do delu[i]:=0.0;
    daout(0,0.0); daout(1,0.0)
  end;
end; { of inity }

-----intregpar}
Procedure intregpar;
  { initializes reglator parameters }
begin
  for i:=1 to n do begin
    t.z[i]:=0.0;
    r.z[i]:=0.0;
    s.z[i]:=0.0;
  end;
  t.d:=0;    t.z[0]:=1;
  r.d:=0;    r.z[0]:=1;
  s.d:=0;    s.z[0]:=0;
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[tsamp]:=     'TSAMP'   ' ';
  assignment[sl0j]:=      'LO'      ' ';
  assignment[slambdaj]:=  'LAMBDA'  ' ';
  assignment[spoj]:=      'PO'      ' ';
  assignment[spyj]:=      'PY'      ' ';
  assignment[sepsj]:=     'EPS'     ' ';
  assignment[sl0limj]:=   'LOLIM'   ' ';
  assignment[shilimj]:=   'HILIM'   ' ';
  assignment[sendj]:=     'ND'      ' ';
  assignment[snij]:=      'NI'      ' ';
  assignment[snsj]:=      'NS'      ' ';
  assignment[snvj]:=      'NT'      ' ';
  assignment[snvtaj]:=    'NTA'     ' ';
  assignment[snvbtj]:=    'NTB'     ' ';
  assignment[sllogj]:=    'LOG'     ' ';
  assignment[sminrj]:=    'MINR'    ' ';
  assignment[smintaj]:=   'INITA'   ' ';
  assignment[smintbj]:=   'INITB'   ' ';
  assignment[smintpj]:=   'INITP'   ' ';
  assignment[svrchanj]:=  'VRCHAN'  ' ';

```

```

assignment[psychar] := 'YCHAN ' ;
assignment[suchar] := 'UCHAN ' ;
assignment[soutchar] := 'OUTCHAN ' ;
assignment[sna] := 'NA ' ;
assignment[saa] := 'AA ' ;
assignment[sab] := 'AB ' ;
assignment[snb] := 'NB ' ;
assignment[sdelayb] := 'DELAYB ' ;
assignment[sba] := 'BA ' ;
assignment[sbb] := 'BB ' ;
assignment[snp] := 'NP ' ;
assignment[spz] := 'PZ ' ;
assignment[spa] := 'PA ' ;
assignment[spb] := 'PB ' ;
assignment[spc] := 'PC ' ;
assignment[spd] := 'PD ' ;
assignment[spe] := 'PE ' ;

with compar do begin
  itspecif;
  itestpar;
  itregpar;
  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 n: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 pc:specificationtype;
begin
  if eoln then error(fewarg);
  pc:=compar;
  repeat getident;
  assignment[stasas]:=identifier;
  sas:=stsamp;
  while assignment[stasas]<>identifier do sas:=succ(sas);
  case sas of
    tsamp: take(pc.tsamp);
    slambda: take(pc.lambda);
    spo: take(pc.po);
    spy: take(pc.py);
    seps: take(pc.eps);
    slo: take(pc.lo);
    slolim: take(pc.lolim);
    shilim: take(pc.hilim);
    saa: take(pc.aa);
    sab: take(pc.ab);
    sba: take(pc.ba);
    sbb: take(pc.bb);
    spz: take(pc.pz);
    spa: take(pc.pa);
    spb: take(pc.pb);
    spc: take(pc.pc);
    spd: take(pc.pd);
    spe: take(pc.pe);
    sna: get(pc.na);
    snb: get(pc.nb);
    sdelayb: get(pc.delayb);
    snp: get(pc.np);
    snd: get(pc.nd);
    sni: get(pc.ni);
    sns: get(pc.ns);
    snt: get(pc.nt);
    snta: get(pc.nta);
    sntb: get(pc.ntb);
    syrchan: get(pc.yrchan);
    sychan: get(pc.ychan);
    suchan: get(pc.uchan);
    soutchan: get(pc.outchan);
  }

```

