ปีการศึกษา 2531

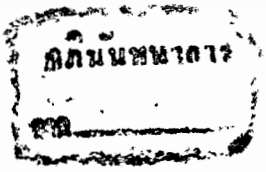
การควบคุมแบบปรับจูนตัวเองชนิดหลายตัวแปร
(Multivariable Self Tuning Control)

โดย

ณพวุฒิ มาลยเวช
อุดมชัย แช่กี่

อาจารย์ที่ปรึกษา

ร.ศ.วิพันธ์ ปรีชาพานิช



ปริญญานิพนธ์ปีการศึกษา 2531

ภาควิชา วิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมแบบจูนปรับตัวเองชนิดหลายตัวแปร
(Multivariable Self Tuning Control)

ผู้จัดทำ

1. นายณพวุฒิ มาลยเวช 281069
2. นายอุดมชัย แซ่กี่ 281318

.....อาจารย์ที่ปรึกษา
(ร.ศ.วิพันธ์ ปรีชาพานิช)

การควบคุมแบบจูนปรับตัวเองชนิดหลายตัวแปร

ณพวุฒิ มาลยเวช
อุดมชัย แซ่กี่
ร.ศ.วิพันธ์ ปรีชาพานิช อาจารย์ที่ปรึกษา
ปีการศึกษา 2531

บทคัดย่อ

ปริญญานิพนธ์นี้ ได้อธิบายถึงทฤษฎีของการควบคุมแบบจูนปรับตัวเองชนิดหลายตัวแปร โดยวิธีการควบคุมใช้การกำหนดโพลของลูปปิดให้อยู่ในตำแหน่งที่ต้องการ โดยคุณลักษณะของระบบแทนอยู่ในรูปของสมการสเตท ตลอดจนได้ทำการสร้างและทดลองตัวควบคุมต้นแบบแล้วด้วย

MULTIVARIABLE SELF TUNING CONTROL

Noppawoot Malayawech
Udomchai Saeky
Wipan Prechapanich Advisor
1988

Abstract

This paper describes the theory of a multi-input/multi-output self tuning controller where the control objective is the assignment of the closed-loop pole set to prespecified locations. Using the new form of state space representation. At the end, the prototype controller was created and tested.

สารบัญ

สารบัญรูปภาพ

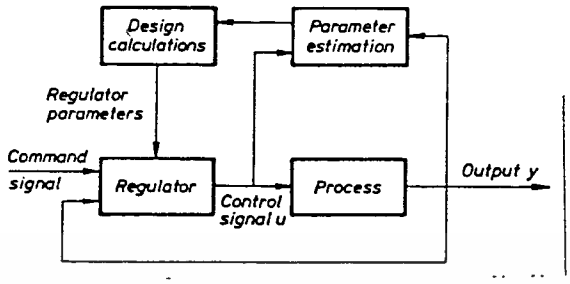
บทที่ 1
บทนำ

ในการสร้างตัวควบคุม (CONTROLLER) ขึ้นควบคุมกระบวนการ (PROCESS)อย่างหนึ่ง โดยทั่วไปเราจะต้องทราบว่ากระบวนการนั้น มีลักษณะทางกายภาพอย่างไรและจะต้องสร้างสมการดิฟเฟอเรนเชียล (DIFFERENTIAL EQUATION) ขึ้นมา ในวงจรอีเลคโทรนิค จะตั้งสมการโดยดูจากค่าความต้านทาน (RESISTANCE) , ความเหนี่ยวนำไฟฟ้า (RELUCTANCE) และความจุไฟฟ้า (CAPACITANCE) ในมอเตอร์อาจตั้งสมการจากคุณลักษณะต่างๆ เช่น ความฝืดของมอเตอร์ โดยเราคิดให้ ค่าพารามิเตอร์ (PARAMITER) ต่างๆเหล่านี้เป็นค่าคงที่ จากนั้นจึงค่อยหาตัวควบคุมขึ้นมาเพื่อทำให้ระบบนั้นได้ผลตอบสนองตามที่ต้องการ แต่จะเกิดอะไรขึ้นมาถ้าพารามิเตอร์เหล่านี้เปลี่ยนค่าไป เช่น มอเตอร์เกิดความฝืดมากขึ้นเมื่อใช้ไปนานๆ หรือ ค่าความต้านทานเปลี่ยนค่าไปตามอุณหภูมิ ผลก็คือผลตอบสนองของระบบไม่เป็นไปตามที่ออกแบบไว้

การควบคุมแบบจูนปรับตัวเอง (SELF TUNING CONTROL)เป็นวิธีหนึ่งในการสร้างตัวควบคุม ที่สามารถใช้ได้กับสภาพที่พารามิเตอร์ของระบบมีการเปลี่ยนแปลงได้ หรือบางทีไม่ทราบค่า และนอกจากนั้นยังสามารถใช้ได้กับระบบที่มีสิ่งรบกวน(DISTURBANCE) ได้อีกด้วย

วิธีการออกแบบตัวควบคุมแบบจูนปรับตัวเองนั้นง่าย และสะดวกในการนำไปประยุกต์ใช้งาน สามารถใช้ควบคุมกระบวนการที่มีการเปลี่ยนแปลงได้อย่างมีประสิทธิภาพ และสมรรถนะที่ดีกว่าการควบคุมแบบ PID แบบเดิม แม้จะไม่ได้เป็นค่าที่เล็งเลิศ (OPTIMAL) ก็ตาม บางครั้งการควบคุมแบบจูนปรับตัวเอง อาจไม่สามารถควบคุมระบบได้ เนื่องจากยังไม่มีทฤษฎีรับรองว่าวิธีการนี้จะลู่เข้า (CONVERGE) เสมอ เพียงแต่รับรองว่าถ้าวิธีการทั้งหมดลู่เข้า ผลตอบสนองของระบบ จะลู่เข้าด้วยเท่านั้น ปัญหาของวิธีนี้ยังมีอีก เช่น อัตราการลู่เข้าไม่แน่นอน ทำให้ไม่ทราบว่า ระบบจะลู่เข้าเมื่อใด หรือแม้ว่าการควบคุมจะอยู่ในเงื่อนไขที่เหมาะสมแล้ว แต่การควบคุมอาจลู่ออก(DIVERGE)ก็ได้

การควบคุมแบบจูนปรับตัวเอง จะมีโครงสร้างดังรูปที่ 1.1



รูปที่ 1.1 โครงสร้างการควบคุมแบบจูนปรับตัวเอง

จากรูปจะเห็นได้ว่า วิธีการในการควบคุมประกอบด้วยสองขั้นตอน คือ

1. การประมาณค่าพารามิเตอร์ มีวิธีการหลายวิธีซึ่งใช้กับรูปแบบต่างๆกัน เช่น

- RLS (RECURSIVE LEAST SQUARE) เป็นการประมาณค่าพารามิเตอร์โดยที่สิ่งรบกวนมีค่าเฉลี่ย เป็น 0
- RML (RECURSIVE MAXIMUM LIKELIHOOD) เป็นการประมาณค่าพารามิเตอร์โดยที่สิ่งรบกวนมีค่าเฉลี่ย ไม่เป็น 0
- ELS (EXTENDED LEAST SQUARE) เป็นการประมาณค่าพารามิเตอร์โดยที่สิ่งรบกวนมีค่าเฉลี่ย ไม่เป็น 0 โดยดัดแปลงมาจาก RLS

2. หลังจากได้ประมาณค่าพารามิเตอร์แล้วก็จะต้องออกแบบตัวควบคุม มีได้หลายวิธี เช่น

- ตัวควบคุมแบบจูนปรับตัวเองโดยใช้วิธีการควบคุมแบบเล็งเลิศ
- ตัวควบคุมแบบจูนปรับตัวเองโดยใช้วิธีกำหนดโพล (POLE - PLACEMENT) ซึ่งใช้ได้ทั้งแบบอิมพลิซิท (IMPLICIT) และ แบบเอ็กพลิซิท - (EXPLICIT)
- ตัวควบคุมแบบจูนปรับตัวเองโดยใช้วิธี PID จูนตามวิธีของ ซีกเล่อร์ และ นิโคล (Ziegler & Nichols)

เมื่อปีการศึกษาที่แล้ว (2530) ได้มีนักศึกษารุ่นพี่กลุ่มหนึ่งได้ทำการศึกษา ตัวควบคุมแบบนี้ไว้บ้างแล้วส่วนหนึ่ง จากวิทยานิพนธ์มีขอบเขตสรุปว่าได้ศึกษาระบบอินพุทเดียว-เอาท์พุตเดียว โดยใช้โปรแกรมคอมพิวเตอร์เป็นตัวควบคุม โปรแกรมนี้เขียนขึ้นบนเครื่องคอมพิวเตอร์ IBM PC โดยจำลองกระ

บวนการจริงให้เป็นวงจรอีเลคโทรนิค แล้วให้ IBM PC ทำการอ่านค่าที่ต้องการเข้ามาทางพอร์ท (PORT) โดยผ่านตัวแปลงสัญญาณต่อเนื่องไปเป็นสัญญาณดิจิตอล (A/D CONVERTER) หลังจากนั้นโปรแกรมจะหาสัญญาณควบคุมที่เหมาะสมส่งออกไปทางพอร์ท โดยผ่านตัวแปลงสัญญาณดิจิตอลไปเป็นสัญญาณต่อเนื่อง (D/A CONVERTER) เข้าไปยังกระบวนการ

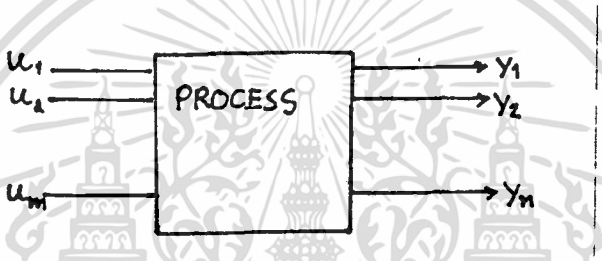
โครงงานนี้สามารถทำงานได้ดี แต่ก็ยังมีอีกหลายเรื่องที่ยังศึกษาได้ไม่หมด เนื่องจากเวลามีไม่พอ ดังนั้น โครงงานที่เราจะทำนี้จึงศึกษาถึงการออกแบบ ตัวควบคุมต่อจากวิทยานิพนธ์เดิม โดยใช้สมการเชิงปริภูมิสถานะ (STATE SPACE EQUATION) ช่วย เพราะสามารถใช้กับระบบหลายอินพุต-หลายเอาท์พุตได้ ในการทดสอบกระบวนการจะใช้โปรแกรมเป็นตัวควบคุมโดยเขียนโปรแกรม บน IBM PC ทำการควบคุมกระบวนการ ทดสอบซึ่งจำลองโดยใช้โปรแกรมเช่นเดียวกัน ทั้งนี้และทั้งนั้นเพื่อเป็นแนวทางในการศึกษาของนักศึกษารุ่นต่อไปอีก

บทที่ 2
ทฤษฎีและหลักการ

2.1 การประมาณค่าพารามิเตอร์

เราจะเลือกใช้วิธี RLS(RECURSIVE LEAST SQUARE) โดยวิธี RLS จะพิสูจน์มาจาก LS(LEAST SQUARE) ดั้งนั้นจะอธิบายวิธี LS ก่อน แล้วจึงจะอธิบาย RLS ต่อ

2.1.1 การประมาณค่าแบบ LS



รูปที่ 2.1 ระบบหลายอินพุท-หลายเอาท์พุท

สามารถเขียนสมการได้ดังนี้

$$\begin{aligned}
y_1 &= a_{11}u_1 + \ldots + a_{1j}u_j + \ldots + a_{1m}u_m \\
y_2 &= a_{21}u_1 + \ldots + a_{2j}u_j + \ldots + a_{2m}u_m \\
&\vdots \\
y_n &= a_{n1}u_1 + \ldots + a_{nj}u_j + \ldots + a_{nm}u_m
\end{aligned} \qquad (2.1)$$

สามารถเขียนในรูปเวคเตอร์ได้ดังนี้

$$Y = A\,U \qquad (2.2)$$

โดยกำหนดให้

$$Y = (y_1 \ldots y_i \ldots y_n)^T \qquad (2.3)$$

$$U = (u_1 \ldots u_j \ldots u_m)^T \qquad (2.4)$$

และ

$$\mathcal{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \qquad (2.5)$$

ในแต่ละแถวของสมการ (2.1) จะมีรูปแบบปัญหาเป็นเอาท์พุทเดียว ดังนั้นเราจะสามารถเขียนสมการ (2.1) ใหม่ได้ดังสมการ (2.6)

$$y_i = U^T A_i \qquad (2.6)$$

โดยที่

$$A_i = [a_{i1} \; a_{i2} \ldots a_{im}]^T \qquad (2.7)$$

จากสมการ (2.6),(2.7) เราจะประมาณค่า A_i ใดๆได้นั้นเราต้องมีจำนวนชุดของการวัด y_i, u_j $(i=1,\ldots,n \; ; \; j=1,\ldots,m)$ อย่างน้อยเท่ากับ m

หรือ $r >= m$ เมื่อ r เป็นจำนวนชุดของการวัด

และ เราจะนิยามได้ว่า

$$\mathcal{Y}_i = \begin{bmatrix} y_i(1) \\ \cdot \\ \cdot \\ \cdot \\ y_i(r) \end{bmatrix} \qquad (2.8)$$

$$\mathcal{U} = \begin{bmatrix} u_1(1) & \cdots & u_m(1) \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ u_1(r) & \cdots & u_m(r) \end{bmatrix} = \begin{bmatrix} U^T(1) \\ \cdot \\ \cdot \\ \cdot \\ U^T(r) \end{bmatrix} \qquad (2.9)$$

จากสมการ (2.8),(2.9) เราจะเขียนสมการ (2.6) ได้ใหม่ว่า

$$y_i = \mathcal{U} A_i \qquad (2.10)$$

ให้ $\hat{A}_i$ เป็นค่าประมาณของ A_i เราจะได้

$$\hat{y}_i = \mathcal{U} \hat{A}_i \qquad (2.11)$$

โดย $\hat{y}_i$ เป็นค่าเอาท์พุทที่ได้จากการประมาณค่าพารามิเตอร์

เพื่อที่จะให้ค่าประมาณของ A_i ให้ใกล้เคียงมากที่สุด เราจะต้องทำให้ค่าเอาท์พุทที่ได้จากการประมาณ มีค่าแตกต่างจากเอาท์พุทจริงน้อยที่สุดนั่นเอง จึงได้มีการปัญหาโดยใช้วิธีกำลังสองน้อยที่สุด (LEAST SQUARE)

นั่นคือ $\hat{A}_i$ จะต้องทำให้ดัชนีราคา (COST FUNCTION) J น้อยที่สุด

เมื่อ $$J(i) = \sum_{i=1} [y_i - \mathcal{U} \hat{A}_i]^2 \qquad (2.12)$$

โดยที่ $\hat{A}_i$ ต้องสอดคล้องกับ $\partial J(i)/\partial \hat{A}_i = 0 \qquad (2.13)$

ซึ่งจะได้ว่า $$\hat{A}_i = (\mathcal{U}^T \mathcal{U})^{-1} . \mathcal{U}^T y_i \qquad (2.14)$$

2.1.2 การประมาณค่าแบบ RLS

RLS จะดีกว่า LS คือ เราสามารถนำข้อมูลจากการประมาณครั้งก่อนๆ นำมาใช้ได้จึงช่วยลดเวลาในการคำนวณ เราจะให้ $\hat{A}_i(k)$ เป็นการประมาณค่าแบบ LS เมื่อมีการวัดข้อมูล k ชุด เทอม U และ Y_i ก็จะกลายเป็น

$$U(k) = \begin{bmatrix} U^T(1) \\ \vdots \\ U^T(k) \end{bmatrix}, \quad Y_i(k) = \begin{bmatrix} y_i(1) \\ \vdots \\ y_i(k) \end{bmatrix}$$

สมการ (2.14) เขียนใหม่เป็น

$$\hat{A}_i(k) = \left[U^T(k)U(k)\right]^{-1} U^T(k)Y_i(k) \qquad (2.15)$$

เมื่อมีการวัดเพิ่มขึ้นมาอีก เราสามารถเขียนเทอมต่างๆ ได้เป็น

$$U(k+1) = \begin{bmatrix} U(k) \\ U(k+1) \end{bmatrix}, \quad Y_i(k+1) = \begin{bmatrix} Y_i(k) \\ y_i(k+1) \end{bmatrix}$$

และ

$$\begin{aligned} \hat{A}_i(k+1) &= \left[U^T(k+1)U(k+1)\right]^{-1} U^T(k+1)Y_i(k+1) \\ &= \left[U^T(k)U(k) + U(k+1)U^T(k+1)\right]^{-1}\left[U^T(k)Y_i(k) + U(k+1)y_i(k+1)\right] \end{aligned} \qquad (2.16)$$

จะได้ว่า

$$P(k+1) = [\,\mathcal{U}^T(k+1)\,\mathcal{U}(k+1)\,]^{-1}$$

$$= [\,\mathcal{U}^T\mathcal{U} + UU^T]^{-1}$$

$$= (\mathcal{U}^T\mathcal{U})^{-1}$$

$$- (\mathcal{U}^T\mathcal{U})^{-1}U^T[1 + U^T(\mathcal{U}^T\mathcal{U})^{-1}U\,]^{-1}U^T(\mathcal{U}^T\mathcal{U})^{-1}$$

$$= P(k) - P(k)U^T[1 + U^TP(k)U\,]^{-1}U^TP(k) \qquad (2.21)$$

สมการ (2.19) มาเขียนใหม่เป็น

$$\hat{A}_i(k+1) = \hat{A}_i(k)$$

$$- P(k+1)U(k)\,[y_i(k+1) - U^T(k)\hat{A}_i(k)\,] \qquad (2.22)$$

สมการ (2.20) ถึง (2.22) จะใช้ประมาณค่าพารามิเตอร์

เพื่อที่จะทำให้สมการ (2.16) ดีขึ้น เราจะเอา k และ k+1 ที่อยู่ทางขวามือออกไป สมการ (2.16) เขียนใหม่เป็น

$$\hat{A}_i(k+1) = [\,U^TU + UU^T\,]\;[\,U^Ty_i + Uy_i\,] \qquad (2.17)$$

$$= (U^TU)^{-1}U^Ty_i + [(U^TU + UU^T)^{-1} - (U^TU)^{-1}\,]U^Ty_i$$

$$+ (U^TU + UU^T)^{-1}Uy_i \qquad (2.18)$$

เทอม $(U^TU)^{-1}U^Ty_i$ ก็คือ $\hat{A}_i(k)$

และ $[\,(U^TU + UU^T)^{-1} - (U^TU)^{-1}\,]U^Ty_i$

$$= (U^TU + UU^T)^{-1}\,(U^TU - U^TU - UU^T)(U^TU)^{-1}U^Ty_i$$

$$= -(U^TU + UU^T)^{-1}\,(UU^T)(U^TU)^{-1}U^Ty_i$$

$$= -(U^TU + UU^T)^{-1}\,(UU^T)\hat{A}_i(k)$$

สมการ (2.18) กลายเป็น

$$\hat{A}_i(k+1) = \hat{A}_i(k) - (U^TU + UU^T)^{-1}\,(UU^T)\hat{A}_i(k)$$

$$- + (U^TU + UU^T)^{-1}\,Uy_i$$

$$= \hat{A}_i(k) - (U^TU + UU^T)^{-1}\,U[\,y_i - U^T\hat{A}_i(k)\,] \qquad (2.19)$$

เทอม $(U^TU + UU^T)$ ก็คือ

$$U^T(k)\,U(k) + U(k)U^T(k) = U^T(k+1)\,U(k+1)$$

ถ้าเรากำหนดให้ $P(k) = [\,U^T(k)\,U(k)\,]^{-1}$ (2.20)

2.2 กฎการควบคุมโดยวิธีกำหนดโพล (POLE PLACEMENT)

พิจารณาระบบ n สถานะ (state) - p อินพุท (input) ดังนี้

$$zX = AX+BU \tag{2.23}$$

โดยที่

X : n x 1 เวกเตอร์สถานะ (state vector)

U : p x 1 อินพุทของระบบ (process input)

A : n x n พารามิเตอร์

B : n x p พารามิเตอร์

ถ้าเราเลือก

$$U = R+KX \tag{2.24}$$

R : p x 1 อินพุทอ้างอิง (reference input)

K : p x n อัตราขยายป้อนกลับ (feedback gain)

เราจะได้

$$zX = (A+BK)X + BR$$

เป็นที่ทราบกันดีว่า ไอเกนวาลู (eingen value) ของ (A+BK) จะเป็นโพล (pole) ของระบบ ถ้าเลือกค่า K ที่เหมาะสม ก็จะทำให้สามารถกำหนด โพล ของระบบได้ตามความต้องการ

วิธีหนึ่งในการหาค่า K ก็คือ หาเมตริกซ์ A" ซึ่งทราบว่ามีไอเกนวาลูเป็นไปตามความต้องการ แล้วหาค่า K ที่ทำให้ A+BK=A" ในกรณีนี้ถ้า B เป็นเมตริกซ์จตุรัส (p=n) เราสามารถหาค่า K ได้จากการแก้สมการเชิงเส้น n ตัวแปร BK = A"-A แต่ถ้า B ไม่ใช่เมตริกซ์จตุรัส (p<n) เราจำเป็นต้องแปลงสมการ(2.23) ให้อยู่ในรูปแบบที่สามารถควบคุมได้ (controllable form) ก่อน ซึ่งเมื่อแปลงแล้วจะทำให้ แรงค์ (rank) ของ B ที่แปลงแล้วเท่ากับ p ซึ่งวิชาคณิตศาสตร์ยืนยันว่าสามารถหาค่า K ได้จากสมการเชิงเส้น A+BK=A"

ต่อไปนี้จะกล่าวถึงการหาค่า K เมื่อการทำการแปลงรูปสมการ (2.23)

ถ้าได้เมตริกซ์ P ซึ่งทำให้ X'= PX ระบบสมการใหม่จะสามารถเขียนได้ดังนี้

$$zX' = PAP^{-1}X'+PBU$$

$$U = R+KP^{-1}X$$

ถ้าให้

$$A' = PAP^{-1}$$

$$B' = PB$$

$$K' = KP^{-1} \qquad (2.25)$$

เราจะได้

$$zX' = (A'+B'X')K'+B'R$$

และถ้าให้ A" มีไอเกนวาลูตามความต้องการ ถ้าสามารถหาค่า K' ซึ่งทำให้

$$A'+B'K'=A"$$

ก็จะสามารถหา K ได้จาก K=K'P (สมการ 2.25)

วิธีหนึ่งในการหา เมตริกซ์แปลงรูป (transformation matrix) P คือ

1) B สามารถเขียนได้ในรูป

$$B = [b_1 | b_2 | \ldots | b_p]$$

โดย b_i เป็น คอลัมเวกเตอร์ (column vector) ที่ i ของ B

2) หา เมตริกซ์ที่สามารถควบคุมได้ (controllable matrix) C ได้จาก

$$C = [B|AB|A^2B|\ldots|A^{n-1}B]$$

หรือ

$$C = [b_1|b_2|\ldots|b_p|Ab_1|\ldots|Ab_p|\ldots|A^{n-1}B_1|\ldots|Ab^{n-1}b_p]$$

3) เลือก เวกเตอร์อิสระ (independant vector) มา n เวกเตอร์ โดยเริ่มจาก

$$b_1, b_2, \ldots, b_p$$

แล้วตามด้วย

$$Ab_1, Ab_2, \ldots, Ab_p$$

จนกว่าจะได้ เวกเตอร์อิสระ จำนวน n vector

4) จัดเรียง C' ดังนี้

$$C' = [b_1|Ab_1|\ldots|A^{g(1)-1}b_1|b_2|\ldots|A^{g(2)-1}b_2|\ldots|b_p|\ldots A^{g(p)-1}b_p]$$

โดยการจัดหมู่จากเมตริกซ์ที่มี b_1 หมดแล้ว จึงจัดหมู่เมตริกซ์ที่มี b_2 เรียงกันไปจนครบ และในขั้นตอนนี้เราก็จะได้ค่า $g(1)\ldots g(p)$ จากการจัดหมู่ว่าจัดหมู่ทีละกี่เวกเตอร์

กำหนด

$$d(1) = g(1)$$

$$d(2) = g(1)+g(2)$$

$$d(k) = g(1)+g(2)+\ldots+g(k) \qquad k = 1,2,\ldots,p$$

ให้ p_1 เท่ากับแถวที่ $d(i)$ ของ C^{-1}

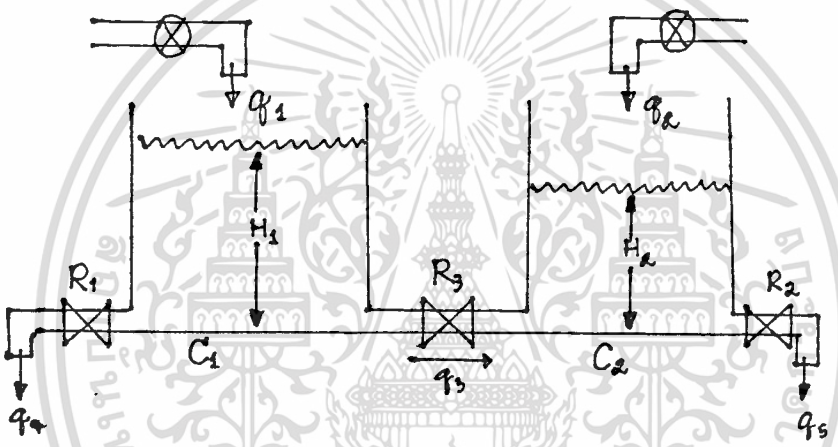
เราสามารถหาค่า P ได้จาก

$$P = \begin{bmatrix} p_1 \\ p_1A \\ \vdots \\ p_1A^{g(1)-1} \\ p_2 \\ \vdots \\ p_2A^{g(2)-1} \\ \vdots \\ p_pA^{g(p)-1} \end{bmatrix}$$

บทที่ 3
การประยุกต์ใช้งานทฤษฎี

3.1 กระบวนการ (process) ที่จะทดลองควบคุม

เป็นกระบวนการหลายอินพุต-หลายเอาท์พุท (MULTI INPUT - MULTI OUTPUT , MIMO) อย่างง่ายดังรูป



รูปที่ 3.1 กระบวนการที่จะทดลองควบคุม

กระบวนการนี้ ประกอบด้วยถังน้ำ ขนาดความจุต่างกันคือ C1,C2 ทั้งสองมีท่อน้ำ เพื่อเป็นทางให้น้ำไหลออกซึ่งมีค่าคงที่ความต้านทานคือ R1,R2 ตามลำดับ ถังน้ำทั้งสองต่อถึงกันด้วยท่อน้ำ ที่มีค่าความต้านทาน R3

ในการควบคุมระบบ เราต้องการควบคุมระดับน้ำทั้งสองถังอย่างอิสระ ซึ่งการจะควบคุมระดับนี้ ทำได้โดยการปรับค่าของอัตราการไหลเข้าของน้ำ Q1,Q2 ซึ่งสังเกตได้ว่าจะมีเอาท์พุท 2 ตัวคือ H1,H2 และมีอินพุทสองตัวคือ Q1,Q2 ระบบนี้จะเป็นระบบ 2 อินพุท - 2 เอาท์พุท

จากระบบที่ใช้ทดลอง เราสามารถตั้งสมการอธิบายระบบได้ดังนี้

$$\frac{dH1}{dt} = -\frac{1}{C1}\left(\frac{1}{R1}+\frac{1}{R3}\right)H1 + \frac{1}{C1 \cdot R3}H2 + \frac{Q1}{C1}$$

$$\frac{dH2}{dt} = \frac{1}{C2 \cdot R3}H1 - \frac{1}{C2}\left(\frac{1}{R2}+\frac{1}{R3}\right)H2 + \frac{Q2}{C2}$$

ซึ่งเขียนในรูปสมการสเตทได้ดังนี้

$$\begin{bmatrix} H1(k+1) \\ H2(k+1) \end{bmatrix} = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} \begin{bmatrix} H1(k) \\ H2(k) \end{bmatrix} + \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix} \begin{bmatrix} Q1(k) \\ Q2(k) \end{bmatrix}$$

หรือ

$$zX = AX+BU$$

$$X = \begin{bmatrix} H1(k) \\ H2(K) \end{bmatrix}$$

$$U = \begin{bmatrix} Q1(k) \\ Q2(K) \end{bmatrix}$$

3.2 การประยุกต์ใช้งานทฤษฎีการประมาณค่าพารามิเตอร์

จากสมการ

$$\begin{bmatrix} H1(k+1) \\ H2(k+1) \end{bmatrix} = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} \begin{bmatrix} H1(k) \\ H2(k) \end{bmatrix} + \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix} \begin{bmatrix} Q1(k) \\ Q2(k) \end{bmatrix}$$

สมการแรกเขียนได้ดังนี้

$$H1(k+1) = A11*H1(K) + A12*H2(k) + B11*Q1(k) + B12*Q2(k)$$

ซึ่งกระจายได้เป็น

$$H1(\ 2\ ) = A11*H1(1) + A12*H2(1) + B11*Q1(1) + B12*Q2(1)$$

$$H1(\ 3\ ) = A11*H1(2) + A12*H2(2) + B11*Q1(2) + B12*Q2(2)$$

. . . . .

. . . . .

$$H1(N+1) = A11*H1(N) + A12*H2(N) + B11*Q1(N) + B12*Q2(N)$$

สมการที่สองเขียนได้ดังนี้

$$H2(k+1) = A21*H1(K) + A22*H2(k) + B21*Q1(k) + B22*Q2(k)$$

ซึ่งกระจายได้เป็น

$$H2(\ 2\ ) = A21*H1(1) + A22*H2(1) + B21*Q1(1) + B22*Q2(1)$$

$$H2(\ 3\ ) = A21*H1(2) + A22*H2(2) + B21*Q1(2) + B22*Q2(2)$$

. . . . .

. . . . .

$$H2(N+1) = A21*H1(N) + A22*H2(N) + B21*Q1(N) + B22*Q2(N)$$

ทั้งสองสมการเขียนใหม่ในรูปเมตริกซ์ได้ดังนี้

$$\begin{bmatrix} H1(2) & H2(2) \\ H1(3) & H2(3) \\ . & . \\ . & . \\ H1(N+1) & H1(N+1) \end{bmatrix} = \begin{bmatrix} H1(1) & H2(1) & Q1(1) & Q2(1) \\ H1(2) & H2(2) & Q1(2) & Q2(2) \\ . & . & . & . \\ . & . & . & . \\ H1(N) & H2(N) & Q1(N) & Q2(N) \end{bmatrix} \begin{bmatrix} A11 & A21 \\ A12 & A22 \\ B11 & B21 \\ B12 & B22 \end{bmatrix}$$

การประมาณค่า A11...A22,B11...B22

จะต้องทราบค่า H1(1)...H1(N+1)

H2(1)...H2(N+1)

Q1(1)...Q1(N)

Q2(2)...Q2(N)

และใช้ทฤษฎีการประมาณค่าพารามิเตอร์จากบทที่ 2 โดยเปรียบเทียบกันได้ดังนี้

$$y_i(k+1) = [\ H1(k+1)\quad H2(k+1)\ ]$$

$$U^T(k) = [\ H1(k)\quad H2(k)\quad Q1(k)\quad Q2(k)\ ]$$

และ

$$A_i(k) = \begin{bmatrix} A11 & A21 \\ A12 & A22 \\ B11 & B21 \\ B12 & B22 \end{bmatrix}$$

3.3 การใช้งานทฤษฎีการกำหนดโพล

3.3.1 การคำนวณหาค่าเมตริกซ์ R

สำหรับระบบ $zX=AX+BU$

และ $Y=CX$ ----------(3.1)

เมื่อเราเลือก $U=R+KX$

จะได้ $zX=(A+BK)X+BR$ ----------(3.2)

A : n x n Real Matrix

B : n x p Real Matrix

C : p x n Real Matrix

X : n x 1 State Vecter

U : p x 1 Process Input

Y : p x 1 Process Output

ถ้ากำหนด SP เป็นเมตริกซ์เป้าหมาย โดยที่ SP : p x 1 เราจะสามารถหาค่า R ได้ดังนี้

จากนิยามของ SP คือ SP เป็นค่าที่ต้องการให้กระบวนการเป็นไปเมื่อกระบวนการเข้าสู่สภาวะอยู่ตัวแล้ว (steady state) ตามปรกติที่สภาวะอยู่ตัวนั้น $X(k+1)=X(k)$ [นั่นคือ $zX=X$] และ $Y=SP$

จาก (3.2) $zX = (A+BK)X+BR$

$X = (A+BK)X+BR$

$X-(A+BK)X = BR$

$[I-(A+BK)]X = BR$

$X = [I-(A+BK)]^{-1}BR$ --------(3.3)

จาก (3.1) $Y = CX$

$SP = CX$

แทนค่า X ด้วย (3.3) $SP = C[I-(A+BK)]^{-1}BR$

$(C[I-(A+BK)]^{-1}B)^{-1}SP = R$

3.3.2 วิธีการหาเมตริกซ์ที่มีไอเกนวาลูตามที่กำหนด

เนื่องจากมีเมตริกซ์มากมายที่มีไอเกนวาลูชุดเดียวกัน ดังนั้น เราจะเสนอวิธีหนึ่งที่จะหาเท่านั้น

สำหรับค่าไอเกนวาลูที่เป็นจำนวนจริงทั้งหมด สามารถหาได้ง่ายๆ โดยการให้สมาชิกนอกเส้นทะแยงมุมเป็น ศูนย์ (0) ทั้งหมด ส่วนสมาชิกในเส้นทะแยงมุม มีค่าเท่ากับไอเกนวาลูแต่ละตัว เช่น

$$\text{เมตริกซ์A} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad \text{มีไอเกนวาลู} = -1, 2, 5$$

ถ้าไอเกนวาลูเป็นจำนวนเชิงซ้อนปนอยู่ จะให้เป็นการรวมกันของเมตริกซ์ 2 x 2 ที่มีไอเกนวาลูเป็นจำนวนเชิงซ้อน กับ เมตริกซ์ที่มีไอเกนวาลูเป็นจำนวนจริง

สำหรับเมตริกซ์ที่มีไอเกนวาลูเป็นจำนวนเชิงซ้อน a+bj สามารถหาได้ในรูปแบบดังนี้

$$A = \begin{bmatrix} 0 & -1 \\ -(a^2+b^2) & 2a \end{bmatrix}$$

เช่นถ้ากำหนดไอเกนวาลู = 0.2+0.3j , 0.2-0.3j จะได้

$$A = \begin{bmatrix} 0 & 1 \\ 0.13 & 0.4 \end{bmatrix}$$

<u>พิสูจน์</u> ระบบที่มี ไอเกนวาลู z = a+bj , a-bj จะมีสมการคุณลักษณะ (charecteristic equation) ดังนี้

$$[z-(a+bj)][z-(a-bj)] = 0$$

$$z^2-2az+a^2+b^2 = 0 \quad \text{----------(3.4)}$$

ถ้าให้ $A = \begin{bmatrix} 0 & 1 \\ a_{21} & a_{22} \end{bmatrix}$ ไอเกนวาลูของ A หาจากสมการ $|zI-A|=0$

$$\left| \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ a_{21} & a_{22} \end{bmatrix} \right| = 0$$

$$\begin{vmatrix} z & -1 \\ -a_{21} & z-a_{22} \end{vmatrix} = 0$$

$$z(z-a_{22})-a_{21} = 0$$

$$z^2-a_{22}z-a_{21} = 0 \quad \text{----------(3.5)}$$

เทียบ (3.4) กับ (3.5) จะได้ $a_{21} = -[a^2+b^2]$

$$a_{22} = 2a$$

ถ้ากำหนดไอเกนวาลูเป็น -0.5 , 0.2+0.3j , 0.2-0.3j , 0.7 เมตริกซ์ที่มีไอเกนวาลูตามนี้คือ

$$A = \begin{bmatrix} [-0.5] & 0 & 0 & 0 \\ 0 & \begin{bmatrix} 0 & 1 \\ 0.13 & 0.4 \end{bmatrix} & & 0 \\ 0 & & & 0 \\ 0 & 0 & 0 & [0.7] \end{bmatrix}$$