```

    slog:      askboolean(pc.log)?
    smirr:     askboolean(pc.mirr)?
    sinita:    askboolean(pc.inita)?
    sinitb:    askboolean(pc.initb)?
    sinitp:    askboolean(pc.initp)?
    slastas:   error(noname)
end!
writeln(Identifier,'has been read')
until eoln!
newpar:=false!
compar:=pc!
with compar do begin
    initabp!
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('tsamp= ')! dataform(tsamp)! write(' ')!
        write('lo= ')! dataform(lo)! write!n!
        write('ns= ')! write(ns:6)! write(' ')!
        write('ni= ')! write(ni:6)! write(' ')!
        write('nd= ')! write(nd:6)! write(' ')!
        write('lambda= ')! dataform(lambda)! write!n!
        write('po= ')! dataform(po)! write(' ')!
        write('py= ')! dataform(py)! write!n!
        write('eps= ')! dataform(eps)! write(' ')!
        write('hillim= ')! dataform(hillim)! write!n!
        write('lolim= ')! dataform(lolim)! write(' ')!
        write('nta= ')! write(нта:6)! write(' ')!
        write('ntb= ')! write(ntb:6)! write(' ')!
        write('yrchan= ')! write(yrchan:6)! write(' ')!
        write('ychan= ')! write(ychan:6)! write(' ')!
        write('uchan= ')! write(uchan:6)! write(' ')!
        write('outchan=')! write(outchan:6)! write(' ')!
        write('log= ')! write(' ':4,log)! write(' ')!
        write('mirr= ')! write(' ':4,mirr)! write(' ')!
        write('inita= ')! write(' ':4,inita)! write(' ')!
        write('initb= ')! write(' ':4,initb)! write(' ')!
        write('initp= ')! write!n(' ':4,initp)! write(' ')!
        write('na= ')! write(na:6)! write(' ')!
        write('aa= ')! dataform(aa)! write!n!
        write('ab= ')! dataform(ab)! write(' ')!
        write('nb= ')! write(nb:6)! write!n!
        write('delayb= ')! write(delayb:6)! write(' ')!
        write('ba= ')! dataform(ba)! write!n!
    end!
end!

```

```

write('bb=      '); dataform(bb); write('      ');
write('np=      '); write(np:6); write('      ');
write('pz=      '); dataform(pz); write('      ');
write('pa=      '); dataform(pa); write('      ');
write('pb=      '); dataform(pb); write('      ');
write('pc=      '); dataform(pc); write('      ');
write('pd=      '); dataform(pd); write('      ');
write('pe=      '); dataform(pe); write('      ');
write('nt=      '); write(nt:6); write('      ');
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(111file);
      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);
            writeIn(outfile)
          end;
          if logtheta then begin
            for i:=1 to a.d do write(outfile,loga[i,w]:8:4);
              for i:=1 to b.d do write(outfile,logb[i,w]:8:4);
                writeIn(outfile)
              end;
            if logreg then begin
              for i:=0 to t.d do write(outfile,logt[i,w]:8:4);
                for i:=0 to r.d do write(outfile,logr[i,w]:8:4);
                  for i:=0 to s.d do write(outfile,logs[i,w]:8:4);
                    writeIn(outfile)
                  end;
                end;
              close(outfile)
            end;
          end; { of buildfile }
        end;
      end;
    end;
  end;
}-----stop}

Procedure stop;
begin
  period:=0; writeIn(k:6,' data points have been collected.')}
end; { of stop }
}-----CODE OF 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:    with specif do
            period:=trunc(50*tsamp);
  xstop,
  xexit:   stop;
  xstore:  begin logdata:=true;
            logtheta:=false;
            buildfile
          end;
  xresultp:begin logtheta:=true;
                logdata:=false;
                buildfile
            end;
  xresultu:begin logreg:=true;
                logdata:=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;
  cksave;
  schedule(foreground,period);
  opcom;
  clkrestore;
end. { of main program }

```

PROGRAM SELF-TUNER;