บทที่ 4
การควบคุมโดยใช้คอมพิวเตอร์

4.1 ขั้นตอนการควบคุมสำหรับการใช้งานจริง

การควบคุมสำหรับการใช้งานจริงโดยให้คอมพิวเตอร์ทำการควบคุมกระบวนการจะมีขั้นตอนหลัก เป็นดังนี้

1 อ่านค่าเอาท์พุททั้งหมดเข้ามา (ในที่นี้คือ H1,H2)

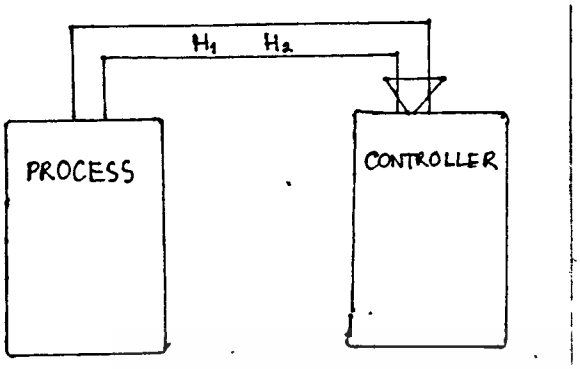
2 ทำการหาค่าสัญญาณควบคุม (ในที่นี้คือ Q1,Q2)
มีขั้นตอนย่อยคือ

- ประมาณค่าพารามิเตอร์ (PARAMITER) ของระบบ
- ใช้กฎการควบคุมเช่น การควบคุมเล็งเลิศ, การย้ายโพล ทำการคำนวณหาค่าสัญญาณควบคุม

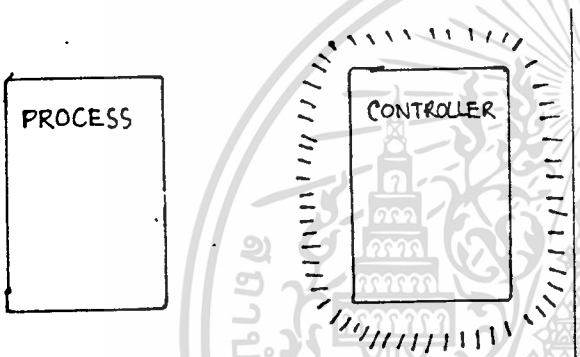
3 ส่งสัญญาณควบคุมออกไปค้างไว้ที่กระบวนการ จนกว่าจะมีการส่งสัญญาณควบคุมใหม่ออกไป

4 กลับไปทำข้อ 1 ใหม่ ซ้ำไปเรื่อยๆ

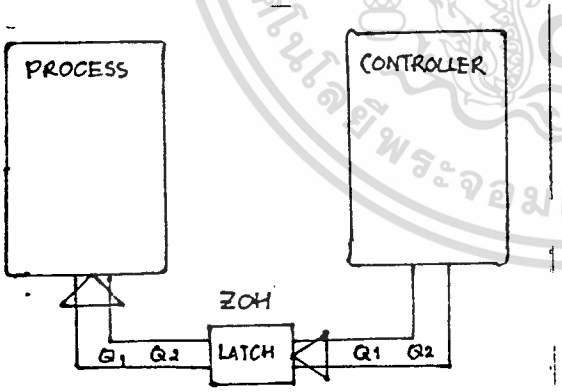
ซึ่งแสดงได้ดังรูปที่ 4.1



(ก) อ่านค่าข้อมูล



(ข) คำนวนหาสัญญาณควบคุม



(ค) ส่งสัญญาณควบคุมออกไปค้างไว้

รูปที่ 4.1 ขั้นตอนการควบคุม

แต่ในโครงงานนี้ จะเป็นการจำลองระบบโดยใช้วิธีการทางซอฟแวร์ (software) ซึ่งใช้วิธีของ รังก์ กัตตา (runge kutta)

4.2 การควบคุมกระบวนการที่ได้ทดลองสร้างขึ้น

จะใช้วิธีการทางซอฟแวร์ทั้งหมด ส่วนของโปรแกรมที่เขียนขึ้นแบ่งได้เป็น 3 ส่วนใหญ่ได้ดังนี้

1. ส่วนตัวจำลองระบบ (process simulater) ทำการแทนระบบถังน้ำในตอนต้นด้วยโปรแกรม

2. ส่วนตัวควบคุม เป็นส่วนที่แสดงวิธีการ (algorithm) ของการควบคุมแบบจูนปรับตัวเองอย่างตรงไปตรงมาตามทฤษฎีที่ได้ศึกษามา

3. ส่วนโปรแกรมย่อยด้านคำนวณเมตริกซ์ ซึ่งมีทั้งการ บวก ลบ คูณ อินเวอร์ส การแยกและการรวมเมตริกซ์ทั้งทาง แถว (row) และ สดมภ์ (column)

ในโครงงานนี้ กระบวนการ (process) มีจำนวนอินพุท=2 , เอาท์พุท=2 หรือ n=2,p=2 แต่ถ้าต้องการเปลี่ยนค่า n,p การแก้ไขทำได้โดยการเปลี่ยนค่าเพียง 2 จุดใน ตอนต้นของโปรแกรมหลักและเปลี่ยนส่วนของตัวจำลองระบบไปพร้อมกันด้วย แต่ถ้านำไปใช้กับกระบวนการจริง ก็เพียงตัดส่วนตัวจำลองระบบออก และเขียนโปรแกรมส่วนรับ-ส่งค่าซึ่งขึ้นกับฮาร์ดแวร์ ก็สามารถใช้ได้แล้ว แต่อย่างไรก็ตามจำนวนของ p,n ยังถูกจำกัดด้วยหน่วยความจำ ของคอมพิวเตอร์ด้วย โดยปรกติถ้าไม่มีการเปลี่ยนค่า data segment แล้ว ไมโครโปรเซสเซอร์8088จะสามารถเข้าถึงข้อมูลได้เพียง 64k byte เท่านั้น ในโปรแกรมจึงได้แก้ปัญหานี้โดยไม่เก็บข้อมูลส่วนใหญ่ใน data segment แต่เก็บค่าไว้ใน heap ซึ่งสามารถเข้าถึงข้อมูลได้เท่าที่จะมีหน่วยความจำเหลือในเครื่อง แต่ก็ยังมีข้อจำกัดอยู่ที่จำนวนหน่วยความจำของเครื่องที่ใช้เอง ซึ่งก็ไม่ได้ขึ้นอยู่กับโปรแกรมอีกต่อไป ปรกติเครื่อง IBM PC ที่มีหน่วยความจำ 640k หลังจากหักระบบจัดการและตัวโปรแกรมที่เราเขียนเองแล้ว จะเหลือหน่วยความจำอย่างน้อย 400k ซึ่งเพียงพอสำหรับระบบที่มี p+n=50 ได้สบายๆ

ทางด้านโปรแกรมย่อยด้านคำนวณเมตริกซ์ สามารถสั่งงานได้ง่าย ทั้งการเขียนและการอ่าน เช่น P = I-ABC (สมการเมตริกซ์) สามารถเขียนเป็น P^:=SUB(I,MUL(MUL(A,B),C))^; เพียงบรรทัดเดียว แทนการ วนลูปหลายๆลูปได้ จึงทำให้โปรแกรมส่วนตัวควบคุมที่เขียนขึ้น สามารถทำความเข้าใจได้ง่าย

แต่แม้ว่าจะมีข้อดีดังกล่าว แต่ยังมีข้อเสียเหลืออยู่คือ การจัดเก็บข้อมูล เมตริกซ์จะใช้หน่วยความจำเท่ากันหมดทุกตัว ไม่ว่าจะเป็นเมตริกซ์จะมีขนาดใหญ่หรือเล็ก ซึ่งทำให้ใช้กับระบบที่มีขนาด p+n ได้เพียง 50 เท่านั้น

ในส่วนของการกำหนดโพล จะต้องมีการตรวจสอบเงื่อนไขหลายครั้ง แต่ถ้าไปใช้กับระบบจริง อาจทำให้เกิดความไม่เท่ากันของคาบเวลาการสุ่มข้อมูล (sampling period) ได้ แต่เราก็ได้แก้ไขโดยการแทรกคำสั่งที่ไม่ก่อให้เกิดผลกระทบถึงระบบอื่น แต่สามารถทำการหน่วงเวลาได้ เช่น แทนที่จะใช้

```
IF   OKCOL[I] THEN  INC(LIVECTOR);        เราจะใช้
IF   OKCOL[I] THEN INC(LIVECTOR) ELSE INC(LDVECTOR);
```

ซึ่งตัวแปร LDVECTOR ไม่มีความจำเป็นเลยในการคำนวณการกำหนดโพล แต่ใส่ไว้เพื่อทำให้โปรแกรมใช้เวลาเท่ากัน ไม่ว่าเงื่อนไข OKCOL[I] จะเป็นจริงหรือไม่ แต่ก็ทำให้มีผลเสียคือ การคำนวณช้ามากขึ้น ประมาณว่า ใน 1 รอบการเก็บข้อมูล ต้องใช้เวลาถึง 1 วินาที การแก้ไขปัญหานี้ทำได้โดยเพิ่มความเร็วของคอมพิวเตอร์เท่านั้น

บทที่ 5

การทดลองและผลการทดลอง

ในการทดสอบตัวควบคุม เราจะวางโพลไว้ที่ z = 0.2+0.3j , 0.2-0.3j การทดลองแบ่งเป็น 3 ตอนคือ

ตอนที่ 1 เปลี่ยนแปลงค่าเป้าหมาย (setpoint) เป็นขั้น (step) และเปลี่ยนเพิ่มขึ้นตามเวลา (ramp) โดยที่ค่า พารามิเตอร์ ไม่เปลี่ยน

การเปลี่ยนค่าเป็นขั้นจะเป็นทั้งค่าเดียวและสองค่าดังนี้

เก็บข้อมูลครั้งที่ 1 , sp1=3 ; sp2=7

เก็บข้อมูลครั้งที่ 20 , sp1=6 ; sp2=5

เก็บข้อมูลครั้งที่ 40 , sp1=5 ; sp2=5

ได้ผลดังรูปที่ 5.1 , 5.2

รูปที่ 5.1 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.7

รูปที่ 5.2 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.5

การเปลี่ยนค่าเพิ่มขึ้นตามเวลาจะเปลี่ยนหลังการเก็บข้อมูลครั้งที่ 20

ได้ผลดังรูปที่ 5.3,5.4

รูปที่ 5.3 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.7

รูปที่ 5.4 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.5

ตอนที่ 2 การใส่ตัวรบกวน (disturbance) โดยที่ค่าเป้าหมาย และพารามิเตอร์ไม่เปลี่ยน โดยใส่ดังนี้

เก็บข้อมูลครั้งที่ 20-39 , dist h1=1; dist h2= 0;

เก็บข้อมูลครั้งที่ 40-59 , dist h1=0; dist h2=-1;

เก็บข้อมูลครั้งที่ 60-.. , dist h1=0; dist h2= 0;

ได้ผลดังรูปที่ 5.5

เก็บข้อมูลครั้งที่ 20 , dist h1=1; dist h2= 0;

เก็บข้อมูลครั้งที่ 40 , dist h1=0; dist h2=-1;

เก็บข้อมูลครั้งที่ 60 , dist h1=0; dist h2= 0;

ได้ผลดังรูปที่ 5.6

รูปที่ 5.5 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.7

รูปที่ 5.6 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.5

ตอนที่ 3 เปลี่ยนแปลง พารามิเตอร์ โดยที่ค่าเป้าหมายไม่เปลี่ยน
การเปลี่ยนค่าจะเป็นดังนี้

เก็บข้อมูลครั้งที่ 1 , r1=0.4 r3=0.2 r2=0.3
c1=1.0 c2=0.7

เก็บข้อมูลครั้งที่ 20 , c1=0.7

เก็บข้อมูลครั้งที่ 40 , r1=0.8

ได้ผลดังรูปที่ 5.7-5.12

รูปที่ 5.7 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.7

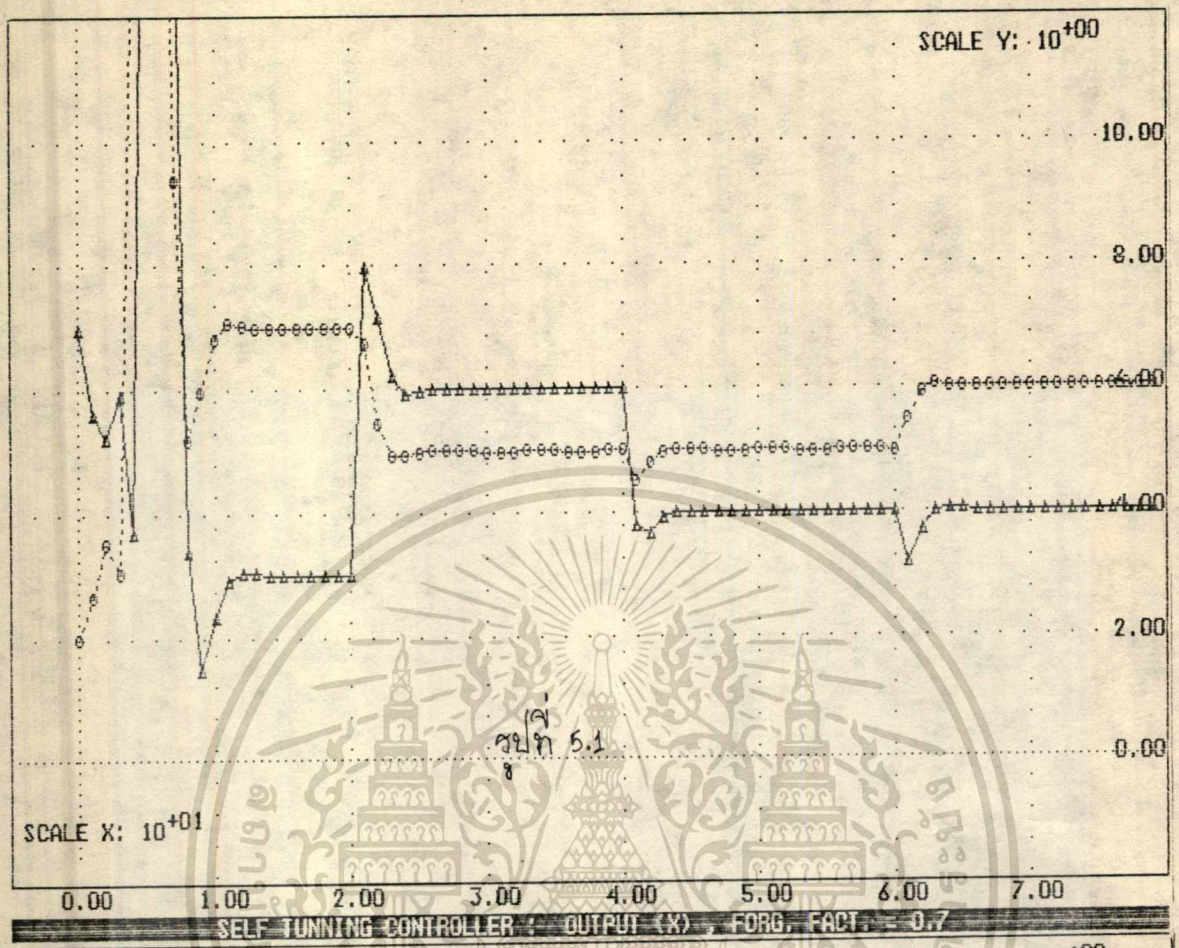
รูปที่ 5.8 : แสดงค่า h1,h2 โดย สัมประสิทธิการลืม = 0.5

รูปที่ 5.9 : แสดงค่า พารามิเตอร์ A โดย สัมประสิทธิการลืม = 0.7

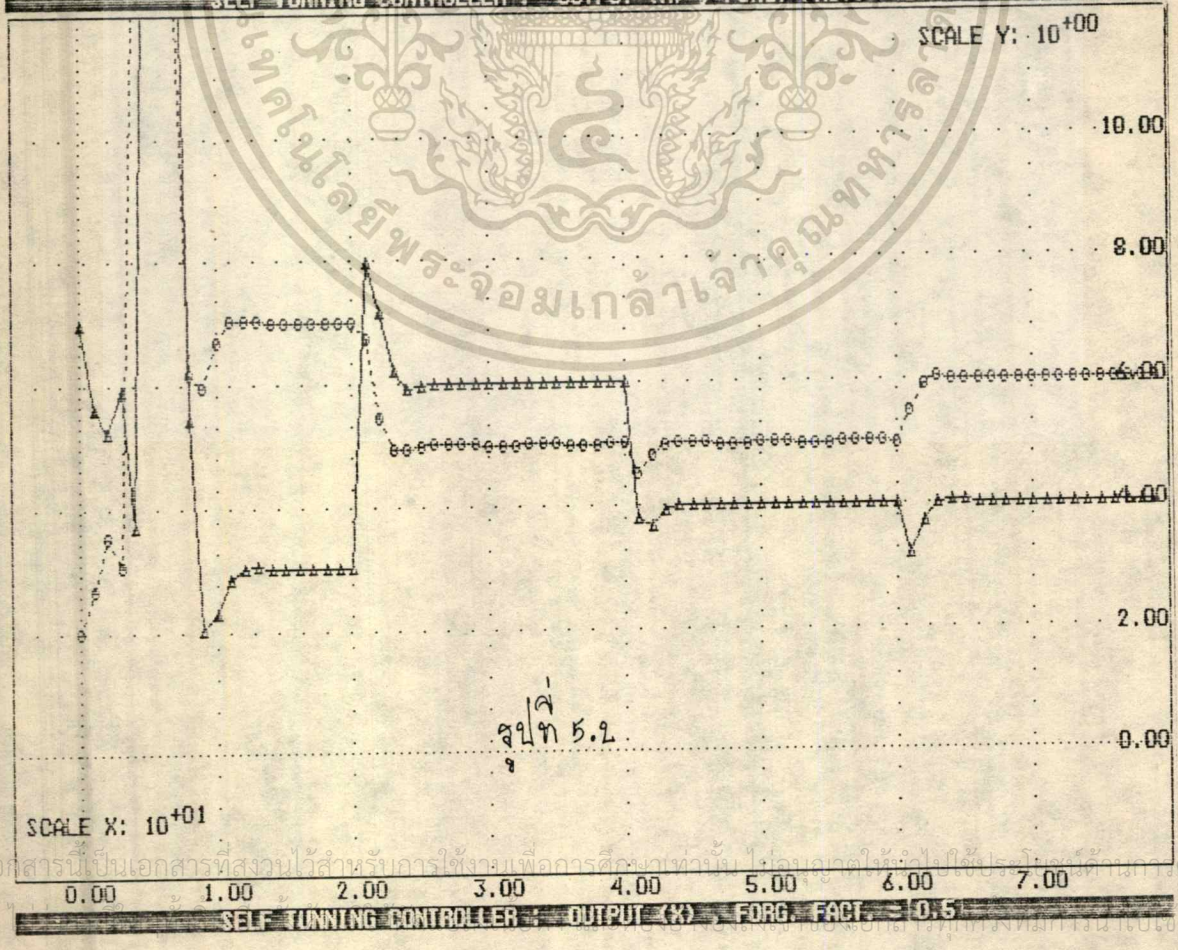
รูปที่ 5.10 : แสดงค่า พารามิเตอร์ A โดย สัมประสิทธิการลืม = 0.5

รูปที่ 5.11 : แสดงค่า พารามิเตอร์ B โดย สัมประสิทธิการลืม = 0.7

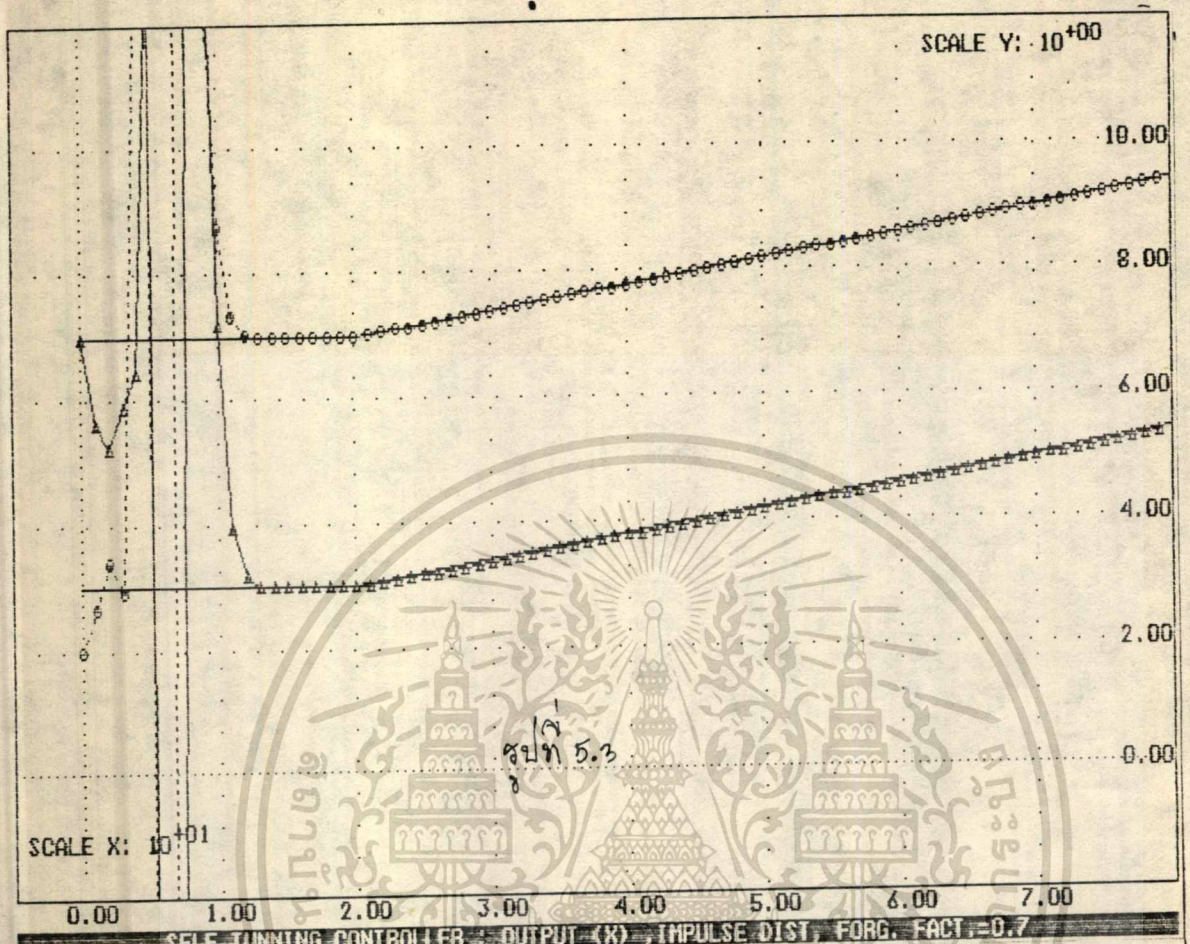
รูปที่ 5.12 : แสดงค่า พารามิเตอร์ B โดย สัมประสิทธิการลืม = 0.5