```
{ Author   Zhou Z.Y.
  Data     May. 1981.
```

References

```
Zhou Z.Y.(1980): A Microcomputer Implementation of
  Datalogging and Recursive Least Squares Estimation.
Zhou Z.Y. and Astrom K.J.(1981): Polynomial Factorization
  Using Recursive Formulas for Complex Integral.
Zhou Z.Y. and Astrom K.J.(1981): Regulator Synthesis
  Based on Polynomial Manipulation.
Mattsson S.E.(1978): A Simple Real-time Scheduler.
Kucera V.(1979): Discrete Linear Control. Academia
  Prague.
```

```
The program performs on-line recursive least squares
(RLS) or square root (SR) estimation, spectral
factorization (SF) and regulator design (RD) for SISO
system. All results can be stored in files
if respective logging command is given. The control
algorithm is based on pole-placement or linear
quadratic control.
```

```
The program consists of these files:
```

```
FG -- foreground program.
OPCDM -- operator communication and main program.
RD -- regulator design which contains
      ES - estimator,
      SF - spectral factorization, and
      RC - regulator calculation.
}
```

{ GLOBAL DATA DECLARATIONS }

```
const n=10;
      m= 5;
      l=30;
```

```
type specificationtype=record
      tsamp,lambda,pp,py,eps,l0,l0lim,hilim,aa,ab,ba,bb,
        pz,pa,pb,pc,pd,pe:real;
      na,nb,nd,ni,np,ns,nt,nta,ntb,yrchan,ychan,uchan,
        outchan,delayb:integer;
      log,minr,inita,initb,initp:boolean;
      end;
      polytype=record
      z:array[0..n] of real;
      d:integer;
      end;
      specif,compar:specificationtype;
      a,b,c,p,r,s,t:polytype;
      i,j,k,m,n,na,b,q,w,period,yr,sworder:integer;
      yr,y,u,fi:array[0..n] of real;
      loga,logb,logy,logt,logr,logs:array[0..m,1..l] of real;
      pp:array[0..n,0..n] of real;
      ul,ey: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(defy:real; var a,b:polytype);
  { The procedure ERLS computes a Extended Recursive Least
    Squares estimation for different numbers of parameters. }
  var rr:real;
      fil:array[1..n] of real;
      kgain:array[1..n] of real;
begin
  with specif do begin
    rr:=1;
    ey:=defy;
    for i:=1 to sumorder do begin
      fil[i]:=0;
      for j:=1 to na do fil[i]:=fil[i]+ppli,j]*fi[j];
      for j:=na+delayb to sumorder do fil[i]:=fil[i]+ppli,j]*fi[j];
      rr:=rr+fil[i]*fil[i];
    end;
    for i:=1 to a.d do ey:=ey-fil[i]*a.z[i];
    for i:=delayb to b.d do ey:=ey-fil[a.d+i]*b.z[i];
    for i:=1 to t.d do ey:=ey-fil[nab+i]*t.z[i];
    for i:=1 to sumorder do
      kgain[i]:=fil[i]/rr;
    for i:=1 to a.d do
      a.z[i]:=a.z[i]+kgain[i]*ey;
    for i:=delayb to b.d do
      b.z[i]:=b.z[i]+kgain[a.d+i]*ey;
    for i:=1 to t.d do
      t.z[i]:=t.z[i]+kgain[nab+i]*ey;
    for i:=1 to sumorder do
      for j:=i to sumorder do begin
        pli,j:=ppli,j]-kgain[i]*fil[j];
        pplj,i:=ppli,j];
        if i=j then ppli,j]:=ppli,j]/lambda;
      end;
    end;
  end; { of estimation }
{=====SF}
Procedure Specfac(a,b:polytype; var p:polytype);
  { The procedure uses Newton's iteration method and recursive
    formulae of complex integral I (1) to perform the spectral
    factorization.
      
$$P(z)*P(z^{-1}) = 10*A(z)*A(z^{-1}) + B(z)*B(z^{-1})$$

    }
  label 10;
  var ao,bo,c,cr,co,col,x:polytype;
      qa,qb,int:array[0..n] of real;
      ra,rb:array[0..n,0..n] of real;
      fra,frb:array[0..n] of real;
      la,si,stab,result:integer;
      fa,fb,fc:real;
      it:integer;
  Procedure Polydivision;
begin
  with specif do begin
    qa[0]:=ao.z[0]/c.z[0];
    qb[0]:=bo.z[0]/c.z[0];
    for i:=1 to nm do begin

```