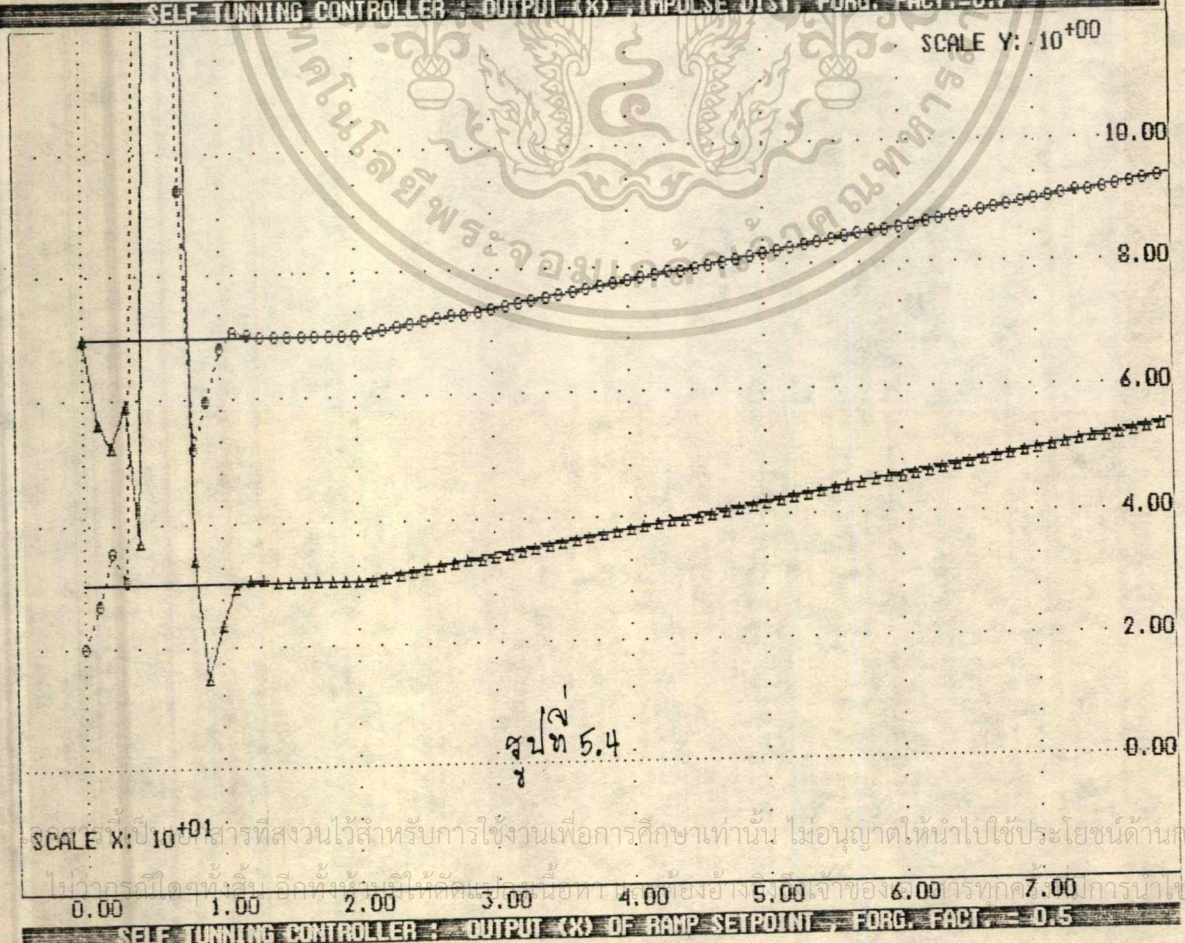
รูปที่ 5.1



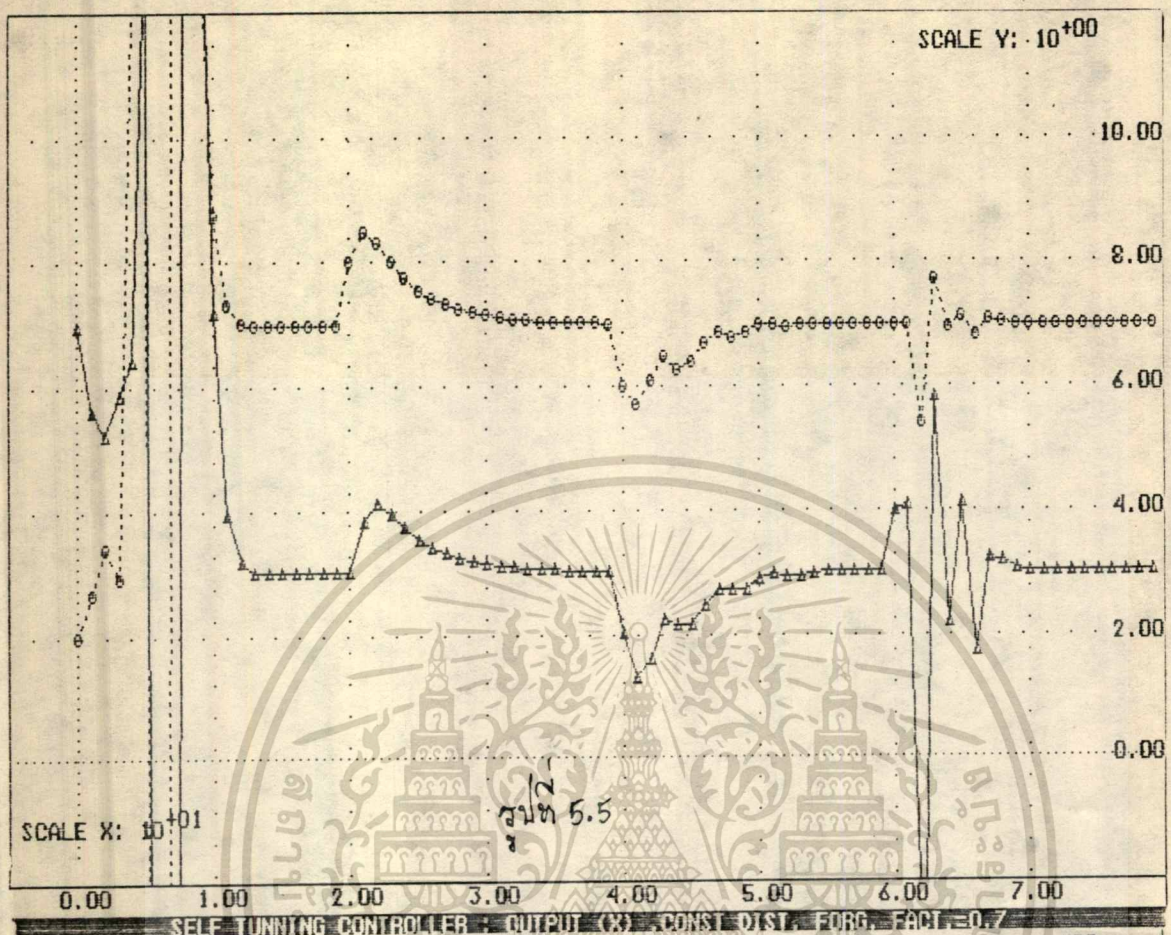
รูปที่ 5.2



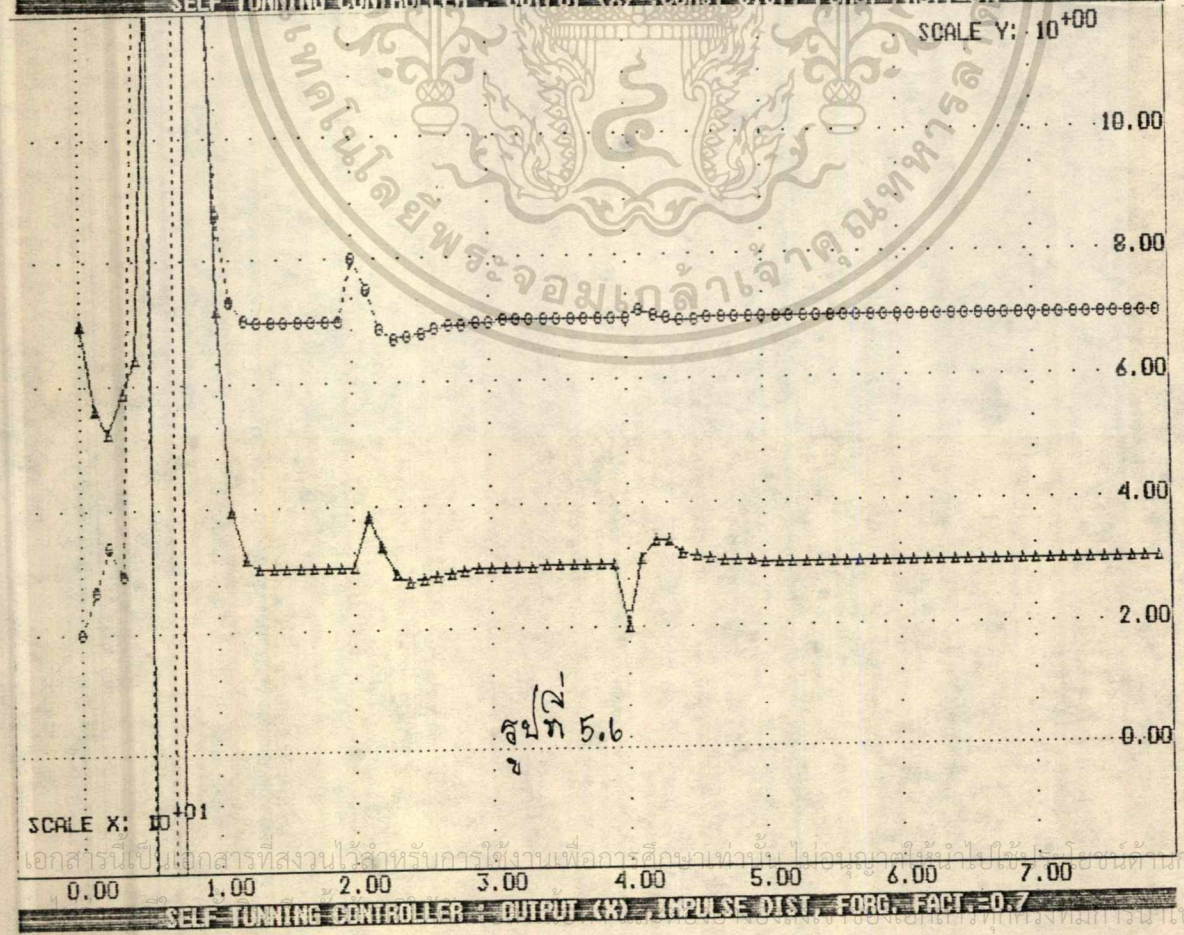
รูปที่ 5.3



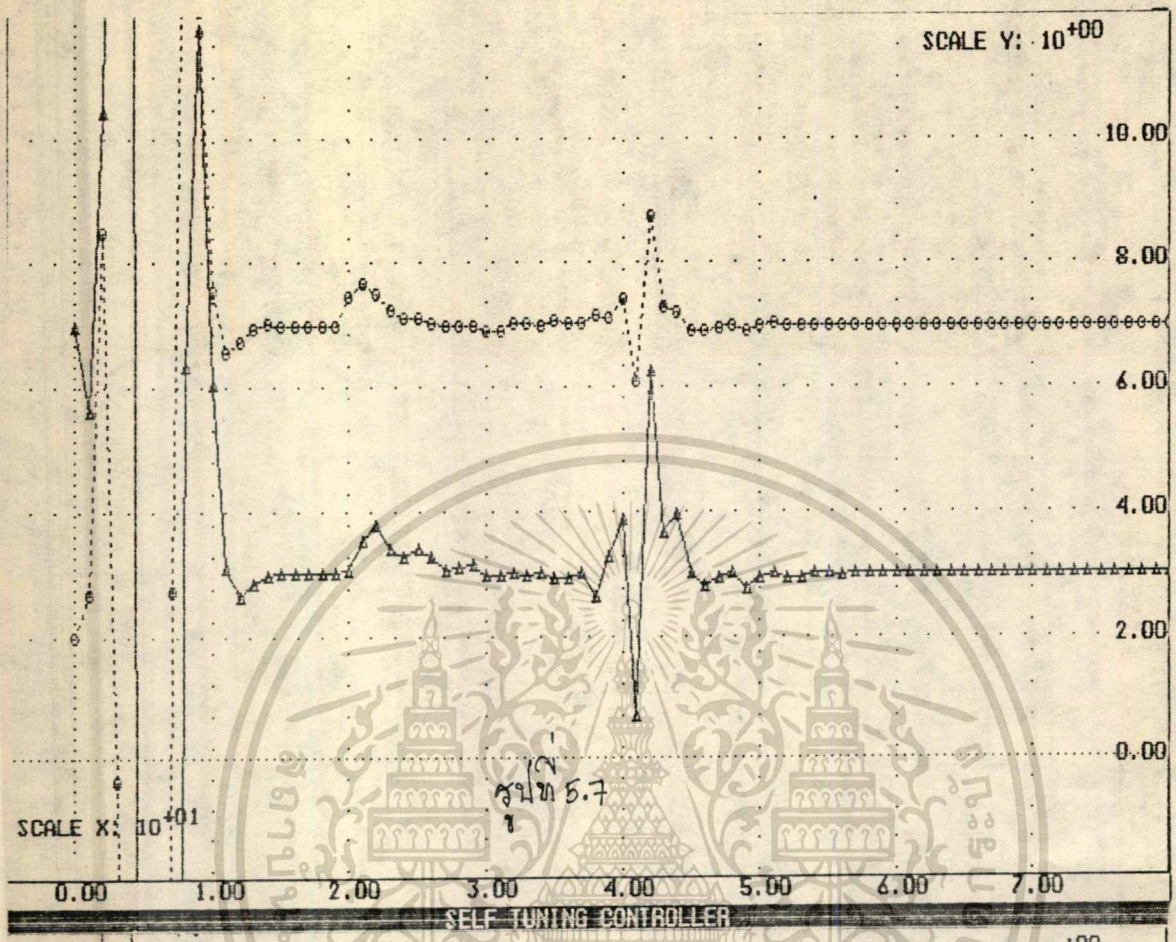
รูปที่ 5.4



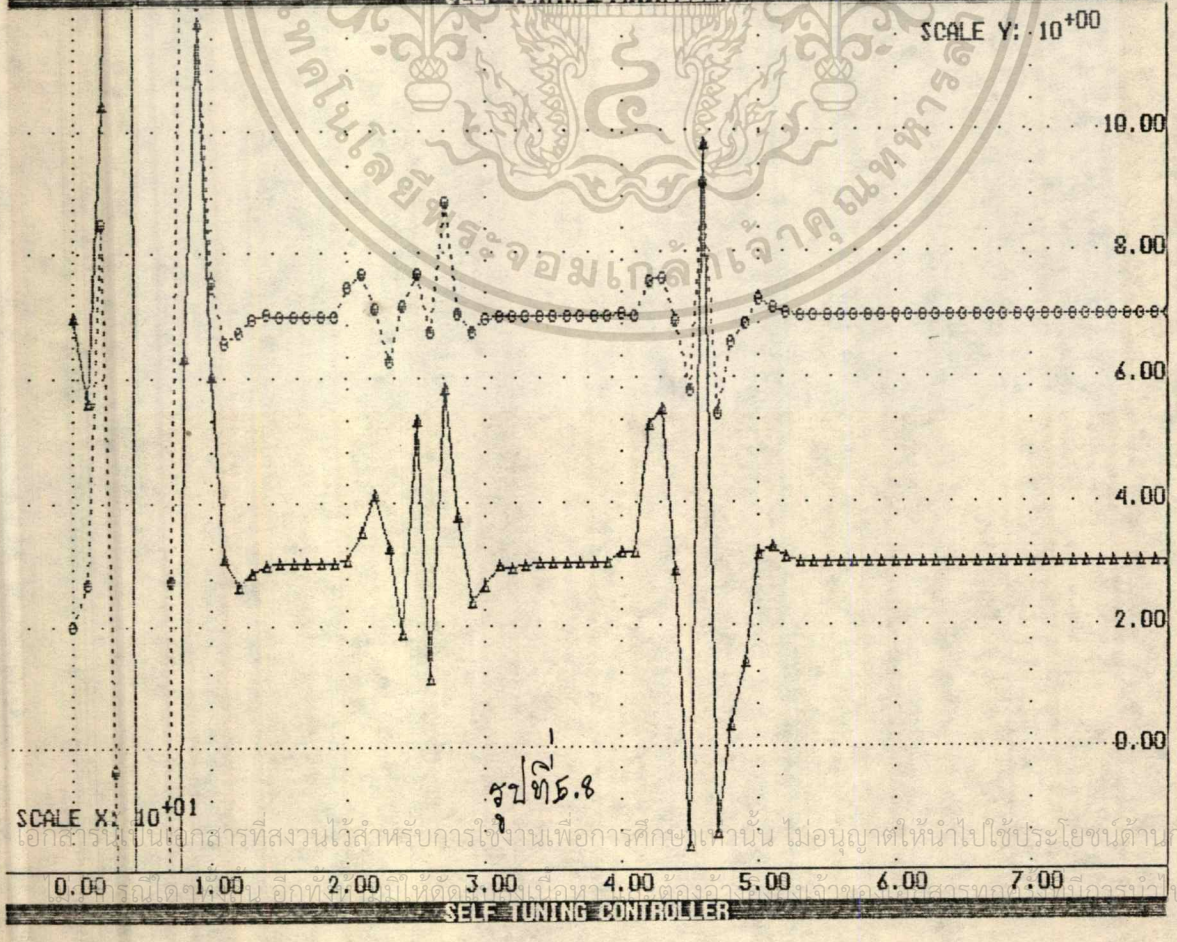
รูปที่ 5.5



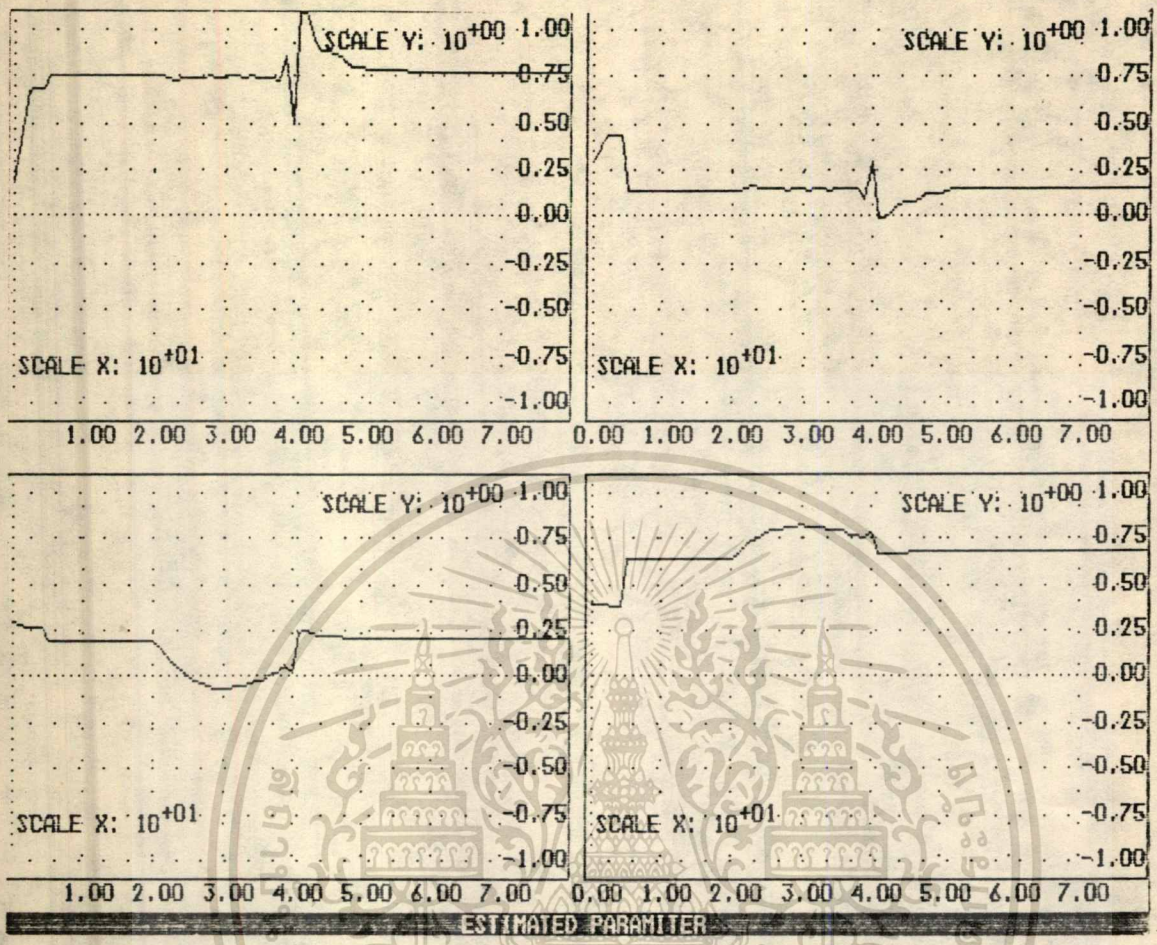
รูปที่ 5.6



รูปที่ 5.7



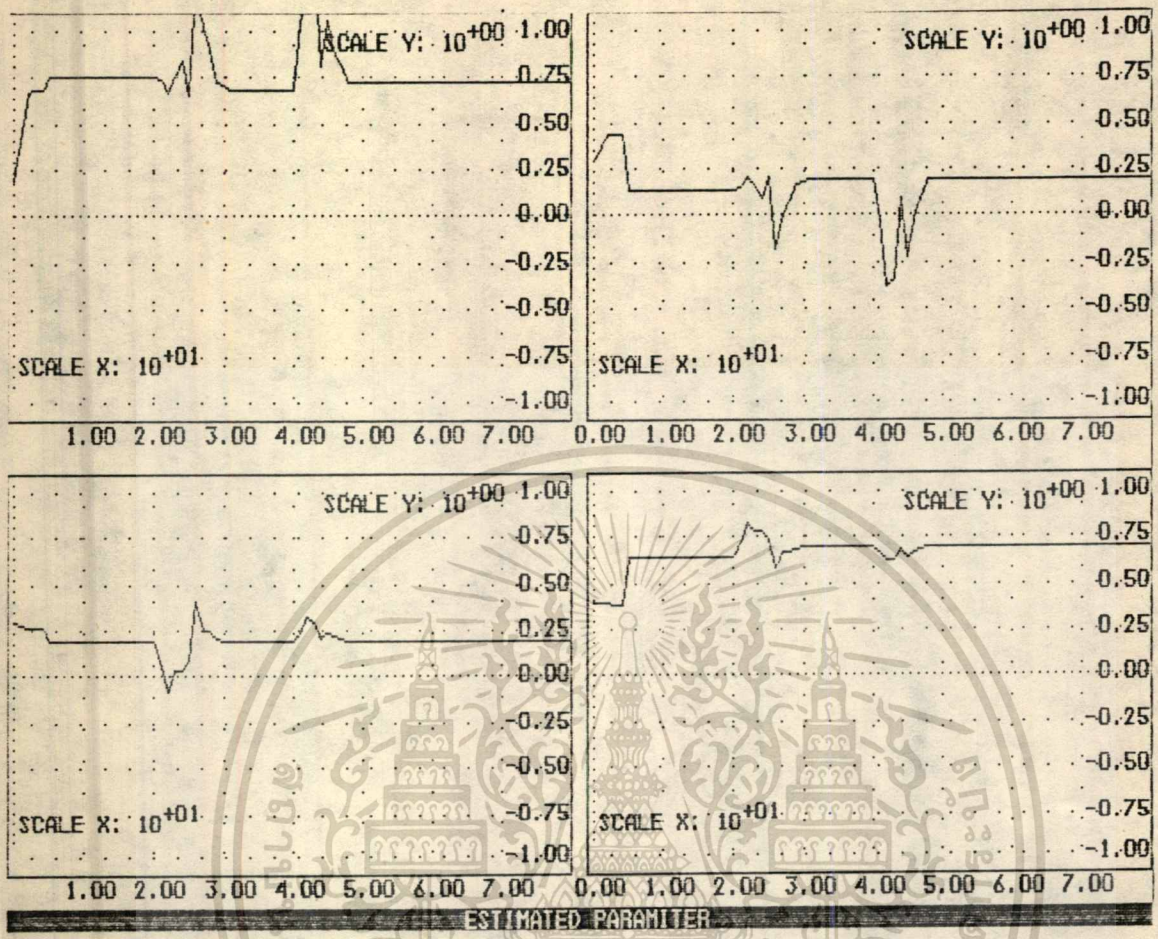
รูปที่ 5.8



รูปที่ 5.9

a_{11} a_{12}

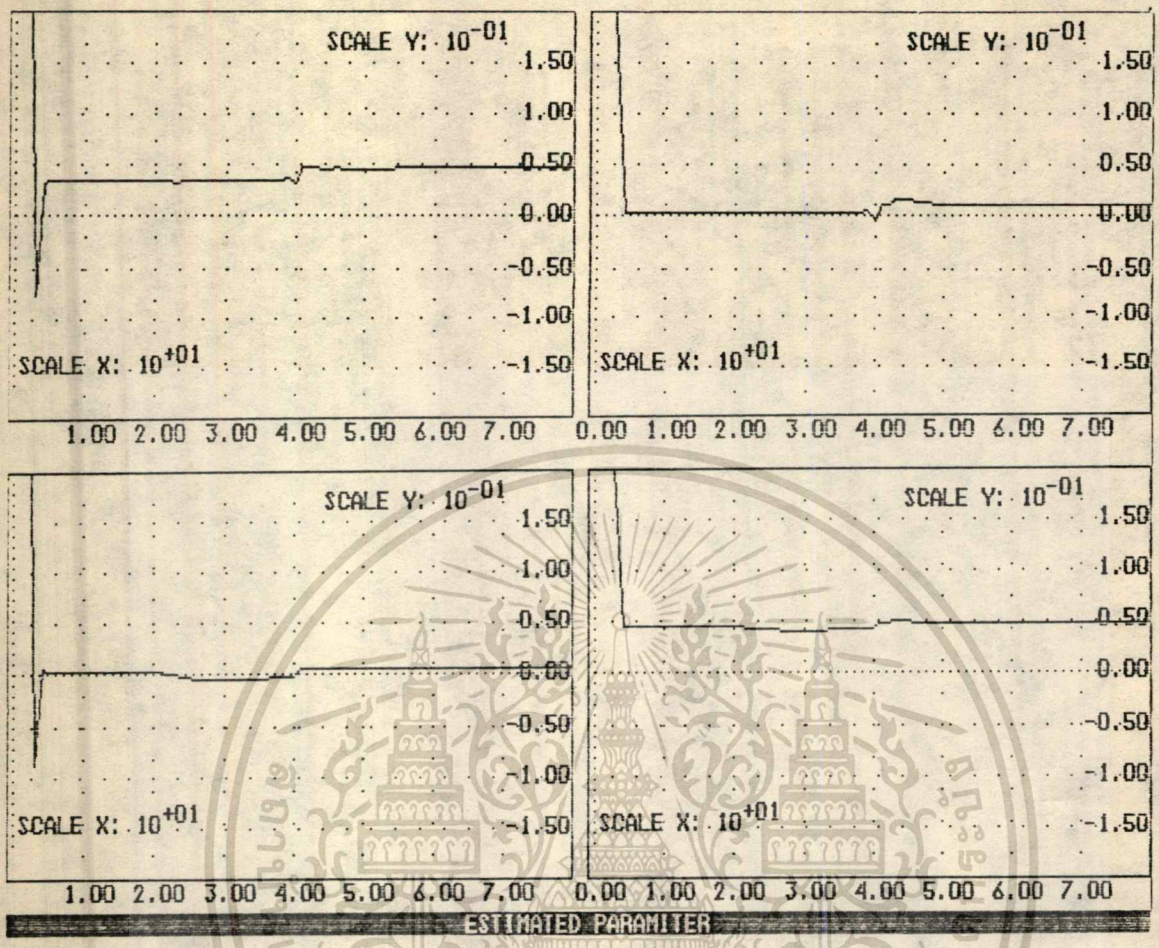
a_{21} a_{22}



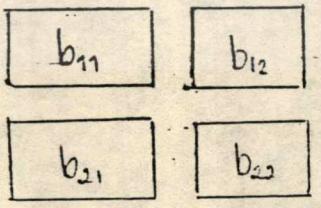
รูปที่ 5.10

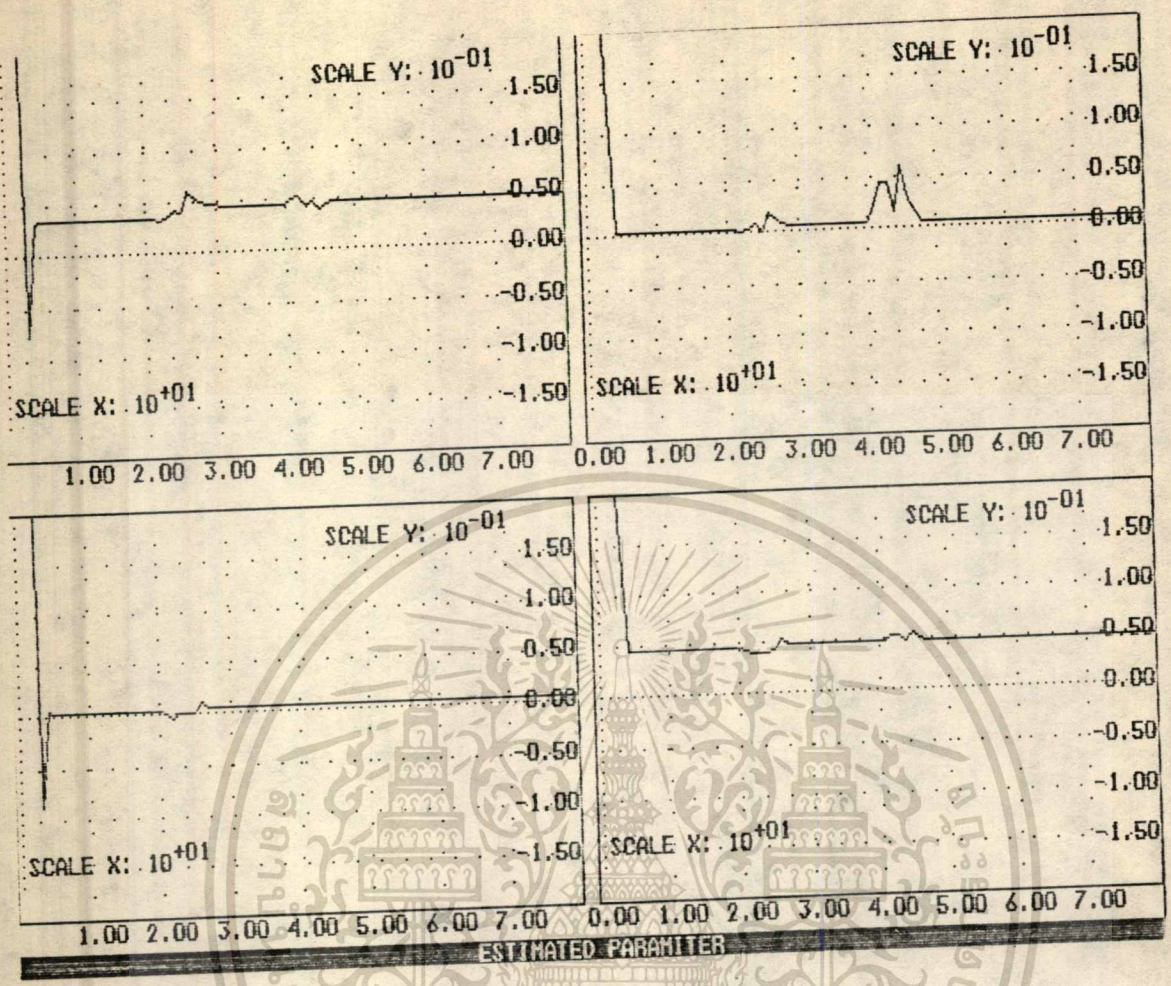
a_{11} a_{12}

a_{21} a_{22}

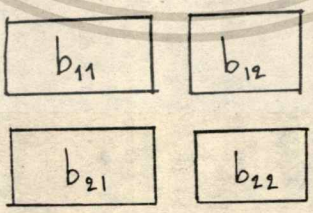


รูปที่ 5.11





รูปที่ 5.12



บทที่ 6
วิจารณ์และสรุป

6.1 สรุปผลการทดลอง

เนื่องจากในช่วงของการเริ่มต้นควบคุมกระบวนการ (start process) จะมีการแกว่งอย่างมากมาย บางครั้งมากถึง 100 เท่าของปรกติ การสรุปนี้จึงไม่คำนึงถึงส่วนนี้ โดยจะนำผลที่ได้จากช่วงการเก็บข้อมูลที่ 10 เป็นต้นไป มาพิจารณาได้ดังนี้

ตอนที่ 1 ตัวควบคุมสามารถทำให้ระบบเปลี่ยนแปลงเข้าหา ค่าเป้าหมายที่เป็นขั้นได้อย่างดี โดยใช้เวลาเพียง 5 ช่วงการเก็บข้อมูล การใช้ค่าสัมประสิทธิการลืม แทบไม่มีผลต่อการเปลี่ยนแปลง ค่าเป้าหมาย เลย

ระบบยังสามารถเปลี่ยนแปลงเข้าหาค่าเป้าหมายที่เปลี่ยนแปลงตามเวลาได้ด้วย โดยมีค่าความผิดพลาดเล็กน้อยพอยอมรับได้

ตอนที่ 2 การใส่ค่าการรบกวน ทำให้ระบบมีการเปลี่ยนแปลงออกจากค่าเป้าหมายในช่วงแรก แต่ในที่สุดระบบยังสามารถเปลี่ยนแปลงเข้าหาค่าเป้าหมายได้

ตอนที่ 3 การใช้ค่าสัมประสิทธิการลืมน้อย จะทำให้ใช้เวลาน้อยในการประมาณค่าพารามิเตอร์ให้ได้ใกล้เคียงค่าจริง แต่ก็ทำให้เกิดการแกว่งของพารามิเตอร์ได้มากกว่า ใช้ค่ามาก ซึ่งการใช้ สัมประสิทธิการลืมมากนี้ก็จะประมาณค่าได้ช้าตามไปด้วย การเกิดการแกว่งของพารามิเตอร์นี้จะทำให้เกิดการเปลี่ยนแปลงค่า เอาท์พุท ของระบบด้วยในทางเดียวกัน

6.2 วิจารณ์ผลและแนวทางการปรับปรุงตัวควบคุม

ตามทฤษฎี least square approximation นั้น ถ้ามีพารามิเตอร์ N ตัว จะสามารถประมาณค่าพารามิเตอร์ได้เมื่อ มีการเก็บข้อมูลครบ N+1 ค่าแล้ว เนื่องจากในการทดสอบนี้ใน 1 สมการเสตท จะมีค่าพารามิเตอร์ 4 ตัว จึงสามารถประมาณค่าพารามิเตอร์ได้เมื่อทำการเก็บข้อมูลครบ 5 ค่าแล้ว ก่อนหน้านั้นพารามิเตอร์ที่คำนวณได้อาจมีการผิดพลาด จึงมีการแกว่งอย่างมาก มายอันเกิดจากการคำนวณหาสัญญาณ U จากค่าพารามิเตอร์ได้มากหรือน้อยเกินไป การแก้ปัญหานี้อาจทำได้โดยใช้ค่าสัญญาณ U ที่มีค่าพอประมาณแล้วแต่กระบวนการไปจำนวน N+1 ครั้ง หลังจากนั้นจึงใช้การคำนวณค่าตามทฤษฎีต่อไป

ในการทดสอบนี้ยังมีส่วนที่ต้องศึกษาต่อไป ซึ่งขณะนี้ยังไม่ได้ศึกษาผลของมันคือ จะถือว่าไม่มีเวลาล่าช้า (delay time) จะถือว่าไม่มีข้อจำกัดขีดขั้น (constrain) หมายถึง ค่า X,U สามารถเป็นได้ทั้งบวกและลบ และไม่กำหนดว่าสูงสุดต่ำสุดเท่าไร จะถือว่าไม่มีค่าผิดพลาดที่เกิดจาก A/D,D/A ค่าสัมประสิทธิการลืม ควรมีค่าเท่าไรจึงจะดีที่สุด ควรวางโพลที่ตำแหน่งใดใน Z-plane จึงจะดีที่สุด เป็นต้น

ภาคผนวก

ตัวโปรแกรม

LISTING OF [C:\selftune.pas]

NAME : SELFTUNE.PAS DATE : 21/MAR/1989
SIZE : 6342 Bytes TIME : 9:49:32

```
  1:PROGRAM SELF TUNNING CONTROL;
  2:USES CRT,MPNTSUB,SELFSUB;
  3:{------ PROGRAM WRITTEN TO BE SELF TUNING CONTROLLER----------}
  4:{          4th YEAR PROJECT                                   }
  5:{      ADVISER  : WIPAN PRICHAPANIJ                           }
  6:{      BY  1. NOPPAWOOT MALAYAWECH   CTRL EN. KMITL  281069  }
  7:{          2. UDOMCHAI  SAEGEE       CTRL EN. KMITL  281318  }
  8:{-------------------------------------------------------------}
  9:{      WRITTEN FOR MULTI-INPUT/MULTI-OUTPUT DISCREAT SYSTEM   }
 10:{      WHICH REPRESENT BY       zX=AX+BU                      }
 11:{                                Y=CX                         }
 12:{      CONTROL LAW BY            U=KX+R    (POLE PLACEMENT)   }
 13:{                                                           }
 14:{                        z :          Shift Operater        }
 15:{                        A : n x n    Real  Matrix          }
 16:{                        B : n x p    Real  Matrix          }
 17:{                        C : p x n    Real  Matrix          }
 18:{                        X : n x 1    State Vector          }
 19:{                        U : p x 1    Precess Input         }
 20:{                        Y : p x 1    process output        }
 21:{                       SP : p x 1    Process Setpoint      }
 22:{-------------------------------------------------------------}
 23:
 24:CONST n=2; { Process to be control here is  N-input P-output }
 25:       p=2; { You can change to suit with your process       }
 26:            { But you must change PROCESS SIMULATER too      }
 27:
 28:
 29:       STARTSP1   =  3.00;
 30:       STARTSP2   =  7.00;
 31:       STARTH1    =  7.00;
 32:       STARTH2    =  2.00;
 33:       STARTTIME  =  0.00;
 34:       STOPTIME   =  3.20;
 35:
 36:VAR  Addat,{ A"          }
 37:       SP,    { SETPOINT  }
 38:       I,     { IDENTITY  }
 39:       A,X,B,U,R,K,C    : MATRIXPOINT;
 40:       T,BUFF           : REAL;
 41:       J,MAXJ           : INTEGER;
 42:       CH               : CHAR;
 43:       CURSORX,CURSORY  : INTEGER;
 44:       REALOUTPUT       : FILE OF REAL;
 45:       TEXTOUTPUT       : TEXT;
 46:
 47:
 48:BEGIN {MAIN}
 49:    CLRSCR;
 50:    NEW(A);     NEW(Addat);
 51:    NEW(X);     NEW(SP    );
 52:    NEW(B);     NEW(R     );
 53:    NEW(U);     NEW(I     );  MakeIDENT(I,N);
 54:    NEW(K);     NEW(C     );
 55:
 56:    A^.N:=n;      B^.M:=n;
 57:    A^.M:=n;      B^.N:=p;
 58:
 59:    X^.M          :=n;          U^.M          :=p;
 60:    X^.N          :=1;          U^.N          :=1;
 61:    X^.DATA[1,1]:=STARTH1;      U^.DATA[1,1]:=0.0;
 62:    X^.DATA[2,1]:=STARTH2;      U^.DATA[2,1]:=0.0;
 63:    R^.M          :=n;          SP^.M         :=n;
 64:    R^.N          :=1;          SP^.N         :=1;
 65:    R^.DATA[1,1]:=0.0;          SP^.DATA[1,1]:=STARTSP1;
 66:    R^.DATA[2,1]:=0.0;          SP^.DATA[2,1]:=STARTSP2;
 67:
 68:
 69:{ PAPER FEED
```

```
}
 70:    C^.M:=p;
 71:    C^.N:=n;
 72:    C^.DATA[1,1]:= 1.0;        C^.DATA[1,2]:=  0.0;
 73:    C^.DATA[2,1]:= 0.0;        C^.DATA[2,2]:=  1.0;
 74:
 75:
 76:    Addat^.M:=A^.N;
 77:    Addat^.N:=A^.M;
 78:    {---EIGEN VALUE OF A" (Addat) ARE DESIGNED POLE-----}
 79:    Addat^.DATA[1,1]:=  0.00000;    Addat^.DATA[1,2]:=  1.0;
 80:    Addat^.DATA[2,1]:= -0.1300;     Addat^.DATA[2,2]:=  0.4;
 81:    {--------YOU CAN CHANGE THEM IF YOU WANT--------}
 82:
 83:    ASSIGN(REALOUTPUT,'REALDATA');  REWRITE(REALOUTPUT);
 84:    ASSIGN(TEXTOUTPUT,'TEXTDATA');  REWRITE(TEXTOUTPUT);
 85:    ASSIGN(PARAMITER ,'PARADATA');  REWRITE(PARAMITER);
 86:
 87:    T             :=STARTTIME;
 88:    X^.DATA[1,1]:=STARTH1;
 89:    X^.DATA[2,1]:=STARTH2;
 90:    WRITE(REALOUTPUT,T,X^.DATA[1,1],X^.DATA[2,1]);
 91:    MAXJ       :=ROUND((STOPTIME-STARTTIME)/Tp);
 92:    GOTOXY(1,22);
 93:    WRITE('  TOTAL.....    ');WRITELN(MAXJ:5);
 94:    WRITE('  CALCULATING...  ');CURSORX:=WHEREX;CURSORY:=WHEREY;
 95:    WRITELN(TEXTOUTPUT,'--k--|---t---|-----h1-----|-----h2-----|');
 96:
 97:
 98:    FOR J:=1 TO MAXJ DO BEGIN
 99:
100:        {**** Place setpoint change section here ******}
101:        {****      (If you want to change)       ******}
102:        CASE J OF
103:          1 : BEGIN SP^.DATA[1,1]:=3; SP^.DATA[2,1]:=7; END;
104:         20 : BEGIN SP^.DATA[1,1]:=6; SP^.DATA[2,1]:=5; END;
105:         40 : BEGIN SP^.DATA[1,1]:=4; SP^.DATA[2,1]:=5; END;
106:         60 : BEGIN SP^.DATA[1,1]:=4; SP^.DATA[2,1]:=6; END;
107:        END;
108:
109:        ESTIMATE_PARAMITER(A,B,X,U);     {--Estimate A,B with X(k),U(k)--}
110:        PolePlacement(A,B,Addat,K);      {--Find K(k) with A,B,A"-------}
111:        R^     :=MUL(INV(MUL(MUL(C,INV(SUB(I,ADD(A,MUL(B,K))))),B)),SP)^ ;
112:                                         {--R=INV(C*INV(I-[A+BK])*B)*SP--}
113:        FINDnewX(X,U);                   {--Sampling for X(k+1) -------}
114:        U^     :=ADD(R,MUL(K,X))^;       {--Send out    U(k+1) -------}
115:        T      :=T+Tp;
116:
117:        {**** Place matrix print section here *****}
118:        {** If you are developing this program ****}
119:
120:        BUFF:=J;
121:        WRITE(REALOUTPUT,BUFF,X^.DATA[1,1],X^.DATA[2,1]);
122:        WRITELN(TEXTOUTPUT,J:5, t:8:2, X^.DATA[1,1]:12:3, X^.DATA[2,1]:12:3 );
123:        GOTOXY(CURSORX,CURSORY);WRITE(J:5);
124:
125:    END;
126:    CLOSE(REALOUTPUT);
127:    CLOSE(TEXTOUTPUT);
128:    CLOSE(PARAMITER);
129:
130:    DISPOSE(A);     DISPOSE(Addat);
131:    DISPOSE(X);     DISPOSE(SP    );
132:    DISPOSE(B);     DISPOSE(R     );
133:    DISPOSE(U);     DISPOSE(I     );
134:    DISPOSE(K);     DISPOSE(C     );
135:
136:END.
137:
138:
139:{ PAPER FEED
```

```
}
{************ SETPOINT CHANGE SECTION ****************************}
{ STEP SETPOINT }
        CASE J OF
           1 : BEGIN SP^.DATA[1,1]:=3; SP^.DATA[2,1]:=7; END;
          20 : BEGIN SP^.DATA[1,1]:=6; SP^.DATA[2,1]:=5; END;
          40 : BEGIN SP^.DATA[1,1]:=4; SP^.DATA[2,1]:=5; END;
          60 : BEGIN SP^.DATA[1,1]:=4; SP^.DATA[2,1]:=6; END;
        END;

{************************************************************************}
{ RAMP SETPOINT }
        IF J>=20 THEN  BEGIN
           SP^.DATA[1,1]:=SP^.DATA[1,1]+Tp;
           SP^.DATA[2,1]:=SP^.DATA[2,1]+Tp;
        END;

{*************** MARTIX PRINT SECTION ******************************}
{***** YOU SHOULD USE THIS SECTION WHEN DEVELOP THIS PROGRAM ******}
        GOTOXY( 1, 3);PRINTMATRIX(SP,'    ----SP----');
        GOTOXY(10, 3);PRINTMATRIX(X ,'    ----X----');
        GOTOXY(30, 3);PRINTMATRIX(U ,'    ----U----');
        GOTOXY(50, 3);PRINTMATRIX(R ,'    ----R----');
        GOTOXY(10, 8);PRINTMATRIX(A ,'    ----A----');
        GOTOXY(30, 8);PRINTMATRIX(B ,'    ----B----');
        GOTOXY(50, 8);PRINTMATRIX(K ,'    ----K----');
        GOTOXY(10,16);PRINTMATRIX(ADD(A,MUL(B,K)),'    ----A+BK----');
{************************************************************************}
------------------------ END OF FILE ------------------------
LIST ON :  TUE:  21/MAR/1989
     AT :        9:51:23
  TOTAL :  166  SOURCE LINES
-------------------------------------------------------------
```

LISTING OF [C:\selfsub.pas]