```

      ra[0,i] := ao.z[i] - qa[0]*c.z[i];
      rb[0,i] := bo.z[i] - qb[0]*c.z[i];
    end;
    int[0] := 0;
    if la > 0 then
      for si := 1 to la do begin
        qa[si] := ra[si-1,i]/c.z[0];
        qb[si] := rb[si-1,i]/c.z[0];
        ra[si-1,nm+1] := 0;
        rb[si-1,nm+1] := 0;
        for i := 1 to nm do begin
          ra[si,i] := ra[si-1,i+1] - qa[si]*c.z[i];
          rb[si,i] := rb[si-1,i+1] - qb[si]*c.z[i];
        end;
        ra[si,0] := 0;
        rb[si,0] := 0;
        int[si] := lo*qa[si]*ao.z[0] + qb[si]*bo.z[0];
      end;
    end;
  end; { of polydivision }

-----Iteration}
Procedure Iteration;
var int1:real;
begin
  with specif do begin
    for i := 0 to nm do begin
      a.z[i] := ao.z[i];
      b.z[i] := bo.z[i];
    end;
    for j := 0 to nm-1 do begin
      int1 := lo*a.z[nm-j]*a.z[nm-j] + b.z[nm-j]*b.z[nm-j];
      int[0] := int[0] + int1/c.z[0];
      if la > 0 then for si := 1 to la do begin
        int1 := lo*a.z[nm-j]*ra[si,nm-j] + b.z[nm-j]*rb[si,nm-j];
        int[si] := int[si] + int1/c.z[0];
      end;
      for i := 0 to nm-j do cr.z[i] := c.z[nm-j-i];
      fc := c.z[nm-j]/c.z[0];
      fa := a.z[nm-j]/c.z[0];
      fb := b.z[nm-j]/c.z[0];
      if la > 0 then for si := 1 to la do begin
        fra[si] := ra[si,nm-j]/c.z[0];
        frb[si] := rb[si,nm-j]/c.z[0];
      end;
      for i := 0 to nm-j-1 do begin
        c.z[i] := c.z[i] - fc*cr.z[i];
        a.z[i] := a.z[i] - fa*cr.z[i];
        b.z[i] := b.z[i] - fb*cr.z[i];
        if la > 0 then for si := 1 to la do begin
          ralsi,i] := ra[si,i] - fra[si]*cr.z[i];
          rbsi,i] := rb[si,i] - frb[si]*cr.z[i];
        end;
      end;
    end;
    if c.z[0] < 0 then stab := stab-1;
    if j = nm-1 then begin
      int1 := lo*a.z[nm-j-1]*a.z[nm-j-1] + b.z[nm-j-1]*b.z[nm-j-1];
      int[0] := int[0] + int1/c.z[0];
      if la > 0 then for si := 1 to la do begin
        int1 := lo*a.z[nm-j-1]*ra[si,nm-j-1] + b.z[nm-j-1]*rb[si,nm-j-1];

```


$AR + BS = C$

The general solution is given by

$R = PE + GV$

$S = PF + HY$

where V is an arbitrary polynomial.

If $\deg p \leq \deg A + \deg B - 1$

then R and S are found uniquely, else there are one

minimum degree solution for R and one for S .

```
const eps1=0.0001;
var  com,r,f,g,h,a0,b0,a1,b1:polynomial;
     r1,r2,s1,s2,v1,v2,ep,fp:polynomial;
     k1,rd:integer;
```