NAME : SELFSUB.PAS DATE : 20/MAR/1989
SIZE : 12234 Bytes TIME : 23:12:36

```
1:UNIT SELFSUB;
2:{-----SELF TUNING SUBRUTINE-----}
3:INTERFACE
4:USES CRT,MPNTSUB;
5:
6:
7:CONST  LAMDA      =  0.70;    { Forgeting Facter      }
8:       Tp         =  0.04;    { Sampling Time Period  }
9:
10:VAR  PARAMITER  : FILE OF REAL;
11:
12:
13:PROCEDURE  FINDnewX(VAR NEWX:MATRIXPOINT;U:MATRIXPOINT);
14:{-------------------SIMULATER-------------------------------------------}
15:{    SAMPLING FOR H1,H2                                                 }
16:{    AND THEN SEND NEW CONTROL SIGNAL Q1,Q2 TO PROCESS                  }
17:{-------------------------------------------------------------------------}
18:
19:PROCEDURE ESTIMATE_PARAMITER(VAR A,B:MATRIXPOINT; X,U:MATRIXPOINT);
20:{-------------------- ESTIMATER ------------------------------}
21:{   GIVE ME A SET OF X,U                                       }
22:{   THEN, I GIVE YOU PARAMITER MATRIX A,B                      }
23:{-------------------------------------------------------------}
24:
25:PROCEDURE PolePlacement(A,B,Addat:MATRIXPOINT;VAR K:MATRIXPOINT);
26:{-------------------- K FINDER -------------------------------}
27:{    WHEN YOU APPLY CONTROL LAW  U=R+KX                        }
28:{    YOU'LL NEED ME                                           }
29:{    GIVE ME PARAMITER MATRIX A,B                             }
30:{    DON'T FORGET THE POLE YOU DESIGN                         }
31:{    IN EIGEN VALUE OF Addat (A")                             }
32:{      THEN I GIVE YOU THE K YOU WANT                         }
33:{-------------------------------------------------------------}
34:
35:IMPLEMENTATION
36:
37:
38:PROCEDURE FindH1H2(VAR H1,H2:REAL;Q1,Q2:REAL);
39:
40:
41:  CONST  R1:REAL=0.4;   R3:REAL=0.2;  R2:REAL=0.3;
42:         C1:REAL=1.0;                 C2:REAL=0.7;
43:         K :INTEGER=1;
44:         DISTB_H1:REAL=0; {  DISTURBANCE  }
45:         DISTB_H2:REAL=0;
46:
47:  FUNCTION  H1DOT(H1,H2:REAL):REAL;
48:  BEGIN
49:     H1DOT:= -1/C1*(1/R1+1/R3) *H1 +1/(C1*R3)      *H2  +Q1/C1;
50:  END;
51:
52:  FUNCTION  H2DOT(H1,H2:REAL):REAL;
53:  BEGIN
54:     H2DOT:= +1/(C2*R3)         *H1 -1/C2*(1/R2+1/R3)*H2  +Q2/C2;
55:  END;
56:
57:
58:
59:{ PAPER FEED
```

```
}
VAR   h,K1,K2,K3,K4,M1,M2,M3,M4:REAL;
      I  :INTEGER;
CONST N=4;

BEGIN
  {—— PLACE PARAMITER CHANGE SECTION HERE (IF YOU WANT TO CHANGE) ——}

  h :=Tp/N;
  FOR I:=1 TO N DO BEGIN
    K1:=h*H1DOT(H1      ,H2      );
    M1:=h*H2DOT(H1      ,H2      );
    K2:=h*H1DOT(H1+K1/2,H2+M1/2);
    M2:=h*H2DOT(H1+K1/2,H2+M1/2);
    K3:=h*H1DOT(H1+K2/2,H2+M2/2);
    M3:=h*H2DOT(H1+K2/2,H2+M2/2);
    K4:=h*H1DOT(H1+K3  ,H2+M3  );
    M4:=h*H2DOT(H1+K3  ,H2+M3  );
    H1:=H1+(K1+2*K2+2*K3+K4)/6;
    H2:=H2+(M1+2*M2+2*M3+M4)/6;
  END;
  {—— PLACE DISTURBANCE SECTION HERE  (IF YOU WANT TO CHANGE) ——}

  INC(K);
END;


PROCEDURE  FINDnewX(VAR NEWX:MATRIXPOINT;U:MATRIXPOINT);
BEGIN
   FindH1H2(NEWX^.DATA[1,1],NEWX^.DATA[2,1],U^.DATA[1,1],U^.DATA[2,1]);
END;

{*********************************************************}

PROCEDURE ESTIMATE_PARAMITER(VAR A,B:MATRIXPOINT; X,U:MATRIXPOINT);
{——GIVE ME A SET OF X,U————————————————————————}
{——THEN, I GIVE YOU PARAMITER MATRIX A,B ——}

CONST FIRST:BOOLEAN    =TRUE;
      FI    :MATRIXPOINT=NIL;
      THETA:MATRIXPOINT=NIL;
      I     :MATRIXPOINT=NIL;
      K     :MATRIXPOINT=NIL;
      P     :MATRIXPOINT=NIL;

VAR   BUFF  :REAL;
      Y,TEMP :MATRIXPOINT;
      L1,L2       : INTEGER;

BEGIN
      IF FIRST THEN BEGIN
         FIRST:=FALSE;
         NEW(FI);  NEW(K);
         NEW(P);   NEW(I);
         NEW(THETA);
         FI^.M:=1;              THETA^.M:=A^.N+B^.N;
         FI^.N:=A^.N+B^.N; THETA^.N:=A^.M;
         FOR L1:=1 TO THETA^.M DO BEGIN
           FOR L2:=1 TO THETA^.N DO BEGIN
             THETA^.DATA[L1,L2]:=10*L1+L2;    { WE CAN SELECT ANY SET OF FIRST  θ }
           END;                                { PROGRAM ITSELF CAN FIND CORRECT θ }
         END;
         FOR L2:=1 TO FI^.N DO FI^.DATA[1,L2]:=0;
         MakeIDENT(I,FI^.N);
         P^:=MULLSCALAR(+1.00E+2,I)^;
      END;
      NEW(Y);
      NEW(TEMP);

{ PAPER FEED
```

```
}
130:        {————————FOLLOWING EQUATIONS ARE FROM ————————}
131:        {     "COMPUTER CONTROLED SYSTEM                }
132:        {        Theory and Design"                     }
133:        {      Karl J. Åström  &  Bjohn Wittenmark      }
134:        {      Prentice-Hall International Edition      }
135:        {      1984     ,  pp. 335                      }
136:        {————————————————————————————————————————————}
137:      {   CONDENSE }
138:                                                        { ########NOTE################## }
139:                                                        {     t=TRANSPOSE               }
140:     Y^     :=TRANSPOSE(X)^;                           {     Y=Xt                      }
141:    TEMP^   :=MUL(FI,MUL(P,TRANSPOSE(FI)))^;           { TEMP =φ#P#φinv                }
142:    BUFF    :=1/(LAMDA+TEMP^.DATA[1,1]);               { BUFF =1/(LAMDA+TEMP)          }
143:     K^     :=MULSCALAR(BUFF,MUL(P,TRANSPOSE(FI)))^;   {     K=(P#φt)/(LAMDA+φ#P#φinv) }
144:     P^     :=MULSCALAR(1/LAMDA,MUL(SUB(I,MUL(K,FI)),P))^;{ P=(I-K#φ)P/LAMDA           }
145:     THETA^ :=ADD(THETA,MUL(K,SUB(Y,MUL(FI,THETA))))^;{     θ=θ+K(Y-φ#θ)              }
146:     FI^    :=COMBINEcol(Y,TRANSPOSE(U))^;              {     φ=[Xt¦Yt]                 }
147:                                                        { ############################ }
148:                                                        { UNCONDENSE                    }
149:
150:        {————IF θt=[A¦B] ————————————————————}
151:        {————WHAT A,B ARE ? ————————————————}
152:        TEMP^.M:=0;
153:        TEMP^.N:=A^.M;
154:        FOR L1:=1 TO A^.N DO BEGIN
155:           TEMP^:=COMBINEROW(TEMP,ROW(L1,THETA))^;
156:        END;
157:        A^:=TRANSPOSE(TEMP)^;
158:
159:        TEMP^.M:=0;
160:        TEMP^.N:=B^.M;
161:        FOR L2:=L1+1 TO L1+B^.N DO BEGIN
162:           TEMP^:=COMBINEROW(TEMP,ROW(L2,THETA))^;
163:        END;
164:        B^:=TRANSPOSE(TEMP)^;
165:
166:        FOR L1:=1 TO 4 DO BEGIN
167:          FOR L2:=1 TO 2 DO BEGIN
168:            WRITE(PARAMITER,THETA^.DATA[L1,L2]);
169:          END;
170:        END;
171:        DISPOSE(Y);
172:        DISPOSE(TEMP);
173:END;
174:
175:
176:PROCEDURE PolePlacement(A,B,Addat:MATRIXPOINT;VAR K:MATRIXPOINT);
177:{————————————— K FINDER ——————————————————}
178:{     WHEN YOU APPLY CONTROL LAW  U=R+KX    }
179:{     YOU'LL NEED ME                       }
180:{     GIVE ME PARAMITER MATRIX A,B         }
181:{     DON'T FORGET THE POLE YOU DESIGN     }
182:{     IN EIGEN VALUE OF Addat (A")         }
183:{       THEN I GIVE YOU THE K YOU WANT     }
184:{————————————————————————————————————————}
185:VAR   C,Cdat,TEMP1,TEMP2,CdatINV,P,Adat,Bdat : MATRIXPOINT;
186:      I,J,C1,C2     : INTEGER;
187:      BUFF,CidotC2  : REAL;
188:      MAG           : ARRAY[1..10] OF REAL;
189:      OKCOL         : ARRAY[1..10] OF BOOLEAN;
190:      LI            : BOOLEAN;
191:      CH            : CHAR;
192:      LIVECTOR,LDVECTOR,LIMITCOL,CurrentCOL : INTEGER;
193:      GAMMA,DELTA : ARRAY[1..10] OF INTEGER;
194:      {  τ , σ  }
195:
196:
197:
198:
199:
200:{ PAPER FEED
```

```
}
BEGIN
   NEW(C);     NEW(TEMP1);    NEW(TEMP2);


   {------- C = [B¦AB¦A²B¦...] ----------}
   C^:=COMBINECOL(B,MUL(A,B))^;
   TEMP1^:=A^;
   FOR I:=3 TO C^.M DO BEGIN
     TEMP1^:=MUL(TEMP1,A)^;
     C^:=COMBINECOL(C,MUL(TEMP1,B))^;
   END;



   {  CONDENSE }
 {---CALCULATE MAGNITUDE OF COLUMN VECTOR Jth OF C INTO MAG[I]--------}
 {  THEOREM : LET A=Xi+Yj+Zk WHERE i,j,k ARE UNIT VECTOR IN XYZ AXIS  }
 {            MAGNITUDE OF A  OR |A| =√(X²+Y²+Z²)                    }
 {-----------------------------------------------------------------}
 { UNCONDENSE }
   FOR J:=1 TO C^.N DO BEGIN
       BUFF:=0;
       FOR I:=1 TO C^.M DO BEGIN
          BUFF:=BUFF+SQR(C^.DATA[I,J]);
       END;
       MAG[J]:=SQRT(BUFF);
   END;

   {  CONDENSE }
 {----- CHECK FOR LINEAR INDEPENDENT OF COLUMN ---------------------------}
 {   CHECHED RESULT IS IN OKCOL[I]                                        }
 {   :=IF OKCOL[I]=TRUE  MEANS COLUMN Ith IS LI WITH ALL  PREVIOUS COLUMN }
 {     IF OKCOL[I]=FALSE MEANS COLUMN Ith IS LD WITH SOME PREVIOUS COLUMN }
 {  THEOREM : LET A,B  BE VECTORS.                                        }
 {            IF  A.B=|A||B| THEN A IS LINEARLY DEPENDENT WITH B.         }
 {   FIRST OKCOL IS TRUE. BUT IF LI BECOME FALSE ONCE,IT PERMANETLY FALSE }
 {-----------------------------------------------------------------------}
 { UNCONDENSE }
   FOR I :=1 TO C^.N DO OKCOL[I]:=TRUE;
   FOR C1:=1 TO C^.N DO BEGIN
       FOR C2:=C1+1 TO C^.N DO BEGIN
          C1dotC2:=0;
          FOR I:=1 TO C^.M DO BEGIN
            C1dotC2:=C1dotC2+C^.DATA[I,C1]*C^.DATA[I,C2];
          END;
          LI:=ABS(C1dotC2-MAG[C1]*MAG[C2]) > 1.0E-8;
          OKCOL[C2]:=OKCOL[C2] AND LI;
       END;
   END;


   {-------TO CREAT Cdat, WE MUST COMBINE N INDEPENDENT VECTOR-------}
   {       FIRST : WE FIND WHICH COLUMN WE MUST COMBINE TO           }
   {---------------------------------------------------------------}
   LIVECTOR:=0;
   LDVECTOR:=0;
   FOR I:=1 TO C^.N DO BEGIN
     IF OKCOL[I] THEN INC(LIVECTOR) ELSE INC(LDVECTOR);
     IF LIVECTOR=C^.M THEN LIMITCOL:=I   {  NOT NEED ELSE HERE BECAUSE :    }
                                         {  IT'S SURE THAT LIMITCOL:=I       }
   END;                                  {  ONLY ONE TIME IN C^.N LOOPS      }

{ PAPER FEED
```

```
}
263:    {-------------------------------------------------------------------------}
264:    {       SECOND : COMBINE AND RE-ORDER                                     }
265:    {-------------------------------------------------------------------------}
266:    NEW(Cdat);
267:    Cdat^.M:=A^.M;      TEMP1^.M:=Cdat^.M;
268:    Cdat^.N:=0;         TEMP1^.N:=Cdat^.N;
269:    FOR I:=1 TO B^.N DO BEGIN
270:       GAMMA[I]:=0;
271:       DELTA[I]:=0;
272:       FOR J:=1 TO A^.N DO BEGIN
273:          CurrentCOL:=B^.N*(J-1)+I;
274:          IF OKCOL[CurrentCOL] AND (CurrentCOL<=LIMITCOL)
275:          THEN BEGIN
276:             Cdat^ :=COMBINECOL(Cdat ,COL(CurrentCOL,C))^;
277:             INC(GAMMA[I]);
278:          END
279:          ELSE BEGIN
280:             TEMP1^:=COMBINECOL(TEMP1,COL(CurrentCOL,C))^;
281:             INC(DELTA[I]);
282:          END;
283:       END;
284:    END;
285:
286:
287:    {-----CALCULATE DELTA-----------------------------------------------}
288:    DELTA[1]:=GAMMA[1];
289:    FOR I:=2 TO B^.N DO DELTA[I]:=DELTA[I-1]+GAMMA[I];
290:
291:    {-----CALCULATE P---------------------------------------------------}
292:    NEW(CdatINV);
293:    NEW(P);
294:    CdatINV^:=INV(Cdat)^;
295:
296:    P^.M:=0;
297:    P^.N:=A^.N;
298:    FOR I:= 1 TO B^.N DO BEGIN
299:       MAKEIDENT(TEMP1,A^.M);
300:       FOR J:=1 TO GAMMA[I] DO BEGIN
301:         P^:=COMBINEROW(P,MUL(ROW(DELTA[I],CdatINV),TEMP1))^;
302:         TEMP1^:=MUL(TEMP1,A)^;
303:       END;
304:    END;
305:    DISPOSE(CdatINV);
306:    DISPOSE(Cdat);
307:
308:    {----------CALCULATE K-------------------------------------}
309:    {       BY : 1. A'   =P*A*Pinv                             }
310:    {            2. B'   =P*B                                  }
311:    {            3. B'K'=A"-A'                                 }
312:    {            4. REDUCE B TO SQUARE MATRIX                  }
313:    {            5. K'   =B'inv*(A"-A')                        }
314:    {            6. K    =K'P                                  }
315:    {--------------------------------------------------------------}
316:    NEW(Adat);
317:    NEW(Bdat);
318:    Adat^    :=MUL(P,MUL(A,INV(P)))^;
319:    TEMP1^   :=MUL(P,B)^;
320:    TEMP2^   :=SUB(Addat,Adat)^;
321:    Bdat^.M :=0;            Adat^.M:=0;
322:    Bdat^.N :=B^.N;         Adat^.N:=A^.N;
323:    FOR I:=1 TO B^.N DO BEGIN
324:       Bdat^:=COMBINEROW(Bdat,ROW(DELTA[I],TEMP1))^;
325:       Adat^:=COMBINEROW(Adat,ROW(DELTA[I],TEMP2))^;
326:    END;
327:    K^ :=MUL(MUL(INV(Bdat),Adat),P)^;
328:
329:    DISPOSE(Adat);      DISPOSE(TEMP1);      DISPOSE(P);
330:    DISPOSE(Bdat);      DISPOSE(TEMP2);
331:
332:END;
333:
334:
335:
336:END.
337:
338:{ PAPER FEED
```

```
}
339:{************** PARAMITER CHANGE SECTION ***************************}
340:  CASE K OF
341:     1 : BEGIN
342:            R1:=0.4;     R3:=0.2;   R2:=0.3;
343:            C1:=1.0;     WRITE(#7); C2:=0.7;   { ALL }
344:         END;
345:    20 : BEGIN
346:            R1:=0.4;     R3:=0.2;   R2:=0.5;   { C1  }
347:            C1:=0.7;     WRITE(#7); C2:=0.7;
348:         END;
349:    40 : BEGIN
350:            R1:=0.8;     R3:=0.2;   R2:=0.5;   { R1  }
351:            C1:=0.7;     WRITE(#7); C2:=0.7;
352:         END;
353:  END;{CASE}
354:
355:{*************** DISTURBANCE SECTION ********************************}
356:{ CONST DIST }
357:  CASE K OF
358:    20 : DISTB_H2:= 1;
359:    40 : DISTB_H1:=-1;
360:    60 : BEGIN DISTB_H1:= 0;DISTB_H2:=0; END;
361:  END;
362:  H1:=H1+DISTB_H1;
363:  H2:=H2+DISTB_H2;
364:{************************************************************************}
365:{ IMPULSE DIST }
366:  CASE K OF
367:    20 : DISTB_H2:= 1;
368:    21 : DISTB_H2:= 0;
369:    40 : DISTB_H1:=-1;
370:    41 : DISTB_H1:= 0;
371:  END;
372:  H1:=H1+DISTB_H1;
373:  H2:=H2+DISTB_H2;
374:{************************************************************************}
------------------------ END OF FILE ------------------------
LIST ON :  MON:  20/MAR/1989
    AT :         23:17:21
 TOTAL :  374  SOURCE LINES
```

LISTING OF [C:\mpntsub.pas]