```
-----Inipoly}
Procedure Inipoly;
{ Set the initial values for polynomials. }
begin
  a0.d:=a.d;
  b0.d:=b.d;
  a1.d:=a.d;
  b1.d:=b.d;
  for i:=0 to a.d do begin
    a0.z[i]:=a.z[i];
    a1.z[i]:=a.z[i];
  end;
  for i:=0 to b.d do begin
    b0.z[i]:=b.z[i];
    b1.z[i]:=b.z[i];
  end;
  e.d:=b.d;
  g.d:=b.d;
  f.d:=a.d;
  h.d:=a.d;
  for i:=0 to n do begin
    e.z[i]:=0;
    f.z[i]:=0;
    g.z[i]:=0;
    h.z[i]:=0;
  end;
  e.z[0]:=1;
  h.z[0]:=1;
end;
-----Exchange}
Procedure exchange(hi,hj:polynomial; var hk,h1:polynomial);
{ Exchange polynomials Hi and Hj. }
begin
  h1.d:=hi.d;
  hk.d:=hj.d;
  for i:=0 to hi.d do h1.z[i]:=hi.z[i];
  for i:=0 to hj.d do hk.z[i]:=hj.z[i];
end;
-----Norm}
Procedure norm(hi:polynomial);
{ Calculate the Euclidean norm of a polynomial Hi. }
var d1:real;
begin
  if hi.d=0 then begin
    d1:=0.0;
```

```

    for i:=0 to hi.d do d1:=d1+hi.z[i]*hi.z[i];
    with specif do d1:=eps*sqrt(d1);
    rd:=round(d1);
  end;
end;
end;

{-----Reduce}
Procedure reduce(var hi:polytype);
{ Reduce Ni of Hi(z) if its leading coefficient is smaller
  in modulus than an external variable. }
label 2,4;
begin
  if hi.d<0 then goto 4;
2:if abs(hi.z[hi.d])<=rd then begin
  hi.d:=hi.d-1;
  if hi.d<0 then goto 4;
  goto 2;
  end;
4:
end;
end;

{-----Transformation}
Procedure Transformation;
{ performs the Euclidean transformation to find a general
  solution for the diophantine equation. }
label 10,20,30,40;
var qab:real;
  bb:integer;
begin
  if a.d<b.d then goto 20 else goto 30;
10: while a.d=b.d do begin
  ki:=a.d-b.d;
  qab:=a.z[a.d]/b.z[b.d];
  a.d:=a.d-1;
  if ki<=a.d then
    for i:=ki to a.d do a.z[i]:=a.z[i]-qab*b.z[i-ki];
  for i:=ki to nm do begin
  e.z[i]:=e.z[i]-qab*g.z[i-ki];
  f.z[i]:=f.z[i]-qab*h.z[i-ki];
  end;
  reduce(a);
  if a.d<b.d then goto 20;
end;

20:
  exchange(a,b,a,b);
  exchange(e,g,e,g);
  exchange(f,h,f,h);
  if (b.d=0) and (abs(b.z[0])<eps1) then goto 40;
30: if b.d=0 then begin
  norm(b);
  goto 10;
end;
40: if abs(a.z[0])<eps1 then qab:=1 else qab:=a.z[0];
  for i:=0 to nm do begin
  e.z[i]:=e.z[i]/qab;
  f.z[i]:=f.z[i]/qab;
  a.z[i]:=a.z[i]/qab;
  end;
  norm(e);
  reduce(e);
end;

```

```

norm(f);
reduce(f);
g.d:=g.d-a.d;
h.d:=h.d-a.d;
end;

{-----Polymul}
Procedure Polymul(p,ef:polytype; var efp:polytype);
{ Polynomial multiplication efp=p*ef }
begin
  efp.d:=p.d+ef.d;
  for i:=0 to efp.d do efp.z[i]:=0;
  for i:=0 to p.d do
    for j:=0 to ef.d do
      efp.z[i+j]:=efp.z[i+j]+p.z[i]*ef.z[j];
    end;
  end;
end;