NAME : MPNTSUB.PAS DATE : 20/MAR/1989
SIZE : 14686 Bytes TIME : 22:44: 2

```
 1:UNIT MPNTSUB;
 2:INTERFACE
 3:USES CRT;
 4:CONST ArraySizeMax=10;
 5:      NearlyZero=1E-7;
 6:TYPE  vector = array[1..ArraySizeMax] of real;
 7:      matrix = array[1..ArraySizeMax] of vector;
 8:
 9:      MYMATRIX     =  RECORD
10:                       DATA:MATRIX;
11:                       M,N :INTEGER;
12:                     END;
13:      MATRIXPOINT = ^MYMATRIX;
14:VAR   MATRIXBUFF   :  MYMATRIX;
15:
16:PROCEDURE PRINTMATRIX(A:MATRIXPOINT;HEADER:STRING);
17:PROCEDURE MakeIDENT(A:MATRIXPOINT;DIMENSION:INTEGER);
18:FUNCTION  TRANSPOSE(A:MATRIXPOINT):MATRIXPOINT;
19:FUNCTION  MUL(A,B:MATRIXPOINT):MATRIXPOINT;
20:FUNCTION  MULSCALAR(A:REAL;B:MATRIXPOINT):MATRIXPOINT;
21:FUNCTION  ADD(A,B:MATRIXPOINT):MATRIXPOINT;
22:FUNCTION  SUB(A,B:MATRIXPOINT):MATRIXPOINT;
23:FUNCTION  COMBINECOL(A,B:MATRIXPOINT):MATRIXPOINT;
24:FUNCTION  COMBINEROW(A,B:MATRIXPOINT):MATRIXPOINT;
25:FUNCTION  ROW(Nth:INTEGER;A:MATRIXPOINT):MATRIXPOINT;
26:FUNCTION  COL(Nth:INTEGER;A:MATRIXPOINT):MATRIXPOINT;
27:FUNCTION  INV(A:MATRIXPOINT):MATRIXPOINT;
28:
29:IMPLEMENTATION
30:
31:
32:
33:PROCEDURE PRINTMATRIX(A:MATRIXPOINT;HEADER:STRING);
34:VAR I,J,X:INTEGER;
35:
36:BEGIN
37:   X:=WHEREX;
38:   WRITELN(HEADER);GOTOXY(X,WHEREY);
39:   FOR I:=1 TO A^.M DO BEGIN
40:     FOR J:=1 TO A^.N DO BEGIN
41:       WRITE(A^.DATA[I,J]:10:3);
42:     END;
43:     WRITELN;
44:     GOTOXY(X,WHEREY);
45:   END;
46:   WRITELN;
47:END;
48:
49:
50:FUNCTION  TRANSPOSE(A:MATRIXPOINT):MATRIXPOINT;
51:VAR  I,J:INTEGER;
52:     BUFF:MYMATRIX;
53:BEGIN
54:  FOR I:=1 TO A^.M DO BEGIN
55:    FOR J:=1 TO A^.N DO BEGIN
56:      BUFF.DATA[J,I]:=A^.DATA[I,J];
57:    END;
58:  END;
59:  BUFF.M:=A^.N;
60:  BUFF.N:=A^.M;
61:  MATRIXBUFF:=BUFF;
62:  TRANSPOSE:=ADDR(MATRIXBUFF);
63:END;
64:
65:
66:
67:
68:
69:{ PAPER FEED
```

```
}
 70:FUNCTION MUL(A,B:MATRIXPOINT):MATRIXPOINT;
 71:VAR I,J,K:INTEGER;
 72:    BUFF :MYMATRIX;
 73:BEGIN
 74:   IF A^.N<>B^.M THEN BEGIN
 75:      SOUND(200);DELAY(300);NOSOUND;
 76:      WRITELN('ERROR FROM MUL!  NOT COMPATIBLE MATRIX');
 77:      HALT(1);
 78:   END;
 79:   FOR J:=1 TO A^.M DO BEGIN
 80:     FOR K:=1 TO B^.N DO BEGIN
 81:       BUFF.DATA[J,K]:=0;
 82:       FOR I:=1 TO A^.N DO BEGIN
 83:         BUFF.DATA[J,K]:=BUFF.DATA[J,K]+A^.DATA[J,I]*B^.DATA[I,K];
 84:       END;
 85:     END;
 86:   END;
 87:   BUFF.M:=A^.M;
 88:   BUFF.N:=B^.N;
 89:   MATRIXBUFF:=BUFF;
 90:   MUL:=ADDR(MATRIXBUFF);
 91:END;
 92:
 93:FUNCTION ADD(A,B:MATRIXPOINT):MATRIXPOINT;
 94:{--------------- A+B ---------------------------}
 95:VAR  I,J:INTEGER;
 96:     BUFF:MYMATRIX;
 97:BEGIN
 98:   IF (A^.M<>B^.M) OR (A^.N<>B^.N) THEN BEGIN
 99:      SOUND(200);DELAY(300);NOSOUND;
100:      WRITELN('ERROR FROM ADD!  NOT COMPATIBLE MATRIX');
101:      HALT(1);
102:   END;
103:   FOR I:=1 TO A^.M DO BEGIN
104:     FOR J:=1 TO A^.N DO BEGIN
105:       BUFF.DATA[I,J]:=A^.DATA[I,J]+B^.DATA[I,J];
106:     END;
107:   END;
108:   BUFF.M:=A^.M;
109:   BUFF.N:=B^.N;
110:   MATRIXBUFF:=BUFF;
111:   ADD:=ADDR(MATRIXBUFF);
112:END;
113:
114:
115:FUNCTION SUB(A,B:MATRIXPOINT):MATRIXPOINT;
116:{--------------- A-B ---------------------------}
117:VAR  I,J:INTEGER;
118:     BUFF:MYMATRIX;
119:BEGIN
120:   IF (A^.M<>B^.M) OR (A^.N<>B^.N) THEN BEGIN
121:      SOUND(200);DELAY(300);NOSOUND;
122:      WRITELN('ERROR FROM SUB!  NOT COMPATIBLE MATRIX');
123:      HALT(1);
124:   END;
125:   FOR I:=1 TO A^.M DO BEGIN
126:     FOR J:=1 TO A^.N DO BEGIN
127:       BUFF.DATA[I,J]:=A^.DATA[I,J]-B^.DATA[I,J];
128:     END;
129:   END;
130:   BUFF.M:=A^.M;
131:   BUFF.N:=B^.N;
132:   MATRIXBUFF:=BUFF;
133:   SUB:=ADDR(MATRIXBUFF);
134:END;
135:
136:
137:
138:
139:{ PAPER FEED
```

```
}
140:FUNCTION MULSCALAR(A:REAL;B:MATRIXPOINT):MATRIXPOINT;
141:{---------- A*B ----------}
142:VAR I,J:INTEGER;
143:BEGIN
144:    FOR I:=1 TO B^.M DO BEGIN
145:      FOR J:=1 TO B^.N DO BEGIN
146:        MATRIXBUFF.DATA[I,J]:=A*B^.DATA[I,J];
147:      END;
148:    END;
149:    MATRIXBUFF.M:=B^.M;
150:    MATRIXBUFF.N:=B^.N;
151:    MULSCALAR:=ADDR(MATRIXBUFF);
152:END;
153:
154:PROCEDURE MakeIDENT(A:MATRIXPOINT;DIMENSION:INTEGER);
155:VAR I,J:INTEGER;
156:BEGIN
157:    A^.M:=DIMENSION;
158:    A^.N:=DIMENSION;
159:    FOR I:=1 TO DIMENSION DO BEGIN
160:      FOR J:=1 TO DIMENSION DO BEGIN
161:        IF I=J THEN A^.DATA[I,J]:=1 ELSE A^.DATA[I,J]:=0;
162:      END;
163:    END;
164:END;
165:
166:
167:FUNCTION  COMBINECOL(A,B:MATRIXPOINT):MATRIXPOINT;
168:VAR  I,J:INTEGER;
169:     BUFF:MYMATRIX;
170:BEGIN
171:   IF A^.M<>B^.M THEN BEGIN
172:     SOUND(200);DELAY(300);NOSOUND;
173:     WRITELN('ERROR FROM COMBINECOL!  NOT COMPATIBLE MATRIX');
174:     HALT(1);
175:   END;
176:   BUFF:=A^;
177:   FOR I:=1 TO B^.M DO BEGIN
178:     FOR J:=1 TO B^.N DO BEGIN
179:       BUFF.DATA[I,A^.N+J]:=B^.DATA[I,J];
180:     END;
181:   END;
182:   BUFF.M:=A^.M;
183:   BUFF.N:=A^.N+B^.N;
184:   MATRIXBUFF:=BUFF;
185:   COMBINECOL:=ADDR(MATRIXBUFF);
186:END;
187:
188:
189:FUNCTION  COMBINEROW(A,B:MATRIXPOINT):MATRIXPOINT;
190:VAR  I,J:INTEGER;
191:     BUFF:MYMATRIX;
192:BEGIN
193:   IF A^.N<>B^.N THEN BEGIN
194:     SOUND(200);DELAY(300);NOSOUND;
195:     WRITELN('ERROR FROM COMBINEROW!  NOT COMPATIBLE MATRIX');
196:     HALT(1);
197:   END;
198:   BUFF:=A^;
199:   FOR I:=1 TO B^.M DO BEGIN
200:     FOR J:=1 TO B^.N DO BEGIN
201:       BUFF.DATA[I+A^.M,J]:=B^.DATA[I,J];
202:     END;
203:   END;
204:   BUFF.M:=A^.M+B^.M;
205:   BUFF.N:=A^.N;
206:   MATRIXBUFF:=BUFF;
207:   COMBINEROW:=ADDR(MATRIXBUFF);
208:END;
209:{ PAPER FEED
```

```
}
210:FUNCTION  ROW(Nth:INTEGER;A:MATRIXPOINT):MATRIXPOINT;
211:{----ROW Nth OF A MATRIX-------}
212:VAR   I,J:INTEGER;
213:      BUFF:MYMATRIX;
214:BEGIN
215:   IF Nth>A^.M THEN BEGIN
216:      SOUND(200);DELAY(300);NOSOUND;
217:      WRITELN('ERROR FROM ROW!   ROW OUT RANGE');
218:      WRITELN('MAX : ',A^.M,'   BUT REQUIRE : ',Nth);
219:      HALT(1);
220:   END;
221:   FOR J:=1 TO A^.N DO BEGIN
222:      BUFF.DATA[1,J]:=A^.DATA[Nth,J];
223:   END;
224:   BUFF.M:=1;
225:   BUFF.N:=A^.N;
226:   MATRIXBUFF:=BUFF;
227:   ROW:=ADDR(MATRIXBUFF);
228:END;
229:
230:FUNCTION  COL(Nth:INTEGER;A:MATRIXPOINT):MATRIXPOINT;
231:{----COLUMN Nth OF A MATRIX-------}
232:VAR   I,J:INTEGER;
233:      BUFF:MYMATRIX;
234:BEGIN
235:   IF Nth>A^.N THEN BEGIN
236:      SOUND(200);DELAY(300);NOSOUND;
237:      WRITELN('ERROR FROM COL!   COL OUT RANGE');
238:      WRITELN('MAX : ',A^.N,'   BUT REQUIRE : ',Nth);
239:      HALT(1);
240:   END;
241:   FOR I:=1 TO A^.M DO BEGIN
242:      BUFF.DATA[I,1]:=A^.DATA[I,Nth];
243:   END;
244:   BUFF.M:=A^.M;
245:   BUFF.N:=1;
246:   MATRIXBUFF:=BUFF;
247:   COL:=ADDR(MATRIXBUFF);
248:END;
249:
250:
251:
252:procedure Inverse(Dimen : integer;
253:                   Data  : MATRIX;
254:               var Inv   : MATRIX;
255:               var Error : byte);
256:
257:{------------------------------------------------------------------------}
258:{-                                                                      -}
259:{-              Input: Dimen, Data                                      -}
260:{-             Output: Inv, Error                                       -}
261:{-                                                                      -}
262:{-          Purpose : calculate the inverse of a matrix with            -}
263:{-                    Gauss-Jordan elimination.                         -}
264:{-                                                                      -}
265:{-  User-defined Types : VECTOR = array[1..ArraySizeMax] of real;       -}
266:{-                      MATRIX = array[1..ArraySizeMax] of VECTOR       -}
267:{-                                                                      -}
268:{-    Global Variables : Dimen : integer;   Dimension of the square matrix  -}
269:{-                      Data  : MATRIX;  Square matrix                  -}
270:{-                      Inv   : MATRIX;  Inverse of Data                -}
271:{-                      Error : integer;   Flags if something goes wrong  -}
272:{-                                                                      -}
273:{-           Errors : 0: No errors;                                     -}
274:{-                    1: Dimen < 1                                      -}
275:{-                    2: no inverse exists                              -}
276:{-                                                                      -}
277:{------------------------------------------------------------------------}
278:
279:{ PAPER FEED
```

```
}
procedure Initial(Dimen : integer;
                  var Data  : MATRIX;
                  var Inv   : MATRIX;
                  var Error : byte);

{-------------------------------------------------------------------------}
{- Input: Dimen, Data                                                    -}
{- Output: Inv, Error                                                    -}
{-                                                                       -}
{- This procedure test for errors in the value of Dimen                  -}
{-------------------------------------------------------------------------}

var
  Row : integer;

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    begin
      { First make the inverse to be the identity matrix }
      FillChar(Inv, SizeOf(Inv), 0);
      for Row := 1 to Dimen do
        Inv[Row, Row] := 1;
      if Dimen = 1 then
        if ABS(Data[1, 1]) < NearlyZero then
          Error := 2   { Singular matrix }
        else
          Inv[1, 1] := 1 / Data[1, 1];
    end;
end; { procedure Initial }

procedure EROdiv(Divisor : REAL;
                 Dimen   : integer;
             var Row     : VECTOR);

{-------------------------------------------------------------------------}
{- Input: Divisor, Dimen, Row                                            -}
{-                                                                       -}
{- elementary row operation - dividing by a constant                    -}
{-------------------------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    Row[Term] := Row[Term] / Divisor;
end; { procedure EROdiv }

procedure EROswitch(var Row1 : VECTOR;
                    var Row2 : VECTOR);

{-------------------------------------------------------------------------}
{- Input: Row1, Row2                                                     -}
{- Output: Row1, Row2                                                    -}
{-                                                                       -}
{- Elementary row operation - switching two rows                         -}
{-------------------------------------------------------------------------}

var
  DummyRow : VECTOR;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }
{ PAPER FEED
```

```
}
procedure EROmultAdd(Multiplier    : REAL;
                     Dimen         : integer;
                 var ReferenceRow  : VECTOR;
                 var ChangingRow   : VECTOR);

{-------------------------------------------------------------------------}
{- Input: Multiplier, Dimen, ReferenceRow, ChangingRow                    -}
{- Output: ChangingRow                                                    -}
{-                                                                        -}
{- Row operation - adding a multiple of one row to another                -}
{-------------------------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
      ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }


procedure Inver(Dimen : integer;
            var Data  : MATRIX;
            var Inv   : MATRIX;
            var Error : byte);

{-------------------------------------------------------------------------}
{- Input: Dimen, Data                                                     -}
{- Output: Inv, Error                                                     -}
{-                                                                        -}
{- This procedure computes the inverse of the matrix Data                 -}
{- and stores it in the matrix Inv.  If the matrix Data                   -}
{- is singular, then Error = 2 is returned.                               -}
{-------------------------------------------------------------------------}

var
  Divisor, Multiplier : REAL;
  Row, ReferenceRow : integer;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Data         : MATRIX;
            var Inv          : MATRIX;
            var Error        : byte);

{-------------------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, Inv                                  -}
{- Output: Data, Inv, Error                                               -}
{-                                                                        -}
{- This procedure searches the ReferenceRow column of                     -}
{- the Data matrix for the first non-zero element below                   -}
{- the diagonal. If it finds one, then the procedure                      -}
{- switches rows so that the non-zero element is on the                   -}
{- diagonal. This same operation is applied to the Inv                    -}
{- matrix. If no non-zero element exists in a column, the                 -}
{- matrix is singular and no inverse exists.                              -}
{-------------------------------------------------------------------------}


{ PAPER FEED
```

```
}
var
  NewRow : integer;

begin
  Error := 2;  { No inverse exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do
  { Try to find a          }
  { row with a non-zero    }
  { diagonal element       }
  begin
    NewRow := Succ(NewRow);
    if ABS(Data[NewRow, ReferenceRow]) > NearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
      { Switch these two rows }
      EROswitch(Inv[NewRow], Inv[ReferenceRow]);
      Error := 0;
    end;
  end; { while }
end; { procedure Pivot }

begin { procedure Inver }
  { Make Data matrix upper triangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the diagonal element is zero }
    if ABS(Data[ReferenceRow, ReferenceRow]) < NearlyZero then
      Pivot(Dimen, ReferenceRow, Data, Inv, Error);
    if Error = 0 then
    begin
      Divisor := Data[ReferenceRow, ReferenceRow];
      EROdiv(Divisor, Dimen, Data[ReferenceRow]);
      EROdiv(Divisor, Dimen, Inv[ReferenceRow]);
      for Row := 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if (Row <> ReferenceRow) and
          (ABS(Data[Row, ReferenceRow]) > NearlyZero) then
        begin
          Multiplier := -Data[Row, ReferenceRow] /
                        Data[ReferenceRow, ReferenceRow];
          EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
          EROmultAdd(Multiplier, Dimen, Inv[ReferenceRow], Inv[Row]);
        end;
    end;
  end;
end; { procedure Inver }

begin { procedure Inverse }
  Initial(Dimen, Data, Inv, Error);
  if Dimen > 1 then
    Inver(Dimen, Data, Inv, Error);
end; { procedure Inverse }


{ PAPER FEED
```

```
}
470:FUNCTION  INV(A:MATRIXPOINT):MATRIXPOINT;
471:VAR  BUFF:MYMATRIX;
472:     ERRORCODE:BYTE;
473:BEGIN
474:    INVERSE(A^.M,A^.DATA,BUFF.DATA,ERRORCODE);
475:    CASE ERRORCODE OF
476:       0 : BEGIN
477:             MATRIXBUFF.M:=A^.M;
478:             MATRIXBUFF.N:=A^.N;
479:             MATRIXBUFF.DATA:=BUFF.DATA;
480:             INV:=ADDR(MATRIXBUFF);
481:           END;
482:       1 : BEGIN
483:             SOUND(300);DELAY(200);NOSOUND;
484:             WRITELN('ERROR FROM INVERSE:= DIMENSION MATRIX < 1');
485:             HALT(1);
486:           END;
487:       2 : BEGIN
488:             SOUND(300);DELAY(200);NOSOUND;
489:             WRITELN('ERROR FROM INVERSE:= NO INVERSE MATRIX ');
490:             HALT(1);
491:           END;
492:    END;
493:END;
494:
495:
496:
497:END.
```

END OF FILE

```
LIST ON :  MON:  20/MAR/1989
    AT :       23:24:41
 TOTAL :  497  SOURCE LINES
```

กิตติกรรมประกาศ

ผู้จัดทำขอขอบพระคุณอาจารย์ที่ปรึกษา คือ อ.วิพันธ์ ปรีชาพานิช ที่กรุณาช่วยเหลือให้คำแนะนำและข้อคิดเห็นต่างๆ ด้วยความเต็มใจเป็นอย่างสูง และขอขอบคุณบรรณารักษ์ห้องสมุดคณะ เสมอมา ที่ได้เอื้อเฟื้อในการค้นหาเอกสารอ้างอิงต่างๆด้วย

หนังสืออ้างอิง

บุญเสริม แจ้งอรุณ และ สุวลัย กลั่นความดี " การควบคุมแบบจูนปรับตัวเอง " วิศวกรรมศาสตร์ 5 (กันยายน 2529) หน้า 46-55

ธัญญวัฒน์ อภิรัตน์วงศา และ พรเทพ ปรางค์ปรักมะ " การควบคุมแบบจูนปรับตัวเอง", ปริญญานิพนธ์ ปีการศึกษา 2530 ภาคเรียนที่ 2 , สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

K.J.ASTROM AND B.WITTENMARK,"COMPUTER CONTROLLED SYSTEM THEORY AND DESIGN" ,PRENTICE-HALL

K.WARWICK " SELF-TUNNING REGULATORS - A STATE SPACE APPROACH", PROC.IEE ,VOL 33, NO 5-1981 , PP 839-858

YIH T.TSAY ,PROF. LEANG S.SHIEH , "STATE-SPACE - APPROACH FOR SELF-TUNING FEEDBACK CONTROL WITH POLE ASSIGN MENT",

D.GRAUPE, "IDENTIFICATION OF SYSTEM ", ROBERT E.KRIEGER PUBLISHING, 276 P,1976

M.GOPAL, "MODERN CONTROL SYSTEM THEORY" ,JOHN WILEY & SONS,644 P,1987