{-----Polydiv}
Procedure Polydiv(efp,gh:polytype; var rs,vv:polytype);
{ Polynomial division rs=efp mod gh.
  vv=-(efp div gh). }
var efp1,gh1,rs1,vv1:array[0..n] of real;
  gg:integer;
begin
  vv.d:=efp.d-gh.d;
  rs.d:=gh.d-1;
  for i:=0 to efp.d do efp1[i]:=efp.z[efp.d-i];
  for i:=0 to gh.d do gh1[i]:=gh.z[gh.d-i];
  for i:=gh.d+1 to efp.d do gh1[i]:=0;
  vv1[0]:=efp1[0]/gh1[0];
  for i:=1 to efp.d do
    rs1[i]:=efp1[i]-vv1[0]*gh1[i];
  for i:=1 to vv.d do begin
    vv1[i]:=rs1[i]/gh1[0];
    rs1[efp.d+1]:=0;
    for j:=1 to efp.d do
      rs1[j]:=rs1[j+1]-vv1[i]*gh1[j];
    end;
    for i:=1 to rs.d+1 do rs.z[i-1]:=rs1[rs.d+2-i];
    for i:=0 to vv.d do vv.z[i]:=-vv1[vv.d-i];
  end;
end;

{-----Polyadd}
Procedure Polyadd(efp,ghv:polytype; var rs:polytype);
{ Polynomial addition rs=efp+ghv. }
var diff:integer;
begin
  diff:=efp.d-ghv.d;
  if diff>0 then begin
    rs.d:=efp.d;
    for i:=0 to efp.d do rs.z[i]:=efp.z[i];
    for i:=0 to ghv.d do
      rs.z[i]:=rs.z[i]+ghv.z[i];
    end
  else begin
    rs.d:=ghv.d;
    for i:=0 to ghv.d do rs.z[i]:=ghv.z[i];
    for i:=0 to efp.d do
      rs.z[i]:=rs.z[i]+efp.z[i];
    end;
  end;
end;

```

```

{-----Regulator}
Procedure Regulator;
{ computes coefficients of V1,V2,R1,R2,S1 and R2.}
var rs1:polytype;

begin
  with specif do begin
    polymul(p,e,ep);
    polymul(p,f,fp);
    if minr then begin
      polydiv(ep,g,r1,v1);
      polymul(v1,h,rs1);
      polyadd(fp,rs1,rs1);
      norm(s1);
      reduce(s1);
    end
  else begin
    polydiv(fp,h,s2,v2);
    polymul(v2,g,rs1);
    polyadd(ep,rs1,r2);
    norm(r2);
    reduce(r2);
  end;
end;

{-----Division}
Procedure Division;
{ Cancell the common factor from A(z) and B(z). }
begin
  a.d:=a0.d-com.d;
  b.d:=b0.d-com.d;
  for i:=0 to a0.d do a.z[i]:=a0.z[i];
  for i:=0 to b0.d do b.z[i]:=b0.z[i];
  for j:=0 to a.d do begin
    a.z[j]:=a.z[j]/com.z[0];
    for i:=j+1 to com.d+j do
      a.z[i]:=a.z[i]-com.z[i-j]*a.z[j];
    end;
    for j:=0 to b.d do begin
      b.z[j]:=b.z[j]/com.z[0];
      for i:=j+1 to com.d+j do
        b.z[i]:=b.z[i]-com.z[i-j]*b.z[j];
      end;
      a1.d:=a.d;
      b1.d:=b.d;
      for i:=0 to a.d do a1.z[i]:=a.z[i];
      for i:=0 to b.d do b1.z[i]:=b.z[i];
    end;
  }
  =====
begin
  inipoly;
  transformation;
  if a.d>0 then begin
    com.d:=a.d;
    for i:=0 to a.d do com.z[i]:=a.z[i];
    division;
  end;
  regulator;
}

```

```
with specif do
  if minr then begin
    r.d:=r1.d; for i:=0 to r.d do r.z[i]:=r1.z[i];
    s.d:=s1.d; for i:=0 to s.d do s.z[i]:=s1.z[i];
    if (p.d<a.d+b.d) and (s.d>a.d-1) then s.d:=a.d-1;
  end
  else begin
    r.d:=r2.d; for i:=0 to r.d do r.z[i]:=r2.z[i];
    s.d:=s2.d; for i:=0 to s.d do s.z[i]:=s2.z[i];
    if (p.d<a.d+b.d) and (r.d>b.d-1) then r.d:=b.d-1;
  end;
end;
